Deep Generative Models for Vision and Language Intelligence

by

Zhe Gan

Department of Electrical and Computer Engineering
Duke University

Date: _____

Approved:

_____
Lawrence Carin, Supervisor

_____
Katherine Heller

_____
Guillermo Sapiro

_____
Henry Pfister

_____
Galen Reeves

Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering in the Graduate School of Duke University
2018

Abstract

Deep Generative Models for Vision and Language Intelligence

by

Zhe Gan

Department of Electrical and Computer Engineering
Duke University

Date: _____

Approved:

_____
Lawrence Carin, Supervisor

_____
Katherine Heller

_____
Guillermo Sapiro

_____
Henry Pfister

_____
Galen Reeves

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering
in the Graduate School of Duke University
2018

Abstract

Deep generative models have achieved tremendous success in recent years, with applications in various tasks involving vision and language intelligence. In this dissertation, I will mainly discuss the contributions that I have made in this field during my Ph.D. study. Specifically, the dissertation is divided into two parts.

In the first part, I will mainly focus on one specific kind of deep directed generative model, called Sigmoid Belief Network (SBN). First, I will present a fully Bayesian algorithm for efficient learning and inference of SBN. Second, since the original SBN can be only used for binary image modeling, I will also discuss the generalization of it to model spare count-valued data for topic modeling, and sequential data for motion capture synthesis, music generation and dynamic topic modeling.

In the second part, I will mainly focus on visual captioning (*i.e.*, image-to-text generation), and conditional image synthesis. Specifically, I will first present Semantic Compositional Network for visual captioning, and emphasize interpretability and controllability revealed in the learning algorithm, via a mixture-of-experts design, and the usage of detected semantic concepts. I will then present Triangle Generative Adversarial Network, which is a general framework that can be used for joint distribution matching and learning the bidirectional mappings between two different domains. We consider the joint modeling of image-label, image-image and image-attribute pairs, with applications in semi-supervised image classification, image-to-image translation and attribute-based image editing.

Dedicated to my wife, and parents.

Contents

List of Tables

List of Figures

Acknowledgements

Being a Ph.D. student in the machine learning group at Duke was lots of fun, and joining it was one of the best decisions that I have ever made. First and foremost I would like to express my sincere gratitude to my advisor, Professor Lawrence Carin, for his excellent guidance, support, and inspiration. Larry taught me how to do research, how to write academic papers, and our group meetings on every Friday morning were always inspiring. Thank you for always being supportive, and giving me the freedom to explore a diverse set of topics.

I would also like to show my appreciation for my dissertation committee members, Professors Guillermo Sapiro, Henry Pfister, Galen Reeves, and Katherine Heller for their time and effort to serve on my committee. I am also very grateful to Professors David Dunson and Robert Calderbank for kindly providing me with guidance and help through classes and the preliminary exam.

I would also like to express my thanks to Dr. Hung Hui for hosting me as a research intern at Adobe Research. I also deeply appreciate Mr. Xiujun Li, Drs. Xiaodong He, Jianfeng Gao, Lihong Li and Li Deng for their enlightening discussions and valuable collaborations when I was a research intern at Microsoft Research.

Besides, I want to thank my fellow group members who either through discussions or collaborations have helped me in my graduate career. Amongst many others, this includes Ricardo Henao, Xin Yuan, Xuejun Liao, Piyush Rai, Zhengming Xing, Shaobo Han, Yingjian Wang, David Carlson, Wenzhao Lian, Changwei Hu, Kyle

Ulrich, Yunchen Pu, Yizhe Zhang, Kai Fan, Changyou Chen, Chunyuan Li, Andrew Stevens, Qinliang Su, Zhao Song, Xinyuan Zhang, Yitong Li, Wenlin Wang, Kevin Liang, Gregory Spell, Dinghan Shen, Guoyin Wang, Liqun Chen, Shuyang Dai, Jianqiao Li, Paidamoyo Chapfuwa, Ruiyi Zhang, Chenyang Tao, Hongteng Xu, Dong Wang, Weiyao Wang, and Hao Liu for their help in and outside research. I also want to thank Jiaming Song from Stanford University, Chuang Gan from Tsinghua University and Tao Xu from Lehigh University for helpful discussions in research.

Finally and most importantly, I would like to dedicate this dissertation to my wife Ya-wen Huang, and my parents Guohua Gan and Guihua Wu, for their love and support.

1

Introduction and Background

Deep generative models have achieved tremendous success in recent years, with applications in different vision and language intelligence tasks. In this chapter, I will first provide a brief overview of several popular deep generative models. Based on this, I will then discuss the contributions that I have made in this field during my Ph.D. study. This includes the deep generative models that I have developed for documents, human motions, visual captioning and joint distribution matching.

1.1 Deep Generative Models: An Overview

One of the most important tasks for artificial intelligence (AI) is to develop algorithms and techniques that endow computers with an understanding of our world. Generative models are one of the most promising approaches towards this goal[1]. Generative models typically have latent variables that are inferred given observed data; the latent variables are often used for a down-stream goal, such as classification. After training, such models are useful for inference tasks given subsequent observed data. On the other hand, generative models are also able to synthesize data by drawing

[1] `https://blog.openai.com/generative-models/`

latent variables from the prior and pushing them through the model. This suggests that generative models are at least useful in two aspects: (*i*) analyzing observed data in terms of latent variables; (*ii*) generating "fake-but-realistic" data from real data. The intuition behind this follows a famous quote:

"What I cannot create, I do not understand."

-Richard Feynman

Generative models that are descriptive of data have been widely employed in statistics and machine learning. Factor models (FMs) represent one commonly used generative model (Tipping and Bishop, 1999), and mixtures of FMs have been employed to account for more-general data distributions (Ghahramani and Hinton, 1997). However, such models are not powerful and flexible enough in terms of extracting meaningful representations from rich sensory inputs. Deep learning has achieved tremendous success in recent years, and arguments have been made to suggest that building such systems requires deep architectures: models composed of several layers of nonlinear processing (Bengio et al., 2013).

In this dissertation, we will mainly focus on deep generative models. Specifically, the following deep generative models will be discussed: (*i*) Restricted Boltzmann Machines (RBMs) (Hinton, 2002); (*ii*) Sigmoid Belief Networks (SBNs) (Neal, 1992); (*iii*) Variational Autoencoders (VAEs) (Kingma and Welling, 2013); (*iv*) Generative Adversarial Networks (GANs) (Goodfellow et al., 2014); and (*v*) autoregressive models.

1.1.1 Restricted Boltzmann Machines

Restricted Boltzmann machines (RBMs) have been used effectively in modeling distributions over binary-valued data (Salakhutdinov, 2015). Recent work on Boltzmann machines and their generalizations to exponential family distributions (Welling et al., 2005) have allowed these models to be successfully used in many applications.

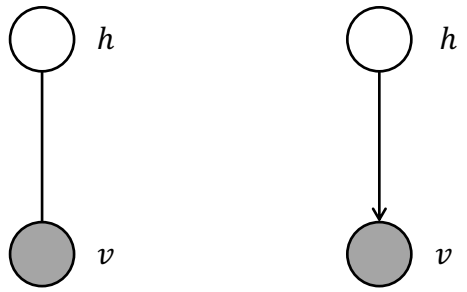


FIGURE 1.1: Graphical model of RBM and SBN, respectively.

Below we will review the standard binary RBM.

An RBM is a particular type of Markov random field that has a two-layer architecture (Smolensky, 1986), in which the "visible" stochastic binary variables $\boldsymbol{v} \in \{0,1\}^D$ are connected to "hidden" stochastic binary variables $\boldsymbol{h} \in \{0,1\}^F$, as shown in Figure 1.1(Left). The energy of the joint state $\{\boldsymbol{v}, \boldsymbol{h}\}$ is defined as follow:

$$E(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta}) = -\boldsymbol{v}^\top \mathbf{W} \boldsymbol{h} - \boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{a}^\top \boldsymbol{h} \tag{1.1}$$

$$= -\sum_{i=1}^{D}\sum_{j=1}^{F} W_{ij} v_i h_j - \sum_{i=1}^{D} b_i v_i - \sum_{j=1}^{F} a_j b_j \,, \tag{1.2}$$

where $\boldsymbol{\theta} = \{\mathbf{W}, \boldsymbol{b}, \boldsymbol{a}\}$ are the model parameters. W_{ij} represents the symmetric interaction term between visible variable i and hidden variable j, and b_i and a_j are bias terms. The joint distribution over the visible and hidden variables is defined by

$$P(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta}) = \frac{1}{\mathcal{Z}(\boldsymbol{\theta})} \exp(-E(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta})) \tag{1.3}$$

$$\mathcal{Z}(\boldsymbol{\theta}) = \sum_{\boldsymbol{v}}\sum_{\boldsymbol{h}} \exp(-E(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta}))\,. \tag{1.4}$$

The model then assigns the following probability to a visible vector $\boldsymbol{v}$:

$$P(\boldsymbol{v}; \boldsymbol{\theta}) = \frac{1}{\mathcal{Z}(\boldsymbol{\theta})} \sum_{\boldsymbol{h}} \exp(-E(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta}))\,. \tag{1.5}$$

$\mathcal{Z}(\boldsymbol{\theta})$ is a computationally intractable partition function that guarantees $P(\boldsymbol{v}; \boldsymbol{\theta})$ is a valid probability distribution.

The conditional distributions over hidden variables $\boldsymbol{h}$ and visible vectors $\boldsymbol{v}$ can be easily derived from Equation (1.3) and are given by the following logistic functions:

$$P(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta}) = \prod_j p(h_j|\boldsymbol{v}), \qquad P(\boldsymbol{v}|\boldsymbol{h};\boldsymbol{\theta}) = \prod_i (v_i|\boldsymbol{h}) \tag{1.6}$$

$$p(h_j = 1|\boldsymbol{v}) = \sigma\left(\sum_i W_{ij} v_i + a_j\right) \tag{1.7}$$

$$p(v_i = 1|\boldsymbol{h}) = \sigma\left(\sum_j W_{ij} h_j + b_i\right), \tag{1.8}$$

where $\sigma(x) = 1/(1+\exp(-x))$ is the logistic function. As can be seen, the conditional distributions in the RBM are factorial, which makes inference fast. Since exact maximum likelihood learning in this model is intractable, in practice, learning is done by using the so-called Contrastive Divergence (CD) algorithm (Hinton, 2002).

The original binary RBM has been generalized to model real-valued data (Hinton et al., 2006), such as pixel intensities of image patches, sparse count data (Salakhutdinov and Hinton, 2009b), such as word count vectors in a document, and sequential data (Sutskever and Hinton, 2007), such as human motion captures. The RBM also serves as the building blocks for the Deep Belief Network (DBN) (Hinton et al., 2006) and Deep Boltzmann Machine (DBM) (Salakhutdinov and Hinton, 2009a), which are two popular deep probabilistic generative models that provide state-of-the-art results in many problems. The RBM model has also recently been generalzied to the Truncated Gaussian Graphical Models (TGGM) (Su et al., 2017), which is a general framework for unsupervised learning that can be used for real-valued, binary and count data. TGGM also has close connections to ReLU-based neural networks.

1.1.2 Sigmoid Belief Networks

Deep directed generative models are considered for binary data, based on the Sigmoid Belief Network (SBN) (Neal, 1992) (using methods like those discussed in Salakhut-

dinov et al. (2013), the model may be readily extended to real-valued data). Assume we have N binary visible vectors, the nth of which is denoted $\boldsymbol{v}_n \in \{0,1\}^J$. An SBN is a Bayesian network that models each $\boldsymbol{v}_n$ in terms of binary hidden variables $\boldsymbol{h}_n \in \{0,1\}^K$ and weights $\mathbf{W} \in \mathbb{R}^{J \times K}$ as

$$p(v_{jn} = 1 | \boldsymbol{w}_j, \boldsymbol{h}_n, c_j) = \sigma(\boldsymbol{w}_j^\top \boldsymbol{h}_n + c_j)\,, \tag{1.9}$$

$$p(h_{kn} = 1 | b_k) = \sigma(b_k)\,, \tag{1.10}$$

where $\sigma(\cdot)$ is the logistic function defined as $\sigma(x) = 1/(1+\exp(-x))$, $\boldsymbol{v}_n = [v_{1n}, \ldots, v_{Jn}]^\top$, $\boldsymbol{h}_n = [h_{1n}, \ldots, h_{Kn}]^\top$, $\mathbf{W} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_J]^\top$, $\boldsymbol{c} = [c_1, \ldots, c_J]^\top$ and $\boldsymbol{b} = [b_1, \ldots, b_K]^\top$ are bias terms. The "local" latent vector $\boldsymbol{h}_n$ is observation-dependent (a function of n), while the "global" parameters $\mathbf{W}$ are used to characterize the mapping from $\boldsymbol{h}_n$ to $\boldsymbol{v}_n$ for all n.

The graphical model of the SBN is shown in Figure 1.1(Right). As can be seen, the SBN is closely related to the RBM. Specifically, the energy function of an RBM is defined as

$$-E(\boldsymbol{v}_n, \boldsymbol{h}_n) = \boldsymbol{v}_n^\top \boldsymbol{c} + \boldsymbol{v}_n^\top \mathbf{W} \boldsymbol{h}_n + \boldsymbol{h}_n^\top \boldsymbol{b}\,, \tag{1.11}$$

In contrast, the energy function of an SBN may be written as

$$-E(\boldsymbol{v}_n, \boldsymbol{h}_n) = \boldsymbol{v}_n^\top \boldsymbol{c} + \boldsymbol{v}_n^\top \mathbf{W} \boldsymbol{h}_n + \boldsymbol{h}_n^\top \boldsymbol{b} - \sum_j \log(1 + \exp(\boldsymbol{w}_j^\top \boldsymbol{h}_n + c_j))\,. \tag{1.12}$$

The additional term in (1.12), when compared to (1.11), makes the energy function no longer a linear function of weights $\mathbf{W}$, but a simple partition function is obtained. Therefore, the full likelihood under an SBN is trivial to calculate. Furthermore, SBNs explicitly exhibit the generative process to obtain data, in which the hidden layer provides a directed "explanation" for patterns generated in the visible layer.

Different algorithms have been developed for efficient learning and inference of SBN, including Gibbs sampling (Neal, 1992; Gan et al., 2015c), mean-field variational inference (Saul et al., 1996), a Gaussian-field approach (Barber and Sollich, 1999),

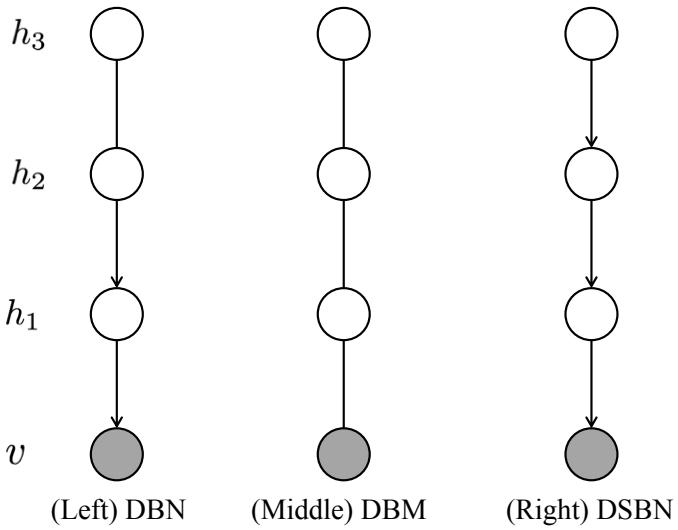


FIGURE 1.2: Graphical model of DBN, DBM and DSBN, respectively.

the wake-sleep algorithm (Hinton et al., 1995b), the Neural Variational Inference and Learning (NVIL) algorithm (Mnih and Gregor, 2014), Monte Carlo expectation maximization (Song et al., 2016b), stochastic gradient MCMC (Chen et al., 2015a; Li et al., 2016a), stochastic spectral descent (Carlson et al., 2016), the recently proposed factorized asymptotic Bayesian method (Song et al., 2017) among many others.

The original SBN model is considered only for binary data, which has been generalized to model real-valued data (Zhang et al., 2016a), sparse count-valued data (Gan et al., 2015d) and sequential data (Gan et al., 2015b). Similar to the way in which deep belief networks and deep Boltzmann machines build hierarchies, one can stack multiple SBNs to obtain a fully directed deep sigmoid belief network (DSBN). The comparison among DBN, DBM and DSBN is illustrated in Figure 1.2. DBM is a fully *undirected* graphical model, DSBN is a fully *directed* graphical model, while DBN is a *hybrid* model, where the top two hidden layers are undirected connected, while all the other layers are directed connected.

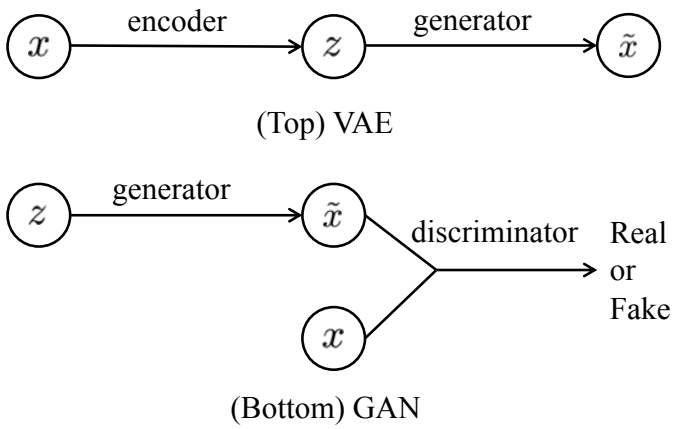


FIGURE 1.3: Illustration of VAE and GAN, respectively.

1.1.3 Variational Autoencoders

In SBN, we restrict the observation and latent variables to be binary. Now, we extend it to a more general setup. Not only the generative model being more powerful, an inference network is further proposed to allow fast approximate inference.

Consider an observed data sample $\boldsymbol{x}$, modeled as being drawn from $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, with model parameters $\boldsymbol{\theta}$ and latent code $\boldsymbol{z}$. The prior distribution on the code is denoted $p(\boldsymbol{z})$, typically a distribution that is easy to draw from, such as isotropic Gaussian. The posterior distribution on the code given data $\boldsymbol{x}$ is $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, and since this is typically intractable, it is approximated as $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, parameterized by learned parameters $\boldsymbol{\phi}$. Conditional distributions $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ are typically designed such that they are easily sampled and, for flexibility, modeled in terms of neural networks (Kingma and Welling, 2013). Since $\boldsymbol{z}$ is a latent code for $\boldsymbol{x}$, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is also termed a stochastic *encoder*, with $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ a corresponding stochastic *decoder*. The observed data are assumed drawn from $q(\boldsymbol{x})$, for which we do not have an explicit form, but from which we have samples, *i.e.*, the ensemble $\{\boldsymbol{x}_i\}_{i=1,N}$ used for learning. An illustration of the VAE is provided in Figure 1.3(Top).

Our goal is to learn the model $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}$ such that it synthesizes

samples that are well matched to those drawn from $q(\boldsymbol{x})$. We simultaneously seek to learn a corresponding encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ that is both accurate and efficient to implement. Samples $\boldsymbol{x}$ are synthesized via $\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ with $\boldsymbol{z} \sim p(\boldsymbol{z})$; $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ provides an efficient coding of observed $\boldsymbol{x}$, that may be used for other purposes (*e.g.*, classification or caption generation when $\boldsymbol{x}$ is an image (Pu et al., 2016b)).

Specifically, the generative model is specified as

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|0, \mathbf{I}), \quad p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = f(\boldsymbol{x}; \boldsymbol{z}, \boldsymbol{\theta}) \,, \tag{1.13}$$

where the function $f(\boldsymbol{x}; \boldsymbol{z}, \boldsymbol{\theta})$ is a suitable likelihood, modeled by a deterministic neural network.

When doing posterior inference, we introduce an inference network, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, *e.g.*,

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}), \mathrm{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x}))) \,, \tag{1.14}$$

where $\boldsymbol{\phi}$ is the recognition parameters, and $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x})$ and $\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})$ are modeled by deterministic neural networks.

The objective function is the variational lower bound on the marginal likelihood, which can be written as

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) \right] \tag{1.15}$$

$$= -D_{KL} \left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) || p(\boldsymbol{z}) \right) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) \right] \,. \tag{1.16}$$

The reason why we express the lower bound as (1.16) is that, (i) the first KL divergence term is usually tractable, hence reducing the uncertainty of the gradients if we calculate it explicitly; (ii) we can see clearly that, the first KL term is the regularization term, while the second term is similar to the reconstruction error term in the traditional auto-encoder framework.

The learning of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ is via optimization methods, usually utilizing stochastic gradient descent. In order to efficiently evaluate the gradient and reduce the variance of the gradient information, the reparameterization trick is used. To be specific, since

$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is specified as (1.14), hence

$$\boldsymbol{z} = \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}) + \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x}) \circ \boldsymbol{\epsilon} \;\; \text{with} \;\; \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \,. \tag{1.17}$$

Therefore,

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] = \mathbb{E}_{\mathcal{N}(\epsilon|0,\mathbf{I})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}) + \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x}) \circ \boldsymbol{\epsilon})\right] \,. \tag{1.18}$$

Now, the gradient can be evaluated as

$$\nabla_{\{\boldsymbol{\theta},\boldsymbol{\phi}\}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] = \mathbb{E}_{\mathcal{N}(\epsilon|0,\mathbf{I})} \left[\nabla_{\{\boldsymbol{\theta},\boldsymbol{\phi}\}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}) + \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x}) \circ \boldsymbol{\epsilon})\right] \,. \tag{1.19}$$

The original VAEs implement a Gaussian assumption for the encoder. More recently, there has been a desire to remove this Gaussian assumption. Normalizing flow (Rezende and Mohamed, 2015) employs a sequence of invertible transformation to make the distribution of the latent codes arbitrarily flexible. This work was followed by inverse auto-regressive flow (Kingma et al., 2016), which uses recurrent neural networks to make the latent codes more expressive. More recently, Stein-VAE (Pu et al., 2017b) applies Stein variational gradient descent (Liu and Wang, 2016) to infer the distribution of latent codes, discarding the assumption of a parametric form of posterior distribution for the latent code.

1.1.4 Generative Adversarial Networks

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) constitute another recent framework for learning a generative model. Specifically, GAN consists of a generator G and a discriminator D that compete in a two-player minimax game, where the generator is learned to map samples from an arbitray latent distribution to data, while the discriminator tries to distinguish between real and generated samples. The goal of the generator is to "fool" the discriminator by producing samples that are as close to real data as possible. An illustration of the GAN model is provided in Figure 1.3(Bottom). Specifically, D and G are learned as

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \,, \tag{1.20}$$

where $p(\boldsymbol{x})$ is the true data distribution, and $p_z(\boldsymbol{z})$ is usually defined to be a simple distribution, such as the standard normal distribution. The generator G implicitly defines a probability distribution $p_g(\boldsymbol{x})$ as the distribution of the samples $G(\boldsymbol{z})$ obtained when $\boldsymbol{z} \sim p_z(\boldsymbol{z})$. For any fixed generator G, the optimal discriminator is $D(\boldsymbol{x}) = \frac{p(\boldsymbol{x})}{p_g(\boldsymbol{x})+p(\boldsymbol{x})}$. When the discriminator is optimal, solving this adversarial game is equivalent to minimizing the Jenson-Shannon Divergence (JSD) between $p(\boldsymbol{x})$ and $p_g(\boldsymbol{x})$ (Goodfellow et al., 2014). The global equilibrium is achieved if and only if $p(\boldsymbol{x}) = p_g(\boldsymbol{x})$, and the optimal value is $-2\log 2$.

Recent extensions of GAN have focused on boosting the performance of image generation by improving the generator (Radford et al., 2016), discriminator (Zhao et al., 2017) or the training algorithm (Salimans et al., 2016; Arjovsky et al., 2017). More recently, some researchers (Dumoulin et al., 2017; Donahue et al., 2017) have employed a bidirectional network structure within the adversarial learning framework, which in theory guarantees the matching of joint distributions over two domains. However, non-identifiability issues are raised in Li et al. (2017a). For example, they have difficulties in providing good reconstruction in latent variable models, or discovering the correct pairing relationship in domain transformation tasks. It was shown that these problems are alleviated in DiscoGAN (Kim et al., 2017), CycleGAN (Zhu et al., 2017) via additional ℓ_1, ℓ_2 or adversarial losses.

1.1.5 Autoregressive Models

Autoregressive models such as PixelRNN (Oord et al., 2016b) and PixelCNN (Oord et al., 2016a) instead train a network that models the conditional distribution of every individual pixel given previous pixels. Similar ideas have been extended for generating raw audio (Oord et al., 2016c) and language modeling (Dauphin et al., 2016).

Besides the popular PixelRNN model, recurrent neural networks (RNNs) also

serve as a popular autoregressive model that has been widely used in natural language processing applications. Below we briefly review RNN.

An RNN is a special type of neural network that is able to handle both variable-length input and output. By training an RNN to predict the next output in a sequence, given all previous outputs, it can be used to model joint probability distribution over sequences. To be specific, an RNN can take as input a sequence $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T]$ by recursively processing each symbol while maintaining its internal hidden state $\boldsymbol{h}$. At time step t, the RNN reads the symbol $\boldsymbol{x}_t \in \mathbb{R}^d$ and updates its hidden state $\boldsymbol{h}_t \in \mathbb{R}^p$ by

$$\boldsymbol{h}_t = f_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) \,, \tag{1.21}$$

where f is a deterministic non-linear transition function, and $\boldsymbol{\theta}$ is the parameter set of f. The transition function f can be implemented with gated activation functions such as long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) or gated recurrent units (GRU) (Cho et al., 2014), introduced below. RNNs model sequences by parameterizing a factorization of the joint sequence probability distribution as a product of conditional probabilities such that

$$p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) = \prod_{t=1}^{T} p(\boldsymbol{x}_t | \boldsymbol{x}_{<t}), \;\; p(\boldsymbol{x}_t | \boldsymbol{x}_{<t}) = g_{\boldsymbol{\phi}}(\boldsymbol{h}_{t-1}) \,, \tag{1.22}$$

where g is a function that maps the RNN hidden state $\boldsymbol{h}_{t-1}$ to a probability distribution onver possible outputs, and $\boldsymbol{\phi}$ is the parameter set of g.

Vanilla RNN Equation (1.21) is defined as a simple non-linear hyperbolic tangent function

$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1} + \boldsymbol{b}) \,. \tag{1.23}$$

Long Short-Term Memory Unfortunately, a problem with RNNs with the above formulation (1.23) is that, during training, the components of the gradient vector can

grow or decay exponentially over long sequences (Hochreiter and Schmidhuber, 1997). This problem with *exploding* or *vanishing* gradients makes it difficult for the RNN model to learn long-range dependencies in a sequence.

The LSTM architecture (Hochreiter and Schmidhuber, 1997) addresses this problem of learning long-term dependencies by introducing a *memory cell* that is able to preserve state over long periods of time. To be specific, each LSTM unit has a cell which has a state $\boldsymbol{c}_t$ at time t. This cell can be thought of as a memory unit. Access to this memory unit for reading or modifying it is controlled through sigmoid gates: input gate $\boldsymbol{i}_t$, forget gate $\boldsymbol{f}_t$, and output gate $\boldsymbol{o}_t$. The hidden units $\boldsymbol{h}_t$ are updated as follows

$$\boldsymbol{i}_t = \sigma(\mathbf{W}_i \boldsymbol{x}_t + \mathbf{U}_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i)\,, \qquad \boldsymbol{f}_t = \sigma(\mathbf{W}_f \boldsymbol{x}_t + \mathbf{U}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f)\,, \tag{1.24}$$

$$\boldsymbol{o}_t = \sigma(\mathbf{W}_o \boldsymbol{x}_t + \mathbf{U}_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o)\,, \qquad \tilde{\boldsymbol{c}}_t = \tanh(\mathbf{W}_c \boldsymbol{x}_t + \mathbf{U}_c \boldsymbol{h}_{t-1} + \boldsymbol{b}_c)\,, \tag{1.25}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t\,, \qquad \boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)\,. \tag{1.26}$$

where $\sigma(\cdot)$ denotes the logistic sigmoid function, and $\odot$ represents the element-wise multiply operator. The key advantage of using an LSTM unit over a traditional neuron in an RNN is that the cell state in an LSTM unit sums activities over time.

Gated Recurrent Units The gated recurrent unit (GRU) was proposed by Cho et al. (2014) to make each recurrent unit to adaptively capture dependencies of different time scales. Similar to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, without using a separate memory cell. Specifically, the GRU has two gates: the reset gate $\boldsymbol{r}_t$ and the update gate $\boldsymbol{z}_t$. The hidden units $\boldsymbol{h}_t$ are updated as follows

$$\boldsymbol{r}_t = \sigma(\mathbf{W}_r \boldsymbol{x}_t + \mathbf{U}_r \boldsymbol{h}_{t-1} + \boldsymbol{b}_r)\,, \qquad \boldsymbol{z}_t = \sigma(\mathbf{W}_z \boldsymbol{x}_t + \mathbf{U}_z \boldsymbol{h}_{t-1} + \boldsymbol{b}_z)\,, \tag{1.27}$$

$$\tilde{\boldsymbol{h}}_t = \tanh(\mathbf{W} \boldsymbol{x}_t + \mathbf{U}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{b})\,, \quad \boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t\,. \tag{1.28}$$

Table 1.1: Summary of thesis contributions.

Deep Generative Models	**Applications in Vision and Language Intelligence**
SBN (Gan et al., 2015c)	Binary image modeling
DPFA (Gan et al., 2015d)	Topic modeling
TSBN (Gan et al., 2015b)	Motion capture synthesis, Dynamic topic modeling
SCN (Gan et al., 2016b)	Visual captioning
TriangleGAN (Gan et al., 2017f)	Face image editing, Image-to-image translation

It has been shown that the GRU can achieve similar performances compared with LSTM in the task of sequence modeling (Chung et al., 2014).

Variants Each of the three RNN architectures considered above can be expanded to the Bidirectional RNN and the Multilayer RNN (also known as the *stacked* or *deep* RNN). A Bidirectional RNN (Graves, 2013) consists of two RNNs that are run in parallel: one on the input sequence and the other on the reverse of the input sequence. At each time step, the hidden state of the bidirectional RNN is the concatenation of the forward and backward hidden states. In Multilayer RNNs, the hidden state of an RNN unit in layer ℓ is used as input to the RNN unit in layer $\ell + 1$ in the same time step (Graves, 2013). The idea here is to let the higher layers capture longer-term dependencies of the input sequence.

1.2 Thesis Contribution

The contributions of this dissertation is summarized in Table 1.1. The first three models are all based on SBNs, which focus on modeling the data distribution $p(\boldsymbol{x})$ in one domain. The fourth model is based on RNN, while the fifth model is based on GAN. These latter two both focus on modeling the joint distribution $p(\boldsymbol{x}, \boldsymbol{y})$ in two related domains. Below, I will briefly review each individual model listed in Table 1.1.

1.2.1 Deep Generative Models for Binary Image Data

Deep directed generative models are developed. The multi-layered model is designed by stacking sigmoid belief networks, with sparsity-encouraging priors placed on the model parameters. Learning and inference of layer-wise model parameters are implemented in a Bayesian setting. By exploring the idea of data augmentation and introducing auxiliary Pólya-Gamma variables, simple and efficient Gibbs sampling and mean-field variational Bayes (VB) inference are implemented. To address large-scale datasets, an online version of VB is also developed. Experimental results are presented for three publicly available datasets: MNIST, Caltech 101 Silhouettes and OCR letters.

1.2.2 Deep Generative Models for Documents

A new framework for topic modeling is developed, based on deep graphical models, where interactions between topics are inferred through deep latent binary hierarchies. The proposed multi-layer model employs a deep sigmoid belief network or restricted Boltzmann machine, the bottom binary layer of which selects topics for use in a Poisson factor analysis model. Under this setting, topics live on the bottom layer of the model, while the deep specification serves as a flexible prior for revealing topic structure. Scalable inference algorithms are derived by applying Bayesian conditional density filtering algorithm, in addition to extending recently proposed work on stochastic gradient thermostats. Experimental results on several corpora show that the proposed approach readily handles very large collections of text documents, infers structured topic representations, and obtains superior test perplexities when compared with related models.

1.2.3 Deep Generative Models for Sequential Data

Deep dynamic generative models are developed to learn sequential dependencies in time-series data. The multi-layered model is designed by constructing a hierarchy of temporal sigmoid belief networks (TSBNs), defined as a sequential stack of sigmoid belief networks (SBNs). Each SBN has a contextual hidden state, inherited from the previous SBNs in the sequence, and is used to regulate its hidden bias. Scalable learning and inference algorithms are derived by introducing a recognition model that yields fast sampling from the variational posterior. This recognition model is trained jointly with the generative model, by maximizing its variational lower bound on the log-likelihood. Experimental results on bouncing balls, polyphonic music, motion capture, and text streams show that the proposed approach achieves state-of-the-art predictive performance, and has the capacity to synthesize various sequences.

1.2.4 Deep Generative Models for Visual Captioning

A Semantic Compositional Network (SCN) is developed for image captioning, in which semantic concepts (*i.e.*, tags) are detected from the image, and the probability of each tag is used to compose the parameters in a long short-term memory (LSTM) network. The SCN extends each weight matrix of the LSTM to an ensemble of tag-dependent weight matrices. The degree to which each member of the ensemble is used to generate an image caption is tied to the image-dependent probability of the corresponding tag. In addition to captioning images, we also extend the SCN to generate captions for video clips. We qualitatively analyze semantic composition in SCNs, and quantitatively evaluate the algorithm on three benchmark datasets: COCO, Flickr30k, and Youtube2Text. Experimental results show that the proposed method significantly outperforms prior state-of-the-art approaches, across multiple evaluation metrics.

1.2.5 Deep Generative Models for Joint Distribution Matching

A Triangle Generative Adversarial Network (Δ-GAN) is developed for semi-supervised cross-domain joint distribution matching, where the training data consists of samples from each domain, and supervision of domain correspondence is provided by only a few *paired* samples. Δ-GAN consists of four neural networks, two generators and two discriminators. The generators are designed to learn the two-way conditional distributions between the two domains, while the discriminators implicitly define a ternary discriminative function, which is trained to distinguish real data pairs and two kinds of fake data pairs. The generators and discriminators are trained together using adversarial learning. Under mild assumptions, in theory the joint distributions characterized by the two generators concentrate to the data distribution. In experiments, three different kinds of domain pairs are considered, image-label, image-image and image-attribute pairs. Experiments on semi-supervised image classification, image-to-image translation and attribute-based image generation demonstrate the superiority of the proposed approach.

2

Learning Sigmoid Belief Networks

In this chapter, I will present sigmoid belief networks for representation learning. Both Gibbs sampling and mean-field variational approximation are implemented for efficient learning and inference in a fully Bayesian setup, by using the idea of data augmentation.

2.1 Introduction

The Deep Belief Network (DBN) (Hinton et al., 2006) and Deep Boltzmann Machine (DBM) (Salakhutdinov and Hinton, 2009a) are two popular deep probabilistic generative models that provide state-of-the-art results in many problems. These models contain many layers of hidden variables, and utilize an *undirected* graphical model called the Restricted Boltzmann Machine (RBM) (Hinton, 2002) as the building block. A nice property of the RBM is that gradient estimates on the model parameters are relatively quick to calculate, and stochastic gradient descent provides relatively efficient inference. However, evaluating the probability of a data point under an RBM is nontrivial due to the computationally intractable partition function, which has to be estimated, for example using an annealed importance sampling

algorithm (Salakhutdinov and Murray, 2008).

A *directed* graphical model that is closely related to these models is the Sigmoid Belief Network (SBN) (Neal, 1992). The SBN has a fully generative process and data are readily generated from the model using ancestral sampling. However, it has been noted that training a deep directed generative model is difficult, due to the "explaining away" effect. Hinton et al. (2006) tackle this problem by introducing the idea of "complementary priors" and show that the RBM provides a good initialization to the DBN, which has the same generative model as the SBN for all layers except the two top hidden layers. In the work presented here we directly deal with training and inference in SBNs (without RBM initialization), using recently developed methods in the Bayesian statistics literature.

Previous work on SBNs utilizes the ideas of Gibbs sampling (Neal, 1992) and mean field approximations (Saul et al., 1996). Recent work focuses on extending the wake-sleep algorithm (Hinton et al., 1995b) to training fast variational approximations for the SBN (Mnih and Gregor, 2014). However, almost all previous work assumes no prior on the model parameters which connect different layers. An exception is the work of Kingma and Welling (2013), but this is mentioned as an extension of their primary work. Previous Gibbs sampling and variational inference procedures are implemented only on the hidden variables, while gradient ascent is employed to learn good model parameter values. The typical regularization on the model parameters is early stopping and/or ℓ^2 regularization. In an SBN, the model parameters are not straightforwardly locally conjugate, and therefore fully Bayesian inference has been difficult.

The work presented here provides a method for placing priors on the model parameters, and presents a simple Gibbs sampling algorithm, by extending recent work on data augmentation for Bayesian logistic regression (Polson et al., 2013a). More specifically, a set of Pólya-Gamma variables are used for each observation, to re-

formulate the logistic likelihood as a scale mixture, where each mixture component is conditionally normal with respect to the model parameters. Efficient mean-field variational learning and inference are also developed, to optimize a data-augmented variational lower bound; this approach can be scaled up to large datasets. Utilizing these methods, sparsity-encouraging priors are placed on the model parameters and the posterior distribution of model parameters is estimated (not simply a point estimate). Based on extensive experiments, we provide a detailed analysis of the performance of the proposed method.

2.2 Model formulation

2.2.1 Sigmoid Belief Networks

Assume we have N binary visible vectors, the nth of which is denoted $\boldsymbol{v}_n \in \{0,1\}^J$. As described in Section 1.1.2, an SBN is a Bayesian network that models each $\boldsymbol{v}_n$ in terms of binary hidden variables $\boldsymbol{h}_n \in \{0,1\}^K$ and weights $\mathbf{W} \in \mathbb{R}^{J\times K}$ as

$$p(v_{jn} = 1|\boldsymbol{w}_j, \boldsymbol{h}_n, c_j) = \sigma(\boldsymbol{w}_j^\top \boldsymbol{h}_n + c_j) \tag{2.1}$$

$$p(h_{kn} = 1|b_k) = \sigma(b_k)\,, \tag{2.2}$$

where $\sigma(\cdot)$ is the logistic function defined as $\sigma(x) = 1/(1+\exp(-x))$, $\boldsymbol{v}_n = [v_{1n}, \ldots, v_{Jn}]^\top$, $\boldsymbol{h}_n = [h_{1n}, \ldots, h_{Kn}]^\top$, $\mathbf{W} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_J]^\top$, $\boldsymbol{c} = [c_1, \ldots, \ c_J]^\top$ and $\boldsymbol{b} = [b_1, \ldots, b_K]^\top$ are bias terms.

2.2.2 Autoregressive Structure

Instead of assuming that the visible variables in an SBN are conditionally independent given the hidden units, a more flexible model can be built by using an autoregressive structure. The autoregressive sigmoid belief network (ARSBN) (Gregor et al., 2014) is an SBN with within-layer dependency captured by a fully connected directed acyclic graph, where each unit x_j can be predicted by its parent units $\boldsymbol{x}_{<j}$,

defined as $\{x_1, \ldots, x_{j-1}\}$. To be specific,

$$p(v_{jn} = 1|\boldsymbol{h}_n, \boldsymbol{v}_{<j,n}) = \sigma(\boldsymbol{w}_j^\top \boldsymbol{h}_n + \boldsymbol{s}_{j,<j}^\top \boldsymbol{v}_{<j,n} + c_j) \tag{2.3}$$

$$p(h_{kn} = 1|\boldsymbol{h}_{<k,n}) = \sigma(\boldsymbol{u}_{k,<k}^\top \boldsymbol{h}_{<k,n} + b_k)\,, \tag{2.4}$$

where $\mathbf{S} = [\boldsymbol{s}_1, \ldots, \boldsymbol{s}_J]^\top$ and $\mathbf{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K]^\top$ are a lower triangular matrix that contains the autoregressive weights within layers, while $\mathbf{W}$ is utilized to capture the dependencies between different layers. If no hidden layer exists, we obtain the fully visible sigmoid belief network (Frey, 1998), in which accurate probabilities of test data points can be calculated.

In the work presented here, only stochastic autoregressive layers are considered, while Gregor et al. (2014) further explore the utilization of deterministic hidden layers. Furthermore, instead of using the simple linear autoregressive structure, one can increase the representational power of the model by using more-complicated autoregressive models, such as the work by Larochelle and Murray (2011), where each conditional $p(v_{jn}|\boldsymbol{v}_{<j,n})$ is modeled by a neural network.

2.2.3 Deep Sigmoid Belief Networks

Similar to the way in which deep belief networks and deep Boltzmann machines build hierarchies, one can stack additional hidden layers to obtain a fully directed deep sigmoid belief network (DSBN). Consider a deep model with L layers of hidden variables. To generate a sample, we begin at the top, layer L. For each layer below, activation $\boldsymbol{h}^{(l)}$ is formed by a sigmoid transformation of the layer above $\boldsymbol{h}^{(l+1)}$ weighted by $\mathbf{W}^{(l+1)}$. We repeat this process until the observation is reached. Therefore, the complete generative model can be written as

$$p(\boldsymbol{v}_n, \boldsymbol{h}_n) = p(\boldsymbol{v}_n|\boldsymbol{h}_n^{(1)})p(\boldsymbol{h}_n^{(L)}) \prod_{l=1}^{L-1} p(\boldsymbol{h}_n^{(l)}|\boldsymbol{h}_n^{(l+1)})\,. \tag{2.5}$$

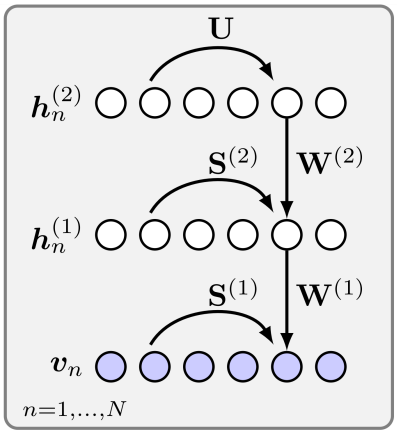


FIGURE 2.1: Graphical model for the deep SBN with autoregressive structure.

Let $\{h_{1n}^{(l)}, h_{2n}^{(l)}, \ldots, h_{K_l n}^{(l)}\}$ represent the set of hidden units for observation n in layer l. For the top layer, the prior probability can be written as $p(h_{kn}^{(L)} = 1) = \sigma(c_k^{(L+1)})$, where $c_k^{(L+1)} \in \mathbb{R}$. Defining $\boldsymbol{v}_n = \boldsymbol{h}_n^{(0)}$, conditioned on the hidden units $\boldsymbol{h}_n^{(l)}$, the hidden units at layer $l-1$ are drawn from

$$p(h_{kn}^{(l-1)} | \boldsymbol{h}_n^{(l)}) = \sigma((\boldsymbol{w}_k^{(l)})^\top \boldsymbol{h}_n^{(l)} + c_k^{(l)}), \tag{2.6}$$

where $\mathbf{W}^{(l)} = [\boldsymbol{w}_1^{(l)}, \ldots, \boldsymbol{w}_{K_{l-1}}^{(l)}]^\top$ connects layers l and $l-1$ and $\boldsymbol{c}^{(l)} = [c_1^{(l)}, \ldots, c_{K_{l-1}}^{(l)}]^\top$ is the bias term.

Figure 2.1 shows the graphical model for the deep SBN with autoregressive structure. $\mathbf{S}^{(\ell)}$ and $\mathbf{U}$ contain the autoregressive weights within layers, while $\mathbf{W}^{(\ell)}$ is utilized to capture the dependencies between different layers.

2.2.4 Bayesian sparsity shrinkage prior

The learned features are often expected to be sparse. In imagery, for example, features learned at the bottom layer tend to be localized, oriented edge filters which are similar to the Gabor functions known to model V1 cell receptive fields (Lee et al., 2008).

Under the Bayesian framework, sparsity-encouraging priors can be specified in a

principled way. Some canonical examples are the spike-and-slab prior, the Student's-t prior, the double exponential prior and the horseshoe prior (see Polson and Scott (2012) for a discussion of these priors). The three parameter beta normal (TPBN) prior (Armagan et al., 2011), a typical global-local shrinkage prior, has demonstrated better (mixing) performance than the aforementioned priors, and thus is employed in this paper. The TPBN shrinkage prior can be expressed as scale mixtures of normals. If $W_{jk} \sim \text{TPBN}(a, b, \phi)$, where $j = 1, \ldots, J, k = 1, \ldots, K$, (leaving off the dependence on the layer l, for notational convenience) then

$$
\begin{aligned}
W_{jk} &\sim N(0, \zeta_{jk}) \,, && && (2.7)\\
\zeta_{jk} &\sim \text{Gamma}(a, \xi_{jk}) \,, && \xi_{jk} \sim \text{Gamma}(b, \phi_k) \,,\\
\phi_k &\sim \text{Gamma}(1/2, \omega) \,, && \omega \sim \text{Gamma}(1/2, 1) \,.
\end{aligned}
$$

When $a = b = \frac{1}{2}$, the TPBN recovers the horseshoe prior. For fixed values of a and b, decreasing ϕ encourages more support for stronger shrinkage. In high-dimensional settings, ϕ can be fixed at a reasonable value to reflect an appropriate expected sparsity rather than inferring it from data.

Finally, to build up the fully generative model, commonly used isotropic normal prior are imposed on the bias term $\boldsymbol{b}$ and $\boldsymbol{c}$, i.e. $\boldsymbol{b} \sim N(0, \nu_b \mathbf{I}_K), \boldsymbol{c} \sim N(0, \nu_c \mathbf{I}_J)$.

Note that when performing model learning, we truncate the number of hidden units at each layer at K, which may be viewed as an upper bound within the model on the number of units at each layer. With the aforementioned shrinkage on $\mathbf{W}$, the model has the capacity to infer the subset of units (possibly less than K) actually needed to represent the data.

2.3 Learning and inference

In this section, Gibbs sampling and mean field variational inference are derived for the sigmoid belief networks, based on data augmentation. From the perspective of

learning, we desire distributions on the model parameters $\{\mathbf{W}^{(l)}\}$ and $\{\boldsymbol{c}^{(l)}\}$, and distributions on the data-dependent $\{\boldsymbol{h}_n^{(l)}\}$ are desired in the context of inference. The extension to ARSBN is straightforward and hence omitted. We again omit the layer index l in the discussion below.

2.3.1 Gibbs sampling

Define $\mathbf{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N]$ and $\mathbf{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N]$. From recent work for the Pólya-Gamma data augmentation strategy (Polson et al., 2013a), that is, if $\gamma \sim \mathrm{PG}(b, 0)$, $b > 0$, then

$$\frac{(e^{\psi})^a}{(1+e^{\psi})^b} = 2^{-b} e^{\kappa\psi} \int_0^{\infty} e^{-\gamma\psi^2/2} p(\gamma) d\gamma \,, \tag{2.8}$$

where $\kappa = a - b/2$. Properties of the Pólya-Gamma variables are summarized in Section 2.7.1. Therefore, the data-augmented joint posterior of the SBN model can be expressed as

$$\begin{aligned} & p(\mathbf{W}, \mathbf{H}, \boldsymbol{b}, \boldsymbol{c}, \gamma^{(0)}, \gamma^{(1)} | \mathbf{V}) \qquad (2.9) \\ \propto & \exp\left\{ \sum_{j,n} (v_{jn} - \frac{1}{2})(\boldsymbol{w}_j^{\top} \boldsymbol{h}_n + c_j) - \frac{1}{2} \gamma_{jn}^{(0)} (\boldsymbol{w}_j^{\top} \boldsymbol{h}_n + c_j)^2 \right\} \\ & \cdot \exp\left\{ \sum_{k,n} (h_{kn} - \frac{1}{2}) b_k - \frac{1}{2} \gamma_k^{(1)} b_k^2 \right\} \cdot p_0(\gamma^{(0)}, \gamma^{(1)}, \mathbf{W}, \boldsymbol{b}, \boldsymbol{c}) \,, \end{aligned}$$

where $\gamma^{(0)} \in \mathbb{R}^{J \times N}$ and $\gamma^{(1)} \in \mathbb{R}^K$ are augmented random variables drawn from the Pólya-Gamma distribution. The term $p_0(\gamma^{(0)}, \gamma^{(1)}, \mathbf{W}, \boldsymbol{b}, \boldsymbol{c})$ contains the prior information of the random variables within. Let $p(\cdot|-)$ represent the conditional distribution given other parameters fixed, then the conditional distributions used in the Gibbs sampling are as follows.

For $\gamma^{(0)}, \gamma^{(1)}$: The conditional distribution of $\gamma^{(0)}$ is

$$p(\gamma_{jn}^{(0)}|-) \propto \exp\left(-\frac{1}{2}\gamma_{jn}^{(0)}(\boldsymbol{w}_j^\top \boldsymbol{h}_n + c_j)^2\right) \cdot \mathrm{PG}(\gamma_{jn}^{(0)}|1,0)$$
$$= \mathrm{PG}(1, \boldsymbol{w}_j^\top \boldsymbol{h}_n + c_j)\,, \tag{2.10}$$

where $\mathrm{PG}(\cdot,\cdot)$ represents the Pólya-Gamma distribution. Similarly, we can obtain $p(\gamma_k^{(1)}|-) = \mathrm{PG}(1, b_k)$.

To draw samples from the Pólya-Gamma distribution, two strategies are utilized: (i) using rejection sampling to draw samples from the closely related exponentially tilted Jacobi distribution (Polson et al., 2013a); (ii) using a truncated sum of random variables from the Gamma distribution and then match the first moment to keep the samples unbiased (Zhou et al., 2012b). Typically, a truncation level of 20 works well in practice.

For H: The sequential update of the local conditional distribution of $\mathbf{H}$ is $p(h_{kn}|-) = \mathrm{Ber}(\sigma(d_{kn}))$, where

$$d_{kn} = b_k + \boldsymbol{w}_k^\top \boldsymbol{v}_n - \frac{1}{2}\sum_{j=1}^{J}\left(w_{jk} + \gamma_{jn}^{(0)}(2\psi_{jn}^{\setminus k} w_{jk} + w_{jk}^2)\right)\,, \tag{2.11}$$

where $\psi_{jn}^{\setminus k} = \boldsymbol{w}_j^\top \boldsymbol{h}_n - w_{jk}h_{kn} + c_j$. Note that $\boldsymbol{w}_k$ and $\boldsymbol{w}_j$ represent the kth column and the transpose of the jth row of $\mathbf{W}$, respectively. The difference between an SBN and an RBM can be seen more clearly from the sequential update. Specifically, in an RBM, the update of h_{kn} only contains the first two terms, which implies the update of the h_{kn} are independent of each other. In the SBN, the existence of the third term demonstrates clearly the posterior dependencies between hidden units. Although the rows of $\mathbf{H}$ are correlated, the columns of $\mathbf{H}$ are independent, therefore the sampling of $\mathbf{H}$ is still efficient.

For W: The prior is a TPBN shrinkage prior with $p_0(\boldsymbol{w}_j) = N(0, \mathrm{diag}(\boldsymbol{\zeta}_j))$, then

we have $p(\boldsymbol{w}_j|-) = N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$, where

$$\boldsymbol{\Sigma}_j = \left[\sum_{n=1}^{N} \gamma_{jn}^{(0)} \boldsymbol{h}_n \boldsymbol{h}_n^\top + \mathrm{diag}(\boldsymbol{\zeta}_j^{-1})\right]^{-1}, \tag{2.12}$$

$$\boldsymbol{\mu}_j = \boldsymbol{\Sigma}_j \left[\sum_{n=1}^{N} (v_{jn} - \frac{1}{2} - c_j \gamma_{jn}^{(0)}) \boldsymbol{h}_n\right]. \tag{2.13}$$

The update of the bias term $\boldsymbol{b}$ and $\boldsymbol{c}$ are similar to the above equation.

For TPBN shrinkage: One advantage of this hierarchical shrinkage prior is the full local conjugacy that allows the Gibbs sampling easily implemented. Specifically, the following posterior conditional distribution can be achieved: (1) $\zeta_{jk}|- \sim \mathcal{GIG}(0, 2\xi_{jk}, W_{jk}^2)$; (2) $\xi_{jk}|- \sim \mathrm{Gamma}(1, \zeta_{jk} + \phi_k)$; (3) $\phi_k|- \sim \mathrm{Gamma}(\frac{1}{2}J + \frac{1}{2}, \omega + \sum_{j=1}^{J} \xi_{jk})$; (4) $\omega|- \sim \mathrm{Gamma}(\frac{1}{2}K + \frac{1}{2}, 1 + \sum_{k=1}^{K} \phi_k)$, where $\mathcal{GIG}$ denotes the generalized inverse Gaussian distribution.

2.3.2 Mean field variational Bayes

Using the VB inference with the traditional mean field assumption, we approximate the posterior distribution with $Q = \prod_{j,k} q_{w_{jk}}(w_{jk}) \prod_{j,n} q_{h_{jn}}(h_{jn}) q_{\gamma_{jn}^{(0)}}(\gamma_{jn}^{(0)})$; for notational simplicity the terms concerning $\boldsymbol{b}, \boldsymbol{c}, \gamma^{(1)}$ and the parameters of the TPBN shrinkage prior are omitted. The variational lower bound can be obtained as

$$\begin{aligned}\mathcal{L} = &\langle \log p(\mathbf{V}|\mathbf{W}, \mathbf{H}, \boldsymbol{c})\rangle + \langle \log p(\mathbf{W})\rangle + \langle \log p(\mathbf{H}|\boldsymbol{b})\rangle \\ &- \langle \log q(\mathbf{W})\rangle - \langle \log q(\mathbf{H})\rangle,\end{aligned} \tag{2.14}$$

where $\langle \cdot \rangle$ represents the expectation w.r.t. the variational approximate posterior.

Note that $\langle \log p(\mathbf{V}|-)\rangle = \sum_{j,n} \langle \log p(v_{jn}|-)\rangle$, and each term inside the summation can be further lower bounded by using the augmented Pólya-Gamma variables. Specifically, defining $\psi_{jn} = \boldsymbol{w}_j^\top \boldsymbol{h}_n + c_j$, we can obtain

$$\begin{aligned}\langle \log p(v_{jn}|-)\rangle \geqslant\ &-\log 2 + (v_{jn} - 1/2)\langle \psi_{jn}\rangle \\ &-\frac{1}{2}\langle \gamma_{jn}^{(0)}\rangle\langle \psi_{jn}^2\rangle + \langle \log p_0(\gamma_{jn}^{(0)})\rangle - \langle \log q(\gamma_{jn}^{(0)})\rangle,\end{aligned} \tag{2.15}$$

by using (2.8) and Jensen's inequality. Therefore, the new lower bound $\mathcal{L}'$ can be achieved by substituting (2.15) into (4.6). Note that this is a looser lower bound compared with the original lower bound $\mathcal{L}$, due to the data augmentation. However, closed-form coordinate ascent update equations can be obtained, shown below.

For $\gamma^{(0)}, \gamma^{(1)}$: optimizing $\mathcal{L}'$ over $q(\gamma_{jn}^{(0)})$, we have

$$\begin{aligned} q(\gamma_{jn}^{(0)}) \propto & \ \exp\left(-\frac{1}{2}\gamma_{jn}^{(0)}\langle\psi_{jn}^2\rangle\right) \cdot \mathrm{PG}(\gamma_{jn}^{(0)}|1,0) \\ = & \ \mathrm{PG}\left(1, \sqrt{\langle\psi_{jn}^2\rangle}\right). \end{aligned} \tag{2.16}$$

Similarly, we can obtain $p(\gamma_k^{(1)}|-) = \mathrm{PG}(1, \sqrt{\langle b_k^2\rangle})$. In the update of other variational parameters, only the expectation of $\gamma_{jn}^{(0)}$ is needed, which can be calculated by $\langle\gamma_{jn}^{(0)}\rangle = \frac{1}{2\sqrt{\langle\psi_{jn}^2\rangle}}\tanh(\frac{\sqrt{\langle\psi_{jn}^2\rangle}}{2})$. The variational distribution for other parameters are in the exponential family, hence the update equations can be derived from the Gibbs sampling, which are straightforward and provided in Section 2.7.2.

In order to calculate the variational lower bound, the augmented Pólya-Gamma variables are integrated out, and the expectation of the logistic likelihood under the variational distribution is estimated by Monte Carlo integration algorithm. In the experiments 10 samples are used and were found sufficient in all cases considered.

The computational complexity of the above inference is $\mathcal{O}(NK^2)$, where N is the total number of training data points. Every iteration of VB requires a full pass through the dataset, which can be slow when applied to large datasets. Therefore, an online version of VB inference is developed, building upon the recent online implementation of latent Dirichlet allocation (Hoffman et al., 2013a). In online VB, stochastic optimization is applied to the variational objective. The key observation is that the coordinate ascent updates in VB precisely correspond to the natural gradient of the variational objective. To implement online VB, we subsample the data,

compute the gradient estimate based on the subsamples and follow the gradient with a decreasing step size.

2.3.3 Learning deep networks using SBNs

Once one layer of the deep network is trained, the model parameters in that layer are frozen (at the mean of the inferred posterior) and we can utilize the inferred hidden units as the input "data" for the training of the next higher layer. This greedy layer-wise pre-training algorithm has been shown to be effective for DBN (Hinton et al., 2006) and DBM (Salakhutdinov and Hinton, 2009a) models, and is guaranteed to improve the data likelihood under certain conditions. In the training of a deep SBN, the same strategy is employed. After finishing pre-training (sequentially for all the layers), we then "un-freeze" all the model parameters, and implement global training (refinement), in which parameters in the higher layer now can also affect the update of parameters in the lower layer.

Discriminative fine-tuning (Salakhutdinov and Hinton, 2009a) is implemented in the training of DBM by using label information. In the work presented here for the SBN, we utilize label information (when available) in a multi-task learning setting (like in Bengio et al. (2013)), where the top-layer hidden units are generated by multiple sets of bias terms, one for each label, while all the other model parameters and hidden units below the top layer are shared. This multi-task learning is only performed when generating samples from the model.

2.4 Related work

The SBN was proposed by Neal (1992), and in the original paper a Gibbs sampler was proposed to do inference. A natural extension to a variational approximation algorithm was proposed by Saul et al. (1996), using the mean field assumption. A Gaussian-field (Barber and Sollich, 1999) approach was also used for inference, by

making Gaussain approximations to the unit input. However, due to the fact that the model is not locally conjugate, all the methods mentioned above are only used for inference of distributions on the hidden variables $\mathbf{H}$, and typically model parameters $\mathbf{W}$ are learned by gradient descent.

Another route to do inference on SBNs are based on the idea of Helmholtz machines (Dayan et al., 1995), which are multi-layer belief networks with recognition models, or inference networks. These recognition models are used to approximate the true posterior. The wake-sleep algorithm (Hinton et al., 1995b) was first proposed to do inference on such recognition models. Recent work focuses on training the recognition models by maximizing a variational lower bound on the marginal log likelihood (Mnih and Gregor, 2014; Gregor et al., 2014; Kingma and Welling, 2013; Rezende et al., 2014).

In the work reported here, we focus on providing a fully Bayesian treatment on the "global" model parameters and the "local" data-dependent hidden variables. An advantage of this approach is the ability to impose shrinkage-based (near) sparsity on the model parameters. This sparsity helps regularize the model, and also aids in interpreting the learned model. The idea of Pólya-Gamma data augmentation was first proposed to do inference on Bayesian logistic regression (Polson et al., 2013a), and later extended to the inference of negative binomial regression (Zhou et al., 2012b), logistic-normal topic models (Chen et al., 2013), and discriminative relational topic models (Chen et al., 2014a). The work reported here serves as another application of this data augmentation strategy, and a first implementation of analysis of a deep-learning model in a fully Bayesian framework.

2.5 Experiments

We present experimental results on three publicly available binary datasets: MNIST, Caltech 101 Silhouettes, and OCR letters. To assess the performance of SBNs trained

FIGURE 2.2: Performance on MNIST. (Left) Training data. (Middle) Averaged synthesized samples. (Right) Learned features at the bottom layer.

using the proposed method, we show the samples generated from the model and report the average log probability that the model assigns to a test datum.

2.5.1 Experiment setup

For all the experiments below, we consider a one-hidden-layer SBN with $K = 200$ hidden units, and a two-hidden-layer SBN with each layer containing $K = 200$ hidden units. The autoregressive version of the model is denoted ARSBN. The fully visible sigmoid belief network without any hidden units is denoted FVSBN.

The SBN model is trained using both Gibbs sampling and mean field VB, as well as the proposed online VB method. The learning and inference discussed above is almost free of parameter tuning; the hyperparameters settings are given in Section 2. Similar reasonable settings on the hyperparameters yield essentially identical results. The hidden units are initialized randomly and the model parameters are initialized using an isotropic normal with standard deviation 0.1. The maximum number of iterations for VB inference is set to 40, which is large enough to observe convergence. Gibbs sampling used 40 burn-in samples and 100 posterior collection samples; while this number of samples is clearly too small to yield sufficient mixing and an accurate representation of the posteriors, it yields effective approximations to parameter means, which are used when presenting results. For online VB, the mini-

Table 2.1: Log probability of test data on MNIST dataset.

Model	Dim	Test log-prob.
SBN (online VB)	25	-138.34
RBM (CD3) (Salakhutdinov and Murray, 2008)	25	-143.20
SBN (online VB)	200	-118.12
SBN (VB)	200	-116.96
SBN.multi (VB)	200	-113.02
SBN.multi (VB)	$200-200$	-110.74
FVSBN (VB)	$-$	-100.76
ARSBN (VB)	200	-102.11
ARSBN (VB)	$200-200$	-101.19
SBN (Gibbs)	200	-94.30
SBN (NVIL) (Mnih and Gregor, 2014)	200	-113.1
SBN (NVIL) (Mnih and Gregor, 2014)	$200-200$	-99.8
DBN (Salakhutdinov and Murray, 2008)	$500-2000$	-86.22
DBM (Salakhutdinov and Hinton, 2009a)	$500-1000$	-84.62

batch size is set to 5000 with a fixed learning rate of 0.1. Local parameters were updated using 4 iterations per mini-batch, and results are shown over 20 epochs.

The properties of the deep model were explored by examining $\mathbb{E}_{p(\boldsymbol{v}|\boldsymbol{h}^{(2)})}[\boldsymbol{v}]$. Given the second hidden layer, the mean was estimated by using Monte Carlo integration. Given $\boldsymbol{h}^{(2)}$, we sample $\boldsymbol{h}^{(1)} \sim p(\boldsymbol{h}^{(1)}|\boldsymbol{h}^{(2)})$ and $\boldsymbol{v} \sim p(\boldsymbol{v}|\boldsymbol{h}^{(1)})$, repeat this procedure 1000 times to obtain the final *averaged* synthesized samples.

The test data log probabilities under VB inference are estimated using the variational lower bound. Evaluating the log probability using the Gibbs output is difficult. For simplicity, the harmonic mean estimator is utilized. As the estimator is biased (Wallach et al., 2009), we refer to the estimate as an upper bound.

ARSBN requires that the observation variables are put in some fixed order. In the experiments, the ordering was simply determined by randomly shuffling the observation vectors, and no optimization of the ordering was tried. Repeated trials with different random orderings gave empirically similar results.

2.5.2 Binarized MNIST dataset

We conducted the first experiment on the MNIST digit dataset which contains $60,000$ training and $10,000$ test images of ten handwritten digits (0 to 9), with 28×28 pixels. The binarized version of the dataset is used according to Murray and Salakhutdinov (2009). Analysis was performed on $10,000$ randomly selected training images for Gibbs and VB inference. We also examine the online VB on the whole training set.

The results for MNIST, along with baselines from the literature are shown in Table 2.1. We report the log probability estimates from our implementation of Gibbs sampling, VB and online VB using both the SBN and ARSBN model. "Dim" represents the number of hidden units in each layer, starting with the bottom one. ($\triangleright$) taken from . SBN.multi denotes SBN trained in the multi-task learning setting.

First, we examine the performance in a low-dimensional model, with $K = 25$, and the results are shown in Table 2.1. All VB methods give similar results, so only the result from the online method is shown for brevity. VB SBN shows improved performance over an RBM in this size model (Salakhutdinov and Murray, 2008).

Next, we explore an SBN with $K = 200$ hidden units. Our methods achieve similar performance to the Neural Varitional Inference and Learning (NVIL) algorithm (Mnih and Gregor, 2014), which is the current state of the art for training SBNs.

Using a second hidden layer, also with size 200, gives performance improvements in all algorithms. In VB there is an improvement of 3 nats for the SBN model when a second layer is learned. Furthermore, the VB ARSBN method gives a test log probability of -101.19. The current state of the art on this size deep sigmoid belief network is the NVIL algorithm with -99.8, which is quantitatively similar to our results. The online VB implementation gives lower bounds comparable to the batch VB, and will scale better to larger data sizes.

The TPBN prior infers the number of units needed to represent the data. The

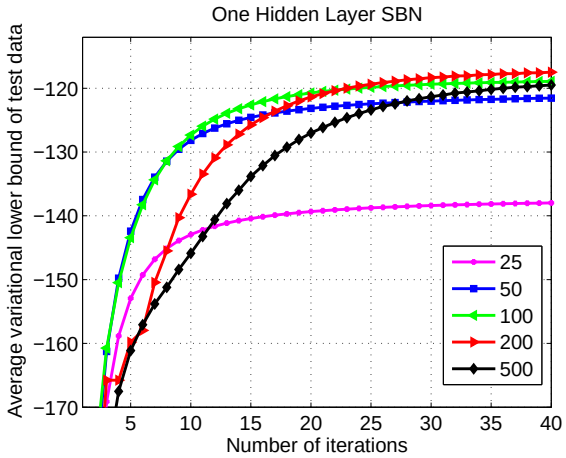


FIGURE 2.3: The impact of the number of hidden units on the average variational lower bound of test data under the one-hidden-layer SBN.

impact on the number of hidden units on the test set performance is shown in Figure 2.3. The models learned using 100, 200 and 500 hidden units achieve nearly identical test set performance, showing that our methods are not overfitting the data as the number of units increase. All models with $K > 100$ typically utilize 81 features. Thus, the TPBN prior gives "tuning-free" selection on the hidden layer size K. The learned features are shown in Figure 2.2. These features are sparse and consistent with results from sparse features learning algorithms (Lee et al., 2008).

The generated samples for MNIST are presented in Figure 2.2. The synthesized digits appear visually good and match the true data well.

We further demonstrate the ability of the model to predict missing data. For each test image, the lower half of the digit is removed and considered as missing data. Reconstructions are shown in Figure 2.4, and the model produces good completions. Because the labels of the images are uncertain when they are partially observed, the model can generate different digits than the true digit (see the transition from 9 to 0, 7 to 9 etc.).

FIGURE 2.4: Missing data prediction. For each subfigure, (Top) Original data. (Middle) Hollowed region. (Bottom) Reconstructed data.

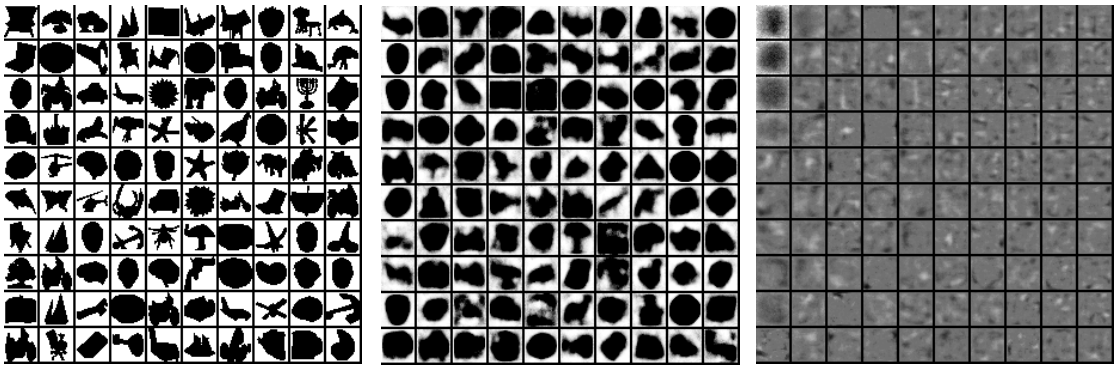


FIGURE 2.5: Performance on Caltech 101 Silhouettes. (Left) Training data. (Middle) Synthesized samples. (Right) Features at the bottom layer.

2.5.3 Caltech 101 Silhouettes dataset

The second experiment is based on the Caltech 101 Silhouettes dataset (Marlin et al., 2010), which contains 6364 training images and 2307 test images. Estimated log probabilities are reported in Table 2.2.

In this dataset, adding the second hidden layer to the VB SBN greatly improves the lower bound. Figure 2.6 demonstrates the effect of the deep model on learning. The first hidden layer improves the lower bound quickly, but saturates. When the second hidden layer is added, the model once again improves the lower bound on the test set. Global training (refinement) further enhances the performance. The two-layer model does a better job capturing the rich structure in the 101 total categories.

Table 2.2: Log probability of test data on Caltech 101 Silhouettes dataset.

Model	Dim	Test log-prob.
SBN (VB)	200	-136.84
SBN (VB)	$200 - 200$	-125.60
FVSBN (VB)	–	-96.40
ARSBN (VB)	200	-96.78
ARSBN (VB)	$200 - 200$	-97.57
RBM (Cho et al., 2013)	500	-114.75
RBM (Cho et al., 2013)	4000	-107.78

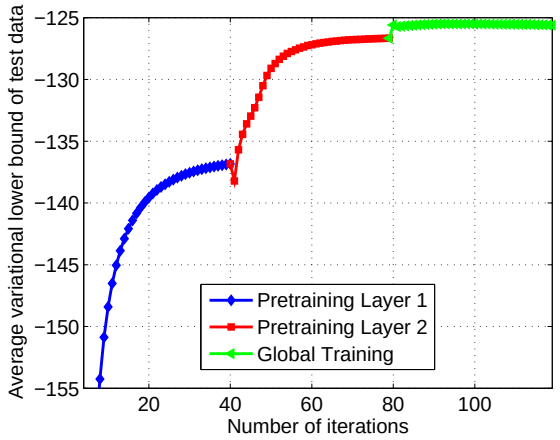


FIGURE 2.6: Average variational lower bound obtained from the SBN $200 - 200$ model on the Caltech 101 Silhouettes dataset.

For the simple dataset (MNIST with 10 categories, OCR letters with 26 categories, discussed below), this large gap is not observed.

Remarkably, our implementation of FVSBN beats the state-of-the-art results on this dataset (Cho et al., 2013) by 10 nats. Figure 2.5 shows samples drawn from the trained model; different shapes are synthesized and appear visually good.

2.5.4 OCR letters dataset

The OCR letters dataset contains 16×8 binary pixel images of 26 letters in the English alphabet. The dataset is split into $42,152$ training and $10,000$ test examples. Results are reported in Table 2.3. The proposed ARSBN with $K = 200$ hidden units achieves a lower bound of -37.97. The state-of-the-art here is a DBM with

Table 2.3: Log probability of test data on OCR letters dataset.

Model	Dim	Test log-prob.
SBN (online VB)	200	−48.71
SBN (VB)	200	−48.20
SBN (VB)	200 − 200	−47.84
FVSBN (VB)	−	−39.71
ARSBN (VB)	200	−37.97
ARSBN (VB)	200 − 200	−38.56
SBN (Gibbs)	200	−40.95
DBM (Salakhutdinov and Larochelle, 2010)	2000 − 2000	−34.24

2000 hidden units in each layer (Salakhutdinov and Larochelle, 2010). Our model gives results that are only marginally worse using a network with 100 times fewer connections.

2.6 Discussion

A simple and efficient Gibbs sampling algorithm and mean field variational Bayes approximation are developed for learning and inference of model parameters in the sigmoid belief networks. This has been implemented in a novel way by introducing auxiliary Pólya-Gamma variables. Several encouraging experimental results have been presented, enhancing the idea that the deep learning problem can be efficiently tackled in a fully Bayesian framework.

While this work has focused on binary observations, one can model real-valued data by building latent binary hierarchies as employed here, and touching the data at the bottom layer by a real-valued mapping, as has been done in related RBM models (Salakhutdinov et al., 2013). Furthermore, the logistic link function is typically utilized in the deep learning literature. The probit function and the rectified linearity are also considered in the nonlinear Gaussian belief network (Frey and Hinton, 1999). Under the Bayesian framework, by using data augmentation (Polson et al., 2011), the max-margin link could be utilized to model the non-linearities between layers when training a deep model.

2.7 Supplementary Material

2.7.1 Properties of Pólya-Gamma distribution

A random variable X has a Pólya-Gamma distribution (Polson et al., 2013a) with parameters $b > 0$ and $c \in \mathbb{R}$, denoted $X \sim \mathrm{PG}(b, c)$, if

$$X = \frac{1}{2\pi^2} \sum_{k=1}^{\infty} \frac{g_k}{(k - 1/2)^2 + c^2/(4\pi^2)}, \tag{2.17}$$

where each $g_k \sim \mathrm{Ga}(b, 1)$ is an independent gamma random variable. We have

$$\mathbb{E}[X] = \frac{b}{2c}\tanh(c/2) = \frac{b}{2c}\left(\frac{e^c - 1}{e^c + 1}\right). \tag{2.18}$$

A key observation is that binomial likelihoods parametrized by log-odds can be written as mixtures of Gaussians with respect to a Pólya-Gamma distribution. Specifically, if $\gamma \sim \mathrm{PG}(b, 0)$, $b > 0$, then

$$\frac{(e^\psi)^a}{(1 + e^\psi)^b} = 2^{-b} e^{\kappa\psi} \int_0^\infty e^{-\gamma\psi^2/2} p(\gamma) d\gamma, \tag{2.19}$$

where $\kappa = a - b/2$. And we have $\gamma|\psi \sim \mathrm{PG}(b, \psi)$. Proof is given in Polson et al. (2013a), Section 3.

The generation of the Pólya-Gamma variables is detailed in Polson et al. (2013a), Section 4. Other approximate methods for generation are discussed in the supplemental material of Zhou et al. (2012b) and Chen et al. (2013).

2.7.2 VB update equations

The VB update equations for the SBN model are listed below.

For $\gamma^{(0)}, \gamma^{(1)}$:

$$q(\gamma_{jn}^{(0)}) = \mathrm{PG}\left(1, \sqrt{\langle(\boldsymbol{w}_j^\top \boldsymbol{h}_n + c_j)^2\rangle}\right), \tag{2.20}$$

$$q(\gamma_k^{(1)}) = \mathrm{PG}\left(1, \sqrt{\langle b_k^2\rangle}\right). \tag{2.21}$$

For H: $q(h_{kn}) = \text{Ber}(\sigma(d_{kn}))$, where

$$d_{kn} = \langle b_k \rangle + \langle \boldsymbol{w}_k^\top \boldsymbol{v}_n \rangle - \frac{1}{2} \sum_{j=1}^{J} \left(\langle w_{jk} \rangle + \langle \gamma_{jn}^{(0)} \rangle (2 \langle \psi_{jn}^{\setminus k} w_{jk} \rangle + \langle w_{jk}^2 \rangle) \right) , \tag{2.22}$$

where $\psi_{jn}^{\setminus k} = \boldsymbol{w}_j^\top \boldsymbol{h}_n - w_{jk} h_{kn} + c_j$.

For W: $q(\boldsymbol{w}_j) = N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$, where

$$\boldsymbol{\Sigma}_j = \left[\sum_{n=1}^{N} \langle \gamma_{jn}^{(0)} \rangle \langle \boldsymbol{h}_n \boldsymbol{h}_n^\top \rangle + \text{diag}(\langle \boldsymbol{\zeta}_j^{-1} \rangle) \right]^{-1} , \tag{2.23}$$

$$\boldsymbol{\mu}_j = \boldsymbol{\Sigma}_j \left[\sum_{n=1}^{N} (v_{jn} - \frac{1}{2} - \langle c_j \rangle \langle \gamma_{jn}^{(0)} \rangle) \langle \boldsymbol{h}_n \rangle \right] . \tag{2.24}$$

For TPBN shrinkage:

$$q(\zeta_{jk}) = \mathcal{GIG}(0, 2\langle \xi_{jk} \rangle, \langle W_{jk}^2 \rangle) , \tag{2.25}$$

$$q(\xi_{jk}) = \text{Gamma}(1, \langle \zeta_{jk} \rangle + \langle \phi_k \rangle) , \tag{2.26}$$

$$q(\phi_k) = \text{Gamma} \left(\frac{1}{2} J + \frac{1}{2}, \langle \omega \rangle + \sum_{j=1}^{J} \langle \xi_{jk} \rangle \right) , \tag{2.27}$$

$$q(\omega) = \text{Gamma} \left(\frac{1}{2} K + \frac{1}{2}, 1 + \sum_{k=1}^{K} \langle \phi_k \rangle \right) . \tag{2.28}$$

3

Deep Poisson Factor Analysis for Topic Modeling

In this chapter, I will present deep Poisson factor analysis for topic modeling. The proposed multi-layer model employs a deep sigmoid belief network or restricted Boltzmann machine, the bottom binary layer of which selects topics for use in a Poisson factor analysis model. Scalable inference algorithms are derived by applying Bayesian conditional density filtering algorithm, and stochastic gradient thermostats.

3.1 Introduction

Considerable research effort has been devoted to developing probabilistic models for documents. In the context of topic modeling, a popular approach is latent Dirichlet allocation (LDA) (Blei et al., 2003), a directed graphical model that aims to discover latent topics (word distributions) in collections of documents that are represented in bag-of-words form. Recent work focuses on linking observed word counts in a document to latent nonnegative matrix factorization, via a Poisson distribution, termed Poisson factor analysis (PFA) (Zhou et al., 2012a). Different choices of priors on the latent nonnegative matrix factorization can lead to equivalent marginal distributions to LDA, as well as to the Focused Topic Model (FTM) of Williamson et al. (2010).

Additionally, hierarchical ("deep") tree-structured topic models have been developed by using structured Bayesian nonparametric priors, including the nested Chinese restaurant process (nCRP) (Blei et al., 2004), and the recently proposed nested hierarchical Dirichlet process (nHDP) (Paisley et al., 2015). The nCRP is limited because it requires that each document select topics from a single path in a tree, while the nHDP allows each document to access the entire tree by defining priors over a *base tree*. However, the relationship between two paths in these models is only explicitly given on shared parent nodes.

Another alternative for topic modeling is to develop undirected graphical models, such as the Replicated Softmax Model (RSM) (Salakhutdinov and Hinton, 2009b), based on a generalization of the restricted Boltzmann machine (RBM) (Hinton, 2002). Also closely related to the RBM is the neural autoregressive density estimator (DocNADE) (Larochelle and Lauly, 2012), a neural-network-based method, that has been shown to outperform the RSM.

Deep models, such as the Deep Belief Network (DBN) (Hinton et al., 2006), the Deep Boltzmann Machine (DBM) (Salakhutdinov and Hinton, 2009a), and layered Bayesian networks (Kingma and Welling, 2013; Mnih and Gregor, 2014; Rezende et al., 2014; Gan et al., 2015c) are becoming popular, as they consistently obtain state-of-the-art performances on a variety of machine learning tasks. A popular theme in this direction of work is to extend shallow topic models to deep counterparts. In such a setting, documents arise from a cascade of layers of latent variables. For instance, DBNs and DBMs have been generalized to model documents by utilizing the RBM as a building block (Hinton and Salakhutdinov, 2011; Srivastava et al., 2013).

Combining ideas from traditional Bayesian topic modeling and deep models, we propose a new deep generative model for topic modeling, in which the Bayesian PFA is employed to interact with the data at the bottom layer, while the Sigmoid Belief

Network (SBN) (Neal, 1992), a directed graphical model closely related to the RBM, is utilized to buildup binary hierarchies. Furthermore, our model is not necessarily restricted to SBN modules, and it is shown how an undirected model such as the RBM can be incorporated into the framework as well.

Compared with the original DBN and DBM, our proposed model: (*i*) tends to infer a more compact representation of the data, due to the "explaining away" effect described by Hinton et al. (2006); (*ii*) allows for more direct exploration of the effect of a single deep hidden node through ancestral sampling; and (*iii*) can be easily incorporated into larger probabilistic models in a modular fashion. Compared with the nCRP and nHDP, our proposed model only infers topics at the bottom layer, but defines a flexible prior to capture high-order relationships between topics via a deep binary hierarchical structure. In practice, this translates into better perplexities and very interesting topic correlations, although not in a tree representation as in nCRP or nHDP.

Another important contribution we present is to develop two scalable Bayesian learning algorithms for our model: one based on the recently proposed *Bayesian conditional density filtering* (BCDF) algorithm (Guhaniyogi et al., 2014), and the other based on the *stochastic gradient Nóse-Hoover thermostats* (SGNHT) algorithm (Ding et al., 2014). We extend the SGNHT by introducing additional *thermostat variables* into the dynamic system, increasing the stability and convergence when compared to the original SGNHT algorithm.

3.2 Model Formulation

3.2.1 Poisson Factor Analysis

Given a discrete matrix $\mathbf{X} \in \mathbb{Z}_+^{P \times N}$ containing counts from N documents and P words, Poisson factor analysis (Zhou et al., 2012a) assumes the entries of $\mathbf{X}$ are summations of $K < \infty$ latent counts, each produced by a latent factor (in the case

of topic modeling, a hidden topic). We represent $\mathbf{X}$ using the following factor model

$$\mathbf{X} = \text{Pois}(\mathbf{\Phi}(\mathbf{\Theta} \circ \mathbf{H}^{(1)})), \tag{3.1}$$

where $\mathbf{\Phi}$ is the factor loading matrix. Each column of $\mathbf{\Phi}$, $\boldsymbol{\phi}_k \in \triangle_P$, encodes the relative importance of each word in topic k, with $\triangle_P$ representing the P-dimensional simplex. $\mathbf{\Theta} \in \mathbb{R}_+^{K \times N}$ is the factor score matrix. Each column, $\boldsymbol{\theta}_n$, contains relative topic intensities specific to document n. $\mathbf{H}^{(1)} \in \{0,1\}^{K \times N}$ is a latent binary feature matrix. Each column, $\boldsymbol{h}_n^{(1)}$, defines a sparse set of topics associated with each document. For the single-layer PFA, the use of the superscript (1) on $\boldsymbol{h}_n^{(1)}$ is unnecessary; we introduce this notation here in preparation for the subsequent deep model, for which $\boldsymbol{h}_n^{(1)}$ will correspond to the associated first-layer latent binary units. The symbol $\circ$ represents the Hadamard, or element-wise multiplication of two matrices. The factor scores for document n are $\boldsymbol{\theta}_n \circ \boldsymbol{h}_n^{(1)}$.

A wide variety of algorithms have been developed by constructing PFAs with different prior specifications (Zhou and Carin, 2015). If $\mathbf{H}^{(1)}$ is an all-ones matrix, LDA is recovered from (3.1) by employing Dirichlet priors on $\boldsymbol{\phi}_k$ and $\boldsymbol{\theta}_n$, for $k = 1, \ldots, K$ and $n = 1, \ldots, N$, respectively. This version of LDA is referred to as Dir-PFA by Zhou et al. (2012a). For our proposed model, we construct PFAs by placing Dirichlet priors on $\boldsymbol{\phi}_k$ and gamma priors on $\boldsymbol{\theta}_n$. This is summarized as,

$$x_{pn} = \sum_{k=1}^{K} x_{pnk}\,, \quad x_{pnk} \sim \text{Pois}(\phi_{pk}\theta_{kn}h_{kn}^{(1)})\,, \tag{3.2}$$

with priors specified as $\boldsymbol{\phi}_k \sim \text{Dir}(a_\phi, \ldots, a_\phi)$, $\theta_{kn} \sim \text{Gamma}(r_k, p_n/(1-p_n))$, $r_k \sim \text{Gamma}(\gamma_0, 1/c_0)$, and $\gamma_0 \sim \text{Gamma}(e_0, 1/f_0)$.

The novelty in our model comes from the prior for the binary feature matrix $\mathbf{H}^{(1)}$. Previously, Zhou and Carin (2015) proposed a beta-Bernoulli process prior on the columns $\{\boldsymbol{h}_n^{(1)}\}_{n=1}^N$ with $p_n = 0.5$. This model was called NB-FTM, tightly related with the focused topic model (FTM) (Williamson et al., 2010). In the work

presented here, we construct $\mathbf{H}^{(1)}$ from a deep structure based on the SBN (or RBM) with binary latent units.

3.2.2 Structured Priors on the Latent Binary Matrix

The second part of our model consists of a deep structure for a binary hierarchy. To this end, we employ the SBN (or RBM). In the following we start by describing a single-layer model with SBN (or RBM), and then we generalize it to a deep model.

Modeling with the SBN We assume the latent vector for document n, $\boldsymbol{h}_n^{(1)} \in \{0,1\}^{K_1}$. This matches most of the RBM and SBN literature, for which typically the *observed* data are binary. In our model, however, these binary variables are not observed; they are hidden and related to the data through the PFA in (3.2).

To construct a structured prior, we define another hidden set of units $\boldsymbol{h}_n^{(2)} \in \{0,1\}^{K_2}$ placed at a layer "above" $\boldsymbol{h}_n^{(1)}$. The layers are related through a set of weights defined by the matrix $\mathbf{W}^{(1)} = [\boldsymbol{w}_1^{(1)} \ \ldots \ \boldsymbol{w}_{K_1}^{(1)}]^\top \in \mathbb{R}^{K_1 \times K_2}$. An SBN model has the generative process,

$$p(h_{k_2 n}^{(2)} = 1) = \sigma(c_{k_2}^{(2)}) \,, \tag{3.3}$$

$$p(h_{k_1 n}^{(1)} = 1 | \boldsymbol{h}_n^{(2)}) = \sigma\left((\boldsymbol{w}_{k_1}^{(1)})^\top \boldsymbol{h}_n^{(1)} + c_{k_1}^{(1)} \right) , \tag{3.4}$$

where $h_{k_1 n}^{(1)}$ and $h_{k_2 n}^{(2)}$ are elements of $\boldsymbol{h}_n^{(1)}$ and $\boldsymbol{h}_n^{(2)}$, respectively. The function $\sigma(x) \triangleq 1/(1+e^{-x})$ is the logistic function, and $c_{k_1}^{(1)}$ and $c_{k_2}^{(2)}$ are bias terms. The global parameters $\mathbf{W}^{(1)}$ are used to characterize the mapping from $\boldsymbol{h}_n^{(2)}$ to $\boldsymbol{h}_n^{(1)}$ for all documents.

Modeling with the RBM The SBN is closely related to the RBM, which is a Markov random field with the same bipartite structure as the SBN. The RBM defines a distribution over a binary vector that is proportional to the exponential of its *energy*,

defined (using the same notation as in SBN) as

$$E(\boldsymbol{h}_n^{(1)}, \boldsymbol{h}_n^{(2)}) = -(\boldsymbol{h}_n^{(1)})^\top \boldsymbol{c}^{(1)} - (\boldsymbol{h}_n^{(1)})^\top \mathbf{W}^{(1)} \boldsymbol{h}_n^{(2)} - (\boldsymbol{h}_n^{(2)})^\top \boldsymbol{c}^{(2)} . \tag{3.5}$$

In the experiments we consider both the deep SBN and deep RBM for representation of the latent binary units, which are connected to topic usage in a given document.

Remark An important benefit of SBNs over RBMs is that in the former sparsity or shrinkage priors can be readily imposed on the global parameters $\mathbf{W}^{(1)}$, and fully Bayesian inference can be implemented as shown in Gan et al. (2015c). The RBM relies on an approximation technique known as contrastive divergence (Hinton, 2002), for which prior specification for model parameters is limited.

3.2.3 Deep Architecture for Topic Modeling

Specifying a prior distribution on $\boldsymbol{h}_n^{(2)}$ as in (3.3) might be too restrictive in some cases. Alternatively, we can use another SBN prior for $\boldsymbol{h}_n^{(2)}$, in fact, we can add multiple layers as in Gan et al. (2015c) to obtain a deep architecture,

$$p(\boldsymbol{h}_n^{(1)}, \ldots, \boldsymbol{h}_n^{(L)}) = p(\boldsymbol{h}_n^{(L)}) \prod_{\ell=2}^{L} p(\boldsymbol{h}_n^{(\ell-1)} | \boldsymbol{h}_n^{(\ell)}), \tag{3.6}$$

where L is the number of layers, $p(\boldsymbol{h}_n^{(L)})$ is the prior for the top layer defined as in (3.3), $p(\boldsymbol{h}_n^{(\ell-1)} | \boldsymbol{h}_n^{(\ell)})$ is defined as in (3.4), and the weights $\mathbf{W}^{(\ell)} \in \mathbb{R}^{K_\ell \times K_{\ell+1}}$ and biases $\boldsymbol{c}^{(\ell)} \in \mathbb{R}^{K_\ell}$ are omitted from the conditional distributions to keep notation uncluttered. A similar deep architecture may be designed for the RBM (Salakhutdinov and Hinton, 2009a).

Instead of employing the beta-Bernoulli specification for $\boldsymbol{h}_n^{(1)}$ as in the NB-FTM, which assumes independent topic usage probabilities, we propose using (3.6) instead as the prior for $\boldsymbol{h}_n^{(1)}$, thus

$$p(\boldsymbol{x}_n, \boldsymbol{h}_n) = p(\boldsymbol{x}_n | \boldsymbol{h}_n^{(1)}) p(\boldsymbol{h}_n^{(1)}, \ldots, \boldsymbol{h}_n^{(L)}) , \tag{3.7}$$

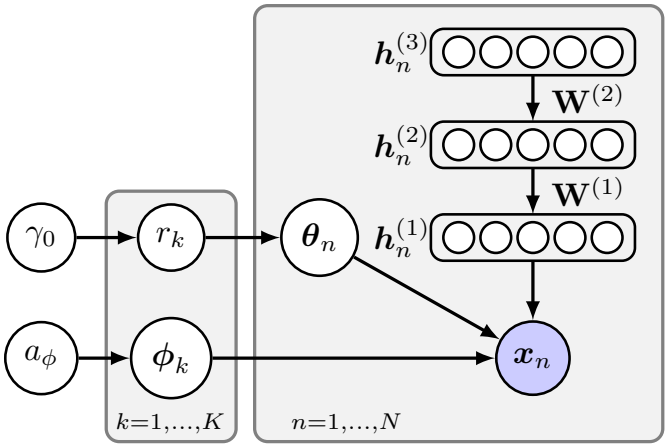


FIGURE 3.1: Graphical model for the Deep Poisson Factor Analysis with three layers of hidden binary hierarchies.

where $\boldsymbol{h}_n \triangleq \{\boldsymbol{h}_n^{(1)}, \ldots, \boldsymbol{h}_n^{(L)}\}$, and $p(\boldsymbol{x}_n|\boldsymbol{h}_n^{(1)})$ as in (3.2). The prior $p(\boldsymbol{h}_n^{(1)}|\boldsymbol{h}_n^{(2)} \ldots, \boldsymbol{h}_n^{(L)})$ can be seen as a flexible prior distribution over binary vectors that encodes high-order interactions across elements of $\boldsymbol{h}_n^{(1)}$. The graphical model for our model, Deep Poisson Factor Analysis (DPFA) is shown in Figure 3.1. The directed binary hierarchy may be replaced by a *deep Boltzmann machine*.

3.3 Scalable Posterior Inference

We focus on learning our model with fully Bayesian algorithms, however, emerging large-scale corpora prohibit standard MCMC inference algorithms to be applied directly. For example, in the experiments, we consider the *RCV1-v2* and the *Wikipedia* corpora, which contain about 800K and 10M documents, respectively. Therefore, fast algorithms for big Bayesian learning are essential. While parallel algorithms based on distributed architectures such as the *parameter server* (Ho et al., 2013; Li et al., 2014) are popular choices, in the work presented here, we focus on another direction for scaling up inference by stochastic algorithms, where mini-batches instead of the whole dataset are utilized in each iteration of the algorithms. Specifically, we develop two stochastic Bayesian inference algorithms based on Bayesian conditional

Algorithm 1 BCDF algorithm for DPFA.

Input: text documents, *i.e.*, a count matrix $\mathbf{X}$.
Initialize $\mathbf{\Psi}_g^{(0)}$ randomly and set $\mathbf{S}_g^{(0)}$ all to zero.
for $t = 1$ **to** ∞ **do**
  Get one mini-batch $\mathbf{X}^{(t)}$.
  Initialize $\mathbf{\Psi}_g^{(t)} = \mathbf{\Psi}_g^{(t-1)}$, and $\mathbf{S}_g^{(t)} = \mathbf{S}_g^{(t-1)}$.
  Initialize $\mathbf{\Psi}_l^{(t)}$ randomly.
  for $s = 1$ **to** S **do**
    Gibbs sampling for DPFA on $\mathbf{X}^{(t)}$.
    Collect samples $\mathbf{\Psi}_g^{1:S}, \mathbf{\Psi}_l^{1:S}$ and $\mathbf{S}_g^{1:S}$.
  end for
  Set $\mathbf{\Psi}_g^{(t)} = \text{mean}(\mathbf{\Psi}_g^{1:S})$, and $\mathbf{S}_g^{(t)} = \text{mean}(\mathbf{S}_g^{1:S})$.
end for

density filtering (Guhaniyogi et al., 2014) and stochastic gradient thermostats (Ding et al., 2014), both of which have theoretical guarantees in the sense of asymptotical convergence to the true posterior distribution.

3.3.1 Bayesian conditional density filtering

Bayesian conditional density filtering (BCDF) is a recently proposed stochastic algorithm for Bayesian online learning (Guhaniyogi et al., 2014), that extends Markov chain Monte Carlo (MCMC) sampling to streaming data. Sampling in BCDF proceeds by drawing from the conditional posterior distributions of model parameters, obtained by propagating surrogate conditional sufficient statistics (SCSS). In practice, we repeatedly update the SCSS using the current mini-batch and draw S samples from the conditional densities using, for example, a Gibbs sampler. This eliminates the need to load the entire dataset into memory, and provides computationally cheaper Gibbs updates. More importantly, it can be proved that BCDF leads to an approximation of the conditional distributions that produce samples from the correct target posterior asymptotically, once the entire dataset is seen (Guhaniyogi et al., 2014).

In the learning phase, we are interested in learning the global parameters $\mathbf{\Psi}_g =$

$(\{\boldsymbol{\phi}_k\}, \{r_k\}, \gamma_0, \{\mathbf{W}^{(\ell)}, \boldsymbol{c}^{(\ell)}\})$. Denote local variables as $\boldsymbol{\Psi}_l = (\boldsymbol{\Theta}, \mathbf{H}^{(\ell)})$, and let $\mathbf{S}_g$ represent the SCSS for $\boldsymbol{\Psi}_g$, the BCDF algorithm can be summarized in Algorithm 1. Specifically, we need to obtain the conditional densities, which can be readily derived granted the full local conjugacy of the proposed model. Using *dot notation* to represent marginal sums, *e.g.*, $x_{\cdot nk} \triangleq \sum_p x_{pnk}$, we can write the key conditional densities for (3.2) as (Zhou and Carin, 2015)

$$x_{pnk}|- \sim \text{Multi}(x_{pn}; \zeta_{pn1}, \ldots, \zeta_{pnK}), \tag{3.8}$$

$$\boldsymbol{\phi}_k|- \sim \text{Dir}(a_\phi + x_{1\cdot k}, \ldots, a_\phi + x_{P\cdot k}), \tag{3.9}$$

$$\theta_{kn}|- \sim \text{Gamma}(r_k h_{kn}^{(1)} + x_{\cdot nk}, p_n), \tag{3.10}$$

$$h_{kn}^{(1)}|- \sim \delta(x_{\cdot nk} = 0)\text{Ber}\left(\tfrac{\tilde{\pi}_{kn}}{\tilde{\pi}_{kn}+(1-\pi_{kn})}\right) + \delta(x_{\cdot nk} > 0), \tag{3.11}$$

where $\tilde{\pi}_{kn} = \pi_{kn}(1-p_n)^{r_k}$, $\pi_{kn} = \sigma((\boldsymbol{w}_k^{(1)})^\top \boldsymbol{h}_n^{(2)} + c_k^{(1)})$, and $\zeta_{pnk} \propto \phi_{pk}\theta_{kn}$. Additional details are provided in Section 3.7.1. For the conditional distributions of $\mathbf{W}^{(\ell)}$ and $\mathbf{H}^{(\ell)}$, we use the same data augmentation technique as in Gan et al. (2015c), where Pólya-Gamma (PG) variables $\gamma_{k_\ell n}^{(\ell)}$ (Polson et al., 2013b) are introduced for hidden unit k_ℓ in layer ℓ corresponding to observation $\boldsymbol{v}_n$. Specifically, each $\gamma_{k_\ell n}^{(\ell)}$ has conditional posterior $\text{PG}(1, (\boldsymbol{w}_{k_\ell}^{(\ell)})^\top \boldsymbol{h}_n^{(\ell+1)} + c_{k_\ell}^{(\ell)})$. If we place a Gaussian prior $N(0, \sigma^2\mathbf{I})$ on $\boldsymbol{w}_{k_\ell}^{(\ell)}$, the posterior will still be Gaussian with covariance matrix $\boldsymbol{\Sigma}_{k_\ell}^{(\ell)} = [\sum_n \gamma_{k_\ell n}^{(\ell)} \boldsymbol{h}_n^{(\ell+1)} (\boldsymbol{h}_n^{(\ell+1)})^\top + \sigma^{-2}\mathbf{I}]^{-1}$ and mean $\boldsymbol{\mu}_{k_\ell}^{(\ell)} = \boldsymbol{\Sigma}_{k_\ell}^{(\ell)}[\sum_n (h_{k_\ell n}^{(\ell)} - 1/2 - c_{k_\ell}^{(\ell)}\gamma_{k_\ell n}^{(\ell)})\boldsymbol{h}_n^{(\ell+1)}]$. Furthermore, for $\ell > 1$, the conditional distribution of $h_{k_\ell n}^{(\ell)}$ can be obtained as[1]

$$h_{k_\ell n}^{(\ell)} \sim \text{Bernoulli}\left(\sigma(d_{k_\ell n})\right), \tag{3.12}$$

[1] Here and in the rest of the paper, whenever $\ell > L$, $\boldsymbol{h}_n^{(\ell)}$ is defined as a zero vector, for conciseness.

where

$$d_{k_\ell n} = (\boldsymbol{w}_{\cdot,k_\ell}^{(\ell-1)})^\top \boldsymbol{h}_n^{(\ell-1)} + (\boldsymbol{w}_{k_\ell}^{(\ell)})^\top \boldsymbol{h}_n^{(\ell+1)} + c_{k_\ell}^{(\ell)} \tag{3.13}$$

$$- \frac{1}{2} \sum_{k_{\ell-1}} \left(w_{k_{\ell-1}k_\ell}^{(\ell-1)} + \gamma_{k_{\ell-1}n}^{(\ell-1)} (2\psi_{k_{\ell-1}n}^{\setminus k_\ell} w_{k_{\ell-1}k_\ell}^{(\ell-1)} + (w_{k_{\ell-1}k_\ell}^{(\ell-1)})^2) \right) , \tag{3.14}$$

and $\psi_{k_{\ell-1}n}^{\setminus k_\ell} = \sum_{k'_\ell \neq k_\ell} w_{k_{\ell-1}k'_\ell}^{(\ell-1)} h_{k'_\ell n}^{(\ell)} + c_{k_{\ell-1}}^{(\ell-1)}$. Note that $\boldsymbol{w}_{\cdot,k_{\ell+1}}^{(\ell)}$ and $\boldsymbol{w}_{k_\ell}^{(\ell)}$ represents the $k_{\ell+1}$th column and the transpose of the k_ℓth row of $\mathbf{W}^{(\ell)}$, respectively. As can be seen, the conditional posterior distribution of $h_{k_\ell n}^{(\ell)}$ is both related to $\boldsymbol{h}_n^{(\ell-1)}$ and $\boldsymbol{h}_n^{(\ell+1)}$.

3.3.2 Stochastic gradient thermostats

Our second learning algorithm adopts the recently proposed SGNHT for large scale Bayesian sampling (Ding et al., 2014), which is more scalable and accurate than the previous BCDF algorithm. SGNHT generalizes the *stochastic gradient Langevin dynamics* (SGLD) (Welling and Teh, 2011) and the *stochastic gradient Hamiltonian Monte Carlo* (SGHMC) (Chen et al., 2014b) by introducing momentum variables into the system, which is adaptively damped using a thermostat. The thermostat exchanges energy with the target system (*e.g.*, a Bayesian model) to maintain a constant temperature; this has the potential advantage of making the system jump out of local modes easier and reach the equilibrium state faster (Ding et al., 2014).

Specifically, let $\boldsymbol{\Psi}_g \in \mathbb{R}^M$ be model parameters[2] which corresponds to the location of particles in a physical system, $\boldsymbol{v} \in \mathbb{R}^M$ be the momentum of these particles, which are driven by stochastic forces $\tilde{f}$ defined as the negative stochastic gradient (evaluated on a subset of data) of a Bayesian posterior, *e.g.*, $\tilde{f}(\boldsymbol{\Psi}_g) \triangleq -\nabla_{\boldsymbol{\Psi}_g} \tilde{U}(\boldsymbol{\Psi}_g)$, where $\tilde{U}(\boldsymbol{\Psi}_g)$ is the negative log-posterior of a Bayesian model. The motion of the particles in the

[2] With a little abuse of notation but for conciseness, we use $\boldsymbol{\Psi}_g$ to denote the reparameterized version of the parameters (such that $\boldsymbol{\Psi}_g \in \mathbb{R}^M$) if any, required in SGNHT.

system are then defined by the following stochastic differential equations:

$$\begin{aligned} \mathrm{d}\boldsymbol{\Psi}_g = \boldsymbol{v}\mathrm{d}t\,, \quad \mathrm{d}\boldsymbol{v} = \tilde{f}(\boldsymbol{\Psi}_g)\mathrm{d}t - \xi\boldsymbol{v}\mathrm{d}t + \sqrt{D}\mathrm{d}\mathcal{W}\,, \\ \mathrm{d}\xi = \left(\tfrac{1}{M}\boldsymbol{v}^\top\boldsymbol{v} - 1\right)\mathrm{d}t\,, \end{aligned} \tag{3.15}$$

where t indexes time, $\mathcal{W}$ is the standard Wiener process, ξ is called the thermostat variable which ensures the system temperature to be constant, and D is the variance of the total noise injected into the system and is assumed to be constant.

It can be shown that under certain assumptions, the equilibrium distribution of system (3.15) corresponds to the model posterior (Ding et al., 2014). As a result, the SDE (3.15) can be solved by using the Euler-Maruyama scheme (Tuckerman, 2010), where a mini-batch of the whole data is used to evaluate the stochastic gradient $\tilde{f}$. Note only one thermostat variable ξ is used in the SDE system (3.15); this is not robust enough to control the system temperature well because of the high dimensionality of $\boldsymbol{\Psi}_g$. Based on the techniques in Ding et al. (2014), we extend the SGNHT by introducing multiple thermostat variables $(\xi_1, \cdots, \xi_M)$ into the system such that each ξ_i controls one degree of the particle momentum. Intuitively, this allows energy to be exchanged between particles and thermostats more efficiently, thus driving the system to equilibrium states more rapidly. Empirically we have also verified the superiority of the proposed modification over the original SGNHT. Formally, let $\boldsymbol{\Xi} = \mathrm{diag}(\xi_1, \xi_2, \cdots, \xi_M)$, $\mathbf{q} = \mathrm{diag}(v_1^2, \cdots, v_M^2)$, we define our proposed SGNHT using the following SDEs

$$\begin{aligned} \mathrm{d}\boldsymbol{\Psi}_g = \boldsymbol{v}\mathrm{d}t\,, \quad \mathrm{d}\boldsymbol{v} = \tilde{f}(\boldsymbol{\Psi}_g)\mathrm{d}t - \boldsymbol{\Xi}\boldsymbol{v}\mathrm{d}t + \sqrt{D}\mathrm{d}\mathcal{W}\,, \\ \mathrm{d}\boldsymbol{\Xi} = (\mathbf{q} - \mathbf{I})\,\mathrm{d}t\,, \end{aligned} \tag{3.16}$$

where $\mathbf{I}$ is the identity matrix. Interestingly, we are still able to prove that the equilibrium distribution of the above system corresponds to the model posterior.

Theorem 1. *The equilibrium distribution of the SDE system in* (3.16) *is*

$$p(\boldsymbol{\Psi}_g, \boldsymbol{v}, \boldsymbol{\Xi}) \propto \exp\left(-\frac{1}{2}\boldsymbol{v}^\top \boldsymbol{v} - U(\boldsymbol{\Psi}_g) - \frac{1}{2} tr\Big\{(\boldsymbol{\Xi} - D)^\top (\boldsymbol{\Xi} - D)\Big\}\right). \tag{3.17}$$

By Theorem 1, it is straightforward to see that the marginal distribution $p(\boldsymbol{\Psi}_g)$ of $p(\boldsymbol{\Psi}_g, \vec{v}, \boldsymbol{\Xi})$ is exactly the posterior of our Bayesian model. As a result, again we can generate approximate samples from $p(\boldsymbol{\Psi}_g, \vec{v}, \boldsymbol{\Xi})$ using the Euler-Maruyama scheme and discard the auxiliary variables $\vec{v}$ and $\boldsymbol{\Xi}$.

Learning for the SBN-based model Our SBN-based model is illustrated in Figure 3.1. In the learning phase we are interested in learning the global parameters $\boldsymbol{\Psi}_g$, the same as in BCDF. The constraints inside the parameters $\{\boldsymbol{\phi}_k\}$, *i.e.*, $\sum_p \phi_{pk} = 1$, prevent the SGNHT from being applied directly. Although we can overcome this problem by using re-parameterization methods as in Patterson and Teh (2013), we find it converges better when considering information geometry for these parameters. As a result, we use stochastic gradient Riemannian Langevin dynamics (SGRLD) (Patterson and Teh, 2013) to sample the topic-word distributions $\{\boldsymbol{\phi}_k\}$, and use the SGNHT to sample the remaining parameters. Based on the data augmentation for x_{pn} above, Section 3.3.1 shows that the posteriors of $\{\boldsymbol{\phi}_k\}$'s are Dirichlet distributions. This enables us to apply the same scheme as the SGRLD for LDA (Patterson and Teh, 2013) to sample $\{\boldsymbol{\phi}_k\}$'s.

The rest of the parameters can be straightforwardly sampled using the SGNHT algorithm. Specifically we need to calculate the stochastic gradients of $\mathbf{W}^{(\ell)}$ and $\boldsymbol{c}^{(\ell)}$ evaluated on a mini-batch of data (denote $\mathcal{D}$ as the index set of a mini-batch). Based

on the model definition in (3.6), these can be calculated as

$$\frac{\partial \tilde{U}}{\partial \boldsymbol{w}_{k_\ell}^{(\ell)}} = \frac{N}{|\mathcal{D}|} \sum_{n \in \mathcal{D}} \mathbb{E}_{\boldsymbol{h}_n^{(\ell)}, \boldsymbol{h}_n^{(\ell+1)}} \left[\left(\tilde{\sigma}_{k_\ell n}^{(\ell)} - h_{k_\ell n}^{(\ell)} \right) \boldsymbol{h}_n^{(\ell+1)} \right], \tag{3.18}$$

$$\frac{\partial \tilde{U}}{\partial c_{k_\ell}^{(\ell)}} = \frac{N}{|\mathcal{D}|} \sum_{n \in \mathcal{D}} \mathbb{E}_{\boldsymbol{h}_n^{(\ell)}, \boldsymbol{h}_n^{(\ell+1)}} \left[\tilde{\sigma}_{k_\ell n}^{(\ell)} - h_{k_\ell n}^{(\ell)} \right], \tag{3.19}$$

where $\tilde{\sigma}_{k_\ell n}^{(\ell)} = \sigma((\boldsymbol{w}_{k_\ell}^{(\ell)})^\top \boldsymbol{h}_n^{(\ell+1)} + c_{k_\ell}^{(\ell)})$, and the expectation is taken over posteriors. As in the case of LDA (Patterson and Teh, 2013), no closed-form integrations can be obtained for the above gradients, we thus use Monte Carlo integration to approximate the quantity. Specifically, given $\{\boldsymbol{w}_{k_\ell}^{(\ell)}, c_{k_\ell}^{(\ell)}\}$, we are able to collect samples of the local variables $(\boldsymbol{h}_n^{(\ell)})_{n \in \mathcal{D}}$ by running a few Gibbs steps and then using these samples to approximate the intractable integrations. Exact conditional distributions for $h_{k_\ell n}^{(\ell)}$ exist without variable augmentation, however, we found that this approach does not mix well due to the highly correlated structure of hidden variables. Instead, we sample $h_{k_\ell n}^{(\ell)}$ based on the same augmentation used in BCDF, given in (3.12).

Learning for the RBM-based model As mentioned above, our RBM-based model is recovered when replacing the SBN with the RBM in Figure 3.1. Despite minor changes in the construction, the intractable normalizer which consists of model parameters (*e.g.*, $\mathbf{W}^{(\ell)}$) prohibits exact MCMC sampling from being applied. As a result, we develop an approximate learning algorithm that alternates between sampling $(\{\boldsymbol{\phi}_k\}, \{\gamma_k\}, \gamma_0\})$ and $(\{\mathbf{W}^{(\ell)}, \boldsymbol{c}^{(\ell)}\})$. Specifically, we use the same conditional posteriors as in the SBN-based model to sample the former, but use the *contrastive divergence* algorithm (CD-1) (Hinton, 2002) for the latter. One main difference of our CD-1 algorithm *w.r.t* the original one is that the inputs (*i.e.*, $\boldsymbol{h}_n^{(1)}$) are hidden variables. To make the CD-1 work, conditioned on other model parameters, we first sample $\boldsymbol{h}_n^{(1)}$ using the posterior given in Section 3.3.1, then conditioned on

$\boldsymbol{h}_n^{(1)}$, we apply the original CD-1 algorithm to calculate the approximate gradients for $(\{\mathbf{W}^{(\ell)}, \boldsymbol{c}^{(\ell)}\})$, which are then used for a gradient descent step in SGNHT. In fact, the CD-1 is also a stochastic approximate algorithm, discussed in Yuille (2005), making it naturally fit into our SGNHT framework.

3.3.3 Discussion

Both the BCDF and SGNHT are stochastic inference algorithms, allowing the models to be applied to large-scale data. In terms of ease of implementation, BCDF beats SGNHT in most cases, especially when the model is conjugate and the domain of parameters is constrained (*e.g.*, variables on a simplex). However, in general BCDF is more restrictive than SGNHT. For example, BCDF prefers the conditional densities for all the parameters, which is unavailable in some cases. Furthermore, BCDF has the limitation of being unable to deal with some *big models* where the number of model parameters is large, for instance, when the dimension of the hidden variables from the SBN in our model is huge. Finally, the conditions for BCDF to converge to the true posterior are more restricted. Altogether, these reasons make SGNHT more robust than BCDF.

3.4 Related Work

In traditional Bayesian topic models, topic correlations are typically modeled with shallow structures, *e.g.*, the correlated topic model (Blei and Lafferty, 2007) with correlation between topic proportions imposed via the logistic normal distribution. There exist also some work on hierarchical ("deep") correlation modeling, *e.g.*, the hierarchical Dirichlet process (Teh et al., 2006), which models topic proportions hierarchically via a stack of DPs. The nested Chinese restaurant process (Blei et al., 2004) (nCRP) models topic hierarchies by defining a tree structure prior based on the Chinese restaurant process, and the nested hierarchical Dirichlet process (Paisley

et al., 2015) extends the nCRP by allowing each document to be able to access all the paths in the tree. One major difference between these models and ours is that they focus on discovering topic hierarchies instead of modeling general topic correlations.

In the deep learning community, topic models are mostly built using the RBM as a building block. For example, Hinton and Salakhutdinov (2011) and Maaloe et al. (2015) extended the DBN for topic modeling, while a deep version of the RSM was proposed by Srivastava et al. (2013). More recent work focuses on employing deep directed generative models for topic modeling, *e.g.*, deep exponential families (Ranganath et al., 2015), a class of latent variable models extending the DBN by defining the distribution of hidden variables in each layer using the exponential family, instead of the restricted Bernoulli distribution.

In terms of learning and inference algorithms, most of existing Bayesian topic models rely on MCMC methods or variational Bayes algorithms, which are impractical when dealing with large scale data. Therefore, stochastic variational inference algorithms have been developed (Hoffman et al., 2010; Mimno et al., 2012; Wang and Blei, 2012; Hoffman et al., 2013b). Although scalable and usually fast converging, one unfavorable shortcoming of stochastic variational inference algorithms is the mean-field assumption on the approximate posterior.

Another direction for scalable Bayesian learning relies on the theory from stochastic differential equations (SDE). Specifically, Welling and Teh (2011) proposed the first stochastic MCMC algorithm, called *stochastic gradient Langevin dynamics* (SGLD), for large scale Bayesian learning. In order to make the learning faster, Patterson and Teh (2013) generalized SGLD by considering information geometry (Girolami and Calderhead, 2011; Byrne and Girolami, 2013) of model posteriors. Furthermore, Chen et al. (2014b) generalized the SGLD by a second-order Langevin dynamic, called *stochastic gradient Hamiltonian Monte Carlo* (SGHMC). This is the stochastic version of the well known Hamiltonian MCMC sampler. One problem with SGHMC

is that the unknown stochastic noise needs to be estimated to make the sampler correct, which is impractical. *Stochastic gradient thermostats* algorithms (SGNHT) overcome this problem by introducing the thermostat into the algorithm, such that the unknown stochastic noise could be adaptively absorbed into the thermostat, making the sampler asymptotically exact. Given the advantages of the SGNHT, in this paper we extend it to a multiple thermostats setting, where each thermostat exchanges energy with a degree of freedom of the system. Empirically we show that our extension improves on the original algorithm.

Since the publication of this paper, some more advanced modeling approaches have been proposed, including deep Poisson factor modeling (Henao et al., 2015, 2016), Poisson gamma belief network (Zhou et al., 2015, 2016), deep latent Dirichlet allocation (Cong et al., 2017), and topic compositional neural language model (Wang et al., 2017).

3.5 Experiments

3.5.1 Datasets and Setups

We present experimental results on three publicly available corpora: a relatively small, *20 Newsgroups*, a moderately large, Reuters Corpus Volume I (*RCV1-v2*), and a large one, *Wikipedia*. The first two corpora are the same as those used in Srivastava et al. (2013). Specifically, the *20 Newsgroups* corpus contains 18,845 documents with a total of 0.7M words and a vocabulary size of 2K. The data was partitioned chronologically into 11,314 training and 7,531 test documents. The *RCV1-v2* corpus contains 804,414 newswire articles. There are 103 topics that form a tree hierarchy. After preprocessing, we are left with about 75M words, with a vocabulary size of 10K. We randomly select 794,414 documents for training and 10,000 for testing. Finally, we downloaded 10M random documents from *Wikipedia* using scripts provided in Hoffman et al. (2010) and randomly selected 1K documents for testing. As in

Hoffman et al. (2010); Patterson and Teh (2013), a vocabulary size of 7,702 was taken from the top 10K words in Project Gutenberg texts.

The DPFA model consisting of SBN is denoted as DPFA-SBN, while its RBM counterpart is denoted DPFA-RBM. The performance of DPFA is compared to that of the following models: LDA (Blei et al., 2003), NB-FTM (Zhou and Carin, 2015), nHDP (Paisley et al., 2015) and RSM (Salakhutdinov and Hinton, 2009b).

For all the models considered, we calculate the predictive perplexities on the test set as follows: holding the global model parameters fixed, for each test document we randomly partition the words into a 80/20% split. We learn document-specific "local" parameters using the 80% portion, and then calculate the predictive perplexities on the remaining 20% subset. Evaluation details are provided in Section 3.7.2.

For *20 Newsgroups* and *RCV1-v2* corpora, we use 2,000 mini-batches for burn-in followed by 1,500 collection samples to calculate test perplexities; while for the *Wikipedia* dataset, 3,500 mini-batches are used for burn-in. The mini-batch size for all stochastic algorithms is set to 100. To choose good parameters for SGNHT, *e.g.*, the step size and the variance of the injected noise, we randomly choose about 10% documents from the training data as validation set. For BCDF, 100 MCMC iterations are evaluated for each mini-batch, with the first 60 samples discarded. We set the hyperparameters of DPFA as $a_\phi = 1.01, c_0 = e_0 = 1, f_0 = 0.01$, and $p_n = 0.5$. The RSM is trained using convergence-divergence with step size 5 and a maximum of 10,000 iterations. For nHDP, we use the publicly available code from Paisley et al. (2015), in which stochastic variational Bayes (sVB) inference is implemented.

3.5.2 Quantitative Evaluation

20 Newsgroups The results for the *20 Newsgroups* corpus are shown in Table 3.1. Perplexities are reported for our implementation of Gibbs sampling, BCDF and SGNHT, and the four considered competing methods. "Dim" represents the number

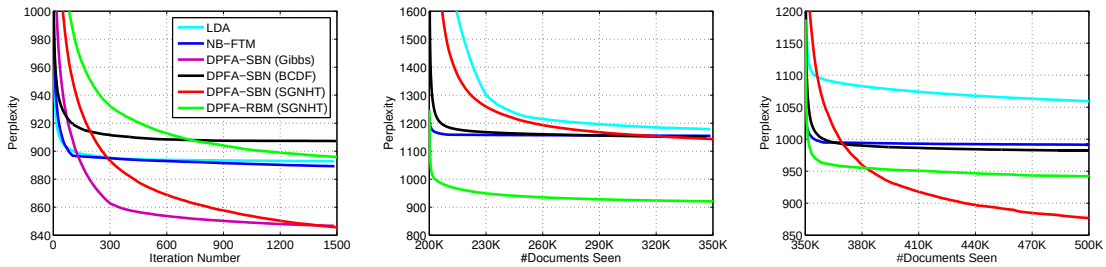


FIGURE 3.2: Predictive perplexities on the test set as a function of training documents seen. (Left) *20 News*. (Middle) *RCV1-v2*. (Right) *Wikipedia*.

Table 3.1: Test perplexities for *20 Newsgroups*.

MODEL	METHOD	DIM	PERP.
DPFA-SBN-t	GIBBS	128-64-32	**827**
DPFA-SBN	GIBBS	128-64-32	**846**
DPFA-SBN	SGNHT	128-64-32	**846**
DPFA-RBM	SGNHT	128-64-32	896
DPFA-SBN	BCDF	128-64-32	905
DPFA-SBN	GIBBS	128-64	851
DPFA-SBN	SGNHT	128-64	850
DPFA-RBM	SGNHT	128-64	893
DPFA-SBN	BCDF	128-64	896
LDA	GIBBS	128	893
NB-FTM	GIBBS	128	887
RSM	CD5	128	877
NHDP	SVB	$(10,10,5)^{\diamond}$	889

of hidden units in each layer, starting from the bottom. DPFA-SBN-t represents the DPFA-SBN model with Student's t prior on $\mathbf{W}^{(\ell)}$. ($\diamond$) represents the *base tree* size in nHDP. First, we examine the performance of different inference algorithms. As can be seen, for the same size model, *e.g.*, 128-64-32 (128 topics and 32 binary nodes on the top of the three-layer model), SGNHT can achieve essentially the same performance as Gibbs sampling, while BCDF is more likely to get trapped in a local mode. Next, we explore the advantage of employing deep models. Using three layers instead of two gives performance improvements in almost all the algorithms. In Gibbs sampling, there is an improvement of 36 units for the DPFA-SBN model, when a second layer is learned (NB-FTM is the one-hidden-layer DPFA). Adding the

Table 3.2: Test perplexities on *RCV1-v2* and *Wikipedia*.

Model	Method	Dim	RCV	Wiki
DPFA-SBN	SGNHT	1024-512-256	964	**770**
DPFA-SBN	SGNHT	512-256-128	1073	799
DPFA-SBN	SGNHT	128-64-32	1143	876
DPFA-RBM	SGNHT	128-64-32	**920**	942
DPFA-SBN	BCDF	128-64-32	1149	986
LDA	BCDF	128	1179	1059
NB-FTM	BCDF	128	1155	991
RSM	CD5	128	1171	1001
nHDP	sVB	(10,5,5)	1041	932

third hidden layer further improves the test perplexity.

Adding a sparsity-encouraging prior on $\mathbf{W}^{(\ell)}$ acts as a more stringent regularization that prevents overfitting, when compared with the commonly used L_2 norm (Gaussian prior). Furthermore, shrinkage priors have the effect of being able to effectively switch off the elements of $\mathbf{W}^{(\ell)}$, which benefits interpretability and helps to infer the number of units needed to represent the data. In our experiment, we observe that the DPFA-SBN model with the Student's t prior on $\mathbf{W}^{(\ell)}$ achieves a better test perplexity when compared with its counterpart without shrinkage.

RCV1-v2 & Wiki We present results for the *RCV1-v2* and *Wikipedia* corpora in Table 3.2. Direct Gibbs sampling in such a (big-data) setting is prohibitive, and is thus not discussed. First, we explore the effect of utilizing a larger deep network. For our DPFA-SBN model using the SGNHT algorithm, we observe that making the network 8 time larger in each hidden layer decreases the test perplexities by 155 and 84 units on *RCV1-v2* and *Wikipedia*, respectively. This demonstrates the ability of our stochastic inference algorithm to scale up both in terms of model and corpus size.

Both SBN and RBM can be utilized as the building block in our deep specification. For the *RCV1-v2* corpus, our best result is obtained by utilizing a three-layer

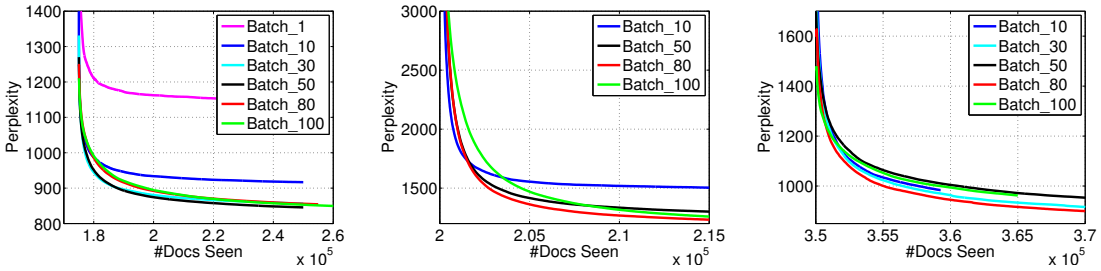


FIGURE 3.3: Test perplexities *w.r.t.* mini-batch sizes on the three corpora. (Left) *20 Newsgroups*. (Middle) *RCV1-v2*. (Right) *Wikipedia*.

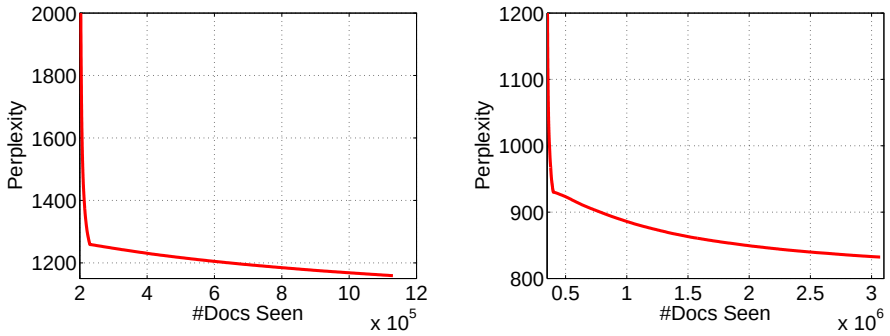


FIGURE 3.4: Test perplexities as a function of training documents seen. (Left) *RCV1-v2*. (Right) *Wikipedia*.

deep Boltzmann machine. However, for the *20 Newsgroups* and *Wikipedia* corpora, with the same size model, we found empirically that the deep SBN achieves better performance.

Compared with nHDP, our DPFA models define a more flexible prior on topic interactions, and therefore in practice we also consistently achieve better perplexity results. We further show test perplexities as a function of documents processed during model learning in Figure 3.2. The number of hidden units in each layer is 128, 64, 32, respectively. As can be seen, performance smoothly improves as the amount of data processed increases.

T1	T3	T8	T9	T10	T14	T15	T19	T21	T24
year	people	group	world	evidence	game	israel	software	files	team
hit	real	groups	country	claim	games	israeli	modem	file	players
runs	simply	reading	countries	people	win	jews	port	ftp	player
good	world	newsgroup	germany	argument	cup	arab	mac	program	play
season	things	pro	nazi	agree	hockey	jewish	serial	format	teams
T25	T26	T29	T40	T41	T43	T50	T54	T55	T64
god	fire	people	wrong	image	boston	problem	card	windows	turkish
existence	fbi	life	doesn	program	toronto	work	video	dos	armenian
exist	koresh	death	jim	application	montreal	problems	memory	file	armenians
human	children	kill	agree	widget	chicago	system	mhz	win	turks
atheism	batf	killing	quote	color	pittsburgh	fine	bit	ms	armenia
T65	T69	T78	T81	T91	T94	T112	T118	T120	T126
truth	window	drive	makes	question	code	children	people	men	sex
true	server	disk	power	answer	mit	father	make	women	sexual
point	display	scsi	make	means	comp	child	person	man	cramer
fact	manager	hard	doesn	true	unix	mother	things	hand	gay
body	client	drives	part	people	source	son	feel	world	homosexual

FIGURE 3.5: Top words from the 30 topics corresponding to the graph in Figure 3.6, learned by DPFA-SBN from the *20Newsgroup* corpus.

3.5.3 Sensitivity analysis

We examined the sensitivity of the model performance with respect to batch sizes in SGNHT on the three corpora considered. The results are shown in Figure 3.3. We found that overall performance, both convergence speed and test perplexity, suffer considerably when the batch size is smaller than 10 documents. However, for batch sizes larger than 50 (100 for *RCV1-v2*) we obtain performances comparable to those shown in Tables 3.1 and 3.2.

We run the SGNHT algorithm on the *RCV1-v2* and *Wikipedia* datasets long enough so that the whole corpora can be traversed. The results are shown in Figure 3.4. As can be seen, performance smoothly improves as the amount of data processed increases.

3.5.4 Visualization

We can obtain a visual representation of the topic structure implied by the deep component of our DPFA model by computing correlations between topics using the weight matrices, $\mathbf{W}^{(\ell)}$, learned by DPFA-SBN, *i.e*, we evaluate the covariance $\mathbf{W}^{(1)}\mathbf{W}^{(2)}(\mathbf{W}^{(1)}\mathbf{W}^{(2)})^\top$, then scale it accordingly. Figure 3.6 shows a graph for a subset of 30 topics (nodes), where edge thickness encodes correlation coefficients and

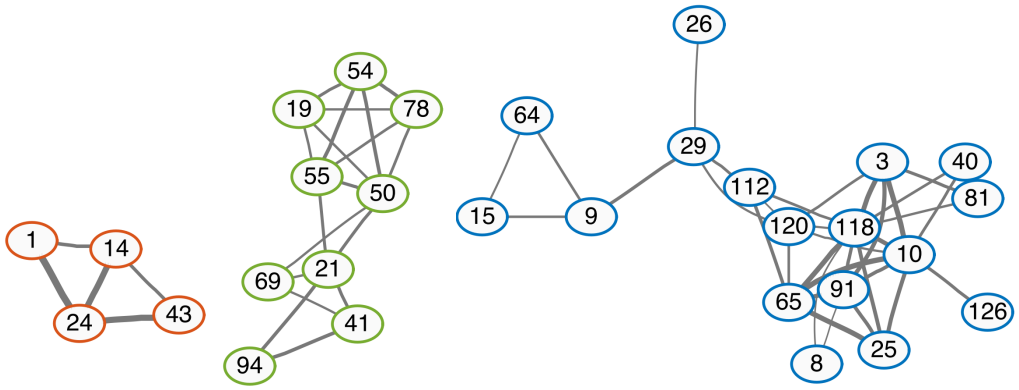


FIGURE 3.6: Graphs induced by the correlation structure learned by DPFA-SBN for the *20 Newsgroups*.

we have chosen, to ease visualization, to show only coefficients larger than 0.85. Each node represents a topic with top words shown in Figure 3.5. In addition, Figure 3.5 shows the top words for each topic depicted in Figure 3.6. We see three very interesting subgraphs representing different categories, namely, sports, computers and politics/law.

3.6 Discussion

We have presented the Deep Poisson Factor Analysis model, an extension of PFA, that models the high-order interactions between topics, via a deep binary hierarchical structure, employing SBNs and RBMs. To address large-scale datasets, two stochastic Bayesian learning algorithms were developed. Experimental results on several corpora show that the proposed approach obtains superior test perplexities and reveals interesting topic structures.

While this work has focused on unsupervised topic modeling, one can extend the model into a supervised version by joint modeling of the text with associated labels via latent binary features as in Zhang and Carin (2012). Furthermore, as mentioned in Section 4.5, *global-local* shrinkage priors (Polson and Scott, 2012) will encourage a large proportion of the elements of $\mathbf{W}^{(\ell)}$ to be shrunk close to zero. By setting the

number of hidden units to a reasonably large value, this provides a natural way to let the model select automatically the number of features actually needed.

3.7 Supplementary Material

3.7.1 Conditional Densities used in BCDF

Using *dot notation* to represent marginal sums, *e.g.*, $x_{\cdot nk} \triangleq \sum_p x_{pnk}$, we can write the conditional densities for DPFA as (Zhou and Carin, 2015)

$$
\begin{aligned}
x_{pnk}|- &\sim \text{Multi}(x_{pn}; \zeta_{pn1}, \ldots, \zeta_{pnK}) \,, \qquad (3.20)\\
\boldsymbol{\phi}_k|- &\sim \text{Dir}(a_\phi + x_{1\cdot k}, \ldots, a_\phi + x_{P\cdot k}) \,,\\
\theta_{kn}|- &\sim \text{Gamma}(r_k h^{(1)}_{kn} + x_{\cdot nk}, p_n) \,,\\
r_k|- &\sim \text{Gamma}\left(\gamma_0 + \sum_{n=1}^{N} l_{kn}, \frac{1}{c_0 - \sum_{n=1}^{N} h^{(1)}_{kn} \ln(1-p_n)}\right) \,, \qquad (3.21)\\
\gamma_0|- &\sim \text{Gamma}\left(e_0 + \sum_{k=1}^{K} l'_k, \frac{1}{f_0 - \sum_{k=1}^{K} \ln(1-p'_k)}\right) \,, \qquad (3.22)\\
h^{(1)}_{kn}|- &\sim \delta(x_{\cdot nk}=0)\text{Ber}\left(\frac{\pi_{kn}(1-p_n)^{r_k}}{\pi_{kn}(1-p_n)^{r_k} + (1-\pi_{kn})}\right)\\
&\quad + \delta(x_{\cdot nk} > 0) \,,
\end{aligned}
$$

where

$$
l_{kn}|- \sim \text{CRT}\left(x_{\cdot nk}, r_k h^{(1)}_{kn}\right) \,, \quad l'_k|- \sim \text{CRT}\left(\sum_{n=1}^{N} l_{kn}, \gamma_0\right) \,, \qquad (3.23)
$$

$$
\zeta_{pnk} = \frac{\phi_{pk}\theta_{kn}}{\sum_{k=1}^{K} \phi_{pk}\theta_{kn}} \,, \quad p'_k = \frac{-\sum_{n=1}^{N} h^{(1)}_{kn} \ln(1-p_n)}{c_0 - \sum_{n=1}^{N} h^{(1)}_{kn} \ln(1-p_n)} \,, \qquad (3.24)
$$

$$
\pi_{kn} = \sigma\left((\boldsymbol{w}^{(1)}_k)^\top \boldsymbol{h}^{(2)}_n + c^{(1)}_k\right) \,. \qquad (3.25)
$$

CRT represents the Chinese Restaurant Table distribution. A CRT random variable $l \sim \text{CRT}(m, r)$ can be generated with the summation of independent Bernoulli

random variables as (Zhou and Carin, 2015)

$$l = \sum_{n=1}^{m} b_n, \quad b_n \sim \text{Ber}\left(\frac{r}{n-1+r}\right). \tag{3.26}$$

3.7.2 Evaluation Details on Perplexities

For each test document, we randomly partition the words into a 80/20% split. We learn document-specific local parameters using the 80% portion, and then calculate the predictive perplexities on the remaining 20% subset, denoted as $\mathbf{Y}$. For the PFA-based models, the test perplexity is calculated as (Zhou et al., 2012a)

$$\exp\left(-\frac{1}{y_{\cdot\cdot}}\sum_{p=1}^{P}\sum_{n=1}^{N} y_{pn}\log\frac{\sum_{s=1}^{S}\sum_{k=1}^{K}\phi_{pk}^{s}\theta_{kn}^{s}}{\sum_{s=1}^{S}\sum_{p=1}^{P}\sum_{k=1}^{K}\phi_{pk}^{s}\theta_{kn}^{s}}\right), \tag{3.27}$$

where S is the total number of collected samples, $y_{\cdot\cdot} = \sum_{p=1}^{P}\sum_{n=1}^{N} y_{pn}$ and y_{pn} is an element of matrix $\mathbf{Y}$.

The conditional distribution of $\boldsymbol{y}_n$ given $\boldsymbol{h}_n$, in the Replicated Softmax model (RSM) is specified as

$$\boldsymbol{y}_n \sim \text{Multi}(D_n; \boldsymbol{\beta}_n), \tag{3.28}$$

$$\beta_{pn} = \frac{\exp(\boldsymbol{w}_p^{\top}\boldsymbol{h}_n + c_p)}{\sum_{p'=1}^{P}\exp(\boldsymbol{w}_{p'}^{\top}\boldsymbol{h}_n + c_{p'})}, \tag{3.29}$$

where $\boldsymbol{y}_n$ is the nth column of $\mathbf{Y}$, and $D_n = \sum_{p=1}^{P} y_{pn}$. $\mathbf{W} = [\boldsymbol{w}_1, \ldots . \boldsymbol{w}_P]^{\top} \in \mathbb{R}^{P\times K}$ is the mapping from $\boldsymbol{h}_n$ to $\boldsymbol{y}_n$, and $\boldsymbol{c} = [c_1, \ldots . c_P]^{\top} \in \mathbb{R}^{P\times 1}$ is the bias term. Based on this, the predictive test perplexity for RSM can be calculated as

$$\exp\left(-\frac{1}{y_{\cdot\cdot}}\sum_{p=1}^{P}\sum_{n=1}^{N} y_{pn}\log\beta_{pn}\right). \tag{3.30}$$

4

Temporal Sigmoid Belief Networks for Sequence Modeling

In this chapter, I will present temporal sigmoid belief networks for sequential data. The proposed model is a sequential stack of sigmoid belief networks (SBNs). Each SBN has a contextual hidden state, inherited from the previous SBNs in the sequence, and is used to regulate its hidden bias. We show in the experiments that the proposed model has the capacity to synthesize various sequences.

4.1 Introduction

Considerable research has been devoted to developing probabilistic models for high-dimensional time-series data, such as video and music sequences, motion capture data, and text streams. Among them, Hidden Markov Models (HMMs) (Rabiner and Juang, 1986) and Linear Dynamical Systems (LDS) (Kalman, 1963) have been widely studied, but they may be limited in the type of dynamical structures they can model. An HMM is a mixture model, which relies on a single multinomial variable to represent the history of a time-series. To represent N bits of information about

the history, an HMM could require 2^N distinct states. On the other hand, real-world sequential data often contain complex non-linear temporal dependencies, while a LDS can only model simple linear dynamics.

Another class of time-series models, which are potentially better suited to model complex probability distributions over high-dimensional sequences, relies on the use of Recurrent Neural Networks (RNNs) (Hermans and Schrauwen, 2013; Martens and Sutskever, 2011; Pascanu et al., 2013; Graves, 2013), and variants of a well-known *undirected* graphical model called the Restricted Boltzmann Machine (RBM) (Taylor et al., 2006; Sutskever and Hinton, 2007; Sutskever et al., 2009; Boulanger-Lewandowski et al., 2012; Mittelman et al., 2014). One such variant is the Temporal Restricted Boltzmann Machine (TRBM) (Sutskever and Hinton, 2007), which consists of a sequence of RBMs, where the state of one or more previous RBMs determine the biases of the RBM in the current time step. Learning and inference in the TRBM is non-trivial. The approximate procedure used in Sutskever and Hinton (2007) is heuristic and not derived from a principled statistical formalism.

Recently, deep *directed* generative models (Kingma and Welling, 2013; Mnih and Gregor, 2014; Rezende et al., 2014; Gan et al., 2015c) are becoming popular. A *directed* graphical model that is closely related to the RBM is the Sigmoid Belief Network (SBN) (Neal, 1992). In the work presented here, we introduce the Temporal Sigmoid Belief Network (TSBN), which can be viewed as a temporal stack of SBNs, where each SBN has a contextual hidden state that is inherited from the previous SBNs and is used to adjust its hidden-units bias. Based on this, we further develop a deep dynamic generative model by constructing a hierarchy of TSBNs. This can be considered as a deep SBN (Gan et al., 2015c) with temporal feedback loops on each layer. Both stochastic and deterministic hidden layers are considered.

Compared with previous work, our model: (i) can be viewed as a generalization of an HMM with distributed hidden state representations, and with a deep architecture;

(*ii*) can be seen as a generalization of a LDS with complex non-linear dynamics; (*iii*) can be considered as a probabilistic construction of the traditionally deterministic RNN; (*iv*) is closely related to the TRBM, but it has a fully generative process, where data are readily generated from the model using ancestral sampling; (*v*) can be utilized to model different kinds of data, *e.g.*, binary, real-valued and counts.

The "explaining away" effect described in Hinton et al. (2006) makes inference slow, if one uses traditional inference methods. Another important contribution we present here is to develop fast and scalable learning and inference algorithms, by introducing a recognition model (Kingma and Welling, 2013; Mnih and Gregor, 2014; Rezende et al., 2014), that learns an inverse mapping from observations to hidden variables, based on a loss function derived from a variational principle. By utilizing the recognition model and variance-reduction techniques from Mnih and Gregor (2014), we achieve fast inference both at training and testing time.

4.2 Model Formulation

4.2.1 Temporal Sigmoid Belief Networks

The proposed Temporal Sigmoid Belief Network (TSBN) model is a sequence of SBNs arranged in such way that at any given time step, the SBN's biases depend on the state of the SBNs in the previous time steps. Specifically, assume we have a length-T binary visible sequence, the tth time step of which is denoted $\boldsymbol{v}_t \in \{0,1\}^M$. The TSBN describes the joint probability as

$$p_{\boldsymbol{\theta}}(\mathbf{V},\mathbf{H}) = p(\boldsymbol{h}_1)p(\boldsymbol{v}_1|\boldsymbol{h}_1)\cdot\prod_{t=2}^{T} p(\boldsymbol{h}_t|\boldsymbol{h}_{t-1},\boldsymbol{v}_{t-1})\cdot p(\boldsymbol{v}_t|\boldsymbol{h}_t,\boldsymbol{v}_{t-1}), \tag{4.1}$$

where $\mathbf{V}=[\boldsymbol{v}_1,\ldots,\boldsymbol{v}_T]$, $\mathbf{H}=[\boldsymbol{h}_1,\ldots,\boldsymbol{h}_T]$, and each $\boldsymbol{h}_t \in \{0,1\}^J$ represents the hidden state corresponding to time step t. For $t=1,\ldots,T$, each conditional distribution

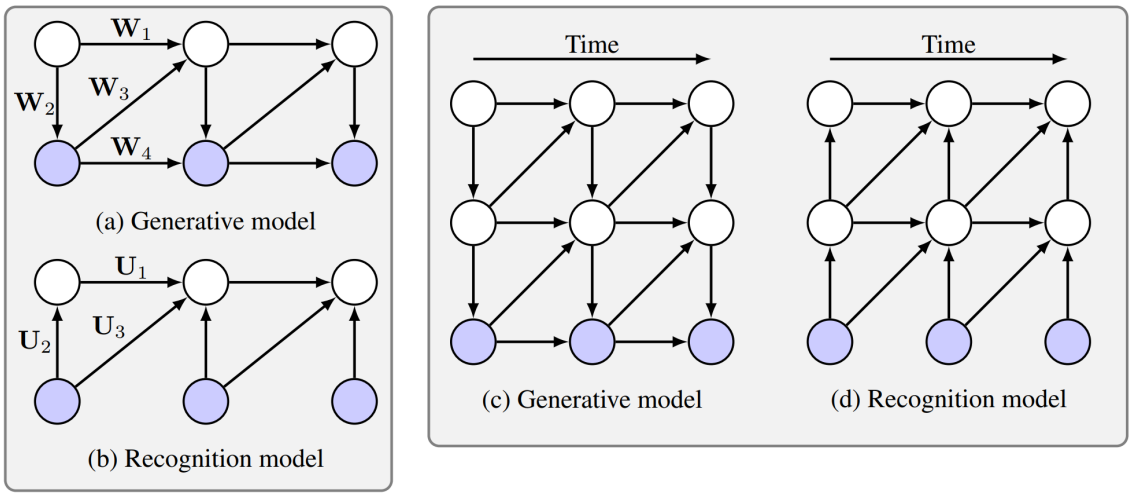


FIGURE 4.1: Graphical model for the Deep Temporal Sigmoid Belief Network.

in (4.1) is expressed as

$$p(h_{jt} = 1|\boldsymbol{h}_{t-1}, \boldsymbol{v}_{t-1}) = \sigma(\boldsymbol{w}_{1j}^\top \boldsymbol{h}_{t-1} + \boldsymbol{w}_{3j}^\top \boldsymbol{v}_{t-1} + b_j), \tag{4.2}$$

$$p(v_{mt} = 1|\boldsymbol{h}_t, \boldsymbol{v}_{t-1}) = \sigma(\boldsymbol{w}_{2m}^\top \boldsymbol{h}_t + \boldsymbol{w}_{4m}^\top \boldsymbol{v}_{t-1} + c_m), \tag{4.3}$$

where $\boldsymbol{h}_0$ and $\boldsymbol{v}_0$, needed for the prior model $p(\boldsymbol{h}_1)$ and $p(\boldsymbol{v}_1|\boldsymbol{h}_1)$, are defined as zero vectors, respectively, for conciseness. The model parameters, $\boldsymbol{\theta}$, are specified as $\mathbf{W}_1 \in \mathbb{R}^{J\times J}$, $\mathbf{W}_2 \in \mathbb{R}^{M\times J}$, $\mathbf{W}_3 \in \mathbb{R}^{J\times M}$, $\mathbf{W}_4 \in \mathbb{R}^{M\times M}$. For $i = 1, 2, 3, 4$, $\boldsymbol{w}_{ij}$ is the transpose of the jth row of $\mathbf{W}_i$, and $\boldsymbol{c} = [c_1, \ldots, c_M]^\top$ and $\boldsymbol{b} = [b_1, \ldots, b_J]^\top$ are bias terms. The graphical model for the TSBN is shown in Figure 4.1(a).

By setting $\mathbf{W}_3$ and $\mathbf{W}_4$ to be zero matrices, the TSBN can be viewed as a Hidden Markov Model (Rabiner and Juang, 1986) with an exponentially large state space, that has a compact parameterization of the transition and the emission probabilities. Specifically, each hidden state in the HMM is represented as a *one-hot* length-J vector, while in the TSBN, the hidden states can be any length-J binary vector. We note that the transition matrix is highly structured, since the number of parameters is only quadratic *w.r.t.* J. Compared with the TRBM (Sutskever and Hinton, 2007), our TSBN is fully directed, which allows for *fast sampling* of "fantasy" data from

the inferred model.

4.2.2 TSBN Variants

Modeling real-valued data The model above can be readily extended to model real-valued sequence data, by substituting (4.3) with $p(\boldsymbol{v}_t|\boldsymbol{h}_t, \boldsymbol{v}_{t-1}) = \mathcal{N}(\boldsymbol{\mu}_t, \text{diag}(\boldsymbol{\sigma}_t^2))$, where

$$\mu_{mt} = \boldsymbol{w}_{2m}^\top \boldsymbol{h}_t + \boldsymbol{w}_{4m}^\top \boldsymbol{v}_{t-1} + c_m, \quad \log \sigma_{mt}^2 = (\boldsymbol{w}'_{2m})^\top \boldsymbol{h}_t + (\boldsymbol{w}'_{4m})^\top \boldsymbol{v}_{t-1} + c'_m, \qquad (4.4)$$

and μ_{mt} and σ_{mt}^2 are elements of $\boldsymbol{\mu}_t$ and $\boldsymbol{\sigma}_t^2$, respectively. $\mathbf{W}'_2$ and $\mathbf{W}'_4$ are of the same size of $\mathbf{W}_2$ and $\mathbf{W}_4$, respectively. Compared with the Gaussian TRBM (Sutskever et al., 2009), in which σ_{mt} is fixed to 1, our formalism uses a diagonal matrix to parameterize the variance structure of $\boldsymbol{v}_t$.

Modeling count data We also introduce an approach for modeling time-series data with count observations, by replacing (4.3) with $p(\boldsymbol{v}_t|\boldsymbol{h}_t, \boldsymbol{v}_{t-1}) = \prod_{m=1}^{M} y_{mt}^{v_{mt}}$, where

$$y_{mt} = \frac{\exp(\boldsymbol{w}_{2m}^\top \boldsymbol{h}_t + \boldsymbol{w}_{4m}^\top \boldsymbol{v}_{t-1} + c_m)}{\sum_{m'=1}^{M} \exp(\boldsymbol{w}_{2m'}^\top \boldsymbol{h}_t + \boldsymbol{w}_{4m'}^\top \boldsymbol{v}_{t-1} + c_{m'})}\,. \qquad (4.5)$$

This formulation is related to the Replicated Softmax Model (RSM) described in Salakhutdinov and Hinton (2009b), however, our approach uses a *directed* connection from the binary hidden variables to the visible counts, while also learning the dynamics in the count sequences.

Furthermore, rather than assuming that $\boldsymbol{h}_t$ and $\boldsymbol{v}_t$ only depend on $\boldsymbol{h}_{t-1}$ and $\boldsymbol{v}_{t-1}$, in the experiments, we also allow for connections from the past n time steps of the hidden and visible states, to the current states, $\boldsymbol{h}_t$ and $\boldsymbol{v}_t$. A sliding window is then used to go through the sequence to obtain n frames at each time. We refer to n as the *order* of the model.

4.2.3 Deep Architecture for Sequence Modeling with TSBNs

Learning the sequential dependencies with the shallow model in (4.1)-(4.3) may be restrictive. Therefore, we propose two deep architectures to improve its representational power: (i) adding stochastic hidden layers; (ii) adding deterministic hidden layers. The graphical model for the deep TSBN is shown in Figure 4.1(c). Specifically, we consider a deep TSBN with hidden layers $\boldsymbol{h}_t^{(\ell)}$ for $t = 1, \ldots, T$ and $\ell = 1, \ldots, L$. Assume layer ℓ contains $J^{(\ell)}$ hidden units, and denote the visible layer $\boldsymbol{v}_t = \boldsymbol{h}_t^{(0)}$ and let $\boldsymbol{h}_t^{(L+1)} = \mathbf{0}$, for convenience. In order to obtain a proper generative model, the top hidden layer $\boldsymbol{h}^{(L)}$ contains stochastic binary hidden variables.

For the middle layers, $\ell = 1, \ldots, L-1$, if stochastic hidden layers are utilized, the generative process is expressed as $p(\boldsymbol{h}_t^{(\ell)}) = \prod_{j=1}^{J^{(\ell)}} p(h_{jt}^{(\ell)} | \boldsymbol{h}_t^{(\ell+1)}, \boldsymbol{h}_{t-1}^{(\ell)}, \boldsymbol{h}_{t-1}^{(\ell-1)})$, where each conditional distribution is parameterized via a logistic function, as in (4.3). If deterministic hidden layers are employed, we obtain $\boldsymbol{h}_t^{(\ell)} = f(\boldsymbol{h}_t^{(\ell+1)}, \boldsymbol{h}_{t-1}^{(\ell)}, \boldsymbol{h}_{t-1}^{(\ell-1)})$, where $f(\cdot)$ is chosen to be a rectified linear function. Although the differences between these two approaches are minor, learning and inference algorithms can be quite different, as shown in Section 4.3.3.

4.3 Scalable Learning and Inference

Computation of the exact posterior over the hidden variables in (4.1) is intractable. Approximate Bayesian inference, such as Gibbs sampling or mean-field variational Bayes (VB) inference, can be implemented (Gan et al., 2015c; Neal, 1992). However, Gibbs sampling is very inefficient, due to the fact that the conditional posterior distribution of the hidden variables does not factorize. The mean-field VB indeed provides a fully factored variational posterior, but this technique increases the gap between the bound being optimized and the true log-likelihood, potentially resulting in a poor fit to the data. To allow for tractable and scalable inference and parameter

learning, without loss of the flexibility of the variational posterior, we apply the *Neural Variational Inference and Learning* (NVIL) algorithm described in Mnih and Gregor (2014).

4.3.1 Variational Lower Bound Objective

We are interested in training the TSBN model, $p_{\boldsymbol{\theta}}(\mathbf{V}, \mathbf{H})$, described in (4.1), with parameters $\boldsymbol{\theta}$. Given an observation $\mathbf{V}$, we introduce a fixed-form distribution, $q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})$, with parameters $\boldsymbol{\phi}$, that approximates the true posterior distribution, $p(\mathbf{H}|\mathbf{V})$. We then follow the variational principle to derive a lower bound on the marginal log-likelihood, expressed as[1]

$$\mathcal{L}(\mathbf{V}, \boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})}[\log p_{\boldsymbol{\theta}}(\mathbf{V}, \mathbf{H}) - \log q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})]\,. \tag{4.6}$$

We construct the approximate posterior $q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})$ as a recognition model. By using this, we avoid the need to compute variational parameters per data point; instead we compute a set of parameters $\boldsymbol{\phi}$ used for all $\mathbf{V}$. In order to achieve fast inference, the recognition model is expressed as

$$q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V}) = q(\boldsymbol{h}_1|\boldsymbol{v}_1) \cdot \prod_{t=2}^{T} q(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{v}_t, \boldsymbol{v}_{t-1})\,, \tag{4.7}$$

and each conditional distribution is specified as

$$q(h_{jt} = 1|\boldsymbol{h}_{t-1}, \boldsymbol{v}_t, \boldsymbol{v}_{t-1}) = \sigma(\boldsymbol{u}_{1j}^\top \boldsymbol{h}_{t-1} + \boldsymbol{u}_{2j}^\top \boldsymbol{v}_t + \boldsymbol{u}_{3j}^\top \boldsymbol{v}_{t-1} + d_j)\,, \tag{4.8}$$

where $\boldsymbol{h}_0$ and $\boldsymbol{v}_0$, for $q(\boldsymbol{h}_1|\boldsymbol{v}_1)$, are defined as zero vectors. The recognition parameters $\boldsymbol{\phi}$ are specified as $\mathbf{U}_1 \in \mathbb{R}^{J\times J}$, $\mathbf{U}_2 \in \mathbb{R}^{J\times M}$, $\mathbf{U}_3 \in \mathbb{R}^{J\times M}$. For $i = 1, 2, 3$, $\boldsymbol{u}_{ij}$ is the transpose of the jth row of $\mathbf{U}_i$, and $\boldsymbol{d} = [d_1, \ldots, d_J]^\top$ is the bias term. The graphical model is shown in Figure 4.1(b).

The recognition model defined in (4.8) has the same form as in the approximate inference used for the TRBM (Sutskever and Hinton, 2007). Exact inference for

[1] This lower bound is equivalent to the marginal log-likelihood if $q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V}) = p(\mathbf{H}|\mathbf{V})$.

our model consists of a forward and backward pass through the entire sequence, that requires the traversing of each possible hidden state. Our feedforward approximation allows the inference procedure to be fast and implemented in an online fashion.

4.3.2 Parameter Learning

To optimize (4.6), we utilize Monte Carlo methods to approximate expectations and stochastic gradient descent (SGD) for parameter optimization. The gradients can be expressed as

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathbf{V}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})}[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{V},\mathbf{H})], \tag{4.9}$$

$$\nabla_{\boldsymbol{\phi}}\mathcal{L}(\mathbf{V}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})}[(\log p_{\boldsymbol{\theta}}(\mathbf{V},\mathbf{H}) - \log q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})) \times \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})]. \tag{4.10}$$

Specifically, in the TSBN model, if we define $\hat{v}_{mt} = \sigma(\boldsymbol{w}_{2m}^\top \boldsymbol{h}_t + \boldsymbol{w}_{4m}^\top \boldsymbol{v}_{t-1} + c_m)$ and $\hat{h}_{jt} = \sigma(\boldsymbol{u}_{1j}^\top \boldsymbol{h}_{t-1} + \boldsymbol{u}_{2j}^\top \boldsymbol{v}_t + \boldsymbol{u}_{3j}^\top \boldsymbol{v}_{t-1} + d_j)$, the gradients for $\boldsymbol{w}_{2m}$ and $\boldsymbol{u}_{2j}$ can be calculated as

$$\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{V},\mathbf{H})}{\partial w_{2mj}} = \sum_{t=1}^{T}(v_{mt} - \hat{v}_{mt}) \cdot h_{jt}, \;\; \frac{\partial \log q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})}{\partial u_{2jm}} = \sum_{t=1}^{T}(h_{jt} - \hat{h}_{jt}) \cdot v_{mt}. \tag{4.11}$$

Other update equations, along with the learning details for the TSBN variants in Section 4.2.2, are provided in Section 4.7.1. We observe that the gradients in (4.9) and (4.10) share many similarities with the *wake-sleep* algorithm (Hinton et al., 1995b). Wake-sleep alternates between updating $\boldsymbol{\theta}$ in the wake phase and updating $\boldsymbol{\phi}$ in the sleep phase. The update of $\boldsymbol{\theta}$ is based on the samples generated from $q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})$, and is identical to (4.9). However, in contrast to (4.10), the recognition parameters $\boldsymbol{\phi}$ are estimated from samples generated by the model, *i.e.*, $\nabla_{\boldsymbol{\phi}}\mathcal{L}(\mathbf{V}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{V},\mathbf{H})}[\nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})]$. This update does not optimize the same objective as in (4.9), hence the wake-sleep algorithm is not guaranteed to converge (Mnih and Gregor, 2014).

Inspecting (4.10), we see that we are using $l_{\boldsymbol{\phi}}(\mathbf{V},\mathbf{H}) = \log p_{\boldsymbol{\theta}}(\mathbf{V},\mathbf{H}) - \log q_{\boldsymbol{\phi}}(\mathbf{H}|\mathbf{V})$ as the *learning signal* for the recognition parameters $\boldsymbol{\phi}$. The expectation of this

learning signal is exactly the lower bound (4.6), which is easy to evaluate. However, this tractability makes the estimated gradients of the recognition parameters very noisy. In order to make the algorithm practical, we employ the variance reduction techniques proposed in Mnih and Gregor (2014), namely: (i) centering the learning signal, by subtracting the data-independent baseline and the data-dependent baseline; (ii) variance normalization, by dividing the centered learning signal by a running estimate of its standard deviation. The data-dependent baseline is implemented using a neural network. Additionally, RMSprop (Tieleman and Hinton, 2012), a form of SGD where the gradients are adaptively rescaled by a running average of their recent magnitude, were found in practice to be important for fast convergence; thus utilized throughout all the experiments. The outline of the NVIL algorithm is shown in Algorithm 2 (reproduced from Mnih and Gregor (2014)). $C_{\boldsymbol{\lambda}}(\boldsymbol{v}_t)$ represents the data-dependent baseline, and $\alpha = 0.8$ throughout the experiments.

Algorithm 2 Compute gradient estimates for the model parameters and recognition parameters.

$\Delta\boldsymbol{\theta} \leftarrow 0, \Delta\boldsymbol{\phi} \leftarrow 0, \Delta\boldsymbol{\lambda} \leftarrow 0$
$\mathcal{L} \leftarrow 0$
for $t \leftarrow 1$ **to** T **do**
  $\boldsymbol{h}_t \sim q_{\boldsymbol{\phi}}(\boldsymbol{h}_t|\boldsymbol{v}_t)$
  $l_t \leftarrow \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t) - \log q_{\boldsymbol{\phi}}(\boldsymbol{h}_t|\boldsymbol{v}_t)$
  $\mathcal{L} \leftarrow \mathcal{L} + l_t$
  $l_t \leftarrow l_t - C_{\boldsymbol{\lambda}}(\boldsymbol{v}_t)$
end for
$c_b \leftarrow \text{mean}(l_1, \ldots, l_T)$
$v_b \leftarrow \text{variance}(l_1, \ldots, l_T)$
$c \leftarrow \alpha c + (1 - \alpha)c_b$
$v \leftarrow \alpha v + (1 - \alpha)v_b$
for $t \leftarrow 1$ **to** T **do**
  $l_t \leftarrow \frac{l_t - c}{\max(1, \sqrt{v})}$
  $\Delta\boldsymbol{\theta} \leftarrow \Delta\boldsymbol{\theta} + \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)$
  $\Delta\boldsymbol{\phi} \leftarrow \Delta\boldsymbol{\phi} + l_t \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{h}_t|\boldsymbol{v}_t)$
  $\Delta\boldsymbol{\lambda} \leftarrow \Delta\boldsymbol{\lambda} + l_t \nabla_{\boldsymbol{\lambda}} C_{\boldsymbol{\lambda}}(\boldsymbol{v}_t)$
end for

4.3.3 Extension to deep models

The recognition model corresponding to the deep TSBN is shown in Figure 4.1(d). Two kinds of deep architectures are discussed in Section 4.2.3. We illustrate the difference of their learning algorithms in two respects: (i) the calculation of the lower bound; and (ii) the calculation of the gradients.

The top hidden layer is stochastic. If the middle hidden layers are also stochastic, the calculation of the lower bound is more involved, compared with the shallow model; however, the gradient evaluation remain simple as in (4.11). On the other hand, if deterministic middle hidden layers (*i.e.*, recurrent neural networks) are employed, the lower bound objective will stay the same as a shallow model, since the only stochasticity in the generative process lies in the top layer; however, the gradients have to be calculated recursively through the *back-propagation through time* algorithm (Werbos, 1990).

4.4 Related Work

The RBM has been widely used as building block to learn the sequential dependencies in time-series data, *e.g.*, the conditional-RBM-related models (Taylor et al., 2006; Taylor and Hinton, 2009), and the temporal RBM (Sutskever and Hinton, 2007). To make *exact* inference possible, the recurrent temporal RBM was also proposed (Sutskever et al., 2009), and further extended to learn the dependency structure within observations (Mittelman et al., 2014).

In the work reported here, we focus on modeling sequences based on the SBN (Neal, 1992), which recently has been shown to have the potential to build deep generative models (Mnih and Gregor, 2014; Gan et al., 2015c,d). Our work serves as another extension of the SBN that can be utilized to model time-series data. Similar ideas have also been considered in Henderson and Titov (2010) and Hinton et al.

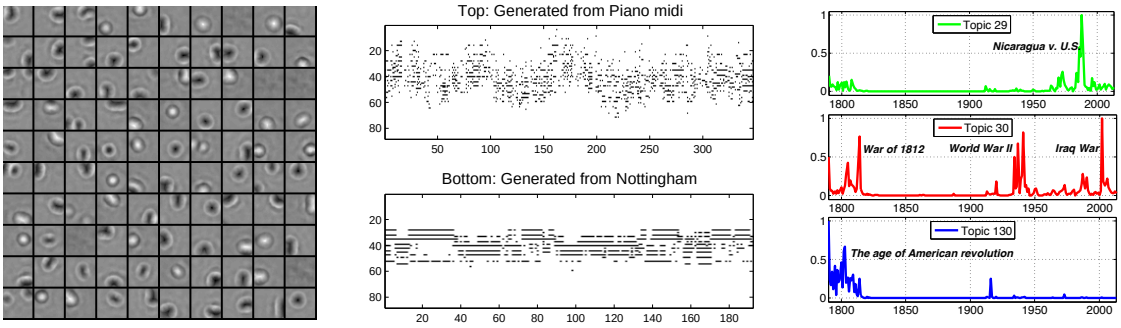


FIGURE 4.2: (Left) Dictionaries learned on the bouncing balls. (Middle) Generated polyphonic music. (Right) Time evolving for 3 topics learned on STU.

(1995a). However, in Henderson and Titov (2010), the authors focus on grammar learning, and use a feed-forward approximation of the mean-field VB to carry out the inference; while in Hinton et al. (1995a), the wake-sleep algorithm was developed. We apply the model in a different scenario, and develop a fast and scalable inference algorithm, based on the idea of training a recognition model by leveraging the stochastic gradient of the variational bound.

There exist two main methods for the training of recognition models. The first one, termed Stochastic Gradient Variational Bayes (SGVB), is based on a reparameterization trick (Kingma and Welling, 2013; Rezende et al., 2014), which can be only employed in models with continuous latent variables, *e.g.*, the variational auto-encoder (Kingma and Welling, 2013) and all the recent recurrent extensions of it (Bayer and Osendorfer, 2014; Fabius et al., 2014; Chung et al., 2015). The second one, called Neural Variational Inference and Learning (NVIL), is based on the log-derivative trick (Mnih and Gregor, 2014), which is more general and can also be applicable to models with discrete random variables. The NVIL algorithm has been previously applied to the training of SBN in Mnih and Gregor (2014). Our approach serves as a new application of this algorithm for a SBN-based time-series model.

4.5 Experiments

We present experimental results on four publicly available datasets: the bouncing balls (Sutskever et al., 2009), polyphonic music (Boulanger-Lewandowski et al., 2012), motion capture (Taylor et al., 2006) and state-of-the-Union (Han et al., 2014). To assess the performance of the TSBN model, we show sequences generated from the model, and report the average log-probability that the model assigns to a test sequence, and the average squared one-step-ahead prediction error per frame.

The TSBN model with $\mathbf{W}_3 = \mathbf{0}$ and $\mathbf{W}_4 = \mathbf{0}$ is denoted Hidden Markov SBN (HMSBN), the deep TSBN with stochastic hidden layer is denoted DTSBN-S, and the deep TSBN with deterministic hidden layer is denoted DTSBN-D.

Model parameters were initialized by sampling randomly from $\mathcal{N}(\mathbf{0}, 0.001^2\mathbf{I})$, except for the bias parameters, that were initialized as 0. The TSBN model is trained using a variant of RMSprop (Graves, 2013), with momentum of 0.9, and a constant learning rate of 10^{-4}. The decay over the root mean squared gradients is set to 0.95. The maximum number of iterations we use is 10^5. The gradient estimates were computed using a single sample from the recognition model. The only regularization we used was a weight decay of 10^{-4}. The data-dependent baseline was implemented by using a neural network with a single hidden layer with 100 tanh units.

For the prediction of $\boldsymbol{v}_t$ given $\boldsymbol{v}_{1:t-1}$, we (*i*) first obtain a sample from $q_{\boldsymbol{\phi}}(\boldsymbol{h}_{1:t-1}|\boldsymbol{v}_{1:t-1})$; (*ii*) calculate the conditional posterior $p_{\boldsymbol{\theta}}(\boldsymbol{h}_t|\boldsymbol{h}_{1:t-1}, \boldsymbol{v}_{1:t-1})$ of the current hidden state; (*iii*) make a prediction for $\boldsymbol{v}_t$ using $p_{\boldsymbol{\theta}}(\boldsymbol{v}_t|\boldsymbol{h}_{1:t}, \boldsymbol{v}_{1:t-1})$. On the other hand, synthesizing samples is conceptually simper. Sequences can be readily generated from the model using ancestral sampling.

Table 4.1: Average prediction error for the bouncing balls dataset.

Model	Dim	Order	Pred. Err.
DTSBN-s	100-100	2	**2.79** $\pm$ 0.39
DTSBN-d	100-100	2	2.99 $\pm$ 0.42
TSBN	100	4	3.07 $\pm$ 0.40
TSBN	100	1	9.48 $\pm$ 0.38
RTRBM (Mittelman et al., 2014)	3750	1	3.88 $\pm$ 0.33
SRTRBM (Mittelman et al., 2014)	3750	1	3.31 $\pm$ 0.33

4.5.1 Bouncing balls dataset

We conducted the first experiment on synthetic videos of 3 bouncing balls, where pixels are binary valued. We followed the procedure in Sutskever et al. (2009), and generated 4000 videos for training, and another 200 videos for testing. Each video is of length 100 and of resolution 30×30.

The dictionaries learned using the HMSBN are shown in Figure 4.2(Left). Compared with previous work (Sutskever et al., 2009; Boulanger-Lewandowski et al., 2012), our learned bases are more spatially localized. In Table 4.1, we compare the average squared prediction error per frame over the 200 test videos, with recurrent temporal RBM (RTRBM) and structured RTRBM (SRTRBM). As can be seen, our approach achieves better performance compared with the baselines in the literature. Furthermore, we observe that a high-order TSBN reduces the prediction error significantly, compared with an order-one TSBN. This is due to the fact that by using a high-order TSBN, more information about the past is conveyed. We also examine the advantage of employing deep models. Using stochastic, or deterministic hidden layer improves performances.

4.5.2 Motion capture dataset

In this experiment, we used the CMU motion capture dataset, that consists of measured joint angles for different motion types. We used the 33 running and walking sequences of subject 35 (23 walking sequences and 10 running sequences). We fol-

Table 4.2: Average prediction error obtained for the motion capture dataset.

Model	Walking	Running
DTSBN-s	**4.40** ± 0.28	**2.56** ± 0.40
DTSBN-d	4.62 ± 0.01	2.84 ± 0.01
TSBN	5.12 ± 0.50	4.85 ± 1.26
HMSBN	10.77 ± 1.15	7.39 ± 0.47
ss-SRTRBM (Mittelman et al., 2014)	8.13 ± 0.06	5.88 ± 0.05
g-RTRBM (Mittelman et al., 2014)	14.41 ± 0.38	10.91 ± 0.27

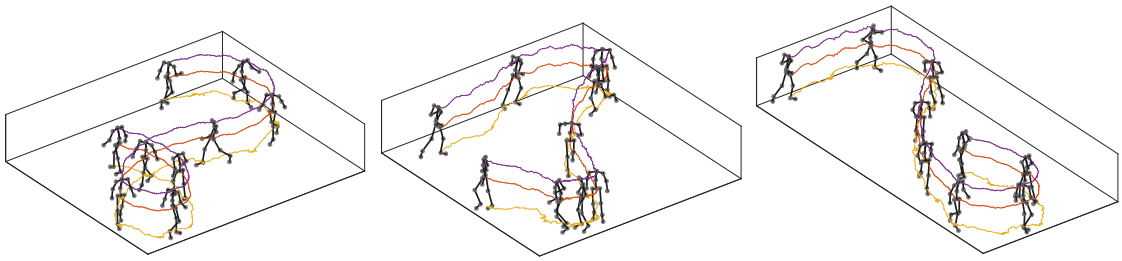


FIGURE 4.3: Generated motion trajectories. (Left) Walking. (Middle) Running-running-walking. (Right) Running-walking.

lowed the preprocessing procedure of Mittelman et al. (2014), after which we were left with 58 joint angles. We partitioned the 33 sequences into training and testing set: the first of which had 31 sequences, and the second had 2 sequences (one walking and another running). We averaged the prediction error over 100 trials, as reported in Table 4.2. The TSBN we implemented is of size 100 in each hidden layer and order 1. It can be seen that the TSBN-based models improves over the Gaussian (G-)RTRBM and the spike-slab (SS-)SRTRBM significantly.

Another popular motion capture dataset is the MIT dataset. To further demonstrate the directed, generative nature of our model, we give our trained HMSBN model different initializations, and show generated, synthetic data and the transitions between different motion styles in Figure 4.3. These generated data are readily produced from the model and demonstrate realistic behavior. The smooth trajectories are walking movements, while the vibrating ones are running. Corresponding video files (AVI) are provided as mocap 1, 2 and 3, which can be downloaded from `https:`

Table 4.3: Test log-likelihood for the polyphonic music dataset.

Model	Piano.	Nott.	Muse.	JSB.
TSBN	-7.98	-3.67	-6.81	-7.48
RNN-NADE (Boulanger-Lewandowski et al., 2012)	**-7.05**	**-2.31**	**-5.60**	**-5.56**
RTRBM (Boulanger-Lewandowski et al., 2012)	-7.36	-2.62	-6.35	-6.35
RNN (Boulanger-Lewandowski et al., 2012)	-8.37	-4.46	-8.13	-8.71

`//drive.google.com/drive/u/0/folders/0B1HR6m3IZSO_SWt0aS1oYmlneDQ`.

4.5.3 Polyphonic music dataset

The third experiment is based on four different polyphonic music sequences of piano (Boulanger-Lewandowski et al., 2012), *i.e.*, Piano-midi.de (Piano), Nottingham (Nott), MuseData (Muse) and JSB chorales (JSB). Each of these datasets are represented as a collection of 88-dimensional binary sequences, that span the whole range of piano from A0 to C8.

The samples generated from the trained HMSBN model are shown in Figure 4.2 (Middle). As can be seen, different styles of polyphonic music are synthesized. The corresponding MIDI files are provided as music 1 and 2, which can be downloaded from `https://drive.google.com/drive/u/0/folders/0B1HR6m3IZSO_SWt0aS1oYmlneDQ`. Our model has the ability to learn basic harmony rules and local temporal coherence. However, long-term structure and musical melody remain elusive. The variational lower bound, along with the estimated log-likelihood in Boulanger-Lewandowski et al. (2012), are presented in Table 4.3. The TSBN we implemented is of size 100 and order 1. Empirically, adding layers did not improve performance on this dataset, hence no such results are reported. The results of RNN-NADE and RTRBM (Boulanger-Lewandowski et al., 2012) were obtained by only 100 runs of the annealed importance sampling, which has the potential to overestimate the true log-likelihood. Our variational lower bound provides a more conservative estimate. Though, our performance is still better than that of RNN.

Table 4.4: Average prediction precision for STU.

Model	Dim	MP	PP
HMSBN	25	$\mathbf{0.327} \pm 0.002$	0.353 ± 0.070
DHMSBN-s	25-25	0.299 ± 0.001	$\mathbf{0.378} \pm 0.006$
GP-DPFA (Acharya et al., 2015)	100	0.223 ± 0.001	0.189 ± 0.003
DRFM (Acharya et al., 2015)	25	0.217 ± 0.003	0.177 ± 0.010

4.5.4 State of the Union dataset

The State of the Union (STU) dataset contains the transcripts of $T = 225$ US State of the Union addresses, from 1790 to 2014. Two tasks are considered, *i.e.*, prediction and dynamic topic modeling.

Prediction The prediction task is concerned with estimating the held-out words. We employ the setup in Acharya et al. (2015). After removing stop words and terms that occur fewer than 7 times in one document or less than 20 times overall, there are 2375 unique words. The entire data of the last year is held-out. For the documents in the previous years, we randomly partition the words of each document into 80%/20% split. The model is trained on the 80% portion, and the remaining 20% held-out words are used to test the prediction at each year. The words in both held-out sets are ranked according to the probability estimated from (4.5).

To evaluate the prediction performance, we calculate the precision @top-Mas in Acharya et al. (2015), which is given by the fraction of the top-M words, predicted by the model, that matches the true ranking of the word counts. $M = 50$ is used. Two recent works are compared, GP-DPFA (Acharya et al., 2015) and DRFM (Han et al., 2014). The results are summarized in Table 4.4. Our model is of order 1. The column MP denotes the mean precision over all the years that appear in the training set. The column PP denotes the predictive precision for the final year. Our model achieves significant improvements in both scenarios.

Table 4.5: Top 6 most probable words associated with the STU topics.

Topic #29	Topic #30	Topic #130	Topic #64	Topic #70	Topic #74
family	officer	government	generations	Iraqi	Philippines
budget	civilized	country	generation	Qaida	islands
Nicaragua	warfare	public	recognize	Iraq	axis
free	enemy	law	brave	Iraqis	Nazis
future	whilst	present	crime	AI	Japanese
freedom	gained	citizens	race	Saddam	Germans

Dynamic Topic Modeling The setup described in Han et al. (2014) is employed, and the number of topics is 200. To understand the temporal dynamic per topic, three topics are selected and the normalized probability that a topic appears at each year are shown in Figure 4.2 (Right). Their associated top 6 words per topic are shown in Table 4.5. The learned trajectory exhibits different temporal patterns across the topics. Clearly, we can identify jumps associated with some key historical events. For instance, for Topic 29, we observe a positive jump in 1986 related to military and paramilitary activities in and against Nicaragua brought by the U.S. Topic 30 is related with war, where the War of 1812, World War II and Iraq War all spike up in their corresponding years. In Topic 130, we observe consistent positive jumps from 1890 to 1920, when the American revolution was taking place. Three other interesting topics are also shown in Table 4.5. Topic 64 appears to be related to education, Topic 70 is about Iraq, and Topic 74 is Axis and World War II. We note that the words for these topics are explicitly related to these matters.

4.6 Conclusion

We have presented the Deep Temporal Sigmoid Belief Networks, an extension of SBN, that models the temporal dependencies in high-dimensional sequences. To allow for scalable inference and learning, an efficient variational optimization algorithm is developed. Experimental results on several datasets show that the proposed approach

obtains superior predictive performance, and synthesizes interesting sequences.

In this work, we have investigated the modeling of different types of data individually. One interesting future work is to combine them into a unified framework for dynamic multi-modality learning. Furthermore, we can use high-order optimization methods to speed up inference (Fan et al., 2015).

4.7 Supplementary Material

4.7.1 Learning and Inference Details on TSBN

In order to implement the NVIL algorithm described in Mnih and Gregor (2014), we need to calculate the lower bound and also the gradients. Specifically, we have the variational lower bound $\mathcal{L} = \sum_{t=1}^{T} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{h}|\boldsymbol{v})}[l_t]$, where l_t is expressed as

$$l_t = \sum_{j=1}^{J} \left(\psi_{jt}^{(1)} h_{jt} - \log(1 + \exp(\psi_{jt}^{(1)})) \right) + \sum_{m=1}^{M} \left(\psi_{mt}^{(2)} v_{mt} - \log(1 + \exp(\psi_{mt}^{(2)})) \right) \quad (4.12)$$

$$- \left[\sum_{j=1}^{J} \left(\psi_{jt}^{(3)} h_{jt} - \log(1 + \exp(\psi_{jt}^{(3)})) \right) \right],$$

and we have defined

$$\psi_{jt}^{(1)} = \boldsymbol{w}_{1j}^{\top} \boldsymbol{h}_{t-1} + \boldsymbol{w}_{3j}^{\top} \boldsymbol{v}_{t-1} + b_j \,, \quad (4.13)$$

$$\psi_{mt}^{(2)} = \boldsymbol{w}_{2m}^{\top} \boldsymbol{h}_{t} + \boldsymbol{w}_{4m}^{\top} \boldsymbol{v}_{t-1} + c_m \,, \quad (4.14)$$

$$\psi_{jt}^{(3)} = \boldsymbol{u}_{1j}^{\top} \boldsymbol{h}_{t-1} + \boldsymbol{u}_{2j}^{\top} \boldsymbol{v}_{t} + \boldsymbol{u}_{3j}^{\top} \boldsymbol{v}_{t-1} + d_j \,. \quad (4.15)$$

By further defining

$$\chi_{jt}^{(1)} = h_{jt} - \sigma(\psi_{jt}^{(1)}), \quad \chi_{mt}^{(2)} = v_{mt} - \sigma(\psi_{mt}^{(2)}), \quad \chi_{jt}^{(3)} = h_{jt} - \sigma(\psi_{jt}^{(3)}), \quad (4.16)$$

The gradients for the model parameters $\boldsymbol{\theta}$ are expressed as

$$\frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial w_{1jj'}} = \chi_{jt}^{(1)} h_{j't-1}, \quad \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial w_{3jm}} = \chi_{jt}^{(1)} v_{mt-1}, \quad \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial b_j} = \chi_{jt}^{(1)}, \tag{4.17}$$

$$\frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial w_{2mj}} = \chi_{mt}^{(2)} h_{jt}, \quad \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial w_{4mm'}} = \chi_{mt}^{(2)} v_{m't-1}, \quad \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial c_m} = \chi_{mt}^{(2)}. \tag{4.18}$$

The gradients for the recognition parameters $\boldsymbol{\phi}$ are expressed as

$$\frac{\partial \log q_{\boldsymbol{\phi}}(\boldsymbol{h}_t|\boldsymbol{v}_t)}{\partial u_{1jj'}} = \chi_{jt}^{(3)} h_{j't-1}, \qquad \frac{\partial \log q_{\boldsymbol{\phi}}(\boldsymbol{h}_t|\boldsymbol{v}_t)}{\partial u_{2jm}} = \chi_{jt}^{(3)} v_{mt}, \tag{4.19}$$

$$\frac{\partial \log q_{\boldsymbol{\phi}}(\boldsymbol{h}_t|\boldsymbol{v}_t)}{\partial u_{3jm}} = \chi_{jt}^{(3)} v_{mt-1}, \qquad \frac{\partial \log q_{\boldsymbol{\phi}}(\boldsymbol{h}_t|\boldsymbol{v}_t)}{\partial d_j} = \chi_{jt}^{(3)}. \tag{4.20}$$

Modeling Real-valued Data

When modeling real-valued data, we substitute (4.3) with $p(\boldsymbol{v}_t|\boldsymbol{h}_t, \boldsymbol{v}_{t-1}) = \mathcal{N}(\boldsymbol{\mu}_t, \mathrm{diag}(\boldsymbol{\sigma}_t^2))$, where

$$\mu_{mt} = \boldsymbol{w}_{2m}^\top \boldsymbol{h}_t + \boldsymbol{w}_{4m}^\top \boldsymbol{v}_{t-1} + c_m, \quad \log \sigma_{mt} = (\boldsymbol{w}'_{2m})^\top \boldsymbol{h}_t + (\boldsymbol{w}'_{4m})^\top \boldsymbol{v}_{t-1} + c'_m, \tag{4.21}$$

and we have $\mathbf{W}'_2 \in \mathbb{R}^{M \times J}$ and $\mathbf{W}'_4 \in \mathbb{R}^{M \times M}$. The recognition model remains the same as in (4.8). Let $\tau_{mt} = \log \sigma_{mt}$, we obtain

$$\begin{aligned} l_t = &\sum_{j=1}^{J} \left(\psi_{jt}^{(1)} h_{jt} - \log(1 + \exp(\psi_{jt}^{(1)})) \right) - \sum_{m=1}^{M} \left(\frac{1}{2} \log 2\pi + \tau_{mt} + \frac{(v_{mt} - \mu_{mt})^2}{2e^{2\tau_{mt}}} \right) \\ &- \left[\sum_{j=1}^{J} \left(\psi_{jt}^{(3)} h_{jt} - \log(1 + \exp(\psi_{jt}^{(3)})) \right) \right]. \end{aligned} \tag{4.22}$$

All the gradient calculation remains the same as (4.17)-(4.20), except the following.

$$\frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial w_{2mj}} = \chi_{mt}^{(4)} h_{jt}, \quad \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial w_{4mm'}} = \chi_{mt}^{(4)} v_{m't-1}, \quad \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial c_m} = \chi_{mt}^{(4)},$$

$$\frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial w'_{2mj}} = \chi_{mt}^{(5)} h_{jt}, \quad \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial w'_{4mm'}} = \chi_{mt}^{(5)} v_{m't-1}, \quad \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial c'_m} = \chi_{mt}^{(5)},$$

where we have defined

$$\chi_{mt}^{(4)} = \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial \mu_{mt}} = \frac{v_{mt} - \mu_{mt}}{e^{2\tau_{mt}}}, \quad \chi_{mt}^{(5)} = \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial \tau_{mt}} = \frac{(v_{mt} - \mu_{mt})^2}{e^{2\tau_{mt}}} - 1\,.$$

Modeling Count Data

We also introduce an approach for modeling time-series data with count observations, by replacing (4.3) with $p(\boldsymbol{v}_t|\boldsymbol{h}_t, \boldsymbol{v}_{t-1}) = \prod_{m=1}^{M} y_{mt}^{v_{mt}}$, where

$$y_{mt} = \frac{\exp(\boldsymbol{w}_{2m}^\top \boldsymbol{h}_t + \boldsymbol{w}_{4m}^\top \boldsymbol{v}_{t-1} + c_m)}{\sum_{m'=1}^{M} \exp(\boldsymbol{w}_{2m'}^\top \boldsymbol{h}_t + \boldsymbol{w}_{4m'}^\top \boldsymbol{v}_{t-1} + c_{m'})}\,. \tag{4.23}$$

The recognition model still remains the same as in (4.8). The l_t now is expressed as

$$\begin{aligned} l_t = &\sum_{j=1}^{J} \left(\psi_{jt}^{(1)} h_{jt} - \log(1 + \exp(\psi_{jt}^{(1)})) \right) + \sum_{m=1}^{M} \left(\psi_{mt}^{(2)} v_{mt} - v_{mt} \log \sum_{m'=1}^{M} \exp(\psi_{mt}^{(2)}) \right) \\ &- \left[\sum_{j=1}^{J} \left(\psi_{jt}^{(3)} h_{jt} - \log(1 + \exp(\psi_{jt}^{(3)})) \right) \right]. \end{aligned} \tag{4.24}$$

All the gradient calculations remain the same as (4.17)-(4.20), except the following

$$\frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial w_{2mj}} = \chi_{mt}^{(6)} h_{jt}, \quad \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial w_{4mm'}} = \chi_{mt}^{(6)} v_{m't-1}, \quad \frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v}_t, \boldsymbol{h}_t)}{\partial c_m} = \chi_{mt}^{(6)}\,.$$

where we have defined $\chi_{mt}^{(6)} = v_{mt} - y_{mt} \sum_{m'=1}^{M} v_{m't}$.

5

Semantic Compositional Networks for Visual Captioning

In this chapter, I will present semantic compositional network for controllable visual captionong. The proposed model uses a mixture-of-experts design, and can be considered as training an ensemble of up to 1000 LSTMs simultaneously. The degree to which each member of the ensemble is used to generate an image caption is tied to the image-dependent probability of the corresponding tag.

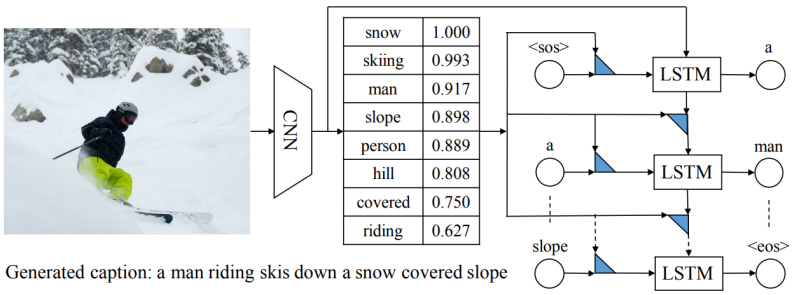
5.1 Introduction

There has been a recent surge of interest in developing models that can generate captions for images or videos, termed visual captioning. Most of these approaches learn a probabilistic model of the caption, conditioned on an image or a video (Mao et al., 2015; Venugopalan et al., 2015b; Fang et al., 2015; Karpathy and Fei-Fei, 2015; Vinyals et al., 2015; Xu et al., 2015; Donahue et al., 2015; Venugopalan et al., 2015a; Pan et al., 2016; Yu et al., 2016). Inspired by the successful use of the encoder-decoder framework employed in machine translation (Bahdanau et al., 2015; Cho et al., 2014;

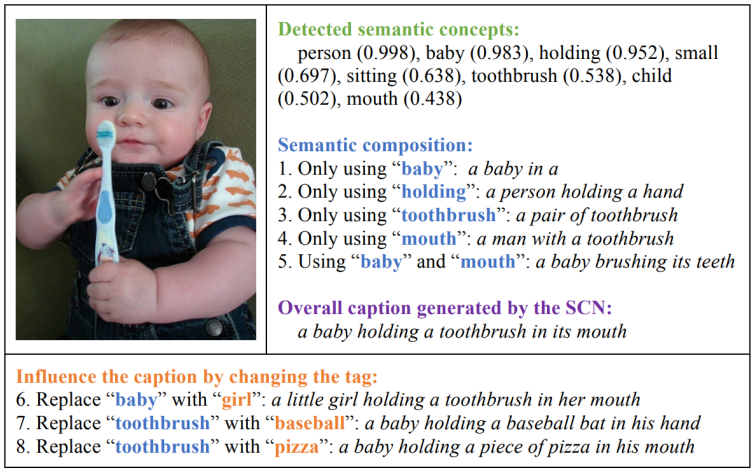
Sutskever et al., 2014), most recent work on visual captioning employs a convolutional neural network (CNN) as an encoder, obtaining a fixed-length vector representation of a given image or video. A recurrent neural network (RNN), typically implemented with long short-term memory (LSTM) units (Hochreiter and Schmidhuber, 1997), is then employed as a decoder to generate a caption. Aided by advances in CNN training on large datasets (*e.g.*, ImageNet (Russakovsky et al., 2015)) (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015; Szegedy et al., 2015; He et al., 2016), the quality of caption generation has improved significantly using this approach.

Recent work shows that adding explicit high-level semantic concepts (*i.e.*, tags) of the input image/video can further improve visual captioning. As shown in Wu et al. (2016a); You et al. (2016), detecting explicit semantic concepts encoded in an image, and adding this high-level semantic information into the CNN-LSTM framework, has improved performance significantly. Specifically, Wu et al. (2016a) feeds the semantic concepts as an *initialization* step into the LSTM decoder. In You et al. (2016), a model of semantic attention is proposed which selectively attends to semantic concepts through a soft attention mechanism (Bahdanau et al., 2015). On the other hand, although significant performance improvements were achieved, integration of semantic concepts into the LSTM-based caption generation process is constrained in these methods; *e.g.*, only through soft attention or initialization of the first step of the LSTM.

In this paper, we propose the Semantic Compositional Network (SCN) to more effectively assemble the meanings of individual tags to generate the caption that describes the overall meaning of the image, as illustrated in Figure 5.1(a). Each triangle symbol represents an ensemble of tag-dependent weight matrices. The number next to a semantic concept (*i.e.*, a tag) is the probability that the corresponding semantic concept is presented in the input image. Similar to the conventional CNN-LSTM-based image captioning framework, a CNN is used to extract the visual feature



(a) Overview of the proposed model.



(b) Examples of SCN-based image captioning.

FIGURE 5.1: Model architecture and illustration of semantic composition.

vector, which is then fed into a LSTM for generating the image caption (for simplicity, in this discussion we refer to images, but the method is also applicable to video). However, unlike the conventional LSTM, the SCN extends each weight matrix of the conventional LSTM to an *ensemble* of tag-dependent weight matrices, subject to the probabilities that the tags are present in the image. These tag-dependent weight matrices form a weight tensor with a large number of parameters. In order to make learning feasible, we factorize that tensor to be a three-way matrix product, which dramatically reduces the number of free parameters to be learned, while also yielding excellent performance.

The main contributions of this paper are as follows: (i) We propose the SCN

to effectively compose individual semantic concepts for image captioning. (*ii*) We perform comprehensive evaluations on two image captioning benchmarks, demonstrating that the proposed method outperforms previous state-of-the-art approaches by a substantial margin. For example, as reported by the COCO official test server, we achieve a BLEU-4 of 33.1, an improvement of 1.5 points over the current published state-of-the-art (You et al., 2016). (*iii*) We extend the proposed framework from image captioning to video captioning, demonstrating the versatility of the proposed model. (*iv*) We also perform a detailed analysis to study the SCN, showing that the model can adjust the caption smoothly by modifying the tags.

5.2 Related work

We focus on recent neural-network-based literature for caption generation, as these are most relevant to our work. Such models typically extract a visual feature vector via a CNN, and then send that vector to a language model for caption generation. Representative works include Chen and Lawrence Zitnick (2015); Devlin et al. (2015); Donahue et al. (2015); Karpathy and Fei-Fei (2015); Kiros et al. (2014a,c); Mao et al. (2015); Vinyals et al. (2015) for image captioning and Donahue et al. (2015); Venugopalan et al. (2015a,b); Yu et al. (2016); Ballas et al. (2016); Pu et al. (2018); Dong et al. (2016) for video captioning. The differences of the various methods mainly lie in the types of CNN architectures and language models. For example, the vanilla RNN (Elman, 1990) was used in Mao et al. (2015); Karpathy and Fei-Fei (2015), while the LSTM (Hochreiter and Schmidhuber, 1997) was used in Vinyals et al. (2015); Venugopalan et al. (2015a,b). The visual feature vector was only fed into the RNN once at the first time step in Vinyals et al. (2015); Karpathy and Fei-Fei (2015), while it was used at each time step of the RNN in Mao et al. (2015).

Most recently, Xu et al. (2015) utilized an attention-based mechanism to learn where to focus in the image during caption generation. This work was followed

by Yang et al. (2016) which introduced a review module to improve the attention mechanism and Liu et al. (2016) which proposed a method to improve the correctness of visual attention. Moreover, a variational autoencoder was developed in Pu et al. (2016b) for image captioning. Other related work includes Pan et al. (2016) for video captioning and Anne Hendricks et al. (2016) for composing sentences that describe novel objects.

Another class of models uses semantic information for caption generation. Specifically, Jia et al. (2015) applied retrieved sentences as additional semantic information to guide the LSTM when generating captions, while Fang et al. (2015); Wu et al. (2016a); You et al. (2016) applied a semantic-concept-detection process (Gan et al., 2016a) before generating sentences. In addition, Fang et al. (2015) also proposes a deep multimodal similarity model to project visual features and captions into a joint embedding space. This line of methods represents the current state of the art for image captioning. Our proposed model also lies in this category; however, distinct from the aforementioned approaches, our model uses weight *tensors* in LSTM units. This allows learning an ensemble of semantic-concept-dependent weight matrices for generating the caption.

Related to but distinct from the hierarchical composition in a recursive neural network (Socher et al., 2014), our model carries out implicit composition of concepts, and there is no hierarchical relationship among these concepts. Figure 5.1(b) illustrates the semantic composition manifested in the SCN model. Specifically, a set of semantic concepts, such as "*baby, holding, toothbrush, mouth*", are detected with high probabilities. If only one semantic concept is turned on, the model will generate a description covering only part of the input image, as shown in sentences 1-5 of Figure 5.1(b); however, by assembling all these semantic concepts, the SCN is able to generate a comprehensive description "*a baby holding a toothbrush in its mouth*". More interestingly, as shown in sentences 6-8 of Figure 5.1(b), the SCN

also has great flexibility to adjust the generation of the caption by changing certain semantic concepts.

The tensor factorization method is used to make the SCN compact and simplify learning. Similar ideas have been exploited in Kiros et al. (2014b); Memisevic and Hinton (2007); Song et al. (2016a); Sutskever et al. (2011); Taylor and Hinton (2009); Wu et al. (2016b); Gan et al. (2017a); Wang et al. (2017). In Donahue et al. (2015); Jin et al. (2015); Kiros et al. (2014a) the authors also briefly discussed using the tensor factorization method for image captioning. Specifically, visual features extracted from CNNs are utilized in Donahue et al. (2015); Kiros et al. (2014a), and an inferred scene vector is used in Jin et al. (2015) for tensor factorization. In contrast to these works, we use the semantic-concept vector that is formed by the probabilities of all tags to weight the basis LSTM weight matrices in the ensemble. Our semantic-concept vector is more powerful than the visual-feature vector (Donahue et al., 2015; Kiros et al., 2014a) and the scene vector (Jin et al., 2015) in terms of providing explicit semantic information of an image, hence leading to significantly better performance, as shown in our quantitative evaluation. In addition, the usage of semantic concepts also makes the proposed SCN more interpretable than Donahue et al. (2015); Jin et al. (2015); Kiros et al. (2014a), as shown in our qualitative analysis, since each unit in the semantic-concept vector corresponds to an explicit tag.

5.3 Semantic compositional networks

5.3.1 Review of RNN for image captioning

Consider an image $\mathbf{I}$, with associated caption $\mathbf{X}$. We first extract feature vector $\boldsymbol{v}(\mathbf{I})$, which is often the top-layer features of a pretrained CNN. Henceforth, for simplicity, we omit the explicit dependence on $\mathbf{I}$, and represent the visual feature vector as $\boldsymbol{v}$. The length-T caption is represented as $\mathbf{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$, with $\boldsymbol{x}_t$ a 1-of-V ("one hot") encoding vector, with V the size of the vocabulary. The length T typically

varies among different captions.

The t-th word in a caption, $\boldsymbol{x}_t$, is linearly embedded into an n_x-dimensional real-valued vector $\boldsymbol{w}_t = \mathbf{W}_e \boldsymbol{x}_t$, where $\mathbf{W}_e \in \mathbb{R}^{n_x \times V}$ is a word embedding matrix (learned), *i.e.*, $\boldsymbol{w}_t$ is a column of $\mathbf{W}_e$ chosen by the one-hot $\boldsymbol{x}_t$. The probability of caption $\mathbf{X}$ given image feature vector $\boldsymbol{v}$ is defined as

$$p(\mathbf{X}|\mathbf{I}) = \prod_{t=1}^{T} p(\boldsymbol{x}_t|\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{t-1}, \boldsymbol{v}) \,, \tag{5.1}$$

where $\boldsymbol{x}_0$ is defined as a special start-of-the-sentence token. All the words in the caption are sequentially generated using a RNN, until the end-of-the-sentence symbol is generated. Specifically, each conditional $p(\boldsymbol{x}_t|\boldsymbol{x}_{<t}, \boldsymbol{v})$ is specified as $\text{softmax}(\mathbf{V}\boldsymbol{h}_t)$, where $\boldsymbol{h}_t$ is recursively updated through $\boldsymbol{h}_t = \mathcal{H}(\boldsymbol{w}_{t-1}, \boldsymbol{h}_{t-1}, \boldsymbol{v})$, and $\boldsymbol{h}_0$ is defined as a zero vector ($\boldsymbol{h}_0$ is *not* updated during training). $\mathbf{V}$ is the weight matrix connecting the RNN's hidden state, used for computing a distribution over words. Bias terms are omitted for simplicity throughout the paper.

Without loss of generality, we begin by discussing an RNN with a simple transition function $\mathcal{H}(\cdot)$; this is generalized in Section 5.3.4 to the LSTM. Specifically, $\mathcal{H}(\cdot)$ is defined as

$$\boldsymbol{h}_t = \sigma(\mathbf{W}\boldsymbol{x}_{t-1} + \mathbf{U}\boldsymbol{h}_{t-1} + \mathbb{1}(t = 1) \cdot \mathbf{C}\boldsymbol{v}) \,, \tag{5.2}$$

where $\sigma(\cdot)$ is a logistic sigmoid function, and $\mathbb{1}(\cdot)$ represents an indicator function. Feature vector $\boldsymbol{v}$ is fed into the RNN at the beginning, *i.e.*, at $t = 1$. $\mathbf{W}$ is defined as the input matrix, and $\mathbf{U}$ is termed the recurrent matrix. The model in (5.2) is illustrated in Figure 5.2(a).

5.3.2 Semantic concept detection

The SCN developed below is based on the detection of semantic concepts, *i.e.*, tags, in the image under test. In order to detect such from an image, we first select a set

of tags from the caption text in the training set. Following Fang et al. (2015), we use the K most common words in the training captions to determine the vocabulary of tags, which includes the most frequent nouns, verbs, or adjectives.

In order to predict semantic concepts given a test image, motivated by Wu et al. (2016a); Tran et al. (2016), we treat this problem as a multi-label classification task. Suppose there are N training examples, and $\boldsymbol{y}_i = [y_{i1}, \ldots, y_{iK}] \in \{0, 1\}^K$ is the label vector of the i-th image, where $y_{ik} = 1$ if the image is annotated with tag k, and $y_{ik} = 0$ otherwise. Let $\boldsymbol{v}_i$ and $\boldsymbol{s}_i$ represent the image feature vector and the semantic feature vector for the i-th image, the cost function to be minimized is

$$\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \Big(y_{ik} \log s_{ik} + (1 - y_{ik}) \log(1 - s_{ik}) \Big) , \tag{5.3}$$

where $\boldsymbol{s}_i = \sigma\big(f(\boldsymbol{v}_i)\big)$ is a K-dimensional vector with $\boldsymbol{s}_i = [s_{i1}, \ldots, s_{iK}]$, $\sigma(\cdot)$ is the logistic sigmoid function and $f(\cdot)$ is implemented as a multilayer perceptron (MLP).

In testing, for each input image, we compute a semantic-concept vector $\boldsymbol{s}$, formed by the probabilities of all tags, computed by the semantic-concept detection model.

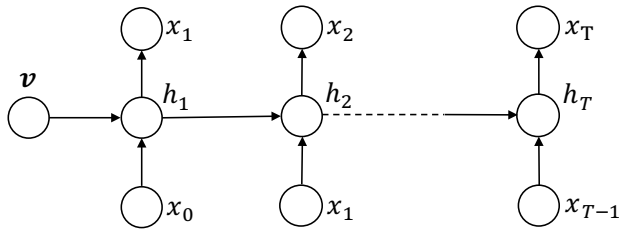
5.3.3 SCN-RNN

The SCN extends each weight matrix of the conventional RNN to be an ensemble of a set of tag-dependent weight matrices, subjective to the probabilities that the tags are present in the image. Specifically, the SCN-RNN computes the hidden states as follows

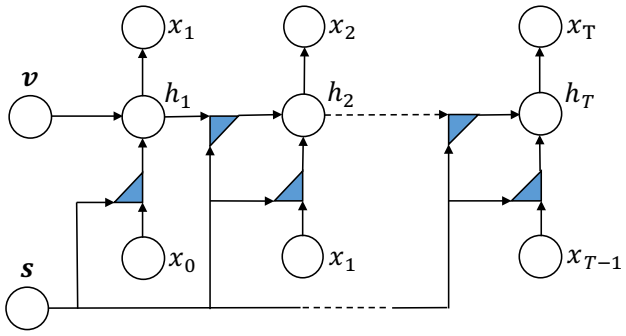
$$\boldsymbol{h}_t = \sigma(\mathbf{W}(\boldsymbol{s})\boldsymbol{x}_{t-1} + \mathbf{U}(\boldsymbol{s})\boldsymbol{h}_{t-1} + \boldsymbol{z}) , \tag{5.4}$$

where $\boldsymbol{z} = \mathbb{1}(t = 1) \cdot \mathbf{C}\boldsymbol{v}$, and $\mathbf{W}(\boldsymbol{s})$ and $\mathbf{U}(\boldsymbol{s})$ are ensembles of tag-dependent weight matrices, subjective to the probabilities that the tags are present in the image, according to the semantic-concept vector $\boldsymbol{s}$.

Given $\boldsymbol{s} \in \mathbb{R}^K$, we define two weight tensors $\mathbf{W}_{\mathcal{T}} \in \mathbb{R}^{n_h \times n_x \times K}$ and $\mathbf{U}_{\mathcal{T}} \in \mathbb{R}^{n_h \times n_h \times K}$, where n_h is the number of hidden units and n_x is the dimension of word embedding.



(a) Basic RNN



(b) SCN-RNN

FIGURE 5.2: Comparison of our proposed model with a conventional recurrent neural network (RNN) for caption generation.

$\mathbf{W}(\boldsymbol{s}) \in \mathbb{R}^{n_h \times n_x}$ and $\mathbf{U}(\boldsymbol{s}) \in \mathbb{R}^{n_h \times n_h}$ can be specified as

$$\mathbf{W}(\boldsymbol{s}) = \sum_{k=1}^{K} s_k \mathbf{W}_{\mathcal{T}}[k],\ \mathbf{U}(\boldsymbol{s}) = \sum_{k=1}^{K} s_k \mathbf{U}_{\mathcal{T}}[k]\,, \tag{5.5}$$

where s_k is the k-th element in $\boldsymbol{s}$; $\mathbf{W}_{\mathcal{T}}[k]$ and $\mathbf{U}_{\mathcal{T}}[k]$ denote the k-th 2D "slice" of $\mathbf{W}_{\mathcal{T}}$ and $\mathbf{U}_{\mathcal{T}}$, respectively. The probability of the k-th semantic concept, s_k, is associated with a pair of RNN weight matrices $\mathbf{W}_{\mathcal{T}}[k]$ and $\mathbf{U}_{\mathcal{T}}[k]$, implicitly specifying K RNNs in total. Consequently, training such a model as defined in (5.4) and (5.5) can be interpreted as *jointly* training an ensemble of K RNNs.

Though appealing, the number of parameters is proportional to K, which is prohibitive for large K (*e.g.*, $K = 1000$ for COCO). In order to remedy this problem, we adopt ideas from Memisevic and Hinton (2007) to factorize $\mathbf{W}(\boldsymbol{s})$ and $\mathbf{U}(\boldsymbol{s})$ defined in (5.5) as

$$\mathbf{W}(\boldsymbol{s}) = \mathbf{W}_a \cdot \text{diag}(\mathbf{W}_b \boldsymbol{s}) \cdot \mathbf{W}_c \,, \tag{5.6}$$

$$\mathbf{U}(\boldsymbol{s}) = \mathbf{U}_a \cdot \text{diag}(\mathbf{U}_b \boldsymbol{s}) \cdot \mathbf{U}_c \,, \tag{5.7}$$

where $\mathbf{W}_a \in \mathbb{R}^{n_h \times n_f}$, $\mathbf{W}_b \in \mathbb{R}^{n_f \times K}$ and $\mathbf{W}_c \in \mathbb{R}^{n_f \times n_x}$. Similiarly, $\mathbf{U}_a \in \mathbb{R}^{n_h \times n_f}$, $\mathbf{U}_b \in \mathbb{R}^{n_f \times K}$ and $\mathbf{U}_c \in \mathbb{R}^{n_f \times n_h}$. n_f is the number of factors. Substituting (5.6) and (5.7) into (5.4), we obtain our SCN with an RNN as

$$\tilde{\boldsymbol{x}}_{t-1} = \mathbf{W}_b \boldsymbol{s} \odot \mathbf{W}_c \boldsymbol{x}_{t-1} \,, \tag{5.8}$$

$$\tilde{\boldsymbol{h}}_{t-1} = \mathbf{U}_b \boldsymbol{s} \odot \mathbf{U}_c \boldsymbol{h}_{t-1} \,, \tag{5.9}$$

$$\boldsymbol{z} = \mathbb{1}(t = 1) \cdot \mathbf{C}\boldsymbol{v} \,, \tag{5.10}$$

$$\boldsymbol{h}_t = \sigma(\mathbf{W}_a \tilde{\boldsymbol{x}}_{t-1} + \mathbf{U}_a \tilde{\boldsymbol{h}}_{t-1} + \boldsymbol{z}) \,. \tag{5.11}$$

where $\odot$ denotes the element-wise multiply (Hadamard) operator.

$\mathbf{W}_a$ and $\mathbf{W}_c$ are shared among all the captions, effectively capturing common linguistic patterns; while the diagonal term, $\text{diag}(\mathbf{W}_b \boldsymbol{s})$, accounts for semantic aspects of the image under test, captured by $\boldsymbol{s}$. The same analysis also holds true for $\mathbf{U}_{a,b,c}$. In this factorized model, the RNN weight matrices that correspond to each semantic concept share "structure." This factorized model (termed SCN-RNN) is illustrated in Figure 5.2(b).

To provide further motivation for and insight into the decompositions in (5.6) and (5.7), let $\boldsymbol{w}_{bk}$ represent the kth column of $\mathbf{W}_b$, then

$$\mathbf{W}(\boldsymbol{s}) = \sum_{k=1}^{K} s_k [\mathbf{W}_a \cdot \text{diag}(\boldsymbol{w}_{bk}) \cdot \mathbf{W}_c] \,. \tag{5.12}$$

A similar decomposition is manifested for $\mathbf{U}(\boldsymbol{s})$. The matrix $\mathbf{W}_a \cdot \text{diag}(\boldsymbol{w}_{bk}) \cdot \mathbf{W}_c$ may be interpreted as the k-th "slice" of a weight tensor, with each slice corresponding to one of the K semantic concepts (K total tensor "slices," each of size $n_h \times n_x$). Hence, via the decomposition in (5.6) and (5.7), we effectively learn an ensemble of K sets of

RNN parameters, one for each semantic concept. This is efficiently done by sharing $\mathbf{W}_a$ and $\mathbf{W}_c$ when composing each member of the ensemble. The weight with which the k-th slice of this tensor contributes to the RNN parameters for a given image is dependent on the respective probability s_k with which the k-th semantic concept is inferred to be associated with image $\mathbf{I}$.

The number of parameters in the basic RNN model (see Figure 5.2(a)) is $n_h \cdot (n_x + n_h)$, while the number of parameters in the SCN-RNN model (see Figure 5.2(b)) is $n_f \cdot (n_x + 2K + 3n_h)$. In experiments, we set $n_f = n_h$. Therefore, the additional number of parameters is $2 \cdot n_h \cdot (n_h + K)$. This increased model complexity also indicates increased training/testing time.

5.3.4 SCN-LSTM

RNNs with LSTM units (Hochreiter and Schmidhuber, 1997) have emerged as a popular architecture, due to their representational power and effectiveness at capturing long-term dependencies. We generalize the SCN-RNN model by using LSTM units. Specifically, we define $\boldsymbol{h}_t = \mathcal{H}(\boldsymbol{x}_{t-1}, \boldsymbol{h}_{t-1}, \boldsymbol{v}, \boldsymbol{s})$ as

$$\boldsymbol{i}_t = \sigma(\mathbf{W}_{ia}\tilde{\boldsymbol{x}}_{i,t-1} + \mathbf{U}_{ia}\tilde{\boldsymbol{h}}_{i,t-1} + \boldsymbol{z})\,, \tag{5.13}$$

$$\boldsymbol{f}_t = \sigma(\mathbf{W}_{fa}\tilde{\boldsymbol{x}}_{f,t-1} + \mathbf{U}_{fa}\tilde{\boldsymbol{h}}_{f,t-1} + \boldsymbol{z})\,, \tag{5.14}$$

$$\boldsymbol{o}_t = \sigma(\mathbf{W}_{oa}\tilde{\boldsymbol{x}}_{o,t-1} + \mathbf{U}_{oa}\tilde{\boldsymbol{h}}_{o,t-1} + \boldsymbol{z})\,, \tag{5.15}$$

$$\tilde{\boldsymbol{c}}_t = \sigma(\mathbf{W}_{ca}\tilde{\boldsymbol{x}}_{c,t-1} + \mathbf{U}_{ca}\tilde{\boldsymbol{h}}_{c,t-1} + \boldsymbol{z})\,, \tag{5.16}$$

$$\boldsymbol{c}_t = \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1}\,, \tag{5.17}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)\,, \tag{5.18}$$

where $\boldsymbol{z} = \mathbb{1}(t = 1) \cdot \mathbf{C}\boldsymbol{v}$. For $\star = i, f, o, c$, we define

$$\tilde{\boldsymbol{x}}_{\star,t-1} = \mathbf{W}_{\star b}\boldsymbol{s} \odot \mathbf{W}_{\star c}\boldsymbol{x}_{t-1}\,, \tag{5.19}$$

$$\tilde{\boldsymbol{h}}_{\star,t-1} = \mathbf{U}_{\star b}\boldsymbol{s} \odot \mathbf{U}_{\star c}\boldsymbol{h}_{t-1}\,. \tag{5.20}$$

Since we implement the SCN with LSTM units, we name this model SCN-LSTM. In experiments, since LSTM is more powerful than classifical RNN, we only report results using SCN-LSTM.

In summary, distinct from previous image-captioning methods, our model has a unique way to utilize and combine the visual feature $\boldsymbol{v}$ and semantic-concept vector $\boldsymbol{s}$ extracted from an image $\mathbf{I}$. $\boldsymbol{v}$ is fed into the LSTM to initialize the first step, which is expected to provide the LSTM an overview of the image content. While the LSTM state is initialized with the overall visual context $\boldsymbol{v}$, an ensemble of K sets of LSTM parameters is utilized when decoding, weighted by the semantic-concept vector $\boldsymbol{s}$, to generate the caption.

Model learning Given the image $\mathbf{I}$ and associated caption $\mathbf{X}$, the objective function is the sum of the log-likelihood of the caption conditioned on the image representation:

$$\log p(\mathbf{X}|\mathbf{I}) = \sum_{t=1}^{T} p(\boldsymbol{x}_t|\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{t-1}, \boldsymbol{v}, \boldsymbol{s}) \,. \tag{5.21}$$

The above objective corresponds to a single image-caption pair. In training, we average over all training pairs.

5.3.5 Extension to video captioning

The above framework can be readily extended to the task of video captioning (Donahue et al., 2015; Venugopalan et al., 2015a,b; Yu et al., 2016; Ballas et al., 2016; Xu et al., 2016). In order to effectively represent the spatiotemporal visual content of a video, we use a two-dimensional (2D) *and* a three-dimensional (3D) CNN to extract visual features of video frames/clips. We then perform a mean pooling process Venugopalan et al. (2015b) over all 2D CNN features and 3D CNN features, to generate two feature vectors (one from 2D CNN features and the other from 3D CNN features). The representation of each video, $\boldsymbol{v}$, is produced by concatenating these

two features. Similarly, we also obtain the semantic-concept vector $\boldsymbol{s}$ by running the semantic-concept detector based on the video representation $\boldsymbol{v}$. After $\boldsymbol{v}$ and $\boldsymbol{s}$ are obtained, we employ the same model proposed above directly for video-caption generation, as described in Figure 5.2(b).

5.4 Experiments

5.4.1 Datasets

We present results on three benchmark datasets: COCO (Lin et al., 2014), Flickr30k (Young et al., 2014) and Youtube2Text (Chen and Dolan, 2011). COCO and Flickr30k are for image captioning, containing 123287 and 31783 images, respectively. Each image is annotated with at least 5 captions. We use the same pre-defined splits as Karpathy and Fei-Fei (2015) for all the datasets: on Flickr30k, 1000 images for validation, 1000 for test, and the rest for training; and for COCO, 5000 images are used for both validation and testing. We further tested our model on the official COCO test set consisting of 40775 images (human-generated captions for this split are not publicly available), and evaluated our model on the COCO evaluation server. We also follow the publicly available code (Karpathy and Fei-Fei, 2015) to preprocess the captions, yielding vocabulary sizes of 8791 and 7414 for COCO and Flickr30k, respectively.

Youtube2Text is used for video captioning, which contains 1970 Youtube clips, and each video is annotated with around 40 sentences. We use the same splits as provided in Venugopalan et al. (2015b), with 1200 videos for training, 100 videos for validation, and 670 videos for testing. We convert all captions to lower case and remove the punctuation, yielding vocabulary size of 12594 for Youtube2Text.

5.4.2 Training procedure

For image representation, we take the output of the 2048-way *pool*5 layer from ResNet-152 (He et al., 2016), pretrained on the ImageNet dataset (Russakovsky

et al., 2015). For video representation, in addition to using the 2D ResNet-152 to extract features on each video frame, we also utilize a 3D CNN (C3D) (Tran et al., 2015) to extract features on each video. The C3D is pretrained on Sports-1M video dataset (Karpathy et al., 2014), and we take the output of the 4096-way $fc7$ layer from C3D as the video representation. We consider the RGB frames of videos as input, with 2 frames per second. Each video frame is resized as 112×112 and 224×224 for the C3D and ResNet-152 feature extractor, respectively. The C3D feature extractor is applied on video clips of length 16 frames (as in Karpathy et al. (2014)) with an overlap of 8 frames.

We use the procedure described in Section 5.3.2 for semantic concept detection. The semantic-concept vocabulary size is determined to reflect the complexity of the dataset, which is set to 1000, 200 and 300 for COCO, Flickr30k and Youtube2Text, respectively. Since Youtube2Text is a relatively small dataset, we found that it is very difficult to train a reliable semantic-concept detector using the Youtube2Text dataset alone, due to its limited amount of data. In experiments, we utilize additional training data from COCO.

For model training, all the parameters in the SCN-LSTM are initialized from a uniform distribution in [-0.01,0.01]. All bias terms are initialized to zero. Word embedding vectors are initialized with the publicly available *word2vec* vectors (Mikolov et al., 2013). The embedding vectors of words not present in the pretrained set are initialzied randomly. The number of hidden units and the number of factors in SCN-LSTM are both set to 512 and we use mini-batches of size 64. The maximum number of epochs we run for all the three datasets is 20. Gradients are clipped if the norm of the parameter vector exceeds 5 (Sutskever et al., 2014). We do not perform any dataset-specific tuning and regularization other than dropout (Zaremba et al., 2014) and early stopping on validation sets. The Adam algorithm (Kingma and Ba, 2015) with learning rate 2×10^{-4} is utilized for optimization. All experiments are

Methods	COCO					
	B-1	B-2	B-3	B-4	M	C
NIC (Vinyals et al., 2015)	0.666	0.451	0.304	0.203	–	–
m-RNN (Mao et al., 2015)	0.67	0.49	0.35	0.25	–	–
Hard-Attention (Xu et al., 2015)	0.718	0.504	0.357	0.250	0.230	–
ATT (You et al., 2016)	0.709	0.537	0.402	0.304	0.243	–
Att-CNN+LSTM (Wu et al., 2016a)	0.74	0.56	0.42	0.31	0.26	0.94
LSTM-R	0.698	0.525	0.390	0.292	0.238	0.889
LSTM-T	0.716	0.546	0.411	0.312	0.250	0.952
LSTM-RT	0.724	0.555	0.419	0.316	0.252	0.970
LSTM-RT_2	0.730	0.568	0.430	0.322	0.249	0.977
SCN-LSTM	0.728	0.566	0.433	0.330	0.257	1.012
SCN-LSTM Ensemble of 5	**0.741**	**0.578**	**0.444**	**0.341**	**0.261**	**1.041**

Table 5.1: Performance of the proposed model (SCN-LSTM) and other state-of-the-art methods on the COCO dataset.

implemented in Theano (Theano Development Team, 2016)[1].

In testing, we use beam search for caption generation, which selects the top-k best sentences at each time step and considers them as the candidates to generate new top-k best sentences at the next time step. We set the beam size to $k = 5$ in experiments.

5.4.3 Evaluation

The widely used BLEU (Papineni et al., 2002), METEOR (Banerjee and Lavie, 2005), ROUGE-L (Lin, 2004), and CIDEr-D (Vedantam et al., 2015) metrics are reported in our quantitative evaluation of the performance of the proposed model and baselines in the literature. All the metrics are computed by using the code released by the COCO evaluation server (Chen et al., 2015b). For COCO and Flickr30k datasets, besides comparing to results reported in previous work, we also re-implemented strong baselines for comparison. The results of image captioning are presented in Table 5.1. The models we implemented are as follows.

[1] Code is publicly available at `https://github.com/zhegan27/Semantic_Compositional_Nets`.

Methods	Flickr30k				
	B-1	B-2	B-3	B-4	M
NIC (Vinyals et al., 2015)	0.663	0.423	0.277	0.183	–
m-RNN (Mao et al., 2015)	0.60	0.41	0.28	0.19	–
Hard-Attention (Xu et al., 2015)	0.669	0.439	0.296	0.199	0.185
ATT (You et al., 2016)	0.647	0.460	0.324	0.230	0.189
Att-CNN+LSTM (Wu et al., 2016a)	0.73	0.55	0.40	0.28	–
LSTM-R	0.657	0.437	0.296	0.201	0.186
LSTM-T	0.691	0.483	0.336	0.232	0.202
LSTM-RT	0.706	0.486	0.339	0.235	0.204
LSTM-RT_2	0.724	0.523	0.370	0.257	0.210
SCN-LSTM	0.735	0.530	0.377	0.265	0.218
SCN-LSTM Ensemble of 5	**0.747**	**0.552**	**0.403**	**0.288**	**0.223**

Table 5.2: Performance of the proposed model (SCN-LSTM) and other state-of-the-art methods on the Flickr30k dataset.

1. *LSTM-R / LSTM-T / LSTM-RT*: *R, T, RT* denotes using different features. Specifically, R denotes ResNet visual feature vector, T denotes Tags (*i.e.*, the semantic-concept vector), and RT denotes the concatenation of R and T. The features are fed into a standard LSTM decoder only at the initial time step. In particular, *LSTM-T* is the model proposed in Wu et al. (2016a).

2. *LSTM-RT_2*: The ResNet feature vector is sent to a standard LSTM decoder at the first time step, while the tag vector is sent to the LSTM decoder at every time step in addition to the input word. This model is similar to You et al. (2016) without using semantic attention. This is the model closest to ours, which provides a direct comparison to our proposed model.

3. *SCN-LSTM*: This is the model presented in Section 5.3.4.

For video captioning experiments, we use the same notation. For example, *LSTM-C* means we leverage the C3D feature for caption generation.

Model	BLEU-1		BLEU-2		BLEU-3		BLEU-4	
	c5	c40	c5	c40	c5	c40	c5	c40
SCN-LSTM	**0.740**	**0.917**	**0.575**	**0.839**	**0.436**	**0.739**	**0.331**	**0.631**
ATT	0.731	0.900	0.565	0.815	0.424	0.709	0.316	0.599
OV	0.713	0.895	0.542	0.802	0.407	0.694	0.309	0.587
MSR Cap	0.715	0.907	0.543	0.819	0.407	0.710	0.308	0.601
Model	METEOR		ROUGE-L		CIDEr-D		–	
	c5	c40	c5	c40	c5	c40	–	–
SCN-LSTM	**0.257**	**0.348**	**0.543**	**0.696**	**1.003**	**1.013**	–	–
ATT	0.250	0.335	0.535	0.682	0.943	0.958	–	–
OV	0.254	0.346	0.530	0.682	0.943	0.946	–	–
MSR Cap	0.248	0.339	0.526	0.680	0.931	0.937	–	–

Table 5.3: Comparison to published state-of-the-art image captioning models on the blind test set as reported by the COCO test server.

5.4.4 Quantitative results

Performance on COCO and Flickr30k We first present results on the task of image captioning, summarized in Table 5.1 and Table 5.2. The use of tags (*LSTM-T*) provides better performance than leveraging visual features alone (*LSTM-R*). Combining both tags and visual features further enhances performance, as expected. Compared with only feeding the tags into the LSTM at the initial time step (*LSTM-RT*), *LSTM-RT_2* yields better results, since it takes as input the tag feature at each time step. Further, the direct comparison between *LSTM-RT_2* and *SCN-LSTM* demonstrates the advantage of our proposed model, indicating that our approach is a better method to fuse semantic concepts into the LSTM.

We also report results averaging an ensemble of 5 identical *SCN-LSTM* models trained with different initializations, which is a common strategy adopted widely You et al. (2016) (note that now we employ ensembles in two ways: an ensemble of LSTM parameters linked to tags, and an overaching ensemble atop the entire model). We obtain state-of-the-art results on both COCO and Flickr30k datasets. Remarkably, we improve the state-of-the-art BLEU-4 score by 3.1 points on COCO.

Model	B-4	M	C
S2VT (Venugopalan et al., 2015a)	–	0.292	–
LSTM-E (Pan et al., 2016)	0.453	0.310	–
GRU-RCN (Ballas et al., 2016)	0.479	0.311	0.678
h-RNN (Yu et al., 2016)	0.499	0.326	0.658
LSTM-R	0.448	0.310	0.640
LSTM-C	0.445	0.309	0.644
LSTM-CR	0.469	0.317	0.688
LSTM-T	0.473	0.324	0.699
LSTM-CRT	0.475	0.316	0.647
LSTM-CRT_2	0.469	0.326	0.706
SCN-LSTM	0.502	0.334	0.770
SCN-LSTM Ensemble of 5	**0.511**	**0.335**	**0.777**

Table 5.4: Results on BLEU-4 (B-4), METEOR (M) and CIDEr-D (C) metrices compared to other state-of-the-art results and baselines on Youtube2Text.

Performance on COCO test server We also evaluate the proposed *SCN-LSTM* model by uploading results to the online COCO test server. Table 5.3 shows the comparison to the published state-of-the-art image captioning models on the blind test set as reported by the COCO test server. We include the models that have been published and perform at top-3 in the table. Compared to these methods, our proposed *SCN-LSTM* model achieves the best performance across all the evaluation metrics on both c5 and c40 testing sets.[2]

Performance on Youtube2Text Results on video captioning are presented in Table 5.4. The SCN-LSTM achieves significantly better results over all competing methods in all metrics, especially in CIDEr-D. For self-comparison, it is also worth noting that our model improves over *LSTM-*CRT_2 by a substantial margin. Again, using an overaching ensemble further enhances performance.

[2] Please check `https://competitions.codalab.org/competitions/3221#results` for the most recent results.

5.4.5 *Qualitative analysis*

Figure 5.3 shows three examples to illustrate the semantic composition on caption generation. Our model properly describes the image content by using the correctly detected tags. By manually replacing specific tags, our model can adjust the caption smoothly. For example, in the left image, by replacing the tag *"grass"* with *"bed"*, our model imagines *"a dog laying on top of a bed"*. Our model is also able to generate novel captions that are highly unlikely to occur in real life. For instance, in the middle image, by replacing the tag *"road"* and *"street"* with *"ocean"*, our model imagines *"a bus driving in the ocean"*; in the right image, by replacing the tag *"field"* with *"snow"*, our model dreams *"a group of zebras standing in the snow"*.

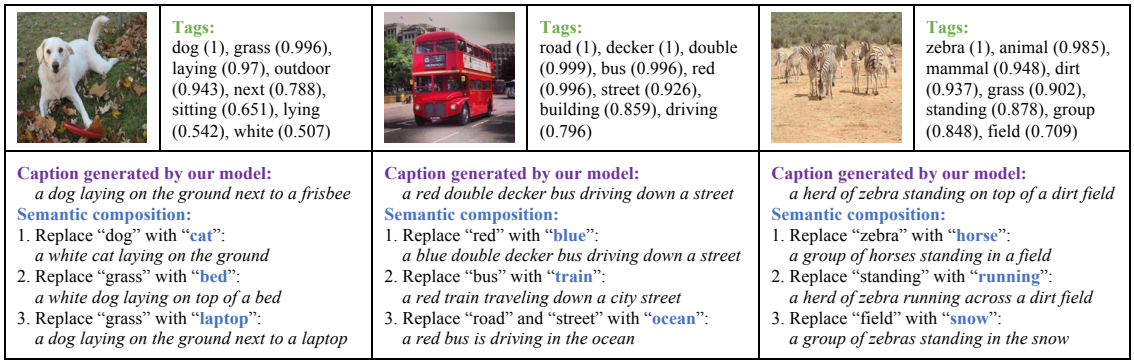


FIGURE 5.3: Illustration of semantic composition. Our model can adjust the caption smoothly as the semantic concepts are modified.

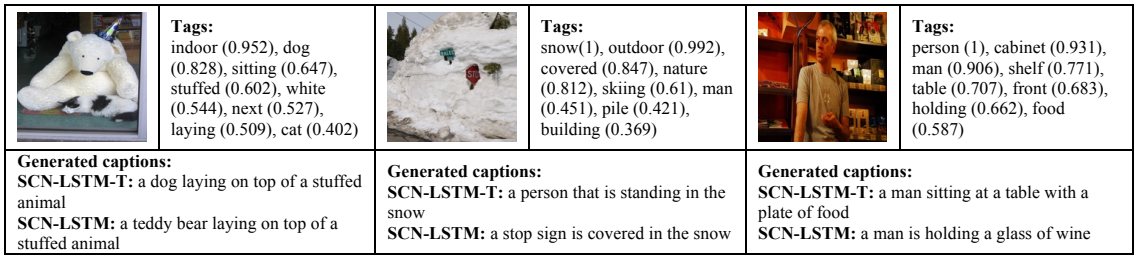


FIGURE 5.4: Detected tags and sentence generation on COCO, by SCN-LSTM-T and SCN-LSTM.

SCN not only picks up the tags well (and imagines the corresponding scenes),

but also selects the right functional words for different concepts to form syntactically correct caption. As illustrated in sentence 6 of Figure 5.1(b), by replacing the tag *"baby"* with *"girl"*, the generated captions not only changes *"a baby"* to *"a little girl"*, but more importantly, changes *"in its mouth"* to *"in her mouth"*. In addition, the SCN also infers the underlying semantic relatedness between different tags. As illustrated in sentence 4 of Figure 5.1(b), when only switching on the tag *"mouth"*, the generated caption becomes *"a man with a toothbrush"*, indicating the semantic closeness between *"mouth"*, *"man"* and *"toothbrush"*. By further switching on *"baby"*, we generate a more detailed description *"a baby brushing its teeth"*.

<table>
<tr>
<td>  </td>
<td>Tags:
book (1), shelf (1), table (0.965), sitting (0.955), person (0.955), library (0.908), room (0.829), front (0.464)</td>
<td>  </td>
<td>Tags:
person (1), table (0.822), wine (0.672), people (0.657), man (0.62), woman (0.601), sitting (0.502), holding (0.494)</td>
<td>  </td>
<td>Tags:
grass (1), red (0.982), fire (0.953), hydrant (0.852), dog (0.723), standing (0.598), next (0.476), field (0.341)</td>
</tr>
<tr>
<td colspan="2">Generated captions:
LSTM-R: a young girl is playing a video game
LSTM-RT$_2$: a group of people sitting at a table
SCN-LSTM: two women sitting at a table in a library</td>
<td colspan="2">Generated captions:
LSTM-R: a group of people standing around a table eating food
LSTM-RT$_2$: a group of people sitting at a table
SCN-LSTM: a man pouring wine into a wine glass</td>
<td colspan="2">Generated captions:
LSTM-R: a dog that is sitting on the ground
LSTM-RT$_2$: a dog standing next to a fire hydrant
SCN-LSTM: a dog standing next to a red fire hydrant</td>
</tr>
</table>

FIGURE 5.5: Detected tags and sentence generation on COCO, by LSTM-R, LSTM-RT$_2$, and SCN-LSTM.

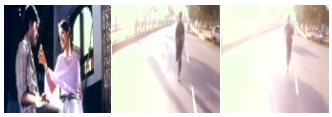
<table>
<tr>
<td>  </td>
<td>  </td>
<td>  </td>
</tr>
<tr>
<td>Tags:
man (0.806), game (0.629), playing (0.577), ball (0.555), football (0.522), men (0.435), running (0.386), soccer (0.252)</td>
<td>Tags:
man (0.976), person (0.881), guy (0.603), boy (0.456), gun (0.41), shooting (0.269), movie (0.232), standing (0.209)</td>
<td>Tags:
man (0.808), person (0.603), street (0.522), road (0.512), doing (0.424), riding (0.405), running (0.397), walking (0.296)</td>
</tr>
<tr>
<td>Generated captions:
LSTM-CR: a man is running
LSTM-CRT$_2$: a man is hitting a goal
SCN-LSTM: the men are playing soccer</td>
<td>Generated captions:
LSTM-CR: a man is playing a guitar
LSTM-CRT$_2$: a man is playing with a machine
SCN-LSTM: a man is shooting a gun</td>
<td>Generated captions:
LSTM-CR: a man is walking
LSTM-CRT$_2$: a man is dancing
SCN-LSTM: a man is running</td>
</tr>
</table>

FIGURE 5.6: Detected tags and sentence generation on Youtube2Text, by LSTM-CR, LSTM-CRT$_2$, and SCN-LSTM.

The above analysis shows the importance of tags in generating captions. However, SCN generates captions using both semantic concepts and the global visual feature

vector. The language model learns to assemble semantic concepts (weighted by their likelihood), in consideration of the global visual information, into a coherent meaningful sentence that captures the overall meaning of the image. In order to demonstrate the importance of visual feature vectors, we train another SCN-LSTM-T model, which is a SCN-LSTM model without the visual feature inputs, *i.e.*, with only tag inputs . As shown in the first example of Figure 5.4, the image tagger detects "*dog*" with high probability. Using only tag inputs, SCN-LSTM-T can only generate the wrong caption "*a dog laying on top of a stuffed animal*". With additional visual feature inputs, our SCN-LSTM model correctly replaces "*dog*" with "*teddy bear*" .

We further present examples of generated captions on COCO with various other methods in Figure 5.5, along with the detected tags. As can be seen, our model often generates more reasonable captions than *LSTM-R*, due to the use of high-level semantic concepts. For example, in the first image, *LSTM-R* outputs an irrelevant caption to the image, while the detection of *"table"* and *"library"* helps our model to generate more sensible caption. Further, although both our model and *LSTM-RT*$_2$ utilize detected tags for caption generation, our model often depicts the image content more comprehensively; *LSTM-RT*$_2$ has a larger potential to miss important details in the image. For instance, in the 3rd image, the tag "*red*" appears in the caption generated by our model, which is missed by *LSTM-RT*$_2$. This observation might be due to the fact that the SCN provides a better approach to fuse tag information into the process of caption generation. Similiar observations can also be found in the video captioning experiments, as demonstrated in Figure 5.6.

5.5 Conclusion

We have presented Semantic Compositional Network (SCN), a new framework to effectively compose the individual semantic meaning of tags for visual captioning. The SCN extends each weight matrix of the conventional LSTM to be a three-

way matrix product, with one of these matrices dependent on the inferred tags. Consequently, the SCN can be viewed an ensemble of tag-dependent LSTM bases, with the contribution of each LSTM basis unit proportional to the likelihood that the tag is present in the image. Experiments conducted on three visual captioning datasets validate the superiority of the proposed approach.

5.6 Supplementary Material

5.6.1 More results for Figure 5.4

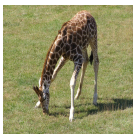
<table>
<tr>
<td>  </td>
<td>Tags:
outdoor (1), elephant (0.995), animal (0.988), grass (0.962), standing (0.89), rock (0.781), zoo (0.682), enclosure (0.619)</td>
<td>  </td>
<td>Tags:
indoor (0.966), table (0.919), food (0.849), kitchen (0.714), sitting (0.545), counter (0.436), top (0.285), doughnut (0.251)</td>
<td>  </td>
<td>Tags:
outdoor (0.998), building (0.996), man (0.613), front (0.434), standing (0.333), woman (0.255), walking (0.249), next (0.247)</td>
</tr>
<tr>
<td colspan="2">Generated captions:
SCN-LSTM-T: a couple of elephants standing next to each other
SCN-LSTM: a large elephant standing next to a tree</td>
<td colspan="2">Generated captions:
SCN-LSTM-T: a kitchen with a lot of food on it
SCN-LSTM: a bunch of doughnuts sitting on top of a counter</td>
<td colspan="2">Generated captions:
SCN-LSTM-T: a man standing in front of a building
SCN-LSTM: a statue of a man standing next to a building</td>
</tr>
<tr>
<td>  </td>
<td>Tags:
table (0.997), indoor (0.893), chair (0.876), room (0.692), sitting (0.583), window (0.58), wooden (0.542), small (0.344)</td>
<td>  </td>
<td>Tags:
fence (1), giraffe (0.994), animal (0.921), wooden (0.677), fenced (0.66), standing (0.592), next (0.493), zoo (0.442)</td>
<td>  </td>
<td>Tags:
grass (1), outdoor (0.992), giraffe (0.985), mammal (0.98), animal (0.978), field (0.93), eating (0.558), standing (0.508)</td>
</tr>
<tr>
<td colspan="2">Generated captions:
SCN-LSTM-T: a dining room with a table and chairs
SCN-LSTM: a wooden table with a laptop on it</td>
<td colspan="2">Generated captions:
SCN-LSTM-T: a giraffe standing next to a wooden fence
SCN-LSTM: a couple of giraffe standing next to each other</td>
<td colspan="2">Generated captions:
SCN-LSTM-T: a giraffe standing on top of a lush green field
SCN-LSTM: a giraffe is eating grass in a field</td>
</tr>
</table>

FIGURE 5.7: More detected tags and sentence generation on COCO, by SCN-LSTM-T and SCN-LSTM.

5.6.2 More results on image captioning

Tags:
polar (0.999), rock (0.998), animal (0.997), bear (0.993), mammal (0.988), zoo (0.881), white (0.779), large (0.748)

Generated captions:
LSTM-R: a polar bear standing on top of a rock
LSTM-RT$_2$: a polar bear standing on a rock
SCN-LSTM: a large white polar bear standing on a rock

Tags:
refrigerator (0.992), food (0.976), open (0.97), cabinet (0.953), shelf (0.582), filled (0.45), door (0.426), lots (0.329)

Generated captions:
LSTM-R: a display case filled with lots of food
LSTM-RT$_2$: a shelf full of different kinds of food
SCN-LSTM: a refrigerator filled with lots of food and drinks

Tags:
person (0.925), building (0.839), people (0.787), umbrella (0.779), group (0.704), child (0.519), standing (0.369), holding (0.272)

Generated captions:
LSTM-R: a group of people standing next to each other
LSTM-RT$_2$: a group of people standing in front of an umbrella
SCN-LSTM: a group of people standing in the rain with umbrellas

Tags:
bicycle (1), parked (0.923), next (0.889), group (0.829), sidewalk (0.783), many (0.698), lot (0.611), rack (0.596)

Generated captions:
LSTM-R: a group of motorcycles parked next to each other
LSTM-RT$_2$: a row of bikes parked in a row
SCN-LSTM: a bunch of bikes parked in a park lot

Tags:
food (0.939), oranges (0.839), fruit (0.836), slice (0.792), sliced (0.783), orange (0.764), plate (0.759), table (0.704)

Generated captions:
LSTM-R: a bowl of fruit sitting on top of a table
LSTM-RT$_2$: a bunch of oranges sitting on a table
SCN-LSTM: a bunch of oranges sitting on a plate

Tags:
clock (1), building (0.999), large (0.902), station (0.876), mounted (0.644), sitting (0.621), tower (0.574), building (0.418)

Generated captions:
LSTM-R: a clock on the wall of a building
LSTM-RT$_2$: a clock on the side of a building
SCN-LSTM: a large clock mounted to the side of a building

Tags:
dog (1), water (0.998), beach (0.805), standing (0.666), walking (0.451), next (0.435), ocean (0.301), white (0.225)

Generated captions:
LSTM-R: a dog that is playing with a frisbee
LSTM-RT$_2$: a couple of dogs standing on a beach
SCN-LSTM: a white dog walking on a beach

Tags:
water (0.985), beach (0.975), ocean (0.655), next (0.493), shore (0.324), sand (0.288), sandy (0.209), bench(0.204)

Generated captions:
LSTM-R: a bench that is sitting on the beach
LSTM-RT$_2$: a person sitting on a bench on a beach
SCN-LSTM: a wooden bench sitting on top of a sandy beach

Tags:
bench (0.997), fence (0.98), park (0.974), grass (0.877), sitting (0.771), wooden (0.582), next (0.511), green (0.377)

Generated captions:
LSTM-R: a park bench sitting in the middle of a forest
LSTM-RT$_2$: a park bench sitting on a park bench
SCN-LSTM: a wooden bench sitting in the middle of a park

Tags:
road (0.958), street (0.911), green (0.856), sign (0.601), traffic (0.549), car (0.401), truck (0.382), city (0.374)

Generated captions:
LSTM-R: a bus that is driving down the road
LSTM-RT$_2$: a bus parked on the side of a road
SCN-LSTM: a green bus driving down a city street

Tags:
person (0.958), woman (0.728), sitting (0.708), bench (0.394), people (0.381), next (0.371), group (0.361), front (0.311)

Generated captions:
LSTM-R: a couple of women standing next to each other
LSTM-RT$_2$: a couple of people sitting on a toilet
SCN-LSTM: a group of people sitting on a bench

Tags:
person (0.932), man (0.787), young (0.458), black (0.439), white (0.43), jumping (0.342), riding (0.242), trick (0.156)

Generated captions:
LSTM-R: a man sitting on top of a wooden bench
LSTM-RT$_2$: a man riding a skateboard down a street
SCN-LSTM: a black and white photo of a skateboarder doing a trick

FIGURE 5.8: More detected tags and sentence generation on COCO, by LSTM-R, LSTM-RT$_2$ and SCN-LSTM.

5.6.3 More results on video captioning

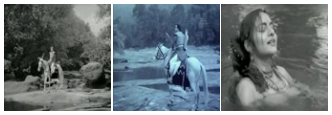
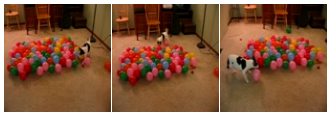
		
Tags: playing (0.694), animal (0.673), baby (0.63), person (0.471), eating (0.419), something (0.333), food (0.329), hand (0.311)	**Tags:** man (0.807), person (0.733), car (0.442), driving (0.39), playing (0.382), road (0.365), moving (0.189), pushing (0.129)	**Tags:** woman (0.88), girl (0.732), lady (0.699), making (0.516), something (0.501), water (0.267), glass (0.244), drinking (0.204)
Generated captions: **LSTM-CR:** a person is eating **LSTM-CRT$_2$:** a person is holding a small animal **SCN-LSTM:** a small animal is eating	**Generated captions:** **LSTM-CR:** a man is doing a wheelie **LSTM-CRT$_2$:** a man is riding a bike **SCN-LSTM:** a man is pushing a car	**Generated captions:** **LSTM-CR:** a woman is pouring sugar in a glass **LSTM-CRT$_2$:** a woman is pouring water **SCN-LSTM:** a woman is drinking something
		
Tags: man (0.635), woman (0.545), riding (0.541), person (0.465), water (0.465), girl (0.4), doing (0.387), horse (0.132)	**Tags:** man (0.843), person (0.774), doing (0.393), playing (0.385), open (0.298), gun (0.283), shooting (0.276), field (0.259)	**Tags:** man (0.958), song (0.869), stage (0.866), singing (0.859), men (0.845), music (0.826), playing (0.762), guitar (0.759)
Generated captions: **LSTM-CR:** a girl is riding a horse **LSTM-CRT$_2$:** a woman is riding a horse **SCN-LSTM:** a man is riding a horse	**Generated captions:** **LSTM-CR:** a girl is firing a gun **LSTM-CRT$_2$:** a girl is shooting **SCN-LSTM:** a man is shooting a gun	**Generated captions:** **LSTM-CR:** a group of people are dancing on stage **LSTM-CRT$_2$:** a man is dancing on stage **SCN-LSTM:** a band is performing on stage
		
Tags: doing (0.616), boy (0.557), room (0.554), playing (0.51), floor (0.493), dancing (0.491), dance (0.361), kid (0.281)	**Tags:** woman (0.829), girl (0.743), doing (0.593), using (0.408), makeup (0.211), applying (0.2), face (0.171), hand (0.139)	**Tags:** playing (0.776), dog (0.625), floor (0.423), trying (0.399), woman (0.356), running (0.293), puppy (0.202), toy (0.182)
Generated captions: **LSTM-CR:** a baby is walking **LSTM-CRT$_2$:** a baby is dancing **SCN-LSTM:** a boy is dancing	**Generated captions:** **LSTM-CR:** a girl is singing **LSTM-CRT$_2$:** a woman is playing **SCN-LSTM:** a woman is plucking her eyebrow	**Generated captions:** **LSTM-CR:** a group of girls are playing with a toy **LSTM-CRT$_2$:** the children are playing **SCN-LSTM:** a dog is playing with a toy

FIGURE 5.9: More detected tags and sentence generation on Youtube2Text, by LSTM-CR, LSTM-CRT$_2$ and SCN-LSTM.

6

Triangle Generative Adversarial Networks

In this chapter, I will present Triangle Generative Adversarial Networks (Δ-GAN) for cross-domain joint distribution matching. Δ-GAN consists of two generators and two discriminators. The generators are designed to learn the two-way conditional distributions between the two domains, while the discriminators are trained to distinguish real data pairs and two kinds of fake data pairs.

6.1 Introduction

Generative adversarial networks (GANs) (Goodfellow et al., 2014) have emerged as a powerful framework for learning generative models of arbitrarily complex data distributions. When trained on datasets of natural images, significant progress has been made on generating realistic and sharp-looking images (Denton et al., 2015; Radford et al., 2016). The original GAN formulation was designed to learn the data distribution in one domain. In practice, one may also be interested in matching two joint distributions. This is an important task, since mapping data samples from one domain to another has a wide range of applications. For instance, matching

the joint distribution of image-text pairs allows simultaneous image captioning and text-conditional image generation (Reed et al., 2016), while image-to-image translation (Isola et al., 2017) is another challenging problem that requires matching the joint distribution of image-image pairs.

In this work, we are interested in designing a GAN framework to match joint distributions. If paired data are available, a simple approach to achieve this is to train a conditional GAN model (Reed et al., 2016; Mirza and Osindero, 2014), from which a joint distribution is readily manifested and can be matched to the empirical joint distribution provided by the paired data. However, fully supervised data are often difficult to acquire. Several methods have been proposed to achieve unsupervised joint distribution matching without any paired data, including DiscoGAN (Kim et al., 2017), CycleGAN (Zhu et al., 2017) and DualGAN (Yi et al., 2017). Adversarially Learned Inference (ALI) (Dumoulin et al., 2017) and Bidirectional GAN (BiGAN) (Donahue et al., 2017) can be readily adapted to this case as well. Though empirically achieving great success, in principle, there exist infinitely many possible mapping functions that satisfy the requirement to map a sample from one domain to another. In order to alleviate this nonidentifiability issue, paired data are needed to provide proper supervision to inform the model the kind of joint distributions that are desired.

This motivates the proposed Triangle Generative Adversarial Network (Δ-GAN), a GAN framework that allows semi-supervised joint distribution matching, where the supervision of domain correspondence is provided by a few paired samples. Δ-GAN consists of two generators and two discriminators. The generators are designed to learn the bidirectional mappings between domains, while the discriminators are trained to distinguish real data pairs and two kinds of fake data pairs. Both the generators and discriminators are trained together via adversarial learning.

Δ-GAN bears close resemblance to Triple GAN (Li et al., 2017b), a recently

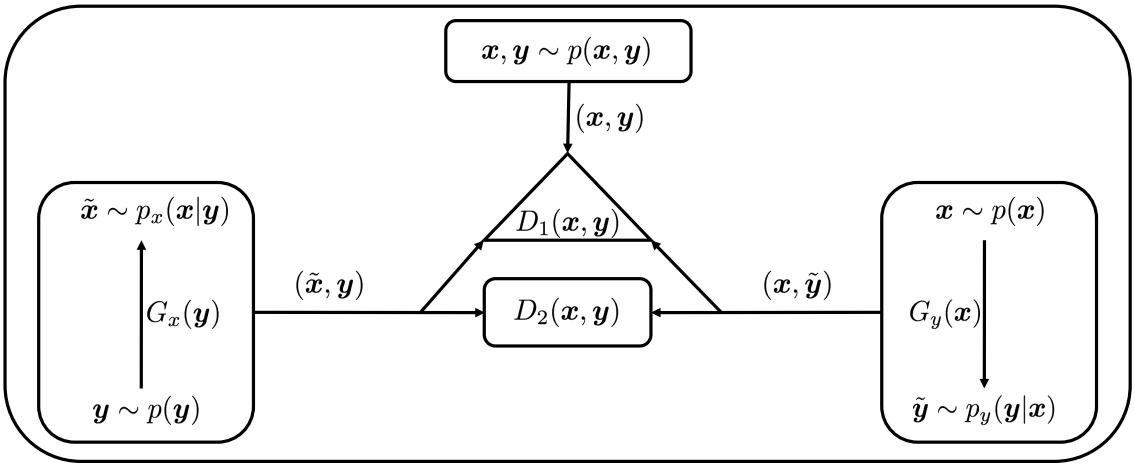


FIGURE 6.1: Illustration of the Triangle Generative Adversarial Network (Δ-GAN).

proposed method that can also be utilized for semi-supervised joint distribution mapping. However, there exist several key differences that make our work unique. First, Δ-GAN uses two discriminators in total, which implicitly defines a ternary discriminative function, instead of a binary discriminator as used in Triple GAN. Second, Δ-GAN can be considered as a combination of conditional GAN and ALI, while Triple GAN consists of two conditional GANs. Third, the distributions characterized by the two generators in both Δ-GAN and Triple GAN concentrate to the data distribution in theory. However, when the discriminator is optimal, the objective of Δ-GAN becomes the Jensen-Shannon divergence (JSD) among three distributions, which is symmetric; the objective of Triple GAN consists of a JSD term plus a Kullback-Leibler (KL) divergence term. The asymmetry of the KL term makes Triple GAN more prone to generating fake-looking samples (Arjovsky and Bottou, 2017). Lastly, the calculation of the additional KL term in Triple GAN is equivalent to calculating a supervised loss, which requires the explicit density form of the conditional distributions, which may not be desirable. On the other hand, Δ-GAN is a fully adversarial approach that does not require that the conditional densities can be computed; Δ-GAN only require that the conditional densities can

be sampled from in a way that allows gradient backpropagation.

Δ-GAN is a general framework, and can be used to match any joint distributions. In experiments, in order to demonstrate the versatility of the proposed model, we consider three domain pairs: image-label, image-image and image-attribute pairs, and use them for semi-supervised classification, image-to-image translation and attribute-based image editing, respectively. In order to demonstrate the scalability of the model to large and complex datasets, we also present attribute-conditional image generation on the COCO dataset (Lin et al., 2014).

6.2 Model

6.2.1 Triangle Generative Adversarial Networks (Δ-GANs)

We now extend GAN to Δ-GAN for joint distribution matching. We first consider Δ-GAN in the supervised setting, and then discuss semi-supervised learning in Section 6.2.3. Consider two related domains, with $\boldsymbol{x}$ and $\boldsymbol{y}$ being the data samples for each domain. We have fully-paired data samples that are characterized by the joint distribution $p(\boldsymbol{x}, \boldsymbol{y})$, which also implies that samples from both the marginal $p(\boldsymbol{x})$ and $p(\boldsymbol{y})$ can be easily obtained.

Δ-GAN consists of two generators: (i) a generator $G_x(\boldsymbol{y})$ that defines the conditional distribution $p_x(\boldsymbol{x}|\boldsymbol{y})$, and (ii) a generator $G_y(\boldsymbol{x})$ that characterizes the conditional distribution in the other direction $p_y(\boldsymbol{y}|\boldsymbol{x})$. $G_x(\boldsymbol{y})$ and $G_y(\boldsymbol{x})$ may also implicitly contain a random latent variable $\boldsymbol{z}$ as input, *i.e.*, $G_x(\boldsymbol{y}, \boldsymbol{z})$ and $G_y(\boldsymbol{x}, \boldsymbol{z})$. In the Δ-GAN game, after a sample $\boldsymbol{x}$ is drawn from $p(\boldsymbol{x})$, the generator G_y produces a pseudo sample $\tilde{\boldsymbol{y}}$ following the conditional distribution $p_y(\boldsymbol{y}|\boldsymbol{x})$. Hence, the fake data pair $(\boldsymbol{x}, \tilde{\boldsymbol{y}})$ is a sample from the joint distribution $p_y(\boldsymbol{x}, \boldsymbol{y}) = p_y(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})$. Similarly, a fake data pair $(\tilde{\boldsymbol{x}}, \boldsymbol{y})$ can be sampled from the generator G_x by first drawing $\boldsymbol{y}$ from $p(\boldsymbol{y})$ and then drawing $\tilde{\boldsymbol{x}}$ from $p_x(\boldsymbol{x}|\boldsymbol{y})$; hence $(\tilde{\boldsymbol{x}}, \boldsymbol{y})$ is sampled from the joint distribution $p_x(\boldsymbol{x}, \boldsymbol{y}) = p_x(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y})$. As such, the generative process between

$p_x(\boldsymbol{x}, \boldsymbol{y})$ and $p_y(\boldsymbol{x}, \boldsymbol{y})$ is reversed.

The objective of Δ-GAN is to match the three joint distributions: $p(\boldsymbol{x}, \boldsymbol{y})$, $p_x(\boldsymbol{x}, \boldsymbol{y})$ and $p_y(\boldsymbol{x}, \boldsymbol{y})$. If this is achieved, we are ensured that we have learned a bidirectional mapping $p_x(\boldsymbol{x}|\boldsymbol{y})$ and $p_y(\boldsymbol{y}|\boldsymbol{x})$ that guarantees the generated fake data pairs $(\tilde{\boldsymbol{x}}, \boldsymbol{y})$ and $(\boldsymbol{x}, \tilde{\boldsymbol{y}})$ are indistinguishable from the true data pairs $(\boldsymbol{x}, \boldsymbol{y})$. In order to match the joint distributions, an adversarial game is played. Joint pairs are drawn from three distributions: $p(\boldsymbol{x}, \boldsymbol{y})$, $p_x(\boldsymbol{x}, \boldsymbol{y})$ or $p_y(\boldsymbol{x}, \boldsymbol{y})$, and two discriminator networks are learned to discriminate among the three, while the two conditional generator networks are trained to fool the discriminators.

The value function describing the game is given by

$$\begin{aligned} \min_{G_x, G_y} \max_{D_1, D_2} V(G_x, G_y, D_1, D_2) &= \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}) \sim p(\boldsymbol{x}, \boldsymbol{y})}[\log D_1(\boldsymbol{x}, \boldsymbol{y})] \\ &+ \mathbb{E}_{\boldsymbol{y} \sim p(\boldsymbol{y}), \tilde{\boldsymbol{x}} \sim p_x(\boldsymbol{x}|\boldsymbol{y})}\Big[\log\Big((1 - D_1(\tilde{\boldsymbol{x}}, \boldsymbol{y})) \cdot D_2(\tilde{\boldsymbol{x}}, \boldsymbol{y})\Big)\Big] \\ &+ \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x}), \tilde{\boldsymbol{y}} \sim p_y(\boldsymbol{y}|\boldsymbol{x})}\Big[\log\Big((1 - D_1(\boldsymbol{x}, \tilde{\boldsymbol{y}})) \cdot (1 - D_2(\boldsymbol{x}, \tilde{\boldsymbol{y}}))\Big)\Big]. \end{aligned} \tag{6.1}$$

The discriminator D_1 is used to distinguish whether a sample pair is from $p(\boldsymbol{x}, \boldsymbol{y})$ or not, if this sample pair is not from $p(\boldsymbol{x}, \boldsymbol{y})$, another discriminator D_2 is used to distinguish whether this sample pair is from $p_x(\boldsymbol{x}, \boldsymbol{y})$ or $p_y(\boldsymbol{x}, \boldsymbol{y})$. D_1 and D_2 work cooperatively, and the use of both implicitly defines a ternary discriminative function D that distinguish sample pairs in three ways. See Figure 6.1 for an illustration of the adversarial game and Section 6.6.2 for an algorithmic description of the training procedure.

6.2.2 Theoretical analysis

Δ-GAN shares many of the theoretical properties of GANs (Goodfellow et al., 2014). We first consider the optimal discriminators D_1 and D_2 for any given generator G_x and G_y. These optimal discriminators then allow reformulation of objec-

tive (6.1), which reduces to the Jensen-Shannon divergence among the joint distribution $p(\boldsymbol{x}, \boldsymbol{y}), p_x(\boldsymbol{x}, \boldsymbol{y})$ and $p_y(\boldsymbol{x}, \boldsymbol{y})$.

Proposition 1. *For any fixed generator G_x and G_y, the optimal discriminator D_1 and D_2 of the game defined by $V(G_x, G_y, D_1, D_2)$ is*

$$D_1^*(\boldsymbol{x}, \boldsymbol{y}) = \frac{p(\boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{x}, \boldsymbol{y}) + p_x(\boldsymbol{x}, \boldsymbol{y}) + p_y(\boldsymbol{x}, \boldsymbol{y})}, \; D_2^*(\boldsymbol{x}, \boldsymbol{y}) = \frac{p_x(\boldsymbol{x}, \boldsymbol{y})}{p_x(\boldsymbol{x}, \boldsymbol{y}) + p_y(\boldsymbol{x}, \boldsymbol{y})}\,.$$

Proof. The proof is a straightforward extension of the proof in Goodfellow et al. (2014). See Section 6.6.1 for details. □

Proposition 2. *The equilibrium of $V(G_x, G_y, D_1, D_2)$ is achieved if and only if $p(\boldsymbol{x}, \boldsymbol{y}) = p_x(\boldsymbol{x}, \boldsymbol{y}) = p_y(\boldsymbol{x}, \boldsymbol{y})$ with $D_1^*(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{3}$ and $D_2^*(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}$, and the optimum value is $-3 \log 3$.*

Proof. Given the optimal $D_1^*(\boldsymbol{x}, \boldsymbol{y})$ and $D_2^*(\boldsymbol{x}, \boldsymbol{y})$, the minimax game can be reformulated as:

$$C(G_x, G_y) = \max_{D_1, D_2} V(G_x, G_y, D_1, D_2) \tag{6.2}$$

$$= -3 \log 3 + 3 \cdot JSD\Big(p(\boldsymbol{x}, \boldsymbol{y}), p_x(\boldsymbol{x}, \boldsymbol{y}), p_y(\boldsymbol{x}, \boldsymbol{y})\Big) \geqslant -3 \log 3\,, \tag{6.3}$$

where JSD denotes the Jensen-Shannon divergence (JSD) among three distributions. See Section 6.6.1 for details. □

Since $p(\boldsymbol{x}, \boldsymbol{y}) = p_x(\boldsymbol{x}, \boldsymbol{y}) = p_y(\boldsymbol{x}, \boldsymbol{y})$ can be achieved in theory, it can be readily seen that the learned conditional generators can reveal the true conditional distributions underlying the data, *i.e.*, $p_x(\boldsymbol{x}|\boldsymbol{y}) = p(\boldsymbol{x}|\boldsymbol{y})$ and $p_y(\boldsymbol{y}|\boldsymbol{x}) = p(\boldsymbol{y}|\boldsymbol{x})$.

6.2.3 Semi-supervised learning

In order to further understand Δ-GAN, we write (6.1) as

$$V = \underbrace{\mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}[\log D_1(\boldsymbol{x},\boldsymbol{y})] + \mathbb{E}_{p_x(\tilde{\boldsymbol{x}},\boldsymbol{y})}[\log(1 - D_1(\tilde{\boldsymbol{x}},\boldsymbol{y}))] + \mathbb{E}_{p_y(\boldsymbol{x},\tilde{\boldsymbol{y}})}[\log(1 - D_1(\boldsymbol{x},\tilde{\boldsymbol{y}}))]}_{\text{conditional GAN}} \tag{6.4}$$

$$+ \underbrace{\mathbb{E}_{p_x(\tilde{\boldsymbol{x}},\boldsymbol{y})}[\log D_2(\tilde{\boldsymbol{x}},\boldsymbol{y})] + \mathbb{E}_{p_y(\boldsymbol{x},\tilde{\boldsymbol{y}})}[\log(1 - D_2(\boldsymbol{x},\tilde{\boldsymbol{y}}))]}_{\text{BiGAN/ALI}} . \tag{6.5}$$

The objective of Δ-GAN is a combination of the objectives of conditional GAN and BiGAN. The BiGAN part matches two joint distributions: $p_x(\boldsymbol{x},\boldsymbol{y})$ and $p_y(\boldsymbol{x},\boldsymbol{y})$, while the conditional GAN part provides the supervision signal to notify the BiGAN part what joint distribution to match. Therefore, Δ-GAN provides a natural way to perform semi-supervised learning, since the conditional GAN part and the BiGAN part can be used to account for paired and unpaired data, respectively.

However, when doing semi-supervised learning, there is also one potential problem that we need to be cautious about. The theoretical analysis in Section 6.2.2 is based on the assumption that the dataset is fully supervised, *i.e.*, we have the ground-truth joint distribution $p(\boldsymbol{x},\boldsymbol{y})$ and marginal distributions $p(\boldsymbol{x})$ and $p(\boldsymbol{y})$. In the semi-supervised setting, $p(\boldsymbol{x})$ and $p(\boldsymbol{y})$ are still available but $p(\boldsymbol{x},\boldsymbol{y})$ is not. We can only obtain the joint distribution $p_l(\boldsymbol{x},\boldsymbol{y})$ characterized by the few paired data samples. Hence, in the semi-supervised setting, $p_x(\boldsymbol{x},\boldsymbol{y})$ and $p_y(\boldsymbol{x},\boldsymbol{y})$ will try to concentrate to the empirical distribution $p_l(\boldsymbol{x},\boldsymbol{y})$. We make the assumption that $p_l(\boldsymbol{x},\boldsymbol{y}) \approx p(\boldsymbol{x},\boldsymbol{y})$, *i.e.*, the paired data can roughly characterize the whole dataset. For example, in the semi-supervised classification problem, one usually strives to make sure that labels are equally distributed among the labeled dataset.

6.2.4 Relation to Triple GAN

Δ-GAN is closely related to Triple GAN (Li et al., 2017b). Below we review Triple GAN and then discuss the main differences. The value function of Triple GAN is defined as follows:

$$\begin{aligned} V =& \mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}[\log D(\boldsymbol{x},\boldsymbol{y})] + (1-\alpha)\mathbb{E}_{p_x(\tilde{\boldsymbol{x}},\boldsymbol{y})}[\log(1-D(\tilde{\boldsymbol{x}},\boldsymbol{y}))] \\ &+\alpha\mathbb{E}_{p_y(\boldsymbol{x},\tilde{\boldsymbol{y}})}[\log(1-D(\boldsymbol{x},\tilde{\boldsymbol{y}}))] + \mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}[-\log p_y(\boldsymbol{y}|\boldsymbol{x})]\,, \end{aligned} \tag{6.6}$$

where $\alpha \in (0,1)$ is a contant that controls the relative importance of the two generators. Let Triple GAN-s denote a simplified Triple GAN model with only the first three terms. As can be seen, Triple GAN-s can be considered as a combination of two conditional GANs, with the importance of each condtional GAN weighted by α. It can be proven that Triple GAN-s achieves equilibrium if and only if $p(\boldsymbol{x},\boldsymbol{y}) = (1-\alpha)p_x(\boldsymbol{x},\boldsymbol{y}) + \alpha p_y(\boldsymbol{x},\boldsymbol{y})$, which is not desirable. To address this problem, in Triple GAN a standard supervised loss $\mathcal{R}_{\mathcal{L}} = \mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}[-\log p_y(\boldsymbol{y}|\boldsymbol{x})]$ is added. As a result, when the discriminator is optimal, the cost function in Triple GAN becomes:

$$2JSD\Big(p(\boldsymbol{x},\boldsymbol{y})||((1-\alpha)p_x(\boldsymbol{x},\boldsymbol{y}) + \alpha p_y(\boldsymbol{x},\boldsymbol{y}))\Big) + KL(p(\boldsymbol{x},\boldsymbol{y})||p_y(\boldsymbol{x},\boldsymbol{y})) + \text{const.} \tag{6.7}$$

This cost function has the good property that it has a unique minimum at $p(\boldsymbol{x},\boldsymbol{y}) = p_x(\boldsymbol{x},\boldsymbol{y}) = p_y(\boldsymbol{x},\boldsymbol{y})$. However, the objective becomes asymmetrical. The second KL term pays low cost for generating fake-looking samples (Arjovsky and Bottou, 2017). By contrast Δ-GAN directly optimizes the *symmetric* Jensen-Shannon divergence among three distributions. More importantly, the calculation of $\mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}[-\log p_y(\boldsymbol{y}|\boldsymbol{x})]$ in Triple GAN also implies that the explicit density form of $p_y(\boldsymbol{y}|\boldsymbol{x})$ should be provided, which may not be desirable. On the other hand, Δ-GAN only requires that $p_y(\boldsymbol{y}|\boldsymbol{x})$ can be sampled from. For example, if we assume $p_y(\boldsymbol{y}|\boldsymbol{x}) = \int \delta(\boldsymbol{y} -$

$G_y(\boldsymbol{x}, \boldsymbol{z}))p(\boldsymbol{z})d\boldsymbol{z}$, and $\delta(\cdot)$ is the Dirac delta function, we can sample $\boldsymbol{y}$ through sampling $\boldsymbol{z}$, however, the density function of $p_y(\boldsymbol{y}|\boldsymbol{x})$ is not explicitly available.

6.2.5 Applications

Δ-GAN is a general framework that can be used for any joint distribution matching. Besides the semi-supervised image classification task considered in Li et al. (2017b), we also conduct experiments on image-to-image translation and attribute-conditional image generation. When modeling image pairs, both $p_x(\boldsymbol{x}|\boldsymbol{y})$ and $p_y(\boldsymbol{y}|\boldsymbol{x})$ are implemented without introducing additional latent variables, *i.e.*, $p_x(\boldsymbol{x}|\boldsymbol{y}) = \delta(\boldsymbol{x} - G_x(\boldsymbol{y}))$, $p_y(\boldsymbol{y}|\boldsymbol{x}) = \delta(\boldsymbol{y} - G_y(\boldsymbol{x}))$.

A different strategy is adopted when modeling the image-label/attribute pairs. Specifically, let $\boldsymbol{x}$ denote samples in the image domain, $\boldsymbol{y}$ denote samples in the label/attribute domain. $\boldsymbol{y}$ is a one-hot vector or a binary vector when representing labels and attributes, respectively. When modeling $p_x(\boldsymbol{x}|\boldsymbol{y})$, we assume that $\boldsymbol{x}$ is transformed by the latent style variables $\boldsymbol{z}$ given the label or attribute vector $\boldsymbol{y}$, *i.e.*, $p_x(\boldsymbol{x}|\boldsymbol{y}) = \int \delta(\boldsymbol{x} - G_x(\boldsymbol{y}, \boldsymbol{z}))p(\boldsymbol{z})d\boldsymbol{z}$, where $p(\boldsymbol{z})$ is chosen to be a simple distribution (*e.g.*, uniform or standard normal). When learning $p_y(\boldsymbol{y}|\boldsymbol{x})$, $p_y(\boldsymbol{y}|\boldsymbol{x})$ is assumed to be a standard multi-class or multi-label classfier without latent variables $\boldsymbol{z}$. In order to allow the training signal backpropagated from D_1 and D_2 to G_y, we adopt the REINFORCE algorithm as in Li et al. (2017b), and use the label with the maximum probability to approximate the expectation over $\boldsymbol{y}$, or use the output of the sigmoid function as the predicted attribute vector.

6.3 Related work

The proposed framework focuses on designing GAN for joint-distribution matching. Conditional GAN can be used for this task if supervised data is available. Various conditional GANs have been proposed to condition the image generation on class

labels (Mirza and Osindero, 2014), attributes (Perarnau et al., 2016), texts (Reed et al., 2016; Zhang et al., 2017a; Xu et al., 2017) and images (Isola et al., 2017; Ledig et al., 2017). Unsupervised learning methods have also been developed for this task. BiGAN (Donahue et al., 2017) and ALI (Dumoulin et al., 2017) proposed a method to jointly learn a generation network and an inference network via adversarial learning. Though originally designed for learning the two-way transition between the stochastic latent variables and real data samples, BiGAN and ALI can be directly adapted to learn the joint distribution of two real domains. Another method is called DiscoGAN (Kim et al., 2017), in which two generators are used to model the bidirectional mapping between domains, and another two discriminators are used to decide whether a generated sample is fake or not in each individual domain. Further, additional reconstructon losses are introduced to make the two generators strongly coupled and also alleviate the problem of mode collapsing. Similiar work includes CycleGAN (Zhu et al., 2017), DualGAN (Yi et al., 2017) and DTN (Taigman et al., 2017). Additional weight-sharing constraints are introduced in CoGAN (Liu and Tuzel, 2016) and UNIT (Liu et al., 2017).

Our work differs from the above work in that we aim at semi-supervised joint distribution matching. The only work that we are aware of that also achieves this goal is Triple GAN. However, our model is distinct from Triple GAN in important ways (see Section 6.2.4). Further, Triple GAN only focuses on image classification, while Δ-GAN has been shown to be applicable to a wide range of applications.

Various methods and model architectures have been proposed to improve and stabilize the training of GAN, such as feature matching (Salimans et al., 2016; Zhang et al., 2016c, 2017b), Wasserstein GAN (Arjovsky et al., 2017), energy-based GAN (Zhao et al., 2017), and unrolled GAN (Metz et al., 2017) among many other related works. Our work is orthogonal to these methods, which could also be used to improve the training of Δ-GAN. Instead of using adversarial loss, there also exists

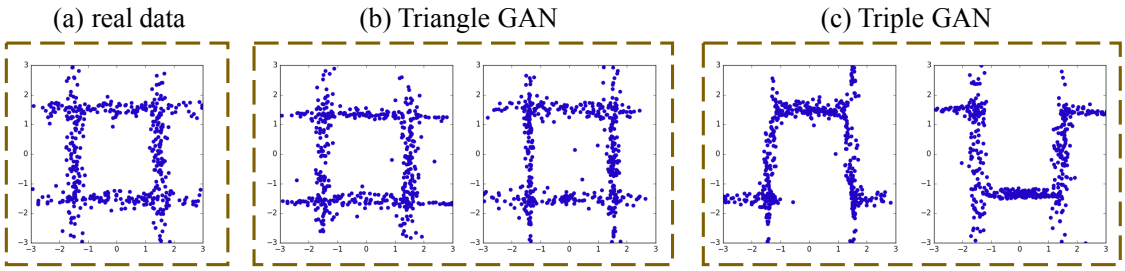


FIGURE 6.2: Toy data experiment on Δ-GAN and Triple GAN.

work that uses supervised learning (Xia et al., 2017) for joint-distribution matching, and variational autoencoders for semi-supervised learning (Pu et al., 2016b, 2017b). Lastly, our work is also closely related to the recent work of Li et al. (2017a); Pu et al. (2017a); Chen et al. (2018), which treats one of the domains as latent variables.

6.4 Experiments

We present results on three tasks: (*i*) semi-supervised classification on CIFAR10 (Krizhevsky, 2009); (*ii*) image-to-image translation on MNIST (LeCun et al., 1998) and the edges2shoes dataset (Isola et al., 2017); and (*iii*) attribute-to-image generation on CelebA (Liu et al., 2015) and COCO (Lin et al., 2014). We also conduct a toy data experiment to further demonstrate the differences between Δ-GAN and Triple GAN. We implement Δ-GAN without introducing additional regularization unless explicitly stated. All the network architectures are provided in the Appendix.

6.4.1 Toy data experiment

We first compare our method with Triple GAN on a toy dataset. We synthesize data by drawing $(x, y) \sim \frac{1}{4}\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + \frac{1}{4}\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) + \frac{1}{4}\mathcal{N}(\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3) + \frac{1}{4}\mathcal{N}(\boldsymbol{\mu}_4, \boldsymbol{\Sigma}_4)$, where $\boldsymbol{\mu}_1 = [0, 1.5]^\top$, $\boldsymbol{\mu}_2 = [-1.5, 0]^\top$, $\boldsymbol{\mu}_3 = [1.5, 0]^\top$, $\boldsymbol{\mu}_4 = [0, -1.5]^\top$, $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_4 = \left(\begin{smallmatrix} 3 & 0 \\ 0 & 0.025 \end{smallmatrix}\right)$ and $\boldsymbol{\Sigma}_2 = \boldsymbol{\Sigma}_3 = \left(\begin{smallmatrix} 0.025 & 0 \\ 0 & 3 \end{smallmatrix}\right)$. We generate 5000 (x, y) pairs for each mixture component.

Table 6.1: Error rates (%) on the partially labeled CIFAR10 dataset.

Algorithm	$n = 4000$
CatGAN (Springenberg, 2015)	19.58 $\pm$ 0.58
Improved GAN (Salimans et al., 2016)	18.63 $\pm$ 2.32
ALI (Dumoulin et al., 2017)	17.99 $\pm$ 1.62
Triple GAN (Li et al., 2017b)	16.99 $\pm$ 0.36
Δ-GAN (ours)	**16.80** $\pm$ **0.42**

In order to implement Δ-GAN and Triple GAN-s, we model $p_x(x|y)$ and $p_y(y|x)$ as

$$p_x(x|y) = \int \delta(x - G_x(y, \boldsymbol{z}))p(\boldsymbol{z})d\boldsymbol{z}, \;\; p_y(y|x) = \int \delta(y - G_y(x, \boldsymbol{z}))p(\boldsymbol{z})d\boldsymbol{z}\,, \qquad (6.8)$$

where both G_x and G_y are modeled as a 4-hidden-layer multilayer perceptron (MLP) with 500 hidden units in each layer. $p(\boldsymbol{z})$ is a bivariate standard Gaussian distribution. Triple GAN can be implemented by specifying both $p_x(x|y)$ and $p_y(y|x)$ to be distributions with explicit density form, *e.g.*, Gaussian distributions. However, the performance can be bad since it fails to capture the multi-modality of $p_x(x|y)$ and $p_y(y|x)$. Hence, only Triple GAN-s is implemented.

Results are shown in Figure 6.2. (a) presents the joint distribution $p(x, y)$ of real data. For (b) and (c), the left and right figure is the learned joint distribution $p_x(x, y)$ and $p_y(x, y)$, respectively. The joint distributions $p_x(x, y)$ and $p_y(x, y)$ learned by Δ-GAN successfully match the true joint distribution $p(x, y)$. Triple GAN-s cannot achieve this, and can only guarantee $\frac{1}{2}(p_x(x, y) + p_y(x, y))$ matches $p(x, y)$. Although this experiment is limited due to its simplicity, the results clearly support the advantage of our proposed model over Triple GAN.

6.4.2 Semi-supervised classification

We evaluate semi-supervised classification on the CIFAR10 dataset with 4000 labels. The labeled data is distributed equally across classes and the results are averaged over 10 runs with different random splits of the training data. For fair comparison, we follow the publically available code of Triple GAN and use the same regularization

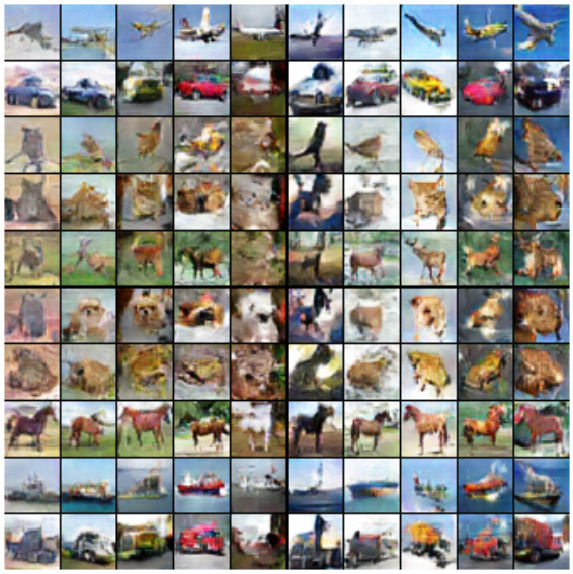


FIGURE 6.3: Generated CIFAR10 samples, where each row shares the same label and each column uses the same noise.

Table 6.2: Classification accuracy (%) on the MNIST-to-MNIST-transpose dataset.

Algorithm	$n = 100$	$n = 1000$	All
DiscoGAN	–	–	15.00± 0.20
Triple GAN	63.79 ± 0.85	84.93 ± 1.63	86.70 ± 1.52
Δ-GAN	**83.20± 1.88**	**88.98± 1.50**	**93.34± 1.46**

terms and hyperparameter settings as theirs. Results are summarized in Table 6.1. Our Δ-GAN achieves the best performance among all the competing methods. We also show the ability of Δ-GAN to disentangle classes and styles in Figure 6.3. Δ-GAN can generate realistic data in a specific class and the injected noise vector encodes meaningful style patterns like background and color.

6.4.3 Image-to-image translation

We first evaluate image-to-image translation on the edges2shoes dataset. Results are shown in Figure 6.4(bottom). Though DiscoGAN is an unsupervised learning method, it achieves impressive results. However, with supervision provided by 10% paired data, Δ-GAN generally generates more accurate edge details of the shoes. In

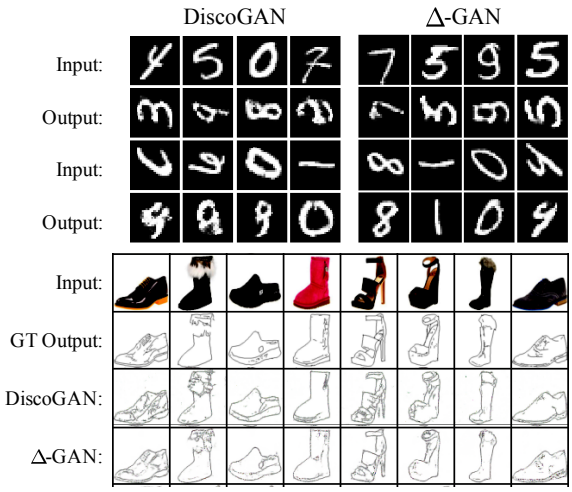


FIGURE 6.4: Image-to-image translation experiments on the MNIST-to-MNIST-transpose and edges2shoes datasets.

order to provide quantitative evaluation of translating shoes to edges, we use mean squared error (MSE) as our metric. The MSE of using DiscoGAN is 140.1; with 10%, 20%, 100% paired data, the MSE of using Δ-GAN is 125.3, 113.0 and 66.4, respectively.

To further demonstrate the importance of providing supervision of domain correspondence, we created a new dataset based on MNIST (LeCun et al., 1998), where the two image domains are the MNIST images and their corresponding tranposed ones. As can be seen in Figure 6.4(top), Δ-GAN matches images betwen domains well, while DiscoGAN fails in this task. For supporting quantitative evaluation, we have trained a classifier on the MNIST dataset, and the classification accuracy of this classifier on the test set approaches 99.4%, and is, therefore, trustworthy as an evaluation metric. Given an input MNIST image $\boldsymbol{x}$, we first generate a transposed image $\boldsymbol{y}$ using the learned generator, and then manually transpose it back to normal digits $\boldsymbol{y}^T$, and finally send this new image $\boldsymbol{y}^T$ to the classifier. Results are summarized in Table 6.2, which are averages over 5 runs with different random splits of the training data. Δ-GAN achieves significantly better performance than Triple GAN

FIGURE 6.5: Results on the face-to-attribute-to-face experiment.

Table 6.3: Results of P@10 and nDCG@10 for attribute predicting on CelebA and COCO.

Dataset	**CelebA**		
Method	**1%**	**10%**	**100%**
Triple GAN	40.97/50.74	62.13/73.56	70.12/79.37
Δ-GAN	**53.21/58.39**	**63.68/75.22**	**70.37/81.47**
Dataset	**COCO**		
Method	**10%**	**50%**	**100%**
Triple GAN	32.64/35.91	34.00/37.76	35.35/39.60
Δ-GAN	**34.38/37.91**	**36.72/40.39**	**39.05/42.86**

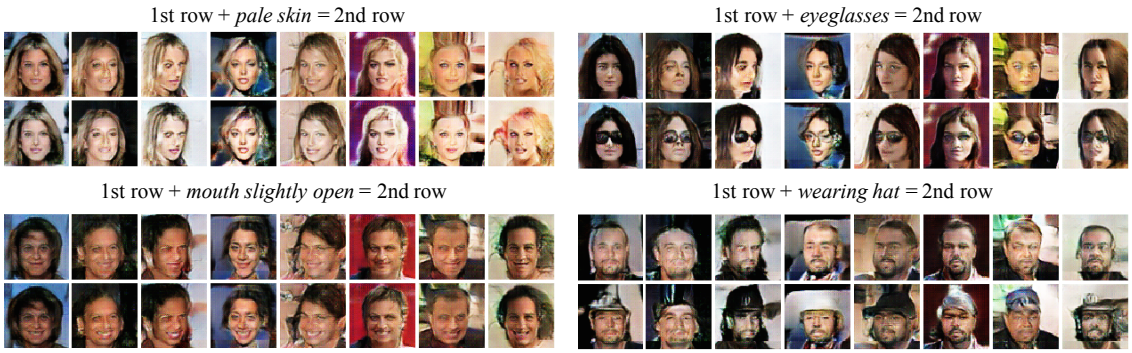


FIGURE 6.6: Results on the image editing experiment.

and DiscoGAN.

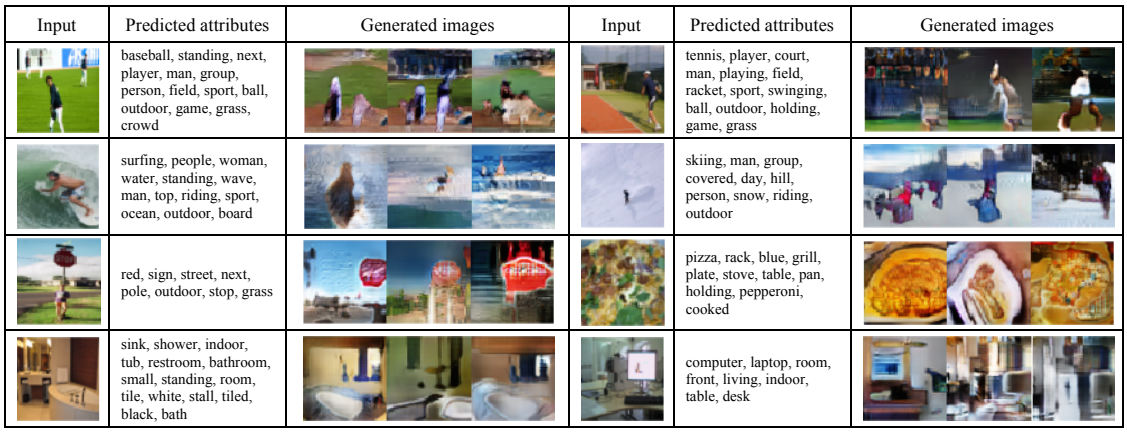


FIGURE 6.7: Results on the image-to-attribute-to-image experiment.

6.4.4 Attribute-conditional image generation

We apply our method to face images from the CelebA dataset. This dataset consists of 202,599 images annotated with 40 binary attributes. We scale and crop the images to 64×64 pixels. In order to qualitatively evaluate the learned attribute-conditional image generator and the multi-label classifier, given an input face image, we first use the classifier to predict attributes, and then use the image generator to produce images based on the predicted attributes. Figure 6.5 shows example results. The 1st row is the input images; the 2nd row is the predicted attributes given the input images; the 3rd row is the generated images given the predicted attributes. Both the learned attribute predictor and the image generator provides good results. We further show another set of image editing experiment in Figure 6.6. For each subfigure, we use a same set of attributes with different noise vectors to generate images. For example, for the top-right subfigure, all the images in the 1st row were generated based on the following attributes: *black hair, female, attractive*, and we then added the attribute of "*sunglasses*" when generating the images in the 2nd row. It is interesting to see that Δ-GAN has great flexibility to adjust the generated images by changing certain input attribtutes. For instance, by switching on the *wearing hat*

attribute, one can edit the face image to have a hat on the head.

In order to demonstrate the scalablility of our model to large and complex datasets, we also present results on the COCO dataset. Following Gan et al. (2017e), we first select a set of 1000 attributes from the caption text in the training set, which includes the most frequent nouns, verbs, or adjectives. The images in COCO are scaled and cropped to have 64×64 pixels. Unlike the case of CelebA face images, the networks need to learn how to handle multiple objects and diverse backgrounds. Results are provided in Figure 6.7. We can generate reasonably good images based on the predicted attributes. The input and generated images also clearly share a same set of attributes. We also observe diversity in the samples by simply drawing multple noise vectors and using the same predicted attributes.

Precision (P) and normalized Discounted Cumulative Gain (nDCG) are two popular evaluation metrics for multi-label classification problems. Table 6.3 provides the quantatitive results of P@10 and nDCG@10 on CelebA and COCO, where @k means at rank k (see Section 6.6.4 for definitions). For fair comparison, we use the same network architecures for both Triple GAN and Δ-GAN. Δ-GAN consistently provides better results than Triple GAN. On the COCO dataset, our semi-supervised learning approach with 50% labeled data achieves better performance than the results of Triple GAN using the full dataset, demonstrating the effectiveness of our approach for semi-supervised joint distribution matching. More results for the above experiments are provided in Section 6.6.3.

6.5 Conclusion

We have presented the Triangle Generative Adversarial Network (Δ-GAN), a new GAN framework that can be used for semi-supervised joint distribution matching. Our approach learns the bidirectional mappings between two domains with a few paired samples. We have demonstrated that Δ-GAN may be employed for a wide

range of applications. One possible future direction is to combine Δ-GAN with sequence GAN (Yu et al., 2017) or textGAN (Zhang et al., 2017b) to model the joint distribution of image-caption pairs.

6.6 Supplementary Material

6.6.1 Detailed theoretical analysis

Proposition 3. *For any fixed generator G_x and G_y, the optimal discriminator D_1 and D_2 of the game defined by the value function $V(G_x, G_y, D_1, D_2)$ is*

$$D_1^*(\boldsymbol{x}, \boldsymbol{y}) = \frac{p(\boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{x}, \boldsymbol{y}) + p_x(\boldsymbol{x}, \boldsymbol{y}) + p_y(\boldsymbol{x}, \boldsymbol{y})},\ D_2^*(\boldsymbol{x}, \boldsymbol{y}) = \frac{p_x(\boldsymbol{x}, \boldsymbol{y})}{p_x(\boldsymbol{x}, \boldsymbol{y}) + p_y(\boldsymbol{x}, \boldsymbol{y})}\,.$$

Proof. The training criterion for the discriminator D_1 and D_2, given any generator G_x and G_y, is to maximize the quantity $V(G_x, G_y, D_1, D_2)$:

$$\begin{aligned} V(G_x, G_y, D_1, D_2) = &\int_{\boldsymbol{x}}\int_{\boldsymbol{y}} p(\boldsymbol{x}, \boldsymbol{y}) \log D_1(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{x} d\boldsymbol{y} + \int_{\boldsymbol{x}}\int_{\boldsymbol{y}} p_x(\boldsymbol{x}, \boldsymbol{y}) \log(1 - D_1(\boldsymbol{x}, \boldsymbol{y})) d\boldsymbol{x} d\boldsymbol{y} \\ &+ \int_{\boldsymbol{x}}\int_{\boldsymbol{y}} p_x(\boldsymbol{x}, \boldsymbol{y}) \log D_2(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{x} d\boldsymbol{y} + \int_{\boldsymbol{x}}\int_{\boldsymbol{y}} p_y(\boldsymbol{x}, \boldsymbol{y}) \log(1 - D_1(\boldsymbol{x}, \boldsymbol{y})) d\boldsymbol{x} d\boldsymbol{y} \\ &+ \int_{\boldsymbol{x}}\int_{\boldsymbol{y}} p_y(\boldsymbol{x}, \boldsymbol{y}) \log(1 - D_2(\boldsymbol{x}, \boldsymbol{y})) d\boldsymbol{x} d\boldsymbol{y}\,. \end{aligned}$$

Following Goodfellow et al. (2014), for any $(a, b) \in \mathbb{R}^2 \backslash \{0, 0\}$, the function $y \to a \log y + b \log(1 - y)$ achieves its maximum in $[0, 1]$ at $\frac{a}{a+b}$. This concludes the proof. □

Proposition 4. *The equilibrium of $V(G_x, G_y, D_1, D_2)$ is achieved if and only if $p(\boldsymbol{x}, \boldsymbol{y}) = p_x(\boldsymbol{x}, \boldsymbol{y}) = p_y(\boldsymbol{x}, \boldsymbol{y})$ with $D_1^*(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{3}$ and $D_2^*(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}$, and the optimum value is $-3 \log 3$.*

Proof. Given the optimal $D_1^*(\boldsymbol{x}, \boldsymbol{y})$ and $D_2^*(\boldsymbol{x}, \boldsymbol{y})$, the minimax game can be refor-

mulated as:

$$C(G_x, G_y) = \max_{D_1, D_2} V(G_x, G_y, D_1, D_2) \tag{6.9}$$

$$= \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p(\boldsymbol{x},\boldsymbol{y})}\Big[\log \frac{p(\boldsymbol{x},\boldsymbol{y})}{p(\boldsymbol{x},\boldsymbol{y}) + p_x(\boldsymbol{x},\boldsymbol{y}) + p_y(\boldsymbol{x},\boldsymbol{y})}\Big] \tag{6.10}$$

$$+ \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p_x(\boldsymbol{x},\boldsymbol{y})}\Big[\log \frac{p_x(\boldsymbol{x},\boldsymbol{y})}{p(\boldsymbol{x},\boldsymbol{y}) + p_x(\boldsymbol{x},\boldsymbol{y}) + p_y(\boldsymbol{x},\boldsymbol{y})}\Big] \tag{6.11}$$

$$+ \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p_y(\boldsymbol{x},\boldsymbol{y})}\Big[\log \frac{p_y(\boldsymbol{x},\boldsymbol{y})}{p(\boldsymbol{x},\boldsymbol{y}) + p_x(\boldsymbol{x},\boldsymbol{y}) + p_y(\boldsymbol{x},\boldsymbol{y})}\Big]. \tag{6.12}$$

Note that

$$C(G_1, G_2) = -3\log 3 + KL\Big(p(\boldsymbol{x},\boldsymbol{y})\Big\|\frac{p(\boldsymbol{x},\boldsymbol{y}) + p_x(\boldsymbol{x},\boldsymbol{y}) + p_y(\boldsymbol{x},\boldsymbol{y})}{3}\Big) \tag{6.13}$$

$$+ KL\Big(p_x(\boldsymbol{x},\boldsymbol{y})\Big\|\frac{p(\boldsymbol{x},\boldsymbol{y}) + p_x(\boldsymbol{x},\boldsymbol{y}) + p_y(\boldsymbol{x},\boldsymbol{y})}{3}\Big) \tag{6.14}$$

$$+ KL\Big(p_y(\boldsymbol{x},\boldsymbol{y})\Big\|\frac{p(\boldsymbol{x},\boldsymbol{y}) + p_x(\boldsymbol{x},\boldsymbol{y}) + p_y(\boldsymbol{x},\boldsymbol{y})}{3}\Big). \tag{6.15}$$

Therefore,

$$C(G_1, G_2) = -3\log 3 + 3 \cdot JSD\Big(p(\boldsymbol{x},\boldsymbol{y}), p_x(\boldsymbol{x},\boldsymbol{y}), p_y(\boldsymbol{x},\boldsymbol{y})\Big) \geqslant -3\log 3\,, \tag{6.16}$$

where $JSD_{\pi_1,\ldots,\pi_n}(p_1, p_2, \ldots, p_n) = H\Big(\sum_{i=1}^n \pi_i p_i\Big) - \sum_{i=1}^n \pi_i H(p_i)$ is the Jensen-Shannon divergence. $\pi_1, \ldots, \pi_n$ are weights that are selected for the probability distribution $p_1, p_2, \ldots, p_n$, and $H(p)$ is the entropy for distribution p. In the three-distribution case described above, we set $n = 3$ and $\pi_1 = \pi_2 = \pi_3 = \frac{1}{3}$.

For $p(\boldsymbol{x},\boldsymbol{y}) = p_x(\boldsymbol{x},\boldsymbol{y}) = p_y(\boldsymbol{x},\boldsymbol{y})$, we have $D_1^*(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{3}$, $D_2^*(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{2}$ and $C(G_x, G_y) = -3\log 3$. Since the Jensen-Shannon divergence is always non-negative, and zero iff they are equal, we have shown that $C^* = -3\log 3$ is the global minimum of $C(G_x, G_y)$ and that the only solution is $p(\boldsymbol{x},\boldsymbol{y}) = p_x(\boldsymbol{x},\boldsymbol{y}) = p_y(\boldsymbol{x},\boldsymbol{y})$, *i.e.*, the generative models perfectly replicating the data distribution. □

Algorithm 3 Δ-GAN training procedure.

$\boldsymbol{\theta}_g, \boldsymbol{\theta}_d \leftarrow$ initialize network parameters
repeat
  $(\boldsymbol{x}_p^{(1)}, \boldsymbol{y}_p^{(1)}), \ldots, (\boldsymbol{x}_p^{(M)}, \boldsymbol{y}_p^{(M)}) \sim p(\boldsymbol{x}, \boldsymbol{y})$ ▷ Get M paired data samples
  $\boldsymbol{x}_u^{(1)}, \ldots, \boldsymbol{x}_u^{(M)} \sim p(\boldsymbol{x})$ ▷ Get M unpaired data samples
  $\boldsymbol{y}_u^{(1)}, \ldots, \boldsymbol{y}_u^{(M)} \sim p(\boldsymbol{y})$
  $\tilde{\boldsymbol{x}}_u^{(i)} \sim p_x(\boldsymbol{x}|\boldsymbol{y} = \boldsymbol{y}_u^{(i)}), \quad i = 1, \ldots, M$ ▷ Sample from the conditionals
  $\tilde{\boldsymbol{y}}_u^{(j)} \sim p_y(\boldsymbol{y}|\boldsymbol{x} = \boldsymbol{x}_u^{(j)}), \quad j = 1, \ldots, M$
  $\rho_{11}^{(i)} \leftarrow D_1(\boldsymbol{x}_p^{(i)}, \boldsymbol{y}_p^{(i)}), \quad i = 1, \ldots, M$ ▷ Compute discriminator predictions
  $\rho_{12}^{(i)} \leftarrow D_1(\tilde{\boldsymbol{x}}_u^{(i)}, \boldsymbol{y}_u^{(i)}),\ \rho_{13}^{(i)} \leftarrow D_1(\boldsymbol{x}_u^{(i)}, \tilde{\boldsymbol{y}}_u^{(i)}), \quad i = 1, \ldots, M$
  $\rho_{21}^{(i)} \leftarrow D_2(\tilde{\boldsymbol{x}}_u^{(i)}, \boldsymbol{y}_u^{(i)}),\ \rho_{22}^{(i)} \leftarrow D_2(\boldsymbol{x}_u^{(i)}, \tilde{\boldsymbol{y}}_u^{(i)}), \quad i = 1, \ldots, M$
  $\mathcal{L}_{d_1} \leftarrow -\frac{1}{M}\sum_{i=1}^{M} \log \rho_{11}^{(i)} - \frac{1}{M}\sum_{j=1}^{M} \log(1 - \rho_{12}^{(j)}) - \frac{1}{M}\sum_{k=1}^{M} \log(1 - \rho_{13}^{(k)})$
  $\mathcal{L}_{d_2} \leftarrow -\frac{1}{M}\sum_{i=1}^{M} \log \rho_{21}^{(i)} - \frac{1}{M}\sum_{j=1}^{M} \log(1 - \rho_{22}^{(j)})$ ▷ Compute discriminator loss
  $\mathcal{L}_{g_1} \leftarrow -\frac{1}{M}\sum_{i=1}^{M} \log \rho_{12}^{(i)} - \frac{1}{M}\sum_{j=1}^{M} \log(1 - \rho_{21}^{(j)})$ ▷ Compute generator loss
  $\mathcal{L}_{g_2} \leftarrow -\frac{1}{M}\sum_{i=1}^{M} \log \rho_{13}^{(i)} - \frac{1}{M}\sum_{j=1}^{M} \log \rho_{22}^{(j)}$
  $\boldsymbol{\theta}_d \leftarrow \boldsymbol{\theta}_d - \nabla_{\boldsymbol{\theta}_d}(\mathcal{L}_{d_1} + \mathcal{L}_{d_2})$ ▷ Gradient update on discriminator networks
  $\boldsymbol{\theta}_g \leftarrow \boldsymbol{\theta}_g - \nabla_{\boldsymbol{\theta}_g}(\mathcal{L}_{g_1} + \mathcal{L}_{g_2})$ ▷ Gradient update on generator networks
until convergence

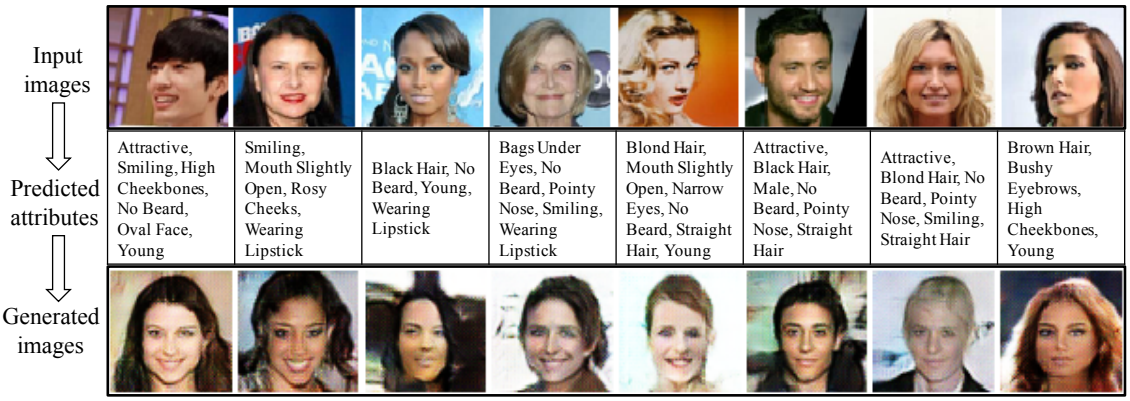


FIGURE 6.8: Additional results on the face-to-attribute-to-face experiment.

6.6.2 Δ-GAN training procedure

6.6.3 Additional experimental results

6.6.4 Evaluation metrics for multi-label classification

Precision@k Precision at k is a popular evaluation metric for multi-label classification problems. Given the ground truth label vector $\boldsymbol{y} \in \{0, 1\}^L$ and the prediction

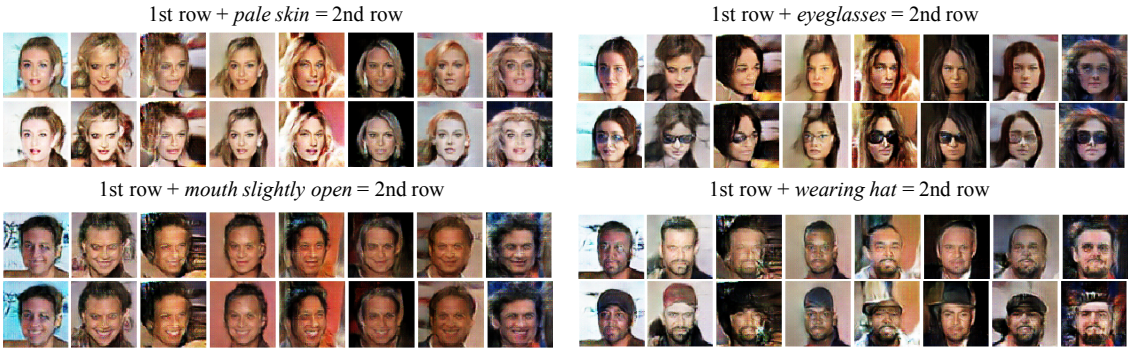


FIGURE 6.9: Additional results on the image editing experiment.

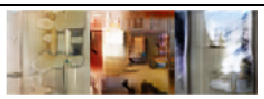
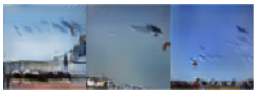
Input	Predicted attributes	Generated images	Input	Predicted attributes	Generated images
	Building, standing, tall, castle, city, top, object, outdoor, tower			airport, airplane, cloudy, large, tarmac, parked, jet, commercial, white, gear, plane, field, flying, landing, aircraft, runway, air, transport	
	furniture, sitting, small, room, living, white, hotel, indoor, table, photo, rug, decorated, window, cabinet			mammal, standing, animal, field, walking, outdoor, grass	
	kite, people, young, blue, boy, standing, playing, colorful, child, air, outdoor, holding, girl, flying			large, red, street, parking, standing, next, decker, tall, train, parked, city, outdoor, transport, tour, road	

FIGURE 6.10: Additional results on the image-to-attribute-to-image experiment.

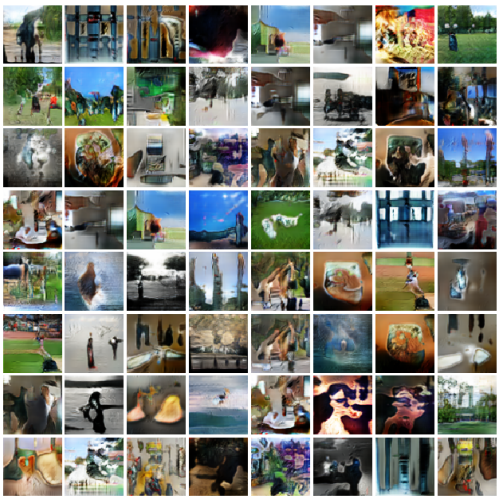


FIGURE 6.11: Attribute-conditional image generation on the COCO dataset. Input attributes are omited for brevity.

$\hat{\boldsymbol{y}} \in [0,1]^L$, $P@k$ is defined as

$$P@k := \frac{1}{k} \sum_{l \in \mathrm{rank}_k(\hat{\boldsymbol{y}})} y^{(l)} . \tag{6.17}$$

Precision at k performs evaluation that counts the fraction of correct predictions in the top k scoring labels.

nDCG@k normalized Discounted Cumulative Gain (nDCG) at rank k is a family of ranking measures widely used in multi-label learning. DCG is the total gain accumulated at a particular rank p, which is defined as

$$DCG@k := \sum_{l \in \mathrm{rank}_k(\hat{\boldsymbol{y}})} \frac{y^{(l)}}{\log(l+1)} . \tag{6.18}$$

Then normalizing DCG by the value at rank k of the ideal ranking gives

$$N@k := \frac{DCG@k}{\sum_{l=1}^{\min(k, \|\boldsymbol{y}\|_0)} \frac{1}{\log(l+1)}} . \tag{6.19}$$

6.6.5 Detailed network architectures

For the CIFAR10 dataset, we use the same network architecture as used in Triple GAN (Li et al., 2017b). For the edges2shoes dataset, we use the same network architecture as used in the pix2pix paper (Isola et al., 2017). For other datasets, we provide the detailed network architectures below.

Table 6.4: Architecture of the models for Δ-GAN on MNIST. BN denotes batch normalization.

Generator A to B	Generator B to A	Discriminator
Input 28 × 28 Gray Image	Input 28 × 28 Gray Image	Input two 28 × 28 Gray Image
5 × 5 conv. 32 ReLU, stride 2, BN	5 × 5 conv. 32 ReLU, stride 2, BN	5 × 5 conv. 32 ReLU, stride 2, BN
5 × 5 conv. 64 ReLU, stride 2, BN	5 × 5 conv. 64 ReLU, stride 2, BN	5 × 5 conv. 64 ReLU, stride 2, BN
5 × 5 conv. 128 ReLU, stride 2, BN	5 × 5 conv. 128 ReLU, stride 2, BN	5 × 5 conv. 128 ReLU, stride 2, BN
Dropout: 0.1	Dropout: 0.1	Dropout: 0.1
MLP output 28 × 28, sigmoid	MLP output 28 × 28, sigmoid	MLP output 1, sigmoid

Table 6.5: Architecture of the models for Δ-GAN on CelebA. BN denotes batch normalization. lReLU denotes Leaky ReLU.

Generator A to B	Generator B to A	Discriminator
Input $64 \times 64 \times 3$ Image	Input 1×40 attributes, 1×100 noise	Input 64×64 Image and 1×40 attributes
4×4 conv. 32 lReLU, stride 2, BN	concat input	concat two inputs
4×4 conv. 64 lReLU, stride 2, BN	MLP output 1024, lReLU, BN	5×5 conv. 64 ReLU, stride 2, BN
4×4 conv. 128 lReLU, stride 2, BN	MP output 8192, lReLU, BN	5×5 conv. 128 ReLU, stride 2, BN
4×4 conv. 256 lReLU, stride 2, BN	concat attributes	
4×4 conv. 512 lReLU, stride 2, BN	5×5 deconv. 256 ReLU, stride 2, BN	5×5 conv. 256 ReLU, stride 2, B
MLP output 512, lReLU	5×5 deconv. 128 ReLU, stride 2, BN	5×5 conv. 512 ReLU, stride 2, BN
MLP output 40, sigmoid	5×5 deconv. 64 ReLU, stride 2, BN	
	5×5 deconv. 3 tanh, stride 2, BN	MLP output 1, sigmoid

Table 6.6: Architecture of the models for Δ-GAN on COCO. BN denotes batch normalization. lReLU denotes Leaky ReLU.

Generator A to B	Generator B to A	Discriminator
Input $64 \times 64 \times 3$ Image	Input 1×40 attributes, 1×100 noise	Input 64×64 Image and $1 \times$ *Dim* attributes
4×4 conv. 32 lReLU, stride 2, BN	concat inputs	concat conditional inputs
4×4 conv. 64 lReLU, stride 2, BN	MLP output 16384, BN	
4×4 conv. 128 lReLU, stride 2, BN	ResNet Block	5×5 conv. 64 ReLU, stride 2, BN
4×4 conv. 256 lReLU, stride 2, BN	4×4 deconv. 512, stride 2	
4×4 conv. 512 lReLU, stride 2, BN	3×3 conv. 512, stride 1, BN	
	ResNet Block	5×5 conv. 128 ReLU, stride 2, BN
ResNet Block	4×4 deconv. 256, stride 2	
	3×3 conv. 256, stride 1, BN	5×5 conv. 256 ReLU, stride 2, BN
	4×4 deconv. 128 ReLU, stride 2	
1×1 conv. 512 lReLU, stride 1, BN	3×3 conv. 128 ReLU, stride 1, BN	5×5 conv. 512 ReLU, stride 2, BN
	4×4 deconv. *Dim*, stride 2	
4×4 conv. *Dim* sigmoid, stride 4	3×3 conv. *Dim* tanh, stride 1	MLP output 1, sigmoid

7

Conclusion and Future Work

7.1 Summary of Contributions

Learning deep generative models is a fast-moving research field. In this thesis, I mainly discuss the models and applications that I have worked on during my Ph.D. study. Specifically, the contributions of this thesis are summarized as follows:

- In Chapter 2, we present a deep generative model for binary image modeling. The proposed deep model is designed by stacking sigmoid belief networks. By exploring the idea of data augmentation, we develop a fully Bayesian algorithm for efficient learning of layer-wise model parameters, and inference of local latent variables.

- In Chapter 3, we present a deep generative model for topic modeling. The proposed deep model employs a deep sigmoid belief network or restricted Boltzmann machine, the bottom binary layer of which selects topics for use in a Poisson factor analysis model. Scalable inference algorithms are derived by applying Bayesian conditional density filtering and stochastic gradient thermostats.

- In Chapter 4, we present a deep generative model for sequence modeling. The proposed deep model is designed by constructing a hierarchy of temporal sigmod belief networks, defined as a sequential stack of sigmoid belief networks. Scalable learning and inference algorithms are derived by introducing an inference network that yields fast sampling from the variational posterior. Both the generative model and the inference network are trained together by maximizing the variational lower bound.

- In Chapter 5, we present a deep generative model for visual captioning. The proposed model has no latent variable, and can be considered as a new variant of LSTM that provides an efficient way to impose side information into the network. The proposed model uses a mixture-of-experts design, and can be considered as training an ensemble of up to 1000 LSTMs simultaneously. Specifically, semantic concepts (*i.e.*, tags) are detected from the image, and the probability of each tag is used to compose the parameters in an LSTM network. The degree to which each member of the ensemble is used to generate an image caption is tied to the image-dependent probability of the corresponding tag.

- In Chapter 6, we present a deep generative model for joint distribution matching. The proposed model is based on generative adversarial networks (GANs). Specifically, the proposed Triangle GAN model consists of four neural networks, two generators and two discriminators. The generators are designed to learn the two-way conditional distributions between the two domains, while the discriminators implicitly define a ternary discriminative function, which is trained to distinguish real data pairs and two kinds of fake data pairs. The generators and discriminators are trained together using adversarial learning.

Besides the SBN and GAN models described in this theis, during my Ph.D. study, I have also studied variational autoencoders (VAEs) (Pu et al., 2016b, 2017b,a). I

have also investigated stochastic gradient MCMC methods (Chen et al., 2016a; Zhang et al., 2017d), with applications in shape classification (Li et al., 2016b) and language modeling (Gan et al., 2017d). Furthermore, I have also studied deep learning techniques for learning better sentence and paragraph representations (Gan et al., 2017c; Zhang et al., 2017c; Gan et al., 2017b; Zhang et al., 2018b). At the beginning of the Ph.D. study, I have also conducted research in gene expression analysis using discriminative factor models (Gan et al., 2015a).

7.2 Future Directions

Deep generative models, and more generally, deep learning, is an exciting field to study in, with lots of new work coming out every week on arXiv. Below I list a few potential research directions for future work.

- **Combination of VAE and GAN**: VAE and GAN are considered as two distinct paradigms for deep generative modeling. Recently, there are many works that attempt to make formal connections between them in a principled way. This is an interesting and active research problem, providing us new tools to connect variational inference and adversarial learning.

- **RL for text generation**: Using Reinforcement Learning (RL) for text generation, such as image captioning and machine translation, has achieved lots of attention, since the text generation problem can be naturally casted as a sequential decision making process, thus RL can be applied. However, all the exisiting RL models for text generation requires careful initialization of the model using maximum likelihood training. Research can be conducted to further understand this problem and improve the performance via the usage of more advanced RL tools.

- **GAN for text generation**: GAN for image synthesis has achieved tremendous success. However, GAN for text generation is a very challenging problem, due to the discrete nature of text, making it difficult to propagate the gradient from the discriminator back to the generator. Currently, there are mainly two ways to explore GAN for text generation. One of them is to use RL techniques to backpropagate gradients, considering the output of the discriminator as the reward for the policy network (*i.e.*, generator) training. Another way is to use efficient gradient approximators to approximate the non-differentiable sampling operations. However, developing a robust GAN model for long text generation is still an open research problem.

In terms of the applications, below I also list a few potential interesting research directions.

- **Vision-to-text generation**: Image and video captioning has been investigated extensively in the literature. Most existing approaches focus on generating a short one-sentence description for the whole image or video. However, typically in a video there are several events happening, organzied in a sequential order. How to generate a coherent long paragraph to describe the whole video is an open research problem. Further, rather than generate a chunk of text, visual dialog is another interesting research direction, which aims to generate a dialogue to mimic the way how a conversation agent can be interacted with humans.

- **Text-to-vision synthesis**: A lot of progresses have been made in generating an image grounded on an input textual description. However, how to generate coherent high-resolution images which potentially contain multiple objects is still an open research problem. Further, generating a whole video based on the text input is also a very interesting and challenging future research direction.

Bibliography

Acharya, A., Ghosh, J., and Zhou, M. (2015), "Nonparametric Bayesian factor analysis for dynamic count matrices," in *AISTATS*.

Anne Hendricks, L., Venugopalan, S., Rohrbach, M., Mooney, R., Saenko, K., and Darrell, T. (2016), "Deep compositional captioning: Describing novel object categories without paired training data," in *CVPR*.

Arjovsky, M. and Bottou, L. (2017), "Towards principled methods for training generative adversarial networks," in *ICLR*.

Arjovsky, M., Chintala, S., and Bottou, L. (2017), "Wasserstein gan," *arXiv:1701.07875*.

Armagan, A., Clyde, M., and Dunson, D. B. (2011), "Generalized beta mixtures of Gaussians," in *NIPS*.

Bahdanau, D., Cho, K., and Bengio, Y. (2015), "Neural machine translation by jointly learning to align and translate," in *ICLR*.

Ballas, N., Yao, L., Pal, C., and Courville, A. (2016), "Delving Deeper into Convolutional Networks for Learning Video Representations," in *ICLR*.

Banerjee, S. and Lavie, A. (2005), "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments," in *ACL workshop*.

Barber, D. and Sollich, P. (1999), "Gaussian Fields for Approximate Inference in Layered Sigmoid Belief Networks." in *NIPS*.

Bayer, J. and Osendorfer, C. (2014), "Learning Stochastic Recurrent Networks," *arXiv:1411.7610*.

Bengio, Y., Courville, A., and Vincent, P. (2013), "Representation learning: A review and new perspectives," *PAMI*.

Blei, D. M. and Lafferty, J. D. (2007), "A CORRELATED TOPIC MODEL OF SCIENCE," *The Annals of Applied Statistics*.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003), "Latent Dirichlet allocation," *JMLR*.

Blei, D. M., Griffiths, T., Jordan, M. I., and Tenenbaum, J. B. (2004), "Hierarchical topic models and the nested Chinese restaurant process," in *NIPS*.

Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. (2012), "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," in *ICML*.

Byrne, S. and Girolami, M. (2013), "Geodesic Monte Carlo on Embedded Manifolds," *Scandinavian J. Statist.*

Carlson, D., Hsieh, Y.-P., Collins, E., Carin, L., and Cevher, V. (2016), "Stochastic spectral descent for discrete graphical models," *IEEE Journal of Selected Topics in Signal Processing*.

Chen, C., Ding, N., and Carin, L. (2015a), "On the convergence of stochastic gradient MCMC algorithms with high-order integrators," in *NIPS*.

Chen, C., Carlson, D., Gan, Z., Li, C., and Carin, L. (2016a), "Bridging the gap between stochastic gradient MCMC and stochastic optimization," in *AISTATS*.

Chen, C., Ding, N., Li, C., Zhang, Y., and Carin, L. (2016b), "Stochastic gradient MCMC with stale gradients," in *NIPS*.

Chen, C., Li, C., Chen, L., Wang, W., Pu, Y., and Carin, L. (2017a), "Continuous-Time Flows for Deep Generative Models," *arXiv preprint arXiv:1709.01179*.

Chen, C., Wang, W., Zhang, Y., Su, Q., and Carin, L. (2017b), "A convergence analysis for a class of practical variance-reduction stochastic gradient mcmc," *arXiv preprint arXiv:1709.01180*.

Chen, D. L. and Dolan, W. B. (2011), "Collecting highly parallel data for paraphrase evaluation," in *ACL*.

Chen, J., Zhu, J., Wang, Z., Zheng, X., and Zhang, B. (2013), "Scalable Inference for Logistic-Normal Topic Models," in *NIPS*.

Chen, L., Dai, S., Pu, Y., Li, C., Su, Q., and Carin, L. (2018), "Symmetric Variational Autoencoder and Connections to Adversarial Learning," in *AISTATS*.

Chen, N., Zhu, J., Xia, F., and Zhang, B. (2014a), "Discriminative Relational Topic Models," *PAMI*.

Chen, T., Fox, E., and Guestrin, C. (2014b), "Stochastic Gradient Hamiltonian Monte Carlo," in *ICML*.

Chen, X. and Lawrence Zitnick, C. (2015), "Mind's eye: A recurrent visual representation for image caption generation," in *CVPR*.

Chen, X., Fang, H., Lin, T.-Y., Vedantam, R., Gupta, S., Dollár, P., and Zitnick, C. L. (2015b), "Microsoft COCO captions: Data collection and evaluation server," *arXiv:1504.00325*.

Cho, K., Raiko, T., and Ilin, A. (2013), "Enhanced gradient for training restricted Boltzmann machines," *Neural computation*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014), "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *EMNLP*.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014), "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *arXiv:1412.3555*.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A., and Bengio, Y. (2015), "A Recurrent Latent Variable Model for Sequential Data," in *NIPS*.

Cong, Y., Chen, B., Liu, H., and Zhou, M. (2017), "Deep Latent Dirichlet Allocation with Topic-Layer-Adaptive Stochastic Gradient Riemannian MCMC," *arXiv preprint arXiv:1706.01724*.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2016), "Language modeling with gated convolutional networks," *arXiv preprint arXiv:1612.08083*.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995), "The Helmholtz machine," *Neural computation*.

Denton, E., Chintala, S., Szlam, A., and Fergus, R. (2015), "Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks," in *NIPS*.

Devlin, J., Cheng, H., Fang, H., Gupta, S., Deng, L., He, X., Zweig, G., and Mitchell, M. (2015), "Language models for image captioning: The quirks and what works," in *ACL*.

Ding, N., Fang, Y., Babbush, R., Chen, C., Skeel, R. D., and Neven, H. (2014), "Bayesian Sampling Using Stochastic Gradient Thermostats," in *NIPS*.

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

Donahue, J., Krähenbühl, P., and Darrell, T. (2017), "Adversarial feature learning," in *ICLR*.

Dong, J., Li, X., Lan, W., Huo, Y., and Snoek, C. G. (2016), "Early Embedding and Late Reranking for Video Captioning," in *ACMMM*.

Dumoulin, V., Belghazi, I., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O., and Courville, A. (2017), "Adversarially learned inference," in *ICLR*.

Elman, J. L. (1990), "Finding structure in time," *Cognitive science*.

Fabius, O., van Amersfoort, J. R., and Kingma, D. P. (2014), "Variational Recurrent Auto-Encoders," *arXiv:1412.6581*.

Fan, K., Wang, Z., Beck, J., Kwok, J., and Heller, K. (2015), "Fast Second-Order Stochastic Backpropagation for Variational Inference," in *NIPS*.

Fan, K., Zhang, Y., Henao, R., and Heller, K. (2016a), "Triply Stochastic Variational Inference for Non-linear Beta Process Factor Analysis," in *ICDM*.

Fan, K., Li, C., and Heller, K. (2016b), "A Unifying Variational Inference Framework for Hierarchical Graph-Coupled HMM with an Application to Influenza Infection," in *AAAI*.

Fang, H., Gupta, S., Iandola, F., Srivastava, R. K., Deng, L., Dollár, P., Gao, J., He, X., Mitchell, M., Platt, J. C., et al. (2015), "From captions to visual concepts and back," in *CVPR*.

Frey, B. J. (1998), *Graphical models for machine learning and digital communication*, MIT press.

Frey, B. J. and Hinton, G. E. (1999), "Variational learning in nonlinear Gaussian belief networks," *Neural Computation*.

Gan, C., Yang, T., and Gong, B. (2016a), "Learning attributes equals multi-source domain generalization," in *CVPR*.

Gan, C., Gan, Z., He, X., Gao, J., and Deng, L. (2017a), "StyleNet: Generating Attractive Visual Captions with Styles," in *CVPR*.

Gan, Z., Yuan, X., Henao, R., Tsalik, E. L., and Carin, L. (2015a), "13 Inference of gene networks associated with the host response to infectious disease," *Big Data over Networks*, p. 365.

Gan, Z., Li, C., Henao, R., Carlson, D. E., and Carin, L. (2015b), "Deep temporal sigmoid belief networks for sequence modeling," in *NIPS*.

Gan, Z., Henao, R., Carlson, D., and Carin, L. (2015c), "Learning Deep Sigmoid Belief Networks with Data Augmentation," in *AISTATS*.

Gan, Z., Chen, C., Henao, R., Carlson, D., and Carin, L. (2015d), "Scalable deep Poisson factor analysis for topic modeling," in *ICML*.

Gan, Z., Gan, C., He, X., Pu, Y., Tran, K., Gao, J., Carin, L., and Deng, L. (2016b), "Semantic compositional networks for visual captioning," *arXiv preprint arXiv:1611.08002*.

Gan, Z., Singh, P., Joshi, A., He, X., Chen, J., Gao, J., and Deng, L. (2017b), "Character-level deep conflation for business data analytics," in *ICASSP*.

Gan, Z., Pu, Y., Henao, R., Li, C., He, X., and Carin, L. (2017c), "Learning generic sentence representations using convolutional neural networks," in *EMNLP*.

Gan, Z., Li, C., Chen, C., Pu, Y., Su, Q., and Carin, L. (2017d), "Scalable bayesian learning of recurrent neural networks for language modeling," in *ACL*.

Gan, Z., Gan, C., He, X., Pu, Y., Tran, K., Gao, J., Carin, L., and Deng, L. (2017e), "Semantic Compositional Networks for Visual Captioning," in *CVPR*.

Gan, Z., Chen, L., Wang, W., Pu, Y., Zhang, Y., Liu, H., Li, C., and Carin, L. (2017f), "Triangle generative adversarial networks," in *NIPS*.

Ghahramani, Z. and Hinton, G. E. (1997), "The EM algorithm for mixtures of factor analyzers," *Technical report, University of Toronto*.

Girolami, M. and Calderhead, B. (2011), "Riemann manifold Langevin and Hamiltonian Monte Carlo methods," *J. R. Statist. Soc. B*.

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

Graves, A. (2013), "Generating sequences with recurrent neural networks," *arXiv:1308.0850*.

Gregor, K., Mnih, A., and Wierstra, D. (2014), "Deep AutoRegressive Networks," in *ICML*.

Guhaniyogi, R., Qamar, S., and Dunson, D. B. (2014), "Bayesian Conditional Density Filtering," *arXiv:1401.3632*.

Han, S., Du, L., Salazar, E., and Carin, L. (2014), "Dynamic Rank Factor Model for Text Streams," in *NIPS*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016), "Deep residual learning for image recognition," in *CVPR*.

Henao, R., Gan, Z., Lu, J., and Carin, L. (2015), "Deep Poisson factor modeling," in *NIPS*.

Henao, R., Lu, J. T., Lucas, J. E., Ferranti, J., and Carin, L. (2016), "Electronic health record analysis via deep poisson factor models," *JMLR*.

Henderson, J. and Titov, I. (2010), "Incremental sigmoid belief networks for grammar learning," *JMLR*.

Hermans, M. and Schrauwen, B. (2013), "Training and analysing deep recurrent neural networks," in *NIPS*.

Hinton, G., Dayan, P., To, A., and Neal, R. (1995a), "The Helmholtz machine through time," in *Proc. of the ICANN*.

Hinton, G., Osindero, S., and Teh, Y.-W. (2006), "A fast learning algorithm for deep belief nets," *Neural computation*.

Hinton, G. E. (2002), "Training products of experts by minimizing contrastive divergence," *Neural computation*.

Hinton, G. E. and Salakhutdinov, R. (2011), "Discovering binary codes for documents by learning deep generative models," *Topics in Cognitive Science*.

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995b), "The "wake-sleep" algorithm for unsupervised neural networks," *Science*.

Ho, Q., Cipar, J., Cui, H., Kim, J. K., Lee, S., Gibbons, P. B., Gibbons, G. A., Ganger, G. R., and Xing, E. P. (2013), "More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server," in *NIPS*.

Hochreiter, S. and Schmidhuber, J. (1997), "Long short-term memory," in *Neural computation*.

Hoffman, M. D., Blei, D. M., and Bach, F. (2010), "Online learning for latent Dirichlet allocation," in *NIPS*.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013a), "Stochastic variational inference," *JMLR*.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013b), "Stochastic Variational Inference," *JMLR*.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017), "Image-to-image translation with conditional adversarial networks," in *CVPR*.

Jia, X., Gavves, E., Fernando, B., and Tuytelaars, T. (2015), "Guiding long-short term memory for image caption generation," in *ICCV*.

Jin, J., Fu, K., Cui, R., Sha, F., and Zhang, C. (2015), "Aligning where to see and what to tell: image caption with region-based attention and scene factorization," *arXiv:1506.06272*.

Kalman, R. (1963), "Mathematical description of linear dynamical systems," *J. the Society for Industrial & Applied Mathematics, Series A: Control*.

Karpathy, A. and Fei-Fei, L. (2015), "Deep visual-semantic alignments for generating image descriptions," in *CVPR*.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014), "Large-scale video classification with convolutional neural networks," in *CVPR*.

Kim, T., Cha, M., Kim, H., Lee, J., and Kim, J. (2017), "Learning to Discover Cross-Domain Relations with Generative Adversarial Networks," in *ICML*.

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

Kingma, D. P. and Welling, M. (2013), "Auto-encoding variational Bayes," *arXiv:1312.6114*.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016), "Improved variational inference with inverse autoregressive flow," in *NIPS*.

Kiros, R., Salakhutdinov, R., and Zemel, R. S. (2014a), "Multimodal Neural Language Models." in *ICML*.

Kiros, R., Zemel, R., and Salakhutdinov, R. R. (2014b), "A multiplicative model for learning distributed text-based attribute representations," in *NIPS*.

Kiros, R., Salakhutdinov, R., and Zemel, R. S. (2014c), "Unifying visual-semantic embeddings with multimodal neural language models," *arXiv:1411.2539*.

Krizhevsky, A. (2009), "Learning multiple layers of features from tiny images," *Citeseer*.

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

Larochelle, H. and Lauly, S. (2012), "A neural autoregressive topic model," in *NIPS*.

Larochelle, H. and Murray, I. (2011), "The neural autoregressive distribution estimator," *JMLR*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998), "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. (2017), "Photo-realistic single image super-resolution using a generative adversarial network," in *CVPR*.

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

Li, C., Chen, C., Fan, K., and Carin, L. (2016a), "High-Order Stochastic Gradient Thermostats for Bayesian Learning of Deep Models." in *AAAI*.

Li, C., Stevens, A., Chen, C., Pu, Y., Gan, Z., and Carin, L. (2016b), "Learning weight uncertainty with stochastic gradient mcmc for shape classification," in *CVPR*.

Li, C., Chen, C., Carlson, D., and Carin, L. (2016c), "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks," in *AAAI*.

Li, C., Liu, H., Chen, C., Pu, Y., Chen, L., Henao, R., and Carin, L. (2017a), "ALICE: Towards Understanding Adversarial Learning for Joint Distribution Matching," in *NIPS*.

Li, C., Xu, K., Zhu, J., and Zhang, B. (2017b), "Triple Generative Adversarial Nets," in *NIPS*.

Li, C., Li, J., Wang, G., and Carin, L. (2018a), "Learning to Sample with Adversarially Learned Likelihood-Ratio," in *ICLR workshop*.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. (2018b), "Measuring the Intrinsic Dimension of Objective Landscapes," in *ICLR*.

Li, M., Andersen, D., Smola, A., and Yu, K. (2014), "Communication Efficient Distributed Machine Learning with the Parameter Server," in *NIPS*.

Lin, C.-Y. (2004), "Rouge: A package for automatic evaluation of summaries," in *ACL workshop*.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014), "Microsoft coco: Common objects in context," in *ECCV*.

Liu, C., Mao, J., Sha, F., and Yuille, A. (2016), "Attention Correctness in Neural Image Captioning," *arXiv:1605.09553*.

Liu, M.-Y. and Tuzel, O. (2016), "Coupled generative adversarial networks," in *NIPS*.

Liu, M.-Y., Breuel, T., and Kautz, J. (2017), "Unsupervised Image-to-Image Translation Networks," in *NIPS*.

Liu, Q. and Wang, D. (2016), "Stein variational gradient descent: A general purpose bayesian inference algorithm," in *NIPS*.

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

Maaloe, L., Arngren, M., and Winther, O. (2015), "Deep Belief Nets for Topic Modeling," *arXiv:1501.04325*.

Mao, J., Xu, W., Yang, Y., Wang, J., Huang, Z., and Yuille, A. (2015), "Deep captioning with multimodal recurrent neural networks (m-rnn)," in *ICLR*.

Marlin, B. M., Swersky, K., Chen, B., and Freitas, N. D. (2010), "Inductive principles for restricted Boltzmann machine learning," in *AISTATS*.

Martens, J. and Sutskever, I. (2011), "Learning recurrent neural networks with hessian-free optimization," in *ICML*.

Memisevic, R. and Hinton, G. (2007), "Unsupervised learning of image transformations," in *CVPR*.

Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. (2017), "Unrolled Generative Adversarial Networks," in *ICLR*.

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

Mimno, D., Hoffman, M. D., and Blei, D. M. (2012), "Sparse stochastic inference for latent Dirichlet allocation," in *ICML*.

Mirza, M. and Osindero, S. (2014), "Conditional generative adversarial nets," *arXiv:1411.1784*.

Mittelman, R., Kuipers, B., Savarese, S., and Lee, H. (2014), "Structured Recurrent Temporal Restricted Boltzmann Machines," in *ICML*.

Mnih, A. and Gregor, K. (2014), "Neural variational inference and learning in belief networks," in *ICML*.

Murray, I. and Salakhutdinov, R. (2009), "Evaluating probabilities under high-dimensional latent variable models," in *NIPS*.

Neal, R. M. (1992), "Connectionist learning of belief networks," *Artificial intelligence*.

Oord, A. v. d., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016a), "Conditional Image Generation with PixelCNN Decoders," *arXiv preprint arXiv:1606.05328*.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016b), "Pixel recurrent neural networks," *arXiv preprint arXiv:1601.06759*.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016c), "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*.

Paisley, J., Wang, C., Blei, D. M., and Jordan, M. I. (2015), "Nested hierarchical Dirichlet processes," *PAMI*.

Pan, Y., Mei, T., Yao, T., Li, H., and Rui, Y. (2016), "Jointly Modeling Embedding and Translation to Bridge Video and Language," in *CVPR*.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002), "BLEU: a method for automatic evaluation of machine translation," in *ACL*.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013), "On the difficulty of training recurrent neural networks," in *ICML*.

Patterson, S. and Teh, Y. W. (2013), "Stochastic Gradient Riemannian Langevin Dynamics on the Probability Simplex," in *NIPS*.

Perarnau, G., van de Weijer, J., Raducanu, B., and Álvarez, J. M. (2016), "Invertible Conditional GANs for image editing," *arXiv:1611.06355*.

Polson, N. G. and Scott, J. G. (2012), "Local shrinkage rules, Lévy processes and regularized regression," *JRSS*.

Polson, N. G., Scott, S. L., et al. (2011), "Data augmentation for support vector machines," *Bayesian Analysis*.

Polson, N. G., Scott, J. G., and Windle, J. (2013a), "Bayesian Inference for Logistic Models Using Pólya–Gamma Latent Variables," *JASA*.

Polson, N. G., Scott, J. G., and Windle, J. (2013b), "Bayesian inference for logistic models using Pólya-Gamma latent variables," *JASA*.

Pu, Y., Yuan, X., and Carin, L. (2015), "Generative Deep Deconvolutional Learning," in *ICLR workshop*.

Pu, Y., Yuan, X., Stevens, A., Li, C., and Carin, L. (2016a), "A Deep Generative Deconvolutional Image Model," in *AISTATS*.

Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., and Carin, L. (2016b), "Variational autoencoder for deep learning of images, labels and captions," in *NIPS*.

Pu, Y., Wang, W., Henao, R., Chen, L., Gan, Z., Li, C., and Carin, L. (2017a), "Adversarial Symmetric Variational Autoencoder," in *NIPS*.

Pu, Y., Gan, Z., Henao, R., Li, C., Han, S., and Carin, L. (2017b), "VAE Learning via Stein Variational Gradient Descent," in *NIPS*.

Pu, Y., Min, M. R., Gan, Z., and Carin, L. (2018), "Adaptive feature abstraction for translating video to text," in *AAAI*.

Rabiner, L. and Juang, B. (1986), "An introduction to hidden Markov models," *ASSP Magazine, IEEE*.

Radford, A., Metz, L., and Chintala, S. (2016), "Unsupervised representation learning with deep convolutional generative adversarial networks," in *ICLR*.

Ranganath, R., Tang, L., Charlin, L., and Blei, D. M. (2015), "Deep Exponential Families," in *AISTATS*.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. (2016), "Generative adversarial text to image synthesis," in *ICML*.

Rezende, D. J. and Mohamed, S. (2015), "Variational inference with normalizing flows," *arXiv preprint arXiv:1505.05770*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014), "Stochastic backpropagation and approximate inference in deep generative models," in *ICML*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015), "Imagenet large scale visual recognition challenge," *IJCV*.

Salakhutdinov, R. (2015), "Learning deep generative models," *Annual Review of Statistics and Its Application*.

Salakhutdinov, R. and Hinton, G. E. (2009a), "Deep Boltzmann machines," in *AISTATS*.

Salakhutdinov, R. and Hinton, G. E. (2009b), "Replicated softmax: an undirected topic model," in *NIPS*.

Salakhutdinov, R. and Larochelle, H. (2010), "Efficient learning of deep Boltzmann machines," in *AISTATS*.

Salakhutdinov, R. and Murray, I. (2008), "On the quantitative analysis of deep belief networks," in *ICML*.

Salakhutdinov, R., Tenenbaum, J. B., and Torralba, A. (2013), "Learning with hierarchical-deep models," *PAMI*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016), "Improved techniques for training gans," in *NIPS*.

Saul, L. K., Jaakkola, T., and Jordan, M. I. (1996), "Mean field theory for sigmoid belief networks," *JAIR*.

Shen, D., Zhang, Y., Henao, R., Su, Q., and Carin, L. (2017), "Deconvolutional latent-variable model for text sequence matching," *arXiv preprint arXiv:1709.07109*.

Simonyan, K. and Zisserman, A. (2015), "Very deep convolutional networks for large-scale image recognition," in *ICLR*.

Smolensky, P. (1986), "Information processing in dynamical systems: Foundations of harmony theory," *Parallel Distributed Processing, chapter 6*.

Socher, R., Karpathy, A., Le, Q. V., Manning, C. D., and Ng, A. Y. (2014), "Grounded compositional semantics for finding and describing images with sentences," *TACL*.

Song, J., Gan, Z., and Carin, L. (2016a), "Factored Temporal Sigmoid Belief Networks for Sequence Learning," in *ICML*.

Song, Z., Henao, R., Carlson, D., and Carin, L. (2016b), "Learning sigmoid belief networks via Monte Carlo expectation maximization," in *AISTATS*.

Song, Z., Muraoka, Y., Fujimaki, R., and Carin, L. (2017), "Scalable model selection for belief networks," in *NIPS*.

Springenberg, J. T. (2015), "Unsupervised and semi-supervised learning with categorical generative adversarial networks," *arXiv:1511.06390*.

Srivastava, N., Salakhutdinov, R., and Hinton, G. E. (2013), "Modeling documents with deep Boltzmann machines," in *UAI*.

Stevens, A., Pu, Y., Sun, Y., Spell, G., and Carin, L. (2017), "Tensor-Dictionary Learning with Deep Kruskal-Factor Analysis," in *AISTATS*.

Su, Q., Liao, X., Li, C., Gan, Z., and Carin, L. (2017), "Unsupervised Learning with Truncated Gaussian Graphical Models." in *AAAI*.

Sutskever, I. and Hinton, G. (2007), "Learning multilevel distributed representations for high-dimensional sequences," in *AISTATS*.

Sutskever, I., Hinton, G., and Taylor, G. (2009), "The recurrent temporal restricted boltzmann machine," in *NIPS*.

Sutskever, I., Martens, J., and Hinton, G. E. (2011), "Generating text with recurrent neural networks," in *ICML*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014), "Sequence to sequence learning with neural networks," in *NIPS*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015), "Going deeper with convolutions," in *CVPR*.

Taigman, Y., Polyak, A., and Wolf, L. (2017), "Unsupervised Cross-Domain Image Generation," in *ICLR*.

Taylor, G. and Hinton, G. (2009), "Factored conditional restricted Boltzmann machines for modeling motion style," in *ICML*.

Taylor, G., Hinton, G., and Roweis, S. (2006), "Modeling human motion using binary latent variables," in *NIPS*.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2006), "Hierarchical Dirichlet Processes," *JASA*.

Theano Development Team (2016), "Theano: A Python framework for fast computation of mathematical expressions," *arXiv: 1605.02688*.

Tieleman, T. and Hinton, G. (2012), "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," in *COURSERA: Neural Networks for Machine Learning*.

Tipping, M. E. and Bishop, C. M. (1999), "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*.

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015), "Learning spatiotemporal features with 3d convolutional networks," in *ICCV*.

Tran, K., He, X., Zhang, L., Sun, J., Carapcea, C., Thrasher, C., Buehler, C., and Sienkiewicz, C. (2016), "Rich image captioning in the wild," in *CVPR Workshops*.

Tuckerman, M. E. (2010), *Statistical Mechanics: Theory and Molecular Simulation*, Oxford University Press.

Vedantam, R., Lawrence Zitnick, C., and Parikh, D. (2015), "Cider: Consensus-based image description evaluation," in *CVPR*.

Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., and Saenko, K. (2015a), "Sequence to sequence-video to text," in *ICCV*.

Venugopalan, S., Xu, H., Donahue, J., Rohrbach, M., Mooney, R., and Saenko, K. (2015b), "Translating videos to natural language using deep recurrent neural networks," in *NAACL*.

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

Wallach, H. M., Murray, I., Salakhutdinov, R., and Mimno, D. (2009), "Evaluation methods for topic models," in *ICML*.

Wang, C. and Blei, D. M. (2012), "Truncation-free Stochastic Variational Inference for Bayesian Nonparametric Models," in *NIPS*.

Wang, W., Gan, Z., Wang, W., Shen, D., Huang, J., Ping, W., Satheesh, S., and Carin, L. (2017), "Topic Compositional Neural Language Model," *arXiv preprint arXiv:1712.09783*.

Wang, W., Pu, Y., Verma, V. K., Fan, K., Zhang, Y., Chen, C., Rai, P., and Carin, L. (2018), "Zero-Shot Learning via Class-Conditioned Deep Generative Models," in *AAAI*.

Welling, M. and Teh, Y. W. (2011), "Bayesian Learning via Stochastic Gradient Langevin Dynamics," in *ICML*.

Welling, M., Rosen-Zvi, M., and Hinton, G. E. (2005), "Exponential family harmoniums with an application to information retrieval," in *NIPS*.

Werbos, P. (1990), "Backpropagation through time: what it does and how to do it," in *Proc. of the IEEE*.

Williamson, S., Wang, C., Heller, K., and Blei, D. M. (2010), "The IBP compound Dirichlet process and its application to focused topic modeling," in *ICML*.

Wu, Q., Shen, C., Liu, L., Dick, A., and Hengel, A. v. d. (2016a), "What value do explicit high level concepts have in vision to language problems?" in *CVPR*.

Wu, Y., Zhang, S., Zhang, Y., Bengio, Y., and Salakhutdinov, R. (2016b), "On Multiplicative Integration with Recurrent Neural Networks," in *NIPS*.

Xia, Y., Qin, T., Chen, W., Bian, J., Yu, N., and Liu, T.-Y. (2017), "Dual Supervised Learning," in *ICML*.

Xian, Y., Pu, Y., Gan, Z., Lu, L., and Thompson, A. (2016), "Modified DCTNet for audio signals classification," *The Journal of the Acoustical Society of America*.

Xian, Y., Pu, Y., Gan, Z., Lu, L., and Thompson, A. (2017), "Adaptive DCTNet for audio signal classification," in *ICASSP*.

Xu, J., Mei, T., Yao, T., and Rui, Y. (2016), "Msr-vtt: A large video description dataset for bridging video and language," in *CVPR*.

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

Xu, T., Zhang, P., Huang, Q., Zhang, H., Gan, Z., Huang, X., and He, X. (2017), "AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks," *arXiv preprint arXiv:1711.10485*.

Yang, Z., Yuan, Y., Wu, Y., Salakhutdinov, R., and Cohen, W. W. (2016), "Review Networks for Caption Generation," in *NIPS*.

Yi, Z., Zhang, H., Tan, P., and Gong, M. (2017), "DualGAN: Unsupervised Dual Learning for Image-to-Image Translation," in *ICCV*.

You, Q., Jin, H., Wang, Z., Fang, C., and Luo, J. (2016), "Image captioning with semantic attention," in *CVPR*.

Young, P., Lai, A., Hodosh, M., and Hockenmaier, J. (2014), "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions," *TACL*.

Yu, H., Wang, J., Huang, Z., Yang, Y., and Xu, W. (2016), "Video Paragraph Captioning using Hierarchical Recurrent Neural Networks," in *CVPR*.

Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017), "Seqgan: sequence generative adversarial nets with policy gradient," in *AAAI*.

Yuan, X. and Pu, Y. (2018), "Parallel lensless compressive imaging via deep convolutional neural networks," *Optics Express*.

Yuan, X., Pu, Y., and Carin, L. (2017), "Compressive Sensing via Convolutional Factor Analysis," *arXiv preprint arXiv:1701.03006*.

Yuille, A. (2005), "The Convergence of Contrastive Divergences," in *NIPS*.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014), "Recurrent neural network regularization," *arXiv:1409.2329*.

Zhang, H., Xu, T., Li, H., Zhang, S., Huang, X., Wang, X., and Metaxas, D. (2017a), "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks," in *ICCV*.

Zhang, R., Li, C., Chen, C., and Carin, L. (2018a), "Learning Structural Weight Uncertainty for Sequential Decision-Making," in *AISTATS*.

Zhang, X. and Carin, L. (2012), "Joint modeling of a matrix with associated text via latent binary features," in *NIPS*.

Zhang, X., Henao, R., Gan, Z., Li, Y., and Carin, L. (2018b), "Multi-Label Learning from Medical Plain Text with Convolutional Residual Models," *arXiv preprint arXiv:1801.05062*.

Zhang, Y., Henao, R., Li, C., and Carin, L. (2015), "Learning Dictionary with Spatial and Inter-dictionary Dependency," *Workshop on representation learning, NIPS*.

Zhang, Y., Henao, R., Li, C., and Carin, L. (2016a), "Bayesian Dictionary Learning with Gaussian Processes and Sigmoid Belief Networks," in *IJCAI*.

Zhang, Y., Zhao, Y., David, L., Henao, R., and Carin, L. (2016b), "Dynamic Poisson Factor Analysis," in *ICDM*.

Zhang, Y., Gan, Z., and Carin, L. (2016c), "Generating text via adversarial training," in *NIPS workshop on Adversarial Training*.

Zhang, Y., Chen, C., Henao, R., and Carin, L. (2016d), "Laplacian Hamiltonian Monte Carlo," in *ECML PKDD*.

Zhang, Y., Henao, R., Carin, L., Zhong, J., and Hartemink, A. J. (2016e), "Learning a Hybrid Architecture for Sequence Regression and Annotation." in *AAAI*.

Zhang, Y., Wang, X., Chen, C., Henao, R., Fan, K., and Carin, L. (2016f), "Towards unifying Hamiltonian Monte Carlo and slice sampling," in *NIPS*.

Zhang, Y., Gan, Z., Fan, K., Chen, Z., Henao, R., Shen, D., and Carin, L. (2017b), "Adversarial Feature Matching for Text Generation," in *ICML*.

Zhang, Y., Shen, D., Wang, G., Gan, Z., Henao, R., and Carin, L. (2017c), "Deconvolutional paragraph representation learning," in *NIPS*.

Zhang, Y., Chen, C., Gan, Z., Henao, R., and Carin, L. (2017d), "Stochastic Gradient Monomial Gamma Sampler," in *ICML*.

Zhao, J., Mathieu, M., and LeCun, Y. (2017), "Energy-based generative adversarial network," in *ICLR*.

Zhou, M. and Carin, L. (2015), "Negative Binomial process count and mixture modeling," *PAMI*.

Zhou, M., Hannah, L., Dunson, D., and Carin, L. (2012a), "Beta-negative Binomial process and Poisson factor analysis," in *AISTATS*.

Zhou, M., Li, L., Dunson, D., and Carin, L. (2012b), "Lognormal and gamma mixed negative binomial regression," in *ICML*.

Zhou, M., Cong, Y., and Chen, B. (2015), "The Poisson gamma belief network," in *NIPS*.

Zhou, M., Cong, Y., and Chen, B. (2016), "Augmentable gamma belief networks," *JMLR*.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017), "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks," in *ICCV*.

Biography

Zhe Gan was born on December 1st, 1989, and was raised in Fuzhou, Jiangxi Province, China. He received the B.S. degree and M.S. degree in Electrical Engineering, both from Peking University, Beijing, China, in 2010 and 2013, respectively. Beginning August 2013, he has been with the Department of Electrical and Computer Engineering at Duke University, Durham, NC, where he is currently studying toward the Ph.D. degree in Electrical and Computer Engineering, under the guidance of Professor Lawrence Carin. His research interests lie in machine learning, especially deep learning, with its applications in computer vision and natural language processing.