Deep Neural Networks for Visual Reasoning, Program Induction, and Text-to-Image Synthesis

by

Scott Ellison Reed

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
2016

Doctoral Committee:

Assistant Professor Honglak Lee, Chair
Professor Benjamin Kuipers
Assistant Professor Emily Mower Provost
Associate Professor Clayton Scott

ACKNOWLEDGEMENTS

I would like to sincerely thank all of the CSE faculty and staff, and family and friends whose support made this thesis possible. There are too many people to list all of them here, but I will highight a few:

Dumitru Erhan, Christian Szegedy, Andrew Rabinovich, Dragomir Anguelov and many others who were on the Photos and Brain team at Google when I was there. Their relentless focus on developing vision systems that *actually work*, motivated directly by solving problems for millions of people, resulted in high-impact research breakthroughs in large-scale visual recognition and detection. Although our work on large-scale object detection is not part of my thesis, working with them had a transformative impact on my research philosophy and subsequent productivity.

Nando de Freitas at DeepMind, who inspired me to work on neural program "libraries", which led to our joint work on Neural Programmer-Interpreters.

My adviser and dissertation committee chair Honglak Lee, who I have worked with since the start of my PhD, who gave lots of valuable research advice and guidance on many successful projects, and pushed me to accomplish more in graduate school.

Benjamin Kuipers, Emily Mower-Provost, and Clayton Scott who generously served on my dissertation committee and provided very valuable feedback on this dissertation and thought-provoking research discussions.

Thanks to all of my lab mates for many great collaborations and discussions: Kihyuk Sohn, Yuting Zhang, Xinchen Yan, Lajanugen Logeswaran, Yi Zhang, Junhyuk Oh, Ruben Villegas, Kibok Lee, Rui Zhang, Changhan Wang, Jimei Yang, Roni Mittelman, Wenling

Shang, Daniel Walter, Yong Peng, Forest Agostinelli, Sami Abu-El-Haija, Kyoungah Kim, Chansoo Lee.

Zeynep Akata and Bernt Schiele at the Max Planck Institute for Informatics, with whom we had three very enjoyable and successful collaborations.

This work was supported mainly by a Rackham Fellowship during my first year, NSF GRFP under Grant No. DGE 1256260 during my second and third years, and also by NSF CAREER IIS-1453651, ONR N00014-13-1-0762 and NSF CMMI-1266184.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF APPENDICES

ABSTRACT

Deep Neural Networks for Visual Reasoning, Program Induction,
and Text-to-Image Synthesis

by

Scott Ellison Reed

Chair: Honglak Lee

Deep neural networks excel at pattern recognition, especially in the setting of large scale supervised learning. A combination of better hardware, more data, and algorithmic improvements have yielded breakthroughs in image classification, speech recognition and other perception problems. The research frontier has shifted towards the weak side of neural networks: reasoning, planning, and (like all machine learning algorithms) creativity. How can we advance along this frontier using the same generic techniques so effective in pattern recognition; i.e. gradient descent with backpropagation? In this thesis I develop neural architectures with new capabilities in visual reasoning, program induction and text-to-image synthesis. I propose two models that disentangle the latent visual factors of variation that give rise to images, and enable analogical reasoning in the latent space. I show how to augment a recurrent network with a memory of programs that enables the learning of compositional structure for more data-efficient and generalizable program induction. Finally, I develop a generative neural network that translates descriptions of birds, flowers and other categories into compelling natural images.

CHAPTER I

Introduction

1.1 Motivation

Deep neural networks have enabled transformative breakthroughs in speech and image recognition in the past several years, fueled by increases in training data and computational power in addition to algorithmic improvements. While deep networks excel at pattern recognition, often with comparable performance to humans in some tasks, the research frontier has shifted to the current weak side of neural networks: reasoning, planning and creativity. In this thesis I propose several approaches to advance along this frontier.

Reasoning and planning are the subject of decades of research in artificial intelligence. The classical approaches rely mainly on symbolic representations of the world; for search-based problem solving and game playing, logical reasoning systems, theorem proving, and many planning problems [129]. For problems involving high-dimensional, noisy perceptual data such as image classification and speech recognition, connectionist approaches (i.e. neural networks) have enjoyed the most success. Further progress in AI can likely be found at the intersection of these fairly disparate research communities. Service robots and digital assistants will need connectionist approaches to distill useful representations of images, speech and text. They will also require symbolic notions; e.g. a knowledge graph of grounded concepts and their relationships, in order to function in the real world. Many research problems will require aspects of both; one that I consider inthis thesis is visual

analogy making.

In addition to reasoning and planning, the notion of *creativity* I refer to in the context of this work should be specified. In this work I am only referring to the creation of something novel and of interest to humans in the context of conditional generative models of images. I explore this in several ways; mainly by (1) conditioning a generator network on a *disentangled* representation of causal factors, which can be recombined arbitrarily and thereby allow the model to generalize combinatorially, and (2) conditioning on informal text descriptions. Clearly this is only a small part of what we mean colloquially by creativity, but research progress here could apply to automatic generation of novel and useful artifacts in other domains as well.

The first problem I study is how to reason about the relevant visual factors of variation that give rise to an image. Given an image of an object, one would like for an image understanding system to not only be able to robustly recognize the object under variations in lighting, pose and scale; but also predict what the scene would look like if one or several of those factors were changed. I develop a variant of Restricted Boltzmann Machine [135] (RBM) that explicitly separates the latent factors into separate groups of units, and apply the model to pose and expression transfer on human faces.

To investigate the reasoning capability of neural networks, I develop a model for visual analogy making: given an image analogy problem A : B :: C : ?, the network predicts the pixels of the image D that completes the analogy. For example, the analogy could involve rotating 3D shapes or animating a video game sprite. In contrast to previous works on analogy-making, this is the first end-to-end differentiable architecture for pixel-level analogy completion. We also show that by learning to disentangle the latent visual factors of variation (e.g. pose and shape), our model can more effectively relate images and perform image transformations.

To improve neural-network-based planning (e.g. mapping percepts to action sequences), I propose a modification to recurrent neural networks that enables them to capture com-

positional structure in the output space. Our proposed network, the Neural Programmer-Interpreter (NPI) is augmented with a memory of program, each consistning of environment-dependent actions and calls to other programs. It learns to execute these programs from example execution traces. By exploiting compositionality, we demonstrate improved data efficiency and strong generalization compared to previous recurrent networks for program induction. We apply our model, the Neural Programmer-Interpreter (NPI), to generating execution trajectories for multi-digit addition, sorting arrays of numbers, and canonicalizing the pose of 3D car models from image renderings. Notably, a single NPI can learn and execute these programs and associated subprograms across very different environments with different affordances.

To more accurately capture the relation between images and text, in another project I implement and evaluate several deep architectures for encoding text descriptions of birds and flowers. We show that using only informal text descriptions, we can learn highly discriminative text features comparable in performance to carefully-engineered and domain-specific attribute vector representations. Next, we leverage these text encoders to improve the usefulness of neural nets for creative tasks. Specifically, I develop several new variants of Generative Adversarial Networks capable of text-to-image synthesis; i.e. generating plausible images from informal descriptions. For example, "a bright yellow bird with a black head and beak". Our system can generate plausible depictions of birds, flowers and many other objects given only textual descriptions. By learning to invert the generator network, we also show how to synthesize images by transferring the style of an unseen photograph (e.g. background appearance) onto the content of a text description.

1.1.1 The disentangling Restricted Boltzmann Machine (Chapter III)

The Restricted Boltzmann Machine [135] (RBM) has gained popularity as an effective model for natural images and image patches [85, 62], and as a fundamental building block for stacking into deep multi-layer architectures and various deep generative models [130].

However, the standard RBM formulation only models the pairwise interactions between units in a single observation vector and units in a single hidden unit vector. For the purpose of modeling higher-order interactions among multiple factors of variation given a single image observation, e.g. pose, expression and identity of human faces, in this project we extend the basic RBM formulation.

We want to preserve the nice properties of the RBM; i.e. efficient and parallelizable inference, simple procedures for generating image samples, and efficient approximation to the likelihood gradient. In addition to preserving these properties, we want to endow the model with new generative capabilities; namely the ability to fix a subset of factors of variation while varying the others, thus traversing image manifolds in a controllable way. Our main contributions are (1) an RBM formulation in which the energy function incorporates both additive and multiplicative interactions among latent factors, (2) a new regularizer encouraging distinct groups of hidden units to represent different concepts (i.e. become disentangled), and (3) a simple and effective method of training the model via backpropagation by casting its inference procedure as a recurrent neural network.

1.1.2 Deep networks for disentangling and visual analogy-making (Chapter IV)

While our disentangling extension to the RBM showed encouraging results, it did not take advantage of recent advances in deep convolutional networks for modeling images. Therefore, in this line of work we explore a deep convolutional model that also learns a disentangled latent representation. Instead of directly optimizing the likelihood for training a fully generative model, we set as the target the actual generative tasks that we would like our model to perform: namely, making analogies [56] and traversing image manifolds [31]. For example, in analogy making we may want to transform a query face image to have the same expression as an example face. For manifold traversal, given a query face, we may want to repeatedly rotate the face in 3D to produce images from novel viewpoints. Since faces were studied in our disentangling Boltzmann machine work, in this work we

studied several non-face datasets: a toy set of colored 2D shapes for control experiments, a set of video game character sprites with many controllable attributes, and a set of 3D car CAD models. We study several transformation mechanisms for our deep convolutional encoder-decoder model: additive, multiplicative and deep multilayer. We demonstrate that a deep multilayer transformation mechanism achieves the best performance in extrapolating repeated image transformations along a manifold, such as repeated rotations of 2D shapes and 3D cars and even extrapolations of sprite animations.

1.1.3 Learning a neural programmer and interpreter (Chapter V)

Visual analogies can be viewed as a form of one-shot program induction followed by execution of that same program on a new query image. In this case, the "program", e.g. rotating an object by 15°, can be represented by a difference of embeddings between the example input and output image transformation pair, as we show in Chapter IV. However, more complex programs with multiple steps and compositional structure are unlikely to be solvable by such a simple approach. Recurrent neural networks (sequence-to-sequence models) have shown some capability to learn simple programs such as sorting very short arrays or addition of binary numbers, but have not scaled to learning more complex tasks and suffer from poor generalization ability.

To tackle this problem, I show how to construct a recurrent neural network with a persistent memory of program embeddings, which I call the Neural Programmer-Interpreter (NPI). At each time step of processing, NPI outputs the next program, optional arguments, and whether to halt the current program, conditioned on a feature representation of the current environment state and the current program. NPI trains on program execution traces consisting of calls of each program to its immediate subprograms (*not* the entire execution subtree) conditioned on the input. I demonstrate that a *single* NPI can learn programs for addition, sorting and canonicalization of 3D car models, and all 21 associated subprograms. By exploiting compositionality, NPI achieves improved sample complexity and stronger

generalization performance compared to baseline sequence-to-sequence models.

1.1.4 Learning to represent fine-grained visual descriptions (Chapter VI)

State-of-the-art methods for zero-shot visual recognition formulate learning as a joint embedding problem of images and side information. In these formulations the current best complement to visual features are attributes: manually-encoded vectors describing shared characteristics among categories. Despite good performance, attributes have limitations: (1) finer-grained recognition requires commensurately more attributes, and (2) attributes do not provide a natural language interface. We propose to overcome these limitations by training neural language models from scratch; i.e. without pre-training and only consuming words and characters. Our proposed models train end-to-end to align with the fine-grained and category-specific content of images. Natural language provides a flexible and compact way of encoding only the salient visual aspects for distinguishing categories. By training on raw text, our model can do inference on raw text as well, providing humans a familiar mode both for annotation and retrieval. Our model achieves strong performance on zero-shot text-based image retrieval and significantly outperforms the attribute-based state-of-the-art for zero-shot classification on the Caltech-UCSD Birds 200-2011 dataset.

1.1.5 Generating images from informal text descriptions (Chapter VII)

Automatic synthesis of realistic images from text would be interesting and useful, but current AI systems are still far from this goal. However, in recent years generic and powerful recurrent neural network architectures have been developed to learn discriminative text feature representations. Meanwhile, deep convolutional generative adversarial networks (GANs) have begun to generate highly compelling images of specific categories such as faces, album covers, room interiors and flowers. In this work, we develop a novel deep architecture and GAN formulation to effectively bridge these advances in text and image modeling, translating visual concepts from characters to pixels. We demonstrate the ca-

pability of our model to generate plausible images of birds and flowers from detailed text descriptions. We also extend our model to a more general dataset of captioned web images.

1.2 Organization of the Thesis

This thesis is organized as follows. In chapter II, I provide background information about restricted Boltzmann machines (RBMs) and Long short-term Memory neural networks (LSTMs). In chapter III, I describe an extension of the RBM that learns to disentangle factors of variation from image data, resulting in improved discriminative performance and yielding new generative capabilities. In chapter IV, I show how deep convolutional networks can be used to learn to disentangle visual factors of variation and also perform visual analogies. In chapter V I show how to learn representations of compositional programs, and demonstrate their effectiveness on addition, sorting, and camera trajectory planning for rotating 3D CAD models. In chapter VI I describe our work on learning deep representations of fine-grained visual descriptions, which achieve superior predictive performance even compared to costly attribute annotations. These sentence embeddings are then used to help generate images. In chapter VII I provide an overview of our text-to-image synthesis model and show results on generating birds, flowers and common scenes.

1.3 Related Publications

The content of this thesis is mostly derived from papers that were published in top-tier machine learning and computer vision conferences. The list that follows connects these publications to their corresponding chapter.

- Chapter III: Scott Reed, Kihyuk Sohn, Yuting Zhang, Honglak Lee. Learning to Disentangle Factors of Variation with Manifold Interaction. In *Proceedings of the 31st International Conference on Machine Learning*, Beijing, China, 2014.

- Chapter IV: Scott Reed, Yi Zhang, Yuting Zhang, Honglak Lee. Deep Visual Analogy-Making. In *Neural Information Processing Systems*, Montreal, Canada, 2015.

- Chapter V: Scott Reed, Nando de Freitas. Neural Programmer-Interpreters. In *International Conference on Learning Representations*, San Juan, Puerto Rico, 2016.

- Chapter VI: Scott Reed, Zeynep Akata, Honglak Lee, Bernt Schiele. Learning Deep Representations of Fine-grained Visual Descriptions. In *IEEE Computer Vision and Pattern Recognition*, Las Vegas, USA, 2016.

- Chapter VII: Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee. Generative Adversarial Text-to-Image Synthesis. Under review for *Proceedings of the 31st International Conference on Machine Learning*, New York, USA, 2016.

In addition, the following co-author publications are also closely related to the material in this thesis. These are also published at top-tier machine learning and vision conferences, and are linked in the following list to the related chapter:

- Chapter IV: Jimei Yang, Scott Reed, Ming-hsuan Yang, Honglak Lee. Weakly-supervised Disentangling with Recurrent Transformations for 3D View Synthesis. In *Neural Information Processing Systems*, Montreal, Canada, 2015.

- Chapter VI: Zeynep Akata, Scott Reed, Daniel Walter, Honglak Lee, Bernt Schiele. Evaluation of Output Embeddings for Fine-Grained Image Classification In *IEEE Computer Vision and Pattern Recognition*, Boston, USA, 2015.

CHAPTER II

Preliminaries

In this section I briefly review two building blocks used in this dissertation. The first is a type of probabilistic graphical model, the RBM, that recently gained popularity in modeling image patches, as well as audio and text data. I extend this model in Chapter III. The second is a version of recurrent neural network designed to mitigate the vanishing and exploding gradients problem. This module is used in Chapters V, VI and VII.

2.1 Restricted Boltzmann machine

The restricted Boltzmann machine (RBM) is a bipartite undirected graphical model composed of D binary visible units[1] $\mathbf{v} \in \{0,1\}^D$ and K binary hidden units $\mathbf{h} \in \{0,1\}^K$. The joint distribution and the energy function are defined as follows:

$$
\begin{aligned}
P(\mathbf{v},\mathbf{h}) = & \ \frac{1}{Z}\exp(-E(\mathbf{v},\mathbf{h})), \\
E(\mathbf{v},\mathbf{h}) = & \ -\sum_{i=1}^{D}\sum_{k=1}^{K} v_i W_{ik} h_k - \sum_{k=1}^{K} b_k h_k - \sum_{i=1}^{D} c_i v_i,
\end{aligned}
$$

where Z is the partition function, W_{ik} is a weight between i-th visible and k-th hidden units, b_k are hidden biases, and c_i are visible biases. In the RBM, the units in the same

[1] The RBM can be extended to model the real-valued visible units [60].

layer are conditionally independent given the units in the other layer:

$$P(v_i = 1 \mid \mathbf{h}) = \sigma(\sum_k W_{ik} h_k + c_i),$$
$$P(h_k = 1 \mid \mathbf{v}) = \sigma(\sum_i W_{ik} v_i + b_k),$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is a logistic function. The RBM can be trained to maximize the log-likelihood of data using stochastic gradient descent. Although the gradient is intractable, we can approximate it using contrastive divergence (CD) [58].

2.2 Long short-term memory (LSTM)

The LSTM [61] is a recurrent model designed to overcome limitations of the basic RNN, namely the problem of exploding and vanishing gradients. The LSTM stores state information in its memory cell $\mathbf{c} \in \mathbb{R}^d$, which is written to and read from via pointwise multiplications with several gating variables: input gate $\mathbf{i}$, forget gate $\mathbf{f}$ and output gate $\mathbf{o}$, which yields the output hidden state $\mathbf{h} \in \mathbb{R}^d$. Having observed input $\mathbf{x}$, the overall update step at time t proceeds as follows:

$$i_t = \sigma(W_{ix} x_t + W_{ih} h_{t-1}) \tag{2.1}$$
$$f_t = \sigma(W_{fx} x_t + W_{fh} h_{t-1}) \tag{2.2}$$
$$o_t = \sigma(W_{ox} x_t + W_{oh} h_{t-1}) \tag{2.3}$$
$$c_t = f_t \odot c_{t-1} + i_t tanh(W_{cx} x_t + W_{ch} h_{t-1}) \tag{2.4}$$
$$h_t = o_t \odot c_t \tag{2.5}$$

where the $W_{...}$ are learnable weights, $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function and $\odot$ denotes pointwise multiplication. Intuitively, LSTMs can model very long temporal dependences by learning to gate the memory cell updates. LSTMs are commonly used for sequence modeling tasks such as next-word prediction. For example, one could compute $p_{t+1} = \text{Softmax}(W_{hp} h_t)$ as the next-word probability over a vocabulary.

CHAPTER III

Disentangling factors of variation with Restricted Boltzmann Machines

3.1 Introduction

A key challenge in understanding sensory data (e.g., image and audio) is to tease apart many factors of variation that combine to generate the observations [15]. For example, pose, shape and illumination combine to generate 3D object images; morphology and expression combine to generate face images. Many factors of variation exist for other modalities, but here we focus on modeling images.

Most previous work focused on building [90] or learning [71, 116, 85, 83, 63, 62, 137] invariant features that are unaffected by nuisance information for the task at hand. However, we argue that image understanding can benefit from retaining information about all underlying factors of variation, because in many cases knowledge about one factor can improve our estimates about the others. For example, a good pose estimate may help to accurately infer the face morphology, and vice versa.

When the input images are generated from multiple factors of variation, they tend to lie on a complicated manifold, which makes learning useful representations very challenging. We approach this problem by viewing each factor of variation as forming a sub-manifold by itself, and modeling the joint interaction among factors. For example, given face images

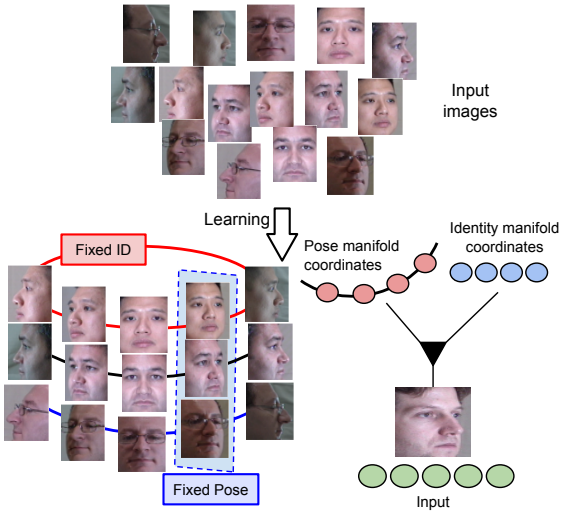


Figure 3.1: Illustration of our approach for modeling pose and identity variations in face images. When fixing identity, traversing along the corresponding "fiber" (red ellipse) changes the pose. When fixing pose, traversing across the vertical cross-section (blue rectangle) changes the identity. Our model captures this via multiplicative interactions between pose and identity coordinates to generate the image.

with different identities and viewpoints, we can envision one sub-manifold for identity and another for viewpoint. As illustrated in Figure 3.1, when we consider face images of a single person taken from different azimuth angles (with fixed altitude), the trajectory of images will form a ring-shaped fiber. Similarly, changing the identity while fixing the angle traverses a high-dimensional sub-manifold from one fiber to other.

Concretely, we use a higher-order Boltzmann machine to model the distribution over image features and the latent factors of variation. Further, we propose correspondence-based training strategies that allow our model to effectively *disentangle* the factors of variation. This means that each group of hidden units is sensitive to changes in its corresponding factor of variation, and relatively invariant to changes in the others. We refer to our model variants as *disentangling Boltzmann machines* (disBMs). Our disBM model achieves state-of-the-art emotion recognition and face verification performance on the Toronto Face Database (TFD), as well as strong performance in pose estimation and face verification on CMU Multi-PIE.

3.2 Related Work

Manifold learning methods [152, 126, 54] model the data by learning low-dimensional structures or embeddings. Existing manifold learning methods can learn low-dimensional structures such as viewpoint manifolds from face images of a single person, but it becomes challenging to model complex high-dimensional manifolds such as the space of face images from millions of people. Deep learning has shown to be effective in learning such high-dimensional data manifolds, as suggested by *Rifai et al.* [122]. However, it remains a challenge to jointly model multiple factors of variation and their interacting manifolds.

Our work is related to multi-task learning [17, 6] if one views each factor as a "task" feature to be learned jointly. However, our approach considers joint interaction among the factors, and benefits from a synergy in which knowledge of one factor can help infer about the others. In addition, our model is generative and can answer higher-order queries involving the input and multiple factors.

There are several related works that use higher-order interactions between multiple latent variables. For example, bilinear models [151] were used to separate style and content within face images (pose and identity) and speech signals (vowels and speaker identity). The tensor analyzer (TA) [150] extended factor analysis by introducing a factor loading tensor to model the interaction among multiple groups of latent factor units, and was applied to modeling lighting and face morphology. Our approach is complementary to these, and is also capable of exploiting correspondence information.

The higher-order spike and slab RBM (ssRBM) [28] extends the ssRBM [23] with higher-order interactions. Our motivation is similar, but our model formulation is different and we propose novel training strategies that significantly improve the disentangling. Finally, we show state-of-the-art performance on several discriminative tasks on face images.

The factored gated Boltzmann machine (FGBM) [94, 144] models the relation between data pairs (e.g. translation, rotation of images, facial expression changes) via 3-way interactions. Both the FGBM and disBM are variants of higher-order Boltzmann machines,

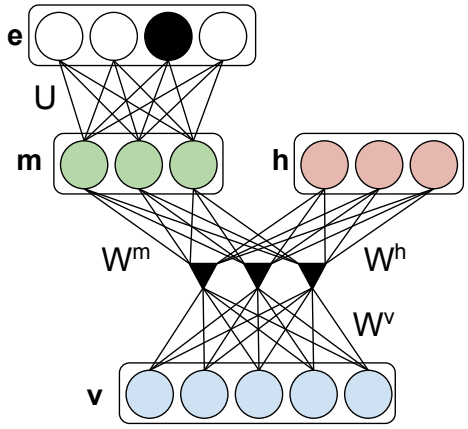


Figure 3.2: An instance of our model with two groups of hidden units. We can optionally include label units (e.g., labels $\mathbf{e}$ are connected to hidden units $\mathbf{m}$).

but the FGBM assumes two sets of visible units interacting with one set of hidden units, whereas the disBM assumes multiple sets of hidden units interacting with a single set of visible units.

The point-wise gated Boltzmann machine [138] is an instance of a higher-order Boltzmann machine that jointly learns and selects task-relevant features. Contractive discriminative analysis [123] also learns groups of task-relevant and irrelevant hidden units using a contractive penalty, but only uses additive interactions between the input and each group of hidden units. These models are complementary to ours in that they learn to separate task-relevant from task-irrelevant features.

The disBM is an undirected graphical model with higher-order interactions between observations and multiple groups of hidden units, as in Figure 3.2. Each group of hidden units can be viewed as manifold coordinates for a distinct factor of variation. Our proposed model is shown in Figure 3.2. For simplicity, we assume two groups of hidden units $\mathbf{h}$ and $\mathbf{m}$, although it is straightforward to add more groups. If labels are available, they can be incorporated with the $\mathbf{e}$ units.

3.2.1 Energy function

As shown in Figure 3.2, our model assumes 3-way multiplicative interaction between D visible units $\mathbf{v} \in \{0,1\}^D$ and two groups of hidden units $\mathbf{h} \in \{0,1\}^K$ and $\mathbf{m} \in \{0,1\}^L$. We define the energy function as:

$$\begin{aligned} E(\mathbf{v},\mathbf{m},\mathbf{h}) = &-\sum_f (\sum_i W^v_{if} v_i)(\sum_j W^m_{jf} m_j)(\sum_k W^h_{kf} h_k) \\ &- \sum_{ij} P^m_{ij} v_i m_j - \sum_{ik} P^h_{ik} v_i h_k \end{aligned} \tag{3.1}$$

We have used factorization of 3D weight tensor $W \in \mathbb{R}^{D\times L\times K}$ into three weight matrices $W^v \in \mathbb{R}^{D\times F}$, $W^m \in \mathbb{R}^{L\times F}$, $W^h \in \mathbb{R}^{K\times F}$ with F factors as

$$W_{ijk} = \sum_{f=1}^{F} W^v_{if} W^m_{jf} W^h_{kf} \tag{3.2}$$

to reduce the number of model parameters [94]. We also include additive connections with weight matrices $P^m \in \mathbb{R}^{D\times L}$ and $P^h \in \mathbb{R}^{D\times K}$ between visible units and each group of hidden units. We omit the bias terms for clarity of presentation. Although the hidden units are not conditionally independent given the visible units, units in each group are conditionally independent given units in all other groups. The conditional distributions are as follows:[1]

$$\begin{aligned} P(v_i = 1 \mid \mathbf{h},\mathbf{m}) = \sigma(&\sum_{jk} W_{ijk} m_j h_k \\ &+ \sum_j P^m_{ij} m_j + \sum_k P^h_{ik} h_k) \end{aligned} \tag{3.3}$$

$$P(m_j = 1 \mid \mathbf{v},\mathbf{h}) = \sigma(\sum_{ik} W_{ijk} v_i h_k + \sum_i P^m_{ij} v_i) \tag{3.4}$$

$$P(h_k = 1 \mid \mathbf{v},\mathbf{m}) = \sigma(\sum_{ij} W_{ijk} v_i m_j + \sum_i P^h_{ik} v_i) \tag{3.5}$$

[1] W_{ijk} denotes factorized weights as in Equation (3.2).

The conditional independence structure allows efficient 3-way block Gibbs sampling.

3.2.2 Inference and learning

Inference. The exact posterior distribution is intractable since $\mathbf{h}$ and $\mathbf{m}$ are not conditionally independent given $\mathbf{v}$. Instead, we use variational inference to approximate the true posterior with a fully factorized distribution $Q(\mathbf{m}, \mathbf{h}) = \prod_j \prod_k Q(m_j)Q(h_k)$. A detailed derivation is presented in Appendix A. By minimizing KL $(Q(\mathbf{m}, \mathbf{h}) \| P(\mathbf{m}, \mathbf{h} \mid \mathbf{v}))$, we obtain the following fixed-point equations:

$$\hat{h}_k = \sigma(\sum_{ij} W_{ijk} v_i \hat{m}_j + \sum_i P^h_{ik} v_i) \tag{3.6}$$

$$\hat{m}_j = \sigma(\sum_{ik} W_{ijk} v_i \hat{h}_k + \sum_i P^m_{ij} v_i) \tag{3.7}$$

where $\hat{h}_k = Q(h_k = 1)$ and $\hat{m}_j = Q(m_j = 1)$. Initialized with all 0's, the mean-field update proceeds by alternately updating $\hat{\mathbf{h}}$ and $\hat{\mathbf{m}}$ using Equation (3.6) and (3.7) until convergence. We found that 10 iterations were enough in our experiments.

Learning. We train the model to maximize the data log-likelihood using stochastic gradient descent. The gradient of the log-likelihood for parameters $\Theta = \{W^v, W^m, W^h, P^m, P^h\}$ can be computed as:

$$-\mathbb{E}_{P(\mathbf{m},\mathbf{h}|\mathbf{v})}\left[\frac{\partial E(\mathbf{v}, \mathbf{m}, \mathbf{h})}{\partial \theta}\right] + \mathbb{E}_{P(\mathbf{v},\mathbf{m},\mathbf{h})}\left[\frac{\partial E(\mathbf{v}, \mathbf{m}, \mathbf{h})}{\partial \theta}\right]$$

Unlike in the RBM case, both the first (i.e., data-dependent) and the second (i.e., model-dependent) terms are intractable. We can approximate the data-dependent term with variational inference and the model-dependent term with persistent CD [153] by running a 3-way sampling using Equation (3.3),(3.4),(3.5). A similar approach has been proposed for training general Boltzmann machines [130].

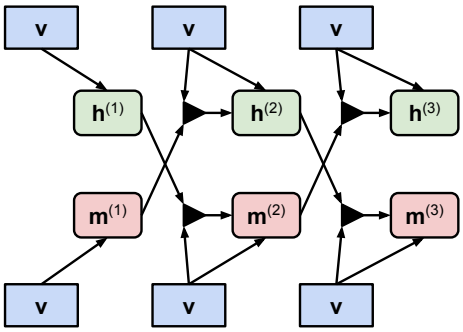


Figure 3.3: Visualization of the RNN structure of our model. Arrows show the direction of the forward propagation.

3.2.3 Computing gradients via backpropagation

When the training objective depends on hidden unit activations, such as correspondence (Section 3.3.1) or sparsity [84, 59], the exact gradient can be computed via backpropagation through the recurrent neural network (RNN) induced by mean-field inference (See Figure 3.3). The forward propagation proceeds as:

$$\hat{h}_k^{(t+1)} = \sigma(\sum_{ij} W_{ijk} v_i \hat{m}_j^{(t)} + \sum_i P_{ik}^h v_i) \tag{3.8}$$

$$\hat{m}_j^{(t+1)} = \sigma(\sum_{ik} W_{ijk} v_i \hat{h}_k^{(t)} + \sum_i P_{ij}^m v_i) \tag{3.9}$$

A similar strategy was rigorously developed by *Stoyanov et al.* [141] and was used to train deep Boltzmann machines [48].

3.3 Training strategies for disentangling

Generative training of the disBM does not explicitly encourage disentangling, and generally did not yield well-disentangled features in practice. However, we can achieve better disentangling by exploiting correspondences between images (e.g. matching identity, expression or pose), and by using labels.

3.3.1 Learning with correspondence

Clamping hidden units for pairs

If we know two data points $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ match in some factor of variation, we can "clamp" the corresponding hidden units to be the same for both data points. For example, given two images from the same person, we clamp the $\mathbf{h}$ units so that they focus on modeling the common face morphology while other hidden units explain the differences such as pose or expression. To do clamping, we augment the energy function as follows:

$$\begin{aligned} & E_{\text{clamp}}(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \mathbf{m}^{(1)}, \mathbf{m}^{(2)}, \mathbf{h}) \\ & \quad = \; E(\mathbf{v}^{(1)}, \mathbf{m}^{(1)}, \mathbf{h}) + E(\mathbf{v}^{(2)}, \mathbf{m}^{(2)}, \mathbf{h}) \end{aligned} \tag{3.10}$$

The fixed-point equations are the same as before, except that Equation (3.6) changes to reflect the contributions from both $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$:

$$\begin{aligned} \hat{h}_k = \sigma(\sum_{ij} W_{ijk} v_i^{(1)} \hat{m}_j^{(1)} + \sum_i P_{ik}^h v_i^{(1)} \\ + \sum_{ij} W_{ijk} v_i^{(2)} \hat{m}_j^{(2)} + \sum_i P_{ik}^h v_i^{(2)}) \end{aligned} \tag{3.11}$$

The model is trained to maximize the joint log-likelihood of data pairs $\log P(\mathbf{v}^{(1)}, \mathbf{v}^{(2)})$.

Manifold-based training

In the manifold learning perspective, we want each group of hidden units to be a useful embedding with respect to its factor of variation. Specifically, corresponding data pairs should be embedded nearby, while the non-corresponding data pairs should be far apart. Clamping forces corresponding pairs into exactly the same point within a sub-manifold, which may be too strong of an assumption depending on the nature of the correspondence. Furthermore, clamping does not exploit knowledge of non-correspondence. Instead, we

propose to learn a representation $\mathbf{h}$ such that

$$
\begin{aligned}
||\mathbf{h}^{(1)} - \mathbf{h}^{(2)}||_2^2 &\approx 0 \quad , \text{ if } (\mathbf{v}^{(1)}, \mathbf{v}^{(2)}) \in \mathcal{D}_{sim} \\
||\mathbf{h}^{(1)} - \mathbf{h}^{(3)}||_2^2 &\geq \beta \quad , \text{ if } (\mathbf{v}^{(1)}, \mathbf{v}^{(3)}) \in \mathcal{D}_{dis}
\end{aligned}
$$

where $\mathcal{D}_{sim}$ is a set of corresponding data pairs and $\mathcal{D}_{dis}$ is a set of non-corresponding data pairs. Formally, the manifold objective for $\mathbf{h}$ is written as:

$$
||\mathbf{h}^{(1)} - \mathbf{h}^{(2)}||_2^2 + \max(0, \beta - ||\mathbf{h}^{(1)} - \mathbf{h}^{(3)}||_2)^2 \tag{3.12}
$$

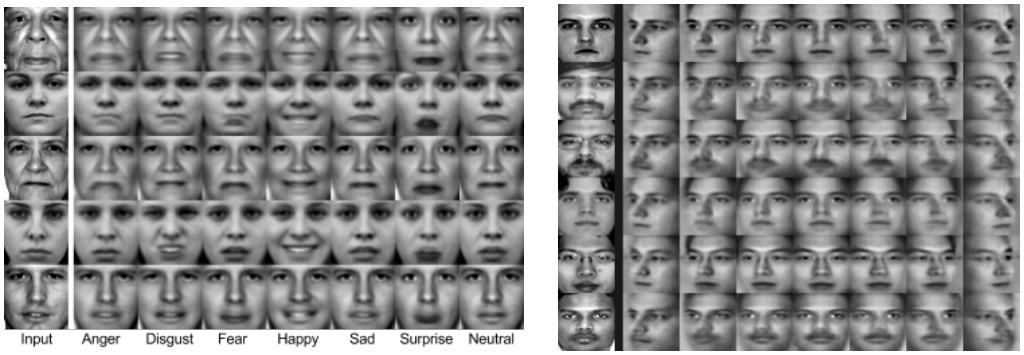
This approach does not directly use label units, but labels can be used to construct correspondence sets $\mathcal{D}_{sim}$ and $\mathcal{D}_{dis}$. The formulation is similar to the one proposed by *Hadsell et al.* [54]. However, our goal is not dimensionality reduction and we consider multiple factors of variation jointly. Furthermore, we can combine the manifold objective together with the generative objective. Since our model uses mean-field inference to compute the hidden units, we compute gradients via RNN backpropagation as discussed in Section 3.2.3.

3.4 Experiments on face image data sets

We evaluated the performance of our proposed model on several image databases:

- **Flipped MNIST**. For each digit of the MNIST dataset, we randomly flipped all pixels (0's to 1's and vice versa) with 50% probability. The dataset consists of 50,000 training images, 10,000 validation images, and 10,000 test images.
- **Toronto Face Database (TFD)** [143]. Contains $112,234$ face images with $4,178$ emotion labels and $3,874$ identity labels. There are seven possible emotion labels.
- **CMU Multi-PIE** [52]. Contains $754,200$ high-resolution face images with variations in pose, lighting, and expression. We manually aligned and cropped the face images.[2]

[2]We annotated two or three fiducial points (e.g., the eyes, nose, and mouth corners) and computed the 2-D similarity transform that best fits them to the predefined anchor locations, which are different for each pose. Then, we warped the image accordingly, and cropped the major facial region with a fixed 4:3 rectangular box.



(a) Expression manifold traversal on TFD (b) Pose manifold traversal on MPIE

Figure 3.4: Visualization of (a) expression and (b) pose manifold traversal. Each row shows samples of varying expressions or pose with same identity as in input (leftmost).

3.4.1 Reasoning about factors of variation

A good generative model that can disentangle factors of variation should be able to traverse the manifold of one factor while fixing the states of the others. For the case of face images, the model should be able to generate examples with different pose or expression while fixing the identity. It should also be able to interpolate within a sub-manifold (e.g. across pose) and transfer the pose or expression of one person to others. *Bengio et al.* [16] showed that linear interpolation across deep representations can traverse closer to the image manifold compared to shallow representations. We would like our model to have these properties with respect to *each* factor of variation separately.

To verify that our model has these properties, we constructed a 2-layer deep belief network (DBN), where the first layer is a Gaussian RBM with tiled overlapping receptive fields similar to those used by *Ranzato et al.* [117] and the second layer is our proposed disBM. For TFD, our model has identity-related hidden units $\mathbf{h}$ and expression-related hidden units $\mathbf{m}$. For Multi-PIE, our model has identity-related units $\mathbf{h}$ and pose-related units which we will also denote $\mathbf{m}$. For some control experiments we also use label units $\mathbf{e}$, corresponding to 1-of-7 emotion labels in TFD and 1-of-15 pose labels in Multi-PIE.

We resized the cropped grayscaled images into 48×48.

(a) Expr. transfer. (b) Pose transfer.

Figure 3.5: Identity units from left column are transferred to (a) expression units and (b) pose units from middle column. Reconstructions shown in right columns.

We first examined how well the disBM traverses the pose or expression manifolds while fixing identity. Given an input image $\mathbf{v}$ we perform posterior inference to compute $\mathbf{h}$ and $\mathbf{m}$. Then we fixed the pose or emotion label units $\mathbf{e}$ to the target and performed Gibbs sampling between $\mathbf{v}$ and $\mathbf{m}$. Example results are shown in Figure 3.4(a) and 3.4(b). Each row shows input image and its generated samples after traversing to the specific target emotion or pose. The identity of the input face image is well preserved across the rows while expressing the correct emotion or pose.

We also performed experiments on pose and expression transfer. The task is to transfer the pose or expression of one image onto the person in a second image. Equivalently, the identity of the second image is transferred to the first image. To do this, we infer $\mathbf{h}$ and $\mathbf{m}$ for both images. Using the pose or expression units $\mathbf{m}$ from the first and identity units $\mathbf{h}$ from the second image, we compute the expect input $\mathbf{v}|\mathbf{h}, \mathbf{m}$. We visualize the samples in Figure 3.5(a) and 3.5(b).

3.4.2 Discriminative performance

To measure the usefulness of our features and the degree of disentangling, we apply our model to emotion recognition, pose estimation and face verification on TFD and Multi-

Table 3.1: Control experiments of our method on Multi-PIE, with naive generative training, clamping identity-related units (ID), using labels for pose-related units (Pose) and using the manifold-based regularization on both groups of units.

Model	Pose units pose est.	Pose units verification	ID units pose est.	ID units verification
Naive	96.60 ± 0.23	0.583 ± 0.004	95.79 ± 0.37	0.640 ± 0.005
Labels (Pose)	$\mathbf{98.07} \pm 0.12$	0.485 ± 0.005	86.55 ± 0.23	0.656 ± 0.004
Clamp (ID)	97.18 ± 0.15	0.509 ± 0.005	57.37 ± 0.45	0.922 ± 0.003
Labels (Pose) + Clamp (ID)	97.68 ± 0.17	0.504 ± 0.006	49.08 ± 0.50	0.934 ± 0.002
Manifold (both)	$\mathbf{98.20} \pm 0.12$	0.469 ± 0.005	8.68 ± 0.38	$\mathbf{0.975} \pm 0.002$

Table 3.2: Control experiments of our method on TFD, with naive generative training, clamping identity-related units (ID), using labels for expression-related units (Expr) and using the manifold-based regularization on both groups of units.

Model	Expr. units emotion rec.	Expr. units verification	ID units emotion rec.	ID units verification
Naive	79.50 ± 2.17	0.835 ± 0.018	79.81 ± 1.94	0.878 ± 0.012
Labels (Expr)	83.55 ± 1.63	0.829 ± 0.021	78.26 ± 2.58	0.917 ± 0.006
Clamp (ID)	81.30 ± 1.47	0.803 ± 0.013	59.47 ± 2.17	$\mathbf{0.978} \pm 0.025$
Labels (Expr) + Clamp (ID)	82.97 ± 1.85	0.799 ± 0.013	59.55 ± 3.04	$\mathbf{0.978} \pm 0.024$
Manifold (both)	$\mathbf{85.43} \pm 2.54$	0.513 ± 0.011	43.27 ± 7.45	0.951 ± 0.025

PIE. For experiments on TFD, we built a 2-layer model whose first layer is constructed with convolutional features extracted using the filters trained with OMP-1 followed by 4×4 max pooling [20]. We used the same model in Section 3.4.1 for the tasks on Multi-PIE.

We did control experiments of our proposed training strategies and provide summary results in Table 3.1 and 3.2. We report the performance of pose estimation and face verification for Multi-PIE, and emotion recognition and face verification for TFD. For pose estimation and emotion recognition, we trained a linear SVM and reported the percent accuracy. For face verification, we used cosine similarity for the image pair and report the AU-ROC. Both numbers are averaged over 5 folds.

We observed that the naive training without any regularization gets mediocre performance on both datasets. By adding pose or emotion labels, we see improvement in pose

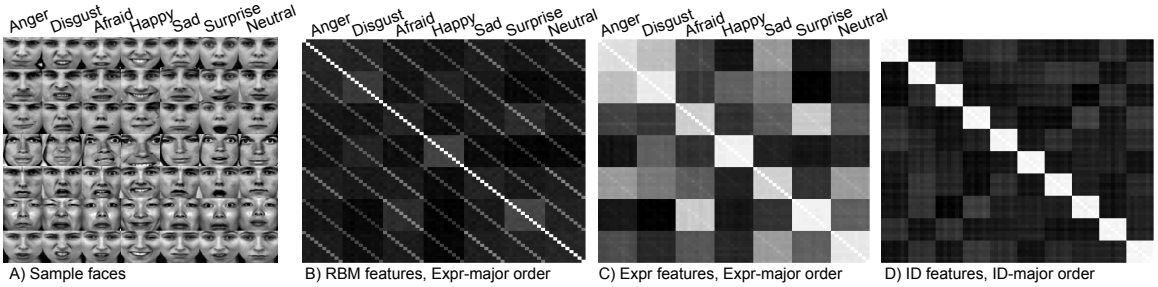


Figure 3.6: A) A sample of several identities with each of the 7 emotions in TFD. We drew 100 such samples and averaged the results. B) Similarity matrix using RBM features. C) Using our expression-related features (Expr). D) Using our identity-related features (ID).

Table 3.3: Performance comparison of discriminative tasks on Multi-PIE. RBM stands for the second layer RBM features trained on the first layer RBM features.

MODEL	POSE ESTIMATION	FACE VERIFICATION
RBM	93.06 ± 0.33	0.615 ± 0.002
DISBM	$\mathbf{98.20} \pm 0.12$	$\mathbf{0.975} \pm 0.002$

estimation and emotion recognition as expected, but also slightly better verification performance on both datasets. In addition, we observed a modest degree of disentangling (e.g., ID units performed poorly on pose estimation). The clamping method for ID units between corresponding image pairs showed substantially improved face verification results on both datasets. Combined with labels connected to the pose or expression units, the pose estimation and emotion recognition performance were improved. Finally, the best performance is achieved using manifold-based regularization, showing not only better absolute performance but also better disentangling. For example, while the expression units showed the best results for emotion recognition, the ID units were least informative for emotion recognition and vice versa. This suggests that good disentangling is not only useful from a generative perspective but also helpful for learning discriminative features.

We provide a performance comparison to the baseline and other existing models. Table 3.3 shows a comparison to a standard (second layer) RBM baseline using the same first layer features as our disBM on Multi-PIE. We note that face verification on Multi-PIE is challenging due to the pose variations. However, our disentangled ID features surpass this

Table 3.4: Performance comparison of discriminative tasks on TFD. RBM stands for the second layer RBM features trained on the first layer OMP features.

MODEL	EMOTION REC.	FACE VERIFICATION
RBM	81.84 ± 0.86	0.889 ± 0.012
DISBM	$\mathbf{85.43} \pm 2.54$	$\mathbf{0.951} \pm 0.025$
Rifai et al. [123]	$\mathbf{85.00} \pm 0.47$	–
Ranzato et al. [116]	82.4	–
Susskind et al. [144]	–	**0.951**

baseline by a wide margin. In Table 3.4, we compare the performance of our model to other works on TFD. The disBM features trained with manifold objectives achieved state-of-the-art performance in emotion recognition and face verification on TFD.

To highlight the benefit of higher-order interactions, we performed additional control experiments on Multi-PIE with more factors of variation, including pose, illumination and jittering. We evaluated the performance of the disBM and its 2-way counterpart by setting the higher-order weights to 0, where both are trained using the manifold objective. The summary results in face verification and pose estimation are given in Table 3.5. When the data have few modes of variation, we found that the 2-way model still shows good pose estimation and face verification performance. However, the higher-order interactions provide increasing benefit with the growth in modes of variation, i.e., joint configurations of pose, lighting or other factors. Such a benefit can be verified in the pose transfer task as well. In Figure 3.7, we visualize the pose transfer results of 2-way and (2+3)-way disBM models. The (2+3)-way model (fourth column) predicts the pose with given identity well, whereas the 2-way model (third column) produces significantly worse qualitative results, showing overlapping face artifacts and ambiguous identity.

3.4.3 Invariance and sensitivity analysis

We computed a similarity matrix by randomly selecting 10 identities (that had at least 7 distinct expressions) at a time, computing the cosine similarity for all pairs across all

MODEL	2-WAY	(2+3)-WAY
POSE	0.971 ± 0.002	$\mathbf{0.975} \pm 0.002$
POSE + JITTER	0.871 ± 0.005	$\mathbf{0.903} \pm 0.006$
POSE + JITTER + ILLUMINATION	0.773 ± 0.004	$\mathbf{0.822} \pm 0.003$
POSE	97.73 ± 0.20	$\mathbf{98.20} \pm 0.12$
POSE + JITTER	82.58 ± 0.53	$\mathbf{83.68} \pm 0.69$
POSE + JITTER + ILLUMINATION	76.42 ± 1.09	$\mathbf{80.34} \pm 1.29$

Table 3.5: Comparison of face verification AUC (top) and pose estimation % accuracy (bottom) between 2-way and (2+3)-way disBM with increasingly many factors of variation (e.g., pose, jittering, illumination) on Multi-PIE.

Figure 3.7: Comparison of pose transfer results between 2-way and (2+3)-way disBM models on Multi-PIE. The task is pose transfer from faces in the second column onto the face in the first column.

IDs and expressions. Then we averaged this feature similarity matrix over 100 trials. In Figure 3.6, we show average cosine similarity of several features across expression and identity variation. In ID-major order, the similarity matrix consists of 7×7-sized blocks; for each pair of IDs we compute similarity for all pairs among 7 different emotions. In Expr-major order, the similarity matrix consists of 10×10-sized blocks; for each pair of emotions we compute similarity for all pairs among 10 different IDs.

The ID features show a clear block-diagonal structure in ID-major order, indicating that they maintain similarity across changes in emotion but not across identity. In Expr-major order, our Expr features show similar structure, although there are apparent off-diagonal similarities for (anger, disgust) and (afraid, surprised) emotion labels. This makes sense because those emotions often have similar facial expressions. For the RBM features we see

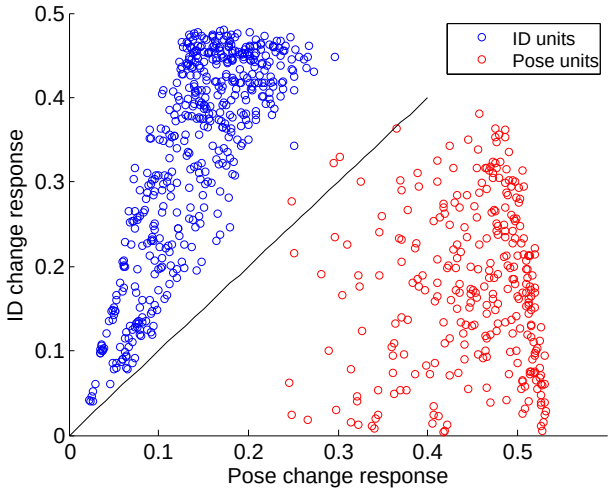


Figure 3.8: A scatter plot of average sensitivity of ID units (blue) and pose units (red) on Multi-PIE. The black line through the origin has slope 1, and approximately separates ID unit responses from pose unit responses.

only a faint block diagonal and a strong single band diagonal corresponding to same-ID, same-expression pairs.

To see whether our disBM features can be both invariant and sensitive to changes in different factors of variation, we generated test set image pairs (1) with the same identity, but different pose, and (2) with different identity, but the same pose. Then we measured the average absolute difference in activation within pose units and within ID units. For every unit k and image pair $(\mathbf{v}^{(1)}, \mathbf{v}^{(2)})$, we compute the average $|h_k^{(1)} - h_k^{(2)}|$. Figure 3.8 shows that ID units are more sensitive to change in ID than to pose, and pose units are likewise more sensitive to pose change than ID change.

3.5 Discussion

We introduced a new method of learning deep representations via disentangling factors of variation. We evaluated several strategies for training higher-order Boltzmann machines to model interacting manifolds such as pose, expression and identity in face images. We demonstrated that our model learns disentangled representations, achieving strong performance in generative and discriminative tasks.

CHAPTER IV

Deep convolutional encoder-decoder models for disentangling and visual analogy-making

4.1 Introduction

Humans are good at considering "what-if?" questions about objects in their environment. What if this chair were rotated a few degrees clockwise? What if I dyed my hair blue? We can easily imagine roughly how objects would look according to various hypothetical questions. However, current generative models of images struggle to perform this kind of task without encoding significant prior knowledge about the environment and restricting the allowed transformations.

Often, these visual hypothetical questions can be effectively answered by analogical reasoning [1]. Having observed many similar objects rotating, one could learn to mentally rotate new objects. Having observed objects with different colors (or textures), one could learn to mentally re-color (or re-texture) new objects.

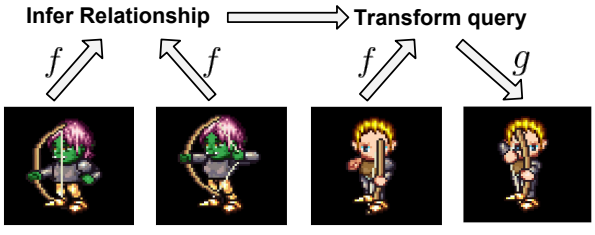


Figure 4.1: Visual analogy making concept. We learn an encoder function f mapping images into a space in which analogies can be performed, and a decoder g mapping back to the image space.

[1] See [11] for a deeper philosophical discussion of analogical reasoning.

Solving the analogy problem requires the ability to identify relationships among images and transform query images accordingly. In this paper, we propose to solve the problem by directly training on visual analogy completion; that is, to generate the transformed image output. Note that we do not make any claim about how humans solve the problem, only that in many cases thinking by analogy is enough to solve it, without exhaustively encoding first principles into a complex model.

We denote a valid analogy as a 4-tuple A : B :: C : D, often spoken as "A is to B as C is to D". Given such an analogy, there are several questions one might ask:

- A ? B :: C ? D - What is the common relationship?
- A : B ? C : D - Are A and B related in the same way that C and D are related?
- A : B :: C : ? - What is the result of applying the transformation A : B to C?

The first two questions can be viewed as discriminative tasks, and could be formulated as classification problems. The third question requires generating an appropriate image to make a valid analogy. Since a model with this capability would be of practical interest, we take this to be our focus.

Our proposed approach is to learn a deep encoder function $f : \mathbb{R}^D \rightarrow \mathbb{R}^K$ that maps images to an embedding space suitable for reasoning about analogies, and a deep decoder function $g : \mathbb{R}^k \rightarrow \mathbb{R}^D$ that maps from the embedding back to the image space. (See Figure 4.1.) Our encoder function is inspired by word2vec [98], GloVe [114] and other embedding methods that map inputs to a space supporting analogies by vector addition. In those models, analogies could be performed via

$$d = \underset{w \in \mathcal{V}}{\operatorname{argmax}} \cos(f(w), f(b) - f(a) + f(c))$$

where $\mathcal{V}$ is the vocabulary and (a, b, c, d) form an analogy tuple such that $a : b :: c : d$. Other variations such as a multiplicative version [87] on this inference have been proposed. The vector $f(b) - f(a)$ represents the transformation, which is applied to a query c by

vector addition in the embedding space. In the case of images, we can modify this naturally by replacing the cosine similarity and $argmax$ over the vocabulary with application of a decoder function mapping from the embedding back to the image space.

Clearly, this simple vector addition will not accurately model transformations for low-level representations such as raw pixels, and so in this work we seek to learn a high-level representation. In our experiments, we parametrize the encoder f and decoder g as deep convolutional neural networks, but in principle other methods could be used to model f and g. In addition to vector addition, we also propose more powerful methods of applying the inferred transformations to new images, such as higher-order multiplicative interactions and multi-layer additive interactions.

We first demonstrate visual analogy making on a 2D shapes benchmark, with variation in shape, color, rotation, scaling and position, and evaluate the performance on analogy completion. Second, we generate a dataset of animated 2D video game character sprites using graphics assets from the Liberated Pixel Cup [1]. We demonstrate the capability of our model to transfer animations onto novel characters from a single frame, and to perform analogies that traverse the manifold induced by an animation. Third, we apply our model to the task of analogy making on 3D car models, and show that our model can perform 3D pose transfer and rotation by analogy.

4.2 Related Work

Automated systems for analogy-making have a long history. *Evans* [39] ANALOGY program was able to predict completions of visual analogies consisting of 2D shape line drawings (see also [100]), further extended to drawings of kinematics devices in [166]. *Hertzmann et al.* [57] developed a method for applying new textures to images by analogy. This problem is of practical interest, e.g., for stylizing animations [13]. Our model can also synthesize new images by analogy to examples, but we study global transformations rather than only changing the texture of the image.

Dollár et al. [31] developed Locally-Smooth Manifold Learning to traverse image manifolds. We share a similar motivation when analogical reasoning requires walking along a manifold (e.g. pose analogies), but our model leverages a deep encoder and decoder trainable by backprop.

Memisevic and Hinton [94] proposed the Factored Gated Boltzmann Machine for learning to represent transformations between pairs of images. This and related models [144, 30, 96] use 3-way tensors or their factorization to infer translations, rotations and other transformations from a pair of images, and apply the same transformation to a new image. In this work, we share a similar goal, but we directly train a deep predictive model for the analogy task *without* requiring 3-way multiplicative connections, with the intent to scale to bigger images and learn more subtle relationships involving articulated pose, multiple attributes and out-of-plane rotation.

Our work is related to several previous works on disentangling factors of variation, for which a common application is analogy-making. As an early example, bilinear models [151] were proposed to separate style and content factors of variation in face images and speech signals. *Tang et al.* [149] developed the tensor analyzer which uses a factor loading tensor to model the interaction among latent factor groups, and was applied to face modeling. Several variants of higher-order Boltzmann machine were developed to attack the disentangling problem, featuring multiple groups of hidden units each corresponding to a single factor [118, 28]. Disentangling was also considered in the discriminative case in the Contractive Discriminative Analysis model [123]. Our work differs from these in that we train a deep end-to-end network for generating images by analogy.

Recently several works have proposed methods of generating high-quality images using deep networks. *Dosovitskiy et al.* [36] used a CNN to generate chair images with controllable variation in appearance, shape and 3D pose. Contemporary to our work, *Kulkarni et al.* [79] proposed the Deep Convolutional Inverse Graphics Network, which is a form of variational autoencoder (VAE) [73] in which the encoder disentangles factors of variation.

Other works have considered a semi-supervised extension of the VAE [74] incorporating class labels associated to a subset of the training images, which can control the label units to perform some visual analogies. *Cohen and Welling* [22] developed a generative model of commutative Lie groups (e.g. image rotation, translation) that produced invariant and disentangled representations. In [21], this work is extended to model the non-commutative 3D rotation group SO(3). *Zhu et al.* [176] developed the multi-view perceptron for modeling face identity and viewpoint, and generated high quality faces subject to view changes. *Cheung et al.* [18] also use a convolutional encoder-decoder model, and develop a regularizer to disentangle latent factors of variation from a discriminative target.

Analogies have been well-studied in the NLP community; *Turney* [155] used analogies from SAT tests to evaluate the performance of text analogy detection methods. In the visual domain, *Hwang et al.* [65] developed an analogy-preserving visual-semantic embedding model that could both detect analogies and as a regularizer improve visual recognition performance. Our work is related to these, but we focus mainly on generating images to complete analogies rather than detecting analogies.

4.3 Deep encoder-decoder models for visual analogy-making

Suppose that $\mathcal{A}$ is the set of valid analogy tuples in the training set. For example, $(a, b, c, d) \in \mathcal{A}$ implies the statement "a is to b as c is to d". Let the input image space for images a, b, c, d be $\mathbb{R}^D$, and the embedding space be $\mathbb{R}^K$ (typically $K < D$). Denote the encoder as $f : \mathbb{R}^D \to \mathbb{R}^K$ and the decoder as $g : \mathbb{R}^K \to \mathbb{R}^D$. Figure 4.2 illustrates our architectures for visual analogy making.

4.3.1 Making analogies by vector addition

Neural word representations (e.g., [98, 114]) have been shown to be capable of analogy-making by addition and subtraction of word embeddings. Analogy making capability appears to be an emergent property of these embeddings, but for images we propose to directly train on the objective of analogy completion. Concretely, we propose the following

objective for vector-addition-based analogies:

$$\mathcal{L}_{add} = \sum_{a,b,c,d \in \mathcal{A}} ||d - g(f(b) - f(a) + f(c))||_2^2 \tag{4.1}$$

This has the advantage of being simple to implement and train. With a modest number of labeled relations, a large number of training analogies can be mined.

4.3.2 Making analogy transformations dependent on the query context

In some cases, a purely additive model of applying transformations may not be ideal. For example, in the case of rotation, the manifold of a rotated object is *circular*, and after enough rotation has been applied, one returns to the original point. In the vector-addition model, we can add the same rotation vector $f(b) - f(a)$ multiple times to a query $f(c)$, but we will never return to the original point (except when $f(b) = f(a)$). The decoder g could (theorectically) solve this problem by learning to perform a "modulus" operation, but this would make the training significantly more difficult. Instead, we propose to parametrize the transformation increment to $f(c)$ as a function of both $f(b) - f(a)$ and $f(c)$ itself. In this way, analogies can be applied in a context-dependent way.

We present two variants of our training objective to solve this problem. The first, which we will call $\mathcal{L}_{mul}$, uses multiplicative interactions between $f(b) - f(a)$ and $f(c)$ to generate the increment. The second, which we call $\mathcal{L}_{deep}$, uses multiple fully connected layers to form a multi-layer perceptron (MLP) *without* using multiplicative interactions:

$$\mathcal{L}_{mul} = \sum_{a,b,c,d \in \mathcal{A}} ||d - g(f(c) + W \times_1 [f(b) - f(a)] \times_2 f(c))||_2^2 \tag{4.2}$$

$$\mathcal{L}_{deep} = \sum_{a,b,c,d \in \mathcal{A}} ||d - g(f(c) + h([f(b) - f(a); f(c)]))||_2^2. \tag{4.3}$$

For $\mathcal{L}_{mul}$, $W \in \mathbb{R}^{K \times K \times K}$ is a 3-way tensor. [2] In practice, to reduce the number of

[2] For a tensor $W \in \mathbb{R}^{K \times K \times K}$ and vectors $v, w \in \mathbb{R}^K$, we define the tensor multiplication $W \times_1 v \times_2 w \in \mathbb{R}^K$ as $(W \times_1 v \times_2 w)_l = \sum_{i=1}^K \sum_{j=1}^K W_{ijl} v_i w_j, \forall l \in \{1, ..., K\}$.

weights we used a factorized tensor parametrized as $W_{ijl} = \sum_f W_{if}^{(1)} W_{jf}^{(2)} W_{lf}^{(3)}$. Multiplicative interactions were similarly used in bilinear models [151], disentangling Boltzmann Machines [118] and Tensor Analyzers [149]. Note that our multiplicative interaction in $\mathcal{L}_{mul}$ is different from [94] in that we use the difference between two encoding vectors (i.e., $f(b) - f(a)$) to infer about the transformation (or relation), rather than using a higher-order interaction (e.g., tensor product) for this inference.

Algorithm 1 Manifold traversal by analogy, with transformation T (Eq. 4.5).

Given images a, b, c, and N (# steps)
$z \leftarrow f(c)$ ▷ Init from query position on the manifold
for $i = 1$ to N **do**
  $z \leftarrow z + T(f(a), f(b), z)$ ▷ Add increment along manifold
  $x_i \leftarrow g(z)$ ▷ Decode from manifold to image space
end for
return generated images x_i $(i = 1, ..., N)$

For $\mathcal{L}_{deep}$, $h : \mathbb{R}^{2K} \rightarrow \mathbb{R}^K$ is an MLP (deep network without 3-way multiplicative interactions) and $[f(b) - f(a); f(c)]$ denotes concatenation of the transformation vector with the query embedding.

Optimizing the above objectives teaches the model to predict analogy completions in image space, but in order to traverse image manifolds (e.g. for repeated analogies) as in Algorithm 1, we also want accurate analogy completions in the embedding space. To encourage this property, we introduce a regularizer to make the predicted transformation increment $T(f(a), f(b), f(c))$ match the difference of encoder embeddings $f(d) - f(c)$:

$$R = \sum_{a,b,c,d \in \mathcal{A}} ||f(d) - f(c) - T(f(a), f(b), f(c))||_2^2 \text{, where} \tag{4.4}$$

$$T(x, y, z) = \begin{cases} y - x & \text{when using } \mathcal{L}_{add} \\ W \times_1 [y - x] \times_2 z & \text{when using } \mathcal{L}_{mul} \\ MLP([y - x; z]) & \text{when using } \mathcal{L}_{deep} \end{cases} \tag{4.5}$$

The overall training objective is a weighted combination of analogy prediction and the

above regularizer, e.g. $\mathcal{L}_{deep} + \alpha R$. We set $\alpha = 0.01$ by validation on the shapes data and found it worked well for all models on sprites and 3D cars as well. All parameters were trained with backpropagation using stochastic gradient descent (SGD).

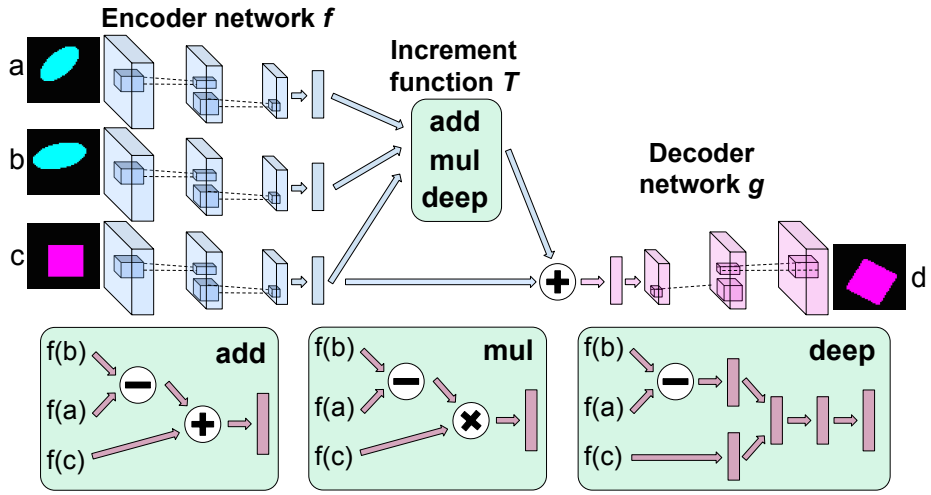


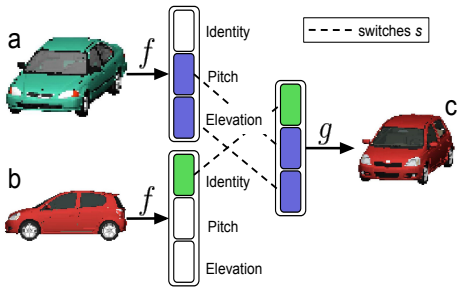
Figure 4.2: Illustration of the network structure for analogy making. The top portion shows the encoder, transformation module, and decoder. The bottom portion illustrates the transformations used for $\mathcal{L}_{add}$, $\mathcal{L}_{mul}$ and $\mathcal{L}_{deep}$. The $\bigotimes$ icon in $\mathcal{L}_{mul}$ indicates a tensor product. We share weights with all three encoder networks shown on the top left.

4.3.3 Analogy-making with a disentangled feature representation

Visual analogies change some aspects of a query image, and leave others unchanged; for example, changing the viewpoint but preserving the shape and texture of an object. To exploit this fact, we incorporate disentangling into our analogy prediction model. A disentangled representation is simply a concatenation of coordinates along each underlying factor of variation. If one can reliably infer these disentangled coordinates, a subset of analogies can be solved simply by swapping sets of coordinates among a reference and query embedding, and projecting back into the image space. However, in general, disentangling alone cannot solve analogies that require traversing the manifold structure of a given factor, and by itself does not capture image relationships.

In this section we show how to incorporate disentangled features into our model. The disentangling component makes each group of embedding features encode its respective

factor of variation and be invariant to the others. The analogy component enables the model to traverse the manifold of a given factor or subset of factors.



Algorithm 2 Disentangling training update. The switches s determine which units from $f(a)$ and $f(b)$ are used to reconstruct image c.

Given input images a, b and target c
Given switches $s \in \{0, 1\}^K$
$z \leftarrow s \cdot f(a) + (1 - s) \cdot f(b)$
$\Delta\theta \propto \partial/\partial\theta \left(||g(z) - c||_2^2\right)$

Figure 4.3: The encoder f learns a disentangled representation, in this case for pitch, elevation and identity of 3D car models. In the example above, switches s would be a block $[\mathbf{0}; \mathbf{1}; \mathbf{1}]$ vector.

For learning a disentangled representation, we require three-image tuples: a pair from which to extract hidden units, and a third to act as a target for prediction. As shown in Figure 4.3, We use a vector of switch units s that decides which elements from $f(a)$ and which from $f(b)$ will be used to form the hidden representation $z \in \mathbb{R}^K$. Typically s will have a block structure according to the groups of units associated to each factor of variation. Once z has been extracted, it is projected back into the image space via the decoder $g(z)$.

The key to learning disentangled features is that images a, b, c should be distinct, so that there is no path from any image to itself. This way, the reconstruction target forces the network to separate the visual *concepts* shared by (a, c) and (b, c), respectively, rather than learning the identity mapping. Concretely, the disentangling objective can be written as:

$$\mathcal{L}_{dis} = \sum_{a,b,c,s \in \mathcal{D}} ||c - g(s \cdot f(a) + (1 - s) \cdot f(b))||_2^2 \tag{4.6}$$

Note that unlike analogy training, disentangling only requires a 3-tuple of images a, b, c along with a switch unit vector s. Intuitively, s describes the sense in which a, b and c are related. Algorithm 2 describes the update used to learn a disentangled representation.

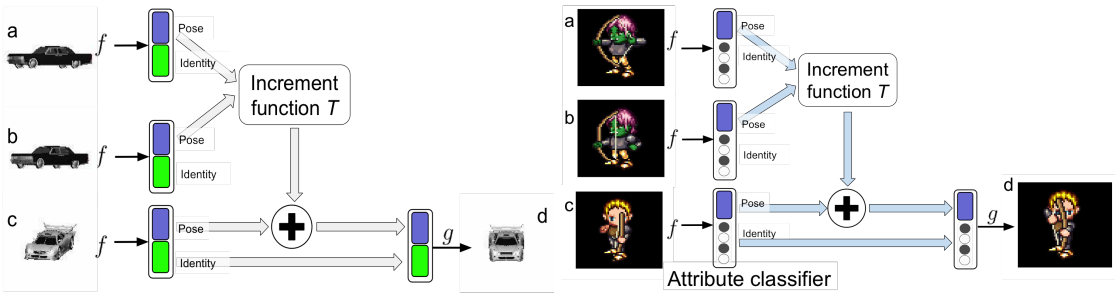


Figure 4.4: Analogy-making with disentangled features. Left: Analogy transformation operates on the pose only (in blue), separated from the identity (in green). Right: Identity units can also take the form of an attribute vector, e.g. for 2D sprite characteristics.

We can also train a single model to perform analogy-making with a disentangled feature representation. Intuitively, the analogy transformation may involve some factors of variation but not others, e.g. a 3D rotation changes the pose but not the identity of an object. Figure 4.4 illustrates two network architectures combining analogy-making and a disentangled feature representation.

4.4 Analogy-making Experiments

We evaluated our methods using three datasets. The first is a set of 2D colored shapes, which is a simple yet nontrivial benchmark for visual analogies. The second is a set of 2D sprites from the open-source video game project called Liberated Pixel Cup [1], which we chose in order to get controlled variation in a large number of character attributes and animations. The third is a set of 3D car model renderings [42], which allowed us to train a model to perform out-of-plane rotation. We used Caffe [67] to train our encoder and decoder networks, with a custom Matlab wrapper implementing our analogy sampling and training objectives. Many additional qualitative results of images generated by our model are presented in the appendix.

4.4.1 Transforming shapes: comparison of analogy models

The shapes dataset was used to benchmark performance on rotation, scaling and translation analogies. We generated 48×48 images scaled to $[0, 1]$ with four shapes, eight

colors, four scales, five row and column positions, and 24 rotation angles.

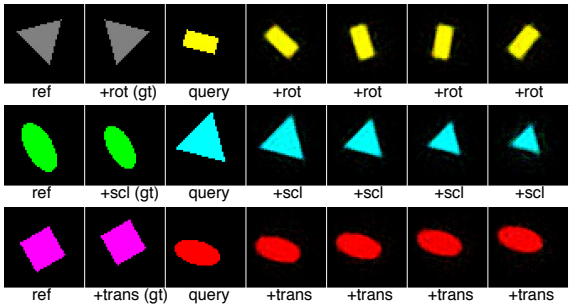


Figure 4.5: Analogy predictions made by $\mathcal{L}_{deep}$ for rotation, scaling and translation, respectively by row. $\mathcal{L}_{add}$ and $\mathcal{L}_{mul}$ perform as well for scaling and transformation, but fail for rotation. The model is able to extrapolate along scale to smaller shapes than were in the training data.

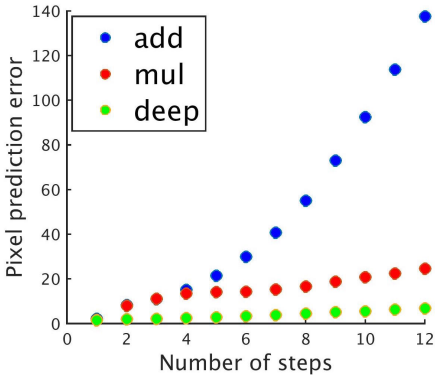


Figure 4.6: Mean-squared prediction error on repeated application of rotation analogies.

We compare the performance of our models trained with $\mathcal{L}_{add}$, $\mathcal{L}_{mul}$ and $\mathcal{L}_{deep}$ objectives, respectively. We did not perform disentangling training in this experiment. The encoder f consisted of 4096-1024-512-dimensional fully connected layers, with rectified linear nonlinearities (relu) for intermediate layers. The final embedding layer did not use any nonlinearity. The decoder g architecture mirrors the encoder, but did not share weights. We trained for 200K steps with mini-batch size 25 (i.e. 25 analogy 4-tuples per mini-batch). We used SGD with momentum 0.9, base learning rate 0.001 and decayed the learning rate by factor 0.1 every 100K steps.

Model	**Rotation steps**				**Scaling steps**				**Translation steps**			
	1	2	3	4	1	2	3	4	1	2	3	4
$\mathcal{L}_{add}$	8.39	11.0	15.1	21.5	5.57	6.09	7.22	14.6	5.44	5.66	6.25	7.45
$\mathcal{L}_{mul}$	8.04	11.2	13.5	14.2	4.36	4.70	5.78	14.8	4.24	4.45	5.24	6.90
$\mathcal{L}_{deep}$	**1.98**	**2.19**	**2.45**	**2.87**	**3.97**	**3.94**	**4.37**	**11.9**	**3.84**	**3.81**	**3.96**	**4.61**

Table 4.1: Comparison of squared pixel error of $\mathcal{L}_{add}$, $\mathcal{L}_{mul}$ and $\mathcal{L}_{deep}$ on shape analogies.

Figure 4.5 shows repeated predictions from $\mathcal{L}_{deep}$ on rotation, scaling and translation test set analogies, showing that our model has learned to traverse these manifolds. Table 4.1 shows that $\mathcal{L}_{add}$ and $\mathcal{L}_{mul}$ perform similarly for scaling and translation, but only $\mathcal{L}_{deep}$ can

perform accurate rotation analogies. Further extrapolation results with repeated rotations are shown in Figure 4.6. Though both $\mathcal{L}_{mul}$ and $\mathcal{L}_{deep}$ both are *in principle* capable of learning the circular pose manifold, we suspect that $\mathcal{L}_{deep}$ has much better performance due to the difficulty of training multiplicative models such as $\mathcal{L}_{mul}$. A visualization of the learned rotation manifold is shown in Figure B.4.

4.4.2 Generating 2D video game sprites

Game developers often use what are known as "sprites" to portray characters and objects in 2D video games (more commonly on older systems, but still seen on phones and indie games). This entails significant human effort to draw each frame of each common animation for each character [3]. In this section we show how animations can be transferred to new characters by analogy.

Our dataset consists of 60×60 color images of sprites scaled to $[0, 1]$, with 7 attributes: body type, sex, hair type, armor type, arm type, greaves type, and weapon type, with 672 total unique characters. For each character, there are 5 animations each from 4 viewpoints: spellcast, thrust, walk, slash and shoot. Each animation has between 6 and 13 frames. We split the data by characters: 500 training, 72 validation and 100 for testing.

We evaluated the $\mathcal{L}_{add}$ and $\mathcal{L}_{deep}$ variants of our objective, with and without disentangled features. We also experimented with a disentangled feature version in which the identity units are taken to be the 22-dimensional character attribute vector, from which the pose is disentangled. In this case, the encoder for identity units acts as multiple softmax classifiers, one for each attribute, hence we refer to this objective in experiments as $\mathcal{L}_{dis+cls}$.

The encoder network consisted of two layers of 5×5 convolution with stride 2 and relu, followed by two fully-connected and relu layers, followed by a projection onto the 1024-dimensional embedding. The decoder mirrors the encoder. To increase the spatial dimension we use simple upsampling in which we copy each input cell value to the upper-

[3] In some cases the work may be decreased by projecting 3D models to 2D or by other heuristics, but in general the work scales with the number of animations and characters.

left corner of its corresponding 2×2 output.

For $\mathcal{L}_{dis}$, we used 512 units for identity and 512 for pose. For $\mathcal{L}_{dis+cls}$, we used 22 categorical units for identity, which is the attribute vector, and the remaining 490 for pose. During training for $\mathcal{L}_{dis+cls}$, we did not backpropagate reconstruction error through the identity units; we only used the attribute classification objective for those units. When $\mathcal{L}_{deep}$ is used, the internal layers of the transformation function T (see Figure 4.2) had dimension 300, and were each followed by relu. We trained the models using SGD with momentum 0.9 and learning rate 0.00001 decayed by factor 0.1 every 100k steps. Training was conducted for 200k steps with mini-batch size 25.

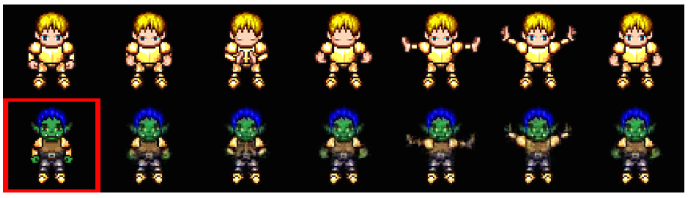


Figure 4.7: Transferring animations. The top row shows the reference, and the bottom row shows the transferred animation, where the first frame (in red) is the starting frame of a test set character.

Model	**spellcast**	**thrust**	**walk**	**slash**	**shoot**	**average**
$\mathcal{L}_{add}$	41.0	53.8	55.7	52.1	77.6	56.0
$\mathcal{L}_{dis}$	40.8	55.8	52.6	53.5	79.8	56.5
$\mathcal{L}_{dis+cls}$	13.3	24.6	17.2	18.9	40.8	23.0

Table 4.2: Mean-squared pixel error on test analogies, by animation.

Figure 4.7 demonstrates the task of animation transfer, with predictions from a model trained on $\mathcal{L}_{add}$. Table 4.2 provides a quantitative comparison of $\mathcal{L}_{add}$, $\mathcal{L}_{dis}$ and $\mathcal{L}_{dis+cls}$. We found that the disentangling and additive analogy models perform similarly, and that using attributes for disentangled identity features provides a further gain. We conjecture that $\mathcal{L}_{dis+cls}$ wins because changes in certain aspects of appearance, such as arm color, have a very small effect in pixel space yielding a weak signal for pixel prediction, but still provides a strong signal to an attribute classifier.

From a practical perspective, the ability to transfer poses accurately to unseen characters

could help decrease manual labor of drawing (at least of drawing the assets comprising each character in each animation frame). However, training this model required that each transferred animation already have hundreds of examples. Ideally, the model could be shown a small number of examples for a new animation, and transfer it to the existing character database. We call this setting "few-shot" analogy-making because only a small number of the analogy targets are provided.

	Num. few-shot examples			
Model	6	12	24	48
$\mathcal{L}_{add}$	42.8	42.7	42.3	41.0
$\mathcal{L}_{dis}$	19.3	18.9	17.4	16.3
$\mathcal{L}_{dis+cls}$	**15.0**	**12.0**	**11.3**	**10.4**

Table 4.3: Mean-squared pixel-prediction error for few-shot analogy transfer of the "spellcast" animation from 4 viewpoints.

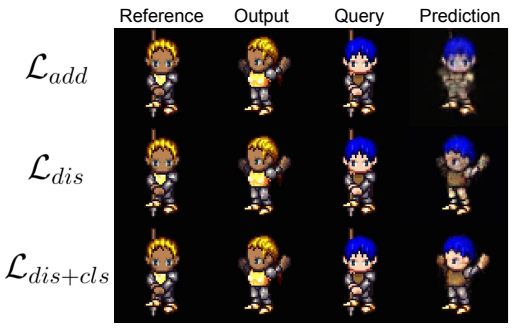


Figure 4.8: Few shot prediction with 48 training examples.

Table 4.3 provides a quantitative comparison and figure 4.8 provides a qualitative comparison of our proposed models in this task. We find that $\mathcal{L}_{dis+cls}$ provides the best performance by a wide margin. Unlike in Table 4.2, $\mathcal{L}_{dis}$ outperforms $\mathcal{L}_{add}$, suggesting that disentangling may allow new animations to be learned in a more data-efficient manner. However, $\mathcal{L}_{dis}$ has an advantage in that it can average the identity features of multiple views of a query character, which $\mathcal{L}_{add}$ cannot do.

The previous analogies only required us to combine disentangled features from two characters, e.g. the identity from one and the pose from another, and so disentangling was sufficient. However, our analogy method enables us to perform more challenging analogies by learning the manifold of character animations, defined by the sequence of frames in each animation. Adjacent frames are thus neighbors on the manifold and each animation sequence can be viewed as a fiber in this manifold.

We trained a model by forming analogy tuples across animations as depicted in Fig. 4.9,

Figure 4.9: A cartoon visualization of the "shoot" animation manifold for two different characters in different viewpoints. The model can learn the structure of the animation manifold by forming analogy tuples during training; example tuples are circled in red and blue above.

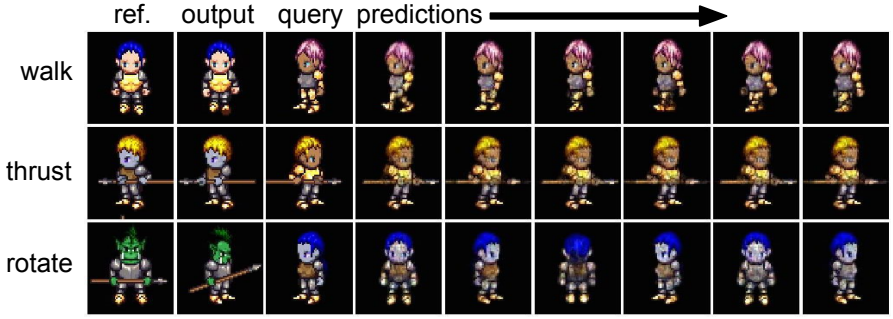


Figure 4.10: Extrapolating animations by analogy. The model is shown the reference and output pair, and repeatedly applies the inferred transformation to the query image, which advances the animation frame through time. Note that this kind of inference requires learning the manifold of animation poses, and cannot be done by simply combining and decoding disentangled features.

using disentangled identity and pose features. Pose transformations were modeled by deep additive interactions, and we used $\mathcal{L}_{dis+cls}$ to disentangle pose from identity units. Figure 4.10 shows the result of several analogies and their extrapolations, including character rotation for which we created animations.

4.4.3 3D car analogies

In this section we apply our model to analogy-making on 3D car renderings subject to changes in appearance and rotation angle. Unlike in the case of shapes, this requires the model to perform out-of-plane rotation, and the depicted objects are more complex.

Features	Pose AUC	ID AUC
Pose units	**95.6**	85.2
ID units	50.1	**98.5**
Combined	94.6	98.4

Table 4.4: Disentangling performance on 3D cars. Pose AUC refers to area under the ROC curve for same-or-different pose, and ID AUC for same-or-different car.

Figure 4.11: 3D car analogies. The column "GT" denotes ground truth.

We use the car CAD models from [42]. For each of the 199 car models, we generated 64×64 color renderings from 24 rotation angles each offset by 15 degrees. We split the models into 100 training, 49 validation and 50 testing. The same convolutional network architecture was used as in the sprites experiments, other than the difference that we used 512 units for identity and 128 units for pose.

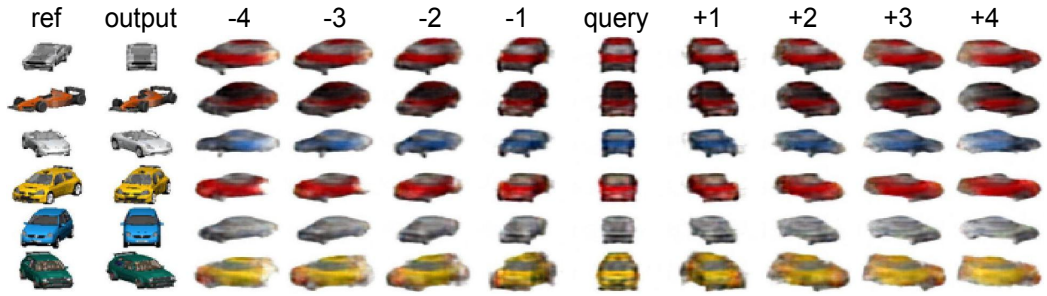


Figure 4.12: Repeated rotation in forward and reverse directions, starting from frontal.

Figure 4.11 shows the analogy completion predictions of our model trained on $\mathcal{L}_{dis}$, where prediction images are synthesized by combining pose units for the first car image and identity units for the second car image. Table 4.4 shows that the learned features are in fact disentangled, and discriminative for identity and pose matching despite not being discriminatively trained. Figure 4.12 shows repeated rotation analogies on test set cars using a model trained on $\mathcal{L}_{deep}$, demonstrating that our model can perform out-of-plane rotation. This type of extrapolation is difficult because the query image shows a different car from a different starting pose. We expect that a recurrent architecture can further improve the results, as shown in [167].

4.4.4 Evans ANALOGY program

In this section we discuss the relation of this chapter to Thomas Evan's pioneering work on visual analogy making with 2D shape line drawings [39].

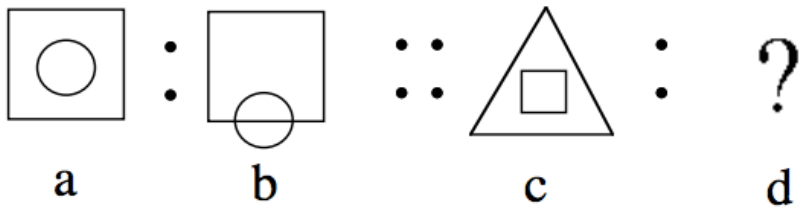


Figure 4.13: Concept figure from Evans's 1964 paper. The task is to correctly predict which of shapes 1 - 6 makes the analogy A : B :: C : D true.

Figure 4.13 shows the concept figure from Evans' paper. In some ways, ANALOGY demonstrated more advanced capabilities than ours; in particular it is able to reason about the relations between multiple objects in a single image and how they change from one image to another. In this chapter, we only considered a single object per image, although our objects can be more complex than simple line shapes.

The ANALOGY algorithm proceeds in two phases. First, each problem figure is decomposed into individual objects. For each object, a specified set of geometric properties is calculated, and the relations among each object are recorded. Second, ANALOGY searches over all rules that would map image A to image B. It then tries to find a rule that "generalizes" in the sense of also mapping image C onto one of the answer figures. The possible transformations considered are compositions of euclidian similarity transformations (rotation and uniform scale) with horizontal and vertical reflections.

Our system is distinct from ANALOGY in several advantageous ways: it learns rather than hand-codes the image features, it learns transformations directly from pixels inputs and outputs, and it produces the output *pixels* to complete an analogy rather than choosing among several pre-specified options. On the other hand, ANALOGY can reason about geometric analogies involving multiple objects with *no* training data; in a sense it has unsurpassable data efficiency.

Comparing with Evans' 1964 work begs the question: how can we combine the generality of connectionist end-to-end learning approaches with the power and efficiency of search-based symbolic approaches? Certainly we could train our model on Evans-style analogy problems involving multiple objects. Several recent works have modeled multi-object dynamics at the pixel level, e.g. for 2D bouncing balls [96] and predicting whether stacked blocks will fall [12, 86].

4.4.5 Limitations and ambiguous analogies

One limitation of our model is that it assumes a deterministic mapping of $(A, B, C) \rightarrow D$. In reality, there many be many possible D that form a true analogy. Therefore, a more general approach would be to model the *distribution* of analogy completions $P(D|A, B, C)$. A similar network architecture could be used, but perhaps the decoder could be trained as a Generative Adversarial Network [49] rather than with per-pixel targets.

What would happen if the present model were trained on ambiguous analogies? That is, if there existed training tuples (A, B, C, D) and (A, B, C, D'), with $D \neq D'$? In that case, the network would "hedge" its prediction to produce a blurred image. Indeed, this is the behavior observed in other convolutional encoder-decoder networks that model noisy natural image data such as the CelebFaces Attributes Dataset [89].

4.4.6 Conclusions

We studied the problem of visual analogy making using deep neural networks, and proposed several new models. Our experiments showed that our proposed models are very general and can learn to make analogies based on appearance, rotation, 3D pose, and various object attributes. We connected analogy making to the notion of disentangling factors of variation, and showed that analogy representations can overcome certain limitations of disentangled representations.

CHAPTER V

Learning to represent and execute programs

5.1 Introduction

Teaching machines to learn new programs, to rapidly compose new programs from existing programs, and to conditionally execute these programs automatically so as to solve a wide variety of tasks is one of the central challenges of AI. Programs appear in many guises in various AI problems; including motor behaviours, image transformations, reinforcement learning policies, classical algorithms, and symbolic relations.

In this work, we develop a compositional architecture that learns to represent and interpret programs. We refer to this architecture as the Neural Programmer-Interpreter (NPI). The core module is an LSTM-based sequence model that takes as input a learnable program embedding, program arguments passed on by the calling program, and a feature representation of the environment. The output of the core module is a key indicating what program to call next, arguments for the following program and a flag indicating whether the program should terminate. In addition to the recurrent core, the NPI architecture includes a learnable key-value memory of program embeddings. This program-memory is essential for learning and re-using programs in a continual manner. Figures 5.1 and 5.2 illustrate the NPI on two different tasks.

We show in our experiments that the NPI architecture can learn 21 programs, including addition, sorting, and trajectory planning from image pixels. Crucially, this can be

achieved using a single core model with the same parameters shared across all tasks. Different environments (for example images, text, and scratch-pads) may require specific perception modules or encoders to produce the features used by the shared core, as well as environment-specific actuators. Both perception modules and actuators can be learned from data when training the NPI architecture.

To train the NPI we use curriculum learning and supervision via example execution traces. Each program has example sequences of calls to the immediate subprograms conditioned on the input. By using neural networks to learn the subprograms from data, NPI can generalize on tasks involving rich perceptual inputs and uncertainty.

We may envision two approaches to provide supervision. In one, we provide a very large number of labeled examples, as in object recognition, speech and machine translation. In the other, the approached followed in this work, the aim is to provide far fewer labeled examples, but the labels contain richer information allowing the model to learn compositional structure. While unsupervised and reinforcement learning play important roles in perception and motor control, other cognitive abilities are possible thanks to rich supervision and curriculum learning; indeed the reason for sending children to school.

An advantage of our approach to model building and training is that the learned programs exhibit *strong generalization*. Specifically, when trained to sort sequences of up to twenty numbers in length, they can sort much longer sequences at test time. In contrast, the experiments will show that more standard sequence to sequence LSTMs only exhibit *weak generalization*, see Figure 5.8.

A trained NPI with fixed parameters and a learned library of programs, can act both as an interpreter and as a programmer. As an interpreter, it takes input in the form of a program embedding and input data and subsequently executes the program. As a programmer, it uses samples drawn from a new task to generate a new program embedding that can be added to its library of programs.

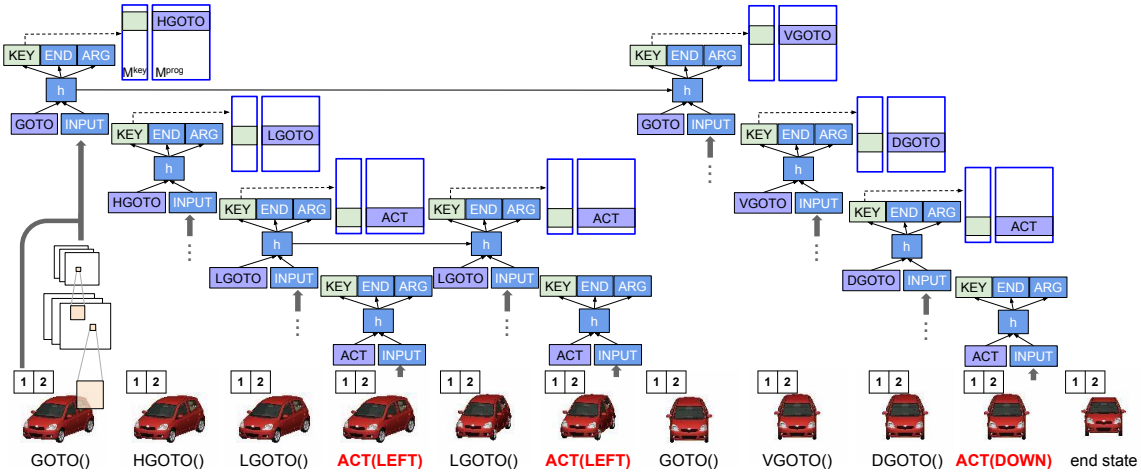


Figure 5.1: Example execution of canonicalizing 3D car models. The task is to move the camera such that a target angle and elevation are reached. There is a read-only scratch pad containing the target (angle 1, elevation 2 here). The image encoder is a convnet trained from scratch on pixels.

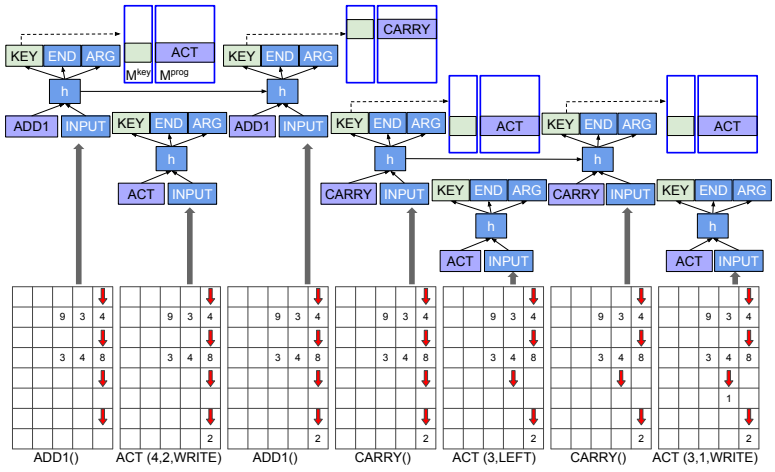


Figure 5.2: Single-digit addition. The task is to perform a single-digit add on the numbers at pointer locations in the first two rows. The carry (row 3) and output (row 4) should be updated to reflect the addition. At each time step, an observation of the environment (viewed from each pointer on a scratch pad) is encoded into a fixed-length vector.

5.2 Related work

Several ideas related to our approach have a long history. For example, the idea of using dynamically programmable networks in which the activations of one network become the weights (the program) of a second network was mentioned in the Sigma-Pi units section of the influential PDP paper [127]. This idea appeared in [145] in the context of learning higher order symbolic relations and in [35] as the key ingredient of an architecture for prefrontal cognitive control. *Schmidhuber* [132] proposed a related meta-learning

idea, whereby one learns the parameters of a slowly changing network, which in turn generates context dependent weight changes for a second rapidly changing network. These approaches have only been demonstrated in very limited settings. In cognitive science, several theories of brain areas controlling other brain parts so as to carry out multiple tasks have been proposed; see for example *Schneider and Chein* [133], *Anderson* [4] and *Donnarumma et al.* [34].

Related problems have been studied in the literature on hierarchical reinforcement learning (*e.g.*, *Dietterich* [29], *Andre and Russell* [5], *Sutton et al.* [147] and *Schaul et al.* [131]), imitation and apprenticeship learning (*e.g.*, *Kolter et al.* [76] and *Rothkopf and Ballard* [125]) and elicitation of options through human interaction [142]. These ideas have held great promise, but have not enjoyed significant impact. We believe the recurrent compositional neural representations proposed in this work could help these approaches in the future, and in particular in overcoming feature engineering.

Several recent advancements have extended recurrent networks to solve problems beyond simple sequence prediction. *Graves et al.* [50] developed a neural Turing machine capable of learning and executing simple programs such as repeat copying, simple priority sorting and associative recall. [156] developed Pointer Networks that generalize the notion of encoder attention in order to provide the decoder a variable-sized output space depending on the input sequence length. This model was shown to be effective for combinatorial optimization problems such as the traveling salesman and Delaunay triangulation. While our proposed model is trained on execution traces instead of input and output pairs, in exchange for this richer supervision we benefit from compositional program structure, improving data efficiency on several problems.

This work is also closely related to program induction. Most previous work on program induction, i.e. inducing a program given example input and output pairs, has used genetic programming [10] to evolve useful programs from candidate populations. *Mou et al.* [102] process program symbols to learn max-margin program embeddings with the help of parse

trees. *Zaremba and Sutskever* [169] trained LSTM models to read in the text of simple programs character-by-character and correctly predict the program output. *Joulin and Mikolov* [68] augmented a recurrent network with a pushdown stack, allowing for generalization to longer input sequences than seen during training for several algorithmic patterns.

Contemporary to this work, several papers have also studied program induction with variants of recurrent neural networks [170, 171, 69, 81, 103]. While we share a similar motivation, our approach is distinct in that we explicitly incorporate compositional structure into the network using a program memory, allowing the model to learn new programs by combining sub-programs.

This chapter can also be connected with decades-old work on inducing finite-state machines from examples with recurrent neural nets [19, 160, 47] and later genetic algorithms [154]. While these systems were demonstrated to be capable of extracting correct state machines to recognize simple languages from examples, they were not designed for induction of programs from execution traces. Also, they operated on sequences consisting of small alphabets of symbols, rather than perceptual inputs such as images as we consider in this work. Still, this earlier work on state machine induction has the remaining advantage of producing automata that *perfectly* generalize; in particular they do not make mistakes due to accumulation of small errors in a recurrent network. It may also be possible to extract discrete *programs* that similarly generalize perfectly; we leave this as future work.

5.3 Model

The NPI core is a long short-term memory (LSTM) network [61] that acts as a router between programs conditioned on the current state observation and previous hidden unit states. At each time step, the core module can select another program to invoke using content-based addressing. It emits the probability of ending the current program with a single binary unit. If this probability is over threshold (we used 0.5), control is returned

to the caller by popping the caller's LSTM hidden units and program embedding off of a program call stack and resuming execution in this context.

The NPI may also write arguments (ARG) passed by reference or value to the invoked sub-programs. For example, an argument could indicate a specific location in the input sequence (by reference), or it could specify a number to write down at a particular location in the sequence (by value). The subsequent state consists of these arguments and observations of the environment. The approach is illustrated in Figures 5.1 and 5.2.

It must be emphasized that there is a single inference core. That is, all the LSTM instantiations executing arbitrary programs share the same parameters. Different programs correspond to program embeddings, which are stored in a learnable persistent memory. The programs therefore have a more succinct representation than neural programs encoded as the full set of weights in a neural network [127, 50].

The output of an NPI, conditioned on an input state and a program to run, is a sequence of actions in a given environment. In this work, we consider several environments: a 1-D array with read-only pointers and a swap action, a 2-D scratch pad with read-write pointers, and a CAD renderer with controllable elevation and azimuth movements. Note that the sequence of actions for a program is not fixed, but dependent also on the input state.

5.3.1 Inference

Denote the environment observation at time t as $e_t \in \mathcal{E}$, and the current program arguments as $a_t \in \mathcal{A}$. The form of e_t can vary dramatically by environment; for example it could be a color image or an array of numbers. The program arguments a_t can also vary by environment, but in the experiments for this work we always used a 3-tuple of integers $(a_t(1), a_t(2), a_t(3))$. Given the environment and arguments at time t, a fixed-length state encoding $s_t \in \mathbb{R}^D$ is extracted by a domain-specific encoder $f_{enc} : \mathcal{E} \times \mathcal{A} \rightarrow \mathbb{R}^D$. In section 5.4 we provide examples of several encoders. Note that a single NPI network can have encoders for multiple environments, and encoders can also be shared across tasks.

We denote the current program embedding as $p_t \in \mathbb{R}^P$. The previous hidden unit

and cell states are $h_{t-1}^{(l)} \in \mathbb{R}^M$ and $c_{t-1}^{(l)} \in \mathbb{R}^M$, $l = 1, ..., L$ where L is the number of layers in the LSTM. The program and state vectors are then propagated forward through an LSTM mapping f_{lstm} as in [146]. How to fuse p_t and s_t within f_{lstm} is an implementation detail, but in this work we concatenate and feed through a 2-layer MLP with rectified linear (ReLU) hidden activation and linear decoder.

From the top LSTM hidden state h_t^L, several decoders generate the outputs. The probability of finishing the program and returning to the caller [1] is computed by $f_{end} : \mathbb{R}^M \rightarrow [0, 1]$. The lookup key embedding used for retrieving the next program from memory is computed by $f_{prog} : \mathbb{R}^M \rightarrow \mathbb{R}^K$. Note that $\mathbb{R}^K$ can be much smaller than $\mathbb{R}^P$ because the key only need act as the identifier of a program, while the program embedding must have enough capacity to conditionally generate a sequence of actions. The contents of the arguments to the next program to be called are generated by $f_{arg} : \mathbb{R}^M \rightarrow \mathcal{A}$. The feed-forward steps of program inference are summarized below:

$$s_t = f_{enc}(e_t, a_t) \tag{5.1}$$

$$h_t = f_{lstm}(s_t, p_t, h_{t-1}) \tag{5.2}$$

$$r_t = f_{end}(h_t),\ k_t = f_{prog}(h_t),\ a_{t+1} = f_{arg}(h_t) \tag{5.3}$$

where r_t, k_t and a_{t+1} correspond to the end-of-program probability, program key embedding, and output arguments at time t, respectively. These yield input arguments at time $t+1$. To simplify the notation, we have abstracted properties such as layers and cell memory in the sequence-to-sequence LSTM of equation (5.2); see [146] for details.

The NPI representation is equipped with key-value memory structures $M^{\texttt{key}} \in \mathbb{R}^{N \times K}$ and $M^{\texttt{prog}} \in \mathbb{R}^{N \times P}$ storing program keys and program embeddings, respectively, where N is the current number of programs in memory. We can add more programs by adding rows to memory.

[1] In our implementation, a program may first call a subprogram before itself finishing. The only exception is the ACT program that signals a low-level action to the environment, e.g. moving a pointer one step left or writing a value. By convention ACT does not call any further sub-programs.

During training, the next program identifier is provided to the model as ground-truth, so that its embedding can be retrieved from the corresponding row of $M^{\texttt{prog}}$. At test time, we compute the "program ID" by comparing the key embedding k_t to each row of $M^{\texttt{key}}$ storing all program keys. Then the program embedding is retrieved from $M^{\texttt{prog}}$ as follows:

$$i^* = \underset{i=1..N}{\arg\max}(M^{\texttt{key}}_{i,:})^T k_t \quad , \quad p_{t+1} = M^{\texttt{prog}}_{i^*,:} \tag{5.4}$$

The next environmental state e_{t+1} will be determined by the dynamics of the environment and can be affected by both the choice of program p_t and the contents of the output arguments a_t, *i.e.*

$$e_{t+1} \sim f_{env}(e_t, p_t, a_t) \tag{5.5}$$

The transition mapping f_{env} is domain-specific and will be discussed in Section 5.4. A description of the inference procedure is given in Algorithm 3.

Algorithm 3 Neural programming inference

Inputs: Environment observation e, program id i, arguments a, stop threshold α
function RUN(i, a)
  $h \leftarrow \mathbf{0}$, $r \leftarrow 0$, $p \leftarrow M^{\texttt{prog}}_{i,:}$ ▷ Init LSTM and return probability.
  while $r < \alpha$ **do**
    $s \leftarrow f_{enc}(e, a)$, $h \leftarrow f_{lstm}(s, p, h)$ ▷ Feed-forward NPI one step.
    $r \leftarrow f_{end}(h), k \leftarrow f_{prog}(h), a_2 \leftarrow f_{arg}(h)$
    $i_2 \leftarrow \underset{j=1..N}{\arg\max}(M^{\texttt{key}}_{j,:})^T k$ ▷ Decide the next program to run.
    if $i ==$ ACT **then** $e \leftarrow f_{env}(e, p, a)$ ▷ Update the environment based on ACT.
    else RUN(i_2, a_2) ▷ Run subprogram i_2 with arguments a_2

Each task has a set of actions that affect the environment. For example, in addition there are LEFT and RIGHT actions that move a specified pointer, and a WRITE action which writes a value at a specified location. These actions are encapsulated into a general-purpose ACT program shared across tasks, and the concrete action to be taken is indicated by the NPI-generated arguments a_t.

Note that the core LSTM module of our NPI representation is completely agnostic to the data modality used to produce the state encoding. As long as the same fixed-length

embedding is extracted, the same module can in practice route between programs related to sorting arrays just as easily as between programs related to rotating 3D objects. In the experimental sections, we provide details of the modality-specific deep neural networks that we use to produce these fixed-length state vectors.

5.3.2 Training

To train we use execution traces $\xi_t^{inp} : \{e_t, i_t, a_t\}$ and $\xi_t^{out} : \{i_{t+1}, a_{t+1}, r_t\}, t = 1, ...T$, where T is the sequence length. Program IDs i_t and i_{t+1} are row-indices in $M^{\texttt{key}}$ and $M^{\texttt{prog}}$ of the programs to run at time t and $t + 1$, respectively. We propose to directly maximize the probability of the correct execution trace output ξ^{out} conditioned on ξ^{inp}:

$$\theta^* = \arg\max_{\theta} \sum_{(\xi^{inp}, \xi^{out})} \log P(\xi^{out}|\xi^{inp}; \theta) \tag{5.6}$$

where θ are the parameters of our model. Since traces are variable in length depending on the input, we apply the chain rule to model the joint probability over $\xi_1^{out}, ..., \xi_T^{out}$:

$$\log P(\xi_{out}|\xi_{inp}; \theta) = \sum_{t=1}^{T} \log P(\xi_t^{out}|\xi_1^{inp}, ..., \xi_t^{inp}; \theta) \tag{5.7}$$

Note that for many problems the input history $\xi_1^{inp}, ..., \xi_t^{inp}$ is critical to deciding future actions because the environment observation at the current time-step e_t alone does not contain enough information. The hidden unit activations of the LSTM in NPI are capable of capturing these temporal dependencies. The single-step conditional probability in equation (5.7) can be factorized into three further conditional distributions, corresponding to predicting the next program, next arguments, and whether to halt execution:

$$\log P(\xi_t^{out}|\xi_1^{inp}, ..., \xi_t^{inp}) = \log P(i_{t+1}|h_t) + \log P(a_{t+1}|h_t) + \log P(r_t|h_t) \tag{5.8}$$

where h_t is the output of f_{lstm} at time t, carrying information from previous time steps. We train by gradient ascent on the likelihood in equation (5.7).

We used an adaptive curriculum in which training examples for each mini-batch are

fetched with frequency proportional to the model's current prediction error for the corresponding program. Specifically, we set the sampling frequency using a softmax over average prediction error across all programs, with configurable temperature. Every 1000 steps of training we re-estimated these prediction errors. Intuitively, this forces the model to focus on learning the program for which it currently performs worst in executing. We found that the adaptive curriculum immediately worked much better than our best-performing hand-designed curriculum, allowing a multi-task NPI to achieve comparable performance to single-task NPI on all tasks.

We also note that our program has a distinct memory advantage over basic LSTMs because all subprograms can be trained in parallel. For programs whose execution length grows *e.g.* quadratically with the input sequence length, an LSTM will by highly constrained by device memory to train on short sequences. By exploiting compositionality, an effective curriculum can often be developed with sublinear-length subprograms, enabling our NPI model to train on order of magnitude larger sequences than the LSTM.

5.4 Experiments

This section describes the environment and state encoder function for each task, and shows example outputs and prediction accuracy results. For all tasks, the core LSTM had two layers of size 256. We trained the NPI model and all program embeddings jointly using RMSprop with base learning rate 0.0001, batch size 1, and decayed the learning rate by a factor of 0.95 every 10,000 steps.

5.4.1 Task and environment descriptions

In this section we provide an overview of the tasks used to evaluate our model. Table C.1 in the appendix provides a full listing of all the programs and subprograms learned by our model.

Addition

The task in this environment is to read in the digits of two base-10 numbers and produce the digits of the answer. Our goal is to teach the model the standard (at least in the US) grade school algorithm of adding, in which one works from right to left applying single-digit add and carry operations.

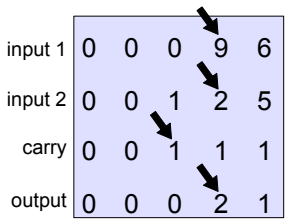


Figure 5.3: Example scratch pad and pointers used for computing "96 + 125 = 221". Carry step is being implemented.

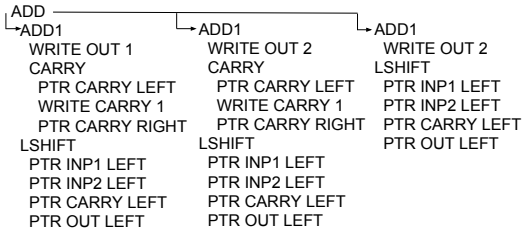


Figure 5.4: Actual trace of addition program generated by our model on the problem shown to the left. Note that we substituted the ACT calls in the trace with more human-readable steps.

In this environment, the network is endowed with a "scratch pad" with which to store intermediate computations; *e.g.* to record carries. There are four pointers; one for each of the two input numbers, one for the carry, and another to write the output. At each time step, a pointer can be moved left or right, or it can record a value to the pad. Figure 5.3 illustrates the environment of this model, and Figure 5.4 provides a real execution trace generated by our model conditioned on an example problem.

For the state encoder f_{enc}, the model is allowed a view of the scratch pad from the perspective of each of the four pointers. That is, the model sees the current values at pointer locations of the two inputs, the carry row and the output row, as 1-of-K encodings, where K is 10 because we are working in base 10. We also append the values of the input argument tuple a_t:

$$f_{enc}(Q, i_1, i_2, i_3, i_4, a_t) = MLP([Q(1, i_1), Q(2, i_2), Q(3, i_3), Q(4, i_4), a_t(1), a_t(2), a_t(3)])$$

where $Q \in \mathbb{R}^{4 \times N \times K}$, and $i_1, ..., i_4$ are pointers, one per scratch pad row. The first dimension of Q corresponds to scratch pad rows, N is the number of columns (digits) and K is the

one-hot encoding dimension. To begin the ADD program, we set the initial arguments to a default value and initialize all pointers to be at the rightmost column. The only subprogram with non-default arguments is ACT, in which case the arguments indicate an action to be taken by a specified pointer.

Sorting

In this section we apply our model to a setting with potentially much longer execution traces: sorting an array of numbers using bubblesort. As in the case of addition we can use a scratch pad to store intermediate states of the array. We define the encoder as follows:

$$f_{enc}(Q, i_1, i_2, a_t) = MLP([Q(1, i_1), Q(1, i_2), a_t(1), a_t(2), a_t(3)])$$

where $Q \in \mathbb{R}^{1 \times N \times K}$ is the pad, N is the array length and K is the array entry embedding dimension. Figures 5.5 and 5.6 show an example series of states and an execution trace.

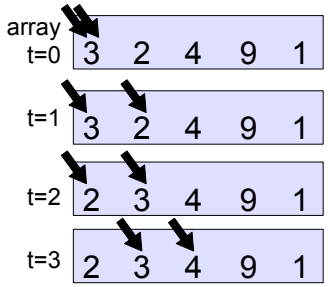


Figure 5.5: Example scratch pad and pointers used for sorting.

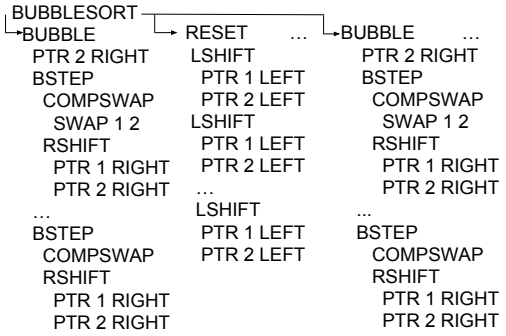


Figure 5.6: Excerpt from the trace of the learned bubblesort program.

Canonicalizing 3D models

We also apply our model to a vision task with a very different perceptual environment - pixels. Given a rendering of a 3D car, we would like to learn a visual program that "canonicalizes" the model with respect to its pose. Whatever the starting position, the program should generate a trajectory of actions that delivers the camera to the target view, *e.g.* frontal pose at a 15° elevation. For training data, we used renderings of the 3D car

CAD models from [41].

This is a nontrivial problem because different starting positions will require quite different trajectories to reach the target. Further complicating the problem is the fact that the model will need to generalize to different car models than it saw during training.

We again use a scratch pad, but here it is a very simple read-only pad that only contains a target camera elevation and azimuth – *i.e.*, the "canonical pose". Since observations here are pixels, we use a convolutional neural network f_{CNN} as the image encoder:

$$f_{enc}(Q, x, i_1, i_2, a_t) = MLP([Q(1, i_1), Q(2, i_2), f_{CNN}(x), a_t(1), a_t(2), a_t(3)])$$

where $x \in \mathbb{R}^{H \times W \times 3}$ is a car rendering at the current pose, $Q \in \mathbb{R}^{2 \times 1 \times K}$ is the pad containing canonical azimuth and elevation, i_1, i_2 are the (fixed at 1) pointer locations, and K is the one-hot encoding dimension of pose coordinates. We set $K = 24$ corresponding to 15° pose increments.

Note, critically, that NPI only has access to pixels of the rendering and the target pose, and is not provided the pose of query frames. We are also aware that one solution would be to train a pose classifier network and then find the shortest path to canonical pose via classical methods. That is also a sensible approach. However, our purpose here is to show that our method generalizes beyond the scratch pad domain to detailed images of 3D objects, and also to other environments with a single multi-task model.

5.4.2 Sample complexity and generalization

Both LSTMs and Neural Turing Machines can learn to perform sorting to a limited degree, although they have not been shown to generalize well to much longer arrays than were seen during training. However, we are interested not only in whether sorting can be accomplished, but whether a particular sorting algorithm (e.g. bubblesort) can be learned by the model, and how effectively in terms of sample complexity and generalization.

We compare the generalization ability of our model to a flat sequence-to-sequence

LSTM [146], using the same number of layers (2) and hidden units (256). Note that a flat [2] version of NPI could also learn sorting of short arrays, but because bubblesort runs in $O(N^2)$ for arrays of length N, the execution traces quickly become far too long to store the required number of LSTM states in memory. Our NPI architecture can train on much larger arrays by exploiting compositional structure; the memory requirements of any given subprogram can be restricted to $O(N)$.

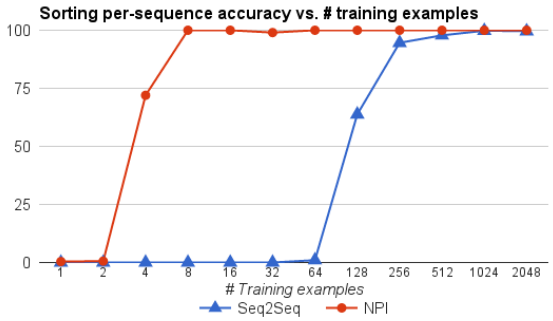


Figure 5.7: **Sample complexity**. Test accuracy of sequence-to-sequence LSTM versus NPI on length-20 arrays of single-digit numbers. Note that NPI is able to mine and train on subprogram traces from each bubblesort example.

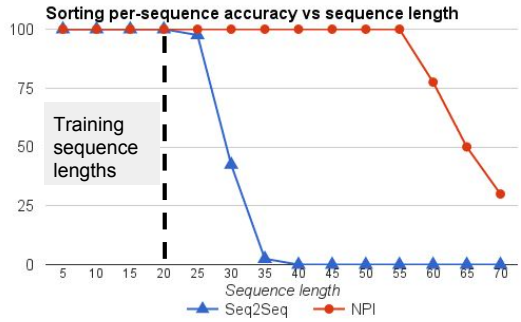


Figure 5.8: **Strong vs. weak generalization**. Test accuracy of sequence-to-sequence LSTM versus NPI on varying-length arrays of single-digit numbers. Both models were trained on arrays of single-digit numbers up to length 20.

A strong indicator of whether a neural network has learned a program well is whether it can run the program on inputs of previously-unseen sizes. To evaluate this property, we train both the sequence-to-sequence LSTM and NPI to perform bubblesort on arrays of single-digit numbers from length 2 to length 20. Compared to fixed-length inputs this raises the challenge level during training, but in exchange we can get a more flexible and generalizable sorting program.

To handle variable-sized inputs, the state representation must have some information about input sequence length and the number of steps taken so far. For example, the main BUBBLESORT program naturally needs to call its helper function BUBBLE a number of times dependent on the sequence length. We enable this in our model by adding a third

[2] By flat in this case, we mean non-compositional, not making use of subprograms, and only making calls to ACT in order to swap values and move pointers.

pointer that acts as a counter; each time BUBBLE is called the pointer is advanced by one step. The scratch pad environment also provides a bit indicating whether a pointer is at the start or end of a sequence, equivalent in purpose to end tokens used in a sequence-to-sequence model. For each length, we provided 64 example bubblesort traces, for a total of 1,216 examples. Then, we evaluated whether the network can learn to sort arrays beyond length 20. We found that the trained model generalizes well, and is capable of sorting arrays up to size 60; see Figure 5.8. At 60 and beyond, we observed a failure mode in which sweeps of pointers across the array would take the wrong number of steps, suggesting that the limiting performance factor is related to counting. In stark contrast, when provided with the 1,216 examples, the sequence-to-sequence LSTMs fail to generalize beyond arrays of length 25 as shown in Figure 5.8.

To study sample complexity further, we fix the length of the arrays to 20 and vary the number of training examples. We see in Figure 5.7 that NPI starts learning with 2 examples and is able to sort almost perfectly with only 8 examples. The sequence-to-sequence model on the other hand requires 64 examples to start learning and only manages to sort well with over 250 examples.

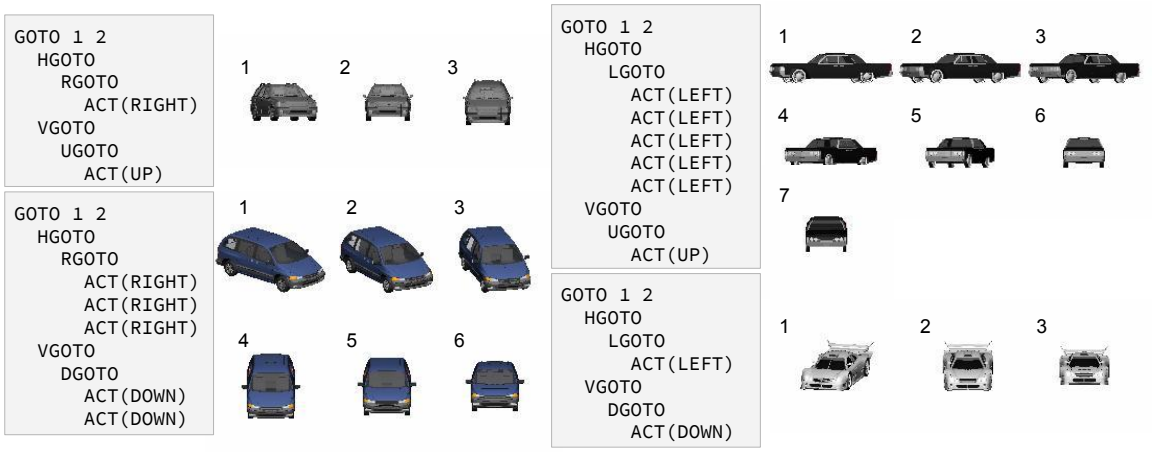


Figure 5.9: Example canonicalization of several different test set cars, of different appearance than the train set cars. The network is able to generate and execute the appropriate plan based on the starting car image. This NPI was trained on trajectories starting at azimuth $(-75°...75°)$, elevation $(0°...60°)$ in $15°$ increments. The training trajectories target azimuth $0°$ and elevation $15°$, as in the generated traces above.

Figure 5.9 shows several example canonicalization trajectories generated by our model, starting from the leftmost car. The image encoder was a convolutional network with three passes of stride-2 convolution and pooling, trained on renderings of size 128×128. The canonical target pose in this case is frontal with 15° elevation. At test time, from an initial rendering, NPI is able to canonicalize cars of varying appearance from multiple starting positions. Importantly, it can generalize to car appearances not encountered in the training set as shown in Figure 5.9.

5.4.3 Learning new programs with a fixed core

One challenge for continual learning of neural-network-based agents is that training on new tasks and experiences can lead to degraded performance in old tasks. The learning of new tasks may require that the network weights change substantially, so care must be taken to avoid catastrophic forgetting[93, 110]. Using NPI, one solution is to fix the weights of the core routing module, and only make sparse updates to the program memory.

When adding a new program the core module's routing computation will be completely unaffected; all the learning for a new task occurs in program embedding space. Of course, the addition of new programs to the memory adds a new choice of program at each time step, and an old program could mistakenly call a newly added program. To overcome this, when learning a new set of program vectors with a fixed core, in practice we train not only on example traces of the new program, but also traces of existing programs. Alternatively, a simpler approach is to prevent existing programs from calling subsequently added programs, allowing addition of new programs without ever looking back at training data for known programs. In either case, note that *only the memory slots of the new programs* are updated, and all other weights, including other program embeddings, are fixed.

Table 5.1 shows the result of adding a maximum-finding program MAX to a multitask NPI trained on addition, sorting and canonicalization. MAX first calls BUBBLESORT and then a new program RJMP, which moves pointers to the right of the sorted array, where the

max element can be read. During training we froze all weights except for the two newly-added program embeddings. We find that NPI learns MAX perfectly without forgetting the other tasks. In particular, after training a single multi-task model as outlined in the following section, learning the MAX program with this fixed-core multi-task NPI results in no performance deterioration for all three tasks.

5.4.4 Solving multiple tasks with a single network

In this section we perform a controlled experiment to compare the performance of a multi-task NPI with several single-task NPI models. Table 5.1 shows the results for addition, sorting and canonicalizing 3D car models. We trained and evaluated on 10-digit numbers for addition, length-5 arrays for sorting, and up to four-step trajectories for canonicalization. As shown in Table 5.1, one multi-task NPI can learn all three programs (and 21 subprograms) with comparable accuracy compared to each single-task NPI.

Task	**Single**	**Multi**	**+ Max**
Addition	100.0	97.0	97.0
Sorting	100.0	100.0	100.0
Canon. seen car	89.5	91.4	91.4
Canon. unseen	88.7	89.9	89.9
Maximum	-	-	100.0

Table 5.1: Per-sequence % accuracy. "+ Max" indicates performance after addition of the additional max-finding subprograms to memory. "unseen" uses a test set with disjoint car models from the training set, while "seen car" uses the same car models but different trajectories.

5.5 Conclusion

We have shown that the NPI can learn programs in very dissimilar environments with different affordances. In the context of sorting we showed that NPI exhibits very strong generalization in comparison to sequence-to-sequence LSTMs. We also showed how a trained NPI with a fixed core can continue to learn new programs without forgetting already learned programs.

CHAPTER VI

Learning to represent fine-grained visual descriptions

6.1 Introduction

A key challenge in image understanding is to correctly relate natural language concepts to the visual content of images. In recent years there has been significant progress in learning visual-semantic embeddings, e.g. for zero-shot learning [111, 124, 82, 107, 43, 136, 3] and automatically generating image captions for general web images [78, 109, 157, 70, 33]. These methods have harnessed large image and text datasets [128, 168, 88], as well as advances in deep neural networks for image and language modeling, already enabling powerful new applications such as auto-captioning images for blind users on the web [95].

Despite these advances, the problem of relating images and text is still far from solved. In particular for the fine-grained regime [161, 37, 26, 172], where images of different classes have only subtle distinctions, sophisticated language models have not been employed, perhaps due to the scarcity of large and high-quality training data. For instance on the Caltech-UCSD birds database (CUB) [161], previous zero-shot learning approaches [44, 3, 8] have used human-encoded attributes [82], or simplified language models such as bag-of-words [55], WordNet-hierarchy-derived features [99], and neural word embeddings such as Word2Vec [97] and GloVE [113].

Previous text corpora used for fine-grained label embedding were either very large but not visually focused, e.g. the entire `wikipedia`, or somewhat visually relevant but very

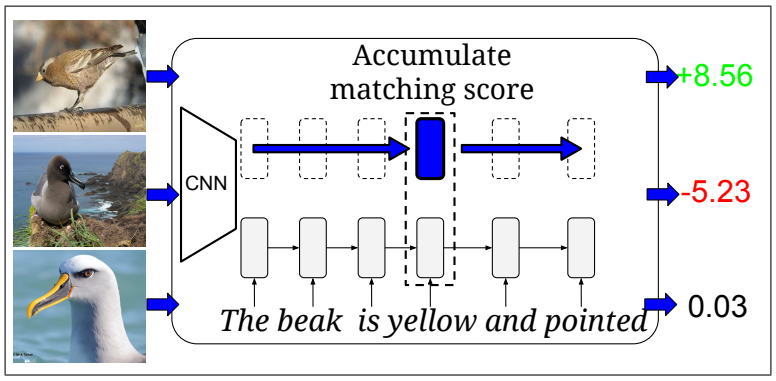


Figure 6.1: A conceptual diagram of our framework for learning visual description embeddings. Our model learns a scoring function between images and natural language descriptions. A word-based LSTM is shown, but we evaluate several alternative models.

short, e.g. the subset of `wikipedia` articles that are related to birds. Furthermore, these corpora do not provide sufficient examples of specific images and their descriptions. Given the data limitations, previous text embedding methods work surprisingly well for zero-shot visual recognition, but there remains a large gap between the text embedding methods and human-annotated attributes (28.4% vs 50.1% average top-1 per-class accuracy on CUB [3]).

In order to close the performance gap between text embeddings and human-annotated attributes for fine-grained visual recognition, we hypothesize that higher-capacity text models are required. However, more sophisticated text models would in turn require more training data, in particular aligned images and multiple visual descriptions per image for each fine-grained category. These descriptions would support both zero-shot image recognition and zero-shot image retrieval, which are strong measures of the generalization ability of both image and text models.

Our contributions in this chapter are as follows. First, we collected two datasets of fine-grained visual descriptions: one for the Caltech-UCSD birds dataset, and another for the Oxford-102 flowers dataset [106]. Second, we propose a novel extension of structured joint embedding [3], and show that it can be used for end-to-end training of deep neural language models. It also dramatically improves zero-shot retrieval performance for

all models. Third, we evaluate several variants of word- and character-based neural language models, including our novel hybrids of convolutional and recurrent networks for text modeling. We demonstrate significant improvements over the state-of-the-art on CUB and Flowers datasets in both zero-shot recognition and retrieval.

6.2 Related work

Over the past several years, advances in deep convolutional networks [77, 32, 148] have driven rapid progress in general-purpose visual recognition on large-scale benchmarks such as ImageNet [25]. The learned features of these networks have proven transferable to many other problems [108]. However, a remaining challenge is fine-grained image classification [161, 37, 26, 172], i.e. classifying objects of many visually similar classes. The difficulty is increased by the lack of extensive labeled images [111, 124, 82, 107, 43, 136], which for fine-grained data sets may even require annotation by human experts.

The setting we study in this work is both fine-grained and *zero-shot*, e.g. we want to do fine-grained classification of previously unseen categories of birds and flowers. This problem is not as contrived as it may at first seem: good performance would strongly indicate the generalization ability of image and text features; in particular that our visual description embeddings represent well the fine-grained visual *concepts* in images, rather than overfitting to known categories. Strong performance metrics for visual-semantic models are especially apropos because of the risk of overfitting recent high-capacity captioning models, e.g. memorizing (and possibly regurgitating) training captions. We compare to previous work on zero-shot recognition, and also report zero-shot text-based retrieval. Zero-shot retrieval and detection have also been studied in [24, 53, 163, 72], but no other work has studied zero-shot text-based retrieval in the fine-grained context of CUB and flowers.

There has been a surge of progress in the field of deep multi-modal representation learning in the past several years. In [105, 140], audio and video signals were combined in an autoencoder framework, yielding improved speech signal classification for noisy inputs,

and learning a shared representation across modalities. In [140], a deep Boltzmann machine architecture was used for multimodal learning on Flickr images and text tags. In addition to improved discriminative performance, it was also able to hallucinate missing modalities, i.e. generate text tags given the image, or retrieve images given text tags. In [139], a novel information theoretic objective is developed, improving the performance of deep multimodal learning for images and text.

Recent image and video captioning models [92, 157, 70, 164, 33] go beyond tags to generate natural language descriptions. These models use LSTMs [61] for modeling captions at word level and focus on generating general high-level visual descriptions of a scene. As an alternative to using LSTMs for language modeling, other works have used character-based convolutional networks [173].

Architecturally, other vision systems have trained convolutional and recurrent components (CNN-RNN) end-to-end, e.g. for encoding spatial dependencies in segmentation [174] and video classification [104]. Here we extend CNN-RNN to learn a visual semantic embedding "from scratch" at the character level, yielding competitive performance, robustness to typos, and scalability to large vocabulary.

A related line of work has been to improve label embeddings for image classification [14, 162, 43, 2, 107]. Embedding labels in an euclidean space is an effective way to model latent relationships between classes [14, 162]. For zero-shot learning, DeViSE [43] and ALE [2] employ two variants of a ranking formulation to learn a compatibility between images and textual side-information. ConSe [107] uses the probabilities of a softmax-output layer to weigh the semantic vectors of all the classes. An evaluation of embeddings for fine-grained and zero-shot classification [3] showed a large performance gap between attributes and unsupervised word embeddings.

In [38] and [9], the zero-shot recognition problem is cast as predicting parameters of a classifier given a text description of the novel category. Our work considers a similar problem, but there are major differences. We consider multi-class zero-shot recognition

and retrieval, whereas those works mainly focus on one-vs-rest detection of novel categories. More importantly, our setting assumes that we have a significant amount of visual descriptions for training high-capacity text models, whereas those works had much less text available and used TF-IDF features.

Our contribution builds on previous work on character-level language models [173] and fine-grained zero-shot learning [2] to train high capacity text encoders from scratch to jointly embed fine-grained visual descriptions and images. We demonstrate that with sufficient training data, text-based label embeddings can outperform the previous attributes-based state-of-the art for zero-shot recognition on CUB (at both word and character level).

6.3 Deep Structured Joint Embedding

In this section we describe our approach to jointly embedding images and fine-grained visual descriptions, which we call deep structured joint embedding. As in previous multimodal structured learning methods [2, 3], we learn a compatibility function of images and text. However, instead of using a bilinear compatibility function we use the inner product of features generated by deep neural encoders. An instantiation of our model using a word-level LSTM is illustrated in Figure 6.1.

Intuitively, we maximize the compatibility between a description and its matching image, and minimize compatibility with images from other classes.

Objective. Given data $\mathcal{S} = \{(v_n, t_n, y_n), n = 1, ..., N\}$ containing visual information $v \in \mathcal{V}$, text descriptions $t \in \mathcal{T}$ and class labels $y \in \mathcal{Y}$, we seek to learn functions $f_v : \mathcal{V} \rightarrow \mathcal{Y}$ and $f_t : \mathcal{T} \rightarrow \mathcal{Y}$ that minimize the empirical risk

$$\frac{1}{N} \sum_{n=1}^{N} \Delta(y_n, f_v(v_n)) + \Delta(y_n, f_t(t_n)) \tag{6.1}$$

where $\Delta : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ is the 0-1 loss. Note that N is the number of image and text *pairs* in the training set, and so a given image can have multiple corresponding captions.

Here we draw a distinction between our method from previous work on structured joint embedding [3]; namely that our objective is symmetric with respect to images and text. This has the benefit that by optimizing equation 6.1, a single model can learn to predict by conditioning on both images and text. We thus name the above objective *deep symmetric structured joint embedding (DS-SJE)*. It is possible to use just one of the two terms in Eq. 6.1. For example in [3] only the first term is used in order to train a zero-shot image classifier, i.e. only image encoder f_v is trained. In our experiments we refer to this as deep asymmetric structured joint embedding (DA-SJE).

It is also possible to build an asymmetric model in the opposite direction, i.e. only train f_t in order to perform zero-shot image retrieval, although we are not aware of previous works doing this. From a practical perspective it is clearly better to have a single model that does both tasks well. Thus in our experiments we compare DS-SJE with DA-SJE (training only f_v) for zero-shot classification.

Inference. We define a compatibility function $F : \mathcal{V} \times \mathcal{T} \rightarrow \mathbb{R}$ that uses features from learnable encoder functions $\theta(v)$ for images and $\varphi(t)$ for text:

$$F(v, t) = \theta(v)^T \varphi(t) \tag{6.2}$$

We then formulate image and text classifiers as follows:

$$f_v(v) = \arg\max_{y \in \mathcal{Y}} \mathbb{E}_{t \sim \mathcal{T}(y)}[F(v, t)] \tag{6.3}$$

$$f_t(t) = \arg\max_{y \in \mathcal{Y}} \mathbb{E}_{v \sim \mathcal{V}(y)}[F(v, t)] \tag{6.4}$$

where $\mathcal{T}(y)$ is the subset of $\mathcal{T}$ from class y, $\mathcal{V}(y)$ is the subset of $\mathcal{V}$ from class y, and the expectation is over text descriptions sampled uniformly from these subsets.

Since the compatibility function is shared by f_t and f_v, in the symmetric objective it must learn to yield accurate predictions for both classifiers. From the perspective of the text encoder, this means that text features must produce a higher compatibility score to a

matching image compared to both 1) the score of that image with any mismatching text, and 2) the score of that text with any mismatching image. We found that both 1) and 2) are important for accurate recognition and retrieval using a single model.

Learning. Since the 0-1 loss is discontinuous, we instead optimize a surrogate objective function (related to equation 6.1) that is continuous and convex:

$$\frac{1}{N}\sum_{n=1}^{N} \ell_v(v_n, t_n, y_n) + \ell_t(v_n, t_n, y_n) \tag{6.5}$$

where the misclassification losses are written as:

$$\ell_v(v_n, t_n, y_n) = \tag{6.6}$$
$$\max_{y \in \mathcal{Y}}(0, \Delta(y_n, y) + \mathbb{E}_{t \sim \mathcal{T}(y)}[F(v_n, t) - F(v_n, t_n)])$$

$$\ell_t(v_n, t_n, y_n) = \tag{6.7}$$
$$\max_{y \in \mathcal{Y}}(0, \Delta(y_n, y) + \mathbb{E}_{v \sim \mathcal{V}(y)}[F(v, t_n) - F(v_n, t_n)])$$

In practice we have many visual descriptions and many images per class. During training, in each mini-batch we first sample an image from each class, and then sample one of its ten corresponding captions. To train the model, we use SGD on Eq. 6.5 with RMSprop. Since our text encoder models are all differentiable, we backpropagate (sub)-gradients through all text network parameters for end-to-end training. For the image encoder, we keep the network weights fixed to the original GoogLeNet.

6.4 Text encoder models

In this section we describe the language models that we use for representing visual descriptions. We compare the performance on zero-shot prediction tasks in Section 6.5.

6.4.1 Text-based ConvNet (CNN)

Text-based convolutional neural networks were studied in depth in [173] for the task of document classification. The text-based CNN can be viewed as a standard CNN for images, except that the image width is 1 pixel and the number of channels is equal to the alphabet size. The 2D convolution and spatial max-pooling are replaced by temporal (1D) convolution and temporal max-pooling. After each convolution layer, we use rectified linear activation unit (ReLU), which is defined as $relu(x) = \max(0, x)$. The overall network is constructed using convolution, pooling and thresholding activation function layers, followed by fully-connected layers to project onto the embedding space. The text embedding function is thus simply $\varphi(t) = \text{CNN}(t)$; the final hidden layer of the CNN.

The maximum input length for character sequences is constrained by the network architecture, but variable length sequences beneath this limit are handled by zero-padding the input past the final input character. The Word-CNN is exactly the same as Char-CNN except that the alphabet of the Char-CNN is replaced with the vocabulary of the Word-CNN. Of course, the vocabulary is much larger, typically at least several thousand words compared to a few dozen characters in an alphabet. However, the sequence length is significantly reduced.

6.4.2 Convolutional Recurrent Net (CNN-RNN)

A potential shortcoming of convolution-only text models is that they lack a strong temporal dependency along the input text sequence. However, the CNN models are extremely fast and scale well to long sequences such as character strings. To get the benefits of both recurrent models and CNNs, we propose to stack a recurrent network on top of a mid-level temporal CNN hidden layer. Intuitively, the CNN hidden activation is split along the time dimension (in our case when the dimension was reduced to 8) and treated as an input sequence of vectors. The entire resulting network is still end-to-end differentiable.

This approach has the advantage that low-level temporal features can be learned effi-

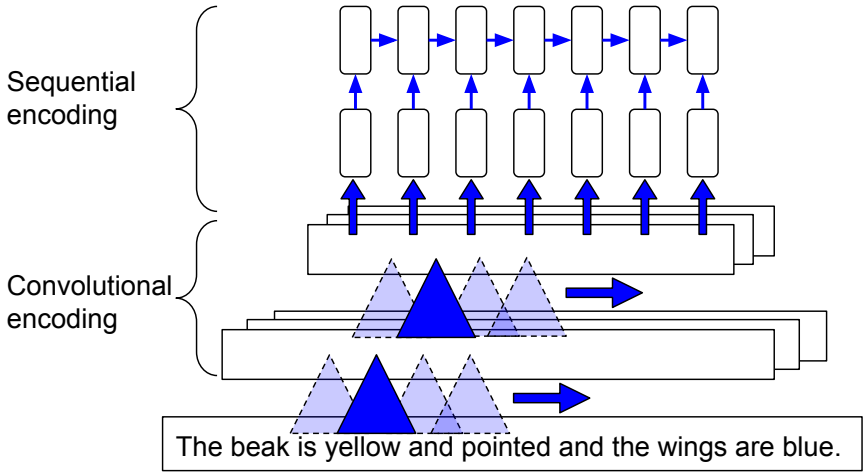


Figure 6.2: Our proposed convolutional-recurrent net.

ciently with fast convolutional networks, and temporal structure can still be exploited at the more abstract level of mid-level features. This can be viewed as modeling temporal structure at the abstract or conceptual level, not strictly dilineated by word boundaries. The approach is well-suited to the case of character-level processing (Char-CNN-RNN). We also evaluate a word-level version (Word-CNN-RNN).

Figure 6.2 illustrates the convolutional-recurrent approach. The final encoded feature is the average hidden unit activation over the sequence, i.e. $\varphi(t) = 1/L \sum_{i=1}^{L} h_i$, where h_i is the hidden activation vector for the i-th frame and L is the sequence length. The resulting scoring function can be viewed as a linear accumulation of evidence for compatibility with a query image (illustrated in Figure 6.1). It is also a linearized version of attention over the text sequence. This has the advantage that at test time for classification or retrieval, one can use the averaged hidden units as a feature, but for diagnostic purposes one can backtrace the score computation to each time step of text processing.

6.4.3 Long Short-Term Memory (LSTM)

As opposed to the CNN models, the LSTM explicitly takes into account the temporal structure starting from words or characters. We refer readers to [61] for full details. To extract a text embedding from the LSTM text encoder, we take the temporal average of the

final layer hidden units, i.e. $\varphi(t) = 1/L \sum_{i=1}^{L} h_i$ (defined similarly as in Section 6.4.2).

6.4.4 Baseline representations

Since we gathered a significant amount of new data, traditional (e.g. non-"deep") text representations should also improve in performance. To evaluate whether the neural encoders provide an additional benefit, we compare against several classical methods.

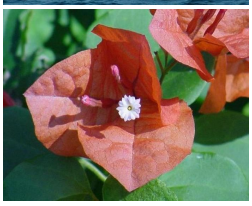
For the BoW model, we first compute the vocabulary V of all of the unique words appearing in the visual descriptions. Then, we encode each description as a binary vector indicating the presence or absence of each word. The embedding function is simply the output of a multi-layer perceptron (MLP), $\varphi(t) = \text{MLP}(I(t))$. where $I(\cdot)$ maps t to an indicator vector in $\{0, 1\}^{|V|}$. In practice we found a single layer linear projection was sufficient for surprisingly good performance.

We also evaluate a baseline that represents descriptions using unsupervised word embeddings learned by word2vec [97]. Previous works on visual-semantic embedding have directly used the word embedings of target classes for zero-shot learning tasks. However, in our case we have access to many visual descriptions, and we would like to extract vector representations of them in real time; i.e. without re-running word2vec training. A very simple way to do this is to average the word embeddings of each word in the visual description. Although this loses the structure of the sentence, this nevertheless yields a strong baseline and in practice performs similarly to bag of words.

Finally, an important point of comparison is attributes, which contain rich structured information far more compactly than informal visual descriptions. As in the case of bag-of-words, we learn a single-layer encoder function mapping attributes to the embedding space. Since the number of attribute vectors is very small (only one per class), the risk of over-fitting strongly limits the encoder network capacity. The CUB dataset also has per-image attributes, but we found that using these does not improve performance compared to using a single averaged attribute vector per class.

The bird has a white underbelly, black feathers in the wings, a large wingspan, and a white beak.

This bird has distinctive-looking brown and white stripes all over its body, and its brown tail sticks up.



This flower has a central white blossom surrounded by large pointed red petals which are veined and leaflike.

Light purple petals with orange and black middle green leaves

Figure 6.3: Example annotations of birds and flowers.

6.5 Experimental results

In this section we describe our experiments on the Caltech-UCSD Birds dataset (CUB) and Oxford Flowers-102 (Flowers) dataset. CUB contains 11,788 bird images from 200 different categories. Flowers contains 8189 flower images from 102 different categories. Following [2], the images in CUB are split into 100 training, 50 validation, and 50 disjoint test categories[1]. As in [9], the images in Flowers are split into 82 training + validation and 20 test classes. For the image features, we extracted $1,024$-dimensional pooling units from GoogLeNet [148] with batch normalization [66] implemented in Torch[2]. For each image, we extracted middle, upper left, upper right, lower left and lower right crops for the original and horizontally-flipped image, resulting in 10 views per training image. At test time we only use the original image resized to 224×224.

For all word-level models (BoW, Word-LSTM, Word-CNN, Word-CNN-RNN), we used all vocabulary words in the dataset. For character-level models (Char-LSTM, Char-CNN, Char-CNN-RNN), the alphabet consisted of all lowercase characters and punctuation. The CNN input size (sequence length) was set to 30 for word-level and 201 for character-level models; longer text inputs are cut off at this point and shorter ones are zero-

[1] Since we evaluate in the zero-shot setting, it is critical that the validation categories be disjoint from the training categories. Once hyperparameters have been cross-validated, the training + validation (150) classes can be taken as the training set. For Flowers, we do not do any parameter cross-validation, we use the same parameters found for CUB.

[2] github.com/soumith/imagenet-multiGPU.torch

padded. All text embeddings used a 1024-dimensional embedding layer to match the size of the image embedding. We kept the image encoder fixed, and used RMSprop with base learning rate 0.0007 and minibatch size 40.

6.5.1 Collecting fine-grained visual descriptions

In this section we describe the collection of our new dataset of fine-grained visual descriptions. For each image in CUB and Flowers, we collected ten single-sentence visual descriptions. We used the Amazon Mechanical Turk (AMT) platform for data collection, using non-"Master" certified workers situated in the US with average work approval rating above 95%. We asked workers to describe only visual appearance in at least 10 words, to avoid figures of speech, to avoid naming the species even if they knew it, and not to describe the background or any actions being taken. The prompt included three example sentences and a diagram labeling specific parts of a bird (e.g. tarsus) and flower (e.g. stamen) so that non-experts could describe many different aspects without reference to external sources such as Wikipedia. Workers were not told the species.

Figure 6.3 shows several representative examples of the results from our data collection. The descriptions almost always accurately describe the image, to varying degrees of comprehensiveness. Thus, in some cases multiple captions might be needed to fully disambiguate the species of bird category. However, as we show subsequently, the data is descriptive and large enough to support training high-capacity text models and greatly improve the performance of text-based embeddings for zero-shot learning.

	Top-1 Acc (%)		AP@50 (%)	
Embedding	**DA-SJE**	**DS-SJE**	**DA-SJE**	**DS-SJE**
ATTRIBUTES	50.9	50.4	20.4	**50.0**
WORD2VEC	38.7	38.6	7.5	33.5
BAG-OF-WORDS	43.4	44.1	24.6	39.6
CHAR CNN	47.2	48.2	2.9	42.7
CHAR LSTM	22.6	21.6	11.6	22.3
CHAR CNN-RNN	54.0	54.0	6.9	45.6
WORD CNN	50.5	51.0	3.4	43.3
WORD LSTM	52.2	53.0	**36.8**	46.8
WORD CNN-RNN	**54.3**	**56.8**	4.8	48.7

Table 6.1: Zero-shot recognition and retrieval on CUB. "DS-SJE" and "DA-SJE" refer to symmetric and asymmetric forms of our joint embedding objective, respectively.

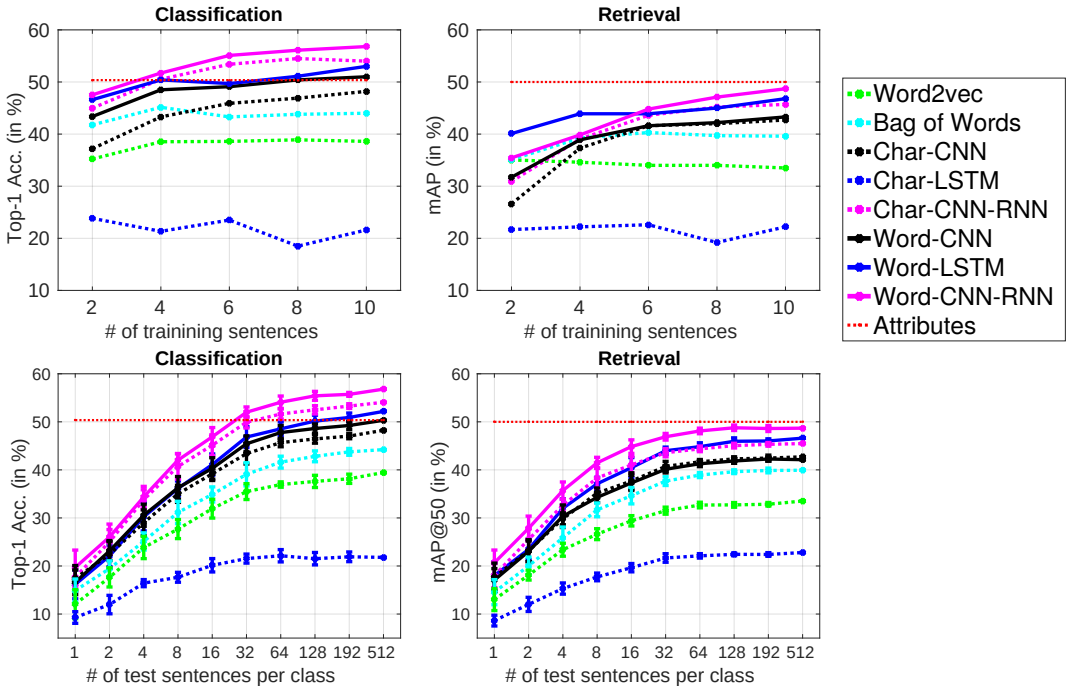


Figure 6.4: Top: Performance impact of increasing the number of training sentences. Bottom: Increasing the number of test sentences used at test time.

6.5.2 CUB zero-shot recognition and retrieval

In this section we describe the protocol and results for our zero-shot tasks. For both recognition and retrieval, we first extract text encodings from test captions and average

them per-class. In this experiment we use *all* test captions and in a later section we vary this number, including using a single caption per class. In recognition, the resulting classifier is defined by equation 6.3. Note that by linearity we can move the expectation inside the compatibility function:

$$f_v(v) = \arg\max_{y \in \mathcal{Y}} \theta(v)^T \mathbb{E}_{t \sim \mathcal{T}(y)}[\varphi(t)] \tag{6.8}$$

The expectation above is estimated by the averaged per-class text embedding that we compute. Hence the accuracy of the classifier is determined not only by the underlying image and text encoders, but also by the quantity of text available at test time.

In the retrieval task, we rank all test set images according to compatibility (equation 6.2) with the averaged text embedding for each class. We report the AP@50, i.e. the percent of top-50 scoring images whose class matches that of the text query, averaged over the 50 test classes. Table 6.1 summarizes our results. Both in the classification (first two columns) and for retrieval (last two columns) settings, the symmetric (DS-SJE) formulation of our model improves over the asymmetric (DA-SJE) formulation. Especially for retrieval, DS-SJE performs much better than DA-SJE consistently for all the text embedding variants. It makes the difference between working very well and failing, particularly for the high-capacity models which likely overfit to the classification task in the asymmetric setting.

In the classification setting there are notable differences between the language models. For DA-SJE (first column), Char-CNN-RNN (54.0% Top-1 Acc) and Word-CNN-RNN (54.3%) outperform the attributes-based state-of-the-art [3] for zero-shot classification (50.1%). In fact we replicated the attribute-based model in [3] and got slightly better results (50.9%, also reported in Table 6.1), probably due to training on 10 image crops instead of a single crop. Similar observations hold for DS-SJE (second column). Notably for DS-SJE, Char-CNN-RNN (54.0%), Word-CNN (51.0%), Word-LSTM (53.0%) and Word-CNN-RNN (56.8%) outperform the attributes. In the case of retrieval and DS-SJE (last column), attributes still performs the best (50.0% AP), but Word-CNN-RNN (48.7%)

approaches this result.

Among the character-level models, Char-CNN is significantly better than Char-LSTM. Additionally, our proposed Char-CNN-RNN, which adds a temporal aspect to Char-CNN, improves over the other two character-based deep methods and also over the attribute-based state-of-the-art for classification. This is notable because it establishes that character-level models can extract visually-discriminative text representations of previously-unseen categories. Furthermore, combining convolutional and temporal processing appears to be a promising approach to learn at the character level. Word-level models improve performance further and can also significantly outperform attributes.

6.5.3 Effect of visual description training set size

In this section we investigate the effect of increasing the number of sentences used in training on zero-shot classification and retrieval performance. Obviously having more data is better, but with this experiment we can see which methods are best at which operating point of data size (hence cost). We start with using one sentence per image and we increase this number gradually to ten sentences per image for training. For testing, the protocol is the same as in Table 6.1, and we use all available captions per class.

We show the performance of several text encoding models in Fig 6.4. In zero-shot classification, attributes are competitive when two captions per-image are available, but with more training captions the deep network models win. For retrieval, the crossover point might happen with more than ten captions per image as the results seem to be increasing. The baseline word2vec and BoW encodings do not gain much from more data. Given a moderate number of training sentences per image, neural text encoders can improve over the state-of-the-art attribute-based methods significantly.

Among neural text encoders, Char-LSTM fares worst and also does not appear to gain consistently from additional data. It may be that the long training sequence length increases the difficulty of LSTM training, relative to the word-based approach. Stacking a recurrent

module on top of a text convolutional network appears to avoid this problem, achieving significantly better performance than the Word-LSTM especially with more than 4 sentences for training. It also has the nice property of robustness to typos. Overall, Word-CNN-RNN achieved the best performance.

6.5.4 Effect of test visual description length

In a real application relating images and text (e.g. text-based image retrieval), most users would prefer to describe a visual concept concisely, rather than writing a detailed article with many sentences. Thus, we evaluate the performance of our model using a varying number of query descriptions per class at test time. The experimental protocol is a slight modification of that used in Table 6.1.

As before, we extract text embeddings from test set captions and average them per-class. In this case, we extract embeddings separately using $\{1, 2, 4, 8, 16, 32, 64, 128\}$ and also *all* descriptions available per class. For each description length, we report the resulting zero-shot classification accuracy and zero-shot retrieval AP@50. Since we do not use all available test captions per class, we perform 10 iterations of this procedure while randomly sampling the descriptions used for each class.

Figure 6.4 shows the averaged results for zero-shot classification and for zero-shot retrieval. Both figures include error bars to ± 1 standard deviation. Note that the error bars are larger towards the left side of both figures because in the few-text case, especially discriminative or especially vague (or wrong) descriptions can have a relatively larger impact on the text embedding quality. BoW again shows a surprisingly good performance, significantly better than word2vec and competitive with Char-CNN. However, the word-level neural text encoders outperform word2vec and BoW at all operating points.

	Top-1 Acc (%)		AP@50 (%)	
Embedding	**DA-SJE**	**DS-SJE**	**DA-SJE**	**DS-SJE**
WORD2VEC	54.6	54.2	16.3	52.1
BAG-OF-WORDS	56.7	57.7	28.2	57.3
CHAR CNN	51.1	47.3	8.3	46.1
CHAR LSTM	29.1	25.8	19.3	27.0
CHAR CNN-RNN	61.7	63.7	13.6	57.3
WORD CNN	60.2	60.7	8.7	56.3
WORD LSTM	**62.3**	64.5	**45.9**	52.3
WORD CNN-RNN	60.9	**65.6**	7.6	**59.6**

Table 6.2: Zero-shot % recognition accuracy and retrieval average precision on Flowers.

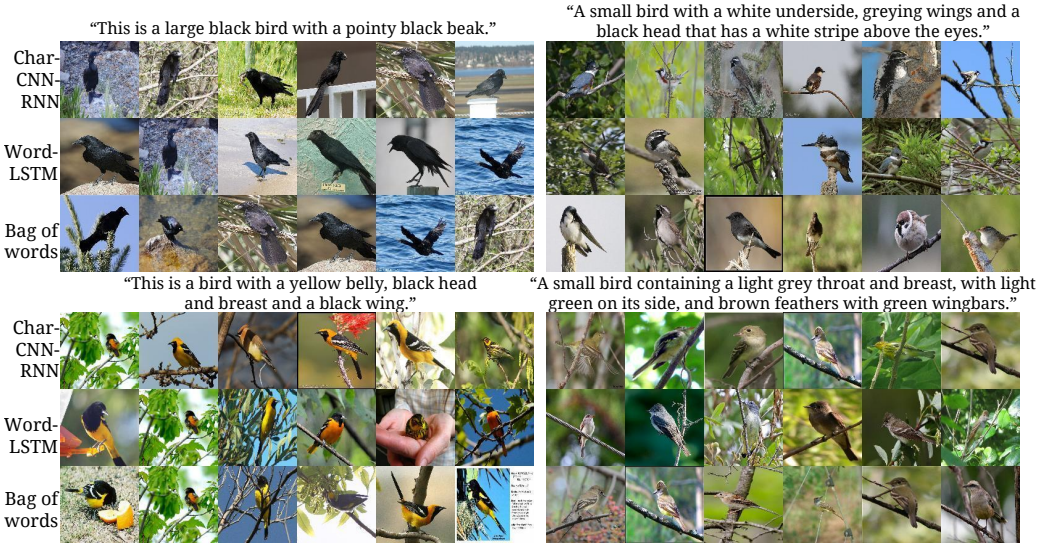


Figure 6.5: Zero-shot retrieval given a single query sentence. Each row corresponds to a different text encoder.

6.5.5 Flowers zero-shot recognition and retrieval

To demonstrate that our results generalize beyond the case of bird images, we report the same set of experiments on the Flowers dataset. The experimental setting here is the same as in Sec 6.5.2, except that there is no attributes baseline due to lack of labeled attributes for this dataset. All neural text model architectures are the same as we used for CUB, and we used the same hyperparameters from cross-validation on CUB. Table 6.2 summarizes our results.

Char CNN-RNN achieves competitive results to word-level models both for DA-SJE

Approach	CUB	Flowers
CSHAP$_H$ [64]	17.5	–
AHLE [2]	27.3	–
TMV-HLP [45]	47.9	–
SJE [3]	50.1	–
DA-SJE (ours)	54.3	62.3
DS-SJE (ours)	**56.8**	**65.6**

Table 6.3: Summary of zero-shot % classification accuracies. Note that different features are used in each work, although [2] uses the same features as in this work.

and DS-SJE. The word-level models achieve the best result, significantly better than both the shallow embeddings and character-level models. Among different models, Word LSTM is the winner for DA-SJE both in classification and retrieval. On the other hand, Word CNN-RNN is the winner for DS-SJE for the same. As in the case for CUB, we found that DS-SJE achieves strong retrieval performance, and DA-SJE often fails in comparison.

6.5.6 Qualitative results

Figure D.1 shows several example zero-shot retrieval results using a single text description. Both the text queries and images are real data points drawn from the test set. We observe that having trained on our dataset of visual descriptions, our proposed method returns results that accurately reflect the text, even when using only a single caption. Quantitatively, BoW achieves 14.6% AP@50 with a single query compared to 18.0% with word-LSTM and 20.7% with Word-CNN-RNN.

Note that although almost all retrieved images match the text query well, the actual class of that image can still be incorrect. This is why the average precision may seem low compared to the generally good qualitative results. The performance appears to degrade gracefully; our model at least returns visually-consistent results if not of the correct class. Furthermore, some queries are inherently ambiguous and could match multiple classes equally well, so low precision is not necessarily the fault of the model. We show a t-SNE embedding of test-set description embeddings in Figure 6.6, successfully clustering

Figure 6.6: t-SNE embedding of *test class* description embeddings from Oxford-102 (left) and CUB (right), marked with corresponding images. Best viewed with zoom.

according to visual similarities (i.e. color, shape). Additional examples from test images and queries are included in the supplementary material.

6.5.7 Comparison to the state-of-the-art

In this section we compare to the previously published results on CUB, including results that use the same zero-shot split. CSHAP$_H$ [64] uses 4K-dim features from the Oxford VGG net [134] and also attributes to learn a hypergraph on the attribute space. AHLE [2] uses Fisher vector image features and attribute embeddings to learn a bilinear compatibility function between these embeddings. TMV-HLP [45] builds a hypergraph on a multi-view embedding space learned via CCA which uses deep image features and attributes. In SJE [3] as in AHLE [2] a compatibility function is learned, in this case between 1K-dim GoogleNet [148] features and various other embeddings including attributes. Our method achieves significant improvements over all of these baselines, despite the fact that we do not use attributes.

Previously-reported zero-shot results on the Flowers dataset [38, 9] do not report multi-class classification (instead reporting binary one-vs-rest detection of unseen categories) or do not currently have published splits. However, it will be interesting to compare these methods of "predicting a classifier" given image descriptions in the large-data setting with our new caption collection.

Overall, the results in Table 6.3 demonstrate that state-of-the-art zero-shot prediction performace can be achieved directly from text descriptions. This does not require access to any form of test label embeddings. Although attributes are richer and more compact than text descriptions, attributes alone form a very small training set. One explanation for the better performance of using our descriptions is that having many noisy human-generated descriptions acts as an effective regularizer on the learned compatibility function. This is especially important when training deep networks, which in our model are used for both the image and text encoding components. Indeed, we observed that when training with attributes, we had to use far fewer epochs (7 compared to 300) to avoid over-fitting.

6.6 Discussion

We developed a deep symmetric joint embedding model, collected a high-quality dataset of fine-grained visual descriptions, and evaluated several deep neural text encoders. We showed that a text encoder trained from scratch on characters or words can achieve state-of-the-art zero-shot recognition accuracy on CUB, outperforming attributes. Our text encoders achieve a competitive retrieval result compared to attributes, and unlike attributes can be directly used to build a language-based retrieval system.

Our visual descriptions data also improved the zero shot accuracy using BoW and word2vec encoders. While these win in the smaller data regime, higher capacity encoders dominate when enough data is available. Thus our contributions (data, objective and text encoders) improve performance at multiple operating points of training text size.

CHAPTER VII

Generating Images from Text Descriptions

7.1 Introduction

In this work we are interested in translating text in the form of single-sentence human-written descriptions directly into image pixels. For example, “this small bird has a short, pointy orange beak and white belly” or ”the petals of this flower are pink and the anther are yellow”. The problem of generating images from visual descriptions gained interest in the research community, but it is far from being solved.

Traditionally this type of detailed visual information about an object has been captured in attribute representations - distinguishing characteristics the object category encoded into a vector [40, 80, 112, 82], in particular to enable zero-shot visual recognition [44, 3], and recently for conditional image generation [165].

While the discriminative power and strong generalization properties of attribute representations are attractive, attributes are also cumbersome to obtain as they may require domain-specific knowledge. In comparison, natural language offers a general and flexible interface for describing objects in any space of visual categories. Ideally, we could have the generality of text descriptions with the discriminative power of attributes.

Recently, deep convolutional and recurrent networks for text have yielded highly discriminative and generalizable (in the zero-shot learning sense) text representations learned automatically from words and characters [120]. These approaches exceed the previous

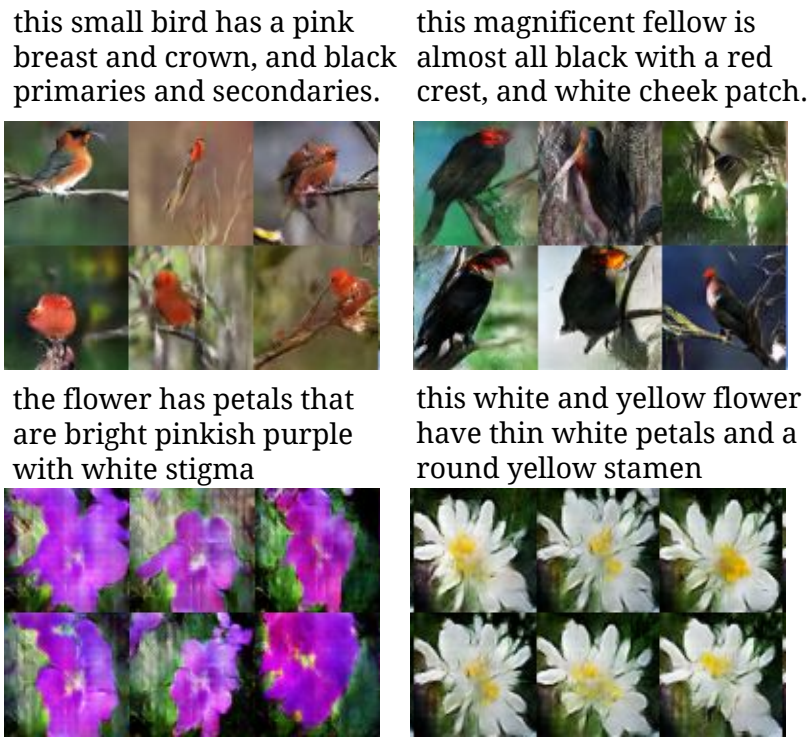


Figure 7.1: Examples of generated images from text descriptions. Left: captions are from zero-shot (held out) categories, unseen text. Right: captions are from the training set.

state-of-the-art using attributes for zero-shot visual recognition on the Caltech-UCSD birds database [158], and also are capable of zero-shot caption-based retrieval. Motivated by these works, we aim to learn a mapping directly from characters to image pixels.

To solve this challenging problem requires solving two sub-problems: first, learn a text feature representation that captures the important visual details; and second, use these features to synthesize a compelling image that a human might mistake for real. Fortunately, deep learning has enabled enormous progress in both subproblems - natural language representation and image synthesis - in the previous several years, and we build on this for our current task.

However, one difficult remaining issue not solved by deep learning alone is that the distribution of images conditioned on a text description is highly multimodal, in the sense that there are very many plausible configurations of pixels that correctly illustrate the description. The reverse direction (image to text) also suffers this problem but learning is made practical by the fact that the word or character sequence can be decomposed sequentially

according to the chain rule; i.e. one trains the model to predict the next token conditioned on the image and all previous tokens; a more tractable prediction problem.

This conditional multi-modality is thus a very natural application for generative adversarial networks [49], in which the generator network is optimized to fool the adversarially-trained discriminator into predicting that synthetic images are real. By conditioning both generator and discriminator on side information (also studied by *Mirza and Osindero* [101] and *Denton et al.* [27]), we can naturally model this phenomenon since the discriminator network acts as a "smart" adaptive loss function.

Our main contribution in this work is to develop a simple and effective GAN architecture and training strategy that enables compelling text to image synthesis of bird and flower images from human-written descriptions. We mainly use the Caltech-UCSD Birds dataset and the Oxford-102 Flowers dataset along with five text descriptions per image we collected as our evaluation setting. Our model is trained on a subset of training categories, and we demonstrate its performance both on the training set categories and on the testing set, i.e. "zero-shot" text to image synthesis. In addition to birds and flowers, we apply our model to more general images and text descriptions in the MS COCO dataset [88].

7.2 Related work

Key challenges in multimodal learning include learning a shared representation across modalities, and to predict missing data (e.g. by retrieval or synthesis) in one modality conditioned on another. *Ngiam et al.* [105] trained a stacked multimodal autoencoder on audio and video signals and were able to learn a shared modality-invariant representation. *Srivastava and Salakhutdinov* [140] developed a deep Boltzmann machine and jointly modeled images and text tags. *Sohn et al.* [139] proposed a multimodal conditional prediction framework (hallucinating one modality given the other) and provided theoretical justification.

Many researchers have recently exploited the capability of deep convolutional *decoder* networks to generate realistic images. *Dosovitskiy et al.* [36] trained a deconvolutional

network (several layers of convolution and upsampling) to generate 3D chair renderings conditioned on a set of graphics codes indicating shape, position and lighting. *Yang et al.* [167] added an encoder network as well as actions to this approach. They trained a recurrent convolutional encoder-decoder that rotated 3D chair models and human faces conditioned on action sequences of rotations. *Reed et al.* [119] encode transformations from analogy pairs, and use a convolutional decoder to predict visual analogies on shapes, video game characters and 3D cars.

Generative adversarial networks [49] have also benefited from convolutional decoder networks, for the generator network module. *Denton et al.* [27] used a Laplacian pyramid of adversarial generator and discriminators to synthesize images at multiple resolutions. This work generated compelling high-resolution images and could also condition on class labels for controllable generation. *Radford et al.* [115] used a standard convolutional decoder, but developed a highly effective and stable architecture incorporating batch normalization to achieve striking image synthesis results.

The main distinction of our work from the conditional GANs described above is that our model conditions on *text descriptions* instead of class labels. To our knowledge it is the first end-to-end differentiable architecture from characters to pixels. Furthermore, we introduce a manifold interpolation regularizer for the GAN generator that significantly improves the quality of generated samples, including on held out zero shot categories on CUB.

The bulk of previous work on multimodal learning from images and text uses retrieval as the target task, i.e. fetch relevant images given a text query or vice versa. However, in the past year, there has been a breakthrough in using recurrent neural network decoders to generate text descriptions conditioned on images [157, 92, 70, 33]. These typically condition a Long Short-Term Memory [61] on the top-layer features of a deep convolutional network to generate captions using the MS COCO [88] and other captioned image datasets. *Xu et al.* [164] incorporated a recurrent visual attention mechanism for improved results.

Other tasks besides conditional generation have been considered in recent work. *Ren*

et al. [121] generate answers to questions about images. This approach was extended to incorporate an explicit knowledge base [159]. *Zhu et al.* [175] applied sequence models to both text (in the form of books) and movies to perform a joint alignment.

In contemporary work *Mansimov et al.* [91] generated images from text captions, using a variational recurrent autoencoder with attention to paint the image in multiple steps, similar to DRAW [51]. Impressively, the model can perform reasonable synthesis of completely novel (unlikely for a human to write) text such as "a stop sign is flying in blue skies", suggesting that it does not simply memorize. While the results are encouraging, the problem is highly challenging and the generated images are not yet realistic, i.e., mistakeable for real. Our model can in many cases generate visually-plausible 64×64 images conditioned on text, and is also distinct in that our entire model is a GAN, rather only using GAN for post-processing.

Building on ideas from these many previous works, we develop a simple and effective approach for text-based image synthesis using a character-level text encoder and class-conditional GAN. We propose a novel architecture and learning strategy that leads to compelling visual results. We focus on the case of fine-grained image datasets, for which we use the recently collected descriptions for Caltech-UCSD Birds and Oxford Flowers with 5 human-generated captions per image [120]. We train and test on class-disjoint sets, so that test performance can give a strong indication of generalization ability which we also demonstrate on MS COCO images with multiple objects and various backgrounds.

7.3 Background

In this section we briefly describe several previous works that our method is built upon.

7.3.1 Generative adversarial networks

Generative adversarial networks (GANs) consist of a generator G and a discriminator D that compete in a two-player minimax game: The discriminator tries to distinguish real training data from synthetic images, and the generator tries to fool the discriminator. Concretely, D and G play the following game on V(D,G):

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{x\sim p_z(z)}[\log(1 - D(G(z)))] \tag{7.1}$$

Goodfellow et al. [49] prove that this minimax game has a global optimium precisely when $p_g = p_{data}$, and that under mild conditions (e.g. G and D have enough capacity) p_g converges to p_{data}. In practice, in the start of training samples from D are extremely poor and rejected by D with high confidence. It has been found to work better in practice for the generator to maximize $\log(D(G(z)))$ instead of minimizing $\log(1 - D(G(z)))$.

7.3.2 Deep symmetric structured joint embedding

To obtain a visually-discriminative vector representation of text descriptions, we follow the approach of *Reed et al.* [120] by using deep convolutional and recurrent text encoders that learn a correspondence function with images. The text classifier induced by the learned correspondence function f_t is trained by optimizing the following structured loss:

$$\frac{1}{N}\sum_{n=1}^{N} \Delta(y_n, f_v(v_n)) + \Delta(y_n, f_t(t_n)) \tag{7.2}$$

where $\{(v_n, t_n, y_n) : n = 1, ..., N\}$ is the training data set, Δ is the 0-1 loss, v_n are the images, t_n are the corresponding text descriptions, and y_n are the class labels. Classifiers f_v and f_t are parametrized as follows:

$$f_v(v) = \arg\max_{y\in\mathcal{Y}} \mathbb{E}_{t\sim\mathcal{T}(y)}[\phi(v)^T\varphi(t))] \tag{7.3}$$

$$f_t(t) = \arg\max_{y\in\mathcal{Y}} \mathbb{E}_{v\sim\mathcal{V}(y)}[\phi(v)^T\varphi(t))] \tag{7.4}$$

where ϕ is the image encoder (e.g. a deep convolutional neural network), φ is the text encoder (e.g. a character-level CNN or LSTM), $\mathcal{T}(y)$ is the set of text descriptions of class y and likewise $\mathcal{V}(y)$ for images. The intuition here is that a text encoding should have a higher compatibility score with images of the correspondong class compared to any other class and vice-versa.

To train the model a surrogate objective related to Equation 7.2 is minimized (see *Akata et al.* [3] for details). The resulting gradients are backpropagated through φ to learn a discriminative text encoder. *Reed et al.* [120] found that different text encoders worked better for CUB versus Flowers, but for full generality and robustness, in this work we always used a hybrid character-level convolutional-recurrent network.

7.4 Method

Our approach is to train a deep convolutional generative adversarial network (DC-GAN) conditioned on text features encoded by a hybrid character-level convolutional-recurrent neural network. Both the generator network G and the discriminator network D perform feed-forward inference conditioned on the text feature.

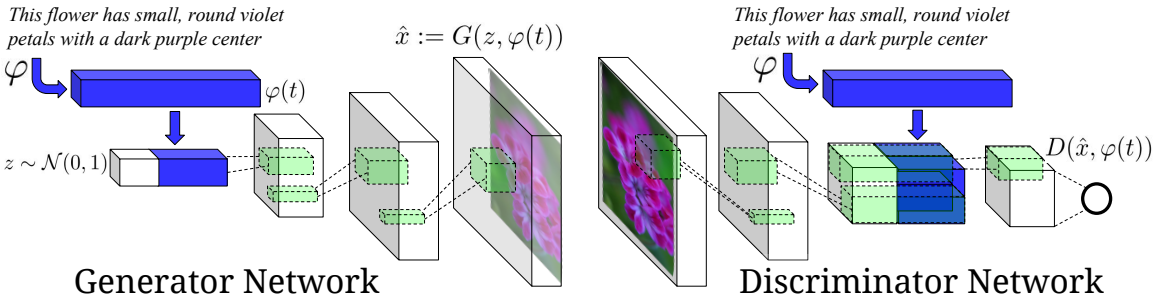


Figure 7.2: Our text-conditional convolutional GAN architecture. Text encoding $\varphi(t)$ is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

7.4.1 Network architecture

We use the following notation. The generator network is denoted $G : \mathbb{R}^Z \times \mathbb{R}^T \rightarrow \mathbb{R}^D$, the discriminator as $D : \mathbb{R}^D \times \mathbb{R}^T \rightarrow \{0, 1\}$, where T is the dimension of the text

description embedding, D is the dimension of the image, and Z is the dimension of the noise input to G. We illustrate our network architecture in Figure 7.2.

In the generator G, first we sample from the noise prior $z \in \mathbb{R}^Z \sim \mathcal{N}(0, 1)$ and we encode the text query t using text encoder φ. The description embedding $\varphi(t)$ is first compressed using a fully-connected layer to a small dimension (we used 128) followed by leaky-ReLU and then concatenated to the noise vector z. Following this, inference proceeds as in a normal deconvolutional network: we feed-forward it through the generator G; a synthetic image $\hat{x}$ is generated via $\hat{x} \leftarrow G(z, \varphi(t))$. Image generation corresponds to feed-forward inference in the generator G conditioned on query text and a noise sample.

In the discriminator D, we perform several layers of stride-2 convolution with spatial batch normalization [66] followed by leaky ReLU. We again reduce the dimensionality of the description embedding $\varphi(t)$ in a (separate) fully-connected layer followed by rectification. When the spatial dimension of the discriminator is 4×4, we replicate the description embedding spatially and perform a depth concatenation. We then perform a 1×1 convolution followed by rectification and a 4×4 convolution to compute the final score from D. Batch normalization is performed on all convolutional layers.

7.4.2 Matching-aware discriminator (GAN-CLS)

The most straightforward way to train a conditional GAN is to view (text, image) pairs as joint observations and train the discriminator to judge pairs as real or fake. This type of conditioning is naive in the sense that the discriminator has no explicit notion of whether real training images match the text embedding context.

However, as discussed also by [46], the dynamics of learning may be different from the non-conditional case. In the beginning of training, the discriminator ignores the conditioning information and easily rejects samples from G because they do not look plausible. Once G has learned to generate plausible images, it must also learn to align them with the conditioning information, and likewise D must learn to evaluate whether samples from G

meet this conditioning constraint.

In naive GAN, the discriminator observes two kinds of inputs: real images with matching text, and synthetic images with arbitrary text. Therefore, it must implicitly separate two sources of error: unrealistic images (for *any* text), and realistic images of the wrong class that mismatch the conditioning information. Based on the intuition that this may complicate learning dynamics, we modified the GAN training algorithm to separate these error sources. In addition to the real / fake inputs to the discriminator during training, we add a third type of input consisting of real images with mismatched text, which the discriminator must learn to score as fake. By learning to optimize image / text matching in addition to the image realism, the discriminator can provide an additional signal to the generator.

Algorithm 4 GAN-CLS training algorithm with step size α, using minibatch SGD.

Input: minibatch images x, matching text t, mismatching $\hat{t}$, # steps S
for $n = 1$ **to** S **do**
$h \leftarrow \varphi(t)$ ▷ Encode matching text description
$\hat{h} \leftarrow \varphi(\hat{t})$ ▷ Encode mis-matching text description
$z \sim \mathcal{N}(0, 1)^Z$ ▷ Draw sample of random noise
$\hat{x} \leftarrow G(z, h)$ ▷ Forward through generator
$s_r \leftarrow D(x, h)$ ▷ real image, right text
$s_w \leftarrow D(x, \hat{h})$ ▷ real image, wrong text
$s_f \leftarrow D(\hat{x}, h)$ ▷ fake image, right text
$\mathcal{L}_D \leftarrow \log(s_r) + (\log(1 - s_w) + \log(1 - s_f))/2$
$D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D$ ▷ Update discriminator
$\mathcal{L}_G \leftarrow \log(s_f)$
$G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G$ ▷ Update generator
end for

Algorithm 4 summarizes the training procedure. After encoding the text, image and noise (lines 3-5) we generate the fake image ($\hat{x}$, line 6). s_r indicates the score of associating a real image and its corresponding sentence (line 7), s_w measures the score of associating a real image with an arbitrary sentence (line 8), and s_f is the score of associating a fake image with its corresponding text (line 9). Note that we use $\partial \mathcal{L}_D / \partial D$ to indicate the gradient of D's objective with respect to its parameters, and likewise for G. Lines 11 and 13 are meant to indicate taking a gradient step to update network parameters.

7.4.3 Learning with manifold interpolation (GAN-INT)

Deep networks have been shown to learn representations in which interpolations between embedding pairs tend to be near the data manifold [16, 118]. Motivated by this property, we can generate a large amount of additional text embeddings by simply interpolating between embeddings of training set captions. Critically, these interpolated text embeddings need not correspond to any actual human-written text, so there is no additional labeling cost. This can be viewed as adding a term to the generator objective:

$$\mathbb{E}_{t_1,t_2\sim p_{data}}[\log(1 - D(G(z, \beta t_1 + (1-\beta)t_2)))] \tag{7.5}$$

where z is drawn from the noise distribution and β interpolates between text embeddings t_1 and t_2. In practice we found that fixing $\beta = 0.5$ works well.

Because the interpolated embeddings are synthetic, D does not have "real" corresponding image and text pairs to train on. However, D learns to predict whether image and text pairs match or not. Thus, if D does a good job at this, then by satisfying D on interpolated text embeddings G can learn to fill in gaps on the data manifold in between training points. Note that t_1 and t_2 may come from different images and even different categories.[1]

7.4.4 Inverting the generator for style transfer

If the text encoding $\varphi(t)$ captures the image content (e.g. flower shape and colors), then in order to generate a realistic image the noise sample z should capture style factors such as background color and pose. With a trained GAN, one may wish to transfer the style of a query image onto the content of a particular text description. To achieve this, one can train a convolutional network to invert G to regress from samples $\hat{x} \leftarrow G(z, \varphi(t))$ back onto z. We used a simple squared loss to train the style encoder:

$$\mathcal{L}_{style} = \mathbb{E}_{t,z\sim\mathcal{N}(0,1)}||z - S(G(z, \varphi(t)))||_2^2 \tag{7.6}$$

[1] In our experiments, we used fine-grained categories (e.g. birds are similar enough to other birds, flowers to other flowers, etc.), and interpolating across categories did not pose a problem.

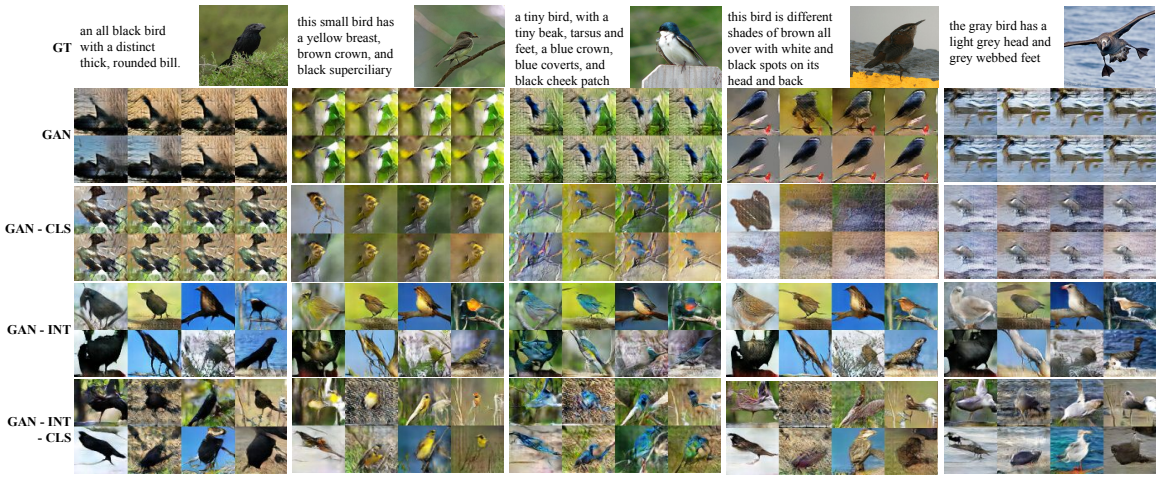


Figure 7.3: Zero-shot (i.e. conditioned on text from unseen test set categories) generated bird images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. We found that interpolation regularizer was needed to reliably achieve visually-plausible results.

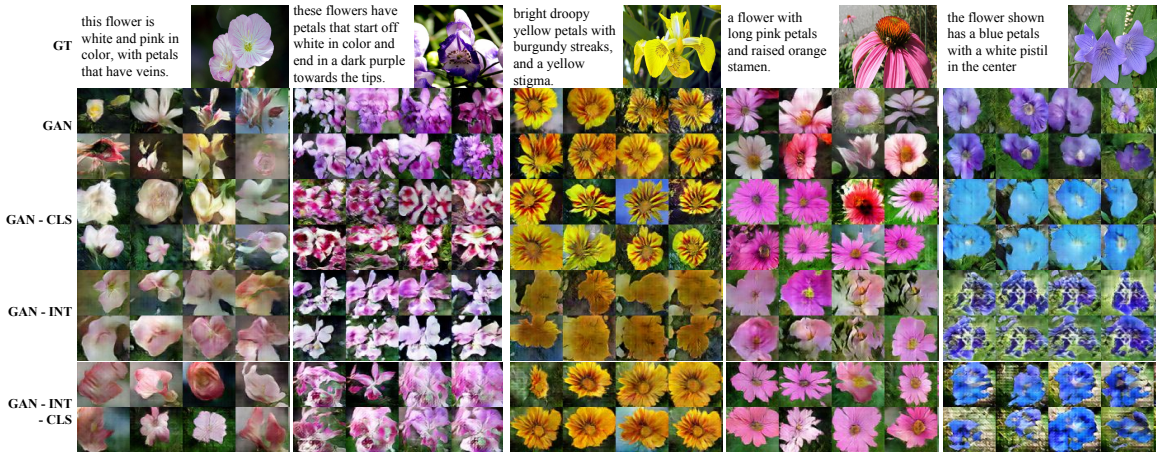


Figure 7.4: Zero-shot generated flower images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. All variants generated plausible images. Although some shapes of test categories were not seen during training (e.g. columns 3 and 4), color is preserved.

where S is the style encoder network. With a trained generator and style encoder, style transfer from a query image x onto text t proceeds as follows:

$$s \leftarrow S(x), \hat{x} \leftarrow G(s, \varphi(t))$$

where $\hat{x}$ is the result image and s is the predicted style.

7.5 Experiments

In this section we present results on the CUB and Oxford-102 datasets. CUB has 11,788 images of birds belonging to one of 200 different categories. The Oxford-102 contains 8,189 images of flowers from 102 different categories.

As in *Akata et al.* [3] and *Reed et al.* [120], we split these into class-disjoint training and test sets. CUB has 150 train+val classes and 50 test classes, while Oxford-102 has 82 train+val and 20 test classes. For both datasets, we used 5 captions per image. During mini-batch selection for training we randomly pick an image view (e.g. crop, flip) of the image and one of the captions.

For text features, we first pre-train a deep convolutional-recurrent text encoder on structured joint embedding of text captions with 1,024-dimensional GoogLeNet image embedings [148] as described in subsection 7.3.2. For both Oxford-102 and CUB we used a hybrid of character-level ConvNet with a recurrent neural network (char-CNN-RNN) as described in [120]. Note, however that pre-training the text encoder is not a requirement of our method and we include some end-to-end results in Appendix E.2. The reason for pre-training the text encoder was to increase the speed of training the other components for faster experimentation. We also provide some qualitative results obtained with MS COCO images of the validation set to show the generalizability of our approach.

We used the same GAN architecture for all datasets. The training image size was set to $64 \times 64 \times 3$. The text encoder produced $1,024$-dimensional embeddings that were projected to 128 dimensions in both the generator and discriminator before depth concatenation into convolutional feature maps.

As indicated in Algorithm 4, we take alternating steps of updating the generator and the discriminator network. We used the same base learning rate of 0.0002, and used the ADAM solver [7] with momentum 0.5. The generator noise was sampled from a 100-dimensional unit normal distribution. We used a minibatch size of 64 and trained for 600 epochs. Our

implementation was built on top of `dcgan.torch`[2].

7.5.1 Qualitative results

We compare the GAN baseline, our GAN-CLS with image-text matching discriminator (subsection 7.4.2), GAN-INT learned with text manifold interpolation (subsection 7.4.3) and GAN-INT-CLS which combines both.

Results on CUB can be seen in Figure 7.3. GAN and GAN-CLS get some color information right, but the images do not look real. However, GAN-INT and GAN-INT-CLS show plausible images that usually match all or at least part of the caption. We include additional analysis on the robustness of each GAN variant on the CUB dataset in Appendix E.1.

Results on the Oxford-102 Flowers dataset can be seen in Figure 7.4. In this case, all four methods can generate plausible flower images that match the description. The basic GAN tends to have the most variety in flower morphology (i.e. one can see very different petal types if this part is left unspecified by the caption), while other methods tend to generate more class-consistent images. We speculate that it is easier to generate flowers, perhaps because birds have stronger structural regularities across species that make it easier for D to spot a fake bird than to spot a fake flower.

Many additional results with GAN-INT and GAN-INT-CLS as well as GAN-E2E (our end-to-end GAN-INT-CLS without pre-training the text encoder $\varphi(t)$) for both CUB and Oxford-102 can be found in Appendix E.2.

7.5.2 Disentangling style and content

In this section we investigate the extent to which our model can separate style and content. By content, we mean the visual attributes of the bird itself, such as shape, size and color of each body part. By style, we mean all of the other factors of variation in the image such as background color and the pose orientation of the bird.

[2] https://github.com/soumith/dcgan.torch

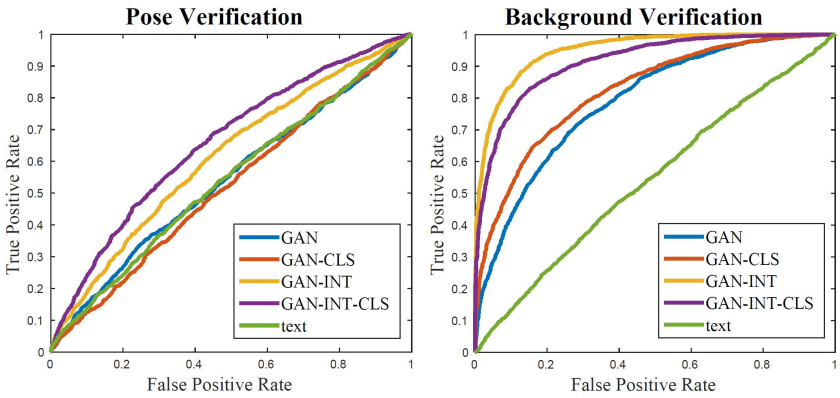


Figure 7.5: ROC curves using cosine distance between predicted style vector on same vs. different style image pairs. Left: image pairs reflect same or different pose. Right: image pairs reflect same or different average background color.

The text embedding mainly covers content and typically nothing about style, e.g. captions do not mention the background or pose. Therefore, in order to generate realistic images then GAN must learn to use noise sample z to account for style variations.

To quantify the degree of disentangling on CUB we set up two prediction tasks with noise z as the input: pose verification and background color verification. For each task, we first constructed similar and dissimilar pairs of images and then computed the predicted style vectors by feeding the image into a style encoder (trained to invert the input and output of generator). If GAN has disentangled style using z from image content, the similarity between images of the same style (e.g. similar pose) should be higher than that of different styles (e.g. different pose).

To recover z, we inverted the each generator network as described in subsection 7.4.4. To construct pairs for verification, we grouped images into 100 clusters using K-means where images from the same cluster share the same style. For background color, we clustered images by the average color (RGB channels) of the background; for bird pose, we clustered images by 6 keypoint coordinates (beak, belly, breast, crown, forehead, and tail).

For evaluation, we compute the actual predicted style variables by feeding pairs of images style encoders for GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. We verify the score using cosine similarity and report the AU-ROC (averaging over 5 folds). As a

baseline, we also compute cosine similarity between text features from our text encoder. We present results on Figure 7.5. As expected, captions alone are not informative for style prediction. Consistent with the qualitative results, we found that models incorporating interpolation regularizer (GAN-INT, GAN-INT-CLS) perform the best for this task. A t-SNE visualization of the extracted style features is shown in Figure E.6.

7.5.3 Pose and background style transfer

We demonstrate that GAN-INT-CLS with trained style encoder (subsection 7.4.4) can perform style transfer from an unseen query image onto a text description. Figure 7.6 shows that images generated using the inferred styles can accurately capture the pose information. In several cases the style transfer preserves detailed background information such as a tree branch upon which the bird is perched.

Disentangling the style by GAN-INT-CLS is interesting because it suggests a simple way of generalization. This way we can combine previously seen content (e.g. text) and previously seen styles, but in novel pairings so as to generate plausible images very different from any seen image during training. Another way to generalize is to use attributes that were previously seen (e.g. blue wings, yellow belly) as in the generated parakeet-like bird in the bottom row of Figure 7.6. This way of generalization takes advantage of text representations capturing multiple visual aspects.

7.5.4 Sentence interpolation

Figure 7.8 demonstrates the learned text manifold by interpolation (Left). Although there is no ground-truth text for the intervening points, the generated images appear plausible. Since we keep the noise distribution the same, the only changing factor within each row is the text embedding that we use. We observe that interpolations can accurately reflect color changes, e.g. from blue to red, while the pose and background are invariant.

As well as interpolating between two text encodings, we show results on Figure 7.8 (Right) with noise interpolation. Here, we sample two random noise vectors. By keeping

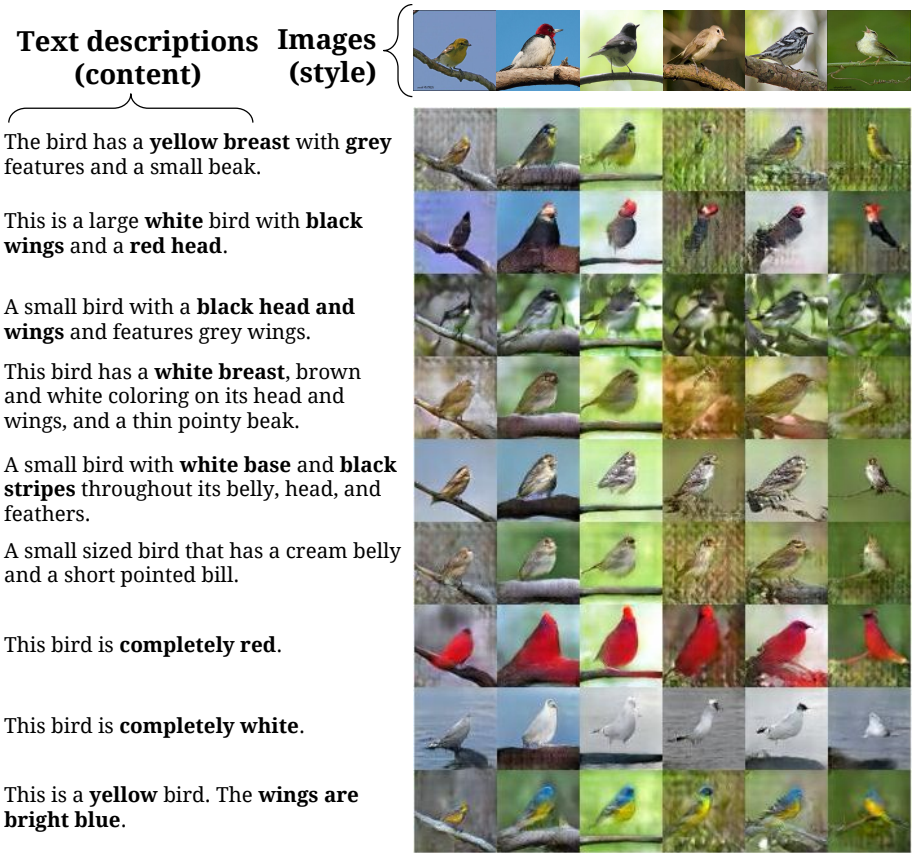


Figure 7.6: Transfering style from the top row (real) images to the content from the text, with G acting as a deterministic decoder. The bottom three captions are made up by us.

the text encoding fixed, we interpolate between these two noise vectors and generate bird images with a smooth transition between two styles by keeping the content fixed.

7.5.5 Beyond birds and flowers

We trained a GAN-CLS on MS-COCO to show the generalization capability of our approach on a general set of images that contain multiple objects and variable backgrounds. We use the same text encoder architecture, same GAN architecture and same hyperparameters (learning rate, minibatch size and number of epochs) as in CUB and Oxford-102. The only difference in training the text encoder is that COCO does not have a single object category per class. However, we can still learn an instance level (rather than category level) image and text matching function, as in [75].

Samples and ground truth captions and their corresponding images are shown on Fig-

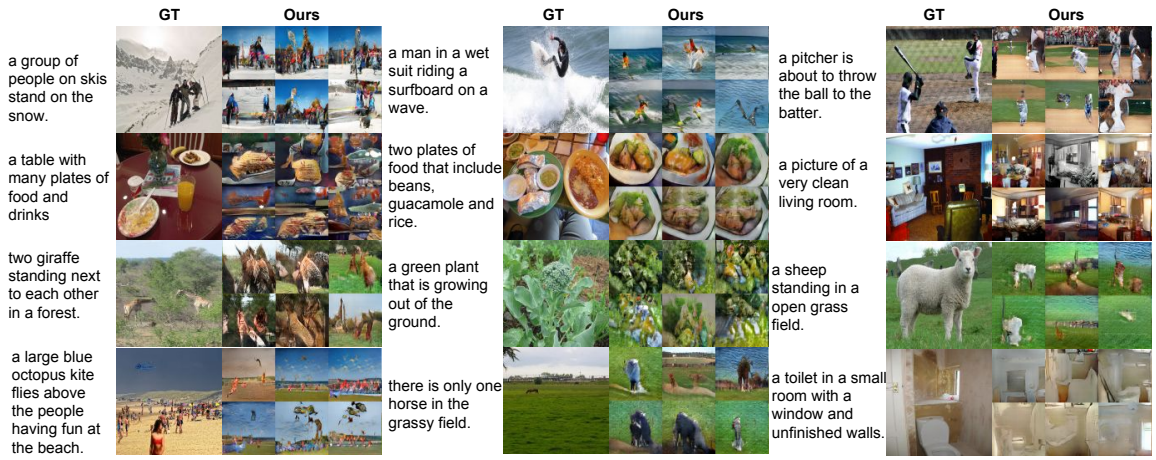


Figure 7.7: Generating images of general concepts using our GAN-CLS on the MS-COCO validation set. Unlike the case of CUB and Oxford-102, the network must (try to) handle multiple objects and diverse backgrounds.

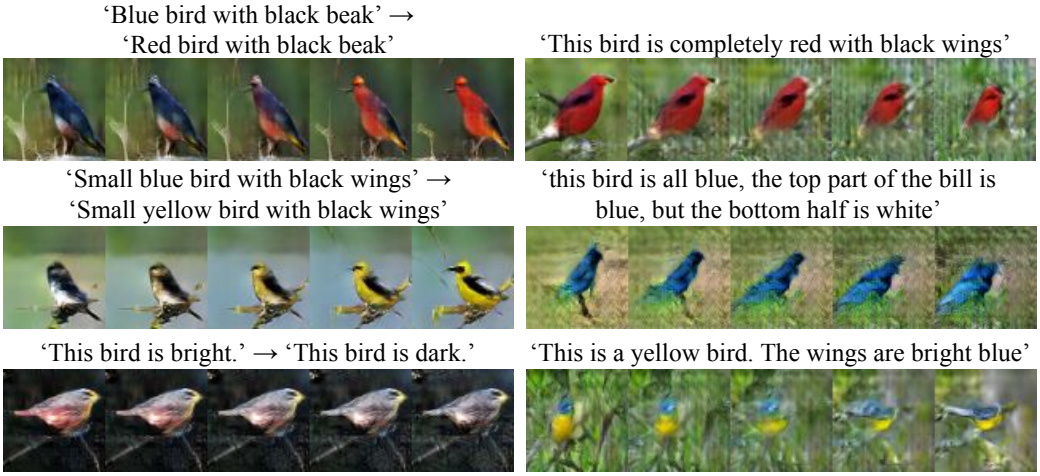


Figure 7.8: Left: Generated bird images by interpolating between two sentences (within a row the noise is fixed). Right: Interpolating between two randomly-sampled noise vectors.

ure 7.7. A common property of all the results is the sharpness of the samples, similar to other GAN-based image synthesis models. We also observe diversity in the samples by simply drawing multiple noise vectors and using the same fixed text encoding.

From a distance the results are encouraging, but upon close inspection it is clear that the generated scenes are not usually coherent; for example the human-like blobs in the baseball scenes lack clearly articulated parts. In future work, it may be interesting to incorporate hierarchical structure into the image synthesis model in order to better handle complex multi-object scenes.

A qualitative comparison with AlignDRAW [91] can be found in Appendix E.2. GAN-CLS generates sharper and higher-resolution samples that roughly correspond to the query, but AlignDRAW samples more noticably reflect single-word changes in the selected queries from that work. Incorporating temporal structure into the GAN-CLS generator network could potentially improve its ability to capture these text variations.

7.6 Conclusions

In this work we developed a simple and effective model for generating images based on detailed visual descriptions. We demonstrated that the model can synthesize many plausible visual interpretations of a given text caption. Our manifold interpolation regularizer substantially improved the text to image synthesis on CUB. We showed disentangling of style and content, and bird pose and background transfer from query images onto text descriptions. Finally we demonstrated the generalizability of our approach to generating images with multiple objects and variable backgrounds with our results on MS-COCO dataset.

CHAPTER VIII

Conclusion

In this thesis, I developed several deep network architectures endowed with new capabilities for solving AI tasks. Here I will summarize these contributions, discuss their limitations, and consider how to build on them in future work.

The disentangling restricted Boltzmann machine (disBM) was able to learn a feature representation separated into distinct components for facial expression, pose and shape. We showed that it could transfer one person's facial expression onto another while preserving apparent identity characteristics, and also that the learned features were highly discriminative for facial emotion recognition compared to contemporary methods. RBMs have since gone out of fashion for generative image modeling; replaced by frameworks incoporating a more flexible choice of function approximators, i.e. deep neural networks. However, the idea of generating images conditioned on an explicitly disentangled latent representation remains influential. In future work, regardless of the currently-fashionable image modeling paradigm, we need to solve the problem of discovering *what the latent factors* are, and also how to separate them with a minimal amount of supervision.

Our visual analogy making network learned to generate the pixels of an image D that solved the visual analogy problem $A : B :: C : D$. Although we only trained it on single-step analogies, we found that the learned analogy transformation could be applied repeatedly, enabling manifold traversal, e.g. repeated rotations of a 2D or 3D shape. The

obvious limitation here is that our model required many examples of analogies in order to learn to perform them on new images, and also that the model was purely deterministic. Sometimes, an analogy completion is only approximate, or there could be multiple correct choices for an analogy completion. In future work, it will be very interesting to explore a *probabilistic* model for analogy completion, allowing one to sample multiple possible images that make an analogy true.

The Neural Programmer-Interpreter was able to learn program embedding vectors with compositional structure by observing execution traces, and was able to learn a library of programs including sorting in a generalizable and data-efficient manner. However, NPI is limited by the strong supervision (execution traces) required to train the model. Furthermore, it assumes that the sequence of actions required to implement a program conditioned on an observation of the environment is deterministic; in reality there could be many action sequences that correctly implement the program. In future work, it will be important to relax these constraints. In particular, it could be promising to train the NPI in a reinforcement learning setting; i.e. provide a reward signal for correctly producing the output, rather than providing explicit execution traces.

Our text-to-image synthesis network can process an informal description of a bird or flower and generate diverse and compelling 64×64 image samples that fit the description. However, text-to-image is very far from solved. Upon close inspection of the images of birds, flowers and also general categories (MS-COCO), it becomes clear that the samples often do not have clearly-articulated parts. Furthermore, the generator network cannot justify its drawing choices; i.e. it cannot indicate that in a particular region it was *meant* to portray a beak. In future work, it will be crucial to incorporate semantically-meaningful structure into the generative process. This will both likely improve the quality of the results, and also provide additional control over the generated images.

These projects are small steps outward along the research frontier of (artificial) neural reasoning, planning and creativity. Human-like capabilities are still far out of reach. Fortu-

nately, data and computational resources have never been more abundant, opening the door to exploring more complex and exotic neural network architectures. Hopefully over time these research efforts will allow computers to be more human-like in their capabilities and as a result more useful for solving human problems.

APPENDICES

APPENDIX A

Derivation of variational approximation to disBM posterior inference

A.1 Derivation of disBM mean-field inference

Since we cannot directly compute the posterior $P(\mathbf{h}, \mathbf{m}|\mathbf{v})$, we choose an approximating factorized distribution $Q(\mathbf{h}, \mathbf{m}) = Q(\mathbf{h})Q(\mathbf{m})$, where $Q(\mathbf{h}) = \prod_k Q(h_k)$, $Q(h_k) \sim Bernoulli(\hat{h}_k)$ and similarly for $Q(\mathbf{m})$. We then choose the parameters $\hat{h}_k$ to minimize

$$KL(Q||P(\mathbf{h}, \mathbf{m}|\mathbf{v})) = \sum_{\mathbf{h},\mathbf{m}} Q(\mathbf{h})Q(\mathbf{m}) \log \frac{Q(\mathbf{h})Q(\mathbf{m})}{P(\mathbf{h}, \mathbf{m}|\mathbf{v})}$$

Note that

$$\begin{aligned}
&KL(Q||P(\mathbf{h}, \mathbf{m}|\mathbf{v})) = \\
&\quad = \sum_{\mathbf{h},\mathbf{m}} Q(\mathbf{h})Q(\mathbf{m}) \log(Q(\mathbf{h})Q(\mathbf{m})) \\
&\quad - \sum_{\mathbf{h},\mathbf{m}} Q(\mathbf{h})Q(\mathbf{m}) log\left(\frac{P(\mathbf{h}, \mathbf{m}, \mathbf{v})}{P(\mathbf{v})}\right) \\
&\quad = \mathbb{E}_{Q(\mathbf{h},\mathbf{m})}[\log Q(\mathbf{h}, \mathbf{m})] \\
&\quad - \sum_{\mathbf{h},\mathbf{m}} Q(\mathbf{h})Q(\mathbf{m}) \log P(\mathbf{h}, \mathbf{m}, \mathbf{v}) + \log P(\mathbf{v})
\end{aligned}$$

$$= -\mathcal{H}(Q) - \mathbb{E}_Q[\log \tilde{P}(\mathbf{h}, \mathbf{m}, \mathbf{v})] + \log Z + \log P(\mathbf{v})$$

Rearranging the terms, we see that

$$\begin{aligned} \log P(\mathbf{v}) &= KL(Q(\mathbf{h}, \mathbf{m}) || P(\mathbf{h}, \mathbf{m}|\mathbf{v})) \\ &\quad + \mathbb{E}_Q[\log \tilde{P}(\mathbf{h}, \mathbf{m}, \mathbf{v})] - \log Z + \mathcal{H}(Q) \\ &\geq \mathbb{E}_Q[\log \tilde{P}(\mathbf{h}, \mathbf{m}, \mathbf{v})] - \log Z + \mathcal{H}(Q) \end{aligned} \tag{A.1}$$

Therefore, minimizing this KL-divergence is equivalent to maximizing a lower bound on the log-likelihood of the data $\mathbf{v}$. $\tilde{P}(\mathbf{h}, \mathbf{m}, \mathbf{v})$ is the unnormalized joint distribution, i.e. $\exp(-E(\mathbf{h}, \mathbf{m}, \mathbf{v}))$, where

$$E(\mathbf{h}, \mathbf{m}, \mathbf{v}) = -\sum_{ijk} W_{ijk} v_i h_k m_j$$

is the energy function (ignoring bias units). To arrive at the mean-field update rule, we can differentiate the lower bound in equation A.1 with respect to the parameters $\hat{h}_k$ and $\hat{m}_j$. First note that $\mathbb{E}_Q[\log \tilde{P}(\mathbf{h}, \mathbf{m}, \mathbf{v})]$ and $\mathcal{H}(Q)$ can be expressed in terms of $\hat{h}_k$ and $\hat{m}_j$:

$$\begin{aligned} \mathbb{E}_Q[\log \tilde{P}(\mathbf{h}, \mathbf{m}, \mathbf{v})] &= \mathbb{E}_Q[\log(\exp(-E(\mathbf{h}, \mathbf{m}, \mathbf{v})))] \\ &= -\sum_{ijk} W_{ijk} v_i \mathbb{E}_Q[h_k] \mathbb{E}_Q[m_j] \\ &= -\sum_{ijk} W_{ijk} v_i \hat{h}_k \hat{m}_j \end{aligned}$$

$$\begin{aligned} \mathcal{H}(Q) &= \mathbb{E}_Q[\log Q(\mathbf{m}, \mathbf{h})] \\ &= \sum_k \mathbb{E}_Q[\log Q(h_k)] - \sum_j \mathbb{E}_Q[\log Q(m_j)] \\ &= \sum_k (\hat{h}_k \log \hat{h}_k + (1 - \hat{h}_k) \log(1 - \hat{h}_k)) \\ &\quad - \sum_j (\hat{m}_j \log \hat{m}_j + (1 - \hat{m}_j) \log(1 - \hat{m}_j)) \end{aligned}$$

Differentiating equation A.1 with respect to each parameter and and setting to zero, we get

$$\frac{\hat{h}_k}{1-\hat{h}_k} = \exp(-\sum_{ijk} W_{ijk} v_i \hat{m}_j)$$

$$\begin{aligned} \hat{h}_k &= \frac{1}{1+\exp(-\sum_{ijk} W_{ijk} v_i \hat{m}_j)} \\ &= \sigma(\sum_{ijk} W_{ijk} v_i \hat{m}_j) \end{aligned}$$

And similarly

$$\hat{m}_j = \sigma(\sum_{ijk} W_{ijk} v_i \hat{h}_k)$$

APPENDIX B

Additional examples of visual analogy-making

B.1 Additional examples of image analogy transformations

B.1.1 Trajectories of multiple shape analogies

Our model can apply analogies in both the forward and reversed direction. In Figure B.1, the first two columns indicate the operation, i.e., the relationship to be applied to the query. The first row demonstrates several steps of clockwise rotation, followed by counter-clockwise, returning to the initial orientation. The second and third rows show that our model can perform the same feat for scaling and translation. Although we only trained for 1-step analogies, the model is able to stay on the manifold even after repeated transformations.

We can also interleave different kinds of transformations by supplying multiple pairs of transformed images. Figure B.2 shows interleaving of rotation, translation and scaling in the same sequence.

Figure B.1 and Figure B.2 are also available in video format: `shape-*.avi`

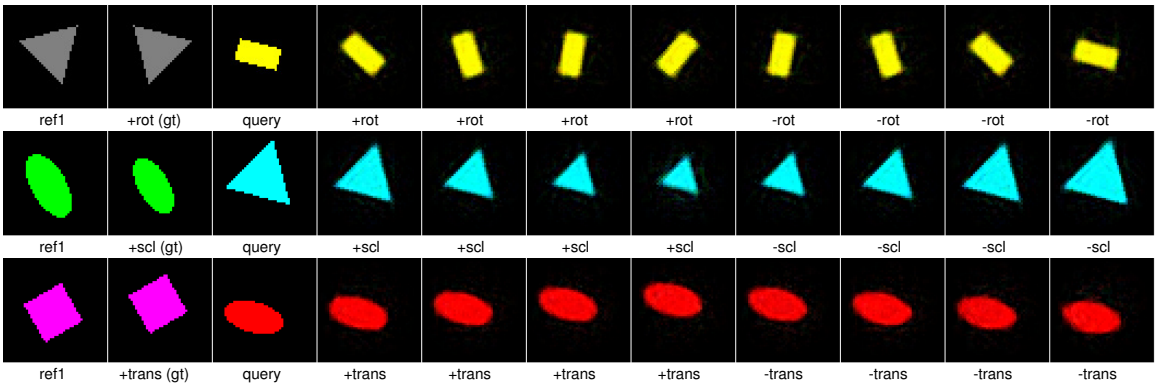


Figure B.1: Repeated application of analogies from the example pair (first two columns), in both forward and reverse mode, using a model trained with $\mathcal{L}_{deep}$.

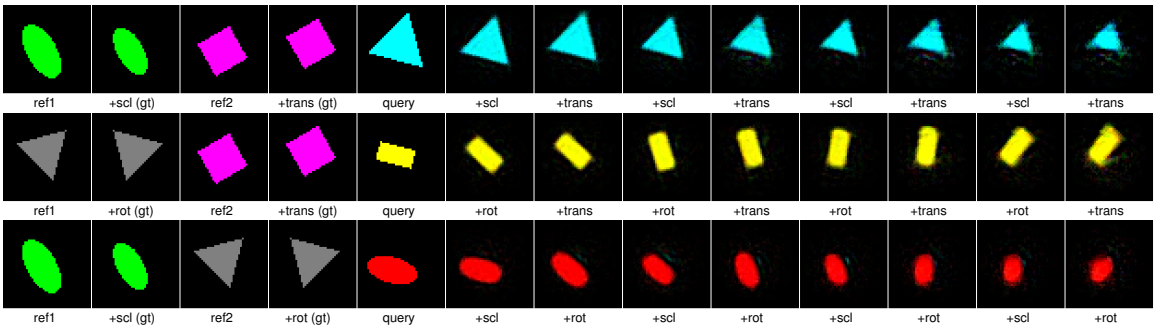


Figure B.2: Repeated application of multiple different analogies from two different example pairs (first four columns) using a model trained with $\mathcal{L}_{deep}$.

B.1.2 Comparing $\mathcal{L}_{add}$, $\mathcal{L}_{mul}$, and $\mathcal{L}_{deep}$ for shape analogies

Following the protocol in the previous section, we apply the models trained with $\mathcal{L}_{add}$, $\mathcal{L}_{mul}$, and $\mathcal{L}_{deep}$ for multi-step shape analogies. As is shown in Figure B.3, the model trained with $\mathcal{L}_{add}$ and $\mathcal{L}_{mul}$ cannot do even 1-step rotation, while the model trained with $\mathcal{L}_{deep}$ can support manifold traversal. Scaling and translation are relatively simple for $\mathcal{L}_{add}$ and $\mathcal{L}_{mul}$, but the qualitative degradation due to multi-step analogies is still noticeably more significant than that of $\mathcal{L}_{deep}$.

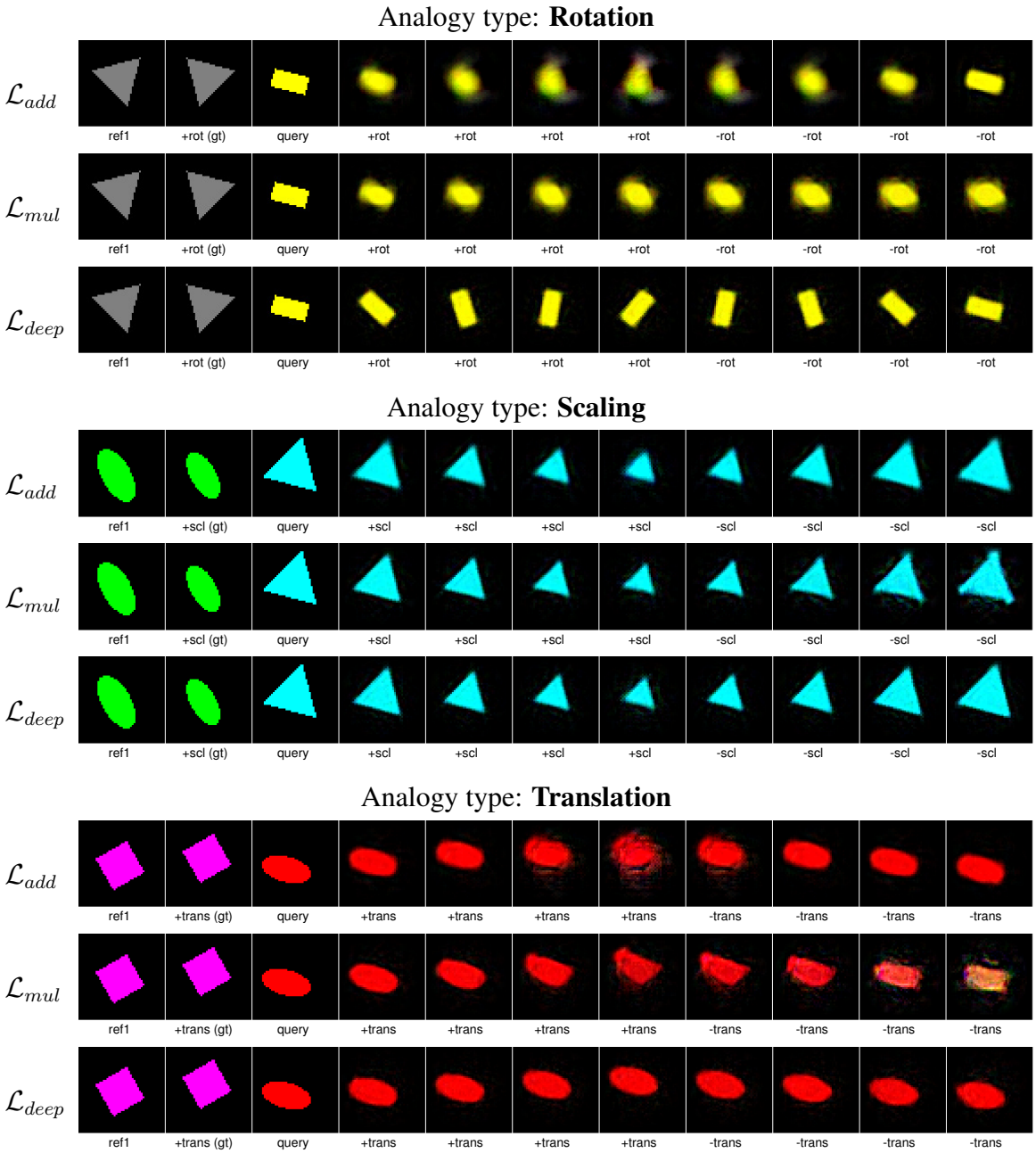


Figure B.3: Repeated application of analogies from the example pair (first two columns), in both forward and reverse mode, using three models trained respectively with $\mathcal{L}_{add}, \mathcal{L}_{mul}, \mathcal{L}_{deep}$.

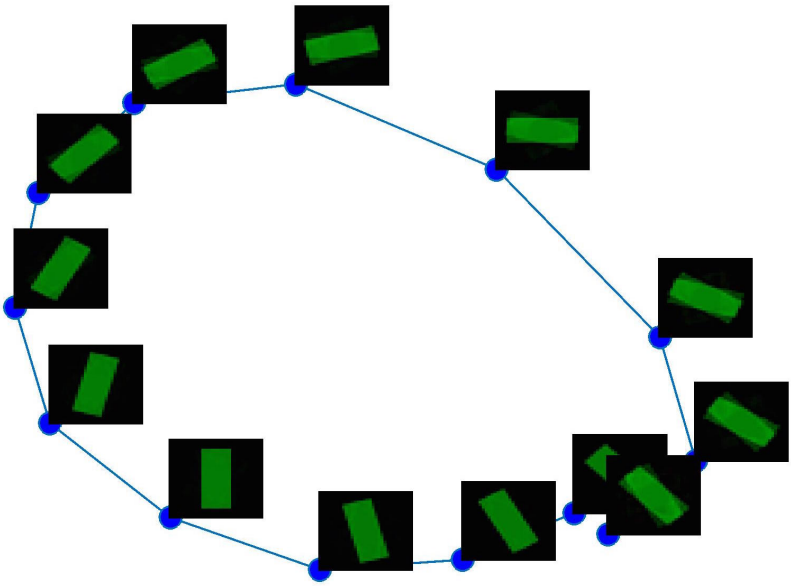


Figure B.4: 2D PCA projections of the image embeddings along the rotation manifold. Each point is marked with an image generated by the model, trained with $\mathcal{L}_{deep}$.

B.1.3 Fine-grained control over sprite attributes

In Figure B.5, we show how disentangling and attribute classification objectives can help with fine-grained control on the discrete-valued attributes of generated sprites.

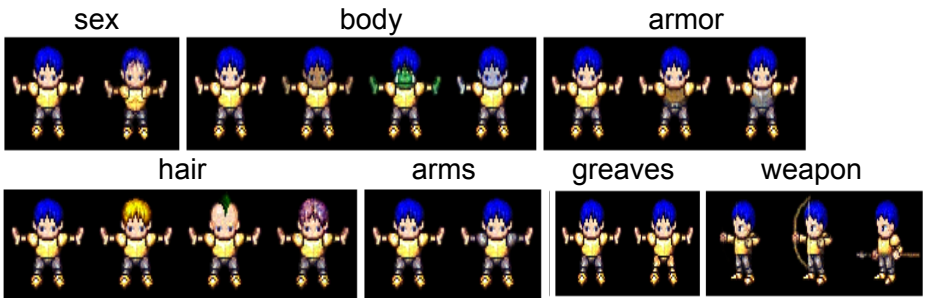


Figure B.5: Using $\mathcal{L}_{dis+cls}$, our model can generate sprites with fine-grained control over character attributes. The above images were generated by using f to encode the leftmost source image for each attribute, and then changing the identity units and re-rendering.

B.1.4 Animation transfer using disentangled features

When we have a model trained by $\mathcal{L}_{dis}$ or $\mathcal{L}_{dis+cls}$, we can extract disentangled identity and pose features for sprites. Performing pose transfer simply requires taking the pose of a reference image and the identity of a query image, and using the decoder g to project their combination back into the image space. In Figures B.6, B.7 and B.8, we show several consecutive frames of pose transfer, which we call animation transfer since we can follow the entire trajectory of an animation.

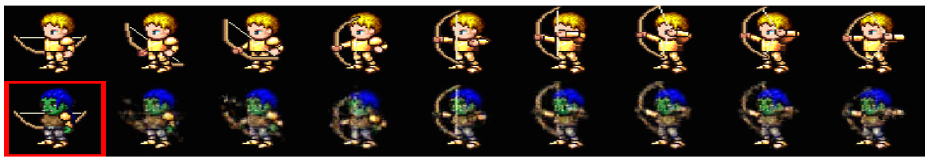


Figure B.6: Shooting a bow.

B.1.5 Sprite animation analogies with extrapolation

Below in Figure B.9, we show cross-identity animation extrapolations for each of the five animations, plus rotation. The analogy model has learned the structure of the animation

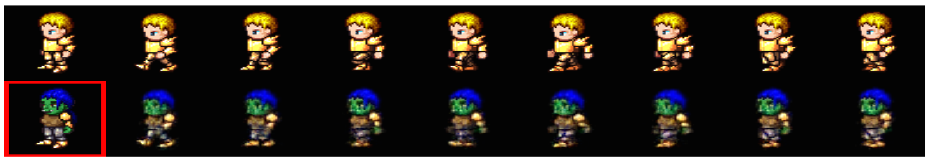


Figure B.7: Walking.

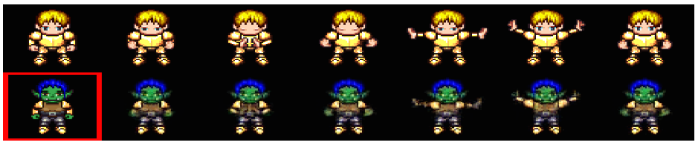


Figure B.8: Casting a spell.

manifolds across variations in viewpoint and character attributes, and is able to advance forward or backward in time based on the example image pair.

Figure B.9: Animation analogies and extrapolation for all character animations plus rotation. The example pair (first two columns) and query image (third column) both come from the test set of characters. $\mathcal{L}_{deep}$ was used for analogy training of pose units, jointly with $\mathcal{L}_{dis+cls}$ to learn a disentangled representation.

APPENDIX C

NPI program listing and sorting execution trace

C.1 Listing of learned programs

Below in Table C.1 we list the programs learned by our model:

C.2 Generated execution trace of BUBBLESORT

Figure C.1 shows the sequence of program calls for BUBBLESORT. Pointers 1 and 2

Figure C.1: Generated execution trace from our trained NPI sorting the array [9,2,5].

```
BUBBLESORT
  BUBBLE
    PTR 2 RIGHT
    BSTEP
      COMPSWAP
        SWAP 1 2
      RSHIFT
        PTR 1 RIGHT
        PTR 2 RIGHT
    BSTEP
      COMPSWAP
        SWAP 1 2
      RSHIFT
        PTR 1 RIGHT
        PTR 2 RIGHT
  RESET
    LSHIFT
      PTR 1 LEFT
      PTR 2 LEFT
    LSHIFT
      PTR 1 LEFT
      PTR 2 LEFT
    PTR 3 RIGHT
BUBBLE
  PTR 2 RIGHT
  BSTEP
    COMPSWAP

    RSHIFT
      PTR 1 RIGHT
      PTR 2 RIGHT
  BSTEP
    COMPSWAP

    RSHIFT
      PTR 1 RIGHT
      PTR 2 RIGHT
RESET
  LSHIFT
    PTR 1 LEFT
    PTR 2 LEFT
  LSHIFT
    PTR 1 LEFT
    PTR 2 LEFT
  PTR 3 RIGHT
BUBBLE
  PTR 2 RIGHT
  BSTEP
    COMPSWAP

    RSHIFT
      PTR 1 RIGHT
      PTR 2 RIGHT
  BSTEP
    COMPSWAP

    RSHIFT
      PTR 1 RIGHT
      PTR 2 RIGHT
RESET
  LSHIFT
    PTR 1 LEFT
    PTR 2 LEFT
  LSHIFT
    PTR 1 LEFT
    PTR 2 LEFT
  PTR 3 RIGHT
```

are used to implement the "bubble" operation involving the comparison and swapping of adjacent array elements. The third pointer (referred to in the trace as "PTR 3") is used to count the number of calls to BUBBLE. After every call to RESET the swapping pointers are

Program	Descriptions	Calls
ADD	Perform multi-digit addition	ADD1, LSHIFT
ADD1	Perform single-digit addition	ACT, CARRY
CARRY	Mark a 1 in the carry row one unit left	ACT
LSHIFT	Shift a specified pointer one step left	ACT
RSHIFT	Shift a specified pointer one step right	ACT
ACT	Move a pointer or write to the scratch pad	-
BUBBLESORT	Perform bubble sort (ascending order)	BUBBLE, RESET
BUBBLE	Perform one sweep of pointers left to right	ACT, BSTEP
RESET	Move both pointers all the way left	LSHIFT
BSTEP	Conditionally swap and advance pointers	COMPSWAP, RSHIFT
COMPSWAP	Conditionally swap two elements	ACT
LSHIFT	Shift a specified pointer one step left	ACT
RSHIFT	Shift a specified pointer one step right	ACT
ACT	Swap two values at pointer locations or move a pointer	-
GOTO	Change 3D car pose to match the target	HGOTO, VGOTO
HGOTO	Move horizontally to the target angle	LGOTO, RGOTO
LGOTO	Move left to match the target angle	ACT
RGOTO	Move right to match the target angle	ACT
VGOTO	Move vertically to the target elevation	UGOTO, DGOTO
UGOTO	Move up to match the target elevation	ACT
DGOTO	Move down to match the target elevation	ACT
ACT	Move camera 15° up, down, left or right	-
RJMP	Move all pointers to the rightmost posiiton	RSHIFT
MAX	Find maximum element of an array	BUBBLESORT,RJMP

Table C.1: Programs learned for addition, sorting and 3D car canonicalization. Note the ACT program has a different effect depending on the environment and on the passed-in arguments.

moved to the beginning of the array and the counting pointer is advanced by 1. When it has reached the end of the scratch pad, the model learns to halt execution of BUBBLESORT.

APPENDIX D

Zero-shot text-based retrieval

D.1 Additional retrieval results

APPENDIX E

Additional text-to-image results

E.1 Robustness of GAN variants

When training the baseline GAN on CUB, we found that simply choosing a different random seed (affecting network initialization and also minibatch selection) could yield results of dramatically varying quality. The classification and interpolation regularizers improved the robustness, and GAN-CLS-INT consistently yielded good results regardless of the random seed. To quantify this, we trained 10 instances each (varying only the random seed) of GAN, GAN-CLS, GAN-INT and GAN-CLS-INT on the 100 CUB training classes for 200 epochs. Using samples from each of these GAN models, we trained a zero-shot image classifier from scratch, following the same protocol described in section 5.5.

Figure E.1 shows the result. All variants perform better than the GAN baseline, and GAN-CLS-INT has the highest average performance and lowest variance. Our impression is that the classification (-CLS) and especially interpolation regularizer (-INT) stabilize the training, significantly reducing the incidence of "failed" GANs.

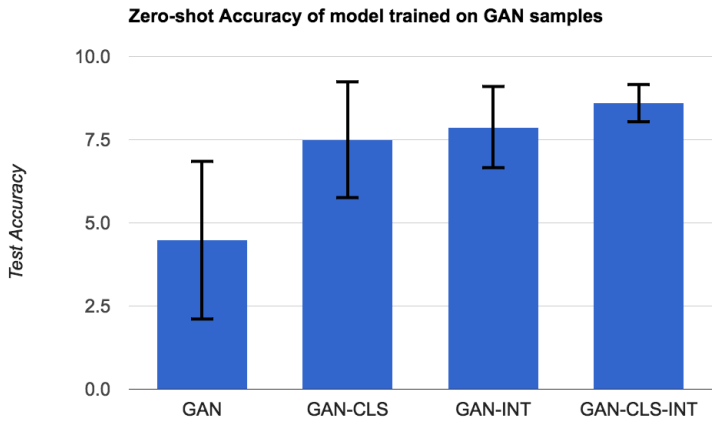


Figure E.1: Zero-shot accuracies for image encoders trained on GAN samples.

E.2 Additional text-to-image examples for birds, flowers and COCO

Many additional text-to-image examples are shown in Figures E.2, E.3 and E.4. We include samples from both GAN-INT-CLS that uses a pre-trained text encoder, as well as an "end-to-end" version in which the text encoder is trained from scratch jointly with the generator and discriminator networks. The GAN-INT-CLS has a slight advantage here because its text encoder could be trained by learning to align with image features extracted from high resolution (224×224) images using a state-of-the-art image encoder. The end-to-end-trained text encoder was trained using only 64×64, so it may not capture small details as accurately. However, as computing hardware becomes faster, it will become feasible to train end-to-end models on much higher resolution.

Comparisons to AlignDRAW are shown in Figure E.5. Our model generates higher-resolution and more detailed images, but AlignDraw is more noticably sensitive to single-word changes in the query text, e.g. color changes. This is likely because AlignDRAW uses a multi-step image generation procedure, in which each word updates the canvas. In future work, it may be beneficial to add temporal structure to our text-conditional GAN.

Figure E.2: Samples from GAN-INT-CLS (top) and end-to-end version (bottom).

Figure E.3: Samples from GAN-INT-CLS (top) and end-to-end version (bottom).

Figure E.4: Additional samples from our GAN-CLS trained on MS-COCO.

.

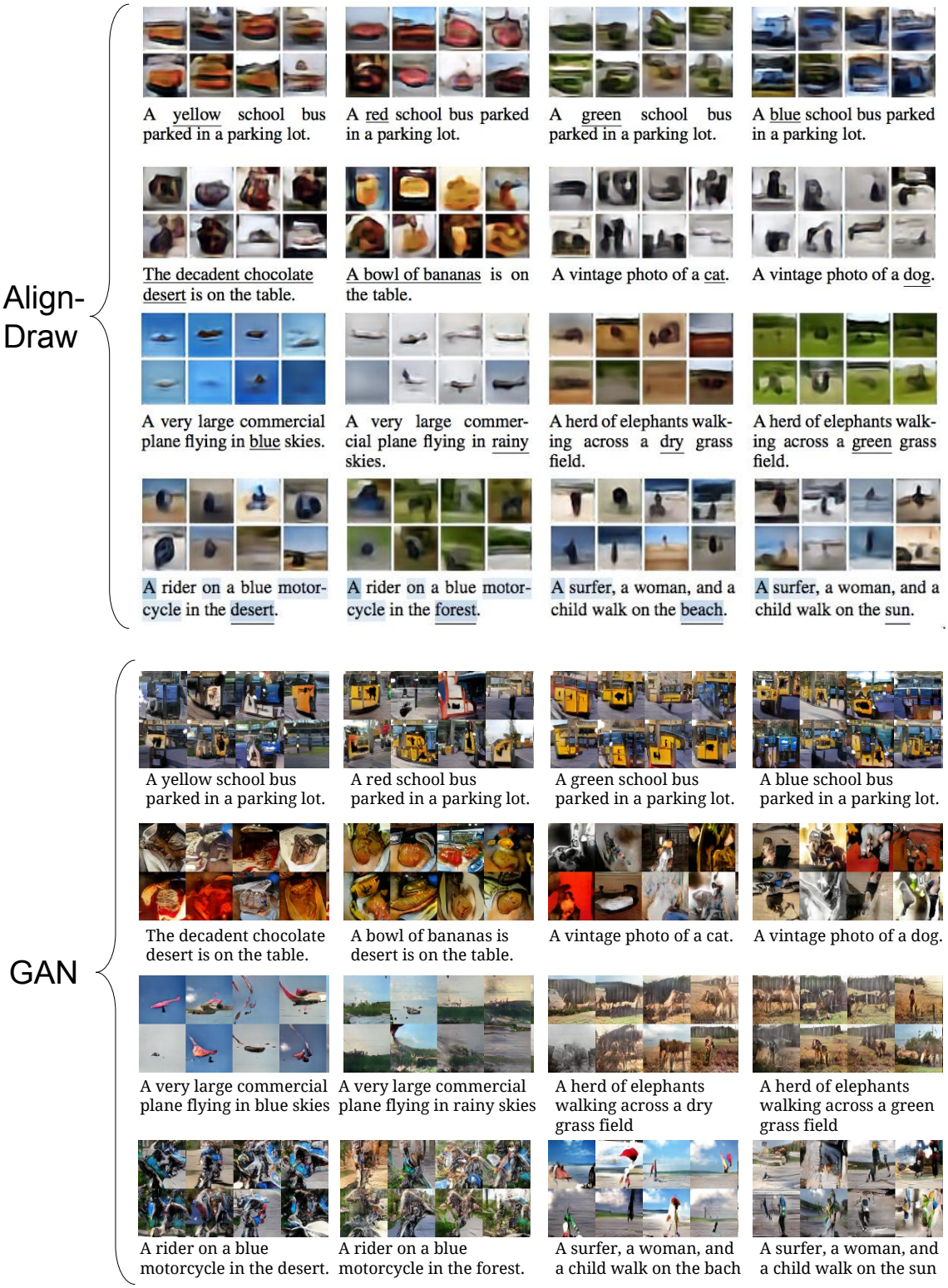


Figure E.5: Comparison to samples included in the AlignDraw paper [91]

.

Figure E.6: t-SNE embedding visualization of extract style features on CUB. It appears to be insensitive to the appearance of the birds (which should be captured by the text content), and mainly varies according to the primary background color.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] (), Liberated pixel cup, http://lpc.opengameart.org/, accessed: 2015-05-21.

[2] Akata, Z., F. Perronnin, Z. Harchaoui, and C. Schmid (2015), Label-embedding for image classification, *IEEE TPAMI*.

[3] Akata, Z., S. Reed, D. Walter, H. Lee, and B. Schiele (2015), Evaluation of Output Embeddings for Fine-Grained Image Classification, in *CVPR*.

[4] Anderson, M. L. (2010), Neural reuse: A fundamental organizational principle of the brain, *Behavioral and Brain Sciences*, *33*, 245–266.

[5] Andre, D., and S. J. Russell (2001), Programmable reinforcement learning agents, in *Advances in Neural Information Processing Systems*, pp. 1019–1025.

[6] Argyriou, A., T. Evgeniou, and M. Pontil (2007), Multi-task feature learning, in *NIPS*.

[7] Ba, J., and D. Kingma (2015), Adam: A method for stochastic optimization, in *ICLR*.

[8] Ba, J., K. Swersky, S. Fidler, and R. Salakhutdinov (2015), Predicting deep zero-shot convolutional neural networks using textual descriptions, in *ICCV*.

[9] Ba, J., K. Swersky, S. Fidler, and R. Salakhutdinov (2015), Predicting deep zero-shot convolutional neural networks using textual descriptions, *arXiv:1506.00511*.

[10] Banzhaf, W., P. Nordin, R. E. Keller, and F. D. Francone (1998), *Genetic programming: An introduction*, vol. 1, Morgan Kaufmann San Francisco.

[11] Bartha, P. (2013), Analogy and analogical reasoning, in *The Stanford Encyclopedia of Philosophy*, edited by E. N. Zalta, fall 2013 ed.

[12] Battaglia, P. W., J. B. Hamrick, and J. B. Tenenbaum (2013), Simulation as an engine of physical scene understanding, *Proceedings of the National Academy of Sciences*, *110*(45), 18,327–18,332.

[13] Bénard, P., F. Cole, M. Kass, I. Mordatch, J. Hegarty, M. S. Senn, K. Fleischer, D. Pesare, and K. Breeden (2013), Stylizing animation by example, *ACM Transactions on Graphics (TOG)*, *32*(4), 119.

[14] Bengio, S., J. Weston, and D. Grangier (2010), Label embedding trees for large multi-class tasks, in *NIPS*.

[15] Bengio, Y. (2009), Learning deep architectures for AI, *Foundations and Trends in Machine Learning*, *2*(1), 1–127.

[16] Bengio, Y., G. Mesnil, Y. Dauphin, and S. Rifai (2013), Better mixing via deep representations, in *ICML*.

[17] Caruana, R. (1997), Multitask learning, *Machine Learning*, *28*(1), 41–75.

[18] Cheung, B., J. A. Livezey, A. K. Bansal, and B. A. Olshausen (2015), Discovering hidden factors of variation in deep networks, in *ICLR Workshop*.

[19] Cleeremans, A., D. Servan-Schreiber, and J. L. McClelland (1989), Finite state automata and simple recurrent networks, *Neural computation*, *1*(3), 372–381.

[20] Coates, A., and A. Y. Ng (2011), The importance of encoding versus training with sparse coding and vector quantization, in *ICML*.

[21] Cohen, T., and M. Welling (2014), Learning the irreducible representations of commutative lie groups, in *ICML*, pp. 1755–1763.

[22] Cohen, T. S., and M. Welling (2015), Transformation properties of learned visual representations, in *ICLR*.

[23] Courville, A., J. Bergstra, and Y. Bengio (2011), A spike and slab restricted Boltzmann machine, in *AISTATS*.

[24] Dalton, J., J. Allan, and P. Mirajkar (2013), Zero-shot video retrieval using content and concepts, in *CIKM*.

[25] Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009), Imagenet: A large-scale hierarchical image database, in *CVPR*.

[26] Deng, J., J. Krause, and L. Fei-Fei (2013), Fine-grained crowdsourcing for fine-grained recognition, in *CVPR*.

[27] Denton, E. L., S. Chintala, R. Fergus, et al. (2015), Deep generative image models using a laplacian pyramid of adversarial networks, in *NIPS*.

[28] Desjardins, G., A. Courville, and Y. Bengio (2012), Disentangling factors of variation via generative entangling, *arXiv preprint arXiv:1210.5474*.

[29] Dietterich, T. G. (2000), Hierarchical reinforcement learning with the MAXQ value function decomposition, *Journal of Artificial Intelligence Research*, *13*, 227–303.

[30] Ding, W., and G. W. Taylor (2014), " mental rotation" by optimizing transforming distance, *arXiv preprint arXiv:1406.3010*.

[31] Dollár, P., V. Rabaud, and S. Belongie (2007), Learning to traverse image manifolds, *Advances in neural information processing systems*, *19*, 361.

[32] Donahue, J., Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell (2013), Decaf: A deep convolutional activation feature for generic visual recognition, *arXiv:1310.1531*.

[33] Donahue, J., L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell (2015), Long-term recurrent convolutional networks for visual recognition and description, in *CVPR*.

[34] Donnarumma, F., R. Prevete, and G. Trautteur (2012), Programming in the brain: A neural network theoretical framework, *Connection Science*, *24*(2-3), 71–90.

[35] Donnarumma, F., R. Prevete, F. Chersi, and G. Pezzulo (2015), A programmer-interpreter neural network architecture for prefrontal cognitive control, *International Journal of Neural Systems*, *25*(6), 1550,017.

[36] Dosovitskiy, A., J. Springenberg, and T. Brox (2015), Learning to generate chairs with convolutional neural networks, in *CVPR*.

[37] Duan, K., D. Parikh, D. J. Crandall, and K. Grauman (2012), Discovering localized attributes for fine-grained recognition, in *CVPR*.

[38] Elhoseiny, M., B. Saleh, and A. Elgammal (2013), Write a classifier: Zero-shot learning using purely textual descriptions, in *ICCV*.

[39] Evans, T. G. (1964), A heuristic program to solve geometric-analogy problems, in *Proceedings of the April 21-23, 1964, spring joint computer conference*, ACM.

[40] Farhadi, A., I. Endres, D. Hoiem, and D. Forsyth (2009), Describing objects by their attributes, in *CVPR*.

[41] Fidler, S., S. Dickinson, and R. Urtasun (2012), 3D object detection and viewpoint estimation with a deformable 3D cuboid model, in *Advances in neural information processing systems*.

[42] Fidler, S., S. Dickinson, and R. Urtasun (2012), 3d object detection and viewpoint estimation with a deformable 3d cuboid model, in *NIPS*, pp. 611–619.

[43] Frome, A., G. S. Corrado, J. Shlens, S. Bengio, J. Dean, and T. Mikolov (2013), Devise: A deep visual-semantic embedding model, in *NIPS*.

[44] Fu, Y., T. M. Hospedales, T. Xiang, Z. Fu, and S. Gong (2014), Transductive multi-view embedding for zero-shot recognition and annotation, in *ECCV*.

[45] Fu, Y., T. M. Hospedales, T. Xiang, and S. Gong (2015), Transductive multi-view zero-shot learning, *TPAMI*.

[46] Gauthier, J. (2015), Conditional generative adversarial nets for convolutional face generation, *Tech. rep.*

[47] Giles, C. L., C. B. Miller, D. Chen, H.-H. Chen, G.-Z. Sun, and Y.-C. Lee (1992), Learning and extracting finite state automata with second-order recurrent neural networks, *Neural Computation*, *4*(3), 393–405.

[48] Goodfellow, I., M. Mirza, A. Courville, and Y. Bengio (2013), Multi-prediction deep Boltzmann machines, in *NIPS*.

[49] Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014), Generative adversarial nets, in *NIPS*.

[50] Graves, A., G. Wayne, and I. Danihelka (2014), Neural Turing machines, *arXiv preprint arXiv:1410.5401*.

[51] Gregor, K., I. Danihelka, A. Graves, D. Rezende, and D. Wierstra (2015), Draw: A recurrent neural network for image generation, in *ICML*.

[52] Gross, R., I. Matthews, J. Cohn, T. Kanade, and S. Baker (2010), Multi-PIE, *Image and Vision Computing*, *28*(5).

[53] Habibian, A., T. Mensink, and C. G. Snoek (2014), Composite concept discovery for zero-shot video event detection, in *Proceedings of International Conference on Multimedia Retrieval*.

[54] Hadsell, R., S. Chopra, and Y. LeCun (2006), Dimensionality reduction by learning an invariant mapping, in *CVPR*.

[55] Harris, Z. (1954), Distributional structure, *Word*, *10*(23).

[56] Hertzmann, A., C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin (2001), Image analogies, in *SIGGRAPH*.

[57] Hertzmann, A., C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin (2001), Image analogies, in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 327–340, ACM.

[58] Hinton, G. E. (2002), Training products of experts by minimizing contrastive divergence, *Neural Computation*, *14*(8), 1771–1800.

[59] Hinton, G. E. (2010), A practical guide to training restricted boltzmann machines, *Tech. rep.*

[60] Hinton, G. E., and R. Salakhutdinov (2006), Reducing the dimensionality of data with neural networks, *Science*, *313*(5786), 504–507.

[61] Hochreiter, S., and J. Schmidhuber (1997), Long short-term memory, *Neural Comput.*, *9*(8), 1735–1780.

[62] Huang, G. B., H. Lee, and E. Learned-Miller (2012), Learning hierarchical representations for face verification with convolutional deep belief networks, in *CVPR*.

[63] Huang, G. B., M. Mattar, H. Lee, and E. Learned-Miller (2012), Learning to align from scratch, in *NIPS*.

[64] Huang, S., M. Elhoseiny, A. Elgammal, and D. Yang (2015), Learning hypergraph-regularized attribute predictors, in *CVPR*.

[65] Hwang, S. J., K. Grauman, and F. Sha (2013), Analogy-preserving semantic embedding for visual object categorization, in *NIPS*.

[66] Ioffe, S., and C. Szegedy (2015), Batch normalization: Accelerating deep network training by reducing internal covariate shift, *arXiv:1502.03167*.

[67] Jia, Y., E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell (2014), Caffe: Convolutional architecture for fast feature embedding, *arXiv preprint arXiv:1408.5093*.

[68] Joulin, A., and T. Mikolov (2015), Inferring algorithmic patterns with stack-augmented recurrent nets, in *NIPS*.

[69] Kaiser, Ł., and I. Sutskever (2015), Neural gpus learn algorithms, *arXiv preprint arXiv:1511.08228*.

[70] Karpathy, A., and F. Li (2015), Deep visual-semantic alignments for generating image descriptions, in *CVPR*.

[71] Kavukcuoglu, K., M. Ranzato, R. Fergus, and Y. LeCun (2009), Learning invariant features through topographic filter maps, in *CVPR*.

[72] Kim, G., S. Moon, and L. Sigal (2015), Ranking and retrieval of image sequences from multiple paragraph queries, in *CVPR*.

[73] Kingma, D. P., and M. Welling (2013), Auto-encoding variational bayes, *arXiv preprint arXiv:1312.6114*.

[74] Kingma, D. P., S. Mohamed, D. J. Rezende, and M. Welling (2014), Semi-supervised learning with deep generative models, in *NIPS*, pp. 3581–3589.

[75] Kiros, R., R. Salakhutdinov, and R. S. Zemel (2014), Unifying visual-semantic embeddings with multimodal neural language models, in *ACL*.

[76] Kolter, Z., P. Abbeel, and A. Y. Ng (2008), Hierarchical apprenticeship learning with application to quadruped locomotion, in *Advances in Neural Information Processing Systems*, pp. 769–776.

[77] Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012), Imagenet classification with deep convolutional neural networks, in *NIPS*.

[78] Kulkarni, G., V. Premraj, S. Dhar, S. Li, Y. choi, A. Berg, and T. Berg (2011), Baby talk: understanding and generating simple image descriptions, in *CVPR*.

[79] Kulkarni, T. D., W. Whitney, P. Kohli, and J. B. Tenenbaum (2015), Deep convolutional inverse graphics network, *arXiv preprint arXiv:1503.03167*.

[80] Kumar, N., A. C. Berg, P. N. Belhumeur, and S. K. Nayar (2009), Attribute and simile classifiers for face verification, in *ICCV*.

[81] Kurach, K., M. Andrychowicz, and I. Sutskever (2015), Neural random-access machines, *arXiv preprint arXiv:1511.06392*.

[82] Lampert, C., H. Nickisch, and S. Harmeling (2013), Attribute-based classification for zero-shot visual object categorization, in *TPAMI*.

[83] Le, Q. V., W. Y. Zou, S. Y. Yeung, and A. Y. Ng (2011), Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis, in *CVPR*.

[84] Lee, H., C. Ekanadham, and A. Y. Ng (2008), Sparse deep belief net model for visual area V2, in *NIPS*.

[85] Lee, H., R. Grosse, R. Ranganath, and A. Y. Ng (2011), Unsupervised learning of hierarchical representations with convolutional deep belief networks, *Communications of the ACM*, *54*(10), 95–103.

[86] Lerer, A., S. Gross, and R. Fergus (2016), Learning physical intuition of block towers by example, *arXiv preprint arXiv:1603.01312*.

[87] Levy, O., Y. Goldberg, and I. Ramat-Gan (2014), Linguistic regularities in sparse and explicit word representations, *CoNLL-2014*, p. 171.

[88] Lin, T.-Y., M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick (2014), Microsoft coco: Common objects in context, in *Computer Vision–ECCV 2014*, pp. 740–755, Springer.

[89] Liu, Z., P. Luo, X. Wang, and X. Tang (2015), Deep learning face attributes in the wild, in *ICCV*.

[90] Lowe, D. G. (1999), Object recognition from local scale-invariant features, in *CVPR*.

[91] Mansimov, E., E. Parisotto, J. L. Ba, and R. Salakhutdinov (2016), Generating images from captions with attention, *ICLR*.

[92] Mao, J., W. Xu, Y. Yang, J. Wang, and A. Yuille (2014), Deep captioning with multimodal recurrent neural networks (m-rnn), *arXiv:1412.6632*.

[93] Mccloskey, M., and N. J. Cohen (1989), Catastrophic interference in connectionist networks: The sequential learning problem, in *The psychology of learning and motivation*, vol. 24, pp. 109–165.

[94] Memisevic, R., and G. E. Hinton (2010), Learning to represent spatial transformations with factored higher-order Boltzmann machines, *Neural Computation*, *22*(6), 1473–1492.

[95] Metz, C. (2015), Facebooks ai can caption photos for the blind on its own, [Online; posted 27-October-2015].

[96] Michalski, V., R. Memisevic, and K. Konda (2014), Modeling deep temporal dependencies with recurrent grammar cells"", in *Advances in neural information processing systems*, pp. 1925–1933.

[97] Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013), Distributed representations of words and phrases and their compositionality, in *NIPS*.

[98] Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013), Distributed representations of words and phrases and their compositionality, in *NIPS*, pp. 3111–3119.

[99] Miller, G. A. (1995), Wordnet: a lexical database for english, *CACM*, *38*.

[100] Minsky, M. (1982), Semantic information processing.

[101] Mirza, M., and S. Osindero (2014), Conditional generative adversarial nets, *arXiv preprint arXiv:1411.1784*.

[102] Mou, L., G. Li, Y. Liu, H. Peng, Z. Jin, Y. Xu, and L. Zhang (2014), Building program vector representations for deep learning, *arXiv preprint arXiv:1409.3358*.

[103] Neelakantan, A., Q. V. Le, and I. Sutskever (2015), Neural programmer: Inducing latent programs with gradient descent, *arXiv preprint arXiv:1511.04834*.

[104] Ng, J. Y.-H., M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici (2015), Beyond short snippets: Deep networks for video classification, in *CVPR*.

[105] Ngiam, J., A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng (2011), Multimodal deep learning, in *ICML*.

[106] Nilsback, M.-E., and A. Zisserman (2008), Automated flower classification over a large number of classes, in *ICCVGI*.

[107] Norouzi, M., T. Mikolov, S. Bengio, Y. Singer, J. Shlens, A. Frome, G. Corrado, and J. Dean (2013), Zero-shot learning by convex combination of semantic embeddings, *arXiv:1312.5650*.

[108] Oquab, M., L. Bottou, I. Laptev, and J. Sivic (2014), Learning and transferring mid-level image representations using convolutional neural networks, in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 1717–1724, IEEE.

[109] Ordonez, V., G. Kulkarni, and T. Berg (2011), Im2Text: Describing images using 1 million captioned photographs, in *NIPS*.

[110] OReilly, R. C., R. Bhattacharyya, M. D. Howard, and N. Ketz (2014), Complementary learning systems, *Cognitive Science*, *38*(6), 1229–1248.

[111] Palatucci, M., D. Pomerleau, G. Hinton, and T. Mitchell (2009), Zero-shot learning with semantic output codes, in *NIPS*.

[112] Parikh, D., and K. Grauman (2011), Relative attributes, in *ICCV*.

[113] Pennington, J., R. Socher, and C. D. Manning (2014), Glove: Global vectors for word representation, in *EMNLP*.

[114] Pennington, J., R. Socher, and C. D. Manning (2014), Glove: Global vectors for word representation, *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, *12*.

[115] Radford, A., L. Metz, and S. Chintala (2015), Unsupervised representation learning with deep convolutional generative adversarial networks, *arXiv preprint arXiv:1511.06434*.

[116] Ranzato, M., F. J. Huang, Y. L. Boureau, and Y. LeCun (2007), Unsupervised learning of invariant feature hierarchies with applications to object recognition, in *CVPR*.

[117] Ranzato, M., J. Susskind, V. Mnih, and G. E. Hinton (2011), On deep generative models with applications to recognition, in *CVPR*.

[118] Reed, S., K. Sohn, Y. Zhang, and H. Lee (2014), Learning to disentangle factors of variation with manifold interaction, in *ICML*.

[119] Reed, S., Y. Zhang, Y. Zhang, and H. Lee (2015), Deep visual analogy-making, in *NIPS*.

[120] Reed, S., Z. Akata, H. Lee, and B. Schiele (2016), Learning deep representations of fine-grained visual descriptions, in *CVPR*.

[121] Ren, M., R. Kiros, and R. Zemel (2015), Exploring models and data for image question answering, in *NIPS*.

[122] Rifai, S., P. Vincent, X. Muller, X. Glorot, and Y. Bengio (2011), Contractive auto-encoders: Explicit invariance during feature extraction, in *ICML*.

[123] Rifai, S., Y. Bengio, A. Courville, P. Vincent, and M. Mirza (2012), Disentangling factors of variation for facial expression recognition, in *Computer Vision–ECCV 2012*, pp. 808–822, Springer.

[124] Rohrbach, M., M. Stark, and B.Schiele (2011), Evaluating knowledge transfer and zero-shot learning in a large-scale setting, in *CVPR*.

[125] Rothkopf, C., and D. Ballard (2013), Modular inverse reinforcement learning for visuomotor behavior, *Biological Cybernetics*, *107*(4), 477–490.

[126] Roweis, S. T., and L. K. Saul (2000), Nonlinear dimensionality reduction by locally linear embedding, *Science*, *290*(5500), 2323–2326.

[127] Rumelhart, D. E., G. E. Hinton, and J. L. McClelland (1986), Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1, chap. A General Framework for Parallel Distributed Processing, pp. 45–76, MIT Press.

[128] Russakovsky, O., et al. (2015), ImageNet Large Scale Visual Recognition Challenge, *International Journal of Computer Vision (IJCV)*, pp. 1–42, doi:10.1007/s11263-015-0816-y.

[129] Russell, S. J., P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards (2003), *Artificial intelligence: a modern approach*, vol. 2, Prentice hall Upper Saddle River.

[130] Salakhutdinov, R., and G. E. Hinton (2009), Deep Boltzmann machines, in *AISTATS*.

[131] Schaul, T., D. Horgan, K. Gregor, and D. Silver (2015), Universal value function approximators, in *International Conference on Machine Learning*.

[132] Schmidhuber, J. (1992), Learning to control fast-weight memories: An alternative to dynamic recurrent networks, *Neural Computation*, *4*(1), 131–139.

[133] Schneider, W., and J. M. Chein (2003), Controlled and automatic processing: behavior, theory, and biological mechanisms, *Cognitive Science*, *27*(3), 525–559.

[134] Simonyan, K., and A. Zisserman (2015), Very deep convolutional networks for large-scale image recognition, in *International Conference on Learning Representations*.

[135] Smolensky, P. (1986), Information processing in dynamical systems: Foundations of harmony theory.

[136] Socher, R., M. Ganjoo, H. Sridhar, O. Bastani, C. Manning, and A. Ng (2013), Zero-shot learning through cross-modal transfer, in *NIPS*.

[137] Sohn, K., and H. Lee (2012), Learning invariant representations with local transformations, in *ICML*.

[138] Sohn, K., G. Zhou, C. Lee, and H. Lee (2013), Learning and selecting features jointly with point-wise gated Boltzmann machines, in *ICML*.

[139] Sohn, K., W. Shang, and H. Lee (2014), Improved multimodal deep learning with variation of information, in *NIPS*.

[140] Srivastava, N., and R. Salakhutdinov (2012), Multimodal learning with Deep Boltzmann machines, in *NIPS*.

[141] Stoyanov, V., A. Ropson, and J. Eisner (2011), Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure, in *AISTATS*.

[142] Subramanian, K., C. Isbell, and A. Thomaz (2011), Learning options through human interaction, in *IJCAI Workshop on Agents Learning Interactively from Human Teachers*.

[143] Susskind, J., A. Anderson, and G. E. Hinton (2010), The Toronto Face Database, *Tech. rep.*, University of Toronto.

[144] Susskind, J., R. Memisevic, G. E. Hinton, and M. Pollefeys (2011), Modeling the joint density of two images under a variety of transformations, in *CVPR*.

[145] Sutskever, I., and G. E. Hinton (2009), Using matrices to model symbolic relationship, in *Advances in Neural Information Processing Systems*, pp. 1593–1600.

[146] Sutskever, I., O. Vinyals, and Q. V. Le (2014), Sequence to sequence learning with neural networks, in *Advances in neural information processing systems*, pp. 3104–3112.

[147] Sutton, R. S., D. Precup, and S. Singh (1999), Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, *Artificial Intelligence*, *112*(1-2), 181–211.

[148] Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015), Going deeper with convolutions, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9.

[149] Tang, Y., R. Salakhutdinov, and G. Hinton (2013), Tensor analyzers, in *ICML*, pp. 163–171.

[150] Tang, Y., R. Salakhutdinov, and G. E. Hinton (2013), Tensor analyzers, in *ICML*.

[151] Tenenbaum, J. B., and W. T. Freeman (2000), Separating style and content with bilinear models, *Neural computation*, *12*(6), 1247–1283.

[152] Tenenbaum, J. B., V. De Silva, and J. C. Langford (2000), A global geometric framework for nonlinear dimensionality reduction, *Science*, *290*(5500), 2319–2323.

[153] Tieleman, T. (2008), Training restricted Boltzmann machines using approximations to the likelihood gradient, in *ICML*.

[154] Tsarev, F., and K. Egorov (2011), Finite state machine induction using genetic algorithm based on testing and model checking, in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pp. 759–762, ACM.

[155] Turney, P. D. (2006), Similarity of semantic relations, *Computational Linguistics*, *32*(3), 379–416.

[156] Vinyals, O., M. Fortunato, and N. Jaitly (2015), Pointer networks, *Advances in Neural Information Processing Systems (NIPS)*.

[157] Vinyals, O., A. Toshev, S. Bengio, and D. Erhan (2015), Show and tell: A neural image caption generator, in *CVPR*.

[158] Wah, C., S. Branson, P. Welinder, P. Perona, and S. Belongie (2011), The caltech-ucsd birds-200-2011 dataset.

[159] Wang, P., Q. Wu, C. Shen, A. v. d. Hengel, and A. Dick (2015), Explicit knowledge-based reasoning for visual question answering, *arXiv preprint arXiv:1511.02570*.

[160] Watrous, R. L., and G. M. Kuhn (1992), Induction of finite-state languages using second-order recurrent networks, *Neural Computation*, *4*(3), 406–414.

[161] Welinder, P., S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona (2010), Caltech-UCSD Birds 200, *Tech. Rep. CNS-TR-2010-001*, Caltech.

[162] Weston, J., S. Bengio, and N. Usunier (2010), Large scale image annotation: Learning to rank with joint word-image embeddings, *ECML*.

[163] Wu, S., S. Bondugula, F. Luisier, X. Zhuang, and P. Natarajan (2014), Zero-shot event detection using multi-modal fusion of weakly supervised concepts, in *CVPR*.

[164] Xu, K., J. Ba, R. Kiros, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio (2015), Show, attend and tell: Neural image caption generation with visual attention, in *ICML*.

[165] Yan, X., J. Yang, K. Sohn, and H. Lee (2015), Attribute2image: Conditional image generation from visual attributes, *arXiv preprint arXiv:1512.00570*.

[166] Yaner, P. W., and A. K. Goel (2007), Understanding drawings by compositional analogy., in *IJCAI*, pp. 1131–1137.

[167] Yang, J., S. Reed, M.-H. Yang, and H. Lee (2015), Weakly-supervised disentangling with recurrent transformations for 3d view synthesis, in *NIPS*.

[168] Young, P., A. Lai, M. Hodosh, and J. Hockenmaier (2014), From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions, *Trans. of the Assoc. for Comp. Ling.*, *2*, 67–78.

[169] Zaremba, W., and I. Sutskever (2014), Learning to execute, *arXiv preprint arXiv:1410.4615*.

[170] Zaremba, W., and I. Sutskever (2015), Reinforcement learning neural turing machines, *arXiv preprint arXiv:1505.00521*.

[171] Zaremba, W., T. Mikolov, A. Joulin, and R. Fergus (2015), Learning simple algorithms from examples, *arXiv preprint arXiv:1511.07275*.

[172] Zhang, N., J. Donahue, R. Girshick, and T. Darrell (2014), Part-based R-CNNs for fine-grained category detection, in *ECCV*.

[173] Zhang, X., and Y. LeCun (2015), Text understanding from scratch, *arXiv:1502.01710*.

[174] Zheng, S., S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr (2015), Conditional random fields as recurrent neural networks, in *ICCV*.

[175] Zhu, Y., R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler (2015), Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, in *ICCV*.

[176] Zhu, Z., P. Luo, X. Wang, and X. Tang (2014), Multi-view perceptron: a deep model for learning face identity and view representations, in *NIPS*, pp. 217–225.