Exponential Family Predictive Representations of State

by

David Wingate

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2008

Doctoral Committee:
Professor Satinder Singh Baveja, Chair
Professor Alfred O. Hero III
Professor Susan A. Murphy
Professor Michael P. Wellman
Assistant Professor Clayton D. Scott

Prediction is difficult, especially of the future.

– Niels Bohr

To my wife, Martha.

Acknowledgments

This work would not have been possible without generous help, both intellectually and financially. I am grateful to my advisor, Satinder Singh, for the long discussions we have had as he has patiently taught me to think clearly through my own ideas, sharpen my writing, and to raise my sights. A special thanks also to my lab mates, Matt Rudary, Britton Wolfe, Vishal Soni, Erik Talviti, Jonathan Sorg and Ishan Chaudhuri for always letting me bounce ideas around, for listening, and for patient tutoring. Thanks to Andrew Nuxoll for being a kindred spirit, to Nick Gorski for the occasional foosball game and to my collaborators at the University of Alberta. Finally, I would like to gratefully acknowledge the National Science Foundation for financially supporting me through most of my studies with a Graduate Research Fellowship.

Finally, a special thank you to my wife Martha for her love, her constancy, her feistiness and for always keeping me on the straight and narrow. Thank you, Grace, Peterson and Andrew for reminding me what life is really about. You have all made this worthwhile. I love you!

Table of Contents

List of Figures

Figure

List of Appendices

Appendix

Abstract

Many agent-environment interactions can be framed as dynamical systems in which agents take actions and receive observations. These dynamical systems are diverse, representing such things as a biped walking, a stock price changing over time, the trajectory of a missile, or the shifting fish population in a lake.

Often, interacting successfully with the environment requires the use of a model, which allows the agent to predict something about the future by summarizing the past. Two of the basic problems in modeling partially observable dynamical systems are selecting a representation of state and selecting a mechanism for maintaining that state. This thesis explores both problems from a learning perspective: we are interested in learning a predictive model directly from the data that arises as an agent interacts with its environment.

This thesis develops models for dynamical systems which represent state as a set of statistics about the short-term future, as opposed to treating state as a latent, unobservable quantity. In other words, the agent summarizes the past into predictions about the short-term future, which allow the agent to make further predictions about the infinite future. Because all parameters in the model are defined using only observable quantities, the learning algorithms for such models are often straightforward and have attractive theoretical properties. We examine in depth the case where state is represented as the parameters of an exponential family distribution over a short-term window of future observations. We unify a number of different existing models under this umbrella, and predict and analyze new models derived from the generalization.

One goal of this research is to push models with predictively defined state towards real-world applications. We contribute models and companion learning algorithms for domains with partial observability, continuous observations, structured observations, high-dimensional observations, and/or continuous actions. Our models successfully capture standard POMDPs and benchmark nonlinear timeseries problems with performance comparable to state-of-the-art models. They also allow us to perform well on novel domains which are larger than those captured by other models with predictively defined state, including traffic prediction problems and domains analogous to autonomous mobile robots with camera sensors.

Chapter 1

Introduction

In a dynamical system, an agent interacts with its environment by taking actions and receiving observations. Such an agent is often interested in predicting the distribution of future observations, given a history of past actions and observations. For example, in reinforcement learning, one observation is a reward signal, which the agent attempts to maximize by taking appropriate actions. In order to accomplish this, the agent must be able to predict something about the future: if the agent is a stock-broker, it must be able to predict future price trends to know whether to buy or sell. If the agent is a baseball player, it must be able to predict the trajectory of the baseball in order to hit it. If the agent is a chess player, it must be able to predict his opponent's future moves in order to outmaneuver him. The actions that an agent takes now will influence the distribution of future observations, so an agent would like to predict them as accurately as possible in order to act optimally.

Models of dynamical systems allow an agent to predict the distribution of future observations. These models can be built by hand, or they can be learned from data, but in either case there are two important components to them. The first component in a model is some representation of *state*, which we will define formally momentarily. For now, we informally define state as a summary of an agent's knowledge about the state of affairs in the environment. This thesis examines different classes of state representations, comparing their computational and representational characteristics and focusing on how a good representation can be learned directly from data captured from agent-environment interactions.

The second component in a model is an algorithm for *maintaining* state as the agent interacts with the environment. The agent must be able to update its internal knowledge about the state of affairs in the environment in response to new actions and observations. In some cases, the mechanisms for accomplishing this are easily defined once a state representation is defined: in probabilistic models, for example, Bayes law will often arise as part of the natural optimal update algorithm. In other cases, however, the choice of state update algorithm helps define the class of systems which can be captured with the model.

This thesis is concerned primarily with learning models of dynamical systems from data, with the goal of allowing the agent to predict the distribution of future interactions, given a string of past

interactions. There is an important restriction that we will place on all of the models we develop: we stipulate that in every model, state must be defined using nothing but statistics of future observable quantities – that is, state must be composed of *predictions about the future.* The restriction is self-imposed and represents part of the central question that this thesis addresses: how far can we push such models? Is anything lost with this restriction? Is anything gained? This thesis describes a trajectory of work which develops increasingly sophisticated models, and generally concludes that this restriction does not limit the capacity of the models. This conclusion is described more thoroughly in Chapter 11.

The idea of representing state as predictions about the future will be made precise later. We now turn our attention to defining the problem setting more formally.

1.1 Modeling Dynamical Systems

The problem of modeling dynamical systems has been widely studied. We lay the groundwork for the models we will build by describing what we mean by an agent, its environment, the decisions it may have to make and the way it represents its knowledge about the world.

1.1.1 The Agent and Its Environment

For all of the dynamical systems we consider, we adopt the perspective that we have an agent who is interacting with an environment by taking actions and receiving observations in discrete time steps. Roughly speaking, the agent is the thing which we are creating or which we are interested in controlling, and the environment is everything external to it. In our examples, the agents are things like stock brokers, chess players, and baseball batters, while the environments are the systems which govern the range of their dynamics – the stock market, the playing room, the baseball game. All of these examples are anthropomorphic, but more abstract kinds of agents and environments are included in our framework: perhaps the environment is the solar system, and the agent is a software program predicting sun-spots; perhaps the environment is a website, and the agent is an algorithm deciding how much to charge for displaying an advertisement.

From the perspective of the agent, the environment is a black box: at each timestep t, the agent executes some action a_t from a (possibly infinite) set of candidate actions $\mathcal{A}$, and receives an observation o_t from a (possibly infinite) set of candidate observations $\mathcal{O}$. In all of our examples, the agent cares about what happens inside the black box only to the extent that it aids him in predicting the future. Throughout this thesis, we explicitly do not adopt the perspective that we are attempting to accurately discover or describe the contents of the black box; we are interested solely in predicting the distributions of future interactions given past interactions.

Should every learning problem be considered a dynamical system? Our framework is general enough that it might be possible, but to keep the definition of a dynamical system clean, we answer no. Classification, for example, is a widely studied machine learning problem, and it may be

tempting to try to reduce it to a dynamical system: the agent is the classifier, the history of interactions is the training data, and the actions that the agent can take are possible classifications. The difference is precisely where we draw the line between dynamical and non-dynamical systems: in a dynamical system, the temporal component is essential, and cannot be neglected. In classification, the training data can be arbitrarily rearranged, and the results are typically the same.[1] In a dynamical system, the *sequence itself* of actions and observations is essential: rearranging them fundamentally changes the nature of the system.

1.1.2 Properties of Dynamical Systems

Dynamical systems can be categorized according to a few standard properties. We outline the most important ones to situate this work and delineate the boundaries of the systems we consider.

- Controlled vs. uncontrolled

  In a controlled dynamical system, we assume that actions taken now will influence the distribution of future observations. In an uncontrolled system, the agent still makes predictions, but they do not affect the system: a weatherman may predict the chance of rain tomorrow, and may be rewarded for better predictions, but he does not affect the weather. Controlled dynamical systems include domains that arise in reinforcement learning, control theory, and operations research, while uncontrolled systems include problems such as timeseries prediction. We will consider both types of systems.

- Finite vs. continuous environments

  How many configurations can the environment be in? Is the environment like a game of checkers, with a finite number of possibilities, or more like the stock market, with an infinite space of real-valued prices? We will consider both, although we emphasize that our goal is not necessarily to identify the environment and its possible configurations, but rather to predict distributions over future observations.

- Finite vs. continuous observations

  Are the agent's observations continuous (like the position and velocity of a car) or discrete (like the letters in an alphabet)? We will consider both. Part of the goal of this thesis is to expand previously contributed models which are capable of dealing with discrete observations to the case of continuous observations. In some of our models, we will add an additional property, which is the presence or absence of structure *within* individual observations. We will therefore consider discrete-and-structured observations and continuous-and-structured observations. For example, a camera image has discrete elements (each pixel can only occupy 255 distinct colors), and so the set of all possible images is finite. However, the set is so large that for practical purposes, structure within the observation must be exploited.

[1] And if they are not, as might be the case for an online classifier, there is a dynamical component to the model which plays an important role, and the result is a dynamical system.

- Fully observable vs. partially observable

  How much information do observations convey about the environment? In a fully observable domain, the current observation conveys all possible information about the environment to the agent – it does not need to remember anything in order to predict the future. The observability of the system is related to whether the system is *Markov* (Markov, 1913): in a Markovian domain the distribution of future observations is conditionally independent of the past, given the current observation o:

  $$p(\text{future}|o, \text{history}) = p(\text{future}|o).$$

  Thus, in a Markovian domain, the agent only needs the current observation to predict the future perfectly. In a non-Markovian (or "partially observable") domain, this property does not hold. This thesis is concerned explicitly with partially observable domains, and does not consider Markovian domains. One of our goals is to learn ways to represent knowledge about the system, which is summarized from the history of interactions. In a fully observable domain, this is not necessary.

- Deterministic vs. stochastic

  How does the environment change over time? Is it deterministic, like a state machine, or stochastic, like the weather? How do the actions of the agent affect the environment? Are the effects deterministic, like the results of the voters in an election, or stochastic, like the wheels of a robot turning on a sandy beach? How are observations generated? Even in a deterministic environment, observations may be stochastically generated: a camera image of a checkerboard may be corrupted with salt-and-pepper noise, for instance. We will consider domains with any combination of the above possibilities.

- Finite vs. continuous actions

  Are the actions available to the agent continuous, like the decisions of a driver in a car (who can smoothly decide between turning left and turning right), or discrete, like the decisions of a temperature regulating agent (who can decide only whether or not to turn on the air conditioner)? We will consider both.

- Episodic vs. sequential

  In an episodic domain, the agent is repeatedly reset to a known initial configuration, or faces the same task again and again. For example, a baseball batter warms up, steps up to the plate, and swings, and later does the same again. In a sequential domain, the agent simply lives forever, with no a priori bound on how long the agent can expect to interact with the environment. We consider sequential domains, although many of the concepts apply directly to the case of episodic domains.

- Stationary vs. non-stationary

Do the laws governing the environment change over time? Here, we are not referring to how the specific configurations of the environment evolve, but rather whether or not the mechanisms that govern that evolution change. To see the difference, consider the example of the weather: the weather may evolve in a complicated, stochastic way, which results in a complex dynamical system. However, the laws governing weather do not change from moment to moment: temperature, pressure, humidity, evaporation and precipitation are all governed by the laws of physics, which are constant. This means that if we were somehow able to formulate a good model of the weather, it would be just as good today as it is 100 years from now. We say that a domain is stationary when the laws governing the environment do not change, and that it is non-stationary otherwise.

It is always theoretically possible to transform a non-stationary environment into a stationary one by broadening our definition of "environment," but from a theoretical and practical perspective, it is often more useful to think of certain domains as non-stationary. For example, in a multi-agent setting, the fact that other agents are learning and evolving effectively results in a non-stationary environment, even though it would be stationary if we somehow had a perfect model of their learning mechanisms. We will only consider stationary environments.

- Domains with reward

  In some controlled dynamical systems, the environment defines a notion of *reward*. Reward is a special observation that helps the agent know what it is supposed to accomplish in the environment: a stock broker might be rewarded based on profit gained from good trades; a baseball player might be rewarded for hitting the ball, and a chess-player might be rewarded for winning the game. Learning how to act such that reward is maximized is the purview of reinforcement learning.

  This thesis *is* related to reinforcement learning Sutton and Barto (1998) because many of our algorithms learn in domains that are the hallmark of reinforcement learning research, but is more about learning how to model an environment than about how to *use* that model to behave optimally. Even so, the existence of a reward signal may impact the model building process. For example, instead of predicting the entire distribution of future observations, perhaps the agent only needs to predict a subset sufficient to act optimally, which may simplify the model building process. In any case, the representation of state should be adequate for a reinforcement learning algorithm to function properly.

In summary, this thesis is concerned with environments which are sequential, stationary and partially observable; which may or may not have a reward signal; and which may have any mix of stochasticity or determinism and any mix of discreteness or continuity.

1.1.3 State

This thesis is intimately concerned with the idea of state and how it can be represented. Throughout this introduction, we have referenced the idea of "state," relying on an intuitive definition. We now define state more precisely, with emphasis on the fact that there are multiple acceptable representations of state. Throughout this thesis, we will always discuss state from the perspective of the agent, as opposed to that of an omniscient observer.

What exactly is "state?" Informally, state is the current situation that the agent finds itself in: for a robot, state might be the position and angle of all its actuators, as well as its map coordinates, battery status, the goal it is trying to achieve, etc. However, in a partially observable system the agent may not have immediate access to all of the information that it would like to know about its situation. For example, a robot with a broken sensor may not know the exact position of its arm; a stock trader may not know exactly what the long-term strategies are of all the companies he is investing in; and a baseball batter may not know exactly how fast the baseball is approaching.

In partially observable domains, there is a more formal definition of state: state is a summary of all of the information that an agent *could have* about its current situation, which is a summary of the history it has experienced. Usually, an agent will want to compress this history, because otherwise it must store an increasingly large amount of information as it interacts with the environment longer and longer. This motivates the canonical formal definition of state: *state is any finite-dimensional statistic of history which is sufficient*[2] *to predict the distribution of the future.* We will sometimes abbreviate this by simply saying that "state is a sufficient statistic for history."

Defining state in this way implies the following conditional independence assertion: that the distribution of the future is conditionally independent of the past, given state:

$$p(\text{future}|\text{state}, \text{past}) = p(\text{future}|\text{state}).$$

This means that since state has summarized all of the information in a history which is relevant for predicting the future, we may discard the history itself. As it turns out, this summary of history is also sufficient for the agent to act optimally (Astrom, 1965). Thus there is a close connection between representing state, maintaining state, summarizing the past, predicting the future and acting optimally.

Unfortunately, there are some conflicting uses for the word "state." In many models of dynamical systems, such as a Partially Observable Markov Decision Process (or POMDP; Monahan, 1982), the model posits underlying "states" which are assumed to represent the "true" state of the process, which is unobserved by the agent (when necessary, we will refer to these underlying states as "latent states.") In a POMDP, the agent summarizes a history with a *belief state*, which is a distribution over latent states. According to our definition, it is the belief state which is the sufficient statistic for his-

[2] Sufficiency is precisely defined in Section 3.2.

tory. However, latent states are not essential for sufficiency; as we shall see, one of the contributions of this thesis is the introduction of numerous concepts of state which make no reference to any sort of latent state.

There are many acceptable summaries of history. To illustrate this, consider the example of a robot localization problem. A small robot wanders through a building, and must track its position, but it is given only a camera sensor. There are many possible representations for the robot's position. Its pose could be captured in terms of a distribution over x, y coordinates, for example. However, it could also be described in terms of, say, a distribution over polar coordinates. Cartesian and polar coordinates are different representations of state which are equally expressive, but *both* are internal to the agent. Neither is more accurate, or more correct, or more useful, and we have said nothing about how either Cartesian or polar coordinates could be accurately maintained given nothing but camera images. It is easy to see that there are an infinite number of such state representations: any one-to-one transformation of state is still state, and adding redundant information to state is still state.

The realization that there are multiple satisfactory representations of state begs the question: among all possible concepts of state, why should one be preferred over another? There are many possible criteria that could be used to compare competing representations. For example, a representation might be preferred if:

- It is easier for a human (designing an agent) to understand and use.
- It somehow "matches" another agent's notion of state.
- It has favorable computational properties.

Not every statistic of history will be sufficient for predicting the future, which means that some representations may only constitute approximate state. If an approximately sufficient statistic is acceptable, a state representation might be preferred if:

- It is more expressive than another representation.
- It is less expressive than another, but is still sufficient to do what we want to do with it (for example, control the system optimally).

Thus, even among state representations which are equally expressive, there might be reasons to prefer one over another.

Because we are interested in learning agents, we are interested in *learnable* representation of state – those for which effective learning algorithms are available. The idea that one representation of state may be more learnable than another motivates our first distinction between different representations of state: *grounded*[3] representations of state are those in which every component of the state

[3] Some disciplines may have other definitions of the word "grounded" which are specific and technical; we avoid them.

is defined using only statistics about observable quantities (which could be either observables in the future, or the past), and *latent* representations of state refer to everything else. In our robot localization example, both Cartesian and polar coordinates are latent representations of state, because neither is explicitly observed by the robot; only a state representation defined in terms of features of camera images could be defined as grounded.

Within the class of grounded representations of state, we will make further distinctions. Some grounded representations may be defined in terms of past observations, as in the case of k-th order Markov models (where the past k observations constitute state), and others could be defined in terms of the current observation (as in Markovian domains, or domains where some feature of the current observation is state).

There is also a third class of grounded representations, which is the class of *predictively defined representations of state.* In a predictively defined representation of state, state is represented as statistics about features of *future* observations. These statistics are flexible: they may be the parameters of a distribution over the short-term future, represent the expectations of random variables in the future, represent the densities of specific futures, represent statements about future strings of observations given possible future actions, etc. It is this class of state representations which we will investigate throughout the thesis, along with algorithms for maintaining that state.

1.2 Why Predictions About the Future?

Why limit our investigation to the class of models with predictively defined representations of state? We are motivated for three reasons:

- **Learnability**. The central problem this thesis addresses is learning models of dynamical systems from data. The fact that all of the parameters of predictively defined models have direct, statistical relationships with observable quantities suggests that predictively defined models may be more learnable than classical counterparts. As an example of this, the parameter estimation algorithm of Rudary et al. (2005) for the Predictive Linear Gaussian (PLG) model (the predictively defined version of the Kalman filter, which is discussed in Chapter 4) was shown to be statistically consistent, which is a strong learning guarantee (Rudary et al., 2005).
- **Representational Ability**. To date, most models with predictively defined state have been shown to be at least as expressive as their classical counterpart. For example, PLGs are as expressive as Kalman filters (Rudary et al., 2005), and linear PSRs (discussed in Chapter 2) are strictly more expressive than POMDPs (James, 2005). That is, there are domains which can be modeled by a finite PSR which cannot be modeled by any finite POMDP, but every finite POMDP can be modeled by a finite PSR. This representational ability is achieved without sacrificing compactness: linear PSRs are never larger than their equivalent POMDPs, PLGs are never larger than their equivalent Kalman filters, and there are examples of PSRs which are exponentially smaller (in terms of the number of parameters) than their equivalent

POMDP (Littman et al., 2002). For a given data set, a model with fewer parameters is likely to have greater statistical efficiency, which is useful for learnability.

- **Generalization**. As a knowledge representation, predictions about the future may have attractive properties from the perspective of function approximation and generalization. For example, Rafols et al. (2005) have provided some preliminary evidence that predictive representations provide a better basis for generalization than latent ones. To see the intuition for this, consider the problem of assigning values to states for a domain in which an agent must navigate a maze. Using a predictively defined representation, two states are "near" each other when their distributions over the future are similar; if that is true, it is likely that they should be assigned similar values. But if the agent uses, say, Cartesian coordinates as a state representation, two states which are nearby in Euclidean space may not necessarily have similar values. The classic example is two states on either side of a wall: although the two states appear to be close, an agent may have to travel long distances through the maze to reach one from the other, and they should be assigned different values, which may be difficult to do with a smooth function approximator. Littman et al. (2002) have also suggested that in compositional domains, predictions could also be useful in learning to make other predictions, stating that in many cases "the solutions to earlier [prediction] problems have been shown to provide features that generalize particularly well to subsequent [prediction] problems."

Learnable, flexible knowledge representations are important for creating autonomous agents capable of learning in real-world domains, which are characterized by rich percepts (such as camera images), highly structured, nonlinear dynamics (perhaps consisting of objects, their material properties, and dynamical relations), and factored, high-dimensional actions. Models with predictively defined state have shown promise with a combination of theoretical results on representational ability, empirical success, and intuitions regarding learnability and generalization. This suggests that exploring the limits and possibilities of such models is important and potentially rewarding.

1.3 Outline and Summary of Contributions

To summarize the setting for this work, the two basic problems in learning a model of a partially observable dynamical system are selecting a representation of state and selecting an algorithm for maintaining that state. The theme of this research is to create models with predictively defined representations of state that are useful in domains with continuous actions, continuous observations, and in some cases, structured and/or high dimensional observations. The contributions are theoretical (proposing new models which expand on previously proposed model classes), algorithmic (proposing new algorithms to learn the parameters of the models) and computational (finding efficient approximations to make the algorithms practical).

- As background, Chapter 2 introduces basic concepts of modeling dynamical systems and presents the PSR model, which is one of the original models with predictively defined state.

This chapter serves to present many ideas which we subsequently generalize.

- Chapter 3 begins with the "Continuous PSR." The PSR model is defined for domains with discrete actions and observations, and the Continuous PSR extends PSRs to handle continuous actions and observations. We examine the basic questions of sufficiency that arise when extending to continuous actions and observations. We contribute the system dynamics distributions and a generic information-theoretic approach to optimizing state representations. To complete a learning algorithm, we present a mathematically elegant set of design decisions which results in a gradient-based learning algorithm, and a set of Nystrom approximations that allows the algorithm to scale well.

- In Chapter 4 we describe the "Predictive Linear-Gaussian" model (PLG)[4], which deals with linear dynamics and simple scalar observations. The PLG defines state as the parameters of a Gaussian distribution over a window of the short-term future, and is formally equivalent in modeling capacity to the Kalman filter.

- In Chapter 5 we extend the PLG to handle nonlinear dynamics by considering linear dynamics in a nonlinear feature space. We show that a nonlinear extension function suffices to capture nonlinear dynamics. We contribute a learning algorithm based on sample statistics and regressions, and an efficient approximate inference algorithm based on sigma-points to track state (which yields an algorithm related to the unscented Kalman filter).

- In Chapter 6, we present an alternative extension to the PLG, named the Mixture of PLGs, which is based on the idea of modeling dynamics in a piecewise linear way. We again contribute a learning algorithm (which can be considered a mixture version of the standard PLG learning algorithm), and we begin to consider exogenous variables in the context of a traffic modeling problem.

- In Chapter 7, we present the "Exponential Family Predictive Representation of State" (EFPSR) model, which was conceived as the natural generalization of our previous models. The EFPSR represents state as the time-varying parameters of an exponential family distribution over a window of n observations in the future. Its close connection to graphical models and maximum entropy modeling allow us to deal with multivariate, high-dimensional, and/or structured observations, and also provably unifies much of the work: we present theorems which unify the PLG, KPLG, MPLG, and PSRs into the umbrella of the EFPSR family.

- In Chapter 8, we specialize the EFPSR with a specific set of features and extension function, and show that the resulting model is an information form of the PLG, in the same way that the information Kalman filter is an information form of the Kalman filter.

- In Chapters 9 and 10 we predict and analyze a new specialization of the EFPSR, named the Linear-Linear EFPSR. We show how this model is well-suited to approximations which

[4]The PLG is largely the work of Matthew Rudary.

allow it to scale to large domains. We contribute a learning algorithm based on maximum likelihood, and an efficient approximation based on stationary distributions.

For each model and learning algorithm, we also contribute empirical evaluations comparing their performance to other models. The models, learning algorithms and approximations allow us to experiment with new domains that are simply not feasible under other models with predictively defined state. From nonlinear time series problems to bouncing balls to domains with robots and cameras, we begin to consider a larger class of interesting domains.

We finish with concluding remarks in Chapter 11.

Chapter 2

Discrete Observations, PSRs and POMDPs

To begin our exposition of models with predictively defined representations of state, this chapter introduces basic concepts of modeling dynamical systems – including learning such models from data and what it means to control them optimally – and briefly reviews several popular models. We will describe in detail a classic model of controlled, partially observable systems with discrete observations called a *Partially Observable Markov Decision Process*, or POMDP. This will allow us to discuss many of the concepts of predictively defined representations of state by contrasting the POMDP with a different model known as a *Predictive State Representation* (or PSR) which uses a predictively defined representation of state.

The PSR model is important historically because it was one of the first models to use a predictively defined representation of state. Analysis of the PSR model was responsible for the definition of many of the theoretical, notational and algorithmic tools used in describing dynamical systems with predictively defined representations of state. In the context of this thesis, the model is also a natural starting point to introduce a variety of modeling concepts, which we will generalize in subsequent chapters.

2.1 Models of Dynamical Systems with Discrete Observations

To arrive at our exposition of POMDPs and PSRs, we will now specialize to the case of dynamical systems with discrete observations, deferring a discussion of dynamical systems with continuous observations until Chapter 4. We will briefly review several popular classes of dynamical systems, with a few notes on learning and controlling them.

The simplest models of discrete observation dynamical systems are known as Markov Chains (Markov, 1913), which are uncontrolled processes in which the observation constitutes state (because of this, the word "observation" and "state" are often used interchangeably). At each timestep, the agent transitions from one state to another, in a possibly stochastic manner. It is straightforward to learn a model of such domains: a consistent maximum likelihood estimator of the transition probabilities can be easily derived from the empirically observed counts of each transition. In some partially observable dynamical systems, the current observation does not constitute state, but a memory of the past k observations *does* constitute state. We term these *history-window models*, but they are also known as k-th order Markov models, or autoregressive models. These models are

simple to use and are theoretically comparable to Markov Chains and MDPs (defined next). There is flexibility in defining the window; for example, in some models, k is allowed to vary depending on the current observation (McCallum, 1995).

The controlled counterpart of the Markov Chain is known as a *Markov Decision Process*, or MDP (Puterman, 1994). Like a Markov chain, the learning problem is not complicated, but in addition to states and transitions, the MDP adds actions and rewards. That is, in every state the agent receives a (possibly stochastic) reward, and transition probabilities now depend on the action the agent selects at each timestep. The *planning problem* is to find a *policy*, or a mapping from states to actions, that maximizes some optimality criteria defined on the rewards. For example, the agent may seek a policy which maximizes the reward over a finite window into the future, the expected discounted reward over the infinite future, or the long-term average reward (Blackwell, 1962)(see also Littman, 1996; Mahadevan, 1996, and references therein). The planning problem is theoretically tractable (it is known to be P-complete; Papadimitriou and Tsitsiklis, 1987), and considerable research has gone into making it practical (see, for example, Wingate and Seppi, 2005, and references therein).

Some of the most widely used models of partially observable dynamical systems posit the existence of latent states, which are not observed but which are assumed to be generating the observations. These latent states are sufficient for history: if they could be observed, they would render the future independent of the past. The most popular uncontrolled model in this class is the *Hidden Markov Model*, or HMM (Rabiner and Juang, 1986). This model is well-studied and has enjoyed wide acceptance, partly due to its performance on real-world problems and partly due to theoretically sound learning algorithms. It has been applied to a variety of applications, such as speech recognition (Rabiner, 1989) and protein classification and alignment (Haussler et al., 1993). Learning HMMs is typically done with the Baum-Welch algorithm (Baum et al., 1970), which is a specialized version of the more general EM algorithm (Dempster et al., 1977), but there are many variations on this basic theme (see, for example, the Bayesian approaches of Stolcke and Omohundro, 1993). The difficulty in learning a model with EM is the existence of several shallow local maxima in the likelihood surface which can result in useless models (Nikovski, 2002).

The controlled counterpart of the HMM is known as a *Partially Observable Markov Decision Process*, or POMDP (Monahan, 1982). The POMDP posits the same latent states as the HMM, but like an MDP, adds actions and rewards. The next section is fully devoted to POMDPs, so we defer our discussion of this model until then.

There have also been several recent attempts to define models of partially observable systems which capture state through the use of predictions about the future. Section 2.4 discusses these thoroughly, after we have presented the PSR and POMDP models.

2.2 POMDPs

POMDPs are a popular class of dynamical system models that have been used in applications as diverse as simultaneous localization and mapping in mines (Thrun et al., 2005) to cognitive assistive technologies (Hoey et al., 2007) to modeling the fishery industry (Lane, 1989). For a thorough survey, see Cassandra (1998b).

A POMDP is described by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R, b_0 \rangle$, where $\mathcal{S}$ is a finite set of latent states, $\mathcal{A}$ is a finite set of actions, and $\mathcal{O}$ is a set of observations. The set T is a set of transition matrices, where T_a is a matrix represents the probability of transitioning from state s to state s' given that action a was taken. The set O is a set of observation matrices, where O_o is a diagonal matrix where $(O_o)_{ii}$ specifies the probability that observation o will be observed in state s_i. The function R is a reward function, where $R(s, a)$ specifies the reward for taking action a in state s. The vector b_0 represents the initial belief state, which is a distribution over latent states.

At each time step i, the agent executes an action $a_i \in \mathcal{A}$ and receives an observation $o_i \in \mathcal{O}$. An agent's internal state in a POMDP is captured as a "belief state," which is the set statistics of a multinomial distribution over latent states $p(s)$. Because a belief state has a probabilistic interpretation, it can be efficiently updated in a recursive way using Bayes law. Given a belief state b, an action a, and an observation o, state can be updated as

$$p(s'|s, a, o) = \frac{p(s', o|s, a)}{p(o|s, a)} = \frac{p(s', o|s, a)}{\sum_{s'} p(s', o|s, a)} = \frac{O_o T_a b}{\vec{1}^\top O_o T_a b}$$

where $\vec{1}$ is an appropriately sized vector of 1's.

2.2.1 Optimal Control of a POMDP

The problem of optimal planning in a POMDP has been addressed by a number of authors. A policy is a (possibly stochastic) mapping from a belief state to an action (because the belief state is a sufficient statistic for history, it contains all of the information necessary for optimal control of the agent). The goal in optimal planning is to find an *optimal* policy – that is, a policy which maximizes some measure of performance, such as average reward or expected discounted reward.

The problem of planning in a POMDP is provably difficult, and is known to be PSPACE-complete (Papadimitriou and Tsitsiklis, 1987). Numerous algorithms (both exact and approximate) have been proposed, such as value iteration (Smallwood and Sondik, 1973), finite policy trees (Sondik, 1971), the Witness algorithm (Littman, 1994), point-based value iteration (Pineau et al., 2003), recurrent nets (Bakker, 2004) finite policy graphs (Meuleau et al., 1999), utile distinction Hidden Markov models (Wierstra and Wiering, 2004), finite-state controllers (Hansen, 1998), and policy gradient methods (Baxter and Bartlett, 2001; Aberdeen and Baxter, 2002). For surveys, see Lovejoy (1991) or Cassandra (1998a).

Some authors do not attempt to estimate a model, and try to find instead "reactive" or memoryless policies which map observations (not states) directly to actions (Jaakkola et al., 1995; Loch and Singh, 1998). It is known that the optimal reactive policy may be stochastic (whereas the optimal policy for POMDPs and MDPs is always deterministic), and it is known that the best memoryless policy can be arbitrarily suboptimal in the worst case (Singh et al., 1994).

2.2.2 Learning a POMDP Model

Often, the model parameters (T, O, R) are not given, and must be learned from data. While not as well studied as the basic problem of planning, many learning algorithms for POMDPs have been proposed. There are two different classes of learning scenarios: in the first, the number of states is given, along with the observation probabilities from each state, but the transitions between states must be estimated. For example, Russell et al. (1994) proposed an algorithm based on steepest gradient ascent in the space of transition probabilities, while Chrisman (1992) and Koenig and Simmons (1998) adapted HMM learning algorithms to learn the transition probabilities.

The more general class of algorithms assume no prior knowledge and attempt to learn everything about the dynamical system: the number of states, the observation probabilities and the transition probabilities must all be estimated. Many approaches have been proposed: Basye et al. (1995) present a learning algorithm which constructs deterministic finite automata under perceptual aliasing. This was later generalized in the work of Shalizi and Shalizi (2004), who has proposed the Causal State Splitting Reconstruction algorithm, which constructs states explicitly based on the distribution over the future they induce. Shatkay and Kaelbling (2002) have shown how domain knowledge may be incorporated into the process of learning a model, with applications to robot localization. Nikovski (2002) has proposed learning algorithms based on state aggregation. Holmes and Isbell (2006) have proposed the looping suffix tree algorithm, which is based on the idea of excising non-informative portions of a history to determine states.

2.3 PSRs

The POMDP model described in the previous section has latent states at its heart – it begins by describing a state space, transitions between those states, the observations that they generate; it defines its sufficient statistic for history as a distribution over these latent states, etc. This model is convenient in several situations, such as when a human designer knows that there really *are* latent states in the system and knows something about their relationships. However, numerous authors have pointed out that while a POMDP is easy to write down, it is notoriously hard to *learn* from data (Nikovski, 2002; Shatkay and Kaelbling, 2002), which is the central concern of this thesis.

In this section, we turn to alternative model of controlled dynamical systems with discrete observations, called a "Predictive State Representation" (or PSR) [1]. The PSR was introduced by Littman et al. (2002), and is one of the first models with a predictively defined representation of state. A PSR

[1] Unfortunately, this name would be more appropriate as a name for an entire class of models.

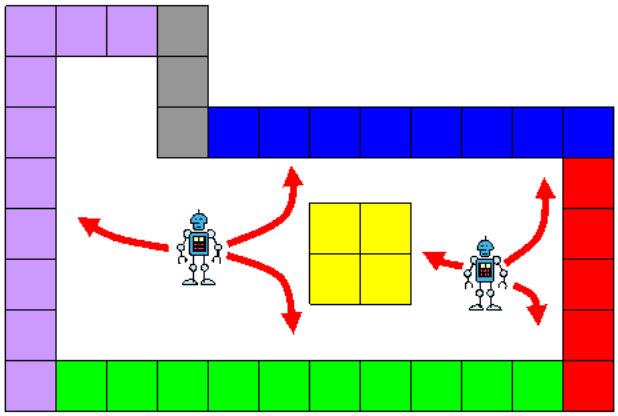


Figure 2.1: An example of state as predictions about the future. Latent state might be x, y coordinates, but predictively defined state is defined in terms of future possibilities.

is a model capable of capturing dynamical systems that have discrete actions and discrete observations, like a POMDP. The important contrast is that while POMDPs are built around latent states, PSRs never make any reference to a latent state. Instead, a PSR represents state as a set of statistics about the future. This will have positive consequences for learning, as we will see, but importantly, we will lose nothing in terms of modeling capacity: we will see that PSRs can model any system that a finite-state POMDP can, and that many POMDP planning algorithms are directly applicable to PSRs.

We will now introduce the terminology needed to explain PSRs.

Histories and tests: Recall that a history is defined as a sequence of actions and observations in the past $a_1 o_1 a_2 o_2 \cdots a_m o_m$, and that a test is defined as a possible sequence of future actions and observations $a^1 o^1 a^2 o^2 \cdots a^n o^n$. A test is a possible future from a given history, and specifies both a sequence of actions and observations. An agent is not obligated to take the actions defined in a test – it merely represents one possible future.

Figure 2.1 illustrates the idea of tests. The figure shows two robots in a brightly colored maze. For the robot on the left, there is a certain action-conditional distribution over the future: if it moves left, it will bump into a pink (light gray) wall; if it goes to the right then up, it will bump into a blue (dark gray) wall, and if it goes to the right and then down, it will bump into a green (medium gray) wall. In this example, the actions include "move-left," "move-right," etc., and the observations include "bump," "blue wall," etc. Tests are strings of these atomic actions and observations. If there is nondeterminism in the world (perhaps due to slippage in wheels, noise in actuators, etc.) these possible futures might not be certain: perhaps if the agent moves right and then moves up, there is some probability that it will see the blue wall, but there is also some possibility that it will overshoot, run into the yellow (very light gray) wall, and the bump sensor will be actuated.

The robot on the right is in a different position in the maze, and therefore there is a different distribution over future possibilities: for him, going left results in bumping into a yellow wall, instead of a blue wall. In this simple example, we can imagine that a sufficiently detailed set of sufficiently long predictions could disambiguate any two positions in the maze. This is precisely the intuition behind the state representation in PSRs, as we will discuss momentarily.

Prediction of a test: From a particular history, there are many possible future sequences of actions and observations, and because of non-determinism in the world, there is some distribution over these possible futures. This distribution can be captured through the use of tests: each possible future corresponds to a different test, and there is some probability that each test will occur from every history.

To formalize the idea of predictions about tests, we say that a test *succeeds* if the observations of the test are obtained, given that the test's actions are taken. A *prediction* for a test $t = a^1 o^1 a^2 o^2 \cdots a^n o^n$ starting in history h is the probability that t will succeed when its actions are executed immediately following h. We define the prediction for a test from history h of length m to be

$$p(t|h) = \Pr(o_{m+1} = o^1, o_{m+2} = o^2, \cdots, o_{m+n} = o^n | h, a_{m+1} = a^1, \cdots, a_{m+n} = a^n).$$

For ease of notation, we use the following shorthand: for a set of tests $T = \{t_1, t_2, \cdots t_n\}$, $p(T|h) = [p(t_1|h), p(t_2|h), \cdots p(t_n|h)]^\top$ is a column vector of predictions.

The idea of tests and their predictions forms a central part of the state representation used by PSRs. They are also central to the mathematical objects that PSRs rely on for theoretical results, as well as most learning algorithms for PSRs.

The system dynamics vector: The *systems dynamics vector* (Singh et al., 2004) is a conceptual construct introduced to define PSRs. This vector describes the evolution of a dynamical system over time: every possible test t has an entry in this vector, which represents $p(t|\emptyset)$ (that is, the prediction of t from the null history), which are conventionally arranged in length-lexicographic order, from shortest to longest. This will be an infinitely long vector, but will still be useful from a theoretical perspective. Here, we will the notation $a_t^m o_t^n$ to denote the m-th action and the n-th observation at time t:

$$\mathcal{V} = \left[p(a_1^1 o_1^1|\emptyset), p(a_1^1 o_1^2|\emptyset), \cdots, p(a_1^m o_1^n|\emptyset), p(a_1^1 o_1^1 a_2^1 o_2^1|\emptyset), \cdots\right]$$

Importantly, the system dynamics vector is representation-independent – that is, *every* dynamical system with discrete observations can be completely described by its system dynamics vector. It makes no reference to latent states of any sort, but still completely characterizes the dynamics of the system. This will be a key part of learning a model of the system.

The system dynamics matrix: The system dynamics matrix $\mathcal{D}$ (shown in Figure 2.2) is obtained by conditioning the system dynamics vector $\mathcal{V}$ on all possible histories. In this matrix, the first row

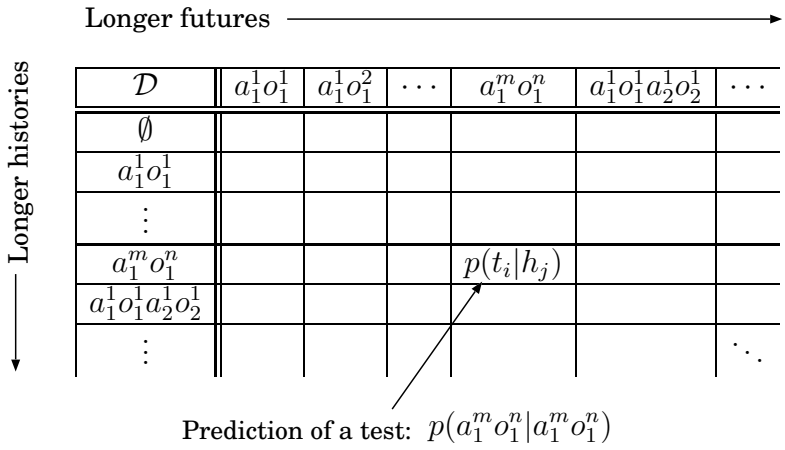


Figure 2.2: The system dynamics matrix.

is the system dynamics vector, which corresponds to the null history. Every possible history has a row in the matrix, and the entries in that row are obtained by conditioning the system dynamics vector on that particular history. An entry in the matrix is the prediction of a particular test from a particular history:

$$\mathcal{D}_{ij} = p(t_j|h_i) = \frac{p(h_it_j|\emptyset)}{p(h_i|\emptyset)}.$$

Tests and histories are arranged length-lexicographically, with ever increasing test and history lengths. The matrix has an infinite number of rows and columns, and like the system dynamics vector, it is a complete description of a dynamical system.

Sufficient statistics: The system dynamics matrix inherently defines a notion of sufficient statistic, and suggests several possible learning algorithms and state update mechanisms. For example, even though the system dynamics matrix has an infinite number of columns and rows, if it has finite rank, there must be a finite set of linearly independent columns. Recall that columns correspond to different tests. We call the tests associated with these linearly independent columns *core tests*. Similarly, there must be a set of linearly independent rows. Recall that rows correspond to different histories. We call the histories associated with these linearly independent rows *core histories*.

In fact, the rank of the system dynamics matrix has been shown to be finite for interesting cases, such as POMDPs (Singh et al., 2004)): a POMDP with n latent states will generate a system dynamics matrix with rank at most n. Furthermore, a set of n linearly independent columns can be found using tests that are all shorter than length n. The set of core tests is not necessarily unique, because any linearly independent set of columns satisfies the definition of a core set of tests.

State: We are now prepared to discuss the key idea of PSRs: PSRs represent state as a set of *predictions about core tests*, which represent the probabilities of possible future observations given possible future actions. Core tests are at the heart of PSRs, because by definition, every other

column can be computed as a weighted combination of these columns.

To see how the predictions of a set of core tests can constitute state, consider a particular history h_t. Suppose that an agent knows which tests are core tests. We will call this set Q, and suppose furthermore that the agent has access to a vector containing their predictions from history h_t:

$$p(Q|h_t) = [p(q_1|h_t), p(q_2|h_t), \cdots, p(q_n|h_t)] .$$

Every other column c in the row corresponding to h_t can be computed as some weighted combination of the entries in $p(Q|h_t)$:

$$p(c|h_t) = m_c^\top p(Q|h_t).$$

Because columns correspond to possible futures, this agent can predict anything about the future that it needs to, assuming it has the appropriate weight vector m_c. Importantly, these weight vectors are independent of history, which will be critical to maintaining state, as we will see in a moment.

Because an agent can predict anything it needs to as a weighted combination of the entries in $p(Q|h_t)$, we say that the predictions of these core tests are *a linearly sufficient statistic* for the system. It is, of course, possible to go through the same line of reasoning to arrive at nonlinearly sufficient statistics for history, although we will not discuss that in this section.

State update: Given a set of core tests Q, their predictions $p(Q|h)$ (which constitute state), an action a and an observation o, the updated prediction for a core test $q_i \in Q$ is given by

$$p(q_i|hao) = \frac{p(aoq_i|h)}{p(ao|h)} .$$

This means that to maintain state, we only need to compute the predictions of the *one step tests* (ao) and the *one-step extensions* (aoq_i) to the core tests as a function of $p(Q|h)$.

This formula is general to all PSRs, whether they use linearly sufficient statistics or nonlinearly sufficient statistics. In the case of linearly sufficient statistics like those discussed previously, the state update takes on a particularly convenient form, as discussed next.

Linear PSRs: Linear PSRs are built around the idea of linearly sufficient statistics. In a linear PSR, for every test c, there is a weight vector $m_c \in \mathbb{R}^{|Q|}$ independent of history h such that the prediction $p(c|h) = m_c^\top p(Q|h)$ for all h. This means that updating the prediction of a single core test $q_i \in Q$ can be done efficiently in closed-form. From history h, after taking action a and seeing observation o:

$$p(q_i|hao) = \frac{p(aoq_i|h)}{p(ao|h)} = \frac{m_{aoq_i}^\top p(Q|h)}{m_{ao}^\top p(Q|h)}. \tag{2.1}$$

This equation shows how a single test can be recursively updated in an elegant, closed-form way. Previously, we said that given the predictions of a set of core tests for a certain history h, any other

column in the same row could be computed as a weighted sum of $p(Q|h)$. Here, we see that in order to update state, only two predictions are needed: the prediction of the *one-step test* $p(ao|h)$, and the prediction of the *one-step extension* $p(aoq_i|h)$. Thus, the agent only needs to know the weight vectors m_{ao}, which are the weights for the one-step tests, and the m_{aoq_i}, which are the weights for the one-step extensions. We can combine the updates for all the core tests into a single update:

$$p(Q|hao) = \frac{M_{ao}p(Q|h)}{m_{ao}^{\top}p(Q|h)}$$

which allows us to recursively update state.

An agent does not need to learn a weight vector for every possible test in the system that it ever wishes to predict. If it has learned the weights for the one-step tests and the one-step extensions, these are sufficient to create a prediction for *any* arbitrary test. This is accomplished by rolling the model forward into the future. The prediction of an arbitrary test $t = a^1o^1 \cdots a^no^n$ can be computed as:

$$p(t|h) = m_{a^no^n}^{\top} M_{a^{n-1}o^{n-1}} \cdots M_{a^1o^1} p(Q|h).$$

This derivation can be arrived at by considering the system dynamics matrix (Singh et al., 2004), or by directly considering the parameters of an equivalent POMDP (Littman et al., 2002).

2.3.1 Learning a PSR Model

Numerous algorithms have been proposed to learn PSRs, but they all address the two key problems which need to be solved in order to learn a model of a dynamical system: the *discovery problem* and the *learning problem* (James and Singh, 2004). The discovery problem is defined as the problem of finding a set of core tests, and is essentially the problem of discovering a state representation. The learning problem is defined as the problem of finding the parameters of the model needed to update state, and is essentially the problem of learning the dynamical aspect of the system. In the case of linear PSRs, this is the m_{q_i} weight vectors for all of the one-step tests and the one-step extensions.

The discovery problem: The idea of linear sufficiency suggests procedures for discovering sufficient statistics: a set of core tests corresponds to a set of linearly independent columns of the system dynamics matrix, and so techniques from linear algebra can be brought to bear on empirical estimates of portions of the system dynamics matrix. Existing discovery algorithms search for linearly independent columns, which is a challenging task because the columns of the matrix are estimated from data, and noisy columns are often linearly independent (Jaeger, 2004). Thus, the *numerical* rank of the matrix must be estimated using a statistical test based on the singular values of the matrix. The entire procedure typically relies on repeated singular value decompositions of the matrix, which is costly. For example, James and Singh (2004) learns a "history-test matrix," which is the predecessor to the systems dynamics matrix. Their algorithm repeatedly estimates larger and larger portions of the matrix, until a stopping criterion is reached.

The learning problem: Once the core tests have been found, the update parameters must be learnt. Singh et al. (2003) presented the first algorithm for the learning algorithm, which assumes that the core tests are given and uses a gradient algorithm to solve the learning problem. The more common approach is with regression and sample statistics (see, for example, James and Singh, 2004). In these methods, once a set of core tests is given, the update parameters can be solved by regressing the appropriate entries of the estimated system dynamics matrix.

Some authors combine both problems into a single algorithm. For example, Wiewiora (2005) presents a method for learning regular form PSRs with an iterative extend-and-regress method, while McCracken and Bowling (2006) propose an online discovery and learning algorithm based on gradient descent. Rosencrantz et al. (2004) present TPSRs, which are like PSRs, but which find an uninterpretable basis for the systems dynamics matrix, as opposed to a basis composed strictly of columns.

Estimating the system dynamics matrix: Many learning and discovery algorithms involve estimating the system dynamics matrix. Generally, these algorithms estimate entries in a small subset of the matrix using sample statistics. In systems with a reset action, the agent may actively reset to the empty history in order to repeatedly sample entries (James and Singh, 2004). In systems without a reset, most researchers use the suffix-history algorithm (Wolfe et al., 2005) to generate samples: given a trajectory of the system, we slice the trajectory into all possible histories and futures (see the discussion of the suffix-history algorithm in Section 3.4 for more detail). Active exploration is also possible, as proposed by Bowling et al. (2006).

2.3.2 Other Results on PSRs

A variety of other results about PSRs have been obtained. For example, Rudary and Singh (2004) showed that more compact models can be created when nonlinearly sufficient statistics are allowed. James et al. (2005b) showed that memory and predictions can be combined to yield smaller models than can be obtained strictly with predictions; James and Singh (2005a) then showed that effective planning is possible with the resulting model. The idea of tests has been generalized to include options (Wolfe and Singh, 2006), set-tests and indexical tests (Wingate et al., 2007). Predictive representations have also been shown to be good bases for generalization (Rafols et al., 2005), and (Tanner et al., 2007) presented a method to learn high-level abstract features from low-level state representations.

There has been comparatively little work on planning in PSRs. James (2005) shows how several POMDP planning algorithms (such as exact value iteration and incremental pruning) can be translated directly into PSR terms, with equivalent computational complexity and optimality guarantees. It is generally believed among PSR researchers that any POMDP planning algorithm can be directly applied to PSRs, although there is no formal proof of this.

A variety of theoretical results have been obtained about PSRs. For example, it was shown early that

every POMDP can be equivalently expressed by a PSR using a constructive proof translating from POMDPs directly to PSRs (Littman et al., 2002). In fact, the resulting PSR is just as compact as the POMDP: a POMDP with n latent states can be captured by a linear PSR with n core tests, and an equivalent number of parameters. In fact, PSRs are strictly more expressive than POMDPs: it has been shown that "there exist finite PSRs which cannot be modeled by any finite POMDP, Hidden Markov Model, MDP, Markov chain, history-window, diversity representation, interpretable OOM, or interpretable IO-OOM" (James, 2005).

2.4 Other Models with Predictively Defined State

There are other models of dynamical systems which capture state through the use of predictions about the future.

2.4.1 Diversity Automaton

The diversity automaton of Rivest and Schapire (1987) is a model based on predictions about the future, although with some severe restrictions. Like the PSR model, diversity models represent state as a vector of predictions about the future. However, these predictions are not as flexible as the usual tests used by PSRs, but rather are limited to be like the e-tests used by Rudary and Singh (2004). Each test t_i is the probability that a certain observation will occur in n_i steps, given a string of n_i actions but *not* given any observations between time $t+1$ and $t+n_i$. Each of these tests corresponds to an equivalence class over the distribution of future observations.

Rivest and Schapire (1987) showed tight bounds on the number of tests needed by a diversity model relative to the number of nominal states a minimal POMDP would need to model the same system. Diversity models can either compress or inflate a system: in the best case, a logarithmic number of tests are needed, but in the worst case, an exponential number of tests are needed. This contrasts with PSRs, where only n tests to model any domain modeled by an n-state POMDP. Another significant restriction for diversity models is that they are limited to systems with deterministic transitions and deterministic observations. This is partly due to the state update mechanism used by the model, as well as the need to restrict the model to a finite number of tests by restricting it to a finite number of equivalence classes of future distributions.

2.4.2 Observable Operator Models

Observable Operator Models (OOMs) were introduced and studied by Jaeger (2000). Like PSRs, there are several variants on the same basic theme, making it more of a framework than a single model. Within the family of OOMs are models which are designed to deal with different versions of dynamical systems: the basic OOM models uncontrolled dynamical systems, while the IO-OOM models controlled dynamical systems. OOMs have several similarities to PSRs. For example, there are analogous constructs to core tests ("characteristic events"), core histories ("indicative events") and the system dynamics matrix. State in an OOM is represented as a vector of predictions about

the future, but the predictions do not correspond to a single test. Instead, each entry in the state vector is the prediction of some set of tests of the same length k. There are constraints on these sets: they must be disjoint, but their union must cover all tests of length k.

There is a significant restriction on IO-OOMs, which is that the action sequence used in tests must be the same for all tests. This restriction is needed to satisfy some assumptions about the state vector, but is severe enough that James (2005) gives an example of why this restriction results in systems which the IO-OOM cannot model, but which PSRs can. There are also variants of OOMs which do not use predictions as part of their state representation, which are called "uninterpretable OOMs," but there are no learning algorithms for these models (James, 2005). We refer the reader to the technical report by Jaeger (2004) for a detailed comparison of PSRs and OOMs.

2.4.3 Temporal-Difference Networks

The Temporal-Difference Network model of Sutton and Tanner (2005) is an important generalization of PSRs. In a TD-Net, state is represented as a set of predictions about the future, like a PSR. However, these predictions are explicitly allowed to depend on each other in a compositional, recursive way. This suggests that temporal difference algorithms could be used to learn the predictions, as opposed to the Monte-Carlo methods used by PSRs, and it is these algorithms which form the basis of the model. The recursive nature of the tests and the use of temporal difference methods in learning naturally generalizes to include multi-step backups of learning by introducing eligibility traces, to form TD(λ)-Nets (Tanner and Sutton, 2005a).

Although TD-Nets are theoretically attractive, they have not enjoyed the same rigorous analysis which PSRs have. Little is known about their representational capacity or the optimality of their state update mechanism. For example, published work on TD-Nets uses a general nonlinear state update mechanism related to a single-layer neural network, although this is not a fundamental component of the model. Other state updates could be used, and it is not clear how the state update relates to, say, the statistically optimal update dictated by Bayes law. PSRs, in contrast, explicitly begin with Bayes law as the foundation of their state update mechanism.

Empirically, TD-Nets have enjoyed about the same level of successes and failures as PSRs, with applications of the model being limited to rather small domains. While there has been less work done on TD-Nets in general, the development of learning algorithms for TD-Nets and PSRs have in some ways paralleled each other. For example, Tanner and Sutton (2005b) proposed to include some history in the state representation to aid learning in a manner that is reminiscent of the memory PSRs proposed by James and Singh (2005a), with improved learning results.

2.5 Conclusions

This chapter has introduced several of the concepts that will be essential to the development of our algorithms in later chapters. The themes sketched here – tests, predictions of the future, distribu-

tions over the future, and theoretical mathematical objects describing dynamical systems – will be repeated and refined throughout the thesis.

We have introduced the core problem addressed by this thesis, which is learning dynamical system models from data. We have briefly reviewed several models of dynamical systems with discrete observations, discussing PSRs in detail, and showing how they use predictively defined representations of state (specifically, the idea of tests) to model dynamical systems with discrete observations and actions. On the theoretical side, we have briefly discussed the idea of the system dynamics matrix, which will be a key inspiration to some of our later models. We have also surveyed some theoretical results demonstrating that the idea of capturing state with predictions about the future is fundamentally sound: PSRs are just as compact, and accurate as POMDPs, and strictly more expressive, and it is possible to use many different planning algorithms to control them optimally. We have also discussed the numerous learning algorithms for PSRs. Because the state and the model parameters are defined in terms of observable quantities, the learning algorithms are straightforward, and many of them use the system dynamics matrix in one form or another as the basis for learning.

We will next turn our attention to dynamical systems with continuous observations.

Chapter 3

Continuous PSRs

The models and learning algorithms of the previous chapter were all limited to discrete observations. This chapter presents the "Continuous PSR" model, which extends PSRs to the case of continuous observations. Chronologically, the Continuous PSR model was developed after the PLG family of algorithms (discussed in Chapters 4 - 6), combining ideas from PSRs and the KPLG and MPLG models. However, for the purposes of this thesis, we explain it now, because several of the ideas presented here facilitate a clean explanation of subsequent algorithms. Portions of this chapter were published in Wingate and Singh (2007b).

While this chapter presents the specific Continuous PSR model and learning algorithm, some of the fundamental questions it grapples with and the definitions it makes impact the entire thesis. For example, the question of sufficiency is immediately raised. How can we ensure that our state is sufficient for history? The theory of discrete PSRs uses the system dynamics matrix and derives the notion of predictive state as a sufficient statistic via the rank of the matrix. With continuous observations and actions, such a matrix and its rank no longer exist. In this chapter, we show how to define an analogous construct for the continuous case, called the *system dynamics distributions*, and use information-theoretic notions to define a sufficient statistic and thus state. These distributions describe the evolution of the system over time, exactly like the system dynamics matrix.

For the specific Continuous PSR model, we will also address the two basic problems of representing state and updating state. We will represent state with the predictions of core tests, like discrete PSRs, except that we replace all probabilities with densities. How do we find a good representation of state? We will cast this as an optimization problem: we will select a class of state representations, which we will parameterize, and we will frame the problem of finding a good state representation as a search problem over the parameter space, using tools from information theory to define an objective function. How do we update state? We tackle this by estimating transition distributions. Our entire algorithm will rely on estimating the system dynamics distributions from data.

These conceptual extensions require companion algorithmic extensions. We use four key ideas: first, to estimate the system dynamics distributions, we use kernel density estimation. Second, to measure sufficiency, we use a generalized form of mutual information based on quadratic Renyi entropy (Renyi, 1976). Third, to discover sufficient statistics, we use random sampling combined with

entropy optimization. Fourth, we show how Nystrom approximations and homotopy optimization yield an efficient implementation. The final combination of ideas has several appealing properties, one of which is a nice mathematical synergy among the elements.

We conclude with experiments showing that Continuous PSRs can be used to model agents in dynamical systems. We demonstrate the ideas on two example problems, one of which is a partially observable dynamical world consisting of an autonomous mobile robot. The agent has realistic perceptual and action models: features of camera images are observations, no a priori information about the effect of actions is given, and no automatic state is given.

3.1 Moving to Continuous PSRs

We begin by comparing key ideas in discrete PSRs and Continuous PSRs. The first problem is that when moving to continuous states and actions, it is no longer possible to order all possible histories and tests, simply because both observations and actions are real-valued. This means that we cannot define the system dynamics matrix, and hence we cannot define sufficiency in terms of its rank. Here, we outline our alternative. The reader may wish to compare these definitions with their PSR counterparts in Section 2.3.

Histories and tests: These are defined in exactly the same way as for discrete PSRs, except that both actions and observations may be continuous and vector-valued. If observations are vectors in $\mathbb{R}^3$ and actions are vectors in $\mathbb{R}^2$, for example, then a length three history is a vector in $\mathbb{R}^{15}$.

The continuous system dynamics vector: We define this somewhat differently than the discrete system dynamics vector: each entry represents one timestep, and so the n'th entry contains $p(F^n|\emptyset)$, which is the full distribution representing densities of tests of length n, measured from the null history (here, F^n is a mnemonic for "n-step future"). Like the system dynamics vector defined in Section 2.3, this vector is a complete description of a dynamical system.

The system dynamics distributions: The relationship between the discrete system dynamics vector and the discrete system dynamics matrix is analogous to the relationship between the continuous system dynamics vector and the system dynamics distributions, except that since all possible histories and tests are no longer enumerable, we instead work with all possible combinations of history length and future length.

The system dynamics distributions are defined by conditioning the continuous system dynamics vector on histories of increasing length. There is one distribution for each combination of a history length and a future length. We say that $p(F^n|h_m) = p(F^n, h^m)/p(h^m)$ is the density of a length n future from a length m history. These distributions play the same role as the system dynamics matrix: they give the density of any given future from any given history. We will often drop the superscripts n and m when no ambiguity results.

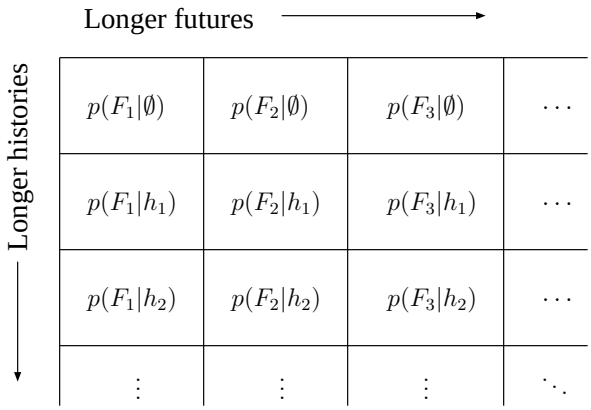


Figure 3.1: The system dynamics distributions. Each entry in the table represents a full distribution. There is one distribution for every combination of future length and history length.

We arrange all of these distributions into a table, which we show graphically in Figure 3.1. Like the system dynamics matrix, there are an infinite number of rows and columns in this table. Also like the system dynamics matrix, every dynamical system with continuous observations has a table of system dynamics distributions, which is a complete characterization of the system.

State: We represent state with a vector of predictions, like a discrete PSR:

$$p(O_1 = o_1^1, O_2 = o_2^1, \cdots | h_t, A_1 = a_1^1, A_2 = a_2^1, \cdots).$$

The only difference is that we replace probabilities with densities. More specifically, given a history h_t, our state at time t is an n vector, the j'th component of which represents the prediction of a specific test from h_t: $s_t^j = p(t_j|h_t)$. This prediction is a single number representing the point density of a particular future from the current history, and is not a full distribution. We collect all of the t_j's (which constitute our core tests) into the set T.

We have defined the form of our state representation, but how can we find the tests such that the state representation is sufficient for history? Defining sufficiency is the subject of the next section, and discovering a sufficient set of core tests is the subject of Section 3.5.

3.2 Using Information-Theoretic Sufficiency

Because of the move to continuous observations, the idea of using the linear independence of columns in the system dynamics matrix to define sufficiency is no longer applicable. To find a new concept of sufficiency, we turn to information theory. In discrete PSRs, the concept of linear sufficiency lead to natural discovery and learning algorithms; likewise, our information-theoretic view of sufficiency will aid us in discovering a good state representation.

Information theory was developed by Shannon (1948) in his seminal work on channel coding theory. Information theory is based on probability theory and statistics, and is intimately concerned

with random variables, efficient coding of those random variables, and the amount of information that one random variable conveys about another. As an engineering tool, it has found application in compression, cryptography, data communication over noisy channels, cybernetics, information retrieval, statistics and even gambling.

The most basic quantity in information theory is entropy, which is a measure of the uncertainty associated with a random variable X. Shannon entropy is defined as

$$H(X) = -\sum_x p(x) \log p(x),$$

where $0 \log 0 \equiv 0$. This quantity is maximal when X is maximally nondeterministic, which occurs when X is distributed uniformly. Conversely, when X is deterministic, the entropy is minimal. The natural generalization of entropy to the case of continuous densities is known as differential entropy, and is defined in the obvious way.

The definition of entropy also generalizes to the case of multiple variables by summing over all possible joint configurations:

$$H(X, Y) = -\sum_{x,y} p(x, y) \log p(x, y),$$

and also generalizes to the case of conditional distributions in the natural way:

$$H(X|Y = y) = -\sum_x p(x|y) \log p(x|y).$$

For our purposes, we will be primarily interested in *mutual information*, denoted $I(X; Y)$, which is the amount of information that can be obtained about one random variable by observing another. Intuitively, mutual information quantifies the reduction in uncertainty about X when Y is known, and vice-versa: if $I(X; Y)$ is large, that means that knowing X will result in a Y that has low entropy (that is, we know what value it will take with high probability), and conversely, if $I(X; Y)$ is zero, it means that knowing X does not reduce the uncertainty in Y at all.

There are several equivalent definitions of mutual information, but for our purposes, we will prefer the interpretation of information as a sum of entropies:

$$\begin{aligned} I(X; Y) &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \\ &= H(X) + H(Y) - H(X, Y) \\ &= I(Y; X). \end{aligned}$$

Information is symmetrical, and is defined additively in terms of entropies. Additionally, if it is the

case the $p(X|Y)$ is a function of Y only through $f(Y)$, then $I(X,Y) = I(X, f(Y))$.

To introduce the concept of information in a dynamical system, we will start by treating history and the future as random variables, whose joint distribution is described by the system dynamics distribution $p(F, H)$. The idea behind our definition of sufficiency and the consequent algorithms is that there is a certain amount of information conveyed by history about the future. That is, history (as a random variable) disambiguates the future (as a random variable) to a certain extent, in exactly the same any two non-dynamical random variables may convey information about each other.

This dynamical information has been called predictive information (Bialek et al., 2001), excess entropy, effective measure complexity, stored information, and so on (Shalizi and Crutchfield, 2001). Bialek also discusses theoretical characterizations of the limiting possibilities of any possible state representation in any dynamical system as a function of this predictive information, and showed that a consequence of the definition is a natural measure of the dimension of the system. He also demonstrates how predictive information can be used as a measure of the stochastic complexity of the system, and provides interesting results on the relationship between information between the past and the future as well as information between the future and the past. However, his work is highly theoretical, and does not translate into models or learning algorithms. Based on these ideas, Shalizi and Shalizi (2004) defined a learning algorithm for dynamical systems which measures the information between history and the future, and attempts to find a state representation that conveys the same information.

We will adopt a similar approach. We have defined state as a sufficient statistic for history, and in fact, we can think of state as a *function* of history: given a particular history h_i and a dynamical model, there is some function f which summarizes that history into a state:

$$s_i = f(h_i).$$

It may be unusual to think of f as operating on an entire history; in most models, it may be more natural to think of f as a composition of a simpler function g which operates on single action and observation and a previous state:

$$s_i = f(h_i) = g(a_t o_t, g(a_{t-1} o_{t-1}, g(\cdots, g(a_1 o_1, s_0)))).$$

However, we will only refer to the function f. Notice that this function encapsulates the entire model, including the state representation and any dynamical parameters.

It is well-known in information theory that no function of a random variable can increase the information between that variable and another variable. This is known as the *data processing inequality,* which states that

$$I(X;Y) \geq I(X; f(Y))$$

with equality if and only if $f(Y)$ is sufficient for Y (Kullback, 1968; Cover and Thomas, 1991). It is this fact that we will use as the basis for our measure of sufficiency, which we now define.

Sufficient statistics: We will say a function f – which encapsulates a choice of state representation and state update mechanism – captures state if

$$I(F^n; H^m) = I(F^n; S = f(H^m))$$

for all n, m. If f can be found such that equality is achieved, the state resulting from $f(h_t)$ has summarized all of the information about the past which is relevant for predicting the future. This also gives a natural definition of approximate state: an approximate state representation is one for which $I(F^n; H^m) > I(F^n; S = f(H^m))$ for some n, m. In order words, state has failed to capture all of the information in history relevant to disambiguating the future.

There are many choices which could be made for the mapping f – state could be summarized as either latent variables or predictively defined variables. The above definition of sufficiency applies to any state representation.

3.3 Measuring Information

How can we measure information between history and the future, or between state and the future? In Section 3.5 we will learn a model of a dynamical system by maximizing the mutual information between state S and the future F. Recall that information can be expressed as $I(F; S) = H(F) + H(S) - H(F, S)$, which is a sum of entropies. When Shannon's entropy measure is used to compute information, the result is known as Shannon information. Unfortunately, it is too difficult to optimize Shannon information directly – computing the needed entropies relies on knowing the underlying densities, and for many densities, we cannot compute Shannon's entropy easily.

However, Kapur (1994) has argued that if the aim is not to compute an exact value of information, but rather to extremize information, Shannon information does not need to be used. A generalized measure of information can be optimized instead, and if this generalized measure has the same maxima and minima as Shannon information, the same results will be obtained. Ideally, it will also have more favorable properties (for example, it might be computationally cheaper).

Shannon information can be considered a divergence between $p(X, Y)$ and $p(X)p(Y)$, or it can be considered a sum of individual entropies, minus a joint entropy. Thus, two different kinds of generalized measures have been proposed, based on each interpretation. Generalized divergences have been the most popular. Kapur (1994) presents a number of possibilities; other proposals include a divergence measure based on the Cauchy-Schwarz inequality (Principe et al., 1999); a divergence based on the triangle inequality (Principe et al., 1999); a measure called "quadratic mutual information" (Torkkola, 2003); and methods based on generalized KL information gain (Borland et al., 1998). He et al. (2003) proposed the Jensen-Renyi divergence. Among generalized entropies, the

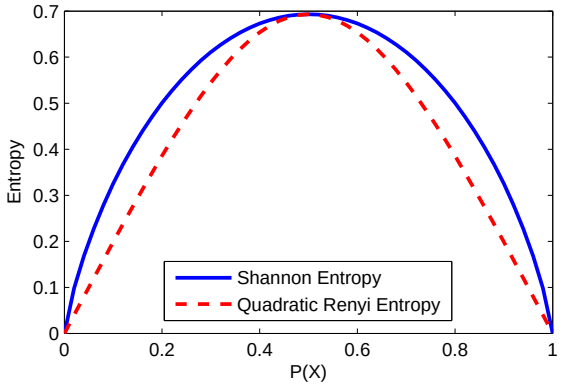


Figure 3.2: A comparison of Shannon and Renyi entropies. The horizontal axis is p(X) for a binary random variable, and the vertical axis is the corresponding entropy for both measures.

most well-known is quadratic Renyi entropy (Renyi, 1976). Hild et al. (2001) proposed to use a sum of Renyi entropies, minus a joint Renyi entropy. More recently, authors have begun to consider measures based on Tsallis entropy (also known as Havrda-Chervat entropy; Borland et al., 1998).

Finding an easily optimizable, generalized information measure while making few assumptions about the density is a problem that has been dealt with by the entropy optimization community. The solution that has emerged in the literature has been to 1) use a generalized information measure based on generalized entropies, and 2) use a kernel density estimate with a Gaussian kernel to model the underlying densities (Principe et al., 1999; Torkkola, 2003; Hild et al., 2001). In our later learning algorithm, this is why we will choose to use Gaussian kernel density estimation.

A popular generalized entropy is Renyi's entropy (Renyi, 1976), which is defined as:

$$H_{R_\alpha}(X) = \frac{1}{\alpha - 1} \log \int p(X)^\alpha dX$$

This measure generalizes Shannon's entropy, because in the limit as α approaches 1, Shannon's entropy is recovered. The choice of $\alpha = 2$ yields quadratic Renyi entropy:

$$H_{R_2}(X) \quad = \quad -\log \int p(X)^2 dX \tag{3.1}$$

which we will write as $H(X)$ when it is clear from context that Renyi entropy is intended.

Quadratic Renyi entropy has the same maxima and minima as Shannon entropy (as shown graphically in Figure 3.2), but importantly for our purposes, it has favorable computational properties. In conjunction with kernel density estimation and a Gaussian kernel, Eq. 3.1 can be evaluated in closed form, as explained later.

Several authors have used the idea of information maximization to solve machine learning problems;

collectively, this field has become known as "entropy optimization." Examples include image edge detection (Hamza, 2006), classification (Torkkola, 2003), speech processing (Schraudolph, 2004), image registration (He et al., 2003), and unsupervised learning (Principe et al., 1999). While other authors have explicitly tried to maximize information about the future while building models of dynamical systems (Shalizi and Shalizi, 2004), we are unaware of others who have used techniques from entropy optimization to do so.

3.4 Estimating the System Dynamics Distributions

In the case of PSRs, many learning algorithms revolve around the system dynamics matrix and its empirical estimate. Similarly, the learning algorithm we will present in Section 3.5 will revolve the system dynamics distributions and their empirical estimates. In this section, we describe how we estimate them, and how we collect the necessary data.

3.4.1 Collecting Data with the Suffix-History Method

To estimate the system dynamics distributions, we need samples from them. Where does the data come from? We will assume that we have been given one long trajectory of data, consisting of T actions and observations $a_1o_1, a_2o_2, \cdots, a_To_T$.

To estimate $p(F, H)$, we need samples of the joint distributions of history and future. We generate these samples using the suffix-history algorithm (Wolfe et al., 2005). This process will be used throughout the thesis, so we explain it in detail. The intuition is simple: given a long trajectory of actions and observations $a_1o_1, \cdots, a_To_T$, we slice the trajectory into all possible combinations of history and future. For example, a length four trajectory is sliced into the following samples:

$$
\begin{array}{lcl}
h^0 = \{\} & ; & f^1 = a_1o_1 \\
h^0 = \{\} & ; & f^1 = a_2o_2 \\
 & \cdots & \\
h^0 = \{\} & ; & f^2 = a_1o_1a_2o_2 \\
h^0 = \{\} & ; & f^2 = a_2o_2a_3o_3 \\
 & \cdots & \\
h^0 = \{\} & ; & f^4 = a_1o_1a_2o_2a_3o_3a_4o_4 \\
 & \cdots & \\
h^1 = a_1o_1 & ; & f^1 = a_2o_2 \\
h^1 = a_2o_2 & ; & f^1 = a_3o_3 \\
 & \cdots & \\
h^1 = a_1o_1 & ; & f^2 = a_2o_2a_3o_3 \\
h^1 = a_2o_2 & ; & f^2 = a_3o_3a_4o_4 \\
 & \cdots & \\
h^4 = a_1o_1a_2o_2a_3o_3 & ; & f^1 = a_4o_4.
\end{array}
$$

This results in samples from the distributions $p(F^1, H^0)$, $p(F^2, H^0)$, $p(F^3, H^0)$, $p(F^4, H^0)$, $p(F^1, H^1)$, $p(F^2, H^1)$, $p(F^3, H^1)$, $p(F^1, H^2)$, $p(F^2, H^2)$, and $p(F^1, H^3)$. Of course, these samples are not independent of each other.

We assume that the system is stationary and ergodic, and that the behavior policy used to generate the samples explores sufficiently to cover the entire state space.

3.4.2 Kernel Density Estimation

We assume that we are given some samples from $p(F, H)$ and $p(F, S)$, and that we wish to infer the approximate distributions. This is a density estimation problem, which is well studied (Hastie et al., 2001). We choose to use kernel density estimation with a Gaussian kernel, which is a choice that is motivated by mathematical convenience: it will enable us to compute test predictions efficiently, as well as derive closed-form, differentiable, and efficiently approximable expressions for information and its derivatives.

Given a set of samples $x_1, \cdots, x_n \in \mathbb{R}^d$ of the random variable X, our estimate of $p(X)$ is

$$p(X = x) = \frac{1}{n} \sum_{j=1}^{n} K(x, x_j; \sigma_j)$$

with a Gaussian kernel:

$$\begin{aligned} K(x, x_j; \sigma_j) &= G(x - x_j; \sigma_j) \\ &= 1/(\sqrt{(2\pi\sigma_j)^d}) \exp\left\{-(x - x_j)^\top (x - x_j)/2\sigma_j\right\} \end{aligned}$$

where we have assumed the use of a spherical covariance matrix $\sigma_j I$, and where d is the dimension of the variable X. Because we are using spherical Gaussians, we can write similar expressions for joint densities of $p(X, Y)$, assuming we are given joint samples (x_j, y_j):

$$p(x, y) = \frac{1}{n} \sum_{j=1}^{n} G(x - x_j; \sigma_j^X) G(y - y_j; \sigma_j^Y).$$

We will estimate every needed distribution using this technique.

3.5 Learning Models of Continuous PSRs

We are now prepared to discuss our learning algorithm for Continuous PSRs. Our goal is to learn a Continuous PSR directly from observed sequences of actions and observations. Like discrete PSRs, the two key problems are the discovery problem (finding a good state representation) and the learning problem (learning how to update state). We discuss the discovery problem in Section 3.5.1 and learning how to maintain state in Section 3.5.3. We assume that we have been given N samples from the system dynamics distributions as discussed in Section 3.4, which we denote (h_i, f_i).

3.5.1 Discovering Core Tests

In the Continuous PSR model, selecting a state representation is equivalent to picking a set of core tests, so we now address core test discovery. There are two elements to discovery: determining how many tests are needed, and determining which actions and observations should comprise the tests. It is possible for discovery algorithms to be concerned with discovering a *minimal* set of statistics. We do not address the issue of minimality; rather, we supply many more tests than needed, and focus on discovering the parameters of the tests.

We will frame discovery as an optimization problem by defining a search space and an objective function. We will additionally show that we can compute the gradients of the objective function, which means that a host of gradient-based optimization methods become available to help search for a good state representation.

- **The search space:** In order to define the state representation for a Continuous PSR, a set of core tests must be specified. Each core test is defined by the sequence of actions and observations in it. To be more specific, let us expand the definition of the prediction vector:

$$p(T|h_t) = \begin{bmatrix} p(O_1 = o_1^1, O_2 = o_2^1, \cdots | h_t, A_1 = a_1^1, A_2 = a_2^1, \cdots) \\ p(O_1 = o_1^2, O_2 = o_2^2, \cdots | h_t, A_1 = a_1^2, A_2 = a_2^2, \cdots) \\ \cdots \\ p(O_1 = o_1^n, O_2 = o_2^n, \cdots | h_t, A_1 = a_1^n, A_2 = a_2^n, \cdots) \end{bmatrix}$$

  (not every test need to be at least length two, as shown in this example; this was only done to clarify the pattern). Our goal is to find the actions and observations which comprise each test.

  We will treat the actions and observations in each test as parameters. The first test, for example, is defined as $p(O_1 = o_1^1, O_2 = o_2^1, \cdots | h_t, A_1 = a_1^1, A_2 = a_2^1, \cdots)$, so the parameters of this test are a_1^1, o_1^1, etc. Each test t therefore has $\text{length}(t)(\dim(\mathcal{O}) + \dim(\mathcal{A}))$ parameters.

  We will collect all of the actions and observations which must be specified for all of our core tests into the vector θ. This means that changing entries in θ is equivalent to specifying a different set of core tests, and therefore a different state representation. We do not constrain the values of θ in any way; in particular, we allow tests to refer to any possible future, whether or not it is possible under the true dynamics.

- **The objective function:** To define an objective function, we return to the information theoretic ideas of Section 3.2. Section 3.2 introduced the idea that we can treat state as a function of history, and that we can measure information between state and the future, as $I(F; S = f(H))$.

  We pointed out that the function f encapsulates both the choice of state representation as well as any dynamical parameters needed. In our case, the state representation is governed by the vector θ – any setting of θ represents a choice of state representation. We therefore make f's

dependence on θ explicit: state is a function of history and a specific state representation, as $s_i = f(h_i; \theta)$. The dependence of information on a state representation can be made similarly explicit: $I(F; S = f(H; \theta))$. We will use this function as our objective function: our goal is to maximize the information content of our state representation by tuning the entries in the parameter vector θ.

To summarize our strategy for solving the discovery problem: we will treat the actions and observations comprising our core tests as a vector of parameters θ, and we will attempt to find the best setting of those parameters by maximizing $I(F; S = f(H; \theta))$. To solve the discovery problem, then, we need three things:

1. We need to be able to compute the function $s_i = f(h_i; \theta)$. That is, given a particular history h_i and a candidate state representation θ, we must be able to compute the predictions of the tests defined by θ. This is also a learning problem, but because our state representation is defined in terms of densities, it reduces to a density estimation problem. Our strategy will be to explicitly estimate the system dynamics distributions to compute these predictions.

2. We need to be able to compute (or estimate) $I(F; S = f(H; \theta))$ for any choice of θ. Since $I(F; S) = H(F) + H(S) - H(F, S)$, this implies that we need entropy estimates of the distributions of the random variables F and S, as well as their joint distribution. Our strategy will be to generate joint samples from $p(F, S)$ by transforming our training samples (f_i, h_i). For each h_i, we can compute a state sample $s_i = f(h_i; \theta)$, which can then be paired with f_i. Thus, (f_i, s_i) become samples from the joint distribution $p(F, S)$, which we can estimate with kernel density estimation.

3. We need to be able to search the space of possible representations (that is, settings of θ) to find the maximal value of information. Our approach will be to compute the gradient of information with respect to the parameters θ, and perform gradient ascent:

$$\theta = \theta + \eta \frac{\partial I}{\partial \theta} = \theta + \eta \sum_i \frac{\partial I}{\partial s_i} \frac{\partial s_i}{\partial \theta} \tag{3.2}$$

   where η is a learning rate. In Appendix A, we show how to compute these derivatives.

At this point, we wish to highlight the fact that our choices of density estimate and information measure have worked nicely together. By choosing to use kernel density estimation with a Gaussian kernel and quadratic Renyi entropy, we were able to compute closed-form expressions for information. Furthermore, we were able to compute the derivatives of information simply.

This completes our approach to discovery. The algorithm is summarized in Figure 3.3.

Algorithm DISCOVER-CORE-TESTS

Input: samples from $p(F, H)$ as (f_i, h_i)

Initialize

- Estimate $p(F, H)$
- Select an initial set of core tests θ

Repeat

- Given a set of core tests θ
- For each (f_i, h_i), compute $s_i = f(h_i; \theta)$
- Using samples (s_i), estimate $p(S)$
- Using paired samples (f_i, s_i), estimate $p(F, S)$
- Measure $I(F, S)$
- Compute gradient $\partial I / \partial \theta$ by computing gradients $\partial I / \partial s_i$ and $\partial s_i / \partial \theta$.
- Improve representation with steepest ascent: $\theta = \theta + \eta \frac{\partial I}{\partial \theta}$

Until ($I(F, S)$ is maximized **)**

Figure 3.3: An algorithm for discovering core tests in Continuous PSRs. The algorithm maximizes the information between state and the future.

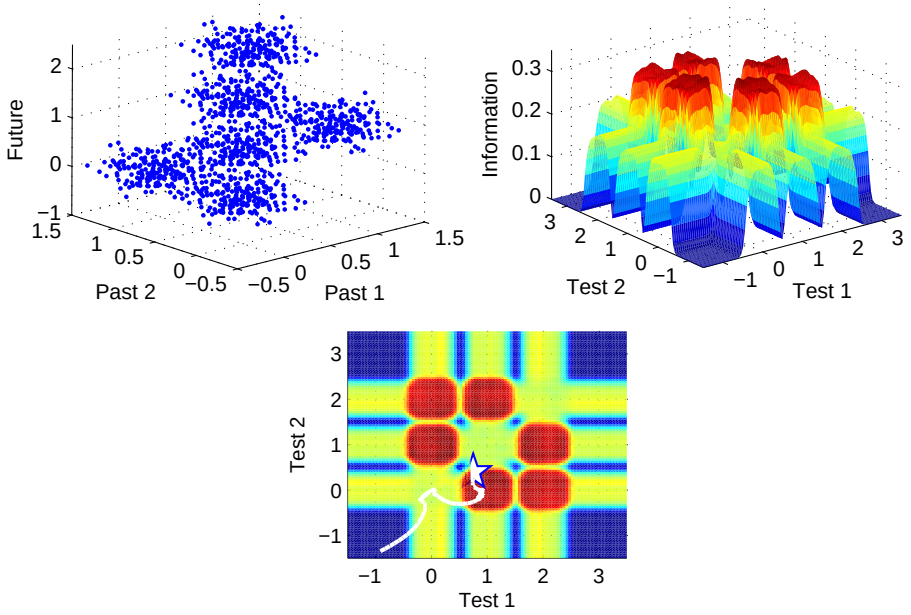


Figure 3.4: Example information landscape. A toy data distribution (top left), the resulting information landscape as a function of test value (top right) and the trajectory taken by the gradient optimizer (bottom).

3.5.2 Example: A Toy Data Set

Here we consider a toy data set to illustrate the concepts of information and gradients, and to clarify exactly what the parameters are that we are trying to find. Consider one particular system dynamics distribution $p(F^1, H^2)$ from an unspecified dynamical system. The system is uncontrolled, and observations are one-dimensional, so the distribution is three-dimensional, with two of the dimensions corresponding to "history" dimensions and one dimension corresponding to a "future" dimension. Samples from the distribution are shown in the top left panel of Figure 3.4 (we are only pretending that this data comes from a dynamical system; in reality, the data is from six Gaussian clusters in $\mathbb{R}^3$).

Suppose we decide to use two tests to summarize history. We will denote the prediction of test 1 as $p(F = l|H)$, and the prediction of test 2 as $p(F = k|H)$. Thus, our state is two-dimensional, and there are two parameters: l and k. Given particular values for l and k, we can compute the mutual information between F and S.

We wish to find the two best parameters to maximize $I(F|h; S = f(h))$. The top right panel of Figure 3.4 shows mutual information between state and the future as a function of the parameters. This information landscape highlights a few points of interest: values for l and k which are very far away from the high-density areas of the data (say, $l = -1, k = -1$) have low information content. It also shows that when l and k have the same value, no new information is added – in other words, the predictions are redundant.

The bottom element of Figure 3.4 shows the results of the gradient optimization starting the tests at $[l = -1, k = -1.4]$, and ending at the star (it does not climb quite as one might expect because of the use of homotopy optimization, as discussed later). The gradient optimizer indeed moves the tests from regions of low information to regions of high information.

3.5.3 Updating State

We now address the question of learning how to update state. We assume that we have some state s_t, which is defined as the the predictions of a set of core tests $p(T|h_t)$. Assume that we have received a new action and observation, and that we must update state to compute $p(T|h_t ao)$.

The natural way to do this is with Bayesian inversion. By direct analogy to Eq. 2.1, we can perform the state update by updating each individual test:

$$p(t_j|h_t ao) = \frac{p(aot_j|h_t)}{p(ao|h_t)},$$

where, like the discrete PSR, the numerator represents the one-step extensions of the core tests and the denominator represents the one-step tests. From any given history, we could compute these densities estimates of the system dynamics distributions. The only problem is that we will only have estimates of a finite number of distributions, and we need a state update mechanism which works for arbitrarily long amounts of time. We therefore prefer to find a recursive solution: we wish to update state in terms of previous state. Since state is sufficient for history, we can equally well model

$$\begin{aligned} p(t_j|h_t ao) &= p(t_j|s_t, ao) \\ &= \frac{p(aot_j|s_t)}{p(ao|s_t)} \\ &= \frac{p(s_t, aot_j)}{p(s_t, ao)}, \end{aligned}$$

where s_t is the state corresponding to history h_t. This bears a strong resemblance to the discrete PSR state update. Like discrete PSRs, we need to compute one-step tests, except that we model these jointly with state as $p(s_t, ao)$. Similarly, we need to compute the one-step extensions of the core tests, which we also model jointly with the state, in $p(s_t, aot_j)$. To compute the needed densities, we require estimates of both $p(s_t, F^1)$ and $p(s_t, F^{n+1})$, where n is the length of the longest core test.

Fortunately, we already have all of the needed information: in our discovery algorithm, we assumed we were given samples (f_i, h_i), and that we computed $s_i = f(h_i; \theta)$ for each sample. This resulted in joint samples of (f_i, s_i), which we used to compute information. Here we see a pleasing efficiency: the same estimate can be used to update state by explicitly estimating $p(F, S)$. This is a consequence of the grounded nature of the state representation. Our discovery algorithm in Sec-

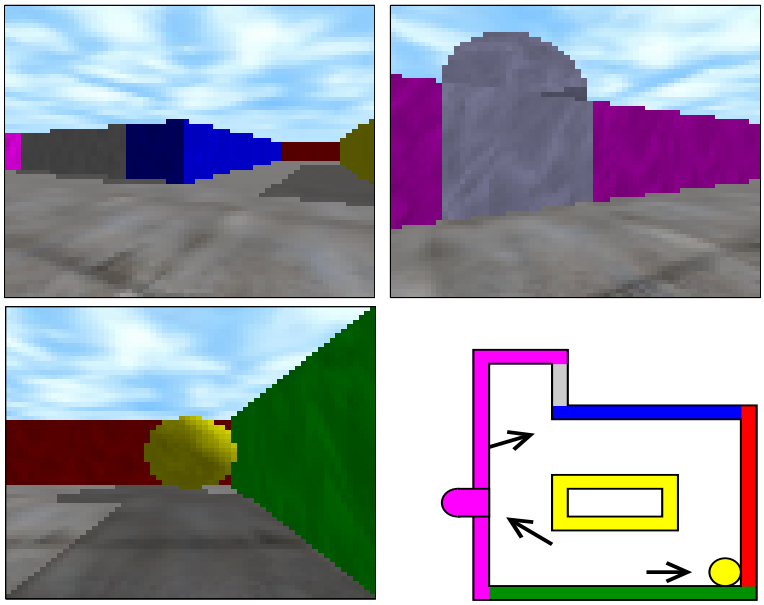


Figure 3.5: The autonomous robot domain. Arrows on the map correspond to viewpoints.

tion 3.5.1 and our method for learning how to update state presented here constitute the complete learning algorithm.

3.6 Experiments and Results

We have introduced several new ideas and made many design choices as part of the Continuous PSR algorithm. Here, we present some experiments designed to explore the decisions we have made. We tested on two problems: a bouncing ball, and a simulated autonomous robot.

The Bouncing Ball. The first domain is an uncontrolled, nonlinear, two-dimensional dynamical system consisting of a ball bouncing. The ball bounces vertically, with a damped restitution when it strikes the floor. True hidden state is the position and velocity of the ball; only position is observed.

To model this system, we made an assumption: we use a suffix-history with a history of length 3 and a future of length 3 (we also model the one-step extended distribution with a history of length 3 and a future of length 4 for the Bayesian inversion state update). This is a choice based on intuition and experimentation (in general, more sophisticated methods of selecting which distributions to use are needed). Different experiments used different numbers of tests, as explained later.

We trained on 2,000 data points. We initialized the state of the system to a random position and velocity, and ran the system for several timesteps; we then sliced the resulting trajectory into samples using suffix-history.

The Autonomous Robot Domain. The second domain is more challenging: a simulated autonomous mobile robot in a 2D maze. The domain is controlled, nonlinear, and partially observable;

no a priori knowledge about the domain is given to the agent. The robot has two continuous actions (the amount by which to rotate and amount by which to move forward/backward), and continuous state coordinates (position x, y and orientation θ). The robot is located in a maze with obstacles and brightly colored walls. The observation is generated as follows: the agent's camera initially samples a 64x64 full color image, but the agent extracts a single feature from the image: the dominant color in the center of the image (done by convolving with a Gaussian). Observations are therefore three dimensional (consisting of RGB color values). With full camera images, about 80% of the states can be disambiguated through an observation, but with the reduction to a single color, the observability is severely reduced. Figure 3.5 shows representative camera images, as well as the map used. All actions are deterministic.

The training data is a single long trajectory of actions and observations (100,000 samples, generated with a movement policy that was a smoothed version of random exploration). Again, we assumed that length 3 histories and length 3 futures were sufficient. In this case, both history and future are 15-dimensional vectors (3 steps of history x (2 action dimensions + 3 observation dimensions)). The gradient optimizer used a subset of 10,000 samples; testing used the full 100,000 (either size data set is feasible with [and *only* with] the Nystrom approximations discussed in Section 3.6.2).

Error measure. We evaluate based on mean-squared error of the one-step predictions. That is, at each timestep, the agent is asked to predict the expected next observation. The true observation is given; there is some error, which we square. The mean is taken over the length of the test sequence. The absolute value of the MSE is not important; but rather the difference before and after application of the gradient optimizer.

Implementation. We used the optimizations discussed in Section 3.6.2. We used a Nystrom-based gradient optimizer with 100 landmarks. We used the same homotopy schedule for both problems ($\lambda = [500, 200, 100, 60, 20, 5, 1]$), and set stepsizes such that the norm of the gradient vector was between 0.5 and 0.01 (depending on λ). All samples used the same covariances; this simplifies the math further.

3.6.1 Results

In this chapter, we have made an implicit assumption: that if we could somehow maximize information, we would have a better state representation, and that that representation would allow us to make better models of the world. We first test that hypothesis directly.

Is there is a correlation between higher information (between state and the future) and lower MSE of one-step predictions? To explore this, we sampled 1,000 random state representations in the ball domain. Each state representation consists of two three-step tests. Since there are no actions in this domain, each test requires three parameters to describe it, resulting in a total of six parameters. For each representation, we built a model by estimating the distributions needed to compute information and update state. We then used the model to make one-step predictions, and measured the resulting

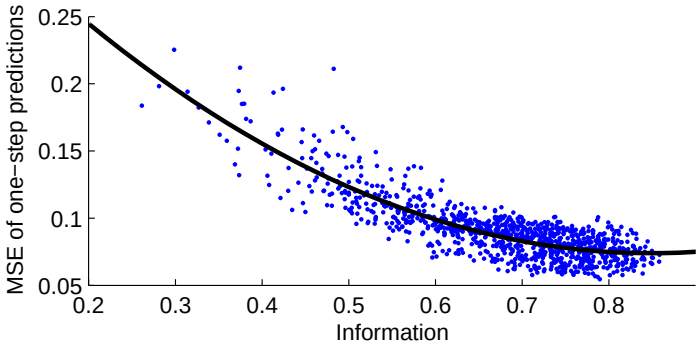


Figure 3.6: Continuous PSR empirical results. Shown is information (horizontal axis) vs. one-step MSEs (vertical axis) in the bouncing ball domain. Each point is a randomly sampled state representation.

MSE. We also measured the information content of states with the future, again using our estimates of the system dynamics distributions.

Figure 3.6 plots the resulting MSEs versus information (and is fit with a 3rd degree polynomial). There is an obvious correlation: as information increases, MSE decreases. There is some variance in the MSEs: with an information content of 0.8, for example, the MSE could be as low as 0.05 or as high as 0.1. This result suggests that the idea of using Renyi entropy to solve the discovery problem is sound.

We next ask: does the information gradient optimizer work? We explored this question for both domains. For each domain, we fixed the number of core tests to n. We randomly sampled a set of core tests by sampling n random sequences of actions and observations from the training data (this effectively results in a random θ). We then built a model, and computed the MSE of using θ to generate states. We then optimized the tests with the gradient method, used the improved tests to generate states, and again computed the MSE. This was done for different numbers of core tests.

Figure 3.7 shows results for the ball domain, while Figure 3.8 shows results for the robot domain. The number of tests used is the horizontal axis, while MSE of random and optimized representations is shown on the vertical axis. For a given number of core tests, two box-and-whisker plot are drawn. The one on the left represents the distribution (mean, 1st and 3rd quartiles as well as maximum and minimum) of MSEs for the random representations. The one on the right represents the same information for the optimized representations.

The results are very encouraging. Both figures demonstrate the same behavior: not only does the optimizer consistently find tests which generate lower MSEs, it also reduces variance in the MSE. Consider Figure 3.7, for 4 core tests. The mean MSE of random representations is about 0.1, and the minimum ever found is about 0.07. In contrast, the mean MSE for the optimized tests is about 0.06, which is lower than any random representation. The optimized tests also show very little variance.

Figure 3.8 suggests another conclusion as well, which is that using a large number of randomly sampled tests often results in a reasonable model. Optimizing tests always helps, but not as much as adding more tests. For example, two optimized tests performs consistently worse than 15 randomly sampled tests. It is unclear if this represents a general principle or not: on the one hand, it becomes more likely that a larger number of randomly sampled tests has the "right" tests inside of it. On the other hand, this also increases the dimension of the state; since so many of our estimates are built around Gaussian kernel density estimation, and Gaussians are known to be poor density estimates in high-dimensional spaces, this may adversely affect the performance of the algorithm.

The extremely low variance in the MSE associated with the optimized representations is because the homotopy optimizer was able to consistently locate almost the same points, regardless of initial conditions. In the ball domain, the variance is particularly high for the randomly sampled tests. Sometimes, a bad set of tests can result in a catastrophic run of the system, resulting in very high MSE; the optimized tests never showed this behavior. Figure 3.9 illustrates this behavior. The random tests, with few exceptions, never performed as well as the optimized tests. It is possible that this is because the optimized tests are not constrained to lie on the manifold of observed trajectories, but more research is needed to investigate that hypothesis.

Just how good are the reported MSEs? Figure 3.10 qualitatively answers this question in the robot domain by showing the agent's predictions before and after the application of the gradient optimizer. We see that before optimization the predictions made by the model are flat, and bear little resemblance to the actual observations. After optimization, there is a marked improvement, with predictions qualitatively following the observations, although there is still room for improvement.

3.6.2 Practical Considerations

We found that learning from large datasets was impossible without the addition of additional computational tools. We found the following techniques indispensable.

Homotopy optimization. As with all gradient methods, ours is guaranteed to only find a local maximum. However, the local maximum can be improved by "smoothing" the information landscape, and finding a local optimum of the smoothed landscape, and then gradually unsmoothing the landscape while continuing to optimize. This is known as homotopy optimization (or deformation optimization) because it uses a homotopy, which is a continuous transformation of an easy optimization problem into a hard one.

We accomplish this by smoothing test predictions: instead of computing $p(T|h) = \sum G(T - f_k|h, \sigma^T)$, we compute $p(T|h) = \sum G(T - f_k|h, \lambda\sigma^T)$. Using λ to scale the variance effectively makes test predictions look more similar, which has the side effect of smoothing the information landscape. By gradually reducing λ while taking gradient steps, a much better optimum is achieved. An example of the resulting information landscapes is shown in Figure 3.11.

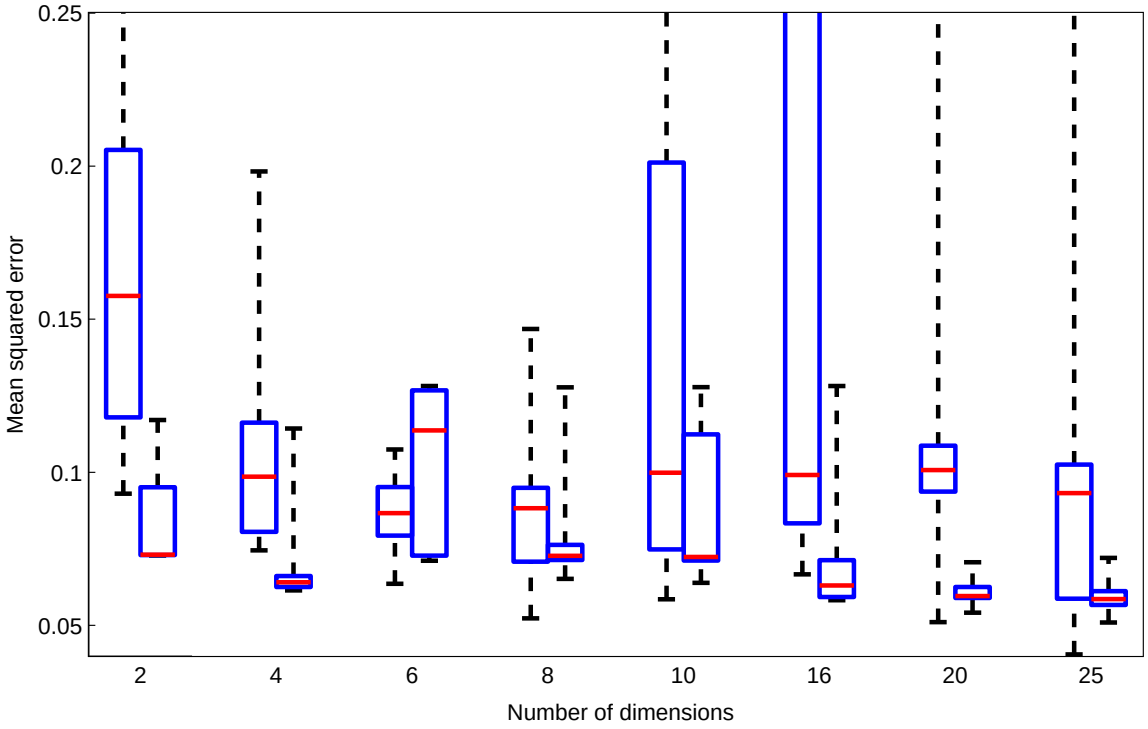


Figure 3.7: Improving tests on the ball domain. The vertical axis shows MSE of one-step predictions. The horizontal axis shows the number of tests used in the state representation. For each number of tests, two bars are drawn: the one on the left is a box-and-whiskers plot representing the distribution of randomly chosen test parameters. The right bar shows the distribution of the corresponding optimized parameters. The optimized representations generate models with lower MSE, and the representations do not have as much variance as the random ones.

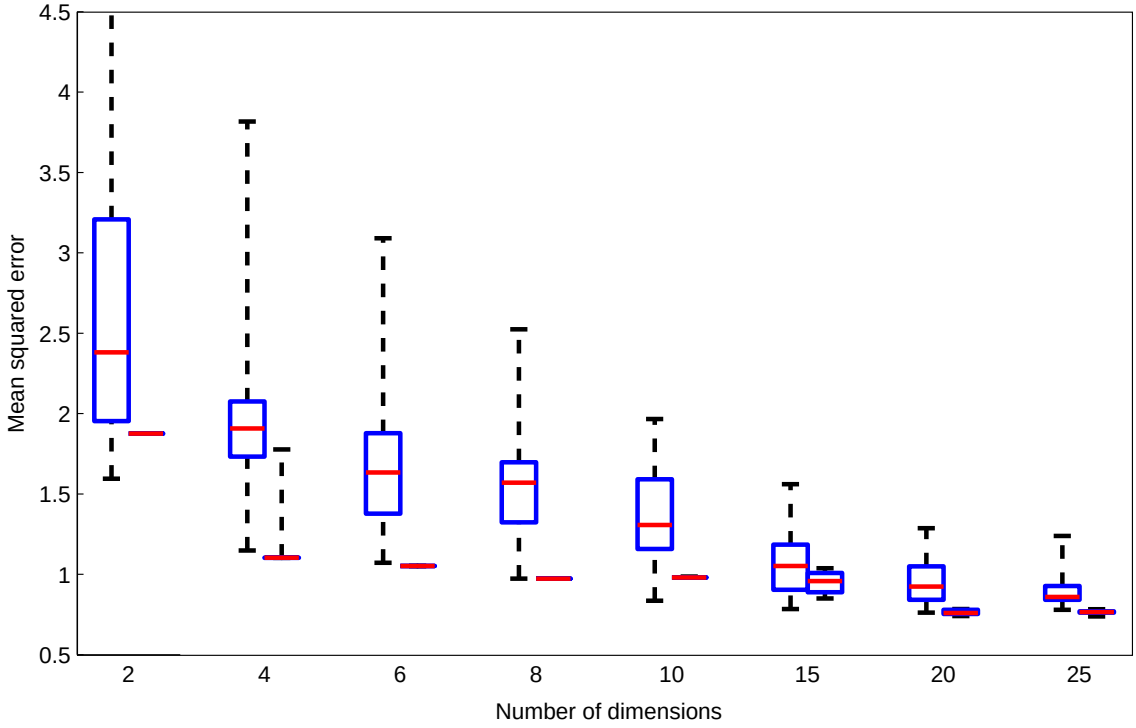


Figure 3.8: Improving tests on the robot domain. The interpretation is the same as in Figure 3.7. There is almost no variance in the optimized representations, which is a consequence of the homotopy optimizer.

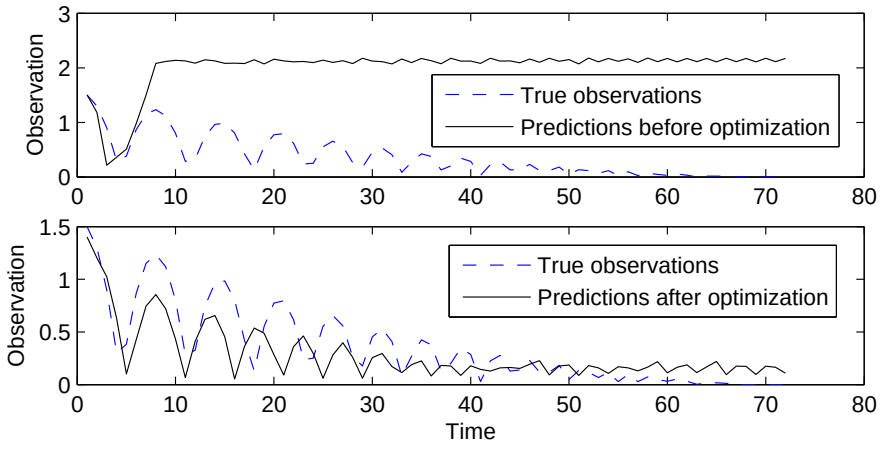


Figure 3.9: A catastrophic run with a Continuous PSR. The top panel shows a catastrophic run resulting from using random tests to build a model. The state reaches a meta-stable state in which incorrect predictions are made. The bottom panel shows the corresponding optimized tests, which avoid catastrophe.

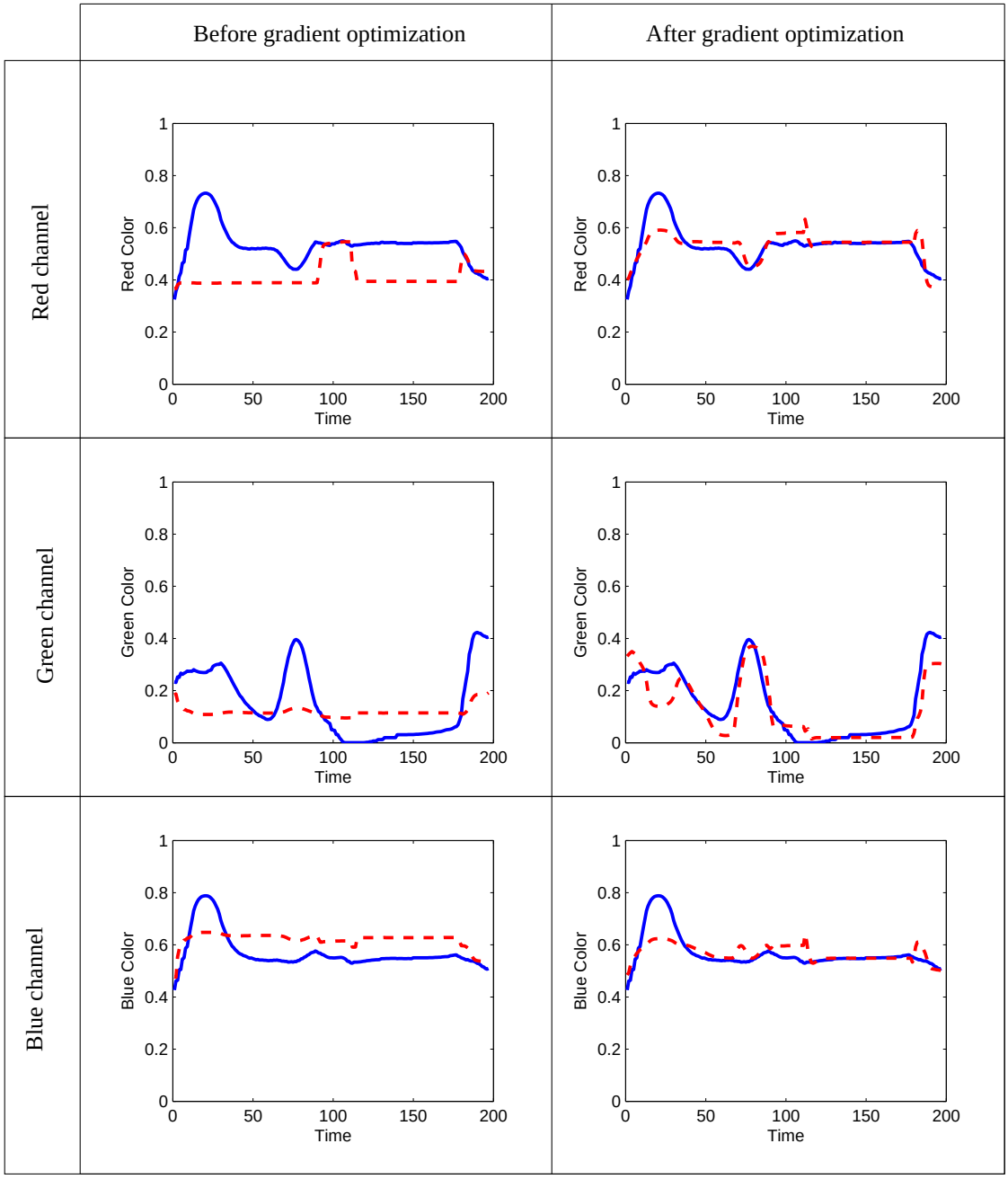


Figure 3.10: Qualitative results in the autonomous robot domain. Shown are the improvements resulting from the application of the gradient optimizer. Each graph compares one-step predictions (shown in red dashed lines) to true values (shown in blue solid lines). The three rows represent the RGB channels of the observation. The left column shows predictions using the best model obtained with the randomly chosen tests. The right column shows predictions made with the best model obtained after application of the gradient optimizer. The predictions in the right column match reality much more closely (see, in particular, the green channel). All figures use the same number of tests.

Nystrom approximations. The complexity of the gradient computations is quadratic in the number of data points. Appendix B discusses how we can combine Nystrom approximations with our information measure to obtain and computationally tractable gradient estimates.

Stochastic gradient descent. We used the chain rule to write the information gradient (Eq. 3.2) in terms of per-sample gradients. Instead of summing over all i, however, we can subsample, or perhaps collect samples of the gradient on-line. This leads to a stochastic version of the descent procedure. Although this is a viable method (especially for on-line use), experiments showed that the Nystrom method was more accurate and computationally cheaper. We believe this is because the Nystrom approximation uses some information from all of the samples in an approximate way, whereas the stochastic gradient descent uses all of the information of only some of the samples.

3.7 Conclusions and Future Work

We have extended PSRs to the continuous case with two core ideas: we have replaced the system dynamics matrix with the system dynamics distributions, and we have replaced the idea of using rank analysis to find sufficient statistics with ideas from information theory. We have argued that mutual information can help quantify the sufficiency of a candidate state representation; because information can be optimized, the representation can be improved.

We have also made several contributions on the algorithmic side, where there is a nice synergy between the elements: we started by using kernel density estimation to estimate the system dynamics distributions. Not only is it a nonparametric estimator, but it leads to closed-form expressions for mutual information. In addition, we can compute gradients of information with respect to test parameters in closed-form, which allows us to help solve the discovery problem. Both measuring information and computing gradients can be approximated efficiently; the resulting algorithms can handle tens or hundreds of thousands of data points. Empirically, our ideas seem viable. The model makes reasonable one-step predictions, and there appears to be a correlation between mutual information and MSE which our optimization procedure exploits; experimentally, it reduces both the MSE of one-step predictions and the variance of the MSE.

Estimating the system dynamics distributions is one of the central problems in the Continuous PSR model. For the rest of thesis, we will no longer represent state as a vector of densities. Instead, subsequent chapters will focus instead on the estimation of these distributions, and will go one step further: our next models will select parametric forms of the system dynamics distributions, and will use the parameters of those distributions as state.

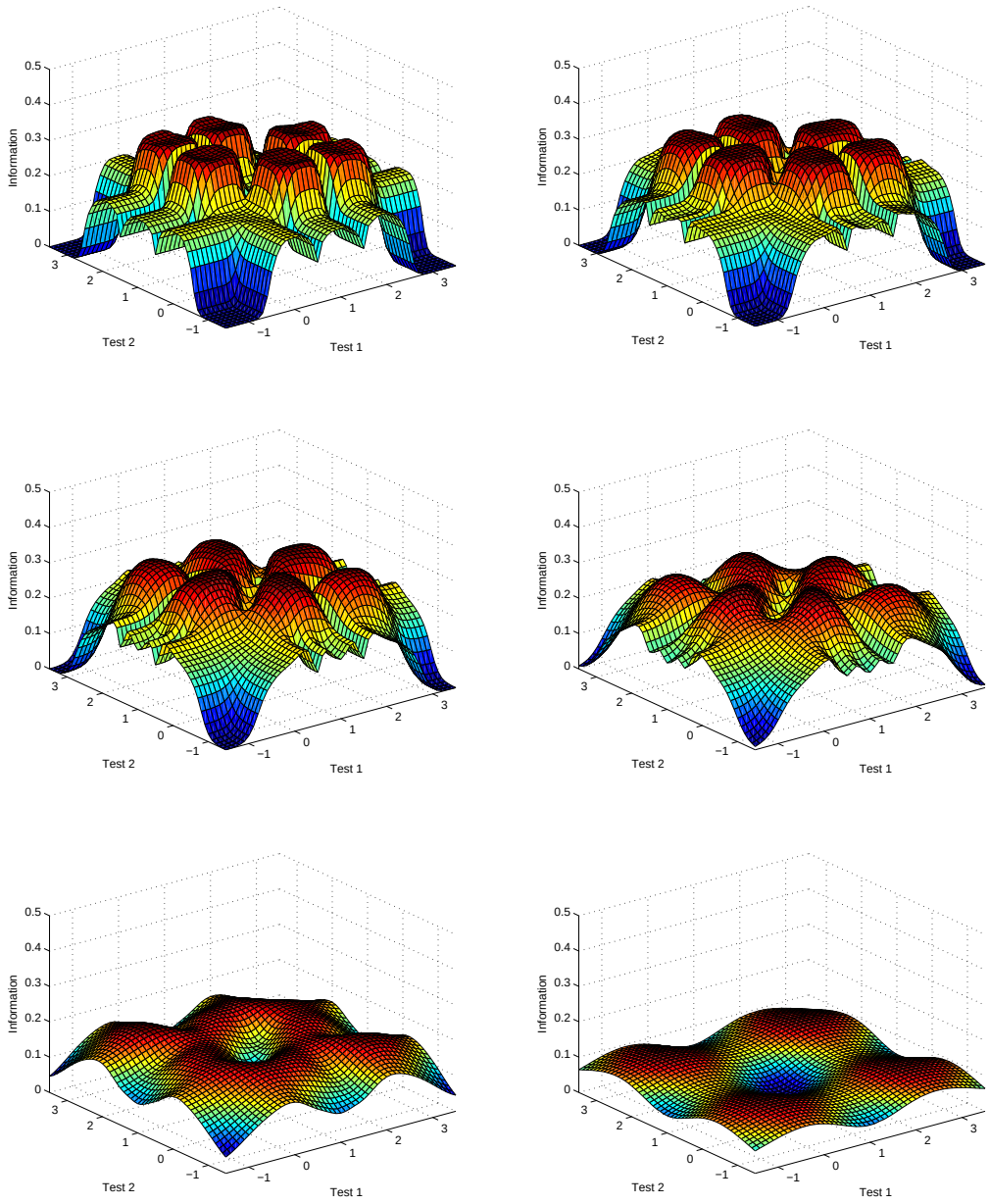


Figure 3.11: Smoothing information landscapes for homotopy optimization. Shown are information landscape as a function of the smoothing parameter λ (increasing from top to bottom, left to right).

Chapter 4

The Predictive Linear-Gaussian Model

The previous chapter introduced the system dynamics distributions, which formed an important part of learning a good model. In this chapter, we introduce the Predictive Linear-Gaussian (or PLG) model. The PLG is the predictively defined equivalent of a linear dynamical system, and also has close connections to the system dynamics distributions: the PLG selects parametric forms for the system dynamics distributions, and represents state as the parameters of those distributions. The rest of this thesis builds upon the PLG model in many ways, so we explain it in detail here. The PLG was introduced by Rudary et al. (2005).

PSRs (both continuous and discrete) represent state as statistics about the future. The original PSR models and the Continuous PSR model of Chapter 3 used the prediction of tests as the statistics of interest. Here, we introduce the more general notion of using parameters that model the distribution of length n futures as the statistics of interest. To clarify this, consider an agent interacting with the system. It observes a history h_t of observations $o_1, \cdots, o_t$. Given any history, there is some distribution over the next n observations:

$$p(O_{t+1}...O_{t+n}|h_t) \equiv p(F^n|h_t)$$

(where O_{t+i} is the random variable representing an observation i steps in the future, and F^n is a mnemonic for *future*). This is one of the system dynamics distributions defined in Section 3.1. We emphasize that this distribution directly models observable quantities in the system.

Our central assumption is that we can select a parametric form for $p(F^n|h_t)$, and that its parameters – which are obviously sufficient to predict the short-term future – are also sufficient to predict the infinite future, and therefore constitute state. As the agent interacts with the system, $p(F^n|h_t)$ changes because h_t changes; therefore the parameters and hence state change. As an example of this, the Predictive Gaussian models discussed in this chapter assume that $p(F^n|h_t)$ is a multivariate Gaussian; state therefore becomes its mean and covariance. In the case of the PLG model nothing is lost by defining state in terms of observable quantities: Rudary et al. (2005) showed the PLG is formally equivalent to the latent-variable approach in linear dynamical systems. In fact, there are some advantages to defining state in observable quantities: for example, because the parameters are grounded, statistically consistent parameter estimators become available for PLGs.

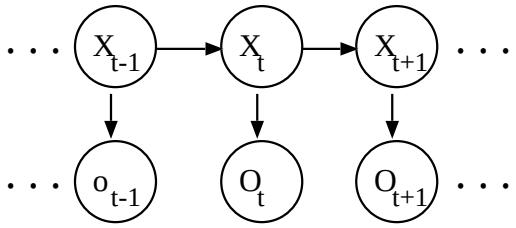


Figure 4.1: A standard graphical model of state-space systems. Latent states generate observations, and determine successor states.

Selecting the form of $p(F^n|h_t)$ and estimating its parameters to capture state is only half of the problem. We must also model the dynamical component, which describes the way that the parameters vary over time (that is, how the parameters of $p(F^n|h_t)$ and $p(F^n|h_{t+1})$ are related). In this chapter, we describe a method called "extend-and-condition," which is a generalization of many state update mechanisms in PSRs, and which will be used for all of the algorithms throughout the thesis.

In this chapter, we will review the important concepts associated with linear dynamical systems, Kalman filters, and the PLG. This will prepare us to discuss the nonlinear versions in Chapter 5 and Chapter 6, which are my main contributions to this line of work.

4.1 Linear Dynamical Systems

In this section, we review a popular model of dynamical systems with continuous observations called a *linear dynamical system* (LDS). In the PSR model of dynamical systems, state was captured as a set of predictions about the future. A more traditional approach is to introduce a set of internal state variables which can be used to capture state. These state variables are not typically observed directly, and so something about them must be inferred from actual observations. In the LDS model, we will infer a distribution over possible latent states that the system could be in, and it is the parameters of this distribution that are the sufficient statistic for history.

A discrete time LDS is defined by a state update equation

$$x_{t+1} = Ax_t + \eta$$

where $x_t \in \mathbb{R}^n$ is the state at time t, $A \in \mathbb{R}^{n \times n}$ is a transition matrix and $\eta \sim \mathcal{N}(0, Q)$ is mean-zero Gaussian noise (where $Q \in \Re^{n \times n}$). These models are very well understood, and can model a surprising variety of phenomena. Given a state x_t, the distribution of future states can be easily computed and will be a Gaussian variable.

Often, we are not be able to observe the state directly. Many LDSs define a companion observation process, in which observations are linear functions of the latent state:

$$o_t = Hx_t + \mathcal{N}(0, R),$$

where $H \in \mathbb{R}^{d \times n}$ and $R \in \mathbb{R}$. There are generally no restrictions on H; in particular, it may collapse an n-dimensional state into a lower-dimensional (or even scalar) observation.

Figure 4.1 shows a graphical model which is defined for such state space systems. The state variables X_t are unobserved, and are represented by capital letters. The conditional independence assertion made by this graph is that the distribution of the future is conditionally independent of the past, given the current state.

In the case of such a partially observable process, several problems arise. The *filtering* problem is posed as follows: given a state estimate x_{t-1} (which can be a Gaussian random variable), and an observation o_t, what is the optimal estimate of the state x_t? This question is fully answered by the Kalman filter, which has been shown to satisfy several different optimality criteria. The Kalman filter also permits easy computation of the distribution of future states and observations (the *prediction* problem). The *smoothing* problem is defined as estimating a sequence of states simultaneously, given a sequence of observations.

In some cases, the parameters of the LDS are not known, and must be estimated from data. There are two principal methods for accomplishing this. Expectation-maximization (EM) algorithms (see, for example Ghahramani and Hinton, 1996) guess at parameters and improve an estimated state sequence, and then hold the estimated state sequence constant and improve the parameters. This method hill-climbs in the space of likelihood, but because of its iterative nature, it suffers from several problems: it can get stuck in local maxima or minima, and it is somewhat slow. The other class of algorithms are subspace identification algorithms (van Overschee and Moor, 1996), where an SVD is performed directly on a block Hankel matrix to determine the state sequence and the H matrix, and then the A matrix is determined through regression. The method is non-iterative and numerically robust, and has proven to be a popular alternative to EM approaches.

4.2 The Kalman Filter

Throughout this thesis, we will reference the Kalman filter because of its close connections to the Predictive Linear-Gaussian model and the Exponential Family PSR. In this section, we review the basic definitions of the Kalman filter and provide pointers to related work.

The Kalman Filter (Kalman, 1960; Kalman and Bucy, 1961) was introduced by Rudolph Kalman in 1960 as an elegant solution to the filtering problem, in which the state of the system can be optimally estimated in an efficient, recursive way. By *optimal*, we mean that the state estimates generated by the Kalman filter are both unbiased and have minimum mean-squared error (although the Kalman

filter is optimal by other standards as well). By *recursive*, we mean that the state at time t can be computing using nothing but knowledge of the state at time $t-1$ and the observation o_t.

State representation: State in a Kalman Filter is defined as the parameters of a Gaussian distribution over the latent variable X. At time $t-1$, the state will be the mean μ_{t-1} and covariance Σ_{t-1} of a Gaussian over X_{t-1}:

$$p(X_{t-1}|h_{t-1}) = \mathcal{N}(\mu_{t-1}, \Sigma_{t-1}).$$

Dynamics: To model the dynamics of the system, our goal is to incorporate a new observation $O_t = o_t$ and recursively update the statistics we use as state, to compute the parameters of the distribution $p(X_t|h_{t-1}o_t) = p(X_t|h_t)$. There are many ways to derive the equations for the Kalman filter. Here, we adopt a simple one in which we construct the joint distribution of X_t and O_t (this is known as the *prediction phase*), and then condition the resulting multivariate Gaussian on the actual observation O_t to obtain an improved estimate of X_t (the *update phase*):

$$\begin{pmatrix} X_t \\ O_t \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} A\mu_{t-1} \\ HA\mu_{t-1} \end{pmatrix}, \begin{pmatrix} \Sigma^+_{t-1} & H\Sigma^+_{t-1} \\ \Sigma^+_{t-1}H^\top & H\Sigma^+_{t-1}H^\top + R \end{pmatrix}\right]$$

where

$$\Sigma^+_{t-1} = A\Sigma_{t-1}A^\top + Q.$$

We now condition on $O_t = o_t$:

$$\begin{aligned} K_t &= (H\Sigma^+_{t-1})(H\Sigma^+_{t-1}H^\top + R)^{-1} \\ \mu_t &= A\mu_{t-1} + K_t(o_t - HA\mu_{t-1}) \\ \Sigma_t &= (I - K_tH)\Sigma^+_{t-1}(I - K_tH)^\top + K_tRK_t^\top \end{aligned}$$

The simplicity of the Kalman filter update equations belie the utility and power of the model. The Kalman filter is almost ubiquitous in control theory and a variety of engineering applications. There are also numerous variants on it: the Extended Kalman Filter (EKF) (Zarchan and Musoff, 2005) generalizes the ideas to nonlinear dynamical systems by linearizing a nonlinear transition function around the current state, and is the defacto standard for nonlinear filtering. The Unscented Kalman Filter (Julier and Uhlmann, 1996) improves upon the EKF by using an unscented transformation instead of a linearization operator (we will have more to say about this in the chapters on the KPLG and MPLG). The Information Kalman Filter (Maybeck, 1979) represents the Gaussians used in the state representation with their natural parameterization instead of the mean parameterization, and has found application in distributed sensor networks (we discuss this more thoroughly in the chapter on Information PLGs). There are also square-root filters (Verhaegen and Dooren, 1986; Kaminski et al., 1971) designed to improve numerical stability, the continuous-time Kalman-Bucy filter and a practically infinite number of variations created by combining different ideas (for example, the square-root unscented Kalman filter of van der Merwe and Wan, 2001). Recently, the Kalman filter

has been unified with several other linear Gaussian models (Roweis and Ghahramani, 1999).

4.3 Predictive Gaussian Systems

The PLG is a predictively defined equivalent to an LDS. In the next two chapters, we will introduce two additional models (the KPLG and MPLG) which generalize the PLG to the case of nonlinear dynamics. Because the PLG, MPLG and KPLG all represent and update state in the same way, we call any one of these three models a *Predictive Gaussian System.* Here, we will discuss the entire family, and will then specialize back to the case of the PLG.

State: In Predictive Gaussian systems, we never refer to an unobservable or latent state x_t. Instead, we capture state as statistics about a random variable F^n, which is defined as a vector of random variables predicting *future* observations. Recall that

$$F^n = [O_{t+1} \cdots O_{t+n}]^\top,$$

as illustrated in Figure 4.2. The vector F^n is therefore $\in \mathbb{R}^{d \times n}$. We assume these variables are jointly Gaussian, with mean μ_t and covariance Σ_t:

$$F^n \sim \mathcal{N}(\mu_t, \Sigma_t).$$

Like the Kalman Filter, we will use the parameters μ_t and Σ_t as the state of the system. However, these parameters refer to the Gaussian distribution over F^n, *not* any sort of Gaussian distribution over a latent variable X_t.

Dynamics: The system dynamics are defined by a special equation:

$$O_{t+n+1} = f(F^n, \eta_{t+n+1}) \tag{4.1}$$

where $\eta_{t+n+1} \in \mathbb{R}^{d \times d}$ is a special noise term. The importance of modeling O_{t+n+1} as a function of F^n will be explained in the next section. In the PLG, O_{t+n+1} is a *linear* function of F^n, which allows it to model linear dynamical systems. In the MPLG and the KPLG, however, O_{t+n+1} is a *nonlinear* function of F^n, which allows them to model nonlinear dynamics.

The noise term is mean-zero with a fixed variance:

$$\eta_{t+n+1} \sim \mathcal{N}(0, \sigma_\eta^2),$$

but is allowed to covary with the next n observations in a way that is independent of history:

$$\mathrm{Cov}[F^n, \eta_{t+n+1}] = C_\eta.$$

Thus, the noise terms are identical, but not independent.

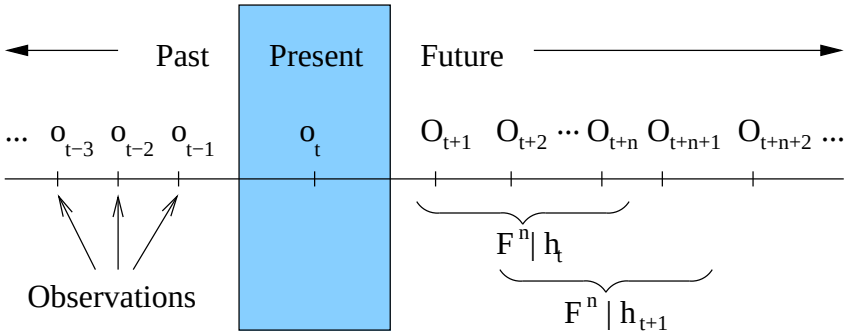


Figure 4.2: Timeline illustrating the random variables we use.

The representational power of Predictive Gaussian Systems comes from this noise term: the fact that it covaries with future observations gives it the infinite memory of the LDS—an observation can have an effect far in the future through the chain of influence created by the correlation in the noise terms. Later, we will see that the differences in this noise term are one of the primary differences between the MPLG and the KPLG.

4.4 Updating State: Extend and Condition

We will now discuss the general strategy of Predictive Gaussian Systems for updating state and modeling dynamical systems, as well as why O_{t+n+1} is modeled as a function of F^n. Modeling the system dynamics requires determining how to update the state of the system. The problem can be stated thus: given a state at time t, how can we incorporate an observation $O_{t+1} = o_{t+1}$ to compute our state at time $t+1$? The strategy is to *extend and condition*, as follows.

We begin with state extension. We assume that we have the state at time t, represented by μ_t and Σ_t. These statistics describe $F^n|h_t \sim \mathcal{N}(\mu_t, \Sigma_t)$, which is an $nd-$dimensional Gaussian describing the next n observations. We will extend this variable to include the variable O_{t+n+1} (ensuring that it is still jointly Gaussian), creating a temporary $(n+1)d$-dimensional Gaussian, which we denote $F^{n+1}|h_t$. We will use the extension function defined in Eq. 4.1:

$$O_{t+n+1} = f(F^n, \eta_{t+n+1}) \tag{4.2}$$

In order to extend F^n to include the variable O_{t+n+1}, we must compute three terms, which are $E_t = \mathrm{E}[O_{t+n+1}]$, $C_t = \mathrm{Cov}[O_{t+n+1}, F^n]$ and $V_t = \mathrm{Var}[O_{t+n+1}]$:

$$\begin{pmatrix} F^n \\ O_{t+n+1} \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \mu_t \\ E_t \end{pmatrix}, \begin{pmatrix} \Sigma_t & C_t \\ C_t^\top & V_t \end{pmatrix} \right].$$

We will then condition on the observation o_{t+1}, which will result in another $nd-$dimensional Gaussian random variable describing $[O_{t+2} \cdots O_{t+n+1}]^\top = F^n|h_{t+1}$. Conditioning is done with standard techniques on multivariate Gaussians, for which it is well-known that the resulting random

variable is Gaussian. This results in $\mathrm{E}[O_{t+2} \cdots O_{t+n+1}] = \mathrm{E}[F^n|h_{t+1}] = \mu_{t+1}$, along with $\mathrm{Cov}[F^n|h_{t+1}] = \Sigma_{t+1}$, which are precisely the statistics representing our new state. Figure 4.2 illustrates $F^n|h_t$ and $F^n|h_{t+1}$.

To condition on the observation at time t, we will repartition the mean and covariance matrices to simplify notation. This is not a mathematical operation. We are simply re-labeling entries as follows:

$$\begin{pmatrix} O_{t+1} \\ F^n \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \mu_{o_{t+1}} \\ \mu_{f^n} \end{pmatrix}, \begin{pmatrix} \Sigma_{o_{t+1}o_{t+1}} & \Sigma_{o_{t+1}f^n} \\ \Sigma_{f^n o_{t+1}} & \Sigma_{f^n f^n} \end{pmatrix}\right]$$

where $\mu_{o_{t+1}}$ is the first d entries of the vector $[\mu_t; E_t]$, and μ_{f+1} is the remaining entries. Similarly, $\Sigma_{o_{t+1}o_{t+1}}$ is an $d \times d$ matrix.

Conditioning is now done with standard formulae for multivariate Gaussians:

$$\mu_{t+1} = \mu_{f^n} + \Sigma_{f^n o_{t+1}}(\Sigma_{o_{t+1}o_{t+1}})^{-1}(o_{t+1} - \mu_{o_{t+1}}) \tag{4.3}$$

$$\Sigma_{t+1} = \Sigma_{f^n f^n} - \Sigma_{f^n o_{t+1}}(\Sigma_{o_{t+1}o_{t+1}})^{-1}\Sigma_{o_{t+1}f^n}. \tag{4.4}$$

Computing E_t, C_t and V_t in closed form for an arbitrary extension function f (see Eq. 4.1) is impossible, which motivates two different possibilities: the first is presented in the next section, which is to select a linear f. This makes the computations analytically tractable by virtue of the statistical properties of linear operators. The second option is to adopt a general approximation. Section 5.1.1 will present an approximation which can be used for any f, and will form the backbone of performing inference in both the KPLG and MPLG.

4.5 Dynamical Model of the PLG

We have now completed the development of the general state update mechanism for any Predictive Gaussian system. We will now briefly show the specific modeling choices the PLG makes. Here, we restrict ourselves to scalar observations, for two reasons: first, to make precise statements about the representational capacity of the PLG in the next section, and second, so that our exposition matches the historical development of the PLG. We emphasize that the restriction to scalar observations does not restrict the dimensionality of the underlying state space. The next section presents the multivariate generalization.

The PLG uses a linear f to model O_{t+n+1}:

$$O_{t+n+1} = f(F^n, \eta_{t+n+1}) = g^\top F^n + \eta_{t+n+1}$$

where $g \in \mathbb{R}^{n,1}$ is the linear trend of the system. The linearity in f allows simple closed-form expressions of the state update equations, because the needed terms can be computed easily:

$$E_t \quad = \quad \mathrm{E}[g^\top F^n + \eta_{t+n+1}] = g^\top \mu_t$$

$$
\begin{aligned}
C_t &= \mathrm{E}[O_{t+n+1}^\top F^n] - \mathrm{E}[O_{t+n+1}^\top]\mathrm{E}[F^n] = \Sigma_t g + C_\eta \\
V_t &= \mathrm{E}[O_{t+n+1}^\top O_{t+n+1}] - \mathrm{E}[O_{t+n+1}^\top]\mathrm{E}[O_{t+n+1}] = g^\top \Sigma_t g + 2g^\top C_\eta + \sigma_\eta^2
\end{aligned}
$$

These three equations, combined with the state update equations (Eq. 4.3 and Eq. 4.4) constitute the complete PLG model.

Rolling together the construction of the temporary Gaussian and the conditioning yields the complete state update:

$$\mu_{t+1} = \mu_{t+1}^- + K_t(o_{t+1} - e_1^\top \mu_t) \tag{4.5}$$

$$\Sigma_{t+1} = (I - K_t e_1^\top)\Sigma_{t+1}^- \tag{4.6}$$

where

$$
\begin{aligned}
\Sigma_{t+1}^- &= I^- \Sigma_t I^{-T} + I^- C_t + C_t^\top I^{-T} + e_n e_n^\top V_t, \\
K_t &= \Sigma_{t+1}^- e_1 (e_1^\top \Sigma_t e_1)^{-1}, \\
\mu_{t+1}^- &= I^- \mu_t + e_n E_t, \\
I^- &= \begin{pmatrix} \mathbf{0} & I_{n-1} \\ \hline & \mathbf{0} \end{pmatrix},
\end{aligned}
$$

and e_i is the i-th column of the identity matrix. These are the original update equations as found in (Rudary et al., 2005). Note that these equations have a strong resemblance to those used by the Kalman filter.

4.5.1 Properties of the PLG

The PLG has several advantages when compared to traditional state-space models. First, the entire model is defined strictly in terms of statistics about future observable quantities. This means that parameters of the model have definite meaning with respect to the observed data, which leads to statistically consistent parameter estimation procedures: estimated parameters will asymptotically converge to their *true* value, which is a stronger guarantee than those which accompany, for example, EM algorithms used to learn state-space model parameters (Ghahramani and Hinton, 1996, which we have empirically observed to be subject to local minima). Second, the PLG estimation procedure works particularly well as the dimension of the system increases.

Third, the PLG model strictly subsumes two popular linear dynamical system models: the celebrated Kalman filter (Kalman, 1960), and autoregressive time-series (or ARMA) models (Pandit and Wu, 1983). The relationship between the PLG and standard LDSs is roughly analogous to the relationship between PSRs and POMDPs: every LDS has an equivalent PLG, which can be proved by a constructive algorithm translating parameters; the resulting PLG is just as compact as an equivalent LDS, requiring only an n-dimensional Gaussian to model an n-dimensional LDS; and the resulting PLG actually uses fewer parameters than the equivalent LDS.

Finally, we have also alluded to learning the parameters of a PLG: Rudary et al. (2005) showed that a consistent parameter estimation algorithm exists, which consists of straightforward regressions and sample statistics. This is a favorable consequence of the grounded nature of the state representation.

As presented here, there are two restrictions on what the PLG can model: the system must 1) have linear dynamics, and 2) be uncontrolled. The original PLG model for uncontrolled systems with scalar observations was introduced by Rudary et al. (2005). Rudary and Singh (2006) have subsequently extended the PLG presented here to allow for control actions, multi-dimensional observations and have done some work on planning.

4.5.2 Extension to Vector-valued Observations

The math for the PLG is not dependent on the fact that the observations o_t are scalars. In fact, the same techniques used to derive the scalar PLG can be easily extended to the case of $d-$dimensional vector-valued observations. Here, we present the "naive" multivariate extension, and discuss why it is naive below.

We define the extension to be

$$O_{t+n+1} = F^n G + \eta_{t+n+1}$$

where $G \in \mathbb{R}^{n \times d}$ and $\eta_{t+n+1} \in \mathbb{R}^{d \times d}$, and let

$$\text{Cov}[F^n, \eta_{t+n+1}] = C_\eta$$

where $C_\eta \in \mathbb{R}^{nd \times d}$. This easily results in the following closed-form expressions for the extension, which are simply the multivariate generalizations of the scalar equations in the previous section:

$$\begin{aligned} E_t &= G\mu_t \\ C_t &= \Sigma_t G^\top + C_\eta^\top \\ V_t &= G^\top \Sigma_t G + G C_\eta^\top + C_\eta G^\top + \sigma_\eta^2. \end{aligned}$$

While extending the PLG from the case of scalar observations to multivariate observations in this way seems straightforward, there are a few technical subtleties. Specifically, Rudary et al. (2005) proved that every LDS with scalar-valued observations has an equivalent PLG with scalar-valued observations. Furthermore, the resulting PLG is just as compact, in the sense that a window of n future observations is all that is needed to model an n-dimensional LDS. This means that both the LDS and PLG will track an n-dimensional mean and an $n \times n$-dimensional covariance matrix as state. Additionally, the number of parameters in the PLG is actually fewer than the number in the corresponding LDS.

Further work by Rudary (unpublished) has extended the equivalences to the case of vector-valued

observations, but with a small twist. While every n-dimensional LDS with vector-valued observations can be modeled by the n dimensional PLG presented above, the reduction is not necessarily minimal in terms of the size of the resulting state or the number of parameters.

To see this, consider the example of an LDS with an $n > 2$ dimensional state space, and a two dimensional observation space. The LDS would have a state defined as an $n-$dimensional mean and an $n \times n$-dimensional covariance matrix. However, the naive PLG would have a state that is a $2n$-dimensional mean and an $2n \times 2n$-dimensional covariance matrix – a two-fold increase in the dimension of the state space and a four-fold increase in the number of parameters. Rudary has shown that the PLG does not need to track all $2n$ of these variables, and that it can in fact select a subset of only n of them. We defer to his work for the proof of this, as well as proof of the equivalence to vector-valued LDSs.

4.6 Conclusions

We have described the Predictive Linear-Gaussian model, which is a good example of the advantages of models with predictively defined representations of state. It loses nothing by defining state predictively, and actually gains something: the PLG is formally equivalent to the Kalman Filter, and has a statistically consistent parameter estimation algorithm. While the PLG itself will form the foundation for the models we will present in the next two chapters, the key ideas introduced here will play important roles throughout the thesis: first, the idea of capturing state as the parameters of the system dynamics distributions, and second, the idea of modeling dynamics with an extend-and-condition algorithm.

As mentioned, there are two drawbacks to the PLG: it is only capable of modeling uncontrolled, linear dynamical systems, and it is limited to the case of scalar observations. Rudary and Singh (2006) have done work on extending the PLG to the controlled, vector-valued case, but the resulting model is still limited to linear dynamics. The next two chapters attack the alternate direction: we extend to the case of nonlinear dynamics.

Algorithm PLG-UPDATE

Input: Current state, represented by μ_t and Σ_t and an observation o_t.

Given: dynamical parameters G, C_η, and σ_η^2.

Compute: (construct extended distribution)

- $E_t = G\mu_t$
- $C_t = \Sigma_t G^\top + C_\eta^\top$
- $V_t = G^\top \Sigma_t G + G C_\eta^\top + C_\eta G^\top + \sigma_\eta^2$

Repartition:

$$\mathcal{N}\left[\begin{pmatrix} \mu_t \\ E_t \end{pmatrix}, \begin{pmatrix} \Sigma_t & C_t \\ C_t^\top & V_t \end{pmatrix}\right] = \left[\begin{pmatrix} \mu_{o_{t+1}} \\ \mu_{f^n} \end{pmatrix}, \begin{pmatrix} \Sigma_{o_{t+1}o_{t+1}} & \Sigma_{o_{t+1}f^n} \\ \Sigma_{f^n o_{t+1}} & \Sigma_{f^n f^n} \end{pmatrix}\right]$$

Compute: (condition on observation o_{t+1})

- $\mu_{t+1} = \mu_{f^n} + \Sigma_{f^n o_{t+1}}(\Sigma_{o_{t+1}o_{t+1}})^{-1}(o_{t+1} - \mu_{o_{t+1}})$
- $\Sigma_{t+1} = \Sigma_{f^n f^n} - \Sigma_{f^n o_{t+1}}(\Sigma_{o_{t+1}o_{t+1}})^{-1}\Sigma_{o_{t+1}f^n}$

Return: μ_{t+1}, Σ_{t+1} .

Figure 4.3: The state update equations for the PLG.

Chapter 5

The Kernel PLG

The PLG is only capable of capturing linear dynamical systems. In this chapter we extend the PLG to model nonlinear dynamical systems by using kernel methods. We name the result the "Kernel Predictive Linear-Gaussian" model, or KPLG. We first present the general model, analyzing in depth the special case of the Gaussian kernel; with a Gaussian kernel, the model admits closed form solutions to the state update equations due to conjugacy between the dynamics and the state representation. For general kernels, closed-form solutions are not possible, so we explore an efficient sigma-point approximation. We show how all of the model parameters can be learned directly from data, either off-line or on-line (with the Kernel Recursive Least-Squares algorithm). We empirically compare the model and its approximation to the original PLG and discuss their relative advantages. Portions of this chapter were published in Wingate and Singh (2006a).

5.1 The Kernel PLG Model

We extend the PLG model to handle nonlinear dynamics by allowing O_{t+n+1} to be a nonlinear function of F^n, which we accomplish by invoking the kernel trick. As discussed in Section 4.4, all that is needed to maintain state is the statistics of the extended Gaussian $[F^n, O_{t+n+1}]$, which requires the expectation $E_t = \mathrm{E}[O_{t+n+1}]$, covariance $C_t = \mathrm{Cov}[F^n, O_{t+n+1}]$ and variance $V_t = \mathrm{Var}[O_{t+n+1}]$. Computing these three quantities, combined with the extension and conditioning equations (4.3) and (4.4), constitutes the complete model.

The KPLG defines the state extension as

$$O_{t+n+1} = \sum_{j=1}^{J} \alpha_j K(\xi_j, F^n) + \eta_{t+n+1}, \tag{5.1}$$

where $K()$ is our kernel. The $\xi_j \in \mathbb{R}^n$ are points that could come from a number of sources: they may come from training data, be derived analytically, or be randomly generated. These will be discussed later.

This is the most obvious way to kernelize the original PLG algorithm, because we have employed the standard technique of rewriting the linear trend g as a weighted combination of data points (this

is justified by the Representer Theorem of Kimeldorf and Wahba, 1971):

$$\begin{aligned} O_{t+n+1} &= g^\top F^n + \eta_{t+n+1} \\ &= (\sum_j \alpha_j \xi_j)^\top F^n + \eta_{t+n+1} \\ &= \sum_j \alpha_j (\xi_j^\top F^n) + \eta_{t+n+1} \\ &= \sum_j \alpha_j K(\xi_j, F^n) + \eta_{t+n+1} \end{aligned}$$

Since this is a linear basis function model (with the kernels $K(\xi_j, \cdot)$ acting as the basis functions), we will refer to the ξ_j's as basis function centers. The model strictly generalizes the PLG, since using the linear kernel recovers it. The variable η_{t+n+1} has the same properties as in the PLG.

With a Gaussian kernel, we can analytically derive expressions for E_t, C_t and V_t. Appendix C contains the lemmas and identities needed for their derivation, and a summary of what the terms mean:

$$\begin{aligned} E_t &= \sum_{j=1}^{J} \alpha_j K'_{tj} \\ C_t &= \sum_{j=1}^{J} \alpha_j K'_{tj} (\mu'_{tj} - \mu_t)^\top + C^\top \\ V_t &= \sum_{i=1}^{J} \sum_{j=1}^{J} K^\dagger_{tij} \alpha_i \alpha_j - E_t^2 + \sigma_\eta^2 + 2(\sum_{j=1}^{J} \alpha_j K'_{tj} (\mu'_{tj} - \mu_t)^\top) \Sigma_t^{-1} C. \end{aligned} \tag{5.2}$$

The parameters of this model are therefore the ξ_j's, the α_j's, C, and σ_η^2. In the case of a Gaussian kernel, we allow an additional parameter ϕ_j (which is the covariance matrix of the Gaussian) and write the kernel as $K(\xi_j, F^n; \phi_j)$. We use a fully normalized Gaussian for analytical purposes.

5.1.1 A Sigma-Point Approximation

With Gaussian kernels, the KPLG model is analytically tractable. While this is appealing, there are some computational liabilities. In particular, computing V_t is a $O(J^2)$ operation (where J is the number of basis functions; see the double summation of Eq. 5.2), which is prohibitively complex, especially since J typically scales exponentially with the dimension n. This motivates some sort of fast approximation. We would also like the approximation to relax the restriction to Gaussians, and free us to use arbitrary kernels. The following method accomplishes both goals (although exploring arbitrary kernels is left for future research).

Sigma-point approximations, or "unscented transformations" (Julier and Uhlmann, 1996), are a general method of propagating an arbitrary distribution through a nonlinear function. The method

is conceptually simple, and should be thought of as a deterministic sampling approach. Suppose we are given a random variable $O = f(F, \eta)$ that is a nonlinear function of another random variable F and a mean-zero Gaussian noise term η. Instead of recording the distribution information of F in terms of a mean and covariance, we represent the same information with a small, carefully chosen number of *sigma points*. These points are selected so that they have the same mean and covariance as F (in fact, they are the minimal such set), but the advantage is that they can be propagated *directly* through the function $f()$. We then compute the posterior statistics of the propagated points to approximate O. This process is demonstrated in Figure 5.1.

There are many advantages to sigma-point approximations. First, they are a good match for our needs: we only want first and second-order moments of the posterior (which they are designed to provide), and their strongest optimality guarantees are when F is normally distributed (as it is in our case). They are provably accurate to at least a second order approximation of the dynamics for any distribution on F and any nonlinearity, and are accurate to third order for a Gaussian distribution on F and any nonlinearity, while fourth order terms can sometimes be corrected as well. They can flexibly incorporate noise and other constraints into $f()$. They are simple to implement because no analytical derivatives (such as Jacobians or Hessians) are required. They are also efficient: they require $2(n + 1)J$ kernel evaluations at each timestep, which is far smaller than the $O(J^2)$ matrix operations required by the KPLG.

Sigma-point approximations should not be confused with particle filters. While they are similar in spirit, there are several important differences. Particle filters typically allow a multi-modal distribution over states, while sigma-point approximations require a Gaussian; it is the Gaussian assumption which gives the sigma-point approximation its strong theoretical guarantees with a small number of points. Also, where particle filters use random sampling, sigma-point approximations use deterministic sampling.

The algorithm is shown in Figure 5.2. If we let $f()$ be the state extension defined by the KPLG model (Eq. 5.1), then the final terms computed may be used in place of the analytical values of E_t, V_t, and C_t.

5.1.2 Complexity and Generalization

The KPLG model has high complexity: computing E_t and C_t is $O(Jn^3)$, but computing V_t is $O(J^2n^3)$ (and can be numerically unstable). There are other ways to estimate these terms, besides the sigma-point approximations. Nearest-neighbor style methods, such as the Fast Gauss Transform (Yang et al., 2003), are one possibility, and would also allow $O(Jn^3)$ computations, although these methods only work well for small n.

In the case of Gaussian kernels, the model can suffer from generalization problems. Because the Gaussians have local receptive fields, the state extension equation (Eq. 5.1) will return something close to zero for all states outside the training region. As we will see in the next chapter, renormal-

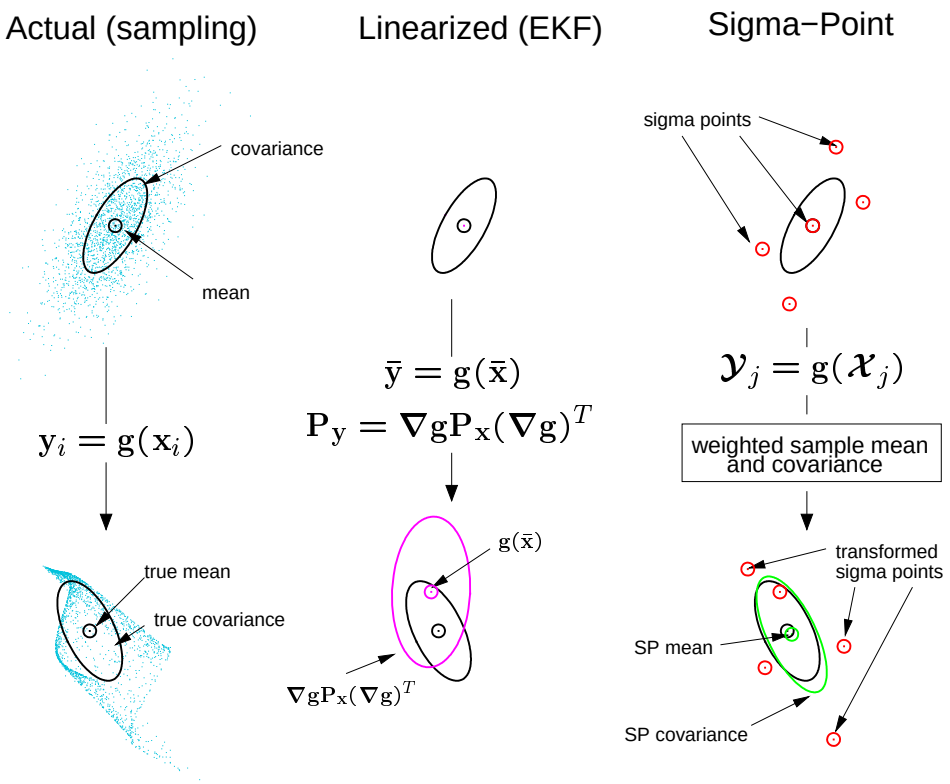


Figure 5.1: Sigma-point approximations. The left side shows how random samples of the variable X could be propagated through the nonlinear function g to estimate posterior statistics of Y. The middle panel shows an approach where g is linearized (this is the approach taken by the Extended Kalman Filter [EKF]). The mean of X is propagated through the original g, but the covariance is propagated through the linearized function. The right side shows the sigma-point approach, which deterministically samples points from X and propagates each through g to compute posterior statistics of Y. Figure courtesy of Eric Wan (used with permission).

Algorithm KPLG-SIGMA-POINT-APPROXIMATION

Input: μ_t, Σ_t

Given: f, C_η, σ_η^2

Compute:

- Construct a random variable relating predictions and noise:

$$P = \begin{pmatrix} F^n \\ \eta_{t+n+1} \end{pmatrix} \sim N\left[\begin{pmatrix} \mu_t \\ 0 \end{pmatrix}, \begin{pmatrix} \Sigma_t & C \\ C^\top & \sigma_\eta^2 \end{pmatrix}\right]$$

- Ensure that $\mathrm{Cov}[P]$ is symmetric positive definite.

- Construct a set of $2(n+1)$ sigma points:

$$[f_t^{n(2i-1)}, \eta_{t+n+1}^{(2i-1)}]^\top = \mathrm{E}[P] + (\sqrt{(n+1)\mathrm{Cov}[P]})_i$$

$$[f_t^{n(2i)}, \eta_{t+n+1}^{(2i)}]^\top = \mathrm{E}[P] - (\sqrt{(n+1)\mathrm{Cov}[P]})_i$$

- Propagate each point: $o_{t+n+1}^{(i)} = f(f_t^{n(i)}, \eta_{t+n+1}^{(i)})$

- Compute the empirical mean and covariance:

$$E_t = \frac{1}{2(n+1)} \sum_{i=1}^{2(n+1)} o_{t+n+1}^{(i)}$$

$$V_t = \frac{1}{2(n+1)} \sum_{i=1}^{2(n+1)} (o_{t+n+1}^{(i)} - \mathrm{E}[O_{t+n+1}])^2$$

$$C_t = \frac{1}{2(n+1)} \sum_{i=1}^{2(n+1)} (f_t^{n(i)} - \mu_t)(o_{t+n+1}^{(i)} - \mathrm{E}[O_{t+n+1}])^\top$$

Return: E_t, C_t, V_t

Figure 5.2: The sigma-point approximation algorithm.

ized kernels combined with linear models will improve this generalization.

5.1.3 Comparison to Nonlinear Autoregression

There is a significant difference between the KPLG model and an n-th order kernel autoregressive (KAR) model. The KAR model is

$$\mathrm{E}[O_{t+1}] = \sum_j \alpha_j K(\xi_j, [o_{t-n-1}, \cdots, o_t]),$$

which states that the next observation is predicted to be a nonlinear function of the *past* n observations $[o_{t-n-1}, \cdots, o_t]$. It has a similar functional form to our predictive model: the same kernels, basis function centers, and coefficients are used, and it can be trained using similar methods as the KPLG (see Section 5.2). However, their differences are as important as their similarities. The KAR model can only predict a point estimate for the future, whereas the KPLG predicts an entire distribution. KAR implicitly assumes that n past observations constitute state, which is effectively a k-th order Markov assumption, while the KPLG can summarize a potentially infinite amount of history into its predictions. These differences are what accounts for the empirical improvement of KPLG over KAR reported in Section 5.3.

5.2 Model Learning

Having defined the form of the model and the state update, we now address the question of learning a good model from data. The KPLG model requires several parameters: the dimension n of the system, the basis function centers ξ_j and weights α_j, as well as the noise statistics C and σ_η^2. In the case of a Gaussian kernel, the covariance matrix ϕ_j is also required. The next two sections discuss off-line and on-line methods of estimating these parameters, with emphasis on relating the procedures to general techniques.

5.2.1 The Off-Line Case

In the off-line case, the data for learning will be given as a set of trajectories from the system, with each trajectory consisting of at least $n+1$ sequential observations. We will slice these trajectories into training pairs $(f_t^{n(i)}, o_{t+n+1})$ where $f_t^{n(i)} \in \mathbb{R}^n$ is a vector of n successive observations (representing a noisy sample of some F^n), and $o_{t+n+1} \in \mathbb{R}$ is the $(n+1)$-th observation (a sample of the corresponding O_{t+n+1}, or the state extension). Each trajectory is sliced into all such pairs and collected into a set S (this is somewhat like using the suffix-history algorithm to generate samples from the corresponding system dynamics distributions; see Section 3.4). Figure 5.3 graphically illustrates the process.

Model Order Selection. We must first estimate the order of the model, which includes the system dimension n and the number of basis functions J. For our experiments, we use cross-validation to select parameters from a set of likely candidates. However, there is nothing unusual about our

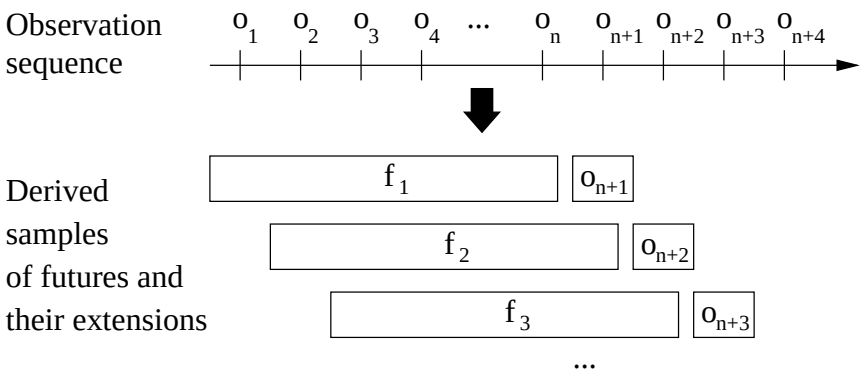


Figure 5.3: Extracting training pairs from a training trajectory.

model or estimation needs, meaning that many existing techniques are also suitable. These include growing and pruning methods, stacked generalization, regularized complexity criteria, or statistical tests such as Z tests (Bishop, 1995; Pandit and Wu, 1983).

Finding Basis Function Parameters. Next, we must determine the basis function centers ξ_j and covariance matrices ϕ_j. We tested three methods: random selection, dictionary-based selection (explained in Section 5.2.2), and expectation maximization. For random selection, we set each ξ_j to be a random training sample $f_t^{n(i)}$, we set $\phi_j = \sigma_\phi^2 I$, and we used cross-validation to select σ_ϕ^2. Expectation maximization (EM) is a well-known method for estimating mixture of Gaussian parameters. We will here summarize our experiments with EM by saying that it did not appear to offer any advantage over the other two methods, and since it was computationally more intensive, it was dropped. Again, many other methods are also suitable. These include nonlinear gradient methods (such as Gauss-Newton or Marquardt-Levenburg), re-estimation methods (such as expectation maximization), adaptive k-means clustering, stochastic sequential estimation, or cross-validation (generalized, leave one out, or k-fold) (Hastie et al., 2001; Bishop, 1995).

Estimating Coefficients. Given ξ_j and ϕ_j, finding the α_j's can be viewed as a simple kernel regression problem. It can be solved with a linear least squares approach, or more sophisticated methods such as support-vector regression (Shawe-Taylor and Cristianini, 2004). We chose regularized least-squares. We construct a regression matrix $A + \lambda I$, where $A_{ij} = K(\xi_j, f_t^{n(i)}; \phi_j)$ and λ is the regularization coefficient. Let O be a vector collecting all the o_{t+n+1}'s. Then, the optimal coefficients α are given by $\alpha = A^\dagger O$, where † denotes the pseudo-inverse (giving a minimum-norm solution to an underconstrained system, and a least-squares solution to an overconstrained system).

5.2.2 The On-Line Case: KRLS

The previous section discussed the selection of the basis function centers ξ_j and their weights α_j as two separate problems. However, both steps may be combined into a single step by using the Kernel Recursive Least-Squares (KRLS) algorithm of Engel et al. (2004). Finding the weights is a least-squares kernel regression problem, which KRLS is designed to solve, but it does so in a

recursive way: instead of presenting all of the training pairs simultaneously, they are presented one at a time, and the algorithm updates the resulting weights with complexity that is independent of the total number of pairs used (in our case, it is equivalently independent of time).

The idea of the KRLS algorithm is to automatically select *dictionary* points from the training samples which are presented on-line. These dictionary points are selected such that the features of other training samples can be expressed as an approximate linear combination of the features of the dictionary points (where "approximate" is a tunable threshold). The set of dictionary points approximately linearize the feature space, and can be thought of as points which are spread "evenly" throughout the feature space.

It is these dictionary points that we use as the basis function centers ξ_j, and the corresponding weights as the α_j's. This gives even coverage to the feature space, and can be controlled by only a single additional parameter. KRLS is an instance of the KAR model in Section 5.1.3, and that the on-line dictionary creation process is actually an on-line version of the Nystrom approximations discussed in Appendix B.

5.2.3 Learning Noise Parameters

Either the off-line or the on-line techniques provide basis function centers, covariances and weights, allowing us to now estimate the noise parameters. For these, we can use sample statistics. Assume we have a set S of training pairs $(f_t^{n(i)}, o_{t+n+1})$. In the off-line case, this may be the training set; in the on-line case, this set may be collected during training, or once the basis function parameters have been fixed. Let

$$\eta_i = o_{t+n+1} - \sum_j \alpha_j K(\xi_j, f_t^{n(i)}; \phi_j).$$

Then, the estimated noise term is

$$\widehat{\sigma_\eta^2} = \frac{1}{|S|-1} \sum_i (\eta_i)^2.$$

To estimate C, we run the algorithm on the training data (or run it online) with $C = 0$ and collect an estimate of μ_t at each t. We then compute

$$\mathrm{Cov}[F^n \eta_{t+n+1}] = \mathrm{E}[(F^n - \mu_t)(\eta_{t+n+1})],$$

which is simply

$$\widehat{C_k} = \frac{1}{|S|-1} \sum_i (f_t^{n(i)} - \mu_i)_k \eta_i.$$

Extending these estimators to be fully on-line is left as future work.

5.3 Experiments and Results

Our experiments were designed to assess the performance of the PLG, KPLG and KPLG-SP (the sigma-point approximation) algorithms across a variety of problems. For completeness, we also tested the KAR algorithm. We tested on five linear and nonlinear dynamical systems (the Rotation, Biped, Peanut, NB3, and Spring problems), where the underlying generative model was known. Since the models are limited to scalar observations, we also tested on three timeseries benchmarks (Santa Fe Laser, Mackey-Glass, and K.U. Leuven). The problems are described in Section 5.3.1.

We ran two types of experiments. The first type was a short-term prediction problem, in which the algorithms were run as explained in the text. This tested the algorithms' state update mechanisms and prediction performance. The second type was a far-horizon prediction test, where the algorithms predicted hundreds of steps into the future, without correcting state based on any observations. This tested modeling capacity and parameter estimation methods.

Parameters were selected by 10-fold cross-validation. Algorithms were judged on the mean-squared error (MSE) of their predictions. All data sets were normalized to be in $[0, 1]$. For the initial state, we set $\Sigma_0 = \widehat{\sigma_\eta^2} I + \sum_{i=1}^{n} (I^-)^i \widehat{C} + ((I^-)^i \widehat{C})^\top$ and μ_0 to be the last n values of the training sequence, and then rolled it forward n timesteps (the test data was structured to be a continuation of the last sequence of training data). All algorithms were tested on $n = 2, 3, 4, 5, 6$, $\sigma_\phi^2 = 0.1, 0.4, 0.8, 1.2$, $\lambda = 0.00001, 0.001, 0.01$, and $\nu = 0.0001, 0.001, 0.01$ (the dictionary threshold).

5.3.1 Problem Descriptions

All problems except Laser were trained on 2000 sequential observations and tested on a 200 observation continuation; the "Laser" series had 1000 training and 100 testing observations.

Rotation, Peanut, Biped, NB3: These are all two-dimensional dynamical systems. Rotation is a linear dynamical system consisting of a simple rotation matrix. Peanut is similar, except that points are rotated around a peanut shape. Biped is a non-linear dynamical system inspired by a foot striking the ground. The system has the same underlying dynamics as the linear rotation problem, except that there is a strict minimum value for the second coordinate x_2 of -0.5 – any time x_2 goes below -0.5, it is clipped to be -0.5, representing the discontinuity of the ground. The resulting system is piecewise linear. NB3 is like biped, except with more noise. For all four, observations and dynamics were noisy. NB3 had about an order of magnitude more noise than the other problems.

Spring: A two-dimensional system with a mass oscillating between damped springs. The springs had nonlinear forcing functions. Only the position of the mass was observed. This problem has deterministic dynamics and noise-free observations.

Mackey-Glass: The Mackey-Glass time series (Mackey and Glass, 1977) is a popular choice for time series benchmarks because it is deterministic but chaotic. It is generated from a delay differential equation that can display a wide variety of behaviors as function of the delay term τ. The

	Problem	Best (10-CV)	Best (5-CV)	Best (direct)
LDSs	Rotation	KPLG-SP	KPLG-SP	KPLG/KPLG-SP
	Biped	KPLG/KAR	KPLG-SP	KPLG/KPLG-SP/KAR
	Peanut	KAR/PLG	KPLG-SP	KPLG/KPLG-SP
	NB3	KAR/PLG	KAR/PLG	KPLG-SP/KAR/PLG
	Spring	KAR	KAR	KAR
Time Series	M.G.	KAR	KAR	KAR
	Leuven	KAR	KAR	KAR
	Laser	KPLG	KPLG-SP	KAR

Figure 5.4: KPLG empirical results on the short-term prediction problem. The best performing algorithm(s) for each problem is shown.

dynamics are given by

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x(t-\tau)^{10}} - bx(t).$$

Two common parameter settings are $\tau = 17$ and $\tau = 30$. With $a = 0.2, b = 0.1, \tau = 17$, the equations give rise to a chaotic, deterministic time series concentrated around a strange attractor of fractal dimension 2.1; the apparent chaos is due to the fact that the value of the series at any point may depend upon the entire history of values. We used $a = 0.2, b = 0.1$, and $\tau = 30$, which are standard settings.

K. U. Leuven: This data set comes from a time series prediction competition, held as part of the International Workshop on *Advanced Black-Box Techniques for Nonlinear Modeling*, K.U. Leuven Belgium, 1998 (Suykens and Vandewalle, 1998). The data set consists of 2000 data points. Contestants were asked to predict an additional 200 beyond the end of the set, and were then ranked based on mean squared error. A wide variety of techniques were used, including several based on dynamical system identification and neural-network style modeling. One of the primary advantages of this series is that it allows us to compare against a number of other advanced techniques, without having to implement them all. Results of the competition can be found in Weigend and Gershenfeld (1994).

Santa Fe Laser: Data from the Santa Fe timeseries competition, which was also used in the K.U. Leuven competition. The series was recorded from a laser in a chaotic state, whose pulsations more or less follow the theoretical Lorenz model of a two-level system.

5.3.2 Short-Term Prediction

For these experiments, we measured 1-step prediction MSEs. It is important to note that the measure of success is the difference between actual and predicted observations. This means that we are not attempting to estimate latent state, but we are allowed to *use* state to make our predictions. Basis function centers were selected with a dictionary, and α_j's were computed using regularized least-squares; this method was superior to selecting basis functions randomly or with EM.

The results are shown in Figure 5.4. Three columns are presented, the first of which shows the best algorithms on each problem when parameters are selected using cross-validation (all algorithms with an MSE within 5% of the lowest are considered equal). The results are mixed but encouraging. In particular, it seems that KPLG(-SP) generally performed well on the dynamical systems, where there really *is* an opportunity to leverage infinite memory via state. In contrast, KAR has performed well on the timeseries problems; in particular, it wins on the Mackey-Glass series, which really *is* an autoregressive model.

This trend is more pronounced when only the best 5 out of the 10 cross-validation runs are used to select parameters, as shown in the second column. The point of doing this is to tell a more complete story: the KPLG model is actually *capable* of doing a better job than the results for the 10-fold CV suggest, if only the correct parameters can be chosen. By using only the best 5 out of the 10 cross-validation runs, we have eliminated outliers in the cross-validation runs, which has given us better parameters. We see that KPLG-SP has won in four out of eight trials, and in the situations we expect it to.

The final column of Figure 5.4 shows best performers when tested *directly against the test set* (that is, without cross-validation). While it doesn't change the fundamental results, there are some noteworthy points: KAR does better on Laser, and KPLG is now competitive on the LDSs. These results should be taken with a grain of salt: there are enough parameters in the algorithms (and the test sequences are short enough) that they may be overfitting on the test data. However, the results show that all of the algorithms have the capacity to model the test data well. We also note that KAR still wins on the Spring problem. This is expected: Spring is deterministic with noiseless observations, so the uncertainty the KPLG(-SP) uses is unneeded.

Together, these results can be interpreted as preliminary evidence that each algorithm is winning when it is supposed to be, although it also appears that the test problems are not as discriminative as we would like. The results suggest three conclusions: first, that the nonlinear models are outperforming their linear counterparts; second, that the sigma-point approximation is competitive with the exact KPLG; and third, that our models are indeed capturing state, which results in an advantage over a simple autoregressive model, especially in noisy cases. Not reflected in these results is the fact that KAR seemed to give more consistent MSEs across parameter settings than the KPLG(-SP). The results also suggest that a better method than cross-validation is needed to select the model parameters.

5.3.3 Long-Term Prediction

For this set of experiments, each algorithm was asked to predict hundreds of timesteps into the future. This was done to assess the models' raw capacity, especially as compared to other methods. We trained KPLG(-SP) using the KRLS algorithm, incorporating the more sophisticated training method suggested by Engel et al. (2004). We set $\Sigma_t = 0$ for all t, making KPLG(-SP) and KAR

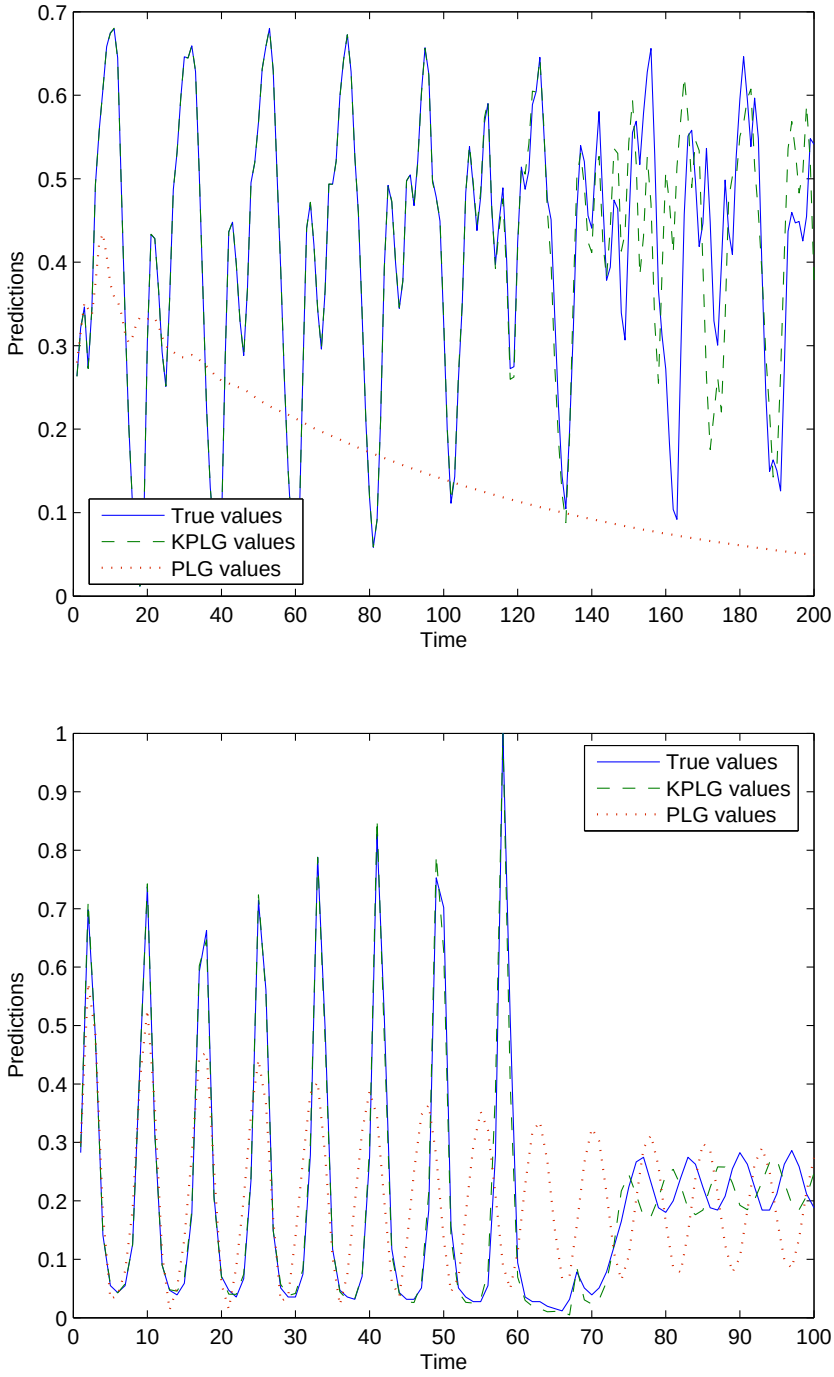


Figure 5.5: KPLG long-term prediction results. Top: the results of predicting the Mackey-Glass series. Bottom: the results of predicting the Santa Fe Laser series.

equivalent; we merely wanted to compare PLG and KPLG. Figure 5.5 shows results on Laser and Mackey-Glass, both of which demonstrate a clear advantage of KPLG over PLG. The Laser result almost exactly reproduces the result obtained by Engel; as noted by him, the MSE incurred here (0.00120; equivalent to their NMSE of 0.026) would have been just enough to place first in the Santa Fe competition. This suggests that the model is capable of competing successfully with other methods.

5.4 Related Approaches

Here, we briefly survey other nonlinear methods that are similar in spirit and application to ours, focusing on nonlinear extensions to the Kalman filter. The first is the Extended Kalman filter (EKF), which updates state by linearizing the system dynamics, and propagating information through this first-order approximation. Unfortunately, it requires that analytical derivatives of the dynamics be available, and cannot capture discontinuities in the dynamics. The Unscented Kalman Filter (Wan and van der Merwe, 2000) improves on the EKF with a sigma-point approximation. It is the closest competitor to our method, except that it posits latent state and provides no parameter estimation methods; our method is also simpler because our observation and transition models are combined. Rudary and Singh (2004) proposed a nonlinear PSR based on "e-tests," but it is restricted to domains with discrete observations. Local modeling methods (such as local linear regression) could also be used (Fan and Gijbels, 1996; Hastie et al., 2001), at the cost of retaining the training data as part of the model.

5.5 Conclusions and Future Work

In this chapter, we set out to extend the PLG to be able to model dynamical systems with continuous observations and nonlinear dynamics. More broadly, we have investigated the question of whether such a model can be learned, and if so, whether or not it is competitive with other models.

Based on our empirical results, the broadest conclusion is that both the idea and our specific model are viable. While more work remains to be done, the KPLG has successfully modeled the real-world and synthetic problems presented here – while learning its parameters directly from data – and appears to provide competitive results to other methods. One of the advantages of the model is the straightforward method of parameter estimation. Only standard regressions and sample statistics are required, which is a direct consequence of the predictive nature of the state. This also lead us easily to an on-line version of the algorithm with KRLS.

An important practical conclusion is the success of the sigma-point approximations, which have provided results close to those of the KPLG for a fraction of the computational effort. We originally picked the Gaussian form of the kernels for analytical tractability, but the success of the approximations suggests that this is unnecessary. In addition to accuracy and speed, they provide freedom: future applications of the KPLG can use other kernels in more flexible models.

We have not focused on raw empirical success, which leaves the door open for several obvious extensions. In particular, combining the strengths of KPLG and KAR into quasi-predictive models (which use history and predictions together) is an open and interesting avenue. It is also important to address the difficulties in parameter estimation and cross-validation, and to improve the algorithm's stability and generalization, but even with these problems, the algorithm is learning reasonable and competitive models.

Picking model parameters is still challenging. In the next section, we will improve on the KPLG by improving the model's generalization and stability with respect to parameter choices.

Chapter 6

Mixtures of PLGs

Chapter 5 introduced a kernelized version of the PLG which was capable of capturing nonlinear dynamics and which successfully modeled several timeseries problems. However, there are a few deficiencies in the model which we seek now to remedy: first, the model tended not to generalize well as a result of the local support of the Gaussian kernel. This is in stark contrast to the original PLG, which automatically generalizes well throughout the entire state space, simply due to its linearity. The other troubling factor of the KPLG is the fact that the parameters of the noise term η_{t+n+1} are constant throughout the state space. It is easy to imagine situations where this is not the desired property. Finally, we noted empirically that the performance of the KPLG was not very consistent across different parameter settings, even for small perturbations in the parameter values.

All three of these flaws can be simultaneously improved with a nonlinear mixture technique. This chapter contributes a probabilistic, generative model of dynamical systems, which we have named the "Mixture of PLGs" (or MPLG). Like the PLG and KPLG, the MPLG assumes that $p(F^n|h_t) \sim \mathcal{N}(\mu_t, \Sigma_t)$, and that state is the parameters of that Gaussian. Like the KPLG, the MPLG captures nonlinear dynamics by modeling O_{t+n+1} as a nonlinear function of F^n. However, there is a significant difference between the way that the KPLG and the MPLG accomplish this, with a nice interpretation of the relationship between the two: while the KPLG models linear dynamics in a nonlinear feature space, the MPLG models dynamics which are piecewise linear. In the MPLG model, interpolation between training points is nonlinear (as in the KPLG), but generalization is linear, and it is this linear generalization that will be the key to improved modeling accuracy and parameter stability. Portions of this chapter were published in Wingate and Singh (2006b).

We also develop a novel technique to perform inference in the model. Like the KPLG, the model is defined in terms of random variables, and like the KPLG, certain statistics of these variables must be computed to update state. Because the needed functions are nonlinear, exact analytical inference is generally impossible. This motivates some sort of approximation technique, so like the KPLG, we have chosen sigma-point approximations. However, to reduce the computational complexity of the method, and to improve the accuracy of the estimates, we develop a method we call "hybrid particle-analytical inference," which is a form of Rao-Blackwellisation. Standard sigma-point approximations sample from all of the random variables in a model simultaneously, but sampling only a subset of variables can lead to significant computational advantage: part of our

model is approximated with sigma-points, but *given* those sigma-points, exact analytical inference is possible on the rest of the model. This is an application of the smoothing properties of expectations, and is general enough to be applied in other contexts.

After introducing the model and our hybrid inference technique, we show how the model's mixture perspective leads to natural parameter estimators which are kernel-weighted versions of the original PLG estimators. We empirically compare the proposed model to two nonlinear alternatives, and conclude that our proposed model exhibits an advantage over all of them, and in particular, over n-th order autoregressive models.

During our initial exposition of the MPLG we will assume, like we did in the KPLG model, that every variable in our observation vector must be modeled. However, in Section 6.5 we will relax this assumption in the context of a traffic modeling problem. This allows the MPLG to use exogenous variables (that is, variables [like actions] which are given at every timestep, but which do not necessarily need to be modeled) to help define the piecewise linear regions. We will demonstrate that using this technique we can obtain improved performance over the PLG.

6.1 The MPLG: A Mixture of PLGs

We now present the MPLG, or Mixture of PLGs model. To see the intuitive justification for the MPLG model, consider the following scenario. Suppose that at time t, we have J PLGs, each with different parameters, and each specifying a different distribution over O_{t+n+1}. How should should we model O_{t+n+1}? A sensible approach is to combine the estimates of all J PLGs in some way, and ideally, we would combine them based on some estimate of the confidence that each PLG has in its prediction. Since this is a dynamical model, it might also make sense to allow that confidence to vary as a function of the state space, which would allow each PLG to become an expert in a certain region of the state space.

This is closely related to a mixture of experts model (Jacobs et al., 1991; Nowlan and Hinton, 1991). A natural way to simultaneously define confidences and mix predictions is with weighted sum, where the weights represent the confidence of each PLG:

$$O_{t+n+1} = \sum_{j=1}^{J} w(F^n)_j \left({g^j}^\top F^n + b^j + \eta^j_{t+n+1} \right). \tag{6.1}$$

Here, the $w(F^n)_j$'s are the mixing weights. It is important that they be a nonlinear function of F^n, because if they were linear they could simply be absorbed into the g^j's, resulting in a linear model. Here, $\mathrm{Var}[\eta^j_{t+n+1}] = (\sigma^2_\eta)^j$ and $\mathrm{Cov}[\eta^j_{t+n+1}, F^n] = C^j_\eta$.

In general, these weights may be positive or negative. However, if we additionally impose the restriction that they are positive and sum to one, the model takes on a new interpretation as a probabilistic mixture model: it becomes a generative model of the dynamics, because we can consider the

weights the distribution of a discrete multinomial variable. The generative procedure is what one would naturally expect: we first pick one of the J PLGs according to the distribution specified by the weights, and then generate a value for O_{t+n+1} by sampling from F^n, multiplying by the trend, adding the bias, and then corrupting with another sample from mean-zero Gaussian noise. In fact, positive weights that sum to one is exactly what we obtain if we derive the MPLG in a slightly more principled way, as we now explain.

6.1.1 A Distribution Over PLGs

Eq. (6.1) is the general form of the MPLG. We will now make specific choices about the function $w()$. We start by creating a new random variable $\mathbb{O}_t$ that describes a distribution over possible O_{t+n+1}'s; each O_{t+n+1} is itself a Gaussian random variable describing distributions over actual observations o_{t+n+1}. We then specify a joint distribution over $\mathbb{O}_t$ and F^n, and use conditional expectation and a density over possible models to arrive at the final mixture of PLGs.

Suppose we use a Parzen kernel estimator to represent the joint density of $\mathbb{O}_t$ and F^n; suppose further that we use Gaussian kernels. Such an estimator would take the form:

$$p(\mathbb{O}_t, F^n) = \frac{1}{J}\sum_{j=1}^{J}\frac{1}{c_j}K(\mathbb{O}_t, \xi_{\mathbb{O}_t}; \phi_j)\frac{1}{c_j}K(F^n, \xi_j; \phi_j) \tag{6.2}$$

where $1/c_j$ is a standard Gaussian normalizer and $K(x, y; \phi)$ is a Gaussian function with covariance matrix ϕ. The $\xi_j \in \mathbb{R}^n$ are points that could come from a number of sources: they may come from training data, be derived analytically, or be randomly generated. The $\xi_{\mathbb{O}_t}$ variables will disappear in the following derivation.

We can use this estimator to derive the MPLG as shown below (derivation adapted from Bishop, 1995; pg. 178). In the fourth line, we will use the Parzen estimator of the joint probabilities (several terms cancel); this resembles the well-known Nadaraya-Watson estimator. In the fifth line, we replace each O^j_{t+n+1} with a PLG that generates it, and in the final line, we summarize the kernel renormalization into a vector of weights $w(F^n)$. As required, these weights are a nonlinear function of F^n and sum to one:

$$\begin{aligned}
O_{t+n+1} &= \mathrm{E}[\mathbb{O}_t|F^n] \\
&= \int \mathbb{O}_t p(\mathbb{O}_t|F^n) d\mathbb{O}_t \\
&= \frac{\int \mathbb{O}_t p(\mathbb{O}_t, F^n) d\mathbb{O}_t}{\int p(\mathbb{O}_t, F^n) d\mathbb{O}_t} \\
&= \frac{\sum_{j=1}^{J} K(\xi_j, F^n; \phi_j) O^j_{t+n+1}}{\sum_{j=1}^{J} K(\xi_j, F^n; \phi_j)}
\end{aligned}$$

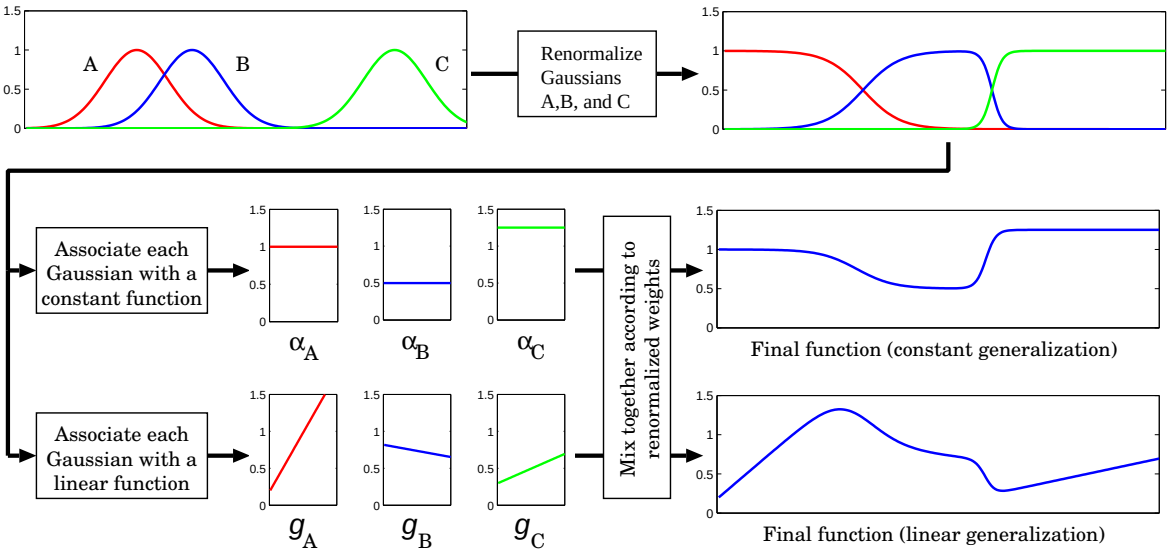


Figure 6.1: Flow chart of MPLG mixing. Shown are the effects of combining kernel renormalization with constant models (top) and linear models (bottom).

$$= \sum_{j=1}^{J} \frac{K(\xi_j, F^n; \phi_j)}{\sum_{k=1}^{J} K(\xi_k, F^n; \phi_k)} \left({g^j}^\top F^n + b^j + \eta^j_{t+n+1} \right)$$
$$= \sum_{j=1}^{J} w(F^n)_j \left({g^j}^\top F^n + b^j + \eta^j_{t+n+1} \right)$$

This leads us to the final MPLG model:

$$O_{t+n+1} = \sum_{j=1}^{J} w(F^n)_j \left({g^j}^\top F^n + b^j + \eta^j_{t+n+1} \right) \tag{6.3}$$

with the mixing weights

$$w(F^n)_j = \frac{K(\xi_j, F^n; \phi_j)}{\sum_{k=1}^{J} K(\xi_k, F^n; \phi_k)}. \tag{6.4}$$

We can think of this as J PLGs, each centered at some ξ_j and responsible for some part of the space defined by F^n. The kernels act as a distance metric between F^n and ξ_j, and help define the regions of responsibility. Within each region, a single PLG is responsible for predicting O_{t+n+1}, and close to the boundaries of the regions, the predictions of multiple PLGs are smoothly mixed together. Figure 6.1 illustrates the process.

How might such a model generalize across the state space? The example of Figure 6.2 (and the lower-right corner of Figure 6.1) builds some intuition: near the Gaussian centers, the individual PLGs are nonlinearly mixed. Further away from the centers, a single PLG becomes responsible for the space, resulting in a linear function. Thus, the model interpolates nonlinearly, but extrapolates

linearly. This contrasts sharply with a mixture of un-renormalized Gaussians (also shown in Figure 6.2): as we go further away from *their* centers, the function defining O_{t+n+1} would go to zero.

The kernels are the mechanism we use to define the regions of responsibility, and so the parameters of the kernels (in this case, the mean and variance of the Gaussian) become a parameterization of the regions. In this context, our choice of Gaussian kernels was not arbitrary. One advantage to them is that changing their parameters allows us to specify in a natural way exactly how the individual PLGs will be mixed together. The left-hand side of Figure 6.1 demonstrates one aspect of this: as the Gaussians overlap less and less, the transition between mixture components becomes sharper and sharper, and approaches a sort of soft Voronoi tessellation of the space. Of course, the distances used to define the Voronoi tessellation are skewed by the covariance matrices in the Gaussians.

6.1.2 Comparison to the KPLG

In the introduction we partly motivated the MPLG with three reasons that the KPLG is insufficient to replace the PLG: generalization, the noise terms, and parameter stability. The first two issues are theoretical, and we now briefly discuss how the MPLG addresses them, but the final issue is empirical, and so a discussion of it is deferred until Section 6.4.

First, the MPLG is expected to generalize the dynamics better outside of the training region than the KPLG, especially when the KPLG uses Gaussian kernels. Figure 6.2 illustrates the difference between the two by illustrating a training set (left panel) and the trained models (middle and right panels). Both models interpolate nonlinearly in the region of the training data, but as F^n goes further away from the training region, the behavior diverges. The left panel illustrates this for the KPLG. Because the magnitude of the Gaussians tends to zero, the model in Eq. (5.1) will always return something close to zero as a prediction for O_{t+n+1}. We expect a priori that this is not the correct behavior for a dynamical model. The right panel shows the behavior for the MPLG: far away from the training data (and from the ξ_j's) the MPLG generalizes linearly, because a single PLG ends up with all of the weight.

The other insufficiency is more subtle. The term η_{t+n+1} in the KPLG has the same properties as in the PLG, and in particular $\mathrm{Cov}[\eta_{t+n+1}, F^n] = C_\eta$. The value of C_η does not depend on F^n. Recall that the representational power of the PLG comes from this property of the noise term, and while a constant value over all of F^n might be fine in a linear system, it is easy to construct nonlinear examples where C_η should vary with F^n. Rewriting the PLG in the dual form to derive the KPLG has failed to capture this. For an example of this, consider again Figure 6.2 on the left. The figure shows O_{t+n+1} as a function of F^n, with dynamics that are piecewise linear. Each piece can be perfectly modeled with with a different PLG: on the right, the parameters are $g = 1, C_\eta = -0.1$, and $b = 0$. On the left, everything flips signs: the trend is $g = -1$, and the noise term is $C_\eta = 0.1$. We see that in this case, we want C_η to vary as a function of F^n, which is impossible with the KPLG. In contrast, the MPLG uses J noise terms, each with different properties. Since these are

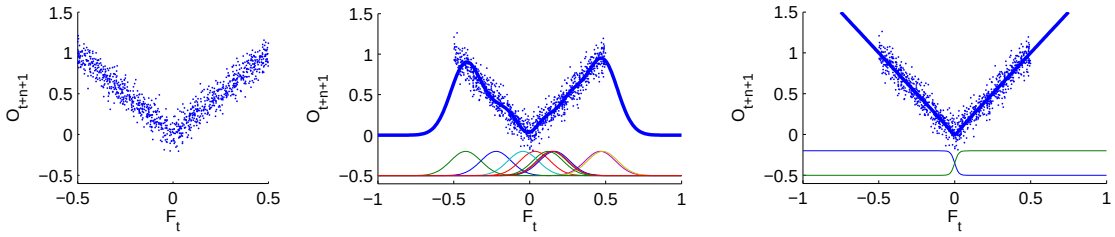


Figure 6.2: A simple piecewise linear dynamical system. The training data is shown on the left panel. The middle panel shows the results of training the KPLG on the data, with a Gaussian kernel (kernel weights are shown on the bottom). Note the poor generalization outside the training region. The right panel shows the MPLG, with two centers (renormalized weights are shown on the bottom). Note the linear generalization.

combined with weights that are a function of F^n, there is effectively a *composite* noise term which indirectly depends on F^n.

6.2 Hybrid Particle-Analytical Inference

We have been discussing how to model O_{t+n+1} as a function of F^n, but this is only part of the total state update mechanism. Recall that the state update (Eqs. 4.3 and 4.4) requires three terms: $E_t = \mathrm{E}[O_{t+n+1}]$, $C_t = \mathrm{Cov}[O_{t+n+1}, F^n]$ and $V_t = \mathrm{Var}[O_{t+n+1}]$. Computing E_t, C_t and V_t in closed form is usually impossible for complicated extension functions. Not only is the extension function of the MPLG nonlinear, it involves far more random variables: by mixing J PLGs together, we now have J noise terms, each of which is a random variable which must be propagated through the extension function.

This section discusses our hybrid inference algorithm, which elegantly sidesteps the difficulties using a form of Rao-Blackwellisation. The discussion relies on an understanding of the sigma-point approximations presented in Section 5.1.1. Given a k-dimensional multivariate Gaussian, a sigma-point approximation instantiates $2k$ sigma-points, each of which is propagated through the nonlinear function $f()$. A naive use of sigma-point approximations in the context of the MPLG would be to construct $2(n+J)$ points, based on the joint Gaussian:

$$P = \begin{pmatrix} F^n \\ \eta^1_{t+n+1} \\ \vdots \\ \eta^J_{t+n+1} \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \mu_t \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} \Sigma_t & C^1_\eta & \cdots & C^J_\eta \\ C^{1T}_\eta & \sigma^{21}_\eta & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ C^{JT}_\eta & 0 & \cdots & \sigma^{2J}_\eta \end{pmatrix} \right].$$

This is particularly inefficient if J is large (say, hundreds or thousands), because it results in $2(n+J)$ distinct values for F^n, and thus in $O(J^2)$ kernel evaluations.

There is a much better way, which is based on the following crucial observation: that although there

are $n + J$ random variables, the nonlinearities in the model which arise because of the weighting function only involve the n variables in the vector F^n. We can combine this insight with the smoothing properties of conditional expectations (also called the conditional expectation identity), which in the case of the MPLG states that

$$E_t = \mathrm{E}_{F^n}\left[\mathrm{E}[O_{t+n+1}|F^n = f^i]\right]$$

$$C_t = \mathrm{E}_{F^n}\left[\mathrm{E}[O_{t+n+1}F^{n\top}|F^n = f^i]\right] - \mathrm{E}[O_{t+n+1}]\mathrm{E}[F^{n\top}]$$

$$V_t = \mathrm{E}_{F^n}\left[\mathrm{E}[O_{t+n+1}O_{t+n+1}^\top|F^n = f^i]\right] - \mathrm{E}[O_{t+n+1}]\mathrm{E}[O_{t+n+1}^\top].$$

Our strategy is to combine these facts by *partially* instantiating sigma-points – in particular, we only instantiate $2n + 1$ sigma-points describing F^n. *Given* those sigma-points, the interior expectations are analytically tractable, and we compute the exterior expectations *over* those sigma-points.

We will begin by computing the "interior expectations," which are the terms E_t, C_t and V_t given $F^n = f^{(i)}$, which we will denote $E_t^{(i)}, C_t^{(i)}$, and $V_t^{(i)}$, respectively.

We start with $E_t^{(i)}$. Let $f^{(i)}$ be the i'th sigma-point, of which there are $2n+1$. For each sigma point, we can use the mixing weight equation Eq. 6.4 to compute a vector of J weights:

$$w_i = w(f^{(i)})$$

with $w_i \in \mathbb{R}^{J\times 1}$. Now

$$\begin{aligned}
E_t^{(i)} &= \mathrm{E}\left[O_{t+n+1}|F^n = f^{(i)}\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{J}(w_i)_j(g^{jT}f^{(i)} + b^j + \eta^j_{t+n+1})|F^n = f^{(i)}\right] \\
&= \sum_{j=1}^{J}(w_i)_j\left(g^{jT}f^{(i)} + b^j + \mathrm{E}\left[\eta^j_{t+n+1}|F^n = f^{(i)}\right]\right) \\
&= \sum_{j=1}^{J}(w_i)_j\left(g^{jT}f^{(i)} + b^j + C_\eta^{jT}(\Sigma_t^{-1})(f^{(i)} - \mu_t)\right). \qquad (6.5)
\end{aligned}$$

This looks almost like a standard PLG, except for the additional term $C_\eta^{jT}(\Sigma_t^{-1})(f^{(i)} - \mu_t)$. This comes from the noise terms, because even though they are mean-zero, they covary with F^n.

It is now convenient to re-express this equation in a matrix-vector form to clarify the rest of the equations. We will start with the noise terms, which we treat specially to simplify the rest of the equations. Let $L \in \mathbb{R}^{J\times n}$ be a matrix whose j-th row is C_η^{jT} and let Q be a vector with

$Q_j = \eta^j_{t+n+1}$. Then:

$$F_i = \mathrm{E}\left[Q|F^n = f^{(i)}\right] = L\Sigma_t^{-1}(f^{(i)} - \mu_t).$$

F_i is a vector $\in \mathbb{R}^{J\times 1}$ which captures the effects of how the sigma point covaries with each of the J noise terms.

We can now re-express Eq. 6.5 in matrix-vector form. Let $G \in \mathbb{R}^{J\times n}$ be a matrix whose j-th row is g^{jT}, and define vector B with $B_j = b^j$. Then

$$\begin{aligned} E_t^{(i)} &= \mathrm{E}\left[w_i^\top(Gf^{(i)} + B + Q)|F^n = f^{(i)}\right] \\ &= w_i^\top(Gf^{(i)} + B) + F_i \\ &= H_i + F_i. \end{aligned}$$

Here, we have split the result into two parts: the H_i part captures the trend and bias, while the F_i captures the noise terms. Now, we can efficiently compute $C_t^{(i)}$ and $V_t^{(i)}$. Let $M \in \mathbb{R}^{J\times J}$ be a diagonal matrix where $M_{j,j} = (\sigma_\eta^2)^j$. Then:

$$\begin{aligned} E_t^{(i)} &= \mathrm{E}\left[O_{t+n+1}|F^n = f^{(i)}\right] \\ &= H_i + F_i \end{aligned} \tag{6.6}$$

$$\begin{aligned} C_t^{(i)} &= \mathrm{E}\left[O_{t+n+1}F^{n\top}|F^n = f^{(i)}\right] \\ &= (H_i + F_i)f^{iT} \end{aligned} \tag{6.7}$$

$$\begin{aligned} V_t^{(i)} &= \mathrm{E}\left[O_{t+n+1}^\top O_{t+n+1}|F^n = f^{(i)}\right] \\ &= H_iH_i^\top + H_iF_i^\top + F_iH_i^\top + \\ &\quad w_i^\top(D - \mathrm{diag}(\mathrm{diag}(L\Sigma_t^{-1}L^\top)) + F_iF_i^\top)w_i \end{aligned} \tag{6.8}$$

We compute Eqs. (6.6)-(6.8) for each sigma-point, and then use expectations over them to compute the final terms:

$$E_t = \frac{1}{2n+1}\sum_{i=1}^{2n+1} E_t^{(i)} \tag{6.9}$$

$$C_t = \frac{1}{2n+1}\sum_{i=1}^{2n+1} C_t^{(i)} \tag{6.10}$$

$$V_t = \frac{1}{2n+1}\sum_{i=1}^{2n+1} V_t^{(i)}. \tag{6.11}$$

This completes our development of the hybrid-particle analytical inference method.

The general method is summarized as follows:

- Instantiate sigma-points for the minimum number of variables needed to make the model tractable.
- Analytically compute terms based on the model given the sigma-points.
- Compute posterior statistics using expectation smoothing over the sigma-points.

The final algorithm is shown in Figure 6.3.

6.3 Model Learning

The MPLG requires several parameters. First, the dimension n of the system, and the parameters describing the mixing weights. In the case of renormalized Gaussian kernels, these are the basis function centers ξ_j, weights α_j, and the covariance matrices ϕ_j. For each PLG, the individual PLG parameters g^j and noise statistics C_η^j and $(\sigma_\eta^2)^j$ are also needed.

We are interested in learning the parameters from training data. This data will be given as a set of trajectories from the system, with each trajectory consisting of at least $n+1$ sequential observations. As in previous algorithms, we will use the suffix history method (which is explained in detail in Section 3.4, and which we briefly recap here). We will slice these trajectories into all possible training pairs $(f_t^{n(i)}, o_{t+n+1})$ where $f_t^{n(i)} \in \mathbb{R}^n$ is a vector of n successive observations (representing a noisy sample of some O_t), and $o_{t+n+1} \in \mathbb{R}$ is the $(n+1)$-th observation (a sample of the corresponding O_{t+n+1}, or the state extension). We then collect the pairs into the set S. Figure 5.3 graphically illustrates the process.

Model Order Selection. We must first estimate the order of the model, which includes the system dimension n and the number of basis functions J. For our experiments, we use cross-validation to select parameters from a set of likely candidates. More detail on this step can be found in Section 5.2, where the same problem is discussed in the context of the KPLG.

Finding Basis Function Parameters. Next, we must determine the basis function centers ξ_j and covariance matrices ϕ_j. As in the KPLG (Section 5.2), we used the dictionary-based selection method of Engel et al. (2004). Specifically, we set $\phi_j = \sigma_\phi^2 I$ and then constructed a set of $f_t^{n(i)}$'s whose features are almost linearly independent.

Estimating Individual PLG Parameters. We can now determine the mixing weights for each PLG at any point, so we estimate the parameters of each PLG individually, using weighted versions of the regressions and sample statistics needed. To start, we collect each o_{t+n+1} into a vector $O \in \mathbb{R}^{|S|}$, and collect each $f_t^{n(i)\top}$ into a matrix $F \in \mathbb{R}^{|S| \times n}$. We then compute the weights for each

Algorithm MPLG-HYBRID-PARTICLE-ANALYTICAL-INFERENCE

Input: μ_t, Σ_t

Given: $f, K, \xi_1, \cdots, \xi_J, \phi_1, \cdots, \phi_J$

Compute:

- Construct a set of $2n$ sigma points describing F^n:

$$f_t^{n(2i-1)} = \mu_t + (\sqrt{n\Sigma_t})_i$$

$$f_t^{n(2i)} = \mu_t - (\sqrt{n\Sigma_t})_i$$

- Compute a weight vector for each sigma point:

$$w(f_t^{n(i)})_j = \frac{K(\xi_j, f_t^{n(i)}; \phi_j)}{\sum_{k=1}^{J} K(\xi_k, f_t^{n(i)}; \phi_k)}$$

- For each $f_t^{n(i)}$, compute
  $E_t^{(i)}$ (Eq. 6.6),
  $C_t^{(i)}$ (Eq. 6.7) and
  $V_t^{(i)}$ (Eq. 6.8).

- Compute empirical posterior statistics:

$$E_t = \frac{1}{2n} \sum_{i=1}^{2n} E_t^{(i)}$$

$$C_t = \frac{1}{2n} \sum_{i=1}^{2n} C_t^{(i)} - E_t \mu_t$$

$$V_t = \frac{1}{2n} \sum_{i=1}^{2n} V_t^{(i)} - E_t^2$$

Return: E_t, C_t, V_t

Figure 6.3: Hybrid particle-analytical MPLG inference.

training point using the renormalized kernels:

$$w(f_t^{n(i)})_j = \frac{K(\xi_j, f_t^{n(i)}; \phi_j)}{\sum_{j=1}^{J} K(\xi_j, f_t^{n(i)}; \phi_j)}.$$

We also define a normalizing constant as the sum of all of the weights for each PLG:

$$N_j = \sum_{i=1}^{|S|} w(f_t^{n(i)})_j.$$

For each PLG, collect the weights $w(f_t^{n(i)})_j$ into a diagonal weight matrix $W_j \in \mathbb{R}^{|S|\times|S|}$.

Now, we can estimate the linear trend for each PLG j using weighted least-squares:

$$\widehat{g^j} = (F^\top W_j F)^{-1} F^\top W_j Y.$$

To estimate the noise statistics, we first compute the noise term for each training point from the perspective of each PLG, which is

$$\eta_{ij} = o_{t+n+1} - \widehat{g^j}^\top f_t^{n(i)}.$$

Then, the estimated variance of η_{t+n+1}^j is

$$\widehat{(\sigma_\eta^2)^j} = \tfrac{1}{N_j - 1} \textstyle\sum_{i=1}^{|S|} w(f_t^{n(i)})_j (\eta_{ij})^2.$$

To estimate C_η^j, we run the algorithm on the training data with $C_\eta^j = 0$ and record our estimate of μ_t at each t, called μ_i. We then compute

$$\mathrm{Cov}[F^n, \eta_{t+n+1}] = \mathrm{E}[(F^n - \mu_t)(\eta_{t+n+1})],$$

which is simply

$$\widehat{C_\eta^j} = \tfrac{1}{N_j - 1} \textstyle\sum_{i=1}^{|S|} w(f_t^{n(i)})_j (f_t^{n(i)} - \mu_i)\eta_{ij}.$$

While estimating both $(\sigma_\eta^2)^j$ and C_η^j we have used the fact that $\mathrm{E}[\eta_{t+n+1}^j] = 0$ for all t.

Estimating KPLG and KAR Parameters. Parameters for the KPLG and KAR algorithms were estimated similarly to those of the MPLG. The ξ_j's were selected with a dictionary, and the α's were computed using regularized least-squares kernel regression, with λ the regularization coefficient.

6.4 Experiments and Results

We tested the PLG, MPLG, KPLG and KAR algorithms on the same problems described in Section 5.3.1. There was one linear dynamical system (the "Rotation" problem) and four nonlinear

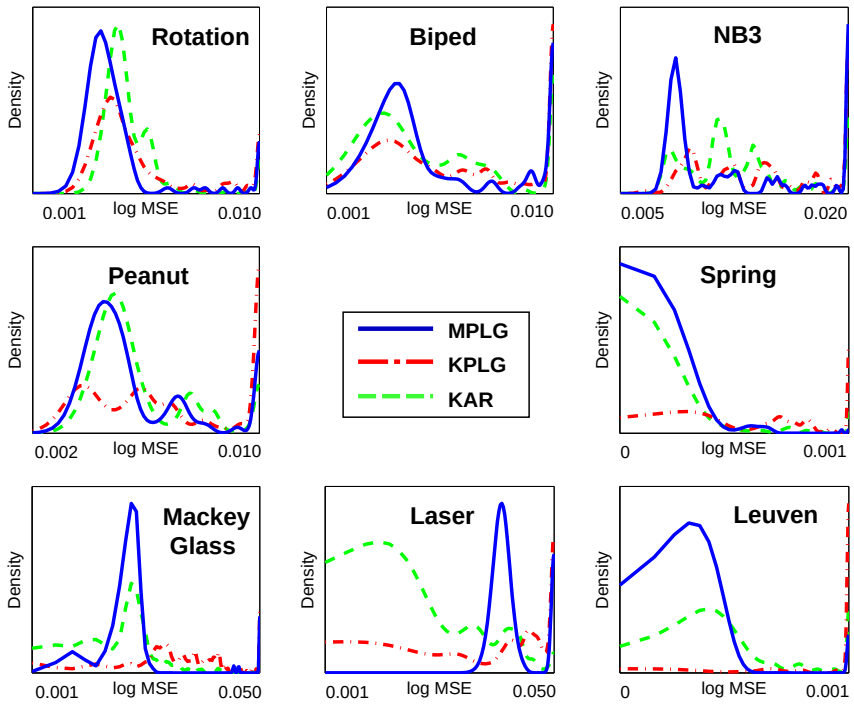


Figure 6.4: Qualitative comparison of parameter stability. Shown is the density of all MSEs generated. Curves that are more tightly peaked indicate greater parameter stability, while curves that are farther to the left indicate better MSE.

dynamical systems (the "Biped," "Peanut," "NB3," and "Spring" problems), where the underlying generative model was known. We also tested on three well-known timeseries benchmarks (Santa Fe Laser, Mackey-Glass, and K.U. Leuven). A sigma-point approximation was used for the KPLG.

Parameters were selected by 10-fold cross-validation. Algorithms were judged on the mean-squared error (MSE) of their predictions, meaning we are not attempting to estimate latent state (but we are allowed to *use* state). All data sets were normalized to be in $[0, 1]$. All algorithms used the Gaussian kernel. For the initial state, we set Σ_0=$1e^{-5}I$ and μ_0 to be the first n values of the test sequence. Algorithms were tested on n=$2, \cdots, 6$, σ_ϕ^2=$0.1, 0.4, 0.8, 1.2$, λ=$0.00001, 0.001, 0.01$, and ν=$0.0001, 0.001, 0.01$ (ν is the dictionary threshold). All problems except Laser were trained on 2000 sequential observations and tested on a 200 observation continuation; Laser had 1000 training and 100 testing observations.

6.4.1 Results

Figures 6.4 and 6.5 summarize our results, which are very encouraging. Figure 6.4 shows the results qualitatively. Each parameter setting for each algorithm generated a MSE; the figure plots their log distribution. This examines the expected performance for any given parameter setting; a sharply peaked distribution on the left side is desired (implying low expected MSE). Outliers are lumped on the right-hand side.

Problem	Best	Problem	Best
Rotation	PLG	Spring	KAR
Biped	MPLG	M.G.	KAR
Peanut	MPLG	Leuven	MPLG/PLG
NB3	MPLG/PLG/KAR	Laser	KAR

Figure 6.5: MPLG empirical results on the short-term prediction problem. Shown are the best performing algorithm(s) for each test problem.

From Figure 6.4, three results are evident. First, the MPLG's density curve is often stacked on the left-hand side, as desired. The curve is also more peaked than that of other algorithms, indicating that its performance is less sensitive to the exact choice of parameter. It also shows fewer outliers than the KPLG.

Figure 6.5 shows our results quantitatively. Here, we have used 10-fold cross-validation to select parameters; all algorithms with an MSE within 5% of the lowest are reported as "Best." The MPLG is among the best performing algorithms on four out of eight problems, and in particular, it performed well on the nonlinear dynamical systems, where there really *is* an opportunity to leverage infinite memory via state. In contrast, KAR has performed well on the timeseries problems; in particular, it wins on the Mackey-Glass series, which really *is* an autoregressive model. The exception is Spring, but this is expected: it is deterministic and noiseless, so n past observations and n predictions are equivalent; the uncertainty the MPLG/KPLG models is unhelpful. PLG won on the linear problem, which is also unsurprising.

Together, the quantitative and qualitative results suggest several conclusions. First, that not only are the very best MSEs often obtained with the MPLG, but for any given parameter setting, the MPLG is likely to outperform other models. Second, that the nonlinear models are outperforming their linear counterparts. Third, that the MPLG is superior to the alternative nonlinear version of the PLG, the KPLG. Fourth, that the MPLG is indeed capturing state, and is therefore superior to autoregressive models in situations where state can be leveraged. Not reflected in these results is the fact that the best parameters were rarely selected for KPLG because of outliers in the cross-validation runs, but this is part of the point: MPLG is more stable than KPLG.

6.5 Application to a Traffic Prediction Problem

We now present an application of MPLGs to a traffic prediction problem. The goal of this section is threefold: first, we wish to apply the MPLG to a richer domain than the timeseries problems we have dealt with so far, by investigating the hypothesis that driving behavior can be captured more accurately with the MPLG than the PLG. The second goal of the section is to generalize the MPLG mixing equations to allow *any* variable to be used to define mixing weights, including unmodeled "context variables". Third, we present a mechanism to help optimize the choice of mixing regions.

The problem statement for this application is to model the dynamics of traffic flow based on the

NGSim traffic data set from the Federal Highway Administration [1]. The data consists of approximately 15 minutes of traffic data on Interstate 80. About 3,000 cars are individually tracked, and variables such as their position, velocity, acceleration, and distance to other cars is reported every 1/10th of a second. At one level of abstraction, the problem is trivial: a PLG trained naively on position, velocity and acceleration variables will recover basic Newtonian dynamics. However, it seems clear that driving behavior is context sensitive: predicting actions such as braking or lane changes for a given car depends on the cars around it.

To use the MPLG in such a problem, the key problem is determining how to split the problem space into regions such that the dynamics are linear within each region. We examine different splitting variables, experimenting with both continuous and discrete splitting variables. The general idea is to define some sort of splitting criteria which divides the problem space into different regions. The splitting criteria can be based on state variables (such as the position and velocity of a car), context variables (such as the relative velocity of a neighboring car), binary variables (presence/absence of a neighboring car), or even more abstract variables (is the driver of the car to my right crazy?).

We ran two sets of experiments, which used two different kinds of context variables in order to define mixing weights. We discuss the choices below, and then present the results, generally concluding that many different choices of context variables could improve the model's quality.

6.5.1 Generalized Mixing Variables

In the previous exposition of the MPLG, we used solely the variable F^n to define our mixing weights. Here, we introduce the idea that *any* variable could be used to compute the mixing weights. Importantly, these variables may or may not be modeled – they may be state variables, control variables, other exogenous variables or any combination of the above. To clarify this, consider the original MPLG extension equation, defined as:

$$O_{t+n+1} = \sum_{j=1}^{J} w(F^n)_j \left({g^j}^\top F^n + b^j + \eta^j_{t+n+1} \right).$$

These weights $w(\cdot)_j$ are a function solely of F^n. Here, we generalize this such that

$$O_{t+n+1} = \sum_{j=1}^{J} w(\mathcal{C}_t)_j \left({g^j}^\top F^n + b^j + \eta^j_{t+n+1} \right).$$

where $\mathcal{C}_t$ is any variable which is available at time t ($\mathcal{C}_t$ is a mnemonic for "context variable").

To see the importance of allowing contextual variables to help define dynamics without modeling those variables, consider the example of weather and driving. We may achieve better accuracy if we have different models depending on the weather: one model might be appropriate for rainy

[1] Available from http://ngsim.fhwa.dot.gov/

weather, and another might be appropriate for dry weather. A mixture of the two models might be appropriate for misty or drizzling weather – but under no circumstances do we wish to have to predict the weather in order to create a good driving model!

Another subtle difference between using context variables to define mixing weights and using the variable F^n is the fact that the context variables are not considered random variables. That is, they are simply given at each time t. This means that, unlike the case of using F^n, we do not have to propagate a random variable through the mixing weight equations, meaning that the sigma-point approximations are not necessary. Thus, at each time t, we observe the context variable, compute the mixing weights, and then compute an "aggregate PLG," which is defined through a weighted combination of parameters. For example, the term E_t is computed as:

$$\begin{aligned} E_t &= \mathrm{E}[O_{t+n+1}] \\ &= \mathrm{E}[\sum_{j=1}^{J} w(\mathcal{C}_t)_j \left({g^j}^\top F^n + b^j + \eta^j_{t+n+1} \right) \\ &= \sum_{j=1}^{J} w(\mathcal{C}_t)_j \left({g^j}^\top \mathrm{E}[F^n] + b^j + \mathrm{E}[\eta^j_{t+n+1}] \right) \\ &= \mathrm{E}_w[g]^\top \mathrm{E}[F^n] + \mathrm{E}_w[b] \\ &= \mathrm{E}_w[g]^\top \mu_t + \mathrm{E}_w[b] \end{aligned}$$

where we have defined $\mathrm{E}_w[\cdot]$ to be the expectation with respect to the weights. Similar expressions hold for C_t and V_t.

We see that this equation looks just like an ordinary PLG extension equation (cf. Eq. 4.5), except the trend parameter $\mathrm{E}_w[g]$ of this PLG is an aggregate of J other PLGs' trend parameters. The same is true for the bias term is $\mathrm{E}_w[b]$. The important point is that although these parameters vary with time, at any given time they are fixed, so the variable F^n can be propagated normally through the resulting equations.

6.5.2 Continuous Contextual Variables

The first contextual variables we used were the quantities *y-velocity*, *y-acceleration*, *headway* and *change-in-headway*. The y variables represent the velocity and acceleration of the car in the direction of travel. The headway variable represents the distance from the current car to the car directly in front of it, in the same lane (measured in feet). The variable change-in-headway is a derived quantity, which effectively representing the relative velocity of the car in front of the current car.

For this experiment, J "centers" were sampled from the training data, and J corresponding PLGs were created. The mixing weights were defined in the ordinary way, except that they are a function

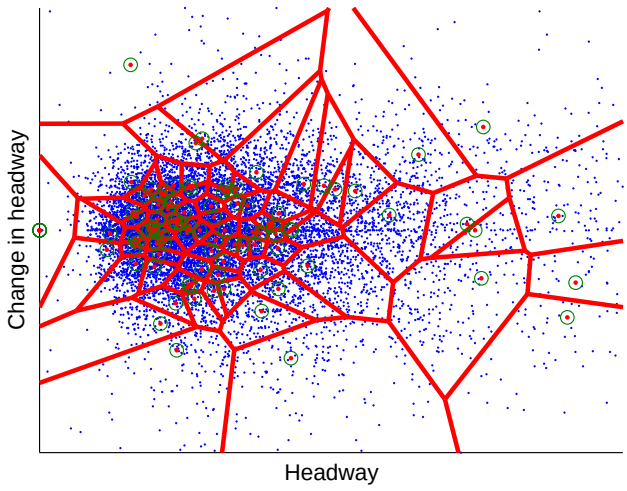


Figure 6.6: A Voronoi diagram plotting the tessellation of context variables. Shown is the tessellation of "Headway" and "Change-in-headway". The centers ξ_j are shown as circles, and the regions are shown with lines.

of the current context variable $\mathcal{C}_t$, instead of F^n:

$$w(\mathcal{C}_t)_j = \frac{G(\mathcal{C}_t; \xi_j; \sigma^2)}{\sum_k G(\mathcal{C}_t; \xi_k; \sigma^2)}. \tag{6.12}$$

Here, the ξ_j's are the randomly sampled centers, $G()$ is a spherical Gaussian kernel and σ^2 is the variance. Context variables were scaled to have approximately unit variance.

Using the context variables in this way effectively creates a soft Voronoi tessellation of the dynamics. Figure 6.6 shows a diagram representing a scatter plot of samples of two of the context variables ("headway" and "change-in-headway") as well as the centers and their respective Voronoi regions. A PLG was then learnt for the set of samples corresponding to each region, using the weighted learning algorithms of Section 6.3.

6.5.3 Binary Contextual Variables

The second set of experiments examined the effect of splitting on "radar variables." Radar variables were derived from the cars around the current car by discretizing the region around the car into a 31 x 21 grid. Inside of each grid square is a binary random variable representing the presence or absence of a car at that location. The grid extended approximately 80 feet in front of the car, and about 20 feet behind and to either side. The gridding was done uniformly. This is shown in Figure 6.7, left and middle panels. Thus, in the case, the context variable $\mathcal{C}_t$ is a binary vector $\in \mathbb{R}^{651}$.

We ran a slightly different kind of experiment using these variables. Most of the weighting schemes we have previously used rely on Gaussians or other kernels and define soft mixing weights. Here,

we use binary weights, which define hard regions. We experimented with each radar variable to determine its quality as a context variable. For a given radar variable i, we assigned training samples one of two sets, depending on whether the variable $\mathcal{C}_{ti}$ was on or off for that sample.

$$\begin{aligned} w(\mathcal{C}_t; i)_0 &= \delta(\mathcal{C}_{ti}, 1) \\ w(\mathcal{C}_t; i)_1 &= \delta(\mathcal{C}_{ti}, 0) \end{aligned} \tag{6.13}$$

where $\delta(i, j)$ is the Kronecker delta. Thus, each variable split the training set into two regions (not necessarily of the same size). PLGs were then learnt for each region.

6.5.4 Experiments

The general experimental procedure is simple. Given a weight equation (either Eq. 6.12 or Eq. 6.13), PLGs are learned from data using the weighted regressions and sample statistics of Section 6.3, and the resulting models are combined together to form a composite dynamical system. At each timestep, the aggregate PLG was formed, and we used the standard PLG update equations.

It is important to note that when learning a PLG, we only used two variables from the NGSim data: x position and y position. Because the PLG is capable of modeling linear dynamical systems, and because of the linear relationship between position, velocity, and acceleration, these were sufficient to account for the vast majority of vehicle dynamics. For all experiments, we set n (the length of the window into the future) to 3. There were about 12 million data points.

We tested all algorithms on a set of data for about 300 cars, asking them to make one-step predictions at each timestep. We judged all algorithms on both MSE and log-likelihood of the predictions. We used steady-state filtering to help alleviate numerical problems with the covariance matrices. For comparison, we trained a 3rd order AR model, as well as a standard PLG with $n = 3$. The overall results are shown in Figure 6.11, but we will present numerous other results first.

Figure 6.8 shows the results of our first experiment using the continuous contextual variables. For this experiment, we tested whether or not using randomly sampled ξ_j's would be beneficial. There are two parameters to choose: the number of basis functions J, and the width of the Gaussians σ^2. We experimented with $J = 2, 4, 8, 16, 32, 64$ and 100, and with $\sigma^2 = 0.01, 0.1$, and 1.0.

The figure shows a number of interesting phenomena. Consider the top row, corresponding to $\sigma^2 = 0.01$. Here, adding more basis functions generally had two effects: first, the log-likelihood of the data went up, and second the MSE went down. In addition, the variance of the likelihood was much lower for larger numbers of basis functions, which is to be expected: the context variables are 4 dimensional, so it is easy to imagine that 100 regions will more or less capture all of the interesting behavior there is, while having only 2 regions could capture very different behavior depending on the exact split between the regions (for a pictorial example of this, see Figure 6.9).

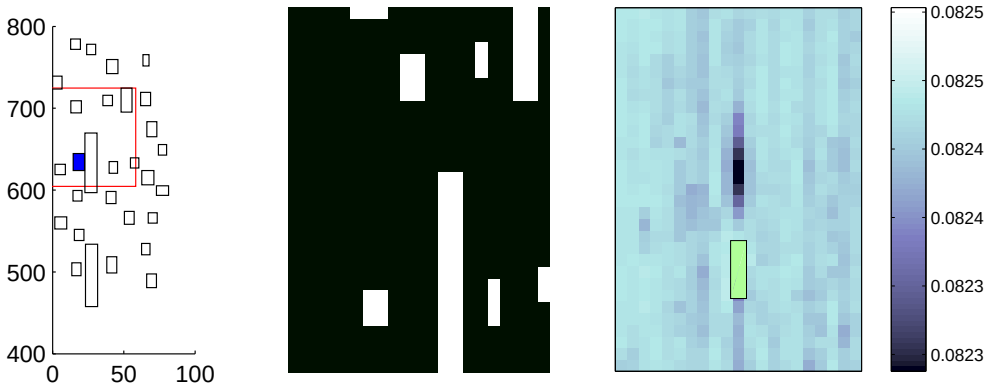


Figure 6.7: Results of using radar variables for splitting. On the left, the road context of a car. The filled square is the current car, and the large box around it is the radar extent. In the middle: the resulting binary radar image. On the right: results of splitting on radar variables. Color represents the norm of the error term, which is related to MSE. Lower is better.

One might wonder why the likelihood went up so dramatically, while the error was only reduced slightly. Higher likelihood results from making more confident predictions, indicating that the learned PLGs generally has smaller noise terms. This is probably due to smaller learnt noise terms, and may be a consequence of eliminating outliers / noisy samples from the data set (since those will tend to be associated with outlying regions). On the other hand, it is difficult to improve upon the basic Newtonian physics of position, velocity and acceleration at the short timescales used here (recall that $n = 3$, and that each timestep represents 1/10th of a second). It is likely that we are not capturing large-scale driving behaviors, but rather more subtle effects. For example, cars might tend to brake more if there is another car close (the headway variable is small) and the distance is decreasing (change in headway is negative).

Another interesting effect is related to the Gaussian's variance. The top row has the smallest Gaussian width, while the bottom row has the largest. With the smallest Gaussian, there is a pronounced difference between small J and large J. With the wider Gaussians, adding more regions does not help as much. This is probably because with very flat Gaussians, the weights don't change very much depending on the number or positions of the Gaussians – effectively, every Gaussian is assigned to cover the entire space, and the result is roughly equivalent to using a single PLG.

Similar results, although not as strong, were obtained when splitting on the radar variables. Figure 6.7 shows a thermal image of each radar variable. The color of each variable represents the norm of the noise term (a rough indicator of MSE) which resulted from splitting the dynamics on that variable. The car is shown as a black box in the middle of the radar image.

Here, it is clear that splitting on variables directly in front of the car tend to yield the best composite

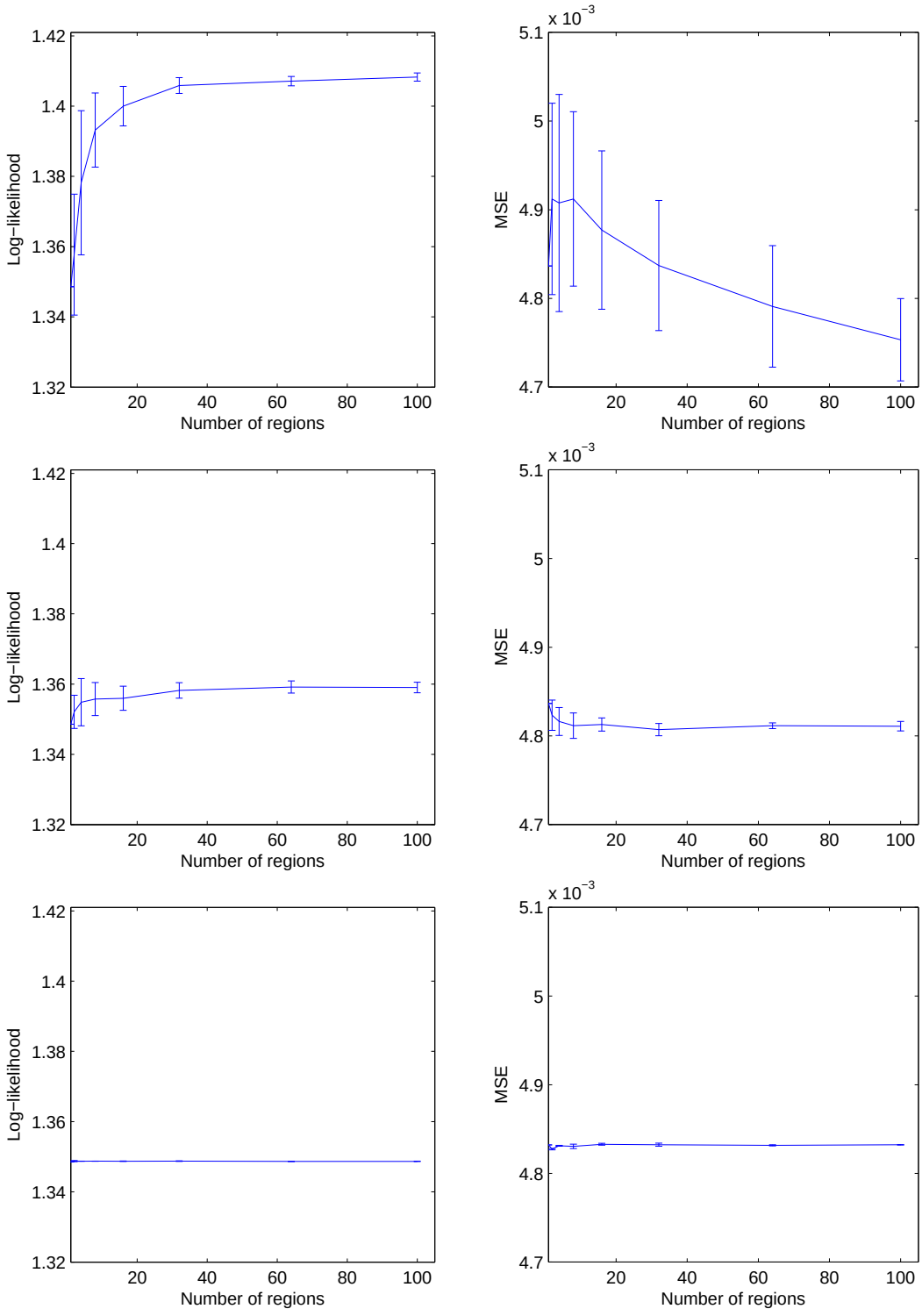


Figure 6.8: Results of sampling random centers on the traffic prediction problem. The left column shows log-likelihood and the right shows MSE. The top row uses $\sigma^2 = 0.01$, the middle row $\sigma^2 = 0.1$, and the bottom row $\sigma^2 = 1.0$.

models. We hypothesize that this is due to the same braking behaviors: if there is a car very close to the front-center of the current car, it is likely that the dynamics will be slightly different than the dynamics of when there are no cars around. A more detailed investigation is needed to validate this hypothesis.

6.5.5 Optimizing Choice of Centers

The results in the previous section indicate that a larger number of regions generally result in higher accuracy and increased confidence. However, especially for small numbers of regions, the variance in both MSE and log-likelihood was fairly large. In this section, we adopt a different modeling strategy: instead of randomly sampling centers, we *optimize* the choice of centers. While such an optimization process would be possible in the context of the more general MPLG (with mixing weights defined as a function of F^n), it is particularly easy in the case of context variables because they are not considered random variables.

In this section we develop a simple gradient based optimizer. In both the PLG and the MPLG, parameter estimation is a straightforward linear regression and sample statistics. Our approach to optimization will be to measure the sum-squared error of the regression, and then take its gradient with respect to the ξ_j's. We will assume that we have a training set which consists of samples from Z_t and O_{t+n+1}. We will denote these samples z_i and o_i. We define the residual of each datapoint as

$$r_i = \sum_j w(\mathcal{C}_i; \xi_j)({g^j}^\top z_i + b^j) - o_i \tag{6.14}$$

where the weights are defined in Eq. 6.12 (we only consider the case of soft weights). The sum squared error for the entire dataset is:

$$\text{SSE} = \sum_{i=1}^{T} r_i^\top r_i. \tag{6.15}$$

In Eq. 6.12, each ξ_j is the basis function center. These are the variables we wish to optimize, but we are also looking for the regression parameters at the same time. Because there are effectively two sets of variables we are optimizing, we adopt an EM-style optimization algorithm (Dempster et al., 1977). We will begin by fixing the basis function centers, computing the weights, regressing and computing the residuals. We will then compute the gradient of the sum squared error with respect to our basis function centers, holding the regression parameters constant.

The gradient of SSE with respect to ξ_l is given by:

$$\frac{\partial \text{SSE}}{\partial \xi_l} = \frac{\partial}{\partial \xi_l}\left[\sum_{i=1}^{T} r_i^\top r_i\right]$$

$$
\begin{aligned}
&= \sum_{i=1}^{T} 2r_i^\top \frac{\partial}{\partial \xi_l} \left[\sum_j w(\mathcal{C}_i; \xi_j)({g^j}^\top z_i + b^j) - o_i \right] \\
&= \sum_{i=1}^{T} 2r_i^\top \sum_j ({g^j}^\top z_i + b^j) \frac{\partial}{\partial \xi_l} \left[w(\mathcal{C}_i; \xi_j) \right] \qquad (6.16)
\end{aligned}
$$

where the third line follows by the fact that we are holding the parameters of PLG constant. We now must compute how the weights change as the basis function centers ξ_l change. In general, this will depend on the kernel used to define the mixing weights. We assume, as we have assumed before, that the kernel is a Gaussian, and that ξ_l is the mean of that Gaussian. The only trickery in this derivative is to notice that the renormalization step means that every weight depends on every basis function center. We will therefore split this derivation into two parts: the first is for when $l = j$, and the second is for when $l \neq j$.

Case 1 ($l = j$):

$$
\begin{aligned}
\frac{\partial w(\mathcal{C}_i; \xi_j)}{\partial \xi_l} &= \frac{\partial}{\partial \xi_l} \frac{G(\mathcal{C}_i; \xi_l; \Sigma_l)}{\sum_k G(\mathcal{C}_i; \xi_k; \Sigma_k)} \qquad (6.17) \\
&= (N - G(\mathcal{C}_i; \xi_l; \Sigma_l)) G(\mathcal{C}_i; \xi_l; \Sigma_l)(-\mathcal{C}_i - \xi_l)(\Sigma_l^{-1})/N^2 \qquad (6.18)
\end{aligned}
$$

where $N = \sum_k G(\mathcal{C}_i; \xi_k; \Sigma_k)$ is the normalizing constant.

Case 2 ($l \neq j$):

$$
\begin{aligned}
\frac{\partial w(\mathcal{C}_i; \xi_j)}{\partial \xi_l} &= \frac{\partial}{\partial \xi_l} \frac{G(\mathcal{C}_i; \xi_j; \Sigma_l)}{\sum_k G(\mathcal{C}_i; \xi_k; \Sigma_k)} \\
&= -G(\mathcal{C}_i; \xi_j; \Sigma_j) G(\mathcal{C}_i; \xi_j; \Sigma_j)(-\mathcal{C}_i - \xi_j)(\Sigma_j^{-1})/N^2 \qquad (6.19)
\end{aligned}
$$

where $N = \sum_k G(\mathcal{C}_i; \xi_k; \Sigma_k)$ is the normalizing constant.

The optimization procedure is now simple. We can compute the gradients of the sum-squared error in Eq. 6.15, by computing Eq. 6.16, Eq. 6.17, and Eq. 6.19. We then use those gradients in any gradient based optimizer we wish, such as steepest descent, LBFGS, etc. In general, this optimization problem is non-convex, which means that more advanced optimization methods (such as homotopy optimization, momentum-assisted gradient descent, etc.) may improve the solution.

6.5.6 Results of the Center Optimizer

To validate the idea of the center optimizer and to build intuition, we first present a small toy data set. Figure 6.9 shows an example of the use of the gradient optimizer. The figure shows data which resembles a standard broken-stick dataset (Toms and Lesperance, 2003), which could be well modeled with two mixture components (if they were known). The left-side shows the results of using two randomly sampled basis functions to define the mixing weights and hence the

regions. Unsurprisingly, the resulting regions do not respect the natural break in the data, and the result is a poor model. The right-hand side shows the results after optimizing the choice of basis function center using a naive steepest descent optimizer with a simple line search, after 10 iterations. We see that the resulting Gaussians define weights which divide the data almost perfectly into the appropriate halves, and the overall model fits much better: the error is lower, and the variance is also lower.

Figure 6.10 shows the results of the center optimizer on the traffic data. There are several points worth noticing in each panel, so we deal with each in turn and then draw a few general conclusions.

The panel in the upper-left shows the change in the objective function before and after optimization. For example, when using 32 basis functions, the average SSE was about 4.2e4 before optimization, and about 4.1e4 after optimization. Both numbers had fairly low variance. This implies that the gradient optimizer was able to successfully reduce the objective function, and that the effect was more pronounced as the number of basis functions increased.

But did this translate into actual gains in modeling accuracy for the MPLG? The results are mixed. The panel in the upper-right shows the log-likelihood of the data before and after optimization. In general, the optimizer was able to produce an increase in likelihood, and the increase was more statistically significant as the number of basis functions increased. However, the data suggests a stronger conclusion, which is that adding more basis functions has a much greater effect than optimizing a fixed number of them. The move from 4 basis functions to 32 basis functions results in a much greater gain in likelihood than the move from 32 unoptimized basis functions to 32 basis functions.

The story is about the same for MSE (shown in the bottom panel of Figure 6.10), although here the trend is less clear. For small numbers of basis functions, the mean MSE actually went up as a result of applying the optimizer, although the error bars are so large it is unlikely this effect is statistically significant. At higher numbers of basis functions, the MSE looks like it is reduced, but again the effect may not be significant. In any case, the change in MSE is very minor: about a 2% reduction when using 32 basis functions.

There are two general conclusions here: first, the optimizer generally appears to successfully reduce the SSE, and second, this reduction does generally translate into the gains we are expecting (increased likelihood and reduced MSE). None of the effects are very pronounced, however, which suggests two things: first, it is difficult to improve on basic Newtonian dynamics at the timescale the data is operating at. Larger windows of time may reveal more interesting effects that could be picked up by different regions, but at 3/10'ths of a second, there isn't much variation. Second, the context variables we have selected may not be the best ones. While they did demonstrate some effect, it is possible that other variables would have a much greater effect. It seems likely that the gradient optimizer would have more pronounced effects with higher dimensional context variables.

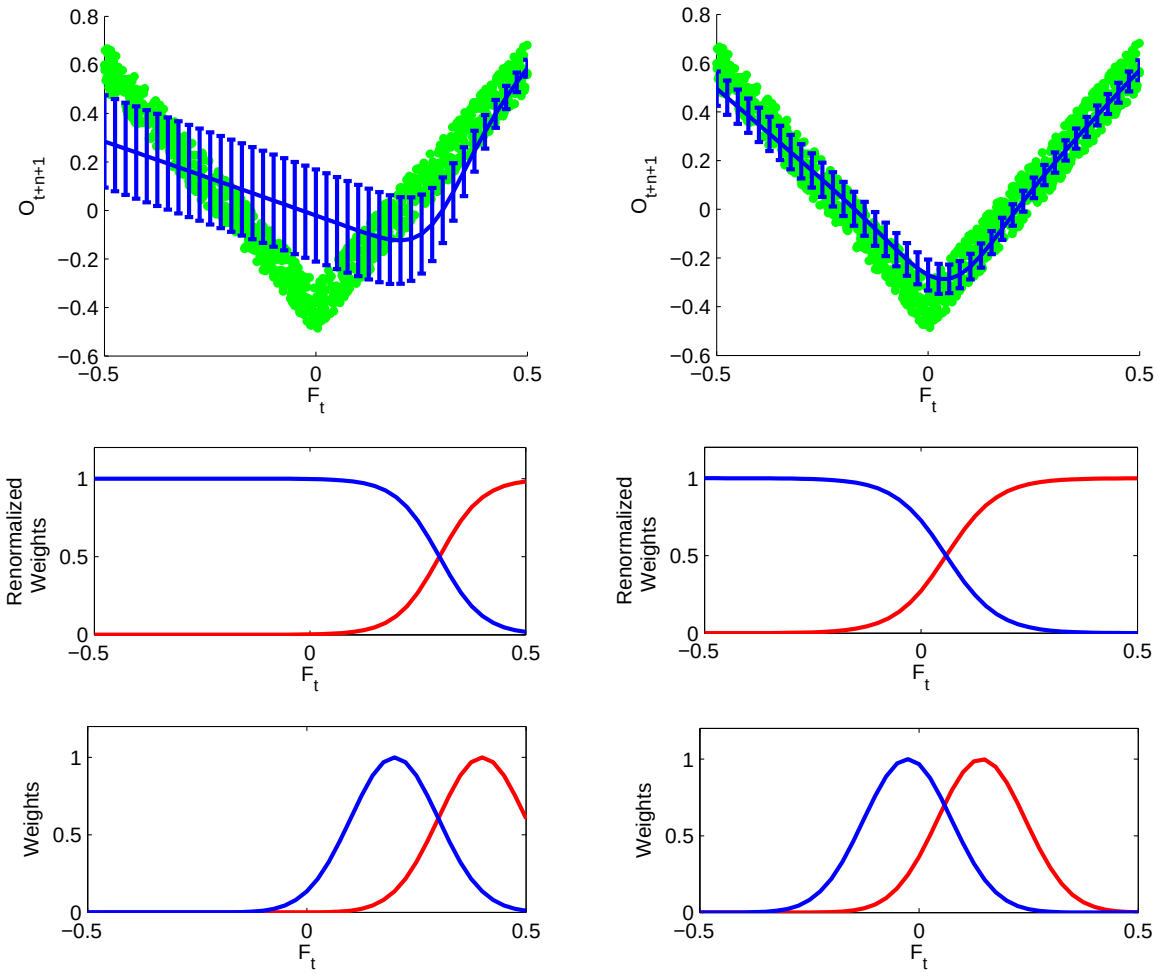


Figure 6.9: Optimizing MPLG centers. Choosing the appropriate centers for the Gaussians defining the mixture components of the MPLG can make a large difference in the quality of the regressions. Shown on the left: the bottom figure shows the Gaussians used, and the middle figure shows the renormalized weights. The top figure shows data points (in green) which represent O_{t+n+1} as a function of F^n. The regression is poor because the weights do not respect the actual break in the data. On the right: the equivalent figures for a different set of centers. These centers define regions aligned much more closely with the natural break in the data, and the resulting function is a much better approximation. The Gaussians on the right were obtained by using the optimization routine discussed in Section 6.5.5.

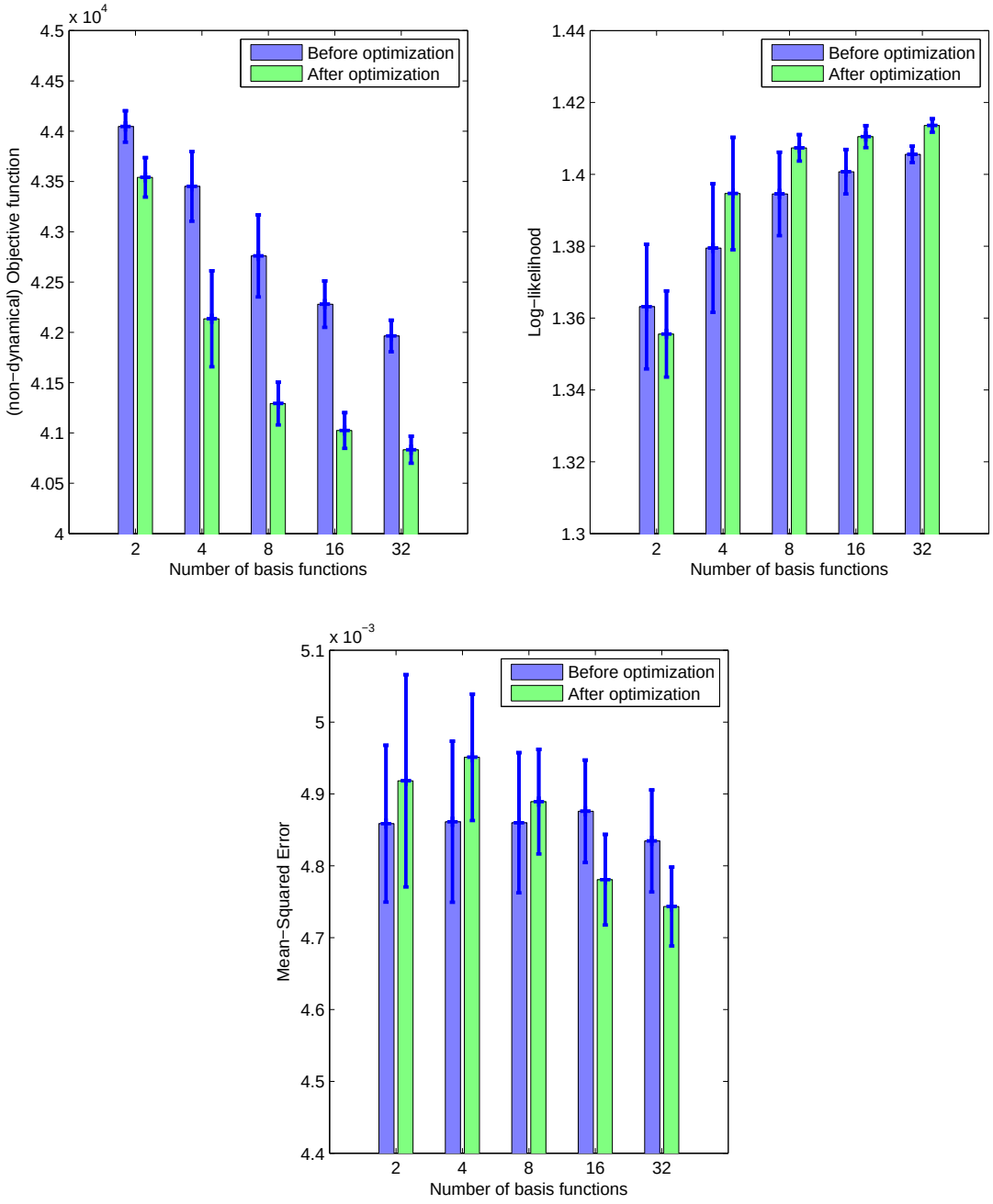


Figure 6.10: Optimization results on traffic prediction problem. Each plot represents a different measure of the data: the upper-left is the sum-squared error objective function, the upper-right is the log-likelihood on the test set, and the bottom is the MSE on the test set. The blue bars represent the distribution before optimization, and the green bars represent the distribution after optimization.

Algorithm	Log-likelihood	MSE
AR(3)	0.9098	0.0951
PLG	1.3489	0.00483
MPLG	**1.4083**	**0.00475**

Figure 6.11: A comparison of accuracies on the traffic prediction problem.

6.5.7 Observations

These results appear to validate our hypothesis: by treating traffic prediction as a nonlinear dynamical system, more accurate models can be learnt. We have also investigated the idea of capturing nonlinearities by modeling the dynamics in a piecewise linear way, where the pieces are defined based on exogenous variables. The idea is simple, and seems to be moderately effective.

We have shown that a few simple splitting choices has resulted in models that are better than naive linear models. However, these choices were largely ad-hoc, and more sophisticated versions would likely result in still better models. For example, when splitting on radar variables, the presence/absence of a car about 10-20 feet in front of the current car seems to be important. However, this variable only incorporates first-order information. Higher-order features may help here: it may be useful, for example, to know if there is a car in front *and* whether or not it is getting closer. For the case of continuous context variables, it appears that using more basis functions has a much greater marginal effect than optimizing the choice of centers, but it is possible that this would change in higher dimensional spaces.

The difficulty in identifying both the appropriate splitting variables, as well as the regions used in the splits suggests that more automated ways of searching for good regions is important. Our gradient optimizer is an example of such an approach, but other techniques are possible. This problem can be viewed as a sort of dynamical clustering problem, which means that many different machine learning techniques could be applied to it. For example, dynamical versions of k-means clustering, decision tree clustering, or spectral clustering could all be considered.

We conclude this section with a summary of the different algorithms. Figure 6.11 shows a summary of three different algorithms trained and tested on the traffic data: a 3rd order autoregressive model (AR), the linear PLG, and the best variant of the MPLG (32 optimized basis functions). We see that the MPLG shows the best results, with the most pronounced difference being the increased likelihood.

6.6 Conclusions and Future Work

We set out in this chapter to improve the KPLG by proposing a generative probabilistic model, which we can interpret as a mixture of PLGs. We have also contributed a general hybrid particle-analytical inference method, which appears to be accurate and which makes our model tractable. It improves sigma-point approximations in general, and could find application in other contexts.

The MPLG has experimentally demonstrated good, stable performance on almost all of the problems tested here. On the timeseries problems of Section 5.3, the MPLG outperformed the KPLG in every test, and often outperformed the KAR algorithm. On the traffic domain, the MPLG outperformed both the PLG and an autoregressive model by using context variables to define linear regions.

In addition to empirical stability of parameter estimates, the MPLG seems to be a more flexible and naturally interpretable model class than the KPLG, and allows us to easily extend the model. We demonstrated this in the context of the traffic modeling problem, where it was easy to incorporate domain knowledge into the creation of mixing weights. The idea of piecewise linear dynamics is simple to grasp, to deal with, and to extend. In contrast, it can often be challenging to design an appropriate kernel which flexibly incorporates domain knowledge and satisfies the Mercer conditions.

Perhaps the most significant drawback to the KPLG and MPLG models is the assumption that the future is Gaussian. While this has worked well for the low-dimensional problems considered here, it seems unlikely that the Gaussian will be an appropriate density estimate of the distribution of future observations in the case of high-dimensional and/or highly structured observations. In those cases, a more appropriate density estimate needs to exploit structure. In other words, a reasonable next step would be to replace the Gaussian with some sort of graphical model.

In the next section we do exactly that by generalizing the Gaussian to a generic exponential family distribution, to create the Exponential Family PSR.

Chapter 7

Exponential Family Predictive Representations of State

Chapters 4 - 6 presented the PLG family of models, which all assume that $p(F^n|h_t)$ is Gaussian and represent state as the parameters of that Gaussian. However, it is generally accepted that Gaussians and mixtures of Gaussians are poor models in high-dimensional spaces (Hinton, 2002; Aggarwal et al., 2001). This has motivated research into graphical models which are able to exploit structure in high-dimensional spaces.

We now present the Exponential Family PSR (or EFPSR). Like the PLG family, the EFPSR model represents state as the parameters of the distribution $p(F^n|h_t)$, but it models $p(F^n|h_t)$ using a general exponential family distribution. Also like the PLG family of algorithms, the EFPSR model updates state with an extend-and-condition algorithm. The model is more general than either the PLG family or PSRs: because of the flexible nature of the exponential family of distributions, the EFPSR is capable of modeling domains with discrete or continuous observations (or a mixture of both), and the extend-and-condition mechanism turns out to be quite general. As a consequence, we show in Section 7.2 that the EFPSR can represent any system which can be modeled by a PSR, a PLG, or a KPLG/MPLG. From that perspective, the EFPSR is an important unification of the work in the rest of this thesis.

Like other models in this thesis, the EFPSR has no hidden variables, which sets it apart from other graphical models of sequential data. It is not directly comparable in terms of state representation to latent-variable models such as HMMs, CRFs (Lafferty et al., 2001), or Maximum-entropy Markov Models (MEMMs) (McCallum et al., 2000), for example. In particular, EM-based procedures used in the latent-variable models for parameter learning are unnecessary, and indeed, impossible. This is a consequence of the fact that the statistics of interest are always related directly to observable quantities.

This chapter presents the EFPSR in its most general form. It lays the groundwork for two different specializations of the general model: in Chapter 8 we specialize the general EFPSR to create the Information PLG, and in Chapters 9 and 10 we specialize it to create the "Linear-Linear EFPSR," which is a model designed to function with high dimensional distributions and large data sets.

7.1 The General EFPSR Model

We define an EFPSR model as a model with the following two properties: 1) it represents state as the natural parameters of an exponential family distribution over $F^n|h_t$; and 2) it maintains state by extending and conditioning. We now discuss each aspect in detail, but would like to point out that this is a very general definition. In order to create a specific model in EFPSR form, three things need to be selected: the features used by the exponential family distribution, the extension function, and the conditioning mechanism.

7.1.1 State Representation

The EFPSR defines state as the parameters of an exponential family distribution modeling $p(F^n|h_t)$. To emphasize that these parameters represent state, we will refer to them as $s_t|h_t$, or simply s_t (instead of λ_t):

$$p(F^n = f^n|h_t; s_t) = \exp\Big\{s_t^\top \phi(f^n) - \log Z(s_t)\Big\}, \tag{7.1}$$

where $\phi(f^n)$ and s_t are both $\in \mathbb{R}^{l\times 1}$. Recall that at each timestep, $F^n|h_t$ is the random variable representing the next n observations, given history until time t: $F^n|h_t = [O_{t+1} \cdots O_{t+n}|h_t]$, where each $O_t \in \mathbb{R}^d$; thus, each $F^n \in \mathbb{R}^{nd}$.

In Appendix D, we present background on the exponential family of distributions, as well as the reasons motivating their use in the EFPSR. We recap them here: 1) the exponential family is the natural generalization of the Gaussian used by the PLG family of algorithms, and is expected to be a more accurate model of high-dimensional densities; 2) it is the maximum entropy distribution subject to empirically determined constraints (and is therefore a reasonably principled way to select a distribution when learning from data); 3) it is the maximum likelihood distribution under reasonable assumptions; and 4) it is capable of exploiting graphical structure by selecting the sufficient statistics of the distribution carefully, as discussed in Section D.3.

Representing state in this way implies that the EFPSR inherits both the advantages and disadvantages of graphical exponential family models: it is possible to describe conditional independencies that exist between variables, but inference and parameter learning in the model is generally hard. Fortunately, all existing research on exponential family distributions is applicable, and in particular, work on approximate inference.

7.1.2 Maintaining State

Selecting the form of $p(F^n|h_t)$ is the density estimation component of the model. However, there is also a dynamical component: given the parameters of $p(F^n|h_t)$, how can we incorporate a new observation to find the parameters of $p(F^n|h_t, o_{t+1})$? Our strategy is to extend and condition, in exactly the same way as the PLG family of algorithms.

We assume that we have the parameters of $p(F^n|h_t)$, which we denote s_t. We extend the distri-

bution of $F^n|h_t$ to include O_{t+n+1}, which forms a new variable $F^{n+1}|h_t$, and we assume it has the distribution $p(F^n, O_{t+n+1}|h_t) = p(F^{n+1}|h_t)$. This is a temporary distribution with $(n+1)d$ random variables.

In order to add the new variable O_{t+n+1}, we must add new features which describe O_{t+n+1} and its relationship to F^n, which we capture with a new feature vector $\phi^+(f^{n+1})$. We define the vector s_t^+ to be the parameters associated with this feature vector. Both $\phi^+(f^{n+1})$ and s_t^+ are vectors $\in \mathbb{R}^{k\times 1}$. In general, when we add new features, the parameters associated with the original features may change in order to retain the same properties of the distribution (for example, the parameters may need to be adjusted to ensure that the marginals defined by $p(F^n|h_t)$ are the same as the corresponding marginals defined by $p(F^{n+1}|h_t)$). We will refer to the function which maps the current state vector to the parameters of the extended distribution as *the extension function*:

$$s_t^+ = \text{extend}(s_t; \theta)$$

where θ is a vector of parameters governing the extension function (and hence, the transition dynamics). Putting this all together, we arrive at the following form for the extended distribution:

$$p(F^{n+1} = f^{n+1}|h_t; s_t^+) = \exp\Big\{s_t^{+\top}\phi^+(f^{n+}) - \log Z(s_t^+)\Big\}. \tag{7.2}$$

Once we have extended the distribution to model the $n+1$'st observation in the future, we then condition on the *actual* observation o_{t+1}, which results in the parameters of a distribution over observations from $t+1$ through $t+n+1$:

$$s_{t+1} = \text{condition}(s_t^+, o_{t+1})$$

which are precisely the statistics representing $p(F^n|h_{t+1})$, which is our state at time $t+1$. Using this method, we can maintain state for arbitrarily long periods, extending and conditioning for every new ao.

Although the sequence of state vectors s_t are the parameters defining the distribution $p(F^n|h_t)$, they are **not** the model parameters – that is, we cannot freely select them. Instead, the model parameters are the parameters θ which govern the extension function. This is a significant difference from standard maximum entropy models, and stems from the fact that our overall problem is that of modeling a dynamical system, rather than just density estimation.

There is only one restriction on the extension function: we must ensure that after extending and conditioning the distribution, the resulting distribution can be expressed as:

$$p(F^n = f^n|h_{t+1}; s_{t+1}) = \exp\Big\{s_{t+1}^{\top}\phi(f^n) - \log Z(s_{t+1})\Big\}. \tag{7.3}$$

This looks like exactly like Eq. 7.1, which is the point: the feature vector ϕ did not change between

timesteps. From a graphical model perspective, this is equivalent to saying that the structure of the graph does not change between state updates.

For many choices of extension function and conditioning function, the overall extend-and-condition operation does not involve any inference. Thus, given an EFPSR model, tracking state can be computationally efficient, and could be (for example) a simple linear operation.

Defining the state representation and state update mechanism completes the definition of the general EFPSR model. We now examine what dynamical systems can be captured by models in this class.

7.2 Representational Capacity

When introducing a new model class, it is natural to wonder about how it relates to other well-known models. In this section, we investigate which other classes of dynamical systems the EFPSR can capture. We show that 1) every domain that can be modeled by a (linear or nonlinear) PSR with a finite number of core tests can be modeled by an EFPSR; 2) every uncontrolled linear dynamical system with scalar observations can be modeled by an EFPSR; and 3) some nonlinear dynamical systems can be modeled by an EFPSR (specifically, those captured by the KPLG/MPLG). The first claim implies that EFPSRs can also model every finite-state MDP, finite-state POMDP, finite-state Hidden Markov Model, finite-state Markov chain, history-window (k-th order Markov) model, diversity representation, interpretable OOM, or interpretable IO-OOM (see Chapter 2).

The EFPSR requires three things in order to be a complete model: features, an extension function, and a conditioning mechanism. For the proofs, we will present all three. In all of our proofs, the features and conditioning mechanisms are straightforward, although the extension functions are not necessarily practical. Efficiency is not the point of this section: to explore representational capacity, we simply need to demonstrate that *some* suitable extension function exists.

7.2.1 EFPSRs and PSRs

To prove that every PSR can be represented by an EFPSR, we present a constructive algorithm. The goal of the proof is to show that given a PSR, we can construct an EFPSR which makes equivalent predictions about the distribution of one-step future observations, and which can be updated such that this equivalence holds for the infinitely long future. We will describe the construction of the EFPSR in Section 7.2.1 and then present the theorem and final proof in Section 7.2.1.

Constructing an EFPSR from a PSR

To define an EFPSR, we must define features, an extension function, and a conditioning mechanism. We assume that we are given a fully specified PSR with two things:

1. Assume we have a set of core tests Q, and that the longest core test has length n. This set of core tests does not have to be minimal.

2. Assume we have two functions which are used for a state update. These two functions are a slight generalization of the Linear PSRs described in Section 2.3, and are used so that the proof accounts for both linear and nonlinear PSRs. To update state, PSRs require some function to compute the probabilities of one-step tests and one-step extensions as a function of the current state:

$$\begin{aligned} p(aoQ|h_t) &= g_{aoQ}(p(Q|h_t)) \\ p(ao|h_t) &= g_{ao}(p(Q|h_t)). \end{aligned}$$

In the case of linear PSRs, these functions are matrix multiplications, as shown in Eq. 2.1. Recall from Section 2.3 that these two functions are sufficient to make any prediction about the future by rolling the model forward.

We now construct the EFPSR by specifying features, and extension and conditioning functions.

- **Features**: We first describe how we can use tests as features. Given a window into the future of length n, the variable F^n is composed of the $2n$ atomic random variables $A_1 O_1 \cdots A_n O_n$. Now let q_i be a length k test, with $k \leq n$. We can think of this as a feature of the future as

$$\phi(F^n)_i = \delta(F^k, q_i).$$

In other words, feature i is binary indicator variable, returning 1 if the first k actions and observations in F^n are equal to test q_i, and zero otherwise.

The state at time t for a PSR is given by $p(Q|h_t)$, where the ith entry of the state vector is the prediction of core test q_i, or $p(q_i|h_t)$. Because the probability of a binary variable is also its expected valued, we can reinterpret $p(Q|h_t)$ as a vector of expectations:

$$p(Q|h_t) = \begin{bmatrix} \mathrm{E}[q_1|h_t] \\ \mathrm{E}[q_2|h_t] \\ \cdots \\ \mathrm{E}[q_{|Q|}|h_t] \end{bmatrix} = \mathrm{E}\left[\phi(F^n|h_t)\right].$$

According to this interpretation, this is a vector of mean parameters. This fact will be used in the extension function.

- **State at time** t: State at time t is the parameters of the exponential family distribution modeling $p(F^n|h_t)$. The features that we use are the core tests Q of the given PSR, as well as all possible one-step tests:

$$p(F^n|h_t) = \exp\left\{\sum_l (\lambda_t)_l q_l + \sum_{mn} (\lambda_t)_{mn} a_m o_n - \log Z^n(\lambda_t)\right\}. \tag{7.4}$$

Note that the domain used to compute the log partition function is $F^n|h_t$, which includes the next n actions and observations. We have made this explicit by naming the partition function Z^n. To foreshadow things a bit, we state now that we will select the parameters of this distribution such that its marginals match the predictions of the core tests $p(Q|h_t)$.

- **The extension function:** To define the extension function, we need to define a mapping from the state vector s_t to the vector s_t^+, which are the parameters describing the extended distribution $p(F^{n+1}|h_t)$.

  Given an EFPSR state s_t, we use inference to compute the corresponding vector of mean parameters, which we denote μ_t. By construction, this vector contains expectations which are the predictions of the core tests Q. We extract these to form the vector $p(Q|h_t)$. We then compute the one-step extensions to all of the core tests, where each core test is extended by each possible action and observation $a_i o_j$. We accomplish this by using the mappings provided as part of the given PSR: $p(a_i o_j Q|h_t) = g_{a_i o_j Q}(p(Q|h_t))$. Note that the longest core test is now length $n+1$. We also compute the predictions of all one-step tests $p(a_i o_j|h_t)$ and two-step tests $p(a_i o_j a_k o_l|h_t)$.

  We now form a new vector of mean parameters, consisting of the expectations for one-step tests, two-step tests, and one-step extended core tests. We can interpret these new probabilities as a mean parameterization μ_t^+ over $F^{n+1}|h_t$, and is a vector which is realizable by construction. We now translate from mean parameters back to natural parameters, to create λ_t^+. If the features defined by the core tests Q are linearly dependent, we may pick any λ_t^+ in the appropriate affine subspace of the image.

  Note that both the mapping from s_t to μ_t, and then from μ_t^+ back to λ_t^+ are always possible by the mean-value mapping theorems of Wainwright and Jordan (2003).

  The vector λ_t^+ now represents the natural parameters of a distribution over $F^{n+1}|h_t$, with features defined by one-step tests, two-step tests and one-step extensions to core tests:

$$p(F^{n+1}|h_t) = \exp\left\{\sum_{ijl}(\lambda_t^+)_l a_i o_j q_l + \sum_{mn}(\lambda_t^+)_{mn} a_m o_n + \sum_{ijkl}(\lambda_t^+)_{ijkl} a_i o_j a_k o_l - \log Z^{n+1}(\lambda_t^+)\right\}$$

  Note that the domain used to compute the log partition function is $F^{n+1}|h_t$, which includes the next $n+1$ actions and observations. We have made this explicit by naming the partition function Z^{n+1}.

- **Conditioning:** We now condition on the given action and observation $a_m o_n$. To do this, we simply freeze $a_m o_n$ to its given value, which is 1 (because they are binary indicator variables). Every term that involves an action and observation which is not $a_m o_n$ drops out, because they

are observed to have a value of 0. This results in the following distribution:

$$\begin{aligned} p(F^n|h_t ao) &= \exp\left\{\sum_l (\lambda_t^+)_l a_m o_n q_l + (\lambda_t^+)_{mn} a_m o_n \right. \\ &\qquad \left. + \sum_{kl} (\lambda_t^+)_{ijkl} a_m o_n a_k o_l - \log Z^n(\lambda_t^+)\right\} \\ &= \exp\left\{\sum_l (\lambda_t^+)_l q_l + (\lambda_t^+)_{mn} + \sum_{kl} (\lambda_t^+)_{ijkl} a_k o_l - \log Z^n(\lambda_t^+)\right\} \\ &= \exp\left\{\sum_l (\lambda_t^+)_l q_l + \sum_{kl} (\lambda_t^+)_{ijkl} a_k o_l - \log Z^n(\lambda_t^+)\right\} \end{aligned}$$

There are several effects that conditioning has had. Notice that this distribution specifies features for each of the core tests q_l. It also specifies features for all of the one-step tests $a_k o_l$ – in other words, the conditioned distribution has exactly the same form as Eq. 7.4. The term $(\lambda_t^+)_{mn}$ cancels with a similar term in the normalizing constant. In addition, the domain used to compute the normalizing constant is now $F^n|h_t ao$, which only includes the n actions and observations following $h_t ao$.

This completes the construction of an EFPSR given a PSR. The idea is that we use core tests as features, and we set up the extended and conditioned distributions such that they have appropriate marginals at every point. This will be the key to our theorem statement, which we present next.

Theorem Statement

Theorem 7.2.1. *For every PSR (linear or nonlinear) with a finite number of core tests, an EFPSR can be constructed such that the EFPSR makes equivalent one-step predictions to the PSR at every timestep.*

Proof. We prove this by induction. For the base case, we assume that we have a PSR state $p(Q|h_t)$, and that we have an EFPSR state s_t modeling $p(F^n|h_t)$. We assume that the marginals of $p(F^n|h_t)$ are equivalent to $p(Q|h_t)$, and that the marginals corresponding to the one-step features are equal to the prediction $p(ao|h_t)$ that the PSR would make. We call these *equivalent states*, in the sense that both states can be used to make equivalent predictions for both core tests and one-step tests. Such an EFPSR state always exists by virtue of the mean-parameter mapping theorems of Wainwright and Jordan (2003).

The proof follows directly by the equivalences established in the constructive algorithm:

- We have shown that an EFPSR state which is equivalent to a PSR state may be extended to form a distribution with marginals that are equal to the PSR predictions of the one-step tests

$p(ao|h_t)$, two-step tests $p(aoao|h_t)$ and the one-step extensions to core tests $p(aoQ|h_t)$.

- We have shown that this EFPSR may be conditioned, and that the form of distribution is a distribution over core tests $p(Q|h_tao)$ and one-step tests $p(ao|h_tao)$.
- By construction, the conditioned distribution must therefore have marginals which are equal to $p(Q|h_tao) = p(aoQ|h_t)/p(ao|h_t)$, which is exactly the distribution we would have obtained with the PSR update. Because we have explicitly constrained both the one-step and two-step predictions to have the same values as the corresponding PSR predictions, it must also have marginals equal to $p(ao|h_tao) = p(aoao|h_t)/p(ao|h_t)$.

The final state has the same properties as the state we began with: it makes equivalent predictions for one-step tests and core tests as the corresponding PSR, but from h_{t+1} instead of h_t. We may repeat the process infinitely long, and thus we conclude that the complete distributions over future observations that the two models compute are identical. □

Action-Conditional Distributions

There are a few minor points worth noting at this point. First, there is a subtle technicality to the way we structured the proof. Recall that the prediction of a test in a PSR is actually action-conditional:

$$p(q_i|h_t) \equiv \Pr(\text{observations in test } |h_t, \text{ actions in test}).$$

However, the EFPSR does not deal with conditional distributions like this. When the PSR talks about the distribution of a one-step test $p(ao|h_t)$, no distribution over the action is implied, but in the EFPSR, $p(ao|h_t)$ implies a distribution over both the action and observation.

This is not a problem for the proof, which is why we have delayed discussing it until now. To get around this, we can impose a distribution upon the actions, which is independent of history and observations: $p(a|h_t) \equiv p(a)$. This distribution is for convenience only, and does not represent any statements about possible policies, nor does it affect the ability of the model to make PSR-style predictions. For example, the EFPSR will capture a one-step test ao as $\Pr(ao|h_t)$, but can still make the PSR-style prediction $\Pr(o|h_t, a)$ by computing two marginals: $\Pr(o|h_t, a) = \Pr(ao|h_t)/\Pr(\mathrm{a}|\mathrm{h_t}) = \Pr(ao|h_t)/\Pr(\mathrm{a})$. Thus, this distribution over actions will drop out anytime we make a prediction, and effectively serves only to weight predictions such that they can be considered entries in a very large multinomial distribution. To simplify the proof flow, we did not mention this; instead, it should be understood that the appropriate conditioning happens whenever it needs to.

Different Predictions

While the EFPSR and the PSR make the same predictions for all one-step tests, the state used by the EFPSR is larger than the state for the PSR, because it includes a parameter for every possible

one-step test. The corresponding PSR may or may not include every one-step test in the core set.

This was necessary to ensure that the models make equivalent one-step predictions, and highlights the following interesting fact about the final EFPSR. Specifically, the state at time t of the EFPSR can be used to make any prediction about the the next n observations, without rolling the state forward. For core tests and one-step tests, these will be equal to the PSR predictions, by construction. However, the EFPSR can also make predictions about non-core tests, and in general, these predictions may be different than the corresponding PSR predictions, because they are the maximum-entropy predictions. However, it is able to make those predictions *without* any additional parameters (a linear PSR would need to learn or compute the appropriate m_t weight vector in order to predict a non-core test) and without referencing the functions g_{ao} or g_{aoQ}. It is interesting that there are two different ways to use the state to compute predictions, and that they will give different results.

It would have been possible to not include all possible one-step tests in the state representation. In that case, we could have claimed that the EFPSR state was sufficient for history and just as compact as the corresponding PSR state. However, it might have given different results for one-step predictions than the corresponding PSR, depending on how the state was used to make the prediction.

7.2.2 EFPSRs and (Non)Linear Dynamical Systems

We now present the relationships between EFPSRs and (non)linear dynamical systems. We will proceed in the same vein as the PSR proof, relying on the mapping between mean and natural parameters and constructing extension functions based on reductions to other models. We will again present an inductive argument, but will spend less time setting up the base case equivalences.

Theorem 7.2.2. *EFPSRs are capable of modeling every uncontrolled linear dynamical system with a scalar observation (as defined in Section 4.1). Furthermore, the model is just as compact as the equivalent LDS.*

Proof. The proof follows directly from two theorems:

1. In Chapter 8, we prove that every uncontrolled PLG can modeled by an EFPSR (Theorem 8.3.1), which we call the "Information PLG."
2. Theorem 1 in Rudary et al. (2005) proves that every uncontrolled n-dimensional LDS with a scalar observation has an equivalent representation as a PLG.

Based on these two theorems, we conclude that EFPSRs are capable of modeling every uncontrolled linear dynamical system with scalar observations. Because both proofs are quite involved, we defer (1) to Chapter 8, and (2) to Rudary et al. (2005). In addition, Chapter 8 shows that the Information

PLG and the PLG are equally compact – both use an n-dimensional state vector. Rudary showed that the PLG requires an n-dimensional state vector to model an n-dimensional LDS, so we conclude that the state vector of the Information PLG is just as compact as the equivalent LDS. In addition, the Information PLG uses exactly the same parameters as the PLG, and the PLG is at least as compact parameter-wise as the equivalent LDS. □

To cover the case of linear dynamical systems with vector-valued observations, we would need to prove that PLGs are capable of capturing every LDS with vector-valued observations. Rudary has done some (unpublished) work to this effect. We defer our proof of this, as well as proofs about the controlled case, until his work is complete, although it is likely that the proofs will be straightforward.

Theorem 7.2.3. *EFPSRs are capable of modeling any dynamical system modeled by the KPLG and MPLG. Furthermore, the model is just as compact as the equivalent KPLG or MPLG.*

Proof. As part of the proof that PLGs have an EFPSR representation, Eq. 8.1 demonstrates that there is a one-to-one correspondence between mean and natural parameters in the Gaussian used by the PLG. Additionally, it shows that the *extended* distribution will always have equivalent mean and natural parameters.

The proof is a combination of those facts and a simple constructive algorithm. Given a KPLG or MPLG, we will build an EFPSR model by defining features, an extension function, and a conditioning mechanism:

1. Let the features be all singleton and pairwise features of $F^n|h_t$, as in the Information PLG.
2. At every timestep, the KPLG/MPLG defines state as the parameters of a Gaussian distribution over $p(F^n|h_t)$. Let the EFPSR state be the information form of that Gaussian.
3. Let the extension function be the following:
   (a) Assume we are given an EFPSR state s_t.
   (b) This state is the natural parameters of the distribution $p(F^n|h_t) \sim \mathcal{N}^{-1}(\lambda_{\mu_t}, \lambda_{\Sigma_t})$
   (c) Translate these natural parameters to their equivalent mean parameters to find $p(F^n|h_t) \sim \mathcal{N}(\mu_t, \Sigma_t)$.
   (d) Use the KPLG/MPLG extension function to compute E_t, C_t and V_t.
   (e) Construct an extended Gaussian describing $p(F^{n+1}|h_t) \sim \mathcal{N}(\mu_t^+, \Sigma_t^+)$ using the mean parameters:

$$\begin{pmatrix} F^n \\ O_{t+n+1} \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \mu_t \\ E_t \end{pmatrix}, \begin{pmatrix} \Sigma_t & C_t \\ C_t^\top & V_t \end{pmatrix}\right].$$

(f) Translate $p(F^{n+1}|h_t) \sim \mathcal{N}(\mu_t^+, \Sigma_t^+)$ back to information form. This is now the distribution $p(F^{n+1}|h_t) \sim \mathcal{N}^{-1}(\lambda_{\mu_t^+}, \lambda_{\Sigma_t^+})$, which is the extended distribution in information form, and the parameters of this distribution represent s_t^+.

4. Condition on the observation in exactly the same way as the Information PLG.

The compactness arguments follow the same pattern as those for the PLG, *mutatis mutandi*. □

7.2.3 EFPSRs and MDPs

We now turn to a proof that EFPSRs can model finite-state Markov Decision Processes, or MDPs. From the point of view of investigating the representational capacity of the EFPSR, this proof is unnecessary, since every finite-state MDP can be modeled with a PSR. However, we wish to make an additional point in this section, which is that the extension function is a strictly linear function of state. This is part of the justification for the Linear-Linear EFPSR in Chapter 9.

An MDP is described by a tuple $\langle \mathcal{X}, \mathcal{A}, T \rangle$, where $\mathcal{X}$ is a finite set of states and $\mathcal{A}$ is a finite set of actions. T is a set of transition matrices, where T^a is a matrix which represents the probability of transitioning from state x_i to state x_j given that action a was taken: $T^a_{ij} = p(x_i|x_j, a)$.

We prepare for the proof by associating a binary random variable x_t^i with each state $\in \mathcal{X}$ and for each time t (only one of these binary variables will be 1 at each time t). We also associate a binary random variable a_t^i with each action $\in \mathcal{A}$.

Theorem 7.2.4. *The EFPSR can model every finite-state MDP with discrete actions. Furthermore, the model uses a strictly linear extension function.*

Proof. The proof is by construction, and largely follows the same pattern as the PSR proof, with several simplifications. However, to arrive at the desired linearity result, it is necessary to stipulate an unchanging distribution over actions, as we now explain.

We will represent state as a multinomial distribution over MDP states one step in the future, and we use $F^1|h_t$ to denote the random variable of the state one step in the future. To give this variable a well-defined distribution, we must have some distribution over actions $p(a)$ (as in the PSR proof, this distribution does not impact the capacity of the model, and drops out when predictions are made). We will additionally impose the restriction that this distribution is independent of state and history. At time t, we assume that we are in a known MDP state x_t. Then:

$$p(F^1|h_t) = p(ax_{t+1}|h_t) = p(a)p(x_{t+1}|a, h_t) = p(a)T^a x_t \equiv \mathrm{E}[\phi(F^1|h_t)].$$

We let our EFPSR state be the natural parameters of this distribution (compare to Eq. D.6):

$$p(F^1|h_t; s_t) = \exp\left\{\sum_{ij}(s_t)_{ij}a^i_{t+1}x^j_{t+1} - \log Z(s_t)\right\}.$$

We define the extended distribution over $F^2|h_t$ as

$$p(F^2|h_t; s^+_t, s_t) = \exp\left\{\sum_{ij}(s_t)_{ij}a^i_{t+1}x^j_{t+1} + \sum_{ijkl}(s^+_t)_{ijkl}a^i_{t+1}x^{t+1}_j a^k_{t+1}x^l_{t+2} - \log Z([s_t; s^+_t])\right\}$$

where we have included quartets consisting of an action, observation, action and observation. Notice that the multipliers associated with the pairwise features $a^i_{t+1}x^j_{t+1}$ are the entries of s_t, not s^+_t. This is because we use a special extension function, defined as:

$$s^+_t = \begin{bmatrix} I \\ 0 \end{bmatrix} s_t + \begin{bmatrix} 0 \\ B \end{bmatrix}$$

That is, the extension leaves every element of the current state vector unchanged, and adds new multipliers for each quartet, in a way that does not depend on the current state (B is a constant). The entries of B are given as

$$B_{ijkl} = \log p(a_i)T^k_{jl}$$

which are just the logs of the transition probabilities, as defined in Eq. D.7.

The observation at time $t+1$ is x^j_{t+1}, and the action is a^i_{t+1} After conditioning, every term with an action and observation at time $t+1$ which is not equal to a^i_{t+1} and x^j_{t+1} drops out (since they are binary indicator variables, and are set to zero since they did not occur). This results in the final distribution over $F^1|h_t ao$:

$$\begin{aligned} p(F^1|h_t, a^{t+1}x^{t+1}; s^+_t, s_t) &= \exp\left\{(s_t)_{ij} + \sum_{kl}(s^+_t)_{ijkl}a^k_{t+2}x^l_{t+2} - \log Z([s_t; s^+_t])\right\} \\ &= \exp\left\{\sum_{kl}(s_{t+1})_{kl}a^k_{t+2}x^l_{t+2} - \log Z(s_{t+1})\right\} \end{aligned}$$

where the constant term $(s_t)_{ij}$ cancels with the same term in the normalizer.

To complete the proof, note that this is the natural parameterization of a multinomial distribution over $p(F^1|h_t, a^{t+1}x^{t+1}$, and furthermore, note that $(s_{t+1})_{kl}a^k_{t+2}x^l_{t+2} = (s^+_t)_{ijkl}a^k_{t+1}x^l_{t+1} = (s^+_t)_{ijkl}$, which is is equal to $B_{ijkl} = \log p(a_i)T^k_{jl}$. This is exactly the same multinomial distribution we would have obtained if we had rolled the MDP forward through $T^{a^i_{t+1}x^j_{t+1}}$. □

7.3 Conclusions

In this chapter, we have introduced the Exponential Family PSR (EFPSR), which generalizes the state representation used by the PLG family of algorithms. It replaces the Gaussian with a more general exponential family distribution, although it uses the same extend-and-condition mechanism to capture the dynamics.

The EFPSR is a very general framework for modeling dynamical systems. We have briefly investigated its representational capacity, and shown that it is capable of capturing a variety of different dynamical systems: the EFPSR can model domains captured by PSRs, POMDPs, linear dynamical systems, and some nonlinear dynamical systems. From this perspective, the EFPSR is a generalization of many of the models presented in earlier chapters of this thesis. This is largely due to the flexibility of the exponential family of distributions, as well as the genericity of the state update mechanism.

In the next chapters, we will specialize the EFPSR to create two different models. Chapter 8 presents the Information PLG, which is the PLG rewritten with a different form of the state representation and update. Chapters 9 and 10 present the Linear-Linear EFPSR, which is a different specialization designed to cope with domains having large numbers of features and large amounts of training data.

Chapter 8

The Information Predictive Linear-Gaussian Model

In this chapter, we connect the Predictive Linear-Gaussian model of Chapter 4 and the EFPSR model of Chapter 7 together by relating both of them to the *Information Kalman Filter.* Recall that the PLG is the predictively defined version of the Kalman filter, and that results by Rudary et al. (2005) prove that every Kalman filter has an equivalent form in terms of the PLG. The difference is that the Kalman filter maintains state as the parameters of a Gaussian distribution over hidden states, while the PLG maintains state as the parameters of a Gaussian distribution over future observations.

The parameterization of the Gaussian is not unique. Like the other members of the exponential family of distributions discussed in Section D.2, the Gaussian can be represented with either mean parameters or natural parameters. The standard Kalman filter and the PLG both use the mean parameterization of their respective Gaussian distributions. In filter theory, it was proposed as early as 1979 that the Kalman filter could be represented using natural parameters instead of mean parameters (Maybeck, 1979). The resulting filter became known as the *Information Kalman Filter* because the parameterization relies on the inverse of the covariance matrix, which is sometimes called the *information matrix* due to its straightforward interpretation as a Fisher information matrix.

It is therefore intriguing to ask: is it possible to write the PLG in information form as well? This chapter answers that question affirmatively. We show how we can transform the parameterization of the PLG into an equivalent natural form, and how state can be recursively updated for approximately the same computational cost as the PLG.

Importantly for our purposes, the final model fits the EFPSR framework. That is, given appropriate choices of features, the state representation used by the Information PLG is in EFPSR form, and there exist choices of extension and conditioning functions which map onto the extend-and-condition method used by the EFPSR. For these choices, then, the EFPSR *is* the information form of the PLG. These results are shown graphically in Figure 8.1. There may some advantages to thinking about the Information PLG as an EFPSR model, beyond the theoretical elegance of reducing one model to another. For example, the EFPSR is capable of imposing graphical structure on the network of variables used. This means that it is likely that the EFPSR could be generalized to cope with a Gaussian Markov Random field instead of a simple Gaussian.

The equivalence of the PLG and the Information PLG is somewhat unsurprising, and so is the

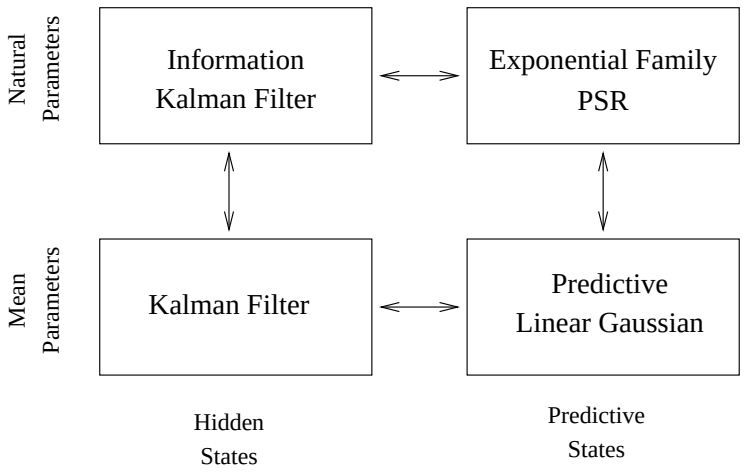


Figure 8.1: The relationship between the Kalman Filter, the Information Kalman Filter, the PLG, and the EFPSR.

road we will take to prove that equivalence. The proof is by construction. We will first show that at every time t, the Gaussian distribution over $p(F^n|h_t)$ defined by the PLG parameters can be transformed into an equivalent natural parameterization. This implies that the representational capacity of the two are equivalent, and that the distribution over observable quantities computed by both are equivalent. We will then show that the state of the Information PLG can be recursively updated – using the same parameters as the PLG – without reference to the PLG state representation, which makes it a standalone model. The proof is rather long, and so the statement of the theorem (Theorem 8.3.1) is delayed until Section 8.3.

From a filtering point of view, there are several potential advantages to working in the information parameterization. The Information Kalman Filter overcomes some problems with initialization, for example, and can have computational advantages in case the state vector is of greater dimension than the observation vector (Manyika and Durrant-Whyte, 1995). One well-known advantage is that multiple measurements can be incorporated by simply summing their information vectors and matrices, which is appealing for distributed sensory networks (Manyika and Durrant-Whyte, 1995). Recently, Thrun et al. (2002) observed that the information matrix used in simultaneous localization and mapping (SLAM) applications has a very sparse form, which has motivated the Sparse Information Filter. This has been motivated more formally by Frese (2005), who proved that information decays exponentially, and by Eustice (2005), who showed that significant computational benefits can be achieved with exactly sparse delayed information filters. This can also perhaps be justified by the results of Boyen and Koller (1998), who also proposed trimming weak information links.

It is possible that these advantages will transfer to the Information PLG as well, although quantifying them is left for future research.

8.1 The Information Parameterization of the Gaussian

As discussed previously, while the Kalman filter maintains state using the mean parameterization, the Information Kalman Filter represents state with the natural parameters. The information parameterization can be obtained by expanding the quadratic in the exponential and regrouping terms:

$$\begin{aligned} p(x) &= \mathcal{N}(\mu, \Sigma) \\ &= \frac{1}{Z} \exp\left\{-0.5(x-\mu)^\top \Sigma^{-1}(x-\mu)\right\} \\ &= \exp\left\{-0.5(x^\top \Sigma^{-1} x - 2\mu^\top \Sigma^{-1} x + \mu^\top \Sigma^{-1}\mu) - \log Z\right\} \\ &= \exp\left\{\sum_{ij} \Sigma^{-1}_{ij} x_i x_j + \sum_i (\mu^\top \Sigma^{-1})_i x_i - \log Z'\right\} \\ &\equiv \mathcal{N}^{-1}(\lambda_\mu, \lambda_\Sigma). \end{aligned} \tag{8.1}$$

where $Z' = \exp\{-0.5\mu^\top \Sigma^{-1}\mu Z$. The two forms are related by

$$\begin{aligned} \lambda_\mu &= \Sigma^{-1}\mu \\ \lambda_\Sigma &= -\frac{1}{2}\Sigma^{-1}. \end{aligned}$$

To relate this more directly to the standard exponential family form discussed previously, we observe that Eq. 8.1 can be written using vector notation:

$$\begin{aligned} p(x) &= \exp\left\{\sum_{ij} \Sigma^{-1}_{ij} x_i x_j + \sum_i (\mu^\top \Sigma^{-1})_i x_i - \log Z'\right\} \\ &= \exp\left\{\lambda^\top \phi(x) - \log Z'\right\} \end{aligned} \tag{8.2}$$

where $\phi(x) = \{x_i\} \cup \{x_i x_j\}$ – that is, the feature vector $\phi(x)$ contains all singleton features as well as all pairwise features. Eq. 8.2 is exactly the form that the EFPSR distribution takes, except that instead of representing the distribution over hidden states X, it represents the distribution over future observations F^n.

Computationally, the covariance form and the information form have complementary strengths and weaknesses, which are summarized in Figure 8.2. The operation of marginalization in the covariance form, for example, simply involves selecting a subset of the mean vector and the covariance matrix, but the operation of conditioning is more involved, involving a matrix inverse. Conversely, in the information form, conditioning is easy, but marginalization is hard.

8.2 Deriving the Information PLG

We now show how every PLG can be written in information form, and derive the state update equations necessary. We begin by reviewing the basics of the PLG.

$$p(\alpha,\beta) = \mathcal{N}(\begin{bmatrix} \mu_\alpha \\ \mu_\beta \end{bmatrix}, \begin{bmatrix} \Sigma_{\alpha\alpha} & \Sigma_{\alpha\beta} \\ \Sigma_{\beta\alpha} & \Sigma_{\beta\beta} \end{bmatrix}) = \mathcal{N}^{-1}(\begin{bmatrix} \eta_\alpha \\ \eta_\beta \end{bmatrix}, \begin{bmatrix} \Lambda_{\alpha\alpha} & \Lambda_{\alpha\beta} \\ \Lambda_{\beta\alpha} & \Lambda_{\beta\beta} \end{bmatrix})$$

	Marginalization $p(\alpha) = \int p(\alpha,\beta)d\beta$	Conditioning $p(\alpha\|\beta) = p(\alpha,\beta)/p(\beta)$
Covariance Form	$\mu = \mu_\alpha$ $\Sigma = \Sigma_{\alpha\alpha}$	$\mu' = \mu_\alpha + \Sigma_{\alpha\beta}\Sigma_{\beta\beta}^{-1}(\beta - \mu_\beta)$ $\Sigma' = \Sigma_{\alpha\alpha} - \Sigma_{\alpha\beta}\Sigma_{\beta\beta}^{-1}\Sigma_{\beta\alpha}$
Information Form	$\eta = \eta_\alpha - \Lambda_{\alpha\beta}\Lambda_{\beta\beta}^{-1}\eta_\beta$ $\Lambda = \Lambda_{\alpha\alpha} - \Lambda_{\alpha\beta}\Lambda_{\beta\beta}^{-1}\Lambda_{\beta\alpha}$	$\eta' = \eta_\alpha - \Lambda_{\alpha\beta}\beta$ $\Lambda' = \Lambda_{\alpha\alpha}$

Figure 8.2: Information and covariance forms of marginalization and conditioning. A multivariate Gaussian distribution can be parameterized with either mean or natural parameters. Shown are marginalization and conditioning operations on a multivariate Gaussian random variable, expressed in both covariance form (mean parameters) and information form (natural parameters). Adapted from Eustice (2005).

The PLG represents state as the parameters of a Gaussian distribution over $p(F^n|h_t)$:

$$F^n|h_t \sim \mathcal{N}(\mu_t, \Sigma_t).$$

Theorem 8.2.1. *At every time t, the distribution $F^n|h_t \sim \mathcal{N}(\mu_t, \Sigma_t)$ has an equivalent information form $F^n|h_t \sim \mathcal{N}^{-1}(\lambda_{\mu_t}, \lambda_{\Sigma_t})$.*

Proof. The proof follows directly from Eq. 8.1. □

This implies that at every t there exists natural parameters which imply an equivalent distribution over $F^n|h_t$. The rest of our derivation will be devoted to finding those parameters.

We therefore assume that the Information PLG also represents state as the natural parameters of the distribution $p(F^n|h_t)$:

$$F^n|h_t \sim \mathcal{N}^{-1}(\lambda_{\mu_t}, \lambda_{\Sigma_t}).$$

We now turn our attention to the extended joint distribution of $[F^n; O_{t+n+1}]$. Since the extended joint distribution is also a Gaussian, it also has a mean and natural parameterization by Eq. 8.1. We wish to additionally show that the extended joint distribution can be recursively computed without reference to the PLG state, and using only the PLG parameters.

The PLG extends the distribution using a linear trend G and noise term η_{t+n+1}:

$$O_{t+n+1} = GF^n + \eta_{t+n+1} \tag{8.3}$$

Using Eq. 8.3 we can easily compute the terms needed to compute the joint extended distribution in covariance form:

$$\begin{pmatrix} F^n \\ O_{t+n+1} \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \mu_t \\ E_t \end{pmatrix}, \begin{pmatrix} \Sigma_t & C_t \\ C_t^\top & V_t \end{pmatrix}\right].$$

which are

$$\begin{aligned} E_t &= \mathrm{E}[GF^n + \eta_{t+n+1}] = G\mu_t \\ C_t &= \mathrm{E}[O_{t+n+1}^\top F^n] - \mathrm{E}[O_{t+n+1}^\top]\mathrm{E}[F^n] = \Sigma_t G^\top + C_\eta^\top \\ V_t &= \mathrm{E}[O_{t+n+1}^\top O_{t+n+1}] - \mathrm{E}[O_{t+n+1}]^\top \mathrm{E}[O_{t+n+1}] = G^\top \Sigma_t G + GC_\eta^\top + C_\eta G^\top + \sigma_\eta^2. \end{aligned}$$

Recall that G is the linear trend, $C_\eta = \mathrm{Cov}[\eta_{t+n+1}F^n]$, and $\sigma_\eta^2 = \mathrm{Var}[\eta_{t+n+1}]$, which are the parameters of the PLG.

We now compute the information form of the extended distribution:

$$\begin{pmatrix} F^n \\ O_{t+n+1} \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \mu_t \\ E_t \end{pmatrix}, \begin{pmatrix} \Sigma_t & C_t \\ C_t^\top & V_t \end{pmatrix}\right] = \mathcal{N}^{-1}\left[\begin{pmatrix} \lambda_{\mu_t}^+ \\ \lambda_{E_t} \end{pmatrix}, \begin{pmatrix} \lambda_{\Sigma_t}^+ & \lambda_{C_t} \\ \lambda_{C_t}^\top & \lambda_{V_t} \end{pmatrix}\right]$$

Note that, in general, $\lambda_{\Sigma_t}^+ \neq \lambda_{\Sigma_t}$ and $\lambda_{\mu_t}^+ \neq \lambda_{\mu_t}$.

Our goal is to find a recursively updateable expression for the information form on the right-side of this equation, which can be expressed using nothing but the standard PLG dynamical parameters. We begin by noting that the following relationships hold between the mean parameters and the natural parameters:

$$\begin{aligned} \lambda_{\mu_t} &= \Sigma_t^{-1}\mu_t &&= -2\lambda_{\Sigma_t}\mu_t \\ \mu_t &= \Sigma_t\lambda_{\mu_t} &&= -\tfrac{1}{2}\lambda_{\Sigma_t}^{-1}\lambda_{\mu_t} \end{aligned}$$

and

$$\begin{aligned} \lambda_{\Sigma_t} &= -\frac{1}{2}\Sigma_t^{-1} \\ \Sigma_t &= -\frac{1}{2}\lambda_{\Sigma_t}^{-1} \\ \Sigma_t^{-1} &= -2\lambda_{\Sigma_t}. \end{aligned}$$

We will use these identities below as we translate between parameterizations.

8.2.1 Computing the Information Matrix Terms

We will begin by computing the terms in the information matrix. We start with a standard identity on block matrix inversion, which allows us to compute the blocks of a inverted block matrix (Golub and Loan, 1996):

$$\begin{bmatrix} A & B \\ B^\top & C \end{bmatrix}^{-1} = \begin{bmatrix} D & E \\ E^\top & F \end{bmatrix}$$

where

$$\begin{array}{rclcrcl} D & = & (A - BC^{-1}B^\top)^{-1} & = & A^{-1} + A^{-1}B(C - B^\top A^{-1}B)^{-1}B^\top A^{-1} & & \\ E & = & -DBC^{-1} & = & A^{-1}B(B^\top A^{-1}B + C)^{-1} & = & -FB^\top A^{-1} \\ F & = & (C - B^\top A^{-1}B)^{-1} & = & C^{-1} + C^{-1}B^\top(A - BC^{-1}B^\top)^{-1}BC^{-1} & & \end{array}$$

Here, we have shown the several equivalent forms for $D, E,$ and F. However, as we will show, one of these forms will be particularly convenient.

Recall that $\lambda_{\Sigma_t} = \Sigma_t^{-1}$. The same is true for the extended distribution: the information matrix is the inverse of the covariance matrix. We now solve for the components of the extended information matrix. Using this inversion lemma, we can express the joint information matrix in terms of elements of the joint covariance as:

$$\begin{aligned} \lambda^+_{\Sigma_t} &= \Sigma_t^{-1} + \Sigma_t^{-1}C_t(V_t - C_t^\top\Sigma_t^{-1}C_t)^{-1}C_t^\top\Sigma_t^{-1} \\ \lambda_{C_t} &= -\Sigma_t^{-1}C_t(V_t - C_t^\top\Sigma_t^{-1}C_t)^{-1} \\ \lambda_{V_t} &= (V_t - C_t^\top\Sigma_t^{-1}C_t)^{-1} \end{aligned}$$

As they stand, these expressions are not useful. While this allows us to express the joint extended information matrix in terms of pieces of the joint extended covariance matrix, our goal is to express the extended information matrix only in terms of information parameters.

Fortunately, these expressions can be greatly simplified. We notice that in each of the three expressions, the term $(V_t - C_t^\top\Sigma_t^{-1}C_t)^{-1}$ appears. We therefore begin with this expression:

$$\begin{aligned} \lambda_{V_t} &= (V_t - C_t^\top\Sigma_t^{-1}C_t)^{-1} \\ &= \left(G^\top\Sigma_t G + GC_\eta^\top + C_\eta G^\top + \sigma_\eta^2 - (G\Sigma_t + C_\eta)\Sigma_t^{-1}(\Sigma_t G^\top + C_\eta^\top)\right)^{-1} \\ &= \left(\sigma_\eta^2 - C_\eta\Sigma_t^{-1}C_\eta^\top\right)^{-1} \\ &= \left(\sigma_\eta^2 + 2C_\eta\lambda_{\Sigma_t}C_\eta^\top\right)^{-1} \end{aligned}$$

Here, we see that we have achieved our goal: the term λ_{V_t} can be computed using nothing but the PLG model parameters C_η, σ_η^2 and the current λ_{Σ_t}. Notice that, like the PLG, computing this term will require inverting a matrix.

A similar derivation is possible for λ_{C_t}:

$$\begin{aligned}
\lambda_{C_t} &= -\Sigma_t^{-1} C_t \lambda_{V_t} \\
&= -(-2\lambda_{\Sigma_t})(\Sigma_t G^\top + C_\eta^\top)\lambda_{V_t} \\
&= -(-2\lambda_{\Sigma_t})(-\frac{1}{2}\lambda_{\Sigma_t}^{-1} G^\top + C_\eta^\top)\lambda_{V_t} \\
&= -(G^\top - 2\lambda_{\Sigma_t} C_\eta^\top)\lambda_{V_t}.
\end{aligned}$$

Again, note that λ_{C_t} can be computed using only G, C_η and the current state λ_{Σ_t}, plus λ_{V_t}, which is a term we have already computed.

Finally, we can derive an expression for $\lambda_{\Sigma_t}^+$:

$$\begin{aligned}
\lambda_{\Sigma_t}^+ &= \Sigma_t^{-1} + \Sigma_t^{-1} C_\eta \lambda_{V_t} C_\eta^\top \Sigma_t^{-1} \\
&= -2\lambda_{\Sigma_t} + \left(-2\lambda\Sigma_t(-\frac{1}{2}\lambda_{\Sigma_t}^{-1} + C_\eta^\top)\right)\lambda_{V_t}\left(-2\lambda\Sigma_t(-\frac{1}{2}\lambda_{\Sigma_t}^{-1} + C_\eta^\top)\right)^\top \\
&= -2\lambda_{\Sigma_t} - (G^\top - 2\lambda_{\Sigma_t} C_\eta^\top)\lambda_{V_t}(G - 2C_\eta\lambda_{\Sigma_t})
\end{aligned}$$

which is similarly expressible in terms of PLG parameters and the current state. Together, $\lambda_{V_t}, \lambda_{C_t}$ and $\lambda_{\Sigma_t}^+$ yield the final extended distribution in information form.

8.2.2 Computing the Information Vector Terms

We now turn out attention to the computation of the extended information vector, which is composed of $\lambda_{\mu_t}^+$ and λ_{E_t}. Like the terms in the extended information matrix, we can find simple, closed form expressions for these vectors.

Recall that the vector:

$$\begin{bmatrix} \lambda_{\mu_t}^+ \\ \lambda_{E_t} \end{bmatrix} = \begin{bmatrix} \Sigma_t & C_t \\ C_t^\top & V_t \end{bmatrix}^{-1} \begin{bmatrix} \mu_t \\ E_t \end{bmatrix} = -2 \begin{bmatrix} \lambda_{\Sigma_t}^+ & \lambda_{C_t} \\ \lambda_{C_t}^\top & \lambda_{V_t} \end{bmatrix} \begin{bmatrix} \mu_t \\ E_t \end{bmatrix}$$

This yields our first equation:

$$\begin{aligned}
\lambda_{\mu_t}^+ &= -2(\lambda_{\Sigma_t}^+ \mu_t + \lambda_{C_t} G\mu_t) \\
&= -2(\lambda_{\Sigma_t} - \lambda_{C_t}(G - 2C_\eta\lambda_{\Sigma_t}))(-\frac{1}{2}\lambda_{\Sigma_t}^{-1}\lambda_{\mu_t}) - 2\lambda_{C_t} G(-\frac{1}{2}\lambda_{\Sigma_t}^{-1}\lambda_{\mu_t}) \\
&= \lambda_{\mu_t} - \lambda_{C_t} G\lambda_{\Sigma_t}^{-1}\lambda_{\mu_t} + 2\lambda_{C_t} C_\eta \lambda_{\mu_t} + \lambda_{C_t} G\lambda_{\Sigma_t}^{-1}\lambda_{\mu_t} \\
&= \lambda_{\mu_t} + 2\lambda_{C_t} C_\eta \lambda_{\mu_t}
\end{aligned}$$

which, like the terms we have previously computed for the information matrix, relies only on PLG

model parameters and previously computed values. Similarly:

$$\begin{aligned}
\lambda_{E_t} &= -2(\lambda_{C_t}^\top \mu_t + \lambda_{V_t} G \mu_t) \\
&= (\lambda_{C_t}^\top \lambda_{V_t} G)(\lambda_{\Sigma_t}^{-1} \lambda_{\mu_t}) \\
&= (\lambda_{V_t}(-G + 2C_\eta \lambda_{\Sigma_t}) + \lambda_{V_t} G)(\lambda_{\Sigma_t}^{-1} \lambda_{\mu_t}) \\
&= (2\lambda_{V_t} C_\eta \lambda_{\Sigma_t}))(\lambda_{\Sigma_t}^{-1} \lambda_{\mu_t}) \\
&= 2\lambda_{V_t} C_\eta \lambda_{\mu_t}.
\end{aligned}$$

This completes the equations needed to compute the information parameters of the extended distribution $p(F^n, O_{t+n+1}|h_t)$.

8.2.3 Conditioning the Distribution

Having completed the derivation of the extension part of the dynamics, we now turn our attention to the problem of conditioning the distribution on an observation $O_{t+1} = o_{t+1}$. As noted previously, conditioning in the information form is an easy operation. To simplify notation, we will now repartition the state vector and information matrix. Like a similar step in the PLG algorithm discussed in Chapter 4, this is not a mathematical operation; rather, we are just re-labeling our matrix and vector to simplify the explanation of the algorithm:

$$\begin{aligned}
\begin{pmatrix} F^n \\ O_{t+n+1} \end{pmatrix} &\sim \mathcal{N}^{-1}\left[\begin{pmatrix} \lambda_{\mu_t}^+ \\ \lambda_{E_t} \end{pmatrix}, \begin{pmatrix} \lambda_{\Sigma_t}^+ & \lambda_{C_t} \\ \lambda_{C_t}^\top & \lambda_{V_t} \end{pmatrix}\right] \\
&= \mathcal{N}^{-1}\left[\begin{pmatrix} \lambda_{o_{t+1}} \\ \lambda_{f^n} \end{pmatrix}, \begin{pmatrix} \lambda_{o_{t+1}o_{t+1}} & \lambda_{o_{t+1}f^n} \\ \lambda_{o_{t+1}f^n}^\top & \lambda_{f^n f^n} \end{pmatrix}\right]
\end{aligned}$$

Recall that conditioning a Gaussian distribution in information form is easy (as shown in Figure 8.2). To condition on $O_{t+1} = o_{t+1}$, we simply set:

$$\begin{aligned}
\lambda_{\mu_{t+1}} &= \lambda_{f^n} + 2\lambda_{o_{t+1}f^n}^\top o_{t+1} \\
\lambda_{\Sigma_{t+1}} &= \lambda_{\Sigma_{f^n f^n}}
\end{aligned}$$

This completes the derivation of the Information PLG. The algorithm is summarized in Figure 8.3.

8.3 Final Theorem

We are finally ready to state our central theorem.

Theorem 8.3.1. *The Information PLG is equivalent to the PLG in two senses: first, it computes an equivalent distribution over observations, and second, it uses exactly the same parameters.*

Proof. The proof follows by the two equivalences established. First, Theorem 8.2.1 established that

Algorithm INFORMATION-PLG-UPDATE

Input: Current state, represented by λ_{μ_t} and λ_{Σ_t} and an observation o_t.

Given: dynamical parameters G, C_η, and σ_η^2.

Compute: (construct extended distribution)

- $\lambda_{V_t} = -0.5(\sigma_\eta^2 + 2C_\eta \lambda_{\Sigma_t} C_\eta^\top)^{-1}$
- $\lambda_{C_t} = -(G^\top - 2\lambda_{\Sigma_t} C_\eta^\top)\lambda_{V_t}$
- $\lambda_{\Sigma_t}^+ = \lambda_{\Sigma_t} - \lambda_{C_t}(G - 2C_\eta \lambda_{\Sigma_t})$
- $\lambda_{\mu_t}^+ = \lambda_{\mu_t} + 2\lambda_{C_t} C_\eta \lambda_{\mu_t}$
- $\lambda_{E_t} = 2\lambda_{V_t} C_\eta \lambda_{\mu_t}$

Repartition:

$$\mathcal{N}^{-1}\left[\begin{pmatrix} \lambda_{\mu_t}^+ \\ \lambda_{E_t} \end{pmatrix}, \begin{pmatrix} \lambda_{\Sigma_t}^+ & \lambda_{C_t} \\ \lambda_{C_t}^\top & \lambda_{V_t} \end{pmatrix}\right] = \mathcal{N}^{-1}\left[\begin{pmatrix} \lambda_{o_{t+1}} \\ \lambda_{f^n} \end{pmatrix}, \begin{pmatrix} \lambda_{o_{t+1}o_{t+1}} & \lambda_{o_{t+1}f^n} \\ \lambda_{o_{t+1}f^n}^\top & \lambda_{f^n f^n} \end{pmatrix}\right]$$

Compute: (condition on observation o_{t+1})

- $\lambda_{\mu_{t+1}} = \lambda_{f^n} + 2\lambda_{o_{t+1}f^n}^\top o_{t+1}$
- $\lambda_{\Sigma_{t+1}} = \lambda_{\Sigma_{f^n f^n}}$

Return: $\lambda_{\mu_{t+1}}, \lambda_{\Sigma_{t+1}}$.

Figure 8.3: The state update equations for the Information PLG.

for every t there are equivalent natural and mean parameterizations for the distribution $p(F^n|h_t)$. Second, the derivations in Section 8.2 prove that given a natural parameterization at time t, the parameters can be updated to find the parameters at time $t+1$, which are exactly the natural parameters of the corresponding mean-parameterized Gaussian. By induction, the equivalence holds for all t (assuming an appropriate initial base case). Furthermore, the equations used to update the parameters at every timestep used only the standard PLG parameters. □

8.4 Steady State Filtering

We will now briefly discuss an interesting relationship between the Information PLG and steady state filters. The goal of this section is modest: we simply wish to demonstrate that for a steady-state filter, the state update is a completely linear function of the previous state and observation. This observation is part of the justification of the Linear-Linear EFPSR in Chapter 9.

The general Information PLG state update shown in Figure 8.3 is nonlinear, primarily because of the matrix inverse needed to compute λ_{V_t}. The same thing is true of the Kalman Filter and the PLG: each involves a matrix inverse performed on the covariance matrices at time t. However, a key observation about the (Information) Kalman Filter and the (Information) PLG is that the sequence of covariance / information matrices *does not depend on the observed data sequence.* That means that the covariance matrix can reach a "steady-state," which we define below, and that it will remain in steady state, because no sequence of observations can affect it.

A filter that is in *steady-state* is defined as (Sayed, 2003):

$$\mathrm{E}[\mu_t] = \mathrm{E}[\mu_{t-1}] = \mu_c \quad \text{as} \quad t \to \infty$$

and

$$\mathrm{E}[\Sigma_t] = \mathrm{E}[\Sigma_{t-1}] = \Sigma_c \quad \text{as} \quad t \to \infty.$$

That is, the mean vector and covariance matrix have approached a constant value that does not depend on time.

For the purposes of this chapter, we will assume that the covariance has approached a steady state, but not necessarily that the mean vector has. If Σ_t is constant, this implies that λ_{Σ_t} is constant, and therefore that λ_{V_t} is constant. Similarly, the fact that λ_{V_t} is constant implies that both λ_{C_t} and $\lambda^+_{\Sigma_t}$ are constant.

Together, these facts simplify the state update of the Information PLG, and render the entire operation linear in the state vector λ_{μ_t} and the observation o_t, as follows. We start by showing that the extended state vector is a linear function of the current state:

$$\lambda^+_{\mu_t} \quad = \quad \lambda_{\mu_t} + 2\lambda_{C_t} C_\eta \lambda_{\mu_t}$$

$$
\begin{aligned}
&= (I - 2\lambda_{C_t} C_\eta)\lambda_{\mu_t} \\
&= A_1 \lambda_{\mu_t}
\end{aligned}
$$

where $A_1 = (I - 2\lambda_{C_t} C_\eta)$. Similarly,

$$
\begin{aligned}
\lambda_{E_t} &= 2\lambda_{V_t} C \lambda_{\mu_t} \\
&= A_2 \lambda_{\mu_t}
\end{aligned}
$$

where $A_2 = 2\lambda_{V_t} C_\eta$. Together, these imply that

$$
\begin{bmatrix} \lambda_{o_{t+1}} \\ \lambda_{f^n} \end{bmatrix} = \begin{bmatrix} \lambda^+_{\mu_t} \\ \lambda_{E_t} \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \lambda_{\mu_t}
$$

where we have rolled the extension and the relabeling into one step. Thus, $\lambda_{f^n} = P[A_1; A_2]\lambda_{\mu_t}$, for an appropriate projection matrix P (this matrix simply selects the elements corresponding to λ_{f^n} out of the vector $[\lambda_{o_{t+1}}; \lambda_{f^n}]$).

Finally, since $\lambda_{o_{t+1} f^n}$ is constant:

$$
\begin{aligned}
\lambda_{\mu_{t+1}} &= \lambda_{f^n} + 2\lambda^\top_{o_{t+1} f^n} o_{t+1} \\
&= \lambda_{f^n} + A_3 o_{t+1} \\
&= P[A_1; A_2]\lambda_{\mu_t} + A_3 o_{t+1}
\end{aligned}
$$

which is a linear operation.

The point of the foregoing analysis is simply to demonstrate that in the case of a steady-state filter, both the extension and conditioning operations are linear. This partly justifies the introduction of the "Linear-Linear EFPSR" in Chapter 9. The reason that this is important is because the linearity of the state update will facilitate a dynamical analysis for future learning algorithms. For example, in certain settings, the stationary distribution of states can be computed as the solution to a linear system of equations. This will also make a variety of approximations possible, which we will discuss in Chapter 10.

8.5 Experiments

To validate the equations governing the Information PLG, we here present a simple experiment. We will define a linear dynamical system, and then convert the parameters directly to the equivalent parameters of the PLG, using the technique explained in Rudary et al. (2005). We will then track the state of both the PLG and the Information PLG, and demonstrate that the two algorithms both perform the same updates and make the same predictions.

Let $x_0 = [0; 0]$ be our initial state. Let $\theta = 0.1$, and let

$$A = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} 1.0 & 0.01 \\ -0.01 & 1.0 \end{bmatrix}$$

be a simple rotation matrix which rotates a given vector by θ degrees around the origin. Let $H = [11]$ be our observation matrix, which simply adds both coordinates together. Let $Q = 0.003 * I$ be the covariance matrix of our transition noise, and let $R = 1$ be the covariance matrix of our observation noise.

The PLG parameters are given by

$$\begin{aligned} G &= [-1.0 \;\; 1.9999] \\ C_\eta &= [1.0 \;\; 2.0059] \\ \sigma_\eta^2 &= 6.0116 \end{aligned}$$

The initial state of the PLG is

$$\begin{aligned} \mu_0 &= [7.0000 \;\; 6.9297]^\top \\ \Sigma_0 &= \begin{bmatrix} 1.0002 & 0.0002 \\ 0.0002 & 1.0062 \end{bmatrix} \end{aligned}$$

and the initial state of the Information PLG is

$$\begin{aligned} \lambda_{\mu_0} &= [6.9972 \;\; 6.8856]^\top \\ \lambda_{\Sigma_0} &= \begin{bmatrix} -4.9999 & 0.0001 \\ 0.0001 & -4.999 \end{bmatrix}. \end{aligned}$$

To compare the PLG and Information PLG, we ran the above linear dynamical system for 1,000 timesteps. We used the PLG and Information PLG state update equations, and asked both models to generate one-step predictions at every timestep. Figure 8.4 shows the results, which basically demonstrate equivalence. On the left side of the figure, we see that there is no discernible difference in the predictions made (the green line, representing the PLG predictions, cannot be seen because it is exactly obscured by the blue line, representing the Information PLG predictions). On the right, we show the difference between the two predictions. In terms of the actual states, we can convert λ_{μ_t} to μ_t and vice-versa, and we can convert λ_{Σ_t} to Σ_t. We did this after the 1,000 updates, and compared the results. $\|\lambda_{\mu_{1000}} - \mu_{1000}\| \approx 1e-11$, and $\|\lambda_{\Sigma_{1000}} - \Sigma_{1000}\| \approx 1e-12$. These differences, as well as the differences in predictions (1e-11), hover around machine precision.

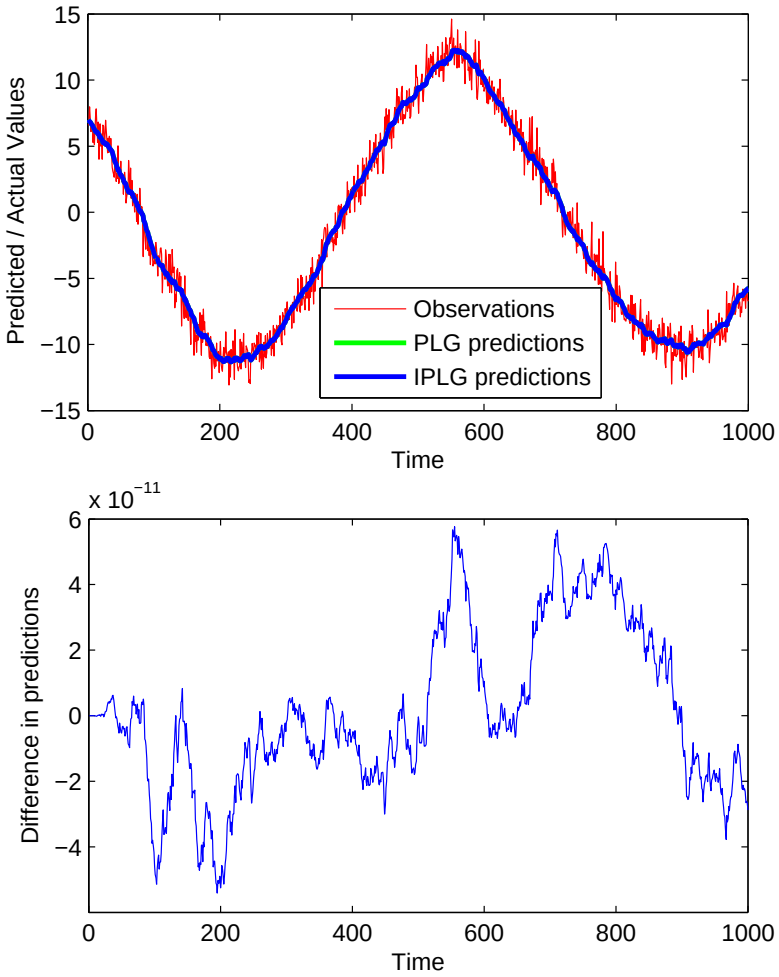


Figure 8.4: Comparing the predictions made by the PLG and the Information PLG. On the top: the predictions made versus the actual observations. On the bottom: the difference in PLG and Information PLG predictions.

8.6 Conclusions and Future Work

We have presented the Information PLG, and demonstrated that it is formally equivalent to the PLG: they use the same parameters; they compute equivalent distributions over the future, and computationally, they have approximately equal complexity.

There are several natural directions for future research. As noted in the introduction to this chapter, the Information Kalman Filter and the Extended Information Kalman Filter have enjoyed considerable success in real-world applications because of sparsity in the information form. This has made a variety of approximations possible, and it is likely that similar results could hold for the Information PLG. It is also possible to extend the Information PLG in all of the same ways that the PLG was extended: a kernel Information PLG is conceivable, as are Mixtures of Information PLGs, or even an Extended Information PLG.

It is interesting that the Information PLG uses exactly the same parameters as the PLG. This implies that every learning algorithm for a PLG is also a learning algorithm for an Information PLG, and vice-versa. One of the advantages of the PLG is the fact that its parameters can be learned directly from data through simple regressions and sample statistics. Here, there is a close connection between the expectations arising from statistics of the data and the mean parameterization of the Gaussian used as state. It would be interesting to know if similar statements were true about the EFPSR in general: is it possible to find the dynamical parameters by regressions (which exploit expectations and are related to the mean parameters), but then operate the model in the natural parameter space? If so, this could simplify the learning algorithms for the EFPSR.

Chapter 9

Exact Linear-Linear EFPSRs

In Chapter 7 we have introduced the general EFPSR model, and have pointed out that there are two design decisions which must be made to specialize the EFPSR to any particular model: features must be selected, and an extension function must be selected. In Chapter 8, we used singleton and pairwise features and a special extension function, which resulted in the Information PLG.

In this chapter, we specialize the EFPSR in a different way, in preparation to handle domains with large numbers of features and large data sets. We select a linear extension function, and we carefully choose features so that conditioning is always a linear operation. The combination of a linear extension and linear conditioning results in a state update that is a linear function, which will help facilitate a dynamical analysis and make a variety of approximations possible. We name the resulting algorithm the "Linear-Linear EFPSR." The word "Exact" in the title of the chapter refers to the fact that we will present an exact maximum likelihood learning algorithm for the model. Portions of this chapter were published by Wingate and Singh (2007a).

The choice of a linear extension function is partly motivated by the steady-state filtering results of Section 8.4, and partly motivated by the proof that EFPSRs are capable of capturing MDPs (found in Section 7.2.3). In both of these cases, the state update is a linear function of the current state and the observation, and suggests that a purely linear state update can model useful things. To obtain linear conditioning, we stipulate that all features be conjunctions of atomic observation variables. This is also partly motivated by broader graphical model research, where high order conjunctions are a common type of feature (Pietra et al., 1997).

We present the specialized model in Section 9.1, where we discuss in detail our choice of features and extension function. We then address the question of model learning in Section 9.2. We will present an exact maximum likelihood learning algorithm in Section 9.2.2, including notes on structure learning in Section 9.2.1. We then present some experimental results in Section 9.3.

9.1 The Linear-Linear EFPSR

We now discuss the specific choices that we make to create the Linear-Linear EFPSR. For the purposes of simplifying the exposition, we will assume that we are working with binary random variables, as follows.

In general, each multidimensional observation O_t that the agent receives (and therefore, the resulting histories and the domain of the future $F^n|h_t$) may be discrete or continuous. In this chapter, we will assume that a number of binary features have been extracted from the observations. We call these the *atomic features*, to distinguish them from the features created by the function ϕ. Hereafter, we will treat these binary features as if they were the observations, and ignore the underlying observations from which they are generated. This will simplify some of the math, and will simplify the problem of inference in the model. Although we use binary features, we emphasize that almost all of the concepts we develop apply equally well to real-valued and discrete-valued random variables.

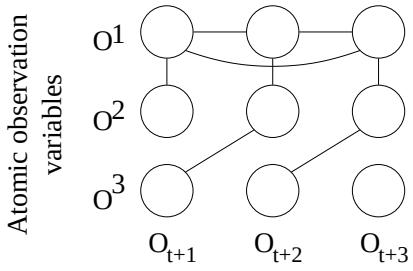
9.1.1 Conjunctions of Features and Graphical Structure

Section 7.1.2 discussed the fact that there is a restriction on extension functions: the distribution $p(F^n|h_t)$ is defined using a set of features $\phi()$. After extending, the distribution $p(F^{n+1}|h_t)$ is defined using a set of features $\phi^+()$. After conditioning to $p(F^n|h_{t+1})$, the distribution $p(F^n|h_{t+1})$ must be expressible with the same set of features $\phi()$ that were used in $p(F^n|h_t)$. In this section, we discuss how we choose features so that this constraint on the extension function is satisfied. We additionally choose features such that conditioning is a linear operation, and will simultaneously show how to impose graphical structure on the model.

The features $\phi()$ and $\phi^+()$ do not depend on time. This is equivalent to saying that the form of the distribution does not vary over time. If the features impose graphical structure on the distribution, it is also equivalent to saying that the form of the graph does not change over time. Because of this fact, we will now discuss how we can use a graph whose form is independent of time to help define structure on our distributions.

First, we describe our observation variables more precisely. Let each $O_t \in \{0,1\}^d$; therefore, each $F^n|h_t \in \{0,1\}^{nd}$. Let $(F^n)^i$ be the i'th random variable in $F^n|h_t$. We construct the feature vectors $\phi()$ and $\phi^+()$ as follows. We assume that we have an undirected graph G which we will use to create the features in the vector $\phi()$, and that we have another graph G^+ which we will use to define the features in the vector $\phi^+()$. Define $G = (V, E)$ where $V = \{1, ..., nd\}$ are the nodes in the graph (one for each $F^n|{h_t}^i$), and $(i, j) \in E$ are the edges. Similarly, we define $G+ = (V+, E+)$ where $V+ = \{1, ..., (n+1)d\}$ are the nodes in the graph (one for each $(F^{n+1}|h_t)^i$), and $(i, j) \in E+$ are the edges. As noted, neither graph depends on time.

Consider the following graph G defined over the variables in $F^n|h_t$. Here, $n = 3$ and $d = 3$, for a total of nine atomic variables:



What is the structure that we want this graph to impose upon the distribution $p(F^n|h_t)$? Like any graphical model, we want this graph to mean that $(F^n|h_t)^i$ and $(F^n|h_t)^j$ are conditionally independent of each other given all other variables in the graph, if there is no edge $(i,j) \in E$. Appendix D.3 describes more examples of imposing graphical structure upon exponential family distributions and how they can be interpreted.

To accomplish this, we will let features be conjunctions of atomic observation variables, like the standard Ising model features discussed in Section D.3. We create one feature in ϕ for each node $i \in V$. Specifically, for $i \in V$, there will be some feature k in the vector such that $\phi(f_t)^k = f_t^i$. We also create one feature for each edge. Specifically, if $(i,j) \in E$, then there will be some feature k in the vector such that $\phi(f_t)^k = f_t^i f_t^j$. For ease of exposition, we will allow only pairwise interactions between random variables, but the extension to higher order features is straightforward. We will use the graph G^+ to define the vector $\phi^+()$ in the same way.

We have discussed how we can use G or G^+ to define the features $\phi()$ and $\phi^+()$, but we must also ensure that after conditioning G^+, we recover the original graph G. Neither G nor G^+ can therefore be arbitrary. We will impose special structure on both so that their form does not change over time. One way to do this is to ensure that temporally shifted copies of each feature exist in the graph, and that conditioned versions of each feature exist in the graph. For example, if there is an edge connecting o^1_{t+5} and o^1_{t+6}, then there must also be edges connecting o^1_{t+4} to o^1_{t+5}, o^1_{t+3} to o^1_{t+4}, o^1_{t+2} to o^1_{t+3}, and o^1_{t+1} to o^1_{t+2}. This ensures that the structure of the graph does not change between state updates (we informally show this pictorially in Figure 9.1).

9.1.2 Conditioning

Because we have stipulated that all features are either atomic variables or conjunctions of variables, finding the parameters of the conditioned distribution is easy (we emphasize that this is true even if the random variables are discrete or real-valued). When we condition on an observation, we freeze the observed variables to their observed values, and then collect parameters.

To see this, consider the following example (to simplify notation, we will drop timescripts). Suppose that we have a particular distribution over two binary random variables $O = [o_1\ o_2]^\top$. We will define

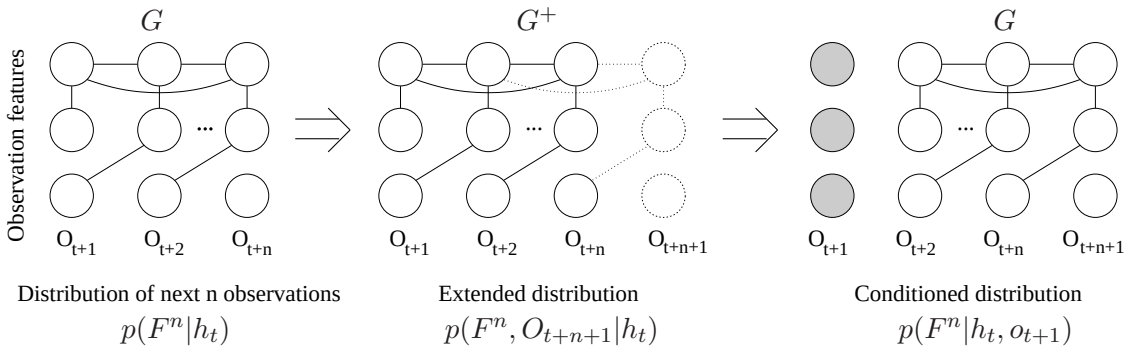


Figure 9.1: Extending and conditioning the EFPSR distribution with graphical structure. The structure of the graph must not change between state updates.

the features of these variables to be two singleton features and one conjunction:

$$\phi(O) = \begin{bmatrix} o_1 \\ o_2 \\ o_1 o_2 \end{bmatrix}.$$

We will define a density over these random variables using an exponential family distribution:

$$\begin{aligned} p(O = o; s) &= \exp\left\{-s^\top \phi(o) - \log Z(s)\right\} \\ &= \exp\left\{-(s_1 o_1 + s_2 o_2 + s_3 o_1 o_2) - \log Z(s)\right\} \end{aligned}$$

so that our state vector $s \in \mathbb{R}^3$.

Suppose we now extend this distribution to include a new variable o_3. We now have three binary random variables, $O^+ = [o_1\ o_2\ o_3]^\top$. Suppose we define the feature vector $\phi^+(O^+)$ to contain all singletons, pairwise conjunctions, and third-order conjunctions (this satisfies the temporal invariance property we discussed earlier: for every feature in the vector, temporally shifted copies are also in the vector, as well as conditioned versions of each feature):

$$\phi^+(O^+) = \begin{bmatrix} o_1 \\ o_2 \\ o_3 \\ o_1 o_2 \\ o_1 o_3 \\ o_2 o_3 \\ o_1 o_2 o_3 \end{bmatrix}.$$

Our extended density will be

$$p(o^+; s^+) = \exp\left\{-s^+ \top \phi^+(o^+) - \log Z(s)\right\}$$

$$
\begin{aligned}
&= \exp\{-(s_1o_1 + s_2o_2 + s_3o_3 + s_4o_1o_2 + s_5o_1o_3 + s_6o_2o_3 + s_7o_1o_2o_3) \\
&\quad - \log Z(s)\}
\end{aligned}
$$

so that now our state vector $s^+ \in \mathbb{R}^7$.

Suppose now that we wish to condition on a particular observation variable – say, o_1. To condition, we freeze o_1 to its observed value, and we notice that we can then collect terms into a new state vector:

$$
\begin{aligned}
p(o^+; s^+, o_1) &= \exp\left\{-s^{+\top}\phi^+(o^+) - \log Z(s^+)\right\} \\
&= \exp\left\{-(s'_1 + s_2o_2 + s_3o_3 + s'_4o_2 + s'_5o_3 + s_6o_2o_3 + s'_7o_2o_3) - \log Z(s^+)\right\} \\
&= \exp\left\{-(s'_1 + (s_2 + s'_4)o_2 + (s_3 + s'_5)o_3 + (s_6 + s'_7)o_2o_3) - \log Z(s^+)\right\} \\
&= \exp\left\{-((s_2 + s'_4)o_2 + (s_3 + s'_5)o_3 + (s_6 + s'_7)o_2o_3) - \log Z(s^+)\right\} \\
&= \exp\left\{-(s'^{\top}[o_2; o_3; o_2o_3] - \log Z(s')\right\} \\
&= \exp\left\{-(s'^{\top}\phi([o_2, o_3]) - \log Z(s')\right\}
\end{aligned}
$$

Some terms in the state vector didn't change because they did not depend on o_1, but others have changed, which we have denoted by s'_x. We have grouped terms together that interact with the same unobserved variables. Also, notice that s'_1 does not interact with the observations at all, and so it has been absorbed into the normalizing constant. We call the entire process of conditioning on an observation and grouping like terms a "freeze-and-collect" operation, because we have frozen the observed variable to a fixed value, and collected the multipliers together for the remaining variables.

Notice that after conditioning, we are now back to 3 parameters–the same number we started with. However, they describe different variables: the state vector s we started with was defined as multipliers for features of $[o_1\ o_2]$, but the state vector we ended up with (s') is defined for features of the variables $[o_2\ o_3]$. In a temporal model, we see that o_2 is now playing the role of o_1, and that o_3 is now playing the role of o_2. We have also satisfied the constraint that the final features of the extended-and-conditioned distribution be expressible in the same form as the original distribution.

This freeze-and-collect operation is linear in the state vector s. Because it is linear, we can define the matrix $G(o)$ to perform the operation, as follows:

$$
G(o_1) = \begin{bmatrix} 0 & 1 & 0 & o_1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & o_1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & o_1 \end{bmatrix}.
$$

Note that $s' = G(o_1)s^+$. $G(o)$ has l rows, where l is the number of features in the conditioned distribution, and k columns, where k is the number of features in the extended distribution. This example also helps illustrate the importance of not allowing the structure of the graph to change

between state updates.

We will apply the same logic of this example to conditioning $p(F^n, O_{t+n+1}|h_t)$ on o_{t+1} by defining the linear conditioning operator $G(o_{t+1})$ as a matrix which adds the appropriate parameters together. This matrix will be used throughout the rest of the chapter.

9.1.3 Extending

We now address the extension function. The extension function extend can take any form. In the PLG family of work, for example, a linear extension allows the model to capture linear dynamics (Rudary et al., 2005), while a non-linear extension allows the model to capture non-linear dynamics (Wingate and Singh, 2006b). Here, we focus on linear extensions:

$$s_t^+ = As_t + B$$

where $A \in \mathbb{R}^{k \times l}$ and $B \in \mathbb{R}^{k \times 1}$ are our model parameters.

The combination of a linear extension and a linear conditioning operator can be rolled together into a single operation. Without loss of generality, we can permute the indices in our state vector such that

$$s_{t+1} \quad = \quad G(o_{t+1}) \left(As_t + B \right).$$

Given model parameters A, B, an initial state s_0, and a sequence of observations, the sequence of s_t's is completely determined. This is analogous to the belief state update in, say, a POMDP: the belief state update is a deterministic function of a prior belief state and an observation.

9.2 Model Learning

We have defined the broad class of our features, as well as our extension function. However, we have still not defined exactly which conjunctions of atomic features the model uses. In addition, the extension function is parameterized by the vectors A and B, which we must determine as part of the model. In this section, we address the question of learning the exact structure of the graph, as well as the parameters A and B from data. We briefly address each in the next two subsections.

We assume that the data we are given is a single long sequence of T observations, $[o_1, \cdots, o_T]$. We will take this sequence of observations and stack n consecutive observations together to create a sequence of samples from the $F^n|h_t$'s. So, we will let $f_1 = [o_1, \cdots, o_n]$ be a sample from $F^n|\emptyset$, $f_2 = [o_2, \cdots, o_{n+1}]$ be a sample from $F^n|o_1$, etc. (this is somewhat like the suffix history algorithm described in Section 3.4). Figure 5.3 graphically illustrates the process.

9.2.1 Structure Learning

There are two aspects to structure learning: first, selecting the length n of the window into the future, and second, determining the graphical structure of the model – that is, deciding which edges to include in our graph (and therefore, which feature conjunctions to include in the feature vector $\phi(\cdot)$). For this part of learning, we make the approximation of ignoring the dynamical component of the model. That is, we treat each f_t as an observation, and try to estimate the density of the resulting unordered set, ignoring the t subscripts (we appeal to density estimation because many good algorithms have been developed for structure induction). We therefore ignore temporal relationships *across* samples, but we preserve temporal relationships *within* samples. For example, if observation a is always followed by observation b, this fact will be captured within the f_t's.

The problem therefore becomes one of inducing graphical structure for a non-sequential data set, which is a problem that has already received considerable attention. In all of our experiments, we used the method of Della Pietra et. al (Pietra et al., 1997). Their method iteratively evaluates a set of candidate features and adds the one with highest expected gain in log-likelihood. To enforce the temporal invariance property, whenever we add a feature, we also add all of the temporally shifted copies of that feature, as well as the conditioned versions of that feature. Other feature selection methods are possible; for example, Chandrasekaran et al. (2007) suggest a feature selection method based on a maximum-entropy relaxation which naturally favors sparse feature sets.

9.2.2 Maximum Likelihood Parameter Estimation

With the structure of the graph in place, we are left to learn the parameters A and B of the state extension. It is now useful that our state is defined in terms of observable quantities, for two reasons: first, because everything in our model is observed, EM-style procedures for estimating the parameters of our model are not needed, simply because there are no unobserved variables over which to take expectations. Second, when trying to learn a sequence of states (s_t's) given a long trajectory of futures (f_t's), each f_t is a sample of information directly from the distribution we're trying to model. Given a parameter estimate, an initial state s_0, and a sequence of observations, the sequence of s_t's is completely determined. This will be a key element to our proposed maximum-likelihood learning algorithm.

The likelihood of the training data is $p(o_1, o_2 ... o_T) = \prod_{t=1}^{T} p(o_t|h_t)$. We will find it more convenient to measure the likelihood of the corresponding f_t's: $p(o_1, o_2 ... o_T) \approx n \prod_{t=1}^{T} p(f_t|h_t)$ (the likelihoods are not the same because the likelihood of the f_t's counts a single observation n times; the approximate equality is because the first n and last n are counted fewer than n times).

The expected log-likelihood of the training f_t's under the model defined in Eq. 7.1 is

$$\mathcal{LL} = \frac{1}{T}\left(\sum_{t=1}^{T} -s_t^\top \phi(f_t) - \log Z(s_t)\right) \tag{9.1}$$

Our goal is to maximize this quantity. Any optimization method can be used to maximize the log-likelihood. Two popular choices are gradient ascent and quasi-Newton methods, such as (L-)BFGS. We use both, for different problems (as discussed later). However, both methods require the gradient of the likelihood with respect to the parameters, which we will now compute.

Using the chain rule of derivatives, we can compute the derivative with respect to the parameters A:

$$\frac{\partial \mathcal{LL}}{\partial A} = \sum_{t=1}^{T} \frac{\partial \mathcal{LL}}{\partial s_t}^{\top} \frac{\partial s_t}{\partial A} \tag{9.2}$$

First, we compute the derivative of the log-likelihood with respect to each state:

$$\begin{aligned} \frac{\partial \mathcal{LL}}{\partial s_t} &= \frac{\partial}{\partial s_t}\left[-s_t^{\top}\phi(f_t) - \log Z(s_t)\right] \\ &= \mathrm{E}_{s_t}[\phi(F^n|h_t)] - \phi(f_t) \\ &\equiv \delta_t \end{aligned} \tag{9.3}$$

where $\mathrm{E}_{s_t}[\phi(F^n|h_t)] \in \mathbb{R}^{l\times 1}$ is the vector of expected sufficient statistics at time t. Computing these values is a standard inference problem in exponential family models, as discussed in Section D.6.

This gradient tells us that we wish to adjust each state to make the expected features of the next n observations closer to the observed features. This is similar to the result obtained in standard maximum entropy gradients (discussed in Section 9.2.2), where the gradient attempts to move the expectation of features under the model such that it is equal to the empirical expectation. There are two differences: first, we only have one sample for each timestep, and so the empirical expectation is simply the observed sample at time t. Second: we cannot adjust s_t directly; instead, we must adjust it implicitly by adjusting the transition parameters A and B.

We now compute the gradients of the state with respect to each parameter:

$$\begin{aligned} \frac{\partial s_t}{\partial A} &= \frac{\partial}{\partial A} G(o_{t+1})\left(As_{t-1} + B\right) \\ &= G(o_{t+1})\left(A\frac{\partial s_{t-1}}{\partial A} + s_{t-1}^{\top} \otimes I\right). \end{aligned}$$

where $\otimes$ is the Kronecker product, and I is an identity matrix the same size as A.

The gradients of the state with respect to B are given by

$$\begin{aligned} \frac{\partial s_t}{\partial B} &= \frac{\partial}{\partial B} G(o_{t+1})\left(As_{t-1} + B\right) \\ &= G(o_{t+1})\left(A\frac{\partial s_{t-1}}{\partial B} + I\right) \end{aligned}$$

The gradients at time t are temporally recursive – they implicitly depend on gradients from all previous timesteps. It might seem prohibitive to compute them: must an algorithm examine all past $t_1 \cdots t_{t-1}$ data points to compute the gradient at time t? This would scale as $O(T^2)$, but fortunately, the answer is no: the necessary statistics can be computed in a recursive fashion as the algorithm walks through the data.

Even though the computation can be done recursively, we wish to make two points about the efficiency of computing these gradients. For the discussion, assume that we have l features in $\phi(f_t)$, and that we have k features in the extended distribution. This means that the matrix $A \in \mathbb{R}^{k \times l}$, that the vector $s_t \in \mathbb{R}^l$, and that there are kl total parameters describing A.

The term $\frac{\partial s_t}{\partial A}$ is a matrix, with l rows and kl columns. Given $\frac{\partial s_{t-1}}{\partial A}$, part of computing $\frac{\partial s_t}{\partial A}$ involves multiplying $\frac{\partial s_t}{\partial A}$ by A. This is an expensive matrix-matrix multiplication, which scales poorly as the number of features in the model grows. In addition, notice that this matrix-matrix multiplication must be performed T times to get the true gradient of the likelihood, which scales poorly as the size of the training set grows.

9.3 Experiments and Results

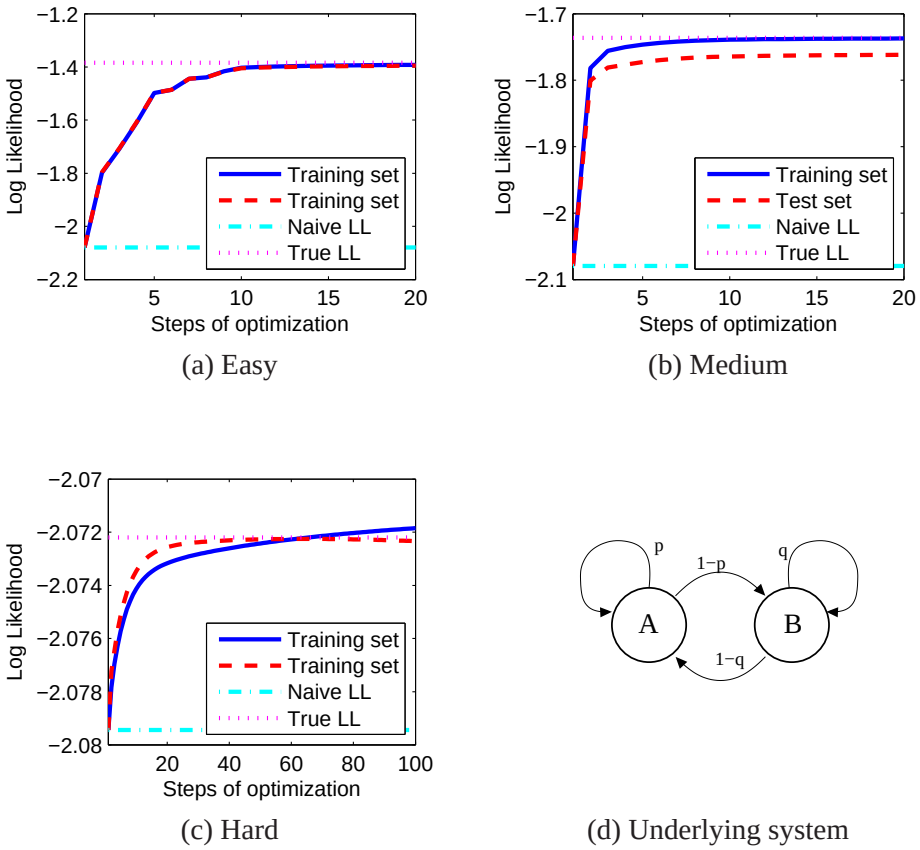
Two sets of experiments were conducted to evaluate the quality of the Linear-Linear EFPSR and the exact learning algorithm. The first set tested whether the model could capture exact state, given a complete set of features and exact inference. We evaluated the learned model using exact inference to compute the likelihood of the data under the model, and compared to the true likelihood.

The second set tested larger models, for which exact inference is not possible. For the second set, bounds can be provided for the likelihoods, but may be so loose as to be uninformative. How can we assess the quality of the final model? One objective gauge is control performance: if the domain has a reward signal, reinforcement learning can be used to determine an optimal policy. Evaluating the reward achieved becomes an objective measure of model quality, even though approximate likelihood is the learning signal.

9.3.1 First set: likelihood evaluations

For these experiments, we tested on three two-state problems, as well as three small, standard POMDPs. For the two-state problems, training and test sets were generated (using a uniformly random policy for controlled systems). We used 10,000 samples, set $n = 3$ and used all possible features (a total of seven, plus an additional four features describing the extended distribution). We used exact inference to compute the $\mathrm{E}[\phi(F^n|h_t)]$ term needed for the gradients. We optimized the likelihood using steepest descent with a line search.

Figure 9.2 (a)-(c) shows the results for three small two-state POMDPs with binary observations. Sub-figure (a) shows results for a two-state MDP (that is, the observation probabilities were set



(a) Easy (b) Medium

(c) Hard (d) Underlying system

Figure 9.2: Empirical results for the EFPSR on two-state systems. Sub-figure (d) shows the generic model used. By varying the transition and observation probabilities, three different POMDPs were generated. Sub-figures (a)-(c) show learning performance on the three models. Likelihoods for naive predictions are shown as a dash-dot line near the bottom; likelihoods for optimal predictions are shown as a dotted line near the top.

Problem	# of states	# of obs.	# of actions	Naive LL	True LL	Training set LL	Training set %	Test set LL	Test set %
Easy	2	2	0	-2.08	-1.38	-1.39	**99.9**	-1.39	**99.1**
Medium	2	2	0	-2.08	-1.74	-1.74	**100.3**	-1.76	**93.5**
Hard	2	2	0	-2.08	-2.07	-2.07	**98.6**	-2.07	**98.6**
Paint	16	2	4	-6.24	-4.66	-4.67	**99.7**	-4.66	**99.9**
Network	7	2	4	-6.24	-4.49	-4.50	**99.5**	-4.52	**98.0**
Tiger	2	2	3	-6.24	-5.23	-5.24	**92.4**	-5.25	**86.0**

Figure 9.3: Empirical results for the EFPSR on benchmark POMDPs. See text for explanation.

such that observation indicates state). The likelihood of the data under the learned model closely approaches the likelihood under the true model – on both training and test sets – indicating that the Linear-Linear EFPSR has learned a virtually perfect model. Sub-figure (b) shows results for a moderately noisy POMDP; again, the learned model is almost perfect, although the generalization is not as strong. Sub-figure (c) shows results for a very noisy POMDP, in which the naive and true log-likelihoods are very close. This indicates that prediction is difficult, even with the true model. Even so, we learn a virtually perfect model, which closely approaches optimal likelihood and generalizes well to the test set. Notice that the optimizer starts to overfit on the training set after about 60 iterations. At this point, performance on the test set begins to decline (although this is difficult to see in the figure).

Figure 9.3 collects these results in a tabular form, and shows additional results for three standard POMDPs, named Paint, Network and Tiger[1]. For these new problems, we also set $n = 3$, but used structure learning as explained in Section 9.2.1 to learn the features. For each dataset, we computed the log-likelihood of the data under the true model, as well as the log-likelihood of a "naive" model, which simply assigns uniform probability to every possible observation. We then learned the best model possible, and compared the final log-likelihood under the learned and true models, for both training and test sets. To help interpret the results, we also report a percentage (highlighted in bold), which indicates the amount of the likelihood gap (between the naive and true models) that was captured by the learned model. Higher is better; again we see that the learned models are quite accurate, and generalize well. Finally, we note that the number of latent states for these POMDPs varies from two to sixteen. In every case, however, the EFPSR used $n = 3$, which appears to be largely sufficient.

9.3.2 Second set: control performance evaluations

We also tested on two standard POMDPs called "Cheesemaze" and "Maze 4x3". We again used $n = 3$ and 10,000 training points. We used "streamer features" for the high order conjunctions, which are shown pictorially in Figure 9.4. These features connect observations with their temporal successors, but there are no connections between different observation variables. For Cheesemaze, this resulted in a total of 66 features describing $p(F^n|h_t)$, plus an additional 33 features to describe $p(F^{n+1}|h_t)$. For Maze 4x3, this resulted in 60 and 30 features, respectively. For comparison, the Cheesemaze model had 6,534 parameters, while the true model has 561 (about 1/10 as many).

For both problems, exact inference is intractable, and so we used approximate methods. To compute the $\mathrm{E}[\phi(F^n|h_t)]$ term needed for the gradients, we experimented with loopy belief propagation (LBP) (Yedida et al., 2001), naive mean field (or variational mean field, VMF), and log-determinant relaxations (LDR) (Wainwright and Jordan, 2006). Since the VMF and LDR bounds on the log-likelihood were so loose (and LBP provides no bound), it was impossible to assess our model by an appeal to likelihood. Instead, we opted to evaluate the models based on control performance.

[1] From Tony Cassandra's POMDP repository at http://www.cs.brown.edu/research/ai/pomdp/index.html

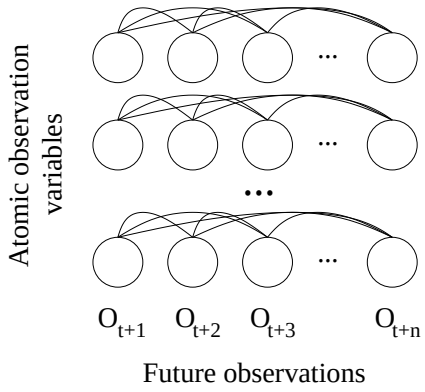


Figure 9.4: "Streamer features" used in the EFPSR feature extractor. Shown is the graph representing the features extracted, which are all possible conjunctions of temporally successive atomic features. Notice that no features cross different atomic variables.

Using LSPI. Our initial experiments used the LSPI (Lagoudakis and Parr, 2003) planning algorithm, which is a value function based algorithm. We generated a fixed sequence of actions and observations using a random policy. After each step of optimization, we used the parameter estimate to generate a corresponding set of states. We fed the states, rewards, and actions to LSPI to generate an approximate Q-function. We then ran the agent in the model using a greedy policy based on the learned Q-function, and report the average reward over 1000 steps.

We found that this did not work well: the algorithm resulted in policy chatter, which gave very poor performance. Roughly speaking, this happens because the error in the function approximation is greater than the gradient in the values. Too often, the agent would get stuck repeatedly taking alternating actions whose effects canceled (for example, alternating between moving forward and backward), which resulted in low reward. This was true of similar experiments in other domains, reported in Section 10.3.2.

Using NAC. We also experimented with the Natural Actor Critic (NAC) algorithm of Peters et al. (2005), which gave better overall performance. NAC is a policy gradient method. Policy gradient methods define a stochastic policy which is parameterized, and compute the gradient of the average long-term reward with respect to the policy parameters. Thus, taking a step in the gradient direction should always increase the average reward. While NAC is a policy gradient algorithm at heart, it combines policy gradients with several additional ideas: it uses control variates to reduce the variance in the gradients, and adds ideas from linear value function approximation, eligibility traces and information geometry.

The NAC algorithm requires two things: a stochastic, parameterized policy which operates as a function of state, and the gradients of the log probability of that policy. A common representation of the policy is to use a softmax function of a linear projection of the state, where the projection operator becomes the parameter to be determined. This is the approach we adopt. We compute the

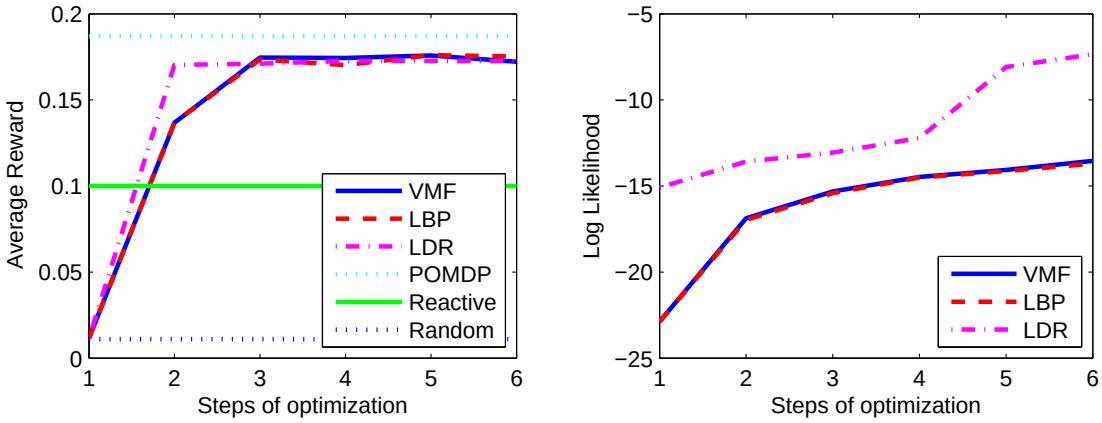


Figure 9.5: EFPSR control performance on Cheesemaze. On the left: average reward results on Cheesemaze for different approximate inference methods. On the right: progress of the optimizer for different approximate inference methods (see text for explanations).

probability of taking action a_i from state s_t given the policy parameters θ as:

$$p(a_i; s_t, \theta) = \frac{\exp\left\{s_t^\top \theta_i\right\}}{\sum_{j=1}^{|\mathcal{A}|} \exp\left\{s_t^\top \theta_j\right\}}. \tag{9.4}$$

Here θ_i is a vector of the same dimension as s_t.

The NAC algorithm requires the gradient of the log of this probability, which is easily computed:

$$\frac{\partial \log p(a_i; s_t, \theta)}{\partial \theta_j} = \begin{cases} (1 - p(a_i; s_t, \theta))s_t & \text{if } i = j \\ -p(a_i; s_t, \theta)s_t & \text{if } i \neq j \end{cases}$$

The NAC algorithm also requires a few additional parameters. We used a TD rate of $\lambda = 0.85$, a stepsize $\alpha = 10.0$, gradient termination test $\epsilon = 0.001$ and remembering factor $\beta = 0.0$.

For comparison, we also ran the NAC planner with the POMDP belief state. That is, we used the same form for the stochastic policy and the same gradients, but in Eq. 9.4 we used the belief state of the true POMDP in place of using the s_t from the Approximate Linear-Linear EFPSR. We also tested against using nothing but the observation to plan with – we used the observation vector instead of using s_t in Eq. 9.4. We also compared to a totally random policy.

Results. Figure 9.5 shows the results for Cheesemaze. The left panel shows the best control performance obtained (measured as average reward per timestep) as a function of steps of optimization. The "POMDP" line shows the best reward obtained using the true belief state as computed under the true model as the input to the NAC algorithm. The "Random" line shows the reward obtained with a random policy, and the "Reactive" line shows the best reward obtained by using the observation

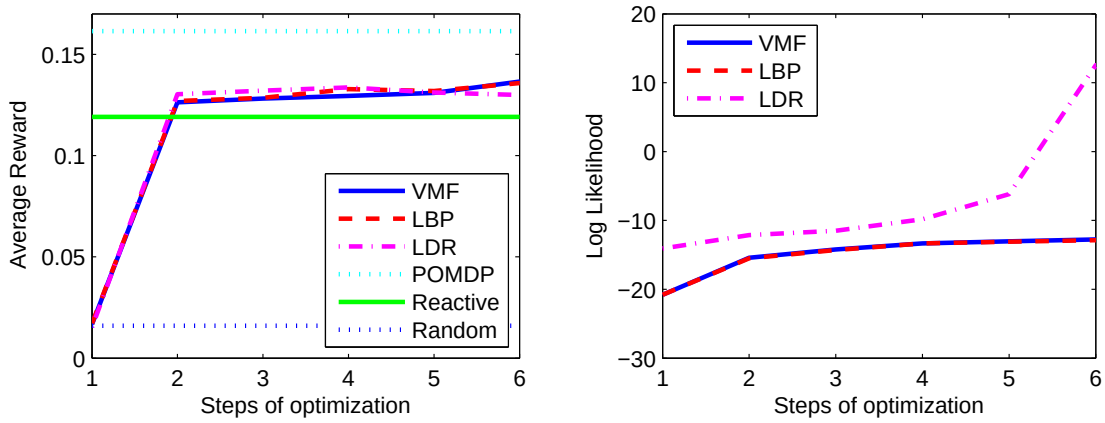


Figure 9.6: EFPSR control performance on Maze 4x3. On the left: average reward results on Maze 4x3 for different approximate inference methods. On the right: progress of the optimizer for different approximate inference methods.

as input to the NAC algorithm. The lines "VMF," "LBP," and "LDR" correspond to the different inference methods discussed previously.

The EFPSR models all start out with performance equivalent to the random policy (reward of 0.01), and quickly hop to an average reward of 0.176. This is close to the reward of using the true POMDP state, which achieves an average 0.187. The EFPSR policy closes about 94% of the gap between a random policy and the policy obtained with the true model. Surprisingly, only a few iterations of optimization were necessary to generate a state representation that was conducive to good control performance: best performance was obtained after two or three iterations.

The right panel in Figure 9.5 shows the progress of each inference method over time. We report the lower bound provided by LDR, the upper bound from VMF, and a quantity derived from LBP (which is not a bound). In all three cases, the curves largely match the performance curves, although LDR has a sudden hop in the middle. This implies that better models (higher likelihoods) yield better control performance, although the correspondence is not exact.

Figure 9.6 shows the results for Maze 4x3. Again, the left panel shows control performance, and the right panel shows optimizer progression. In some ways, these results parallel those on Cheesemaze: there was no significant difference between the different inference methods, and only a few steps of optimization were needed to reach the best performing levels. As the optimizer increased likelihood, control performance also improved to a point (with the LDR algorithm showing the same bump at the end of learning). However, the best performance obtained here was not as good as in the Cheesemaze domain: the EFPSR policies do better than simple random or reactive policies, but they are only a little bit better than reactive. In this domain, the EFPSR policy closes about 77.8% of the gap between a random policy and the policy obtained with the true model.

This domain was very challenging: experiments with different sets of features resulted in substantially similar results. The EFPSR was always able to win out over a reactive policy, but not by as large of a margin as in the Cheesemaze domain. Determining why this difference exists is important. Is is the features? The extension function? A fundamental characteristic of the domain? Determining exactly when the EFPSR is expected to work well is still an open question, but it is evident that there are at least some domains for which the learning algorithms work well.

We can draw two conclusions from these results. For both domains, we conclude that the EFPSR has learned a model which successfully incorporates information from history into the state representation, and that it is this information which the NAC algorithm uses to obtain better-than-reactive control performance. This implies that the model and learning algorithm are useful even with approximate inference methods, and even in cases where we cannot compare to the exact likelihood. We can also conclude that the combination of features and learning algorithm work well for some domains, and not as well for others. Characterizing this precisely is an important direction for future research.

9.4 Conclusions and Future Work

In this chapter, we have presented a specialization of the EFPSR named the Linear-Linear EFPSR. In this model, both the extension and conditioning functions are linear operators, which resulted in attractive practical properties. Tracking state in the model is simple, consisting of matrix-vector multiply at each timestep, and computing the gradients needed for maximum likelihood learning was straightforward because the derivative operator is also linear. Empirically, the exact maximum likelihood learning is able to learn almost perfect models of the small systems presented here, when we had a gold standard to compare against. Even when we could not evaluate model quality by comparing to exact likelihoods, we were able to use the NAC algorithm in conjunction with our model to control the system successfully.

It is interesting that we were able to learn fairly accurate models of the systems considered here even with the simple linear state update we have proposed. There was no a priori guarantee that a POMDP would be representable by the model class we have chosen, and indeed, it intuitively would have seemed necessary to have some sort of nonlinearity in the state update, since many popular models do: PSRs and POMDPs both require a nonlinear normalization operation to condition on the observation, and PLGs and Kalman Filters require a matrix inverse to accomplish the same thing. These results suggest that strictly linear state updates can work well.

However, the exact learning algorithm for the model is not expected to scale well, due to the repeated inference calls needed to compute $\mathrm{E}[\phi(f_t|h_t)]$ and the expensive matrix-matrix multiplications needed to propagate the gradients (this was discussed in Section 9.2.2). Chapter 10 addresses these issues with an approximate learning algorithm for domains with many features and large training sets.

Chapter 10

Approximate Linear-Linear EFPSRs

In the previous chapter, we presented the Linear-Linear EFPSR and an exact maximum likelihood learning algorithm. The model and learning algorithm were able to capture several small POMDP style domains with high fidelity, which suggests that both are sound. However, not reflected in the results is the fact that the exact learning algorithm is fundamentally unscalable to large numbers of features and large training set sizes. Ideally, we would like to be able to compute a few sufficient statistics of our training set – in the best case, from a single pass through the data – and then run a gradient optimizer which requires only the sufficient statistics. We would also like to minimize the number of inference calls per gradient step, since it is an expensive operation.

We will now present an approximate learning algorithm which achieves all of these desiderata: it requires only a few sufficient statistics of a training set, which are computable in linear time, and it only requires one inference call per gradient step. We accomplish this with two approximations: the first, presented in Section 10.1, allows the model to cope with large sets of training data. The second, presented in Section 10.2, allows the model to cope with large numbers of features and parameters.

Together, the combination of the Linear-Linear EFPSR and the approximate maximum likelihood learning algorithm allow the algorithm to work on domains with tens of thousands of features, which is larger than any other model with a predictive representation of state. Section 10.3 presents results for the algorithms on our standard POMDPs, a discretized bouncing ball task, as well as on a visual navigation task, where a robot must navigate a maze using nothing but features of camera images as observations.

10.1 Approximation #1: Eliminate Dependence on Time

In order to achieve an efficient learning algorithm, we will first address the dependence of the exact learning algorithm on T, the size of the training set. We will revisit the basic likelihood equation, and examine what happens in the limit as $T \to \infty$. We will present an approximate expression for likelihood, and show that its gradient can be efficiently computed.

Recall that the exact expected log-likelihood of the training f_t's under the model defined in Eq. 7.1

is

$$\mathcal{LL} = \frac{1}{T}\left(\sum_{t=1}^{T} -s_t^\top \phi(f_t) - \log Z(s_t)\right)$$

and that the gradient of the likelihood with respect to a state s_t is

$$\frac{\partial \mathcal{LL}}{\partial s_t} = \mathrm{E}_{s_t}[\phi(F^n|h_t)] - \phi(f_t)$$

This implies that some sort of inference method must be applied at each timestep in order to compute $\mathrm{E}[\phi(F^n|h_t)]$ – inference is repeated T times (the length of the training trajectory) in order to get one gradient, which is then used in an outer optimization loop. Most inference methods are slow enough that this is simply not feasible if T is large. Section 9.2.2 also discussed the computational burdens of the matrix-matrix multiplications needed to compute exact gradients.

We will now make one central assumption:

Assumption 10.1.1. We assume that $\mathrm{Cov}[s_t, \phi(f_t)] = 0$ and that $\mathrm{Cov}[s_t, o_t] = 0, \forall t$.

This assumption states that the state does not covary with observable quantities. In particular, it implies that $\mathrm{E}[s_t^\top \phi(f_t)] = \mathrm{E}[s_t]^\top \mathrm{E}[\phi(f_t)]$, which will be repeatedly used in the following derivation. This is not as severe of an assumption as it may appear to be – in particular, that this does not imply that s_t and $\phi(f_t)$ are independent.

We begin by introducing an approximate log-likelihood $\widehat{\mathcal{LL}}$, which is be a lower bound on the exact likelihood. It is derived using our assumption and a lower bound:

$$\begin{aligned}
\mathcal{LL} &= \frac{1}{T}\left(\sum_{t=1}^{T} -s_t^\top \phi(f_t) - \log Z(s_t)\right) \\
&= \mathrm{E}_T\left[-s_t^\top \phi(f_t) - \log Z(s_t)\right] \\
&= \mathrm{E}_T\left[-s_t^\top \phi(f_t)\right] - \mathrm{E}_T\left[\log Z(s_t)\right] \\
&\approx \mathrm{E}_T\left[-s_t\right]^\top \mathrm{E}_T\left[\phi(f_t)\right] - \mathrm{E}_T\left[\log Z(s_t)\right] \\
&\geq \mathrm{E}_T\left[-s_t\right]^\top \mathrm{E}_T\left[\phi(f_t)\right] - \log Z(\mathrm{E}_T\left[s_t\right]) \\
&\equiv \widehat{\mathcal{LL}}
\end{aligned}$$

where we have defined the operator

$$\mathrm{E}_T[X] \equiv \frac{1}{T}\sum_{t=1}^{T} X.$$

The fourth line in the derivation follows because of Assumption 10.1.1. The fifth line is obtained

by a double application of Jensen's inequality:

$$\begin{aligned}
\mathrm{E}[-\log Z(s_t)] &= \mathrm{E}\left[-\log(\int \exp(-s_t^\top \phi(F))dF)\right] \\
&\geq -\log(\mathrm{E}\left[\int \exp(-s_t^\top \phi(F))dF\right]) \\
&\geq -\log(\int \exp(\mathrm{E}\left[-s_t^\top \phi(F)\right])dF) \\
&\approx -\log(\int \exp(\mathrm{E}\left[-s_t\right]^\top \mathrm{E}\left[\phi(F)\right])dF) \\
&= -\log Z(\mathrm{E}[s_t])
\end{aligned}$$

The second and third lines follow because of the convexity of the functions $-\log$ and $\exp$, and the fourth line follows by Assumption 10.1.1.

The approximate log-likelihood involves several new terms, which we now explain. Consider $\mathrm{E}_T[s_t]$. Because this is an unconditional expectation, this can be interpreted as the stationary distribution of states induced by the parameters A and B, and will play a central role in the learning algorithm to follow. At first glance, this term would appear to defeat the point of our approximations: it appears to depend on T and on A and B, which means that we would have to recompute it, at cost T, every time A or B change (as they would inside any sort of optimization loop). However, we will show that this can be efficiently computed as the solution to a linear system of equations in a way that does not depend on T.

The other terms have simple interpretations. $\mathrm{E}_T[\phi(f_t)]$ is the mean of the empirically observed features of n-step trajectories, and can be considered a suffix-history estimate of the features of the n-step system dynamics vector. It can be computed in a single pass through the data, and does not depend on the parameters A or B; it can therefore be computed once at the beginning of learning. The quantity $\log Z(\mathrm{E}_T[s_t])$ also has a simple interpretation: it is the partition function Z computed using the vector $\mathrm{E}_T[s_t]$, and can be computed in the same way as the partition function associated with any ordinary state s_t.

10.1.1 Algorithm Summary

Let us now pause to summarize what we have accomplished. The exact log-likelihood $\mathcal{LL}$ defined in Eq. 9.1 is intractable due to expensive matrix-matrix multiplications and the repeated inference calls necessary to compute the expected sufficient statistics at each timestep.

To remedy this, we have defined an approximate log-likelihood $\widehat{\mathcal{LL}}$. Appendix E shows that both $\widehat{\mathcal{LL}}$ and the derivative of $\widehat{\mathcal{LL}}$ with respect to the model parameters can be computed efficiently: at each iteration of the parameter optimizer, we must only solve two sparse linear systems of equations and perform inference on the graphical model once.

Algorithm APPROX-LEARN-EFPSRS

Input: $\mathrm{E}_T[o_t]$, $\mathrm{E}_T[\phi(f_t)]$

Initialize: Let $A = 0$, $B = 0$.

Repeat:

- Compute the gradients and approximate log-likelihood:
  $(\widehat{\mathcal{LL}}, \nabla_A \widehat{\mathcal{LL}}, \nabla_B \widehat{\mathcal{LL}}) =$APPROX-LL-GRADS$(\mathrm{E}_T[o_t], \mathrm{E}_T[\phi(f_t)], A, B)$
- Use the gradients to improve the parameters, using any optimization method. In the case of steepest descent, set $A = A + \alpha \nabla_A \widehat{\mathcal{LL}}$ and $B = B + \alpha \nabla_B \widehat{\mathcal{LL}}$, where α is a suitably chosen stepsize.

Until ($\widehat{\mathcal{LL}}$ is maximized)

Return: A, B

Figure 10.1: Approximate Linear-Linear EFPSR learning algorithm. The complexity of this algorithm does not depend on T (the number of training samples).

During the development of this approximate quantity, we have introduced several new terms. Putting them all together, we see that this learning algorithm is attempting:

- to find a setting of the parameters A and B
- which generate a stationary distribution of states $\mathrm{E}_T[s_t]$,
- based on a transition operator defined using the stationary distribution of observations $\mathrm{E}_T[o_t]$,
- which imply a stationary distribution of features of length n trajectories $\mathrm{E}_{\mathrm{E}_T[s_t]}[\phi(F^n|h_t)]$ as close as possible to the empirically observed stationary distribution of features of length n trajectories $\mathrm{E}_T[\phi(f_t)]$.

With gradients in hand, any optimization method may be used to find the optimal settings for A and B. The final gradient algorithm is shown in Figure E.1 (in Appendix E), and a simple companion steepest descent optimizer is shown in Figure 10.1.

10.2 Approximation #2: A Low-Rank Parameterization

We now turn our attention to the parameter matrices A and B. So far, we have implicitly assumed that the matrix A is reasonably sized, but this assumption is false in the case of a large number of features. For the rest of this section we describe this problem in detail, but in the interests of clarity, we defer a detailed description of our solution to Section E.3.

While the approximate maximum likelihood learning algorithm shown in Figure 10.1 has several appealing properties, it is not yet suitable for very large problems, because the *naive parameterization* of A yields too many parameters. To clarify this, recall that our state s_t is a vector $\in \mathbb{R}^{l\times 1}$, where l is the number of features of the future. When we extend and condition, we implicitly compute s_t^+, which is a vector of parameters describing $n+1$ observations:

$$s_t^+ = As_t + B$$

If we assume that there are k extended features, the A matrix is $\in \mathbb{R}^{k\times l}$.

One of the goals of EFPSRs is to be able to use many features in order to capture state. If the number of features l is very large (say, tens of thousands, or even millions), the number of extended features k will be even larger, and the matrix A will be too large to work with. For example, suppose that there are 10,000 features, and that the extended distribution has 15,000 features. Naively, the matrix $A \in \mathbb{R}^{15{,}000\times 10{,}000}$, which is simply too large to deal with.

There are two possible solutions to this problem: one is to enforce some sort of sparsity on the matrix A, resulting in a manageable number of parameters. While appealing, it begs the questions: of all the possible parameters, which should be constrained to be zero? An alternative solution is to force A to be low-rank – that is, $\mathrm{rank}(A) = d \ll \min(l,k)$.

We adopt the low-rank approach. We will replace the matrix A with its low-rank decomposition $A = USV^\top$, which we will learn from data. We select this strategy for three complementary reasons:

1. Recall that the algorithm in Figure 10.1 involves the solution of two sparse linear systems of equations. Iterative solvers for such systems require only a function which can compute a matrix-vector product, which can be done efficiently for a low-rank matrix.
2. The gradients $\nabla_A \widehat{\mathcal{LL}}$ used for parameter updates have a natural rank-one form, and therefore mesh well with singular value decomposition (SVD) update algorithms: given the SVD of a matrix and a rank-one update, the parameters of the updated SVD can be efficiently computed.
3. An efficient optimization procedure based on line searches is possible. We will be using a gradient based algorithm to optimize our parameters. The performance of such optimizers is impacted by their stepsize parameters: if they are too large, the optimizer hops over solutions, but if they are too small, convergence is unacceptably slow. A line search is a common way to adaptively select a stepsize. We develop an optimizer which is rank-aware – that is, which explicitly deals with low-rank updates to the parameter matrix.

Section E.3 in the appendix discusses these points in more detail, and presents a method for computing the gradients of $\widehat{\mathcal{LL}}$ with a candidate rank-one update. In addition, it presents Brand's algorithm

Algorithm LEARN-LOW-RANK-EFPSRS

Input: $\mathrm{E}_T[o_t]$, $\mathrm{E}_T[\phi(f_t)]$, d

Initialize: Let $U = 0, S = 0, V = 0, B = 0, x = 0, y = 0, b = 0$.

Repeat:

- Compute the gradients and approximate log-likelihood:$(\widehat{\mathcal{LL}}, \Gamma_G, \mathrm{E}_T\left[s_t\right], \Delta_G) =$ LOWRANK-APPROX-LL-GRADS$(\mathrm{E}_T[o_t], \mathrm{E}_T[\phi(f_t)], U, S, V, B, 0, 0, 0)$
- Remark: $\nabla_A \widehat{\mathcal{LL}} = \Gamma_G \mathrm{E}_T\left[s_t\right]^\top$, but is never explicitly formed.
- Remark: $\nabla_B \widehat{\mathcal{LL}} = \Delta_G$.
- Conduct a line search: Find $\alpha > 0$ which maximizes $(\widehat{\mathcal{LL}}) =$LOWRANK-APPROX-LL-GRADS$(\mathrm{E}_T[o_t], \mathrm{E}_T[\phi(f_t)], U, S, V, B, \alpha\Gamma_G, \mathrm{E}_T\left[s_t\right], \Delta_G)$
- Update the parameterization of A: $(U, S, V) =$UPDATE-SVD$(d, U, S, V, \alpha\Gamma_G, \mathrm{E}_T\left[s_t\right])$
- Update B: $B = B + \alpha\Delta_G$.

Until ($\widehat{\mathcal{LL}}$ is maximized)

Return: U, S, V, B

Figure 10.2: Rank-aware EFPSR learning algorithm. The algorithm ensures that A remains rank d.

for updating the SVD. Figure 10.2 shows a steepest descent optimizer which performs a rank-aware line search, and then updates the parameter matrix while maintaining a low-rank decomposition. The parameter d can be selected with cross-validation or more sophisticated methods; we found that in all of our experiments, small values on the order of five or six worked well.

10.3 Experiments and Results

In this section, we present experimental results assessing the quality of the approximations proposed in Sections 10.1 and 10.2. In Section 10.3.1 we present results on the small POMDP benchmark domains used in Section 9.3, and measure the performance of the algorithm with exact likelihoods.

For larger problems, exact likelihoods are not an option. Instead, we use reinforcement learning to help measure the quality of the model. In Section 10.3.2 we discuss how we use the state of the Approximate Linear-Linear EFPSR in the natural actor-critic planning algorithm, and present results showing that in both of our test domains, the final performance is very close to that obtained with the exact model learning algorithm.

In Section 10.3.3, we present results for a hand-crafted domain called the "Bouncing Ball." This is a domain where the observations are known to factor in a certain way and which should be amenable to solution with our algorithm. Empirically, we demonstrate that in this domain the Approximate Linear-Linear EFPSR indeed performs better than either a random or reactive policy.

Finally, in Section 10.3.4 we present results for the largest domain of all, which is the Robot Vision domain. This domain is somewhat similar to the autonomous robot domain described in Section 3.6, but there are significant differences. Both versions use the same underlying engine: the latent state space, the actions, and the rendered camera images are all the same. However, in this version of the domain, the agent observes about 800 binary features which are extracted from each image. In Section 3.6, the agent only received a vector of three real values, representing the dominant color in the image.

10.3.1 Testing on Small POMDPs

We first examine the effects of Approximation #1 from Section 10.1, where we eliminated the dependence on T, which is the number of samples in the training set. We refer to the gradients computed by the algorithm *timeless gradients*. We present the results of two sets of experiments which are designed analogously to those reported for the Exact Linear-Linear EFPSR in Section 9.3.

The first set of experiments tested on the same benchmark POMDPs Paint Network and Tiger, as well as the Easy, Medium and Hard two-state problems. Figure 10.3 shows the results by reporting "Model Quality." Two bars are reported for each problem; for now, we focus on the bar labeled "Full parameter matrix." Like Figure 9.3, the "Model Quality" number reports the amount by which the gap between true and naive likelihoods was closed. For example, the naive LL on the Easy problem is -2.08, and the true LL is -1.38. The timeless algorithm generated a model with a LL of -1.52, which closes about 80% of the gap. The figure shows results that are comparable to those for the exact learning algorithm: on the Paint and Medium problems, the models were almost perfect, with a score of 99%. On the Network problem, the score was lower, at 92%, while the Easy and Hard models scored about 80%.

There are a few points worth noting here. First, the algorithm is capable of generating models with the same quality as the exact algorithm, as demonstrated by the Paint, Medium and Tiger problems, but it does not always work perfectly. It is interesting that among the two-state problems, the algorithm performed best on the Medium problem, while exact algorithm performed equally well on all three. The reason for this is unknown, but could be due to reaching different local minima.

The second set of experiments is reported in the same figure. The figure shows results testing the quality of Approximation #2 presented in Section 10.2. The figure reports a bar for "USV parameter matrix," which is the model quality using the USV approximation plus the rank-aware line search. In every case, the dimension d was limited to be no more than 5. The results suggest that this

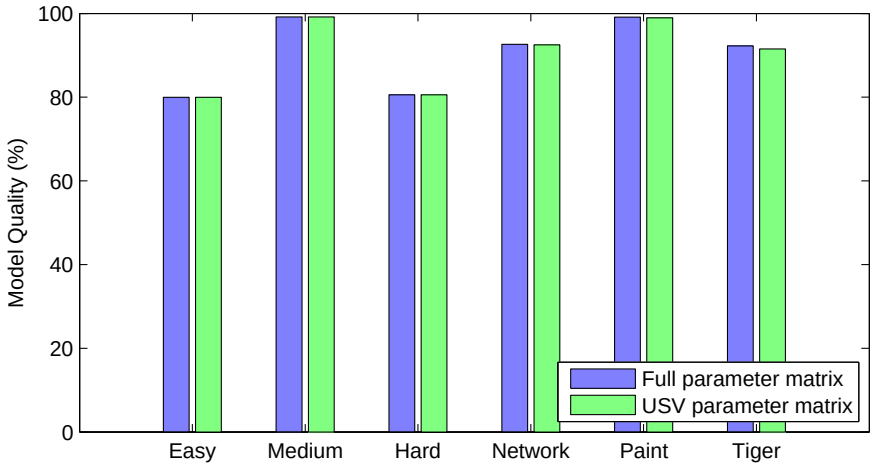


Figure 10.3: Learning with timeless gradients. Shown are results for the full parameterization of A and the low-rank parameterization of A.

approximation and the corresponding line search have virtually no impact on model quality when compared to their exact counterparts, while reducing the number of parameters (see Fig. 10.3). This is a positive result: it suggests that the idea of using a low-rank decomposition for A is sound, that the SVD update algorithm works well, and that the rank-aware line search works well. Of course, it is likely that reducing d too far would have an adverse effect on model quality. Determining the optimal d is an interesting direction for future experiments.

10.3.2 Planning in the Approximate Linear-Linear EFPSR

To test the Approximate Linear-Linear EFPSR in larger domains, we cannot appeal to likelihood. Instead, we measure model quality by using the states in a reinforcement learning algorithm, as we did in Section 9.3.2.

We tested on the same two domains used in the Exact Linear-Linear EFPSR: Cheesemaze and Maze 4x3. The observations, actions, and features are all described in Section 9.3.2. The only difference is that in the Maze 4x3 domain we experimented with different settings for n. We learned models for both domains with the Approximate Linear-Linear EFPSR, using both the timeless gradients and the lowrank parameterization of the A matrix, with d constrained to be less than 20.

Figure 10.4 shows the results of running the NAC planner in the Cheesemaze domain for five different algorithms. The best performing is the true POMDP model, which achieves an average reward of 0.187 per timestep. The worst performing is the reactive model, which achieves an average reward of 0.1 per timestep. Three different inference methods were used to learn models for the EFPSR, with the VMF and LBP models performing just under the performance of the true model at an average reward of 0.177. Here, we see that the Approximate Linear-Linear EFPSR has learned a good model of the system, with relatively few features and a rather small amount of training data.

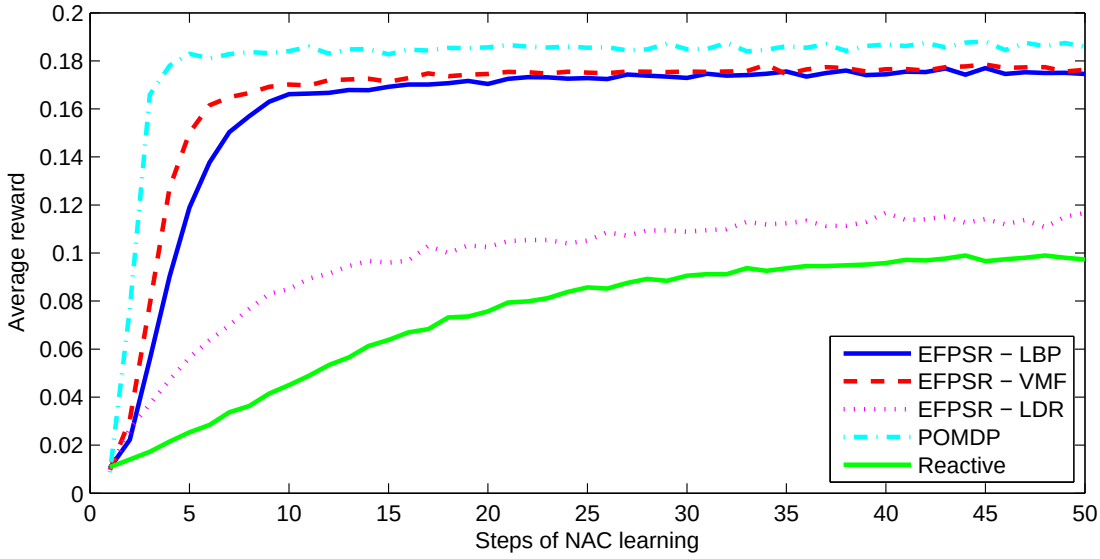


Figure 10.4: Planning results in the cheesemaze domain. Shown are NAC learning curves for five different algorithms. There is one caveat to the results: the learning curve for the LDR inference method has been compressed by a factor of 20 (that is, it ran for 1000 iterations instead of 20). For some reason, the performance improved much more slowly under the model generated with LDR than with other inference methods.

These results basically match exactly the results reported in Section 9.3.2 when using the Exact Linear-Linear EFPSR learning algorithm, and imply that both learning algorithms have learned models that almost identical. We conclude that in this domain, the approximations have worked very well.

Figure 10.5 shows the results of the NAC planner on the Maze 4x3 domain. The panel on the left shows the NAC learning curves for different window sizes, and indicates that for best performance, a window of at least $n = 5$ is necessary. The panel on the right compares the best EFPSR model with using reactive and POMDP states. The performance of the best EFPSR model (average reward of 0.1266) is slightly better than a reactive model (average reward of 0.1191), and is worse than the results obtained using the POMDP state (average reward of 0.1615).

These results also basically match exactly the results reported in Section 9.3.2 when using the Exact Linear-Linear EFPSR learning algorithm, with two caveats: first, the reward obtained here is slightly lower (the Exact algorithm obtained 0.1295, while the Approximate algorithm obtained 0.1266), and second, we had to use $n = 6$ to obtain this result. Even though the performance in this domain is not as good as the performance obtained with the true POMDP model, the performance is only slightly worse than that obtained with the exact model. We therefore conclude that in this domain, the approximations have worked very well.

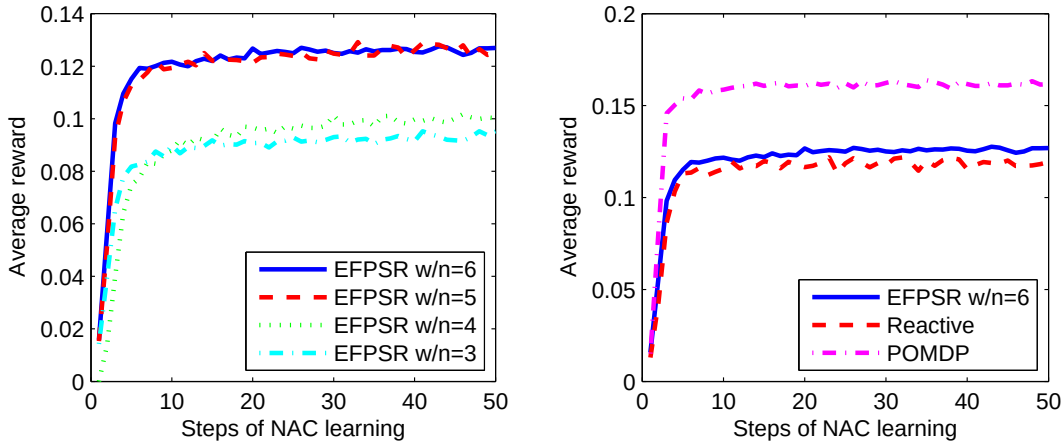


Figure 10.5: Planning results in the Maze 4x3 domain. The left figure shows learning curves for the EFPSR with different window sizes into the future. The best model uses a window of $n = 6$. The right figure compares planning results using the best EFPSR model, the reactive model the true POMDP model.

10.3.3 Bouncing Ball

The next domain we tested on is called the Bouncing Ball domain. In this domain, the observations are factored in a way that is closely related to the dynamics of the system. This domain was hand-crafted to be a larger domain in which the EFPSR would perform well: the domain has significant structure in the observation space, and basically requires the use of a model which is able to capture that structure.

Figure 10.6 describes the domain pictorially. The upper-left figure shows the dynamics of the ball bouncing. At each timestep, the agent observes an 11x10 array of pixels which may be black or white. One of these pixels represents the "ball," which bounces diagonally around the box (shown as a gray trail in the figure). The agent has a single action: an action of 0 means "do nothing," and an action of "1" means "reverse the direction of the ball." The reward signal is shown in the upper-right corner of Figure 10.6. The highest reward is obtained by keeping the ball near the center of the box. This domain is episodic: every 50 timesteps, the ball is reset to a random initial starting configuration.

We define three different versions of the domain. In the noiseless version, the agent sees the exact position of the ball. This domain is second-order Markov, because the position and direction of the ball can be determined from two successive observations. Note that there are only $11 \times 10 = 110$ possible observations in this domain. The second version of the domain adds observation noise: each white pixel has a 1% chance of becoming a black pixel. This has several interesting effects: the new domain is no longer second-order Markov, and the observation space is now exponentially large (there are 2^{110} possible observations, although most of them are highly unlikely). The third version is like the second, except that each pixel has a 10% chance of turning from white to black.

Figure 10.6 also shows the features we used for the EFPSR. For this domain, we hand-coded the features to correspond with the known dynamics. We set $n = 2$. To model $p(F^2|h_t)$, we added singleton features for each observation. Pairwise features were added for each variable to its diagonal neighbors in the next timestep (these features were designed to capture the diagonal motion of the ball). The extended distribution $p(F^3|h_t)$ was modeled with quartets consisting of an action and observation at time t, a diagonal observation at time $t+1$, and a diagonal observation at time $t+2$. There were 584 features describing $p(F^2|h_t)$ and 1,292 features describing $p(F^3|h_t)$.

Given the features, we learned a model using the Approximate Linear-Linear EFPSR, with both timeless gradients and the USV approximation of A, and 100,000 training samples. We then used the resulting states as the input to the NAC planning algorithm. Figure 10.8 shows a representative learning curve for the planner in the 10% noise version of the domain. The horizontal axis represents steps of NAC learning, while the vertical axis represents average reward obtained. Shown is the performance of a uniformly random policy, the best performance obtained using the observation as state in the NAC planner, and the performance of using the EFPSR state in the NAC planner. We see that the EFPSR has a clear performance benefit over both a uniformly random policy and the best reactive policy.

These results hold for both the noiseless and 1% domains as well. Figure 10.8 collects the results for these domains. Again, the EFPSR is able to consistently improve over the best reactive policy, generating a policy with 30% higher reward in the noiseless version, a policy with 25% higher reward in the 1% noisy domain, and a policy with 13% higher reward in the 10% noisy domain. It is an open question as to whether different feature sets would improve these results further.

10.3.4 Robot Vision Domain

Together, the combination of the Linear-Linear EFPSR, the approximate maximum likelihood objective function, and the low-rank decomposition of the parameter matrix allow experimentation on domains with hundreds of observation variables and tens of thousands of features, which is larger than any other model with a predictive representation of state. Here, we apply the entire suite of techniques to the task of visual navigation, where a robot must navigate a maze using nothing but features of camera images as observations. The raw atomic features consist of binary random variables, such as edges, corners, and quantized colors, which are then used by the EFPSR to construct higher-order conjunctions. We also define a reward signal which rewards the agent for navigating to a goal location, and attempt to find a good policy using the NAC planning algorithm.

Vision Domain Setup

Figure 10.9 explains many parts of the setup of the vision domain. The latent state space consists of a position x, y and orientation θ. The agent inhabits a maze with brightly colored walls. The initial observations are 64x64 full color images, which are post-processed to extract binary features. The agent has four actions: move forward, move backward, turn left and turn right.

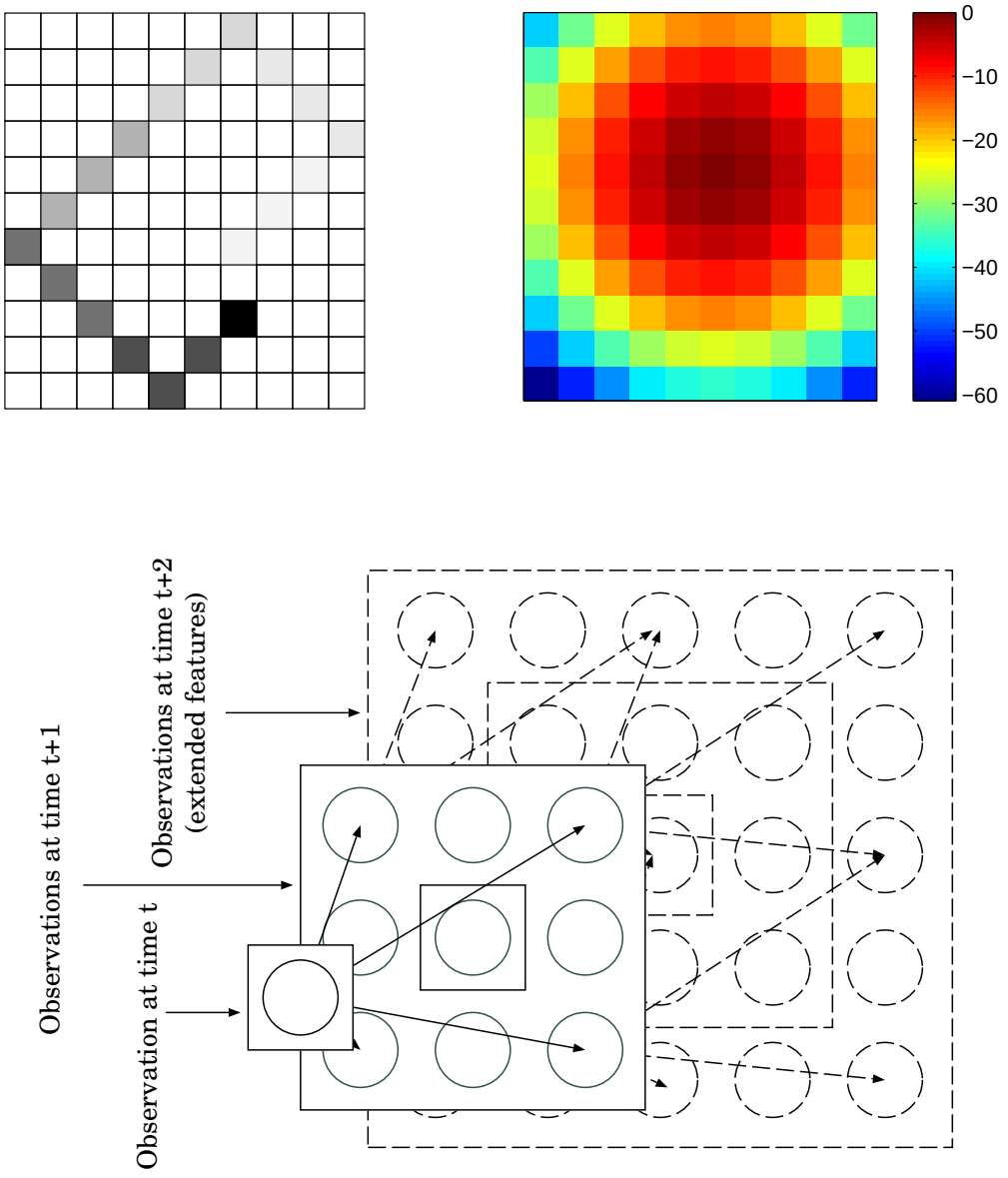


Figure 10.6: The setup of the bouncing ball problem. In the upper left: the dynamics of the ball bouncing. The black square represents the current observation. The gray squares are not observed; they represent the trajectory that the ball has taken. In the upper-right: the reward function. On the bottom: the features used to describe the distribution of the future.

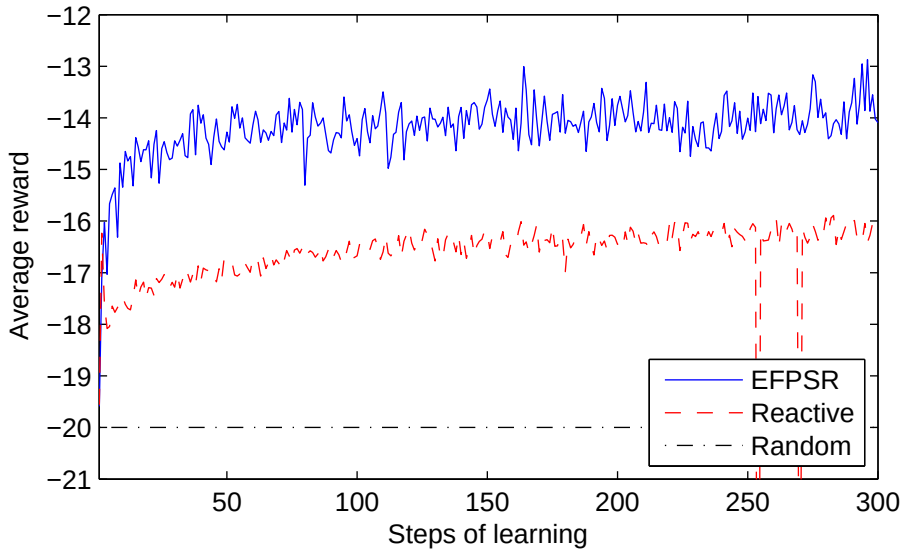


Figure 10.7: A representative learning curve for the bouncing ball domain. This version had maximal (10%) observation noise. The horizontal axis shows steps of NAC learning, while the vertical axis shows average reward (higher is better). The EFPSR yields the best results overall.

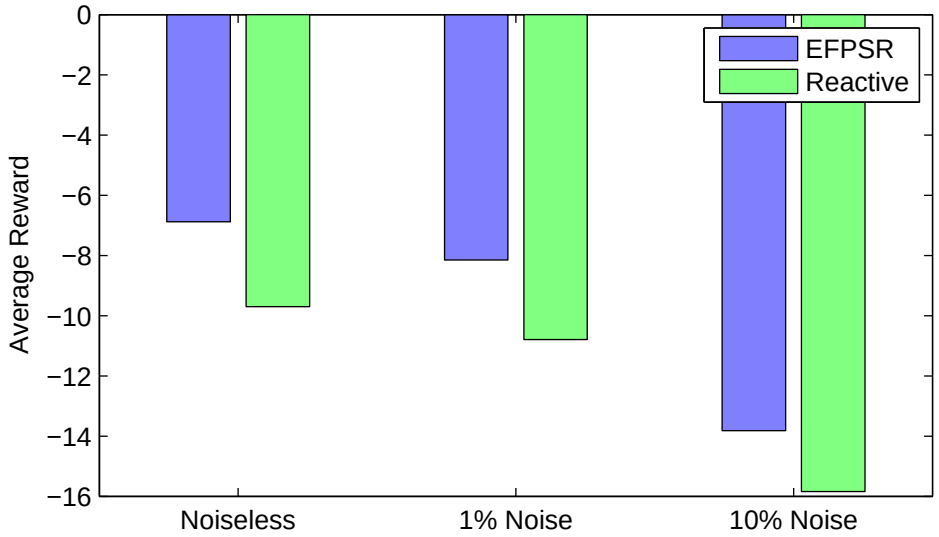


Figure 10.8: Empirical results for the EFPSR on the bouncing ball domain. Each bar represents the average reward of the best policy found.

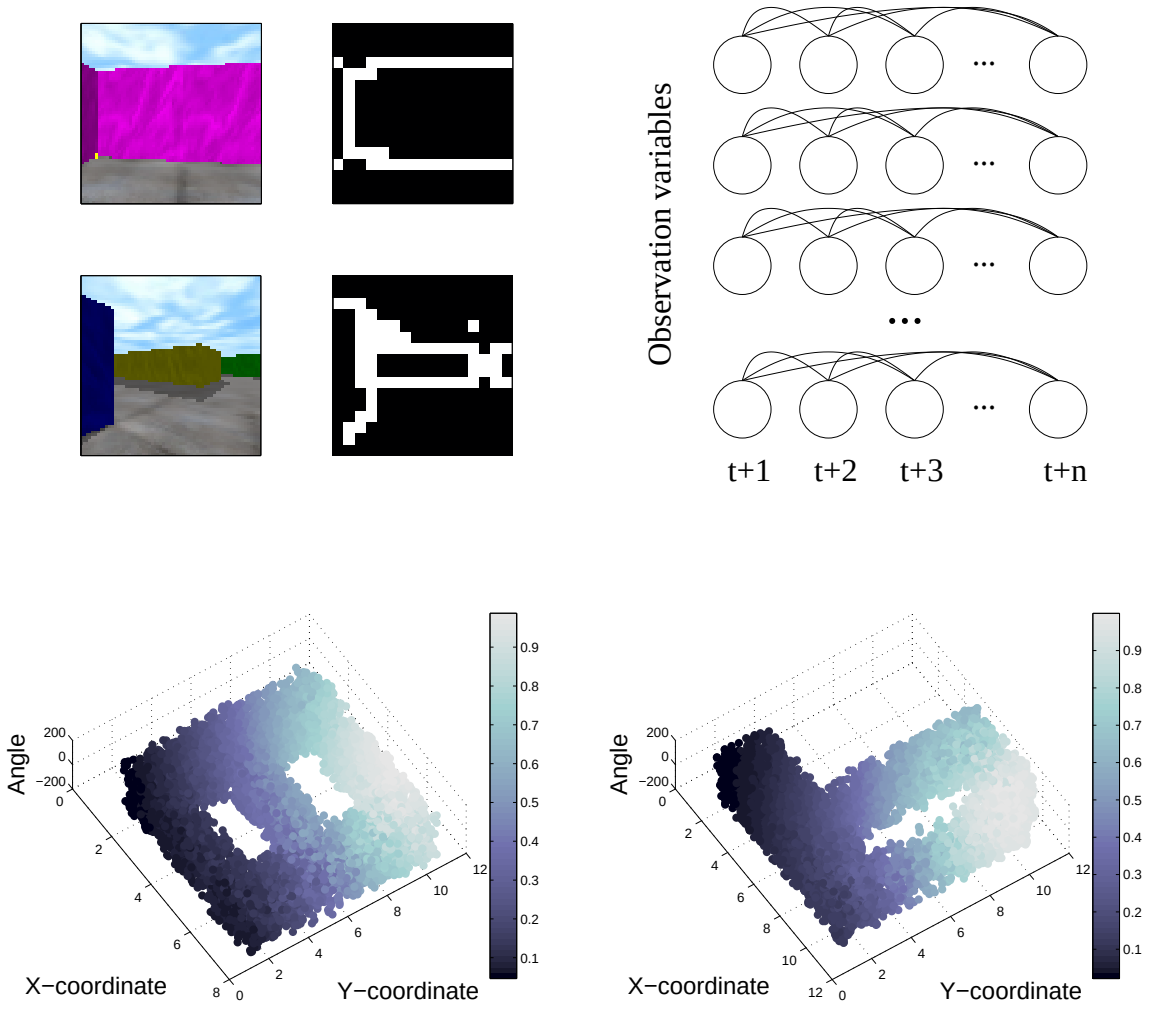


Figure 10.9: Setup of the vision domain. The upper left shows examples of the raw camera images and the extracted 16x16 array of edge features. Color features and corner features are also extracted. The upper-right shows the higher-order features used (which we call "streamer features"). These features connect the same observation variable with temporal successors, but do not connect different variables at all. The bottom row shows reward as a function of the x, y, θ coordinates of the robot for Map #1 (left) and Map #2 (right). Higher is better.

The experiments used two different maps. A stylized version of Maze #1 is found in Figure 3.5, and the outline of each map can be seen in the diagrams representing the reward function in Figure 10.9 (bottom row). Maze #1 had brightly colored walls, which is helpful for determining a location in the maze. Maze #2 also had brightly colored walls, but was designed to have slightly stronger perceptual aliasing than Maze #1.

The experiments used two different sets of binary features. The first set of features consisted of edges, corners and colors. The edge features were extracted as a 16x16 array of binary variables, representing presence or absence of an edge (shown in Figure 10.9, upper-left corner). Corner features were extracted as an 8x8 array of binary variables. Color features were extracted by convolving different regions of the image with a localized Gaussian to extract the dominant color in that region, and then quantizing the color to the nearest of 54 base colors. Each dominant color was encoded as a vector of 54 binary variables, only one of which was active. This feature set resulted in 884 total features.

The second feature set was a post-processed version of the first set. The idea of the second set of features was to create higher-order features which represented things like walls and hallways. To accomplish this, images from the Maze #1 were clustered according to the latent states from which the images were captured, and then the binary features were averaged together to create a sort of filter. New images were tested against each filter, and if the response exceeded an empirically determined threshold, that particular feature was triggered. The images were clustered into 373 groups, meaning that there were 373 atomic features in this feature set. While the images were all taken from Maze #1, they were also used in Maze #2, where the colors, hall geometry, etc. were all different.

We set $n = 3$. For the feature vector $\phi()$, we used "streamer features." Streamer features connect each observation variables to each of its temporal successors, but do not make any connections across different observation variables. This essentially means that each variable is fully factored, and without any dynamics, this would mean that we would be modeling the evolution of each observation variable through time independently. This is not as severe as it may seem, because the state update changes allows variables to depend on each other: the factored distribution of each variable is updated based on the distributions of all of the other variables. Numerous experiments not reported here were conducted with different feature sets, and these features seemed to work as well as any other choice, while creating a relatively compact feature set. With the raw atomic features and the streamer features, there were between 12,000 and 50,000 total features in the final feature set.

Training was done on 200,000 data samples generated with a random policy. A different reward function was defined for each maze, which are shown in Figure 10.9. Both reward functions encourage the agent to navigate to a specific point in the maze, so the resulting problem can be viewed as a shortest path problem.

The experiments also used two different kinds of dynamics. In the "coarse" dynamics, the agent's actions had large effects: turns were 90 degrees, and steps forward and backward moved a full maze unit. This meant that, for example, the agent could navigate from any point to the goal in about 15 steps. In the "fine" dynamics, the agent turned 15 degrees, and moved 0.1 maze units. This results in much smoother looking movements. For both sets of dynamics, n was still set to 3.

After learning a model with the Approximate Linear-Linear EFPSR, we used the NAC algorithm to find a good control policy (experiments with LSPI generally resulted in policy chatter, as discussed previously). For the NAC parameters, we used a TD rate of $\lambda = 0.85$, a stepsize $\alpha = 10.0$, gradient termination test $\epsilon = 0.001$ and remembering factor $\beta = 0.0$. For comparison, we also used NAC to learn a policy using the binary features as state (either feature set #1 or feature set #2, depending on which set was used to train the EFPSR). We call this the "reactive" policy. Finally, we also tested against a random policy.

Vision Domain Results

Learning the model was relatively easy, taking only about 30 seconds to compute the expected feature vectors and optimize the approximate log-likelihood. It took longer to collect the raw data (about five minutes) due to the intensive rendering and image processing. It took quite a long time to learn a good control policy. In order to compute a gradient for the policy parameters, the agent needs to estimate the gradient of the average reward, which is implicitly defined as an expectation over the entire state space. Thus, the agent typically had to wander around the maze for very long periods of time before the gradients converged to a reasonable tolerance. Recall that we used 200,000 training points to learn the model; each NAC step required about 1.5 million steps in the world, and for each step, images must be rendered, features extracted, etc. For the results to be reported, the policies were obtained after about a day of computation.

Figure 10.10 shows the results. There are several points worth noticing in the graphs. As a baseline observation, the random policy obtained approximately the same reward in both domains, regardless of map or dynamics. Higher rewards were obtained in general with coarse dynamics, regardless of map, feature set, or learning algorithm. Presumably, this is because the agent takes bigger steps, and so it does not need to spend as much time traveling to regions of high reward.

It is the difference between the two feature sets that is most interesting, for several reasons. First, note that using feature set #1, the EFPSR consistently performs just under the performance of the reactive policy, regardless of map or dynamics. This could perhaps be explained if the EFPSR was generally unable to capture any meaningful dynamics, and instead learned to predict the identity function, with some noise. This would mean that, for example, the predictions for the next observation would be the same as the current observation, which would effectively result in a policy equivalent to the reactive policy. The story is different with feature set #2. Here, the EFPSR consistently outperforms the reactive policy.

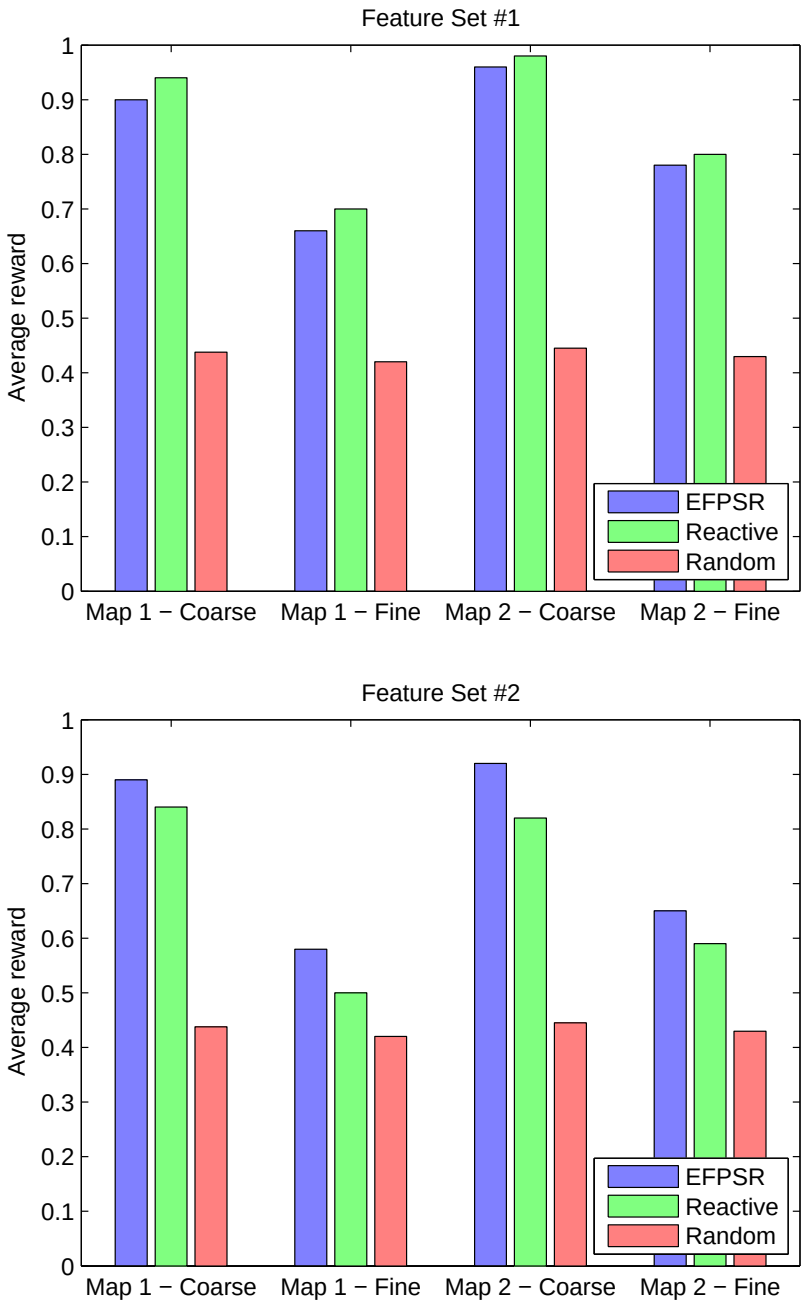


Figure 10.10: Results on the vision domain. The bar charts show the average obtained for different algorithms on different versions of the robot vision domain. Higher is better. The top chart is for feature set #1, and the bottom set is for feature set #2. The performance of the random policy does not depend on the features used, so the same bars are replicated on the top and bottom charts.

Combined with knowledge of our experimental setup, these observations imply a coherent story. It is clear that the performance of the EFPSR is at the mercy of the high order feature conjunctions. In the reported experiments, we used the same set of streamer features, regardless of what the underlying feature set was. The highest order of conjunctions was between four or five.

One plausible explanation for the difference in performance is that low-order conjunctions of more abstract features gives more modeling benefit than low-order conjunctions of granular, low-level features. To see this, consider the following cases:

1. It is easy to imagine that low-order conjunctions of granular features is insufficient to capture useful abstract structure in the domain. For example, to represent the presence or absence of an entire vertical edge, the agent might need a conjunction of 10 features. To represent something hallway-like, the agent might need a conjunction on the order of 50 or so features. Without high order conjunctions of low-level features, it is likely that the agent simply cannot distinguish between a blue wall and a pink wall, because it cannot distinguish walls from a bag of edges. To create higher-order features, we experimented with streamer features, hand-coded features, randomly sampled features, and all possible pairwise features. However, in all of these cases, the highest-order conjunction obtained was four or five. It may be that this is simply not high enough to be useful.

2. This was part of the motivation for feature set #2. Consider the way the clusters were generated, and how they might be capturing more abstract features. Because the camera images were clustered according to latent states, they were typically images of the same thing, from slightly different positions and angles. Imagine two clusters, one set of images which are all looking at the corner of a pink hallway from different angles, and another which are all looking out across an open courtyard. The features in feature set #2 are analogous to asking if the current image is more like cluster (1) or cluster (2). Using this feature set, the highest-order conjunction was still four or five. However, these conjunctions may represent much more abstract kinds of knowledge: if one feature represents “pink wall” and another represents “pink corner,” perhaps a low-order conjunction could express “I’m looking at a pink wall, but if I turn left, I’ll see a pink corner.”

The idea that low-order conjunctions of more abstract features gives more modeling benefit than low-order conjunctions of granular, low-level features suggests several directions for future improvement of these results. It is possible that creating very high-order conjunctions of the low-level features would lead to the best model. To do this, better structure learning algorithms would be needed, because the technique discussed in Section 9.2.1 is difficult to scale well. However, it seems more likely that clustering images as a preprocessing step and extracting more meaningful features will yield better performance, which would encourage research into filters and clusters.

10.4 Conclusions and Future Work

In this chapter, we have presented an approximate maximum likelihood learning algorithm for the Linear-Linear EFPSR model. In many ways, the algorithm has taken a good step towards the goal of modeling large domains. One goal was to eliminate the dependence on T, the size of the training set. Not reflected in these results is the computational savings over the exact algorithm: for all of the models here (including the Robot Vision domain), learning the actual dynamical model only took a few seconds. In contrast, preliminary experiments indicated that computing just one exact gradient for the robot vision domain would have taken days.

The models resulting from this learning algorithm are not perfect. In every case, the approximate learning algorithm performed worse than its exact counterpart. Sometimes, the difference was noticeable, but other times, the two models were virtually indistinguishable. This is perhaps to be expected: it is an approximate algorithm, and is not expected to work perfectly on every domain. While it has been pleasantly surprising how far the approximation has been able to take us, characterizing the circumstances under which the algorithm is expected to work well is an important open problem.

Even though the approximate models perform worse than the exact models, it is only by virtue of these approximations that we were able to attempt at all domains like the Bouncing Ball or the Robot Vision domain. In that sense, the most encouraging results came from the Robot Vision domain. While not always better than a reactive policy, there are some combinations of features and dynamics in which the EFPSR model and the approximate learning algorithms yield better-than-reactive control policies, suggesting that information from history has successfully been incorporated into the state representation. This is a positive result considering the size of the data set and the number of features involved.

Future work needs to address the problem of learning good atomic features and the graphical structure, since this seems to be one of the key factors affecting performance. Additional classes of extension functions should also be considered, possibly with appropriate nonlinearities. Theoretical work needs to be done to determine what types of extension functions are needed for best performance, and algorithmic advances are still possible: for example, it would be nice to find an algorithm with a finer-grained tradeoff between performing T inference calls and performing one inference call per gradient step. It may be possible to use the approximations presented here to learn a coarse model quickly and then refine it. Or models could be minimized by examine the resulting parameter matrix and pruning unused features.

Chapter 11

Concluding Remarks

Models with predictively defined representations of state are still relatively young. To date, there has been little systematic research devoted to pushing these models towards domains with continuous observations, large data sets and/or structured observations. This thesis takes steps in all of those directions. We now look back and summarize where we have been, and present some thoughts on what the future might hold and some conclusions we can draw.

11.1 Review of Contributions

This thesis describes a trajectory of work that explores increasingly complex domains. Our goal at the outset was to explore models with predictively defined representations of state, with the intent of pushing such models towards practical applications. We take a moment to review the domains we have examined, the models we have proposed for them, and their companion learning algorithms.

11.1.1 Continuous PSRs

The Continuous PSR algorithm of Chapter 3 extended PSRs directly to the case of continuous observations by generalizing many aspects of PSRs. One of the most important contributions of this chapter was the introduction of the *system dynamics distributions*, which are a generalization of the system dynamics matrix to the case of continuous observations. Estimating these distributions played a key role in the development of subsequent algorithms. This chapter also presents an information-theoretic framework for addressing the question of finding an approximate representation of state and how to determine the sufficiency of such a representation. Computationally, we showed that the combination of kernel density estimation, quadratic Renyi entropy and Nystrom approximations yields efficient, differentiable expressions for information, which results in a gradient optimizer capable of optimizing state representations. Empirically, we demonstrated that information and modelling error are correlated, and that the representation optimizer is capable of improving randomly chosen state representations.

The Continuous PSR algorithm uses a vector of densities as a state representation. However, there is a stage in the Continuous PSR algorithm where some of the system dynamics distributions (a complete distribution over a window of future observations) must be modeled. This highlights the fact that directly estimating these distributions appears to be the central problem of learning

dynamical systems using predictive representations of state, which led to the development of the Predictive Linear-Gaussian family of algorithms and their generalization, the Exponential Family PSR.

11.1.2 The Predictive Linear-Gaussian Model

Chapter 4 presented the Predictive Linear-Gaussian model (or PLG). This model is largely the work of Matthew Rudary (with some help from the present author), but has served as the jumping-off point for much of the research presented here. The PLG uses a predictive representation of state, defined as the statistics of a Gaussian distribution over a finite window of future observations – in essence, the PLG assumes the system dynamics distributions are Gaussian. Importantly, the PLG is formally equivalent to the Kalman Filter, in the sense that it has an equal number of parameters and predicts the same distribution over future observations given any history. The learning algorithms are simple, consisting largely of linear regressions and sample statistics, and in some cases have attractive theoretical guarantees. Rudary's empirical results demonstrated that when learning PLG models from data, the PLG typically outperforms LDS models learned with the EM algorithm.

11.1.3 The Kernel PLG

The PLG is capable of modeling only linear domains. Chapter 5 extended the PLG to model nonlinear dynamical systems by using the kernel trick. The resulting Kernel PLG model can be interpreted as performing linear dynamics in a high-dimensional, nonlinear feature space. We also contributed a learning algorithm which, like the PLG, consists mainly of regressions and sample statistics. The state update requires some inference on the model, which can be performed exactly (although not efficiently) in the case of a Gaussian kernel. Computationally, we introduced sigma-point approximations to perform the inference needed by the dynamics, which allowed the model to be easily extended to any kernel, and which has the side effect of relating the KPLG to the Unscented Kalman Filter. Empirically, this model outperformed the PLG and a kernel autoregressive model on several nonlinear test problems, and is competitive on other problems.

11.1.4 Mixtures of PLGs

An alternative way to leverage the PLG framework to model nonlinear dynamical systems is to consider dynamics that are piecewise linear. Chapter 6 presented the MPLG (or "Mixture of PLGs") model, which uses a probabilistic generative model to mix different PLGs together to form a composite dynamical system. Because the mixing method has probabilistic semantics, the learning algorithm for the MPLG has a simple form which can be interpreted as a weighted version of the PLG learning algorithm. Again, inference is needed during the state update, but no exact expressions are available. Computationally, we contributed an extension of the sigma-point approximation used by the KPLG to create a hybrid particle-analytical inference method, where nonlinear terms of the state update are approximated using sigma-points, but linear portions are computed exactly. Empirically, we demonstrated that the MPLG outperforms the KPLG on the test domains, and that

it demonstrates more robustness to parameter perturbations.

11.1.5 The Exponential Family PSR

Chapters 7 - 10 presented the Exponential Family PSR (EFPSR) model, which combines and generalizes many of the ideas from previous models. Instead of defining state as the parameters of a Gaussian over a window of the future, it uses a general exponential family distribution. The flexibility of the exponential family distribution and the genericity of the state update mechanism permits the unification of the PSR, the PLG, the KPLG and the MPLG as specializations of the EFPSR, which lead to theorems about the representational capacity of the EFPSR in Chapter 7. It additionally connects PSRs to graphical models.

Chapter 8 presented the Information PLG, which unifies the PLG, the Information Kalman Filter and the EFPSR. By choosing the features and extension equations carefully, we showed that the EFPSR *is* the information form of the PLG. We also showed that state updates can be performed efficiently using nothing but the ordinary PLG parameters.

Chapter 9 presented the Exact Linear-Linear EFPSR, which is a special case of the EFPSR model designed to be tractable. We presented an exact maximum likelihood learning algorithm, which is intractable for all but the smallest domains. Even so, we demonstrated empirically that the model can capture several benchmark POMDP domains, and that the state representation generated by the EFPSR is useful for planning in a traditional reinforcement learning sense, in that it can be used to learn almost optimal control policies for some domains.

Chapter 10 presented the Approximate Linear-Linear EFPSR, which invoked several approximations to help the Exact Linear-Linear EFPSR scale to larger domains. These domains had tens of thousands of features (which is far larger than any other model with a predictive representation of state) and large training data sets. Using this suite of techniques, and in conjunction with the natural actor-critic planning algorithm, we demonstrated that there are situations where the EFPSR can successfully incorporate history into its state representation, which translates into improved control performance. The largest such domain is a visual navigation task, where a robot must navigate a maze using nothing but camera images as observations.

11.2 Conclusions and Future Work

What broad conclusions can we draw from the work presented here? The first conclusion is that the idea of predictively defined representations of state is a flexible one, and that many variations seem to result in learnable models which capture their domains well. We have considered a variety of types within the umbrella of predictions about the future: we have used probabilities of specific tests (the PSR model), densities of specific tests (the Continuous PSR model), expectations of features of the short-term future (the PLG family of models), and the natural parameters of a distribution over a window of short-term future observations (the EFPSR). These appear to have worked well, and

there are likely many other possibilities.

What can we conclude about the idea of predictively defined state itself? We began this thesis by noting that our restriction to models with predictively defined state was self-imposed, and we wondered: would this be a limitation? What would we gain? Our conclusion is that this restriction does not appear to have limited the capacity of our models. Empirically, we have often outperformed other approaches, and even when we did not, we were competitive. There have been positive theoretical results as well. For example, the fact that PSRs are formally more expressive than POMDPs, the fact that every LDS can be captured by a PLG, and the fact that EFPSRs can capture a variety of models all give strong reasons to believe that we may never lose anything by representing state as statistics about the short-term future (although proving this in complete generality would be a significant contribution). Our conclusion is this: so far, there is no known representational or empirical limitation that has resulted from adopting a predictively defined representation of state. The future for predictively defined state seems bright.

Where to from here? Further theoretical work is clearly needed: are there any unique advantages to predictively defined state? Will there ever be a limitation? When are our approximations expected to work well? Algorithmic advances are crucial: further approximations or computational improvements are possible, and completely different learning strategies could also be fruitful (perhaps not based on maximum likelihood, or designed to take advantage of domain-specific structure). And of course, applications to ever larger domains are needed: richer, more structured domains than those we have considered may influence models and learning algorithms.

Throughout this thesis, we have exercised discipline in never allowing our representations to use latent variables. This is part of the theory of predictively defined state representations, and this thesis is an attempt to take that theory seriously. However, for practical applications, where the goal is to solve a problem rather than to investigate a theory, it makes sense to marry both approaches. This is a rich source of possibilities: for example, the ability to add structured domain knowledge (possibly in the form of a prior in a Bayesian framework) seems important. Every model we have presented could be combined with latent variables in one form or another, and the result would sit somewhere in between classical state-space models and predictively defined models. How this would change the models' representational capacity, their learning algorithms, or their empirical performance is an open question.

Appendices

Appendix A

Computing the Gradients of Information

In this appendix, we discuss how the Continuous PSR algorithm of Chapter 3 computes the gradient of information with respect to the parameters governing its state representation.

As discussed in Chapter 3, we assume that we have been given data. This data comes in the form of n triples (h_i, s_i, f_i). The samples h_i and f_i are from the joint distribution $p(F, H)$, and $s_i = f(h_i; \theta)$ is the state corresponding to each history. Thus, the samples s_i and f_i are samples from $p(F, S)$. Recall that using these samples, we will infer the distributions using kernel density estimation with a Gaussian kernel:

$$p(X = x) = \frac{1}{n} \sum_{j=1}^{n} G(x - x_j; \sigma_j).$$

The Continuous PSR algorithm frames the problem of discovering a good set of core tests as an optimization problem. The objective function is $I(F, S = f(H; \theta))$, which must be maximized over θ. The algorithm searches the space of possible θ by selecting an initial θ and then performing gradient ascent:

$$\theta = \theta + \eta \frac{\partial I}{\partial \theta} = \theta + \eta \sum_{i}^{n} \frac{\partial I}{\partial s_i} \frac{\partial s_i}{\partial \theta}$$

In this appendix, we derive the two quantities necessary to compute the gradients: $\partial I / \partial s_i$ and $\partial s_i / \partial \theta$.

A.1 Information with Respect to State Samples

We begin by taking derivatives of information with respect to individual state samples:

$$\begin{aligned} \frac{\partial I}{\partial s_i} &= \frac{\partial H(F)}{\partial s_i} + \frac{\partial H(S)}{\partial s_i} - \frac{\partial H(F, S)}{\partial s_i} \\ &= \frac{\partial H(S)}{\partial s_i} - \frac{\partial H(F, S)}{\partial s_i} \end{aligned}$$

This will result in a vector describing which way sample i wishes to move in order to increase information. The entropy measure that we will use is quadratic Renyi entropy:

$$H(X) = -\log \int p(X)^2 dx$$

$$
\begin{aligned}
&= -\log \int \left(\frac{1}{n} \sum_{j}^{n} G(X - x_j; \sigma_j^X) \right)^2 dX \\
&= -\log \frac{1}{n^2} \sum_{ij}^{n} G(x_i - x_j; \sigma_i^X + \sigma_j^X) \qquad \text{(A.1)}
\end{aligned}
$$

where the second line follows because we are using kernel density estimation with a Gaussian kernel to estimate $p(X)$, and where the third lines follows because we have used the identity

$$\int G(X - x_i; \sigma_i) G(X - x_j; \sigma_j) dX = G(x_i - x_j; \sigma_i + \sigma_j).$$

Similarly, the entropy of a joint density can be written as:

$$H(X, Y) = -\log \frac{1}{n^2} \sum_{ij}^{n} G(x_i - x_j; \sigma_i^X + \sigma_j^X) G(y_i - y_j; \sigma_i^Y + \sigma_j^Y). \qquad \text{(A.2)}$$

Our choices of entropy and density estimator have yielded a closed-form expression for entropy which has reduced to computing pairwise interactions between the data points used to construct the density. There is no approximation in this integral, apart from the use of a kernel density estimate to begin with.

Another useful identity involves the derivative of a Gaussian:

$$\frac{\partial}{\partial x_i} G(x_i - x_j; \Sigma) = -G(x_i - x_j; \Sigma)(\Sigma^{-1})(x_i - x_j)$$

which we use to find the gradients of information:

$$
\begin{aligned}
\frac{\partial H(S)}{\partial s_i} &= \frac{\partial}{\partial s_i} \left[-\log \frac{1}{n^2} \sum_{ij}^{n} G(s_i - s_j; \sigma_i^S + \sigma_j^S) \right] \\
&= \frac{-1}{H(S)} \frac{1}{n^2} \frac{\partial}{\partial s_i} \left[\sum_{ij}^{n} G(s_i - s_j; \sigma_i^S + \sigma_j^S) \right] \\
&= \frac{2}{H(S) n^2} \sum_{j}^{n} G(s_i - s_j; \sigma_i^S + \sigma_j^S)(\sigma_i^S + \sigma_j^S)^{-1}(s_i - s_j).
\end{aligned}
$$

The result is obtained because the sample s_i appears twice in the double summation over i, j. Similarly,

$$\frac{\partial H(F, S)}{\partial s_i} = \frac{2}{H(F, S) n^2} \sum_{j}^{n} G(s_i - s_j; \sigma_i^S + \sigma_j^S) G(f_i - f_j; \sigma_i^F + \sigma_j^F)(\sigma_i^S + \sigma_j^S)^{-1}(s_i - s_j).$$

A.2 State Sample with Respect to Parameters

Finally, we need to compute the change in sample s_i with respect to the parameters θ. Up to this point, none of the math that we have laid out says anything about predictive representations of state, and applies equally well to any parametric mapping from past to state. We will now introduce the choices that make this a PSR. Recall that $s_i = f(h_i; \theta)$. s_i is a vector, the j'th component of which is the prediction of test t_j:

$$\begin{aligned} s_i^j &= p(t_j|h_i) = \frac{p(h_i, t_j)}{p(h_i)} \\ &= \frac{\frac{1}{n}\sum_k^n G(h_i - h_k; \sigma_k^H) G(t_j - f_k; \sigma_k^F)}{\frac{1}{n}\sum_l^n G(h_i - h_l; \sigma_l^H)} \\ &= \sum_k^n N(h_i)_k G(t_j - f_k; \sigma_k^F) \end{aligned}$$

where we have summarized the conditioning of the past into a function N (which only depends on h_i, and not on t_j).

The parameters θ are the actions and observations within the tests themselves. We can compute the partial derivative of a given state variable with respect to the test values that generated it:

$$\frac{\partial s_i^j}{\partial t_j} = -\sum_k^n N(h_i)_k G(t_j - f_k; \sigma_k^F)(\sigma_k^F)^{-1}(t_j - f_k).$$

This completes the derivation.

Appendix B

Approximate Information Gradients Using Nystrom Approximations

The Continuous PSR algorithm requires the gradient of information with respect to the test parameters. Naively done, this operation is quadratic in the number of samples used. Here, we present an approximation to the gradient calculations which uses Nystrom approximations to speed up the gradient calculation. This approximation is tunable, meaning that we can trade-off computation time for approximation accuracy, and allows us to scale up the procedure to large datasets.

This appendix is divided into two broad sections. First, in Section B.1 we review the assumptions of the continuous PSR algorithm: that we are using a kernel density estimate with a Gaussian kernel, and a measure of information based on quadratic Renyi entropy, and show how this leads to a closed form expression for entropy.

We then discuss general Nystrom approximations in Section B.2. In Section B.3 we show how to use Nystrom approximations to approximate entropy, and give formulae for the derivative of entropy with respect to one of the samples which composes the density estimate. Finally, in Section B.4 we combine everything together to show how to compute the full gradients of information which are needed by the Continuous PSR algorithm.

B.1 Closed Form Entropy

First, we recall a few of the key definitions in the Continuous PSR model, but we will do so in a generic way. We wish to estimate $p(X)$ given samples $x_1, \cdots, x_n$, with each $x_i \in \mathbb{R}^{d\times 1}$. We choose to use a kernel density estimate with a spherical Gaussian kernel:

$$p(X = x) = \frac{1}{n}\sum_{j=1}^{n} G(x - x_j; \sigma_j)$$

where

$$G(x; \sigma_j) = \frac{1}{\sqrt{(2\pi\sigma_j)^d}} \exp\left\{-x^\top x/2\sigma_j\right\}$$

is the kernel.

Ultimately, we will be interested in computing the information that X conveys about another random variable. This will involve computing an entropy term. Computing Shannon entropy in closed-form

for our density estimate is not tractable, but we *can* compute a generalized entropy quantity, known as quadratic Renyi entropy, in closed form.

Quadratic Renyi entropy for a random variable X is defined as:

$$H_{R_2}(X) = -\log \int p(X)^2 dX \tag{B.1}$$

Plugging in our kernel density estimate yields the following:

$$\begin{aligned} H(X) &= -\log \int p(X)^2 dx \\ &= -\log \frac{1}{n^2} \sum_{ij}^{n} G(x_i - x_j; \sigma_i + \sigma_j). \end{aligned} \tag{B.2}$$

While this expression has a simple closed form, it unfortunately has quadratic sample complexity – we must compute the Gaussian of every sample with every other sample. This $O(n^2)$ scaling makes this approach infeasible for large datasets. However, this term may be approximated, as we discuss next.

B.2 Nystrom Approximations

One of the keys to scalability is the observation that the Gaussian used in our kernel density estimator is not only a kernel in the local modeling sense (Fan and Gijbels, 1996), it is also a Mercer kernel. This means that it is a positive-definite quantity. We will now explain how a so-called *Nystrom approximation* can be used to approximate any positive-definite quantity, and how its use leads to several computational efficiencies for the Continuous PSR model.

Let $G \in \mathbb{R}^{n \times n}$ be any symmetric, positive definite matrix. In our case, G results from the application of a Gaussian kernel: $G_{ij} = G(x_i - x_j; \sigma)$. Nystrom approximations work by selecting a number of *landmark* points (sometimes known as *dictionary* points), and instead of computing a Gaussian of every point with every other point (G_{ij}), we only compute the Gaussians with respect to a set of landmarks.

Select m of the x_i's at random as landmarks, and without loss of generality, permute them to be the first m data points. The Nystrom approximation of G is given by:

$$\begin{aligned} G &= \begin{bmatrix} A & B \\ B^\top & C \end{bmatrix} \\ &\approx \begin{bmatrix} A \\ B^\top \end{bmatrix} \left(A^{-1}\right) \begin{bmatrix} AB \end{bmatrix} \\ &= \begin{bmatrix} A & B \\ B^\top & B^\top A^{-1} B \end{bmatrix} \end{aligned}$$

where $A \in \mathbb{R}^{m \times m}$ is the kernel matrix associated with the m landmarks. This is essentially a low-rank approximation to G, where it is assumed that every non-landmark point can be expressed as a linear combination of landmarks. This approximation is exact when G is rank m or less; otherwise, the quality of the approximation is proportional to $\|C - B^\top A^{-1} B\|$ (Platt, 2004). The Gaussian is an infinite-dimensional feature extractor: assuming that none of the points are equal, when G is derived from the Gaussian kernel, it will always have full rank. Nystrom approximations also have an interpretation as an eigenvector approximation method, where the eigenvectors of A are extrapolated to approximate the eigenvectors of G (Platt, 2004).

Using a Nystrom approximation, we can simplify expressions of the form

$$E = \sum_{ij}^{n} G(x_i - x_j; \Sigma)$$

which have quadratic sample complexity as follows:

$$\begin{aligned}
E &= \sum_{ij}^{n} G(x_i - x_j; \sigma) \\
&= 1^\top G 1 \\
&= 1^\top \begin{bmatrix} A \\ B^\top \end{bmatrix} \left(A^{-1}\right) [AB]\, 1 \\
&= \left(1^\top \begin{bmatrix} A \\ B^\top \end{bmatrix}\right) \left(A^{-1}\right) \left([AB]\, 1\right) \\
&= \sum_{kl}^{m} \left(\sum_{i}^{n} G(x_i - l_k; \sigma)\right) \left(A^{-1}\right)_{kl} \left(\sum_{j}^{n} G(l_l - x_j; \sigma)\right)
\end{aligned} \tag{B.3}$$

where $1 \in \mathbb{R}^{n \times 1}$ is a column vector of ones.

The essential insight is the fact that the operations can be grouped in a way that results in lower overall complexity. The complexity of computing E using this method is $O(m^3 + nm)$, where n is the total number of samples and $m \ll n$ is the number of landmark points. In contrast, the complexity of the naive version is $O(n^2)$. For many of the robot experiments to be explained later for example, about $n = 100,000$ samples were used (which would be totally intractable), but for which good results were obtained with as few as $m = 100$ landmark points.

Landmark points can be selected in a variety of ways. One particularly appealing way is the dictionary-based methods of Engel et al. (2003), which use a tunable "approximate linear dependence threshold." By changing the threshold, the number of landmarks can be tuned. The method automatically ensures that the landmarks are linearly independent, and therefore that A is invertible.

B.3 Combining Entropy Gradients with Nystrom Approximations

With our Nystrom approximation in hand, we now turn to the calculation of the Renyi entropy of a random variable X. We wish to compute Eq. A.1, which has quadratic sample complexity. Using the Nystrom approximation in Eq. B.3, we may approximate the Renyi entropy as:

$$\begin{aligned}
H(X) &= -\log \frac{1}{n^2} \sum_{ij}^{n} G(x_i - x_j; \sigma) \\
&\approx -\log \frac{1}{n^2} \sum_{ij}^{n} \sum_{kl}^{m} G(x_i - l_k; \sigma) \left(A^{-1}\right)_{kl} G(l_l - x_j; \sigma) \\
&= -\log \frac{1}{n^2} \sum_{kl}^{m} \left[\sum_{i}^{n} G(x_i - l_k; \sigma)\right] \left(A^{-1}\right)_{kl} \left[\sum_{j}^{n} G(l_l - x_j; \sigma)\right] \\
&= -\log \frac{1}{n^2} \sum_{kl}^{m} B_k M_{kl} B_l \\
&= -\log \frac{1}{n^2} B^\top M B
\end{aligned}$$

where B is a vector $\in \mathbb{R}^{m \times 1}$, and

$$B_k = \sum_{i}^{n} G(x_i - l_k; \sigma),$$

and the matrix

$$M = A^{-1}$$

is $\in \mathbb{R}^{m \times m}$. The vector B contains the kernel evaluation of every sample x_i with every landmark, while M is the inverse of A, which contains the kernel evaluation of every landmark with every other landmark.

We can now compute the derivative of $H(X)$ with respect to one of the samples composing it:

$$\begin{aligned}
\frac{\partial H(X)}{\partial x_i} &= \frac{\partial}{\partial x_i} \left[-\log \frac{1}{n^2} B^\top M B\right] \\
&= -\frac{1}{H(X)} \frac{1}{n^2} \frac{\partial}{\partial x_i} \left[B^\top M B\right] \\
&= -\frac{1}{H(X)} \frac{1}{n^2} \left[\frac{\partial}{\partial x_i} B\right]^\top (2M) B \\
&= -\frac{1}{H(X)} \frac{1}{n^2} (B^i)(2M) B
\end{aligned} \tag{B.4}$$

where we define B^i to be a vector $\in \mathbb{R}^{d \times m}$, representing the derivatives of each B_k with respect to x_i. The kth column of B^i is defined as

$$B_k^i = -G(x_i - l_k; \sigma) 2\sigma(x_i - l_k).$$

B.4 Specializing to the Case of Information Gradients

We now specialize the presentation of the previous two sections to the case of full information gradients. Recall that information is defined as a sum of entropies:

$$I(F; S = f(H; \theta)) = H(F) + H(S) - H(F, S).$$

The term $H(F)$ does not depend on θ, so we can ignore it. We can specialize Eq. B.4 to the case of computing $H(S)$ trivially, and so we focus on computing $\partial H(F,S)/\partial s_i$. To simplify the notation, we will not show the variance parameters $(\sigma_i^S + \sigma_j^S)$ and $(\sigma_i^F + \sigma_j^F)$ for the Gaussian kernels:

$$\frac{\partial H(F,S)}{\partial s_i} = \frac{\partial}{\partial s_i}\left[-\log \frac{1}{n^2}\sum_{ij}^{n} G(s_i - s_j)G(f_i - f_j)\right] \tag{B.5}$$

$$\approx \frac{2}{H(F,S)n^2} B^i M B. \tag{B.6}$$

where B is a vector $\in \mathbb{R}^{m \times 1}$ with

$$B_k = \sum_{i}^{n} G(s_i - l_k^s)G(f_i - l_k^f).$$

The vector B^i is a matrix $\in \mathbb{R}^{d \times m}$, with the k'th column defined as

$$B_k^i = \sum_{i}^{n} G(s_i - l_k^s)G(f_i - l_k^f)(\sigma_i^S + \sigma_j^S)(s_i - l_k^s).$$

We arrive at this result by first approximating the Gaussians as

$$\sum_{ij}^{n} G(s_i - s_j)G(f_i - f_j) \approx \sum_{kl}^{m}\left(\sum_{i}^{n} G(s_i - l_k^s)G(f_i - l_k^f)\right)(A^{-1})_{kl}\left(\sum_{j}^{n} G(s_j - l_l^s)G(f_j - l_l^f)\right)$$

which is the usual Nystrom approximation, and $M = A^{-1}$, and A is the kernel matrix of landmarks. We have applied the Nystrom approximation based on the fact that a Gaussian multiplied by a Gaussian is still a Gaussian. Conceptually, the actual Nystrom approximation part of this equation is exactly the same as treating samples of (s_i, f_i) as higher-dimensional joint samples (that is, the landmarks now exist in the joint space of (s, f)). However, because we are only taking derivatives with respect to the s_i component of the joint vectors, the result involves more s terms than f terms.

Appendix C

Derivation of the KPLG State Extension Equations

Here we present detailed derivations for the state extension equations of the KPLG with the Gaussian kernel. Recall that we assume that the Gaussian kernel is fully normalized (meaning we can use it as a PDF).

Before we begin we will present an important preliminary lemma, which is a standard result about products of Gaussians:

$$(a-b)^\top (A)^{-1}(a-b) + (b-c)^\top (B)^{-1}(b-c) = (a-c)^\top (C)^{-1}(b-c) + (b-d)^\top (D)^{-1}(b-d)$$

where

$$C = A + B$$

$$D = A^{-1} + B^{-1} = A(A+B)^{-1}B = B(A+B)^{-1}A$$

$$d = A(A+B)^{-1}b + B(A+B)^{-1}c.$$

This fact, coupled with standard identities on determinants, yields the following important fact:

$$K(a,b;A)K(b,c;B) = K(a,c;C)K(b,d;D).$$

We use this identity to express the essence of the conjugacy between our state (which is a Gaussian random variable) and our basis functions (which are also Gaussians) – it will effectively allow us to factor out key terms from integrals, thus making their analysis tractable. It therefore forms the backbone of all the following derivations.

We will now show in detail how to compute two of the key terms in the derivation of our models; the rest of the terms are easily computed by analogy to them. The first term is perhaps the most important, and gives the flavor of all the others. This result states that the expected value of the kernel is the kernel of the expected value, with added variance:

$$\begin{aligned} \mathrm{E}[K(\xi_j, F^n; \phi_j)] &= \int K(\xi_j, F^n; \phi_j) p(F^n) dF^n \\ &= \int K(\xi_j, F^n; \phi_j) K(F^n, \mu_t; \Sigma_t) dF^n \end{aligned}$$

$$
\begin{aligned}
&= K(\xi_j, \mu_t; \phi_j + \Sigma_t) \int K(F^n, \mu'_{tj}; \Sigma'_{tj}) dF^n \\
&= K(\xi_j, \mu_t; \phi_j + \Sigma_t) \qquad \text{(C.1)}
\end{aligned}
$$

where

$$
\begin{aligned}
\Sigma'_{tj} &= \Sigma_t^{-1} + \phi_j^{-1} \\
\mu'_{tj} &= \phi_j(\Sigma_t + \phi_j)^{-1}\mu_t + \Sigma_t(\Sigma_t + \phi_j)^{-1}x_j.
\end{aligned}
$$

The last line of Eq. C.1 follows because the integral is over an entire PDF with unit volume. As a corollary, it is easy to show that

$$
\mathrm{E}[K(\xi_j, F^n; \phi_j)F^n] = K(\xi_j, \mu_t; \phi_j + \Sigma_t)\mu'_{tj} \qquad \text{(C.2)}
$$

because the integral in the penultimate line of Eq. C.1 will become the expected value of F^n.

At a later point, we will also need to be able to compute the covariance of the kernel with the noise term η_{t+n+1}. This is more complicated, and relies on a subtle insight. Recall that the noise term and F^n effectively form a jointly Gaussian random variable:

$$
\begin{pmatrix} \eta_{t+n+1} \\ F^n \end{pmatrix} \sim N\left[\begin{pmatrix} 0 \\ \mu_t \end{pmatrix}, \begin{pmatrix} \sigma^2 & C^\top \\ C & \Sigma_t \end{pmatrix} \right]
$$

We immediately see that $E[\eta_{t+n+1}|F^n] = C^\top\Sigma_t^{-1}(F^n - \mu_t)$ due to standard results on multivariate Gaussian random variables. This allows us to rewrite the following integral in a simpler way, and thus to solve it simply (we drop the time subscripts in the following for clarity):

$$
\begin{aligned}
\mathrm{E}[K(\xi_j, F; \phi_j)\eta] &= \int\int K(\xi_j, F; \phi_j)\ \eta\ p(F, \eta)\ d\eta\ dF \\
&= \int\int K(\xi_j, F; \phi_j)\ \eta\ p(F)\ p(\eta|F)\ d\eta\ dF \\
&= \int K(\xi_j, F; \phi_j)\ p(F) \left(\int \eta p(\eta|F) d\eta \right) dF \\
&= \int K(\xi_j, F; \phi_j)\ p(F)\ C^\top\Sigma_t^{-1}(F - \mu)\ dF \\
&= C^\top\Sigma_t^{-1} \int K(\xi_j, F; \phi_j) F p(F) dF \\
&\quad -C^\top\Sigma_t^{-1} \int K(\xi_j, F; \phi_j)\ p(F)\ dF\ \mu \\
&= C^\top\Sigma_t^{-1} K(\xi_j, \mu_t; \phi_j + \Sigma)(\mu'_j - \mu)
\end{aligned}
$$

We can perform an analysis similar to the two preceding ones and generate many identities which are necessary for the derivation of our model. These identities are summarized below, letting $K_{tj} =$

$K(\xi_j, F^n; \phi_j)$:

$$\begin{aligned}
\mathrm{E}[K_{tj}] &= K'_{tj} \\
\mathrm{E}[K_{tj}\eta_{t+n+1}] &= K'_{tj} C^\top \Sigma_t^{-1}(\mu'_{tj} - \mu) \\
\mathrm{E}[K_{tj}K_{ti}] &= K^\dagger_{tij}
\end{aligned}$$

where

$$\begin{aligned}
K'_{tj} &= K(\xi_j, \mu_t; \phi_j + \Sigma_t) \\
\mu'_{tj} &= \phi_j(\Sigma_t + \phi_j)^{-1}\mu_t + \Sigma_t(\Sigma_t + \phi_j)^{-1}\xi_j \\
K^\dagger_{tij} &= K(\mu^\dagger_{tij}, \mu_t; \Sigma_t + \Sigma^\dagger_{tij})K(\xi_i, \xi_j; \phi_i + \phi_j) \\
\mu^\dagger_{tij} &= \phi_i(\phi_i + \phi_j)^{-1}\xi_j + \phi_j(\phi_i + \phi_j)^{-1}\xi_i \\
\Sigma^\dagger_{tij} &= \phi_i(\phi_i + \phi_j)^{-1}\phi_j.
\end{aligned}$$

We are now ready to compute the three principal terms needed for the KPLG state update:

$$\begin{aligned}
\mathrm{E}[O_{t+n+1}] &= \mathrm{E}[\sum_j \alpha_j K(\xi_j, F^n; \phi_j) + \eta_{t+n+1}] \\
&= \sum_j \alpha_j \mathrm{E}[K(\xi_j, F^n; \phi_j)] + \mathrm{E}[\eta_{t+n+1}] \\
&= \sum_j \alpha_j K'_{tj}
\end{aligned}$$

because η_{t+n+1} is mean-zero. Similarly,

$$\begin{aligned}
\mathrm{Cov}[O_{t+n+1}, F^n] &= \mathrm{E}[O_{t+n+1}F^{n\top}] - \mathrm{E}[O_{t+n+1}]E[F^{n\top}] \\
&= \sum_j \alpha_j \mathrm{E}[K(\xi_j, F^n; \phi_j)F^{n\top}] + \mathrm{E}[\eta_{t+n+1}F^{n\top}] \\
&\quad - \sum_j \alpha_j \mathrm{E}[K(\xi_j, F^n; \phi_j)]\mathrm{E}[F^{n\top}] \\
&= \sum_j \alpha_j K'_{tj}(\mu'_{tj} - \mu_t)^\top + C^\top.
\end{aligned}$$

The computation of the variance is only slightly harder. To clarify notation, we will drop time subscripts. Let $S = \sum_j \alpha_j K(\xi_j, F; \phi_j)$. We have

$$\begin{aligned}
\mathrm{E}[(O - E[O])^2] &= \mathrm{E}[(S + \eta - \mathrm{E}[S])^2] \\
&= \mathrm{E}[S^2] - \mathrm{E}[S]^2 + 2\mathrm{E}[S\eta] + \mathrm{E}[\eta^2] \\
&= \sum_i \sum_j \alpha_i \alpha_j \mathrm{E}[K(\xi_i, F; \phi_i)K(\xi_j, F; \phi_j)] - \mathrm{E}[S]^2
\end{aligned}$$

$$
\begin{aligned}
& +2\sum_j \alpha_j \mathrm{E}[K(\xi_j, F; \phi_j)\eta] + \sigma^2 \\
= \quad & \sum_i \sum_j K^{\dagger}_{tij}\alpha_i\alpha_j - E_t^2 + \sigma^2 + 2(\sum_j \alpha_j K'_{tj}(\mu'_{tj} - \mu_t)^{\top})\Sigma^{-1}C.
\end{aligned}
$$

This completes the derivation.

Appendix D

Exponential Family Distributions

This appendix is devoted to discussing the properties of general exponential family distributions. We will discuss the motivation behind them (both for our purposes, and in general), touch upon their representational abilities, and discuss some of important computational issues surrounding them.

To be able to model domains with structured and/or high-dimensional observations, the EFPSR model generalizes the PLG family of models by replacing the Gaussian distribution representing $p(F^n|h_t)$ with a more general exponential family distribution. The exponential family distribution can capture a wide range of distributions (including the Gaussian) and is capable of capturing graphical structure. This choice of state representation explicitly connects PSRs and PLGs to state-of-the-art probabilistic modeling, which allows the model to take advantage of current efforts in high-dimensional density estimation, graphical models and maximum entropy models.

This appendix discusses these general exponential family distributions. The material is background, and can be skipped by the reader already familiar with the concepts. The sections of this appendix are devoted to an explanation of maximum entropy models (Section D.1), the exponential family of distributions (Section D.2), examples of graphical exponential family models (Section D.3) the relationship between mean and natural parameters (Section D.4), density estimation with natural exponential family models (Section D.5), and inference in graphical exponential family models (Section D.6)

D.1 Maximum Entropy Models

In the introduction of this appendix, we motivated the choice of an exponential family distribution from the perspective of generalizing the Gaussian used in the PLG family of algorithms. There is an alternative justification which is tied more directly to the broader goal of learning dynamical systems from data: using an exponential family form for $p(F^n|h_t)$ has close connections to maximum entropy modeling.

To introduce the concept of maximum entropy modeling, consider the following example. Suppose that we are given samples $x_1, ...x_n$ from an unknown density $p(X)$, and we are asked to infer a complete distribution $\hat{p}(X)$. Additionally, we will suppose that we are given several *features* which we can extract from each sample x_i. Each feature $\phi(x)_i$ is some function of a data sample – if

samples are simple scalars, a feature might be $\phi(x) = x^2$, for example. If samples are images, it may extract an edge, or check for certain colors. There is virtually unlimited flexibility in defining these features.

Given the data and the feature set, it is easy to compute the empirical expectation of the features:

$$\alpha_i = \frac{1}{n} \sum_{i=1}^{n} \phi_i(x_i).$$

It may not be reasonable to suppose that the samples or their features are sufficient to uniquely determine the distribution $\hat{p}(X)$. However, it *is* reasonable to assume that the expectation of each feature under the data set is close to the expectation under the unknown density, by the central limit theorem and the Gaussian nature of the sampling distribution of the mean. We thus assume that:

$$\frac{1}{n} \sum_{i=1}^{n} \phi_i(x_i) \approx \sum_{x} \phi_i(x) p(x).$$

It therefore seems reasonable to ensure that the expectations of the features under our inferred distribution match the empirically observed expectations:

$$\sum_{x} \phi_i(x) \hat{p}(x) = \alpha_i \tag{D.1}$$

If, for example, the features are things like $\phi_1(x) = x$ and $\phi_2(x) = x^2$, then we have effectively placed mean and variance constraints on the distribution. However, these constraints still may not be enough information to uniquely determine the distribution.

The *principle of maximum entropy* states that among all candidate distributions with feature marginal that agree with the empirically observed feature marginals, the one with *maximum entropy* should be selected. We refer the reader to Jaynes (1991) for detailed justification of this idea; briefly, he states that the maximum entropy distribution "agrees with everything that is known, but carefully avoids assuming anything that is not known," which "is the fundamental property which justifies its use for inference."

There are several interpretations to the principle of maximum entropy modeling. Intuitively, the maximum entropy distribution is as "flat" as possible, meaning that it assigns probability which is as close to uniform as possible (indeed, the uniform distribution is the maximum entropy distribution with no constraints). There are also information-theoretic interpretations: the maximum entropy distribution minimizes the information which is assumed when constructing the distribution. In other words, if any other form is selected, then additional external assumptions have been made which are not explicitly justified by the data.

The standard exponential family is the form of the maximum entropy distribution under the marginal

constraints in Eq. D.1, as we now show[1]. We begin with the constraints listed in Eq. D.1 and add an additional constraint that $p(x)$ form a valid density:

$$\sum_x \hat{p}(x) = 1. \tag{D.2}$$

Our goal is to maximize the entropy of $\hat{p}(x)$, subject to the constraints in Eq. D.1 and Eq. D.2. Recall that the entropy of a distribution is defined as

$$H(p(X)) = -\sum_x \hat{p}(x) \log \hat{p}(x). \tag{D.3}$$

Maximizing Eq. D.3 subject to the constraints in Eq. D.1 and Eq. D.2 is a constrained optimization problem. To transform it into an unconstrained optimization problem, we form the Lagrangian:

$$L = -\sum_x \hat{p}(x) \log \hat{p}(x) + \lambda_0 \left(1 - \sum_x \hat{p}(x)\right) + \sum_i \lambda_i \left(\alpha_i - \sum_x \phi_i(x)\hat{p}(x)\right)$$

We can maximize this by differentiating with respect to the entries composing the vector $\hat{p}(x)$ and setting to zero:

$$\frac{\partial L}{\partial \hat{p}(x)} = -\log \hat{p}(x) - 1 - \lambda_0 + \sum_i \lambda_i \phi_i(x) = 0$$

Now, we solve for $\hat{p}(x)$:

$$\hat{p}(x) = \exp\left\{\sum_i -\lambda_i \phi_i(x) + \lambda_0 - 1\right\}. \tag{D.4}$$

This is known as a Gibbs distribution, and is a member of the standard exponential family of distributions. We have only specified the form of the distribution here; it still remains to find the actual values of the λ_i's. Ultimately, these values should be chosen to minimize the amount by which the constraints are not satisfied, which means that the term $\lambda_0 - 1$ will become the log partition function which ensures a valid distribution (this is discussed more in the next section). While we have presented the derivation for the case of a discrete domain for X, these results extend to continuous domains as well with virtually no change, although calculus of variations must be used to arrive at the final form.

There are rich connections between the maximum entropy distribution and the maximum likelihood distribution. For example, maximum entropy modeling and maximum likelihood estimation are dual problems which yield the same optimum: if we assume the exponential distribution, it can be shown that the empirical and model marginals must match at the maximum likelihood solution; if we assume that the empirical and model marginals must match, the maximum likelihood distribution must be an exponential family distribution (Jordan, Unpublished). Thus, the form of Eq. D.4 is also

[1] Apparently, this is originally due to a theorem by Boltzmann.

the form of the distribution which maximizes the likelihood of the data, subject to the constraints listed in Eqs. D.1 and D.2. Moreover, the λ_i's of both the maximum entropy distribution and the maximum likelihood distribution are the same, and they are unique: the optimization problem is convex, with a single global optima. In addition, maximizing the likelihood of the data happens to be equivalent to minimizing the KL divergence to the empirical distribution, and again, the same distribution results Jordan (Unpublished).

D.2 Standard Exponential Family Distributions

We now review the basic properties of exponential family distributions. For a random variable X, a member of the standard exponential family of probability distributions has the form

$$p(X = x; \lambda) = \exp\{\lambda^\top \phi(x) - \log Z(\lambda)\} \tag{D.5}$$

where $\lambda \in \mathbb{R}^k$ is the canonical vector of parameters and $\phi(x)$ is a vector of features of variable x. The vector $\phi(x)$ also forms the sufficient statistics of the distribution. The term $\log Z(\lambda)$ is known as the log-partition function, and is the log of a normalizing constant which ensures that $p(x; \lambda)$ defines a valid distribution:

$$\log Z(\lambda) = \log \int \exp\Big\{\lambda^\top \phi(x)\Big\} dx.$$

Figure D.1 shows a table with a few of the standard basic distributions in standard exponential family form (adapted from (Wainwright and Jordan, 2003)). The normal, gamma, chi-square, beta, Dirichlet, Bernoulli, binomial, multinomial, Poisson, negative binomial, geometric, inverse Gaussian, lognormal and Weibull distributions are all exponential family distributions. Other distributions, such as the Cauchy, Laplace, and uniform families of distributions are not members of the exponential family.

Equation D.5 is the general form of all exponential family distributions. In order to specify a particular member of the exponential family distribution is used, two things must be selected: the *domain* of X, and *features* of the variable X. For example, selecting the domain to be $\mathbb{R}$ and the features to be $\phi_1(x) = x, \phi_2(x) = x^2$, we recover a one-dimensional Gaussian. By carefully selecting the features $\phi(x)$, graphical structure may be imposed on the resulting distribution. This will be an important aspect of the way we use the distribution in our dynamical models, and will be discussed further in Section D.3.

The examples in Figure D.1 can be deceptive: not every member of the exponential family of distributions will have such simple closed-form expressions for the log partition function. We will have more to say about this when discuss parameter estimation and inference in later sections.

Family	Domain	$\log p(x;\lambda)$	$\log Z(\lambda)$	Parameter domain
Bernoulli	{0,1}	$\lambda x - Z(\lambda)$	$\log(1+\exp(\lambda))$	$\mathbb{R}$
Gaussian	$\mathbb{R}$	$\lambda_1 x + \lambda_2 x^2 - Z(\lambda)$	$0.5(\lambda_1 + \log(2\pi e/ - \lambda_2))$	$\{\lambda \in \mathbb{R} \mid \lambda_2 < 0\}$
Exponential	(0,+∞)	$\lambda(-x) - Z(\lambda)$	$-\log\lambda$	$(0,+\infty)$
Poisson	{0,1,2...}	$\lambda x - Z(\lambda)$	$\exp(\lambda)$	$\mathbb{R}$
Beta	(0,1)	$\lambda_1 \log x + \lambda_2 \log(1-x) - Z(\lambda)$	$\sum_{i=1}^{2} \log\Gamma(\lambda_i+1) - \log\Gamma(\sum_{i=1}^{2}(\lambda_i+1))$	$(-1,+\infty)^2$

Figure D.1: A table of distributions in exponential family form.

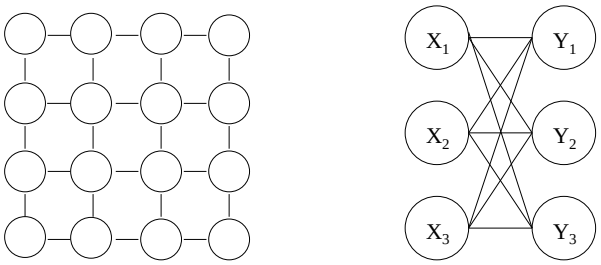


Figure D.2: Two example graphical models. On the left, the Ising model. On the right, a mixed-variable model with a bipartite graph structure.

D.3 Graphical Models Using Exponential Family Distributions

We have mentioned that exponential family distributions can capture graphical structure. This is accomplished by carefully selecting the features $\phi(x)$. To illustrate this, we present a few standard examples of graphical exponential family distributions. The following examples are adapted from Wainwright and Jordan (2003) and Taylor et al. (2007). For each example, we assume we are given a graph $G = (V, E)$, where V denotes the set of vertices and E denotes the set of edges.

Example: Ising Model

The canonical example of an exponential family distribution with graphical structure is the *Ising Model* from statistical physics. We associate with each vertex a binary random variable, and stipulate that variables are only allowed to interact if they are connected by an edge. This model was originally developed to study the phenomena of spontaneous magnetization in ferromagnetic materials. Each node represents a molecule, the two states represent different magnetic orientations, and the links model the fact that molecules tend to affect their neighbors. The model is also used to model some types of binary images in machine vision.

Given the nodes and edges, the resulting exponential model is of the form

$$p(x; \lambda) = \exp\left\{\sum_{i \in V} \lambda_i x_i + \sum_{(i,j) \in E} \lambda_{ij} x_i x_j - \log Z(\lambda)\right\}$$

where the domain of the variables is $\{0, 1\}^{|V|}$. An example of a second-order Ising graph is shown in the left-hand side of Figure D.2, in which the probability of a node is conditionally independent of all the other nodes given its Markov blanket. While we have only demonstrated the model for the case of pairwise interactions, it is trivial to extend to higher-order interactions. For example, to extend to third-order interactions, we add triplets of the form $x_i x_j x_k$ and a corresponding parameter λ_{ijk}.

Example: Gaussian MRF

A Gaussian Markov Random Field is a Gaussian distribution that respects the structure of a graph. For the Gaussian MRF, the features used are $\{x_i, x_i^2 \mid s \in V\} \cup \{x_i x_j | (i,j) \in E\}$, with a corresponding parameter for each feature. The density function is exactly the same as the Ising model:

$$p(x;\lambda) = \exp\left\{\sum_{i\in V} \lambda_i x_i + \lambda_{ii} x_i^2 + \sum_{(i,j)\in E} \lambda_{ij} x_i x_j - \log Z(\lambda)\right\}$$

except that the domain of the variables is now $\mathbb{R}^{|V|}$. In this case, there are restrictions on the domain of the parameter vector λ: the integral defining $Z(\theta)$ is only finite if the matrix

$$A(\lambda) = \begin{bmatrix} \lambda_{1,1} & \cdots & \lambda_{1,n} \\ \vdots & \ddots & \vdots \\ \lambda_{n,1} & \cdots & \lambda_{n,n} \end{bmatrix}$$

of parameters is negative definite (where $A_{ij} = 0 \; \forall (i,j) \notin E$).

Example: Mixed Binary / Gaussian

As another example of a mixed distribution, consider the following energy function (from Taylor et al., 2007):

$$p(x,y;\lambda) = \exp\left\{-\sum_i (x_i - \lambda_i^x)^2 + \sum_j y_j \lambda_j^y + \sum_{ij} x_i y_j \lambda_{ij}^{xy} - Z(\lambda)\right\}$$

which is associated with the bipartite graph in the right-hand side of Figure D.2. Here, the y variables are binary random variables, while the x variables are continuous variables. While there are interactions between x's and y's, there are no interactions *within* the x's or y's. The advantage of this model is that the continuous variables have a simple Gaussian distribution when conditioned on the binary variables, and the binary variables have a simple Bernoulli distribution when conditioned on the continuous variables, with the probability determined by a sigmoidal function:

$$\begin{aligned} p(x_i | y;\lambda) &= \mathcal{N}(\lambda_i^x + \sum_j y_j \lambda_{ij}^{xy}; 1) \\ p(y_i = 1 | x;\lambda) &= \sigma(\lambda_i^y + \sum_i x_i \lambda_{ij}^{xy}). \end{aligned}$$

This means that the model is particularly amenable to Gibbs sampling, because the x's can easily be sampled given the y's, and vice-versa. The point of this example is to illustrate two things: first, there is a great deal of flexibility in setting up these distributions, and second, the correct graphical structure can have significant consequences for the tractability of the model.

D.4 Natural and Mean Parameters

It may not be common to think about some distributions in the form that is listed in Figure D.1. For example, the multinomial distribution is usually written as

$$p(x_i) = p_i \qquad \text{with } 1 \geq p_i \geq 0 \qquad \text{such that } \sum_i p_i = 1.$$

This is known as the *mean parameterization* of the distribution, because $\mathrm{E}[x_i] = p_i$. However, the multinomial distribution can also be written in exponential form as:

$$p(x; \lambda) = \exp\left\{\sum_{i=1}^{k} \lambda_i x_i - \log Z(\lambda)\right\} \tag{D.6}$$

where

$$\lambda_i = \log p_i \tag{D.7}$$

and

$$\log Z(\lambda) = \sum_{i=1}^{k} e^{\lambda_i}$$

for suitable choice of measure space (Brown, 1986). Here, the parameters λ_i are known as the *natural parameters* (or *canonical parameters*).

The idea that members of the exponential family of distributions can be parameterized in multiple ways holds more generally. For example, the Gibbs distribution can use either a mean parameterization or a natural parameterization, as can a Gaussian. For any standard exponential family distribution, the mean parameters of the distribution are given by the expectation of their sufficient statistics:

$$\begin{aligned} \mathrm{E}_\lambda\left[\phi(X)\right] &= \int \phi(x) p(x) dx \\ &= \int \phi(x) \exp\left\{\lambda^\top \phi(x) - \log Z(\lambda)\right\} dx \\ &= \nabla_\lambda \log Z(\lambda). \end{aligned}$$

This vector of expected sufficient statistics is of central interest in standard exponential families, and has several interesting properties. Like the vector λ, $\mathrm{E}_\lambda\left[\phi(X)\right]$ uniquely defines the distribution $p(X)$. The vector $\mathrm{E}_\lambda\left[\phi(X)\right]$ also plays an important role when learning the parameters of an exponential family from data, as discussed in Section D.5.

The idea that mean and natural parameters are both sufficient to describe a distribution will be a key part of our proofs of representational capacity for the EFPSR model in Section 7.2, and so we discuss the idea in more detail. Translating between mean and natural parameters is non-trivial in a practical sense (computing the mean parameters from a given set of natural parameters is a specific kind of

inference problem, which we discuss in Section D.6), but from a theoretical perspective, there are many appealing relationships. For example, each possible set of canonical parameters λ induces one set of mean parameters; assuming that the features are linearly independent (equivalently, that the features form a minimal set), each set of valid mean parameters is uniquely determined by one set of canonical parameters (Wainwright and Jordan, 2003). The mean parameters can also be arrived at by taking derivatives of the term $\log Z(\lambda)$, which is why it is sometimes known as the *cumulant generating function.*

There are some technical conditions on which sets of mean and natural parameters give rise to valid distributions. Not every vector in $\mathbb{R}^d$ forms a valid set of mean parameters because of consistency constraints between members of the set (valid mean parameter vectors are known as *realizable* parameters). However, for every realizable mean parameter vector, there exists some exponential family distribution which has marginals equal to that vector. This is an important result, since the exponential family describes a strict subset of all possible densities, whereas the definition of a realizable mean parameter vector is not restricted to any particular form of a distribution.

D.5 Maximum Likelihood Learning

In Section D.1 we motivated the form of the exponential family distribution through a connection to maximum entropy modeling of a dataset. While we showed that both the maximum entropy distribution and the maximum likelihood distribution had the same exponential family form, we did not present any algorithm to actually find the parameters of the distribution. In this section, we discuss maximum likelihood parameter estimation, with the specific goal of connecting the vector of mean parameters to the problem of learning the parameters of the distribution.

The modelling problem is as follows: we are given samples $x_1, ..., x_N$ from an unknown density $p(X)$ and a feature extractor $\phi(x)$, and we are asked to infer a complete distribution $\hat{p}(X)$. We assume that the final distribution will have an exponential family form:

$$\hat{p}(X) = \exp\left\{-\lambda^\top \phi(x) - \log Z(\lambda)\right\},$$

so the task becomes finding the parameter vector λ which maximizes the likelihood of the data.

The likelihood of the data is $p(x_1, x_2, \cdots, x_N)$. Assuming that the samples are independent, the likelihood factors as $p(x_1)p(x_2)\cdots p(x_N)$. The expected log likelihood of the data is therefore

$$\mathcal{LL} = \frac{1}{N}\left(\sum_{i=1}^{N} -\lambda^\top \phi(x_i) - \log Z(\lambda)\right) \tag{D.8}$$

This objective function is convex in the parameter λ, and therefore has a unique global maxima. We

now compute the gradient of $\mathcal{LL}$ with respect to the parameters λ:

$$\begin{aligned}
\frac{\partial \mathcal{LL}}{\partial \lambda} &= \frac{\partial}{\partial \lambda}\left[\frac{1}{N}\left(\sum_{i=1}^{N} -\lambda^\top \phi(x_i) - \log Z(\lambda)\right)\right] \\
&= \frac{1}{N}\left(\sum_{i=1}^{N} \frac{\partial}{\partial \lambda}\left[-\lambda^\top \phi(x_i)\right]\right) - \frac{\partial}{\partial \lambda}\log Z(\lambda) \\
&= \frac{1}{N}\left(\sum_{i=1}^{N} -\phi(x_i)\right) - \frac{\partial}{\partial \lambda}\log Z(\lambda) \\
&= \mathrm{E}_\lambda[\phi(X)] - \mathrm{E}_{\text{empirical}}[\phi(x)].
\end{aligned}$$

We see that the derivative is the difference between two terms: $\mathrm{E}_\lambda[\phi(X)]$ is the expectation of $\phi(X)$ under the distribution defined by the current λ, and $\mathrm{E}_{\text{empirical}}[\phi(X)]$ is the empirical expectation of the features. Setting this derivative to zero implies that the model expectations and the empirical expectations are equal when we have found the maximum likelihood parameters, which is the constraint we imposed when introducing maximum entropy modeling.

Interestingly, the vector $\mathrm{E}_\lambda[\phi(X)]$ is exactly the vector of mean parameters defined in Section D.4. This term arose when computing the gradient $\frac{\partial}{\partial \lambda}\log Z(\lambda)$, and implicitly involves integrating over the entire domain of X. Computing this vector is an inference problem, as we now discuss.

D.6 Inference in Exponential Family Distributions

Inference is a general term which refers to any operation where a probabilistic query is made against a graphical model. Typical inference problems include

- Computing the likelihood of observed data.
- Computing the marginal distribution over a particular subset of data.
- Computing a conditional distribution.
- Computing the mode of a distribution.

As discussed in Section D.5, computing the vector of mean parameters is an inference problem. This sort of inference needs to happen when using a gradient method to learn the maximum likelihood parameters of a distribution, or when translating between natural and mean parameters as discussed in Section D.4.

Exact inference in an exponential family graphical model is known to NP-Hard (Cooper, 1990). For example, the problem of computing the density of a sample x is intractable because of the global normalization constant Z: computing Z implicitly involves a sum over all possible configurations of X, which scales exponentially with the domain of X. In special cases, such as a fully factored

distribution, or a distribution where the graph is tree-structured, the operations can be factored such that inference is tractable. Efficient inference has been the subject of intense research, and many approximate algorithms exist: there are variational methods such as naive mean-field, tree-reweighted belief propagation, and log-determinant relaxations (Wainwright and Jordan, 2006); other methods include Bethe-Kikuchi approximations, expectation propagation (Minka, 2001), (loopy) belief propagation (Yedida et al., 2001), MCMC methods (Neal, 1993), and approximate gradient methods such as contrastive divergence (Hinton, 2002). However, even approximate inference is known to be NP-Hard (Dagum and Luby, 1993).

Depending on the exact features used and the domain of the model, different inference methods may be more or less useful. For example, in the Gaussian Markov Random Field of Section D.3, inference involves solving a large, sparse system of equations. In the mixed binary/Gaussian model in Section D.3, Gibbs sampling is a tractable choice because of the convenient conditional forms of the distribution. In graphs which are highly connected, naive mean field is a good choice, whereas in domains with pairwise features, loopy belief propagation is a popular choice. Understanding inference algorithms, their properties, and the graphical models with which they are compatible is an entire area of expertise, of which we have only presented the most basic outline.

Appendix E

Approximate Log-Likelihood Derivations

In Chapter 10, we presented an approximate learning algorithm for the Linear-Linear EFPSR. This learning algorithm was based on a quantity we called the Approximate Log-Likelihood, which was derived in Section 10.1. For convenience, we repeat the definition here:

$$\widehat{\mathcal{LL}} = \mathrm{E}_T\left[-s_t\right]^\top \mathrm{E}_T\left[\phi(f_t)\right] - \log Z(\mathrm{E}_T\left[s_t\right]). \tag{E.1}$$

This appendix discusses how to compute this quantity and its derivatives with respect to the model parameters in a computationally efficient manner. In Section E.3, we also include our method for computing gradients when the parameter matrix is rank-constrained.

E.1 Computing the Approximate Log-Likelihood

To compute the approximate log-likelihood we must compute three terms: $\mathrm{E}_T[s_t]$ (the stationary distribution of states), $\mathrm{E}_T[\phi(f_t)]$ (which is computed once from data), and the log partition function $\log Z(\mathrm{E}_T[s_t])$. We begin with the computation of $\mathrm{E}_T[s_t]$. Recall that our goal is to compute this term in a way that is independent of T, which is the length of the training trajectory. This will be possible using Assumption 10.1.1, the linearity of the state update, and an insight related to stationary distributions:

$$\begin{aligned}
\mathrm{E}_T[s_t] &= \mathrm{E}_T\left[G(o_t)\left(As_{t-1} + B\right)\right] \\
&\approx \mathrm{E}_T\left[G(o_t)A\right]\mathrm{E}_T\left[s_{t-1}\right] + \mathrm{E}_T\left[G(o_t)\right]B \\
&= \mathrm{E}_T\left[G(o_t)A\right]\mathrm{E}_T\left[s_t\right] + B_G \\
&= G(\mathrm{E}_T[o_t])A\mathrm{E}_T\left[s_t\right] + B_G \\
&= \left(I - G(\mathrm{E}_T[o_t])A\right)^{-1} B_G
\end{aligned} \tag{E.2}$$

where I is an appropriately sized identity matrix, and where $B_G = G(\mathrm{E}_T[o_t])B$. The second line follows by Assumption 10.1.1.

The third and fourth lines are both interesting for different reasons. The fourth line follows by the linearity of the operator $G(\cdot)$. The matrix $G(\mathrm{E}[o_t])$ can be interpreted as the expected transition operator, and is a simple function of the expected observation $\mathrm{E}_T[o_t]$ – that is, we just compute the empirical expectation of the observations, and generate a transition operator as we would with any

other observation. Just as $\mathrm{E}_T[s_t]$ can be interpreted as the stationary distribution of states, $\mathrm{E}[o_t]$ is the empirical average of the observations, and in the limit of large T can be interpreted as the stationary distribution of observations induced by A and B.

The third line is interesting because it follows by the limiting properties of our expectations: we assume that $\mathrm{E}_T[s_t] = \mathrm{E}_T[s_{t-1}]$ because as $t \to \infty$, the unconditional distribution of these states approach the stationary distribution of states.

Taken together, the result is that the stationary distribution of states can be computed as the solution to a linear system of equations. The fact that the solution can be obtained efficiently (in a variety of manners) is key to making a practical learning algorithm. The inverse in Eq. E.2 should not be computed explicitly. The quantity

$$I - G(\mathrm{E}_T[o_t])A$$

is a large matrix with favorable computational properties. The $G(E_T[o_t])$ part will typically be very sparse, and a designer may force the A part to be sparse or low-rank (a low-rank parameterization will be a critical factor in further efficiencies, as explained in Section 10.2). In either case, a matrix-vector product can be computed efficiently, so instead of computing the inverse, the linear system of equations defined by

$$\left(I - G(\mathrm{E}_T[o_t])A\right) x = B_G \tag{E.3}$$

should be solved using an iterative solver designed for the solution of large, sparse linear systems of equations, such as conjugate gradients, GMRES, BiCGSTAB, TFQMR, etc. (Saad, 1996).

Once Eq. E.3 is solved, $\mathrm{E}_T[s_t] = x$. With $\mathrm{E}_T[s_t]$ in hand, we can compute the log partition function $\log Z(\mathrm{E}_T[s_t])$, using the vector $\mathrm{E}_T[s_t]$ in place of an ordinary state s.

E.2 Computing Derivatives of the Approximate Log-Likelihood

We now compute the derivatives of the approximate log-likelihood with respect to an arbitrary parameter Θ:

$$\frac{\partial \widehat{\mathcal{LL}}}{\partial \Theta} = \frac{\partial \widehat{\mathcal{LL}}}{\partial \mathrm{E}_T[s_t]}^\top \frac{\partial \mathrm{E}_T[s_t]}{\partial \Theta}$$

We begin with the left-hand term:

$$\begin{aligned}
\frac{\partial \widehat{\mathcal{LL}}}{\partial \mathrm{E}_T[s_t]} &= \frac{\partial}{\partial \mathrm{E}_T[s_t]} \left[\mathrm{E}_T[-s_t]^\top \mathrm{E}_T[\phi(f_t)] - \log Z(\mathrm{E}_T[s_t]) \right] \\
&= -\mathrm{E}_T[\phi(f_t)] - \frac{\partial}{\partial \mathrm{E}_T[s_t]} \log Z(\mathrm{E}_T[s_t]) \\
&= \mathrm{E}_{\mathrm{E}_T[s_t]}[\phi(F)] - \mathrm{E}_T[\phi(f_t)] \\
&\equiv \Delta
\end{aligned}$$

This result has an appealing intuitive interpretation. $\mathrm{E}_{\mathrm{E}_T[s_t]}[\phi(F)]$ can be interpreted as the expected sufficient statistics that would be obtained if inference were performed using the stationary distribution of states $\mathrm{E}_T[s_t]$ as the state – in other words, it represents the stationary distribution of features of n-step trajectories, as computed by the model. Recall that $\mathrm{E}_T[\phi(f_t)]$ represents the empirically observed stationary distribution of features of n-step trajectories. We see that the gradient wishes to move the expected sufficient statistics as computed under the model's distribution to more closely match the expected sufficient statistics under the empirical distribution.

If we use a variational method to compute the log partition function $\log Z(\mathrm{E}_T[s_t])$, which is needed to determine the value of the log-likelihood, then the expected sufficient statistics $\mathrm{E}_{\mathrm{E}_T[s_t]}[\phi(F)]$ are available as a byproduct of the optimization. This is a pleasing efficiency.

However, we are not done. Because this is a dynamical system, we must find the transition parameters which allow us to move the expected sufficient statistics closer. This is captured as we compute the right-hand term. We will start by computing the derivative with respect to the A parameter:

$$\begin{aligned}
\frac{\partial \mathrm{E}_T[s_t]}{\partial A} &= \frac{\partial}{\partial A}\left[G(\mathrm{E}[o_t])(A\mathrm{E}_T[s_t] + B)\right] \\
&= G(\mathrm{E}[o_t])A\left(\frac{\partial}{\partial A}\mathrm{E}_T[s_t]\right) + \left(\frac{\partial}{\partial A}G(\mathrm{E}[o_t])A\right)\mathrm{E}_T[s_t] \\
&= (I - G(\mathrm{E}[o_t])A)^{-1}\left(\frac{\partial}{\partial A}G(\mathrm{E}[o_t])A\right)\mathrm{E}_T[s_t]
\end{aligned}$$

We now find it convenient to remember that the full derivative also includes the term $\partial\widehat{\mathcal{LL}}/\partial\mathrm{E}_T[s_t] = \Delta$, which is a column vector:

$$\begin{aligned}
\Delta^\top\frac{\partial \mathrm{E}_T[s_t]}{\partial A} &= \Delta^\top(I - G(\mathrm{E}[o_t])A)^{-1}\left(\frac{\partial}{\partial A}G(\mathrm{E}[o_t])A\right)\mathrm{E}_T[s_t] \\
&= \Gamma^\top\left(\frac{\partial}{\partial A}G(\mathrm{E}[o_t])A\right)\mathrm{E}_T[s_t] \\
&= \frac{\partial}{\partial A}\Gamma^\top G(\mathrm{E}[o_t])A\mathrm{E}_T[s_t] \\
&= \frac{\partial}{\partial A}\Gamma_G^\top A\mathrm{E}_T[s_t] \\
&= \Gamma_G\mathrm{E}_T[s_t]^\top \qquad \text{(E.4)}
\end{aligned}$$

where we have defined Γ to be the solution to the linear system of equations:

$$\Delta = (I - G(\mathrm{E}[o_t])A)^\top\Gamma.$$

and $\Gamma_G = \Gamma^\top G(\mathrm{E}[o_t])$.

Algorithm APPROX-LL-GRADS

Input: $\mathrm{E}_T[o_t]$, $\mathrm{E}_T[\phi(f_t)]$, A, B

Compute:

- Compute the approximate log-likelihood:
  - Compute $\mathrm{E}_T[s_t] = (I - G(\mathrm{E}_T[o_t])A)^{-1} B$
  - Use $\mathrm{E}_T[s_t]$ as the weights on the graphical model, and compute, using the inference method of choice:
    - the expected sufficient statistics $\mathrm{E}_{\mathrm{E}_T[s_t]}[\phi(F)]$.
    - the log partition function $\log Z(\mathrm{E}_T[s_t])$.
  - Let $\widehat{\mathcal{LL}} = -\mathrm{E}_T[s_t]^\top \mathrm{E}_{\mathrm{E}_T[s_t]}[\phi(F)] - \log Z(\mathrm{E}_T[s_t])$.
- Compute the gradient of the approximate log-likelihood with respect to the parameters:
  - Let $\Delta = \mathrm{E}[\phi(f_t)] - \mathrm{E}_{\mathrm{E}_T[s_t]}[\phi(F)]$.
  - Solve $\Delta = (I - G(\mathrm{E}[o_t])A)^\top \Gamma$ for Γ.
  - Let $\Gamma_G = G(\mathrm{E}[o_t])^\top \Gamma$.
  - Let $\Delta_G = G(\mathrm{E}[o_t])^\top \Delta$.
  - Let $\nabla_A \widehat{\mathcal{LL}} = \Gamma_G \mathrm{E}_T[s_t]^\top$.
  - Let $\nabla_B \widehat{\mathcal{LL}} = \Delta_G$.

Return: $\widehat{\mathcal{LL}}, \nabla_A \widehat{\mathcal{LL}}, \nabla_B \widehat{\mathcal{LL}}$

Figure E.1: Approximate maximum likelihood gradients for Linear-Linear EFPSRs.

The derivative with respect to the B parameter is computed similarly:

$$\begin{aligned}
\Delta^\top \frac{\partial \mathrm{E}_T[s_t]}{\partial B} &= \Delta^\top \frac{\partial}{\partial B}\left[G(\mathrm{E}[o_t])(A\mathrm{E}_T[s_t] + B)\right] \\
&= \frac{\partial}{\partial B}\Delta^\top G(\mathrm{E}[o_t])B \\
&= \frac{\partial}{\partial B}\Delta_G^\top B \\
&= \Delta_G
\end{aligned}$$

where Δ_G is defined analogously to Γ_G.

This completes the derivation of the gradients of the approximate log-likelihood. The complete algorithm is summarized in Figure 10.1.

E.3 Computing Gradients with a Low-Rank Parameter Matrix

In this section, we propose a method of learning a low-rank decomposition of the parameter matrix A. We will discuss the four elements of our algorithm: 1) how we represent the low-rank parameterization; 2) how we compute the gradients; 3) how the parameter matrix can can be updated given a gradient and a stepsize; and 4) how the stepsize can be selected. We discuss each topic in turn.

1. **Begin with a low-rank matrix:** Instead of maintaining the full matrix $A \in \mathbb{R}^{k \times l}$, our strategy is to maintain the thin SVD of A, which is constrained to have rank d: $A = USV^\top$, where $U \in \mathbb{R}^{k \times d}$ is an orthonormal matrix, $S \in \mathbb{R}^{d \times d}$ is a diagonal matrix of singular values, and $V \in \mathbb{R}^{l \times d}$ is another orthonormal matrix. We will constrain d to be $\ll \min(k, l)$.

2. **Rank-one gradients:** A key property of the approximate log-likelihood parameter learning algorithm is that the gradients of $\widehat{\mathcal{LL}}$ with respect to A are rank-one. Specifically, recall from Eq. E.4 that

$$\nabla_A \widehat{\mathcal{LL}} = \Gamma_G \mathrm{E}_T \left[s_t\right]^\top$$

   Note that $\Gamma_G \in \mathbb{R}^{k \times 1}$, and $\mathrm{E}_T \left[s_t\right] \in \mathbb{R}^{l \times 1}$. Thus, while $\nabla_A \widehat{\mathcal{LL}}$ is $\in \mathbb{R}^{k \times l}$, the rank of this matrix is one. That is, $\frac{\partial \widehat{\mathcal{LL}}}{\partial A(i,j)} = (\Gamma_G)_i (\mathrm{E}_T \left[s_t\right])_j$. This is not true of the exact likelihood gradients defined in Eq. 9.2; they can have rank as large as $\min(k, l, T)$. Thus, the algorithm we are about to develop will be unique to the learning algorithms defined with $\widehat{\mathcal{LL}}$.

3. **Updating the low-rank decomposition for a given stepsize:** Given a low-rank decomposition of $A = USV^\top$, and the rank-one decomposition of the gradient matrix, how can we update the decomposition of A? To answer this, let us be more specific about how the gradients are used. We use the gradients in a steepest descent optimizer. Recall that our goal is to maximize $\widehat{\mathcal{LL}}$, so to increase $\widehat{\mathcal{LL}}$, we take a small step in the gradient direction: $A = A + \alpha \nabla_A \widehat{\mathcal{LL}}$, where α is a suitably chosen stepsize parameter.

   Given the parameters $A = USV^\top$, and a rank-one candidate update $ab^\top$, can we compute the parameters of $A + ab^\top = U'S'V'^\top$, using nothing but U, S, V, a and b? Fortunately, the answer is yes: Brand (2006) has an algorithm for doing exactly this. It operates in $O(lkd^3)$ time, and never requires the matrix A to be formed explicitly. The algorithm is summarized in Figure E.2. In other words, we can begin learning by choosing some initial guess for the matrix A – say, $A = 0$, which has a known low-rank decomposition. We can then compute the gradients and update this low-rank decomposition without ever having to form A.

4. **Selecting a stepsize adaptively:** We can do better than a naive steepest descent algorithm, which uses a fixed stepsize. Many modern optimization algorithms operate based on the concept of *line searches*. The idea is simple: suppose we have a function $f(\theta)$ we wish to maximize. We first select a search direction (perhaps a gradient [as in steepest descent], or perhaps a direction which is a combination of previous gradients [as in the conjugate gradient

approach]). Given a search direction, we loosely maximize f by finding a stepsize parameter α which hops to a local maximum, as in the following stylized algorithm:

- Repeat
  - Compute search direction ∇_θ
  - Perform a line search: find α which maximizes $f(\theta + \alpha\nabla_\theta)$
  - Update parameters: $\theta = \theta + \alpha\nabla_\theta$
- Until $\|\nabla_\theta\| < \epsilon$.

The line search can be done exactly, or we may simply ask for a sufficient increase in the objective function using the Wolfe conditions (Nocedal and Wright, 1999).

Thus, as long as we can compute $\widehat{\mathcal{LL}}$ for a given parameter setting U, S, V, B plus a candidate rank-one addition $\alpha ab^\top$, we can perform an efficient line search. Recall that the most computationally intensive part of computing $\widehat{\mathcal{LL}}$ algorithm is the solution to two systems of linear equations. However, we have emphasized that iterative solvers can and should be used to solve this equation. Most iterative solvers never require the explicit formation of a matrix A. Instead, they require only matrix-vector products, so as long as the matrix-vector product can be computed efficiently, the entire procedure will be efficient. In our case, the matrix-vector product can be computed efficiently even with a candidate rank-one update, as long as the operation is factored appropriately:

$$\begin{aligned} G(o_t)Ax &= G(o_t)(USV^\top + ab^\top)x \\ &= G(o_t)\left[U(S(V^\top x)) + a(b^\top x)\right] \\ &= G(o_t)x'. \end{aligned}$$

The entire operation only involves low-rank matrix-vector products (line 2), followed by a full-rank (but sparse) matrix-vector product.

We term this entire procedure a *rank-aware line search.*

Figure E.3 shows an algorithm for computing the gradients of $\widehat{\mathcal{LL}}$ with an optional rank-one modification, and returns the gradient components in factored form (that is, it never explicitly forms A). Using this algorithm and Brand's SVD update algorithm in Figure E.2, Figure 10.2 shows a steepest descent optimizer which performs a rank-aware line search, and then updates the parameter matrix while maintaining a low-rank decomposition.

It is an open question as to whether this method can be extended to higher-order optimization methods, such as quasi-Newton methods or other iterative solvers with quadratic convergence.

Algorithm UPDATE-SVD

Input: d, U, S, V, a, b

Remark: d is the maximum acceptable rank of the resulting matrix.

Compute:

$$\begin{aligned} m &= U^\top a & n &= V^\top b \\ p &= a - Um & q &= b - Vn \\ P &= p/\|p\| & Q &= q/\|q\| \end{aligned}$$

Let $K = \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} m \\ \|p\| \end{bmatrix} \begin{bmatrix} n \\ \|q\| \end{bmatrix}^\top$

Compute the SVD of K, restricting it to be rank d: $K = U'S'V'^\top$.

Now, $X + ab^\top = ([UP]U')S'([VQ]V')^\top$

Return: $([UP]U')$, S', $([VQ]V')$.

Figure E.2: An algorithm to update the SVD of a matrix. Given $X = USV^\top$, update it to be the SVD of $X + ab^\top$.

Algorithm LOWRANK-APPROX-LL-GRADS

Input: $\mathrm{E}_T[o_t]$, $\mathrm{E}_T[\phi(f_t)]$, U, S, V, B, x, y, b

Remark: $A = USV + xy^\top$.

Compute:

- Compute the approximate log-likelihood:
  - Compute $\mathrm{E}_T[s_t] = \left(I - G(\mathrm{E}_T[o_t])(USV^\top + xy^\top)\right)^{-1}[B + b]$
  - Use $\mathrm{E}_T[s_t]$ as the weights on the graphical model, and compute, using the inference method of choice:
    - the expected sufficient statistics $\mathrm{E}_{\mathrm{E}_T[s_t]}[\phi(F)]$.
    - the log partition function $\log Z(\mathrm{E}_T[s_t])$.
  - Let $\widehat{\mathcal{LL}} = -\mathrm{E}_T[s_t]^\top \mathrm{E}_{\mathrm{E}_T[s_t]}[\phi(F)] - \log Z(\mathrm{E}_T[s_t])$.
- Compute the gradient of the approximate log-likelihood with respect to the parameters:
  - Let $\Delta = \mathrm{E}[\phi(f_t)] - \mathrm{E}_{\mathrm{E}_T[s_t]}[\phi(F)]$.
  - Solve $\Delta = \left(I - G(\mathrm{E}_T[o_t])(USV^\top + xy^\top)\right)^\top \Gamma$ for Γ.
  - Let $\Delta_G = G(\mathrm{E}_T[o_t])^\top \Delta$.
  - Let $\Gamma_G = G(\mathrm{E}_T[o_t])^\top \Gamma$.
  - Remark: $\nabla_A \widehat{\mathcal{LL}} = \Gamma_G \mathrm{E}_T[s_t]^\top$.
  - Remark: $\nabla_B \widehat{\mathcal{LL}} = \Delta_G$.

Return: $\widehat{\mathcal{LL}}, \Gamma_G, \mathrm{E}_T[s_t], \Delta_G$

Figure E.3: An algorithm for computing rank-aware approximate ML gradients for EFPSRs. Assumes that $A = USV^\top$, with an optional rank-one modification $xy^\top$. The vector B may have an optional modification b.

Bibliography

Douglas Aberdeen and Jonathan Baxter. Scalable internal-state policy-gradient methods for POMDPs. In *International Conference on Machine Learning (ICML)*, pages 3–10, 2002.

Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. *Lecture Notes in Computer Science*, 1973:420–434, 2001.

K. J. Astrom. Optimal control of Markov decision processes with the incomplete state estimation. *Journal of Computer and System Sciences*, 10:174–205, 1965.

Bram Bakker. *The State of Mind: Reinforcement Learning with Recurrent Neural Networks*. PhD thesis, Leiden University, 2004.

Kenneth Basye, Thomas Dean, and Leslie Pack Kaelbling. Learning dynamics: System identification for perceptually challenged agents. *Artificial Intelligence*, 72(1-2):139–171, 1995.

Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

Jonathan Baxter and Peter Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research (JAIR)*, 15:319–350, 2001.

William Bialek, Ilya Nemenman, and Naftali Tishby. Predictability, complexity and learning. *Neural Computation*, 13:2409–2463, 2001.

Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

David Blackwell. Discrete dynamic programming. *Annals of Mathematical Statistics*, 33:719–726, 1962.

Lisa Borland, Angel R. Plastino, and Constantino Tsallis. Information gain within nonextensive thermostatistics. *Journal of Mathematical Physics*, 39(12):6490–6501, 1998.

Michael Bowling, Peter McCracken, Michael James, James Neufeld, and Dana Wilkinson. Learning predictive state representations using non-blind policies. In *International Conference on Machine Learning (ICML)*, pages 129–136, 2006.

Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Uncertainty in Artificial Intelligence (UAI)*, pages 33–42, 1998.

Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 415:20–30, 2006.

Lawrence D. Brown. *Fundamentals of Statistical Exponential Families*. Institute of Mathematical Sciences, 1986.

Anthony R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, 1998a.

Anthony R. Cassandra. A survey of POMDP applications. In *Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes*, pages 17–24, 1998b.

Venkat Chandrasekaran, Jason K. Johnson, and Alan S. Willsky. Maximum entropy relaxation for graphical model selection given inconsistent statistics. In *IEEE Statistical Signal Processing Workshop*, August 2007.

Lonnie Chrisman. Reinforcement learning with perceptual aliasing: the perceptual distinctions approach. In *National Conference on Articial Intelligence (AAAI)*, pages 183–188, 1992.

Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

Paul Dagum and Michael Luby. Approximate probabilistic reasoning in bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

Arthur Pentland Dempster, Nan McKenzie Laird, and Donald Bruce Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B(39):1–38, 1977.

Yaakov Engel, Shie Mannor, and Ron Meir. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *International Conference on Machine Learning*, pages 154–161, 2003.

Yaakov Engel, Shie Mannor, and Ron Meir. The kernel recursive least squares algorithm. *IEEE Transactions on Signal Processing*, 52(8):2275–2285, 2004.

Ryan Eustice. *Large-Area Visually Augmented Navigation for Autonomous Underwater Vehicles*. PhD thesis, Massachussetts Institute of Technology, 2005.

J. Fan and I. Gijbels. *Local Polynomial Modelling and Its Applications*. Chapman and Hall, 1996.

Udo Frese. A proof for the approximate sparsity of SLAM information matrices. In *IEEE International Conference on Robotics and Automation*, pages 329– 335, 2005.

Zoubin Ghahramani and Geoffrey E. Hinton. Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, Dept. of Computer Science, U. of Toronto, 1996.

Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins University Press, Baltimore, MD, 1996.

A. Ben Hamza. A nonextensive information-theoretic measure for image edge detection. *Journal of Electronic Imaging*, 15(1):13011.1–13011.8, 2006.

Eric A. Hansen. Solving POMDPs by searching in policy space. In *Uncertainty in Articial Intelligence (UAI)*, pages 211–219, 1998.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2001.

David Haussler, Andrews Krogh, I. Saira Mian, and Kimmen Sjolander. Protein modeling using hidden Markov models: Analysis of globins. In *Proceedings of the 26th Hawaii International Conference on System Sciences*, pages 792–802, Honolulu, 1993. IEEE Computer Society Press.

Yun He, A. Ben Hamza, and Hamid Krim. A generalized divergence measure for robust image registration. *IEEE Transactions on Signal Processing*, 51(5):1211– 1220, 2003.

Kenneth E. Hild, Deniz Erdogmus, and Jose C. Principe. Blind source separation using renyi's mutual information. *IEEE Signal Processing Letters*, 8(6):174–176, 2001.

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

Jesse Hoey, Axel von Bertoldi, Pascal Poupart, and Alex Mihailidis. Assisting persons with dementia during handwashing using a partially observable Markov decision process. In *International Conference on Computer Vision Systems*, 2007.

Michael P. Holmes and Charles Lee Isbell. Looping suffix tree-based inference of partially observable hidden state. In *International Conference on Machine Learning (ICML)*, pages 409–416, 2006.

Tommi Jaakkola, Satinder Singh, and Michael I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In *Neural Information Processing Systems (NIPS)*, volume 7, pages 345–352, 1995.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrety E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

Herbert Jaeger. Observable operator processes and conditioned continuation representations. *Neural Computation*, 12(6):1371–1398, 2000.

Herbert Jaeger. Discrete-time, discrete-valued observable operator models: A tutorial. Technical report, International University Bremen, 2004.

Michael R. James. *Using Predictions for Planning and Modeling in Stochastic Environments*. PhD thesis, University of Michigan, 2005.

Michael R. James and Satinder Singh. Learning and discovery of predictive state representations in dynamical systems with reset. In *International Conference on Machine Learning (ICML)*, pages 417–424, 2004.

Michael R. James and Satinder Singh. Planning in models that combine memory with predictive representations of state. In *National Conference on Artificial Intelligence (AAAI)*, pages 987–992, 2005a.

Michael R. James, Britton Wolfe, and Satinder Singh. Combining memory and landmarks with predictive state representations. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 734–739, 2005b.

Edwin T. Jaynes. Notes on present status and future prospects. In W.T. Grandy and L.H. Schick, editors, *Maximum Entropy and Bayesian Methods*, pages 1–13, 1991.

Michael I. Jordan. *An Introduction to Probabilistic Graphical Models*. Unpublished.

Simon Julier and Jeffrey K. Uhlmann. A general method for approximating nonlinear transformations of probability distributions. Technical report, University of Oxford, 1996.

Rudolph E. Kalman. A new approach to linear filtering and prediction problem. *Transactions of the ASME—Journal of Basic Engineering*, 82(Series D):35–45, 1960.

Rudolph E. Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Transactions of the ASME—Journal of Basic Engineering*, 83:95–107, 1961.

Paul Kaminski, Arthur Bryson Jr., and Stanley Schmidt. Discrete square root filtering: A survey of current techniques. *IEEE Transactions on Automatic Control*, 16(6):727–736, 1971.

J.N. Kapur. *Measures of Information and their Application*. John Wiley, 1994.

George S. Kimeldorf and Grace Wahba. Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33:82–95, 1971.

Sven Koenig and Reid Simmons. Xavier: A robot navigation architecture based on partially observable Markov decision process models. In *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91 – 122. MIT Press, 1998.

Solomon Kullback. *Information Theory and Statistics*. Dover, New York, NY, 1968.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, pages 282–289, 2001.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research (JMLR)*, 4:1107–1149, 2003.

Daniel E. Lane. A partially observable model of decision making by fishermen. *Operations Research*, 37(2):240–254, 1989.

Michael L. Littman. The witness algorithm: Solving partially observable Markov decision processes. Technical Report CS-94-40, Brown University, 1994.

Michael L. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, 1996.

Michael L. Littman, Richard S. Sutton, and Satinder Singh. Predictive representations of state. In *Neural Information Processing Systems (NIPS)*, pages 1555–1561, 2002.

John Loch and Satinder Singh. Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In *International Conference on Machine Learning (ICML)*, pages 323–331, 1998.

William S. Lovejoy. A survey of algorithmic methods for partially observable Markov decision processes. *Annals of Operations Research*, 28:47–66, 1991.

Michael C. Mackey and Leon Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, 1977.

Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22:159–196, 1996.

James Manyika and Hugh Durrant-Whyte. *Data Fusion and Sensor Management: A Decentralized Information-Theoretic Approach*. Prentice Hall PTR, 1995.

Andrei A. Markov. An example of statistical investigation in the text of 'Eugene Onegin' illustrating coupling 'tests' in chains. *Proceedings of the Academy of Science of St. Petersburg*, 7:153–162, 1913.

Peter S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 141 of *Mathematics in Science and Engineering*. 1979.

Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. In *International Conference on Machine Learning (ICML)*, pages 591–598, 2000.

Andrew K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.

Peter McCracken and Michael Bowling. Online discovery and learning of predictive state representations. In *Neural Information Processings Systems (NIPS)*, pages 875–882, 2006.

Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra. Solving POMDPs by searching the space of finite policies. In *Uncertainty in Artificial Intelligence (UAI)*, pages 417–426, 1999.

Thomas Minka. *A Family of Algorithms for Approximate Bayesian Inference*. PhD thesis, Massachussetts Institute of Technology, 2001.

George E. Monahan. A survey of partially observable markov decision processes: theory, models, and algorithms. *Management Science*, 28:1–16, 1982.

Radford M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

Daniel Nikovski. *State-Aggregation Algorithms for Learning Probabilistic Models for Robot Control*. PhD thesis, Carnegie Mellon University, 2002.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Verlag, New York, NY, 1999.

Steven J. Nowlan and Geoffrety E. Hinton. Evaluation of adaptive mixtures of competing experts. In *Neural Information Processing Systems (NIPS)*, 1991.

Sudhakar M. Pandit and Shien-Ming Wu. *Time Series and System Analysis with Applications*. John Wiley, 1983.

Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processses. *Mathematics of Operations Research*, 12(3):441–450, 1987.

Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Natural actor-critic. In *European Conference on Machine Learning (ECML)*, pages 280–291, 2005.

Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1025–1032, 2003.

John C. Platt. FastMap, MetricMap and Landmark MDS are all Nystrom algorithms. Technical Report MSR-TR-2004-26, Microsoft Research, 2004.

Jose C. Principe, Dongxin Xu, and John W. Fisher. *Information Theoretic Learning*, pages 265–319. Wiley, 1999.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.

Lawrence R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE Acoustic Speech Signal Processing Magazine*, 3:4–16, 1986.

Eddie J. Rafols, Mark B. Ring, Richard S. Sutton, and Brian Tanner. Using predictive representations to improve generalization in reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 835–840, 2005.

Alfred Renyi. On measures of entropy and information. *Selected Papers of Alfred Renyi*, 2:525–580, 1976.

Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata. In *IEEE Symposium on the Foundations of Computer Science*, pages 78–87, 1987.

Matthew Rosencrantz, Geoff Gordon, and Sebastian Thrun. Learning low dimensional predictive representations. In *International Conference on Machine Learning (ICML)*, pages 695–702, 2004.

Sam Roweis and Zoubin Ghahramani. A unifying review of linear Gaussian models. *Neural Computation*, 11(2):305–345, 1999.

Matthew R. Rudary and Satinder Singh. A nonlinear predictive state representation. In *Neural Information Processing Systems (NIPS)*, pages 855–862, 2004.

Matthew R. Rudary and Satinder Singh. Predictive linear-Gaussian models of controlled stochastic dynamical systems. In *International Conference on Machine Learning (ICML)*, pages 777–784, 2006.

Matthew R. Rudary, Satinder Singh, and David Wingate. Predictive linear-Gaussian models of stochastic dynamical systems. In *Uncertainty in Artificial Intelligence (UAI)*, pages 501–508, 2005.

Stuart Russell, John Binder, and Daphne Koller. Adaptive probabilistic networks. Technical report, University of California at Berkeley, Berkeley, CA, 1994.

Yousef Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing, Boston, 1996.

Ali H. Sayed. *Fundamentals of Adaptive Filtering*. Wiley, 2003.

Nicol N. Schraudolph. Gradient-based manipulation of non-paramtric entropy estimates. *IEEE Transactions on Neural Networks*, 14(2):828–837, 2004.

Cosma R. Shalizi and James P. Crutchfield. Computational mechanics: Pattern and prediction, structure and simplicity. *Journal of Statistical Physics*, 104:817–879, 2001.

Cosma Rohilla Shalizi and Kristina Lisa Shalizi. Blind construction of optimal nonlinear recursive predictors for discrete sequences. In *Uncertainty in Artificial Intelligence (UAI)*, pages 504–511, 2004.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27: 379–423, 1948.

Hagit Shatkay and Leslie P. Kaelbling. Learning geometrically-constrained hidden Markov models for robot navigation: Bridging the geometrical-topological gap. *Journal of AI Research (JAIR)*, pages 167–207, 2002.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

Satinder Singh, Tommi Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *International Conference on Machine Learning (ICML)*, pages 284–292, 1994.

Satinder Singh, Michael R. James, and Matthew R. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Uncertainty in Artificial Intelligence (UAI)*, pages 512–519, 2004.

Satinder Singh, Michael Littman, Nicholas Jong, David Pardoe, and Peter Stone. Learning predictive state representations. In *International Conference on Machine Learning (ICML)*, pages 712–719, 2003.

R.D. Smallwood and E. J. Sondik. The optimal control of partially observable processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

E. J. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.

Andreas Stolcke and Stephen Omohundro. Hidden Markov Model induction by Bayesian model merging. In *Neural Information Processing Systems (NIPS)*, volume 5, pages 11–18, 1993.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

Richard S. Sutton and Brian Tanner. Temporal-difference networks. In *Neural Information Processing Systems (NIPS)*, pages 1377–1384, 2005.

Johan A. K. Suykens and Joos Vandewalle. *Nonlinear Modeling: Advanced Black-box Techniques*. Kluwer Academic Publishers, 1998.

Brian Tanner, Vadim Bulitko, Anna Koop, and Cosmin Paduraru. Grounding abstractions in predictive state representations. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1077–1082, 2007.

Brian Tanner and Richard Sutton. Td(lambda) networks: Temporal difference networks with eligibility traces. In *International Conference on Machine Learning (ICML)*, pages 888–895, 2005a.

Brian Tanner and Richard Sutton. Temporal difference networks with history. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 865–870, 2005b.

Graham W. Taylor, Geoffrey E. Hinton, and Sam T. Roweis. Modeling human motion using binary latent variables. In *Neural Information Processing Systems (NIPS)*, pages 1345–1352, 2007.

Sebastian Thrun, Daphne Koller, Zoubin Ghahramani, Hugh Durrant-Whyte, and Andrew Ng. Simultaneous mapping and localization with sparse extended information filters: Theory and initial results. Technical Report CMU-CS-02-112, Carnegie Mellon University, 2002.

Sebastian Thrun, Scott Thayer, William Whittaker, Christopher Baker, Wolfram Burgard, David Ferguson, Dirk Hahnel, Michael Montemerlo, Aaron Morris, Zachary Omohundro, Charlie Reverte, and Warren Whittaker. Autonomous exploration and mapping of abandoned mines. *IEEE Robotics and Automation*, 11(4), 2005.

Judith D. Toms and Mary L. Lesperance. Piecewise regression: A tool for identifying ecological thresholds. *Ecology*, 84(8):2034–2041, 2003.

Kari Torkkola. Feature extraction by non-parametric mutual information maximization. *Journal of Machine Learning Research (JMLR)*, 3:1415–1438, 2003.

Rudolph van der Merwe and Eric A. Wan. The square-root unscented Kalman filter for state and parameter-estimation. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 3461–3464, 2001.

Peter van Overschee and Bart De Moor. *Subspace Identification for Linear Systems: Theory, Implementation Applications*. Springer, 1996.

Michel Verhaegen and Paul Van Dooren. Numerical aspects of different Kalman filter implementations. *IEEE Transactions on Automatic Control*, AC-31(10):907–917, 1986.

Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. Technical Report 649, UC Berkeley, 2003.

Martin J. Wainwright and Michael I. Jordan. Log-determinant relaxation for approximate inference in discrete Markov random fields. *IEEE Transactions on Signal Processing*, 54(6):2099–2109, 2006.

Eric A. Wan and Rudolph van der Merwe. The unscented Kalman filter for nonlinear estimation. In *Proceedings of Symposium 2000 on Adaptive Systems for Signal Processing, Communication and Control*, 2000.

Andreas S. Weigend and Neil A. Gershenfeld. *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison-Wesley, Reading, MA, 1994.

Daan Wierstra and Marco Wiering. Utile distinction hidden Markov models. In *International Conference on Machine Learning (ICML)*, pages 108–115, 2004.

Eric Wiewiora. Learning predictive representations from a history. In *International Conference on Machine Learning (ICML)*, pages 964–971, 2005.

David Wingate and Kevin D. Seppi. Prioritization methods for accelerating MDP solvers. *Journal of Machine Learning Research (JMLR)*, 6:851–881, 2005.

David Wingate and Satinder Singh. Kernel predictive linear Gaussian models for nonlinear stochastic dynamical systems. In *International Conference on Machine Learning (ICML)*, pages 1017–1024, 2006a.

David Wingate and Satinder Singh. Mixtures of predictive linear Gaussian models for nonlinear stochastic dynamical systems. In *National Conference on Artificial Intelligence (AAAI)*, 2006b.

David Wingate and Satinder Singh. Exponential family predictive representations of state. In *Neural Information Processing Systems (NIPS)*, page (to appear), 2007a.

David Wingate and Satinder Singh. On discovery and learning of models with predictive representations of state for agents with continuous actions and observations. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1128–1135, 2007b.

David Wingate, Vishal Soni, Britton Wolfe, and Satinder Singh. Relational knowledge with predictive representations of state. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2035–2040, 2007.

Britton Wolfe, Michael R. James, and Satinder Singh. Learning predictive state representations in dynamical systems without reset. In *International Conference on Machine Learning*, pages 980–987, 2005.

Britton Wolfe and Satinder Singh. Predictive state representations with options. In *International Conference on Machine Learning (ICML)*, pages 1025–1032, 2006.

Changjiang Yang, Ramani Duraiswami, Nail Gumerov, and Larry Davis. Improved fast Gauss transform and efficient kernel density estimation. In *IEEE International Conference on Computer Vision*, pages 464–471, 2003.

Jonathan S. Yedida, William T. Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. Technical Report TR-2001-22, Mitsubishi Electric Research Laboratories, 2001.

Paul Zarchan and Howard Musoff. *Fundamentals of Kalman Filtering: A Practical Approach*. Progress in Astronautics and Aeronautics, Volume 208, 2005.