EMPOWERING PROBABILISTIC INFERENCE WITH STOCHASTIC DEEP NEURAL NETWORKS

GUOQING ZHENG

CMU-LTI-18-012

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

THESIS COMMITTEE:

Yiming Yang, Co-Chair	(Carnegie Mellon University)
Jaime Carbonell, Co-Chair	(Carnegie Mellon University)
Pradeep Ravikumar	(Carnegie Mellon University)
John Paisley	(Columbia University)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Language and Information Technologies

Keywords: probabilistic modeling, probabilistic inference, deep neural networks

To Ni and my family.

ACKNOWLEDGMENTS

I would like to thank my PhD advisors, Yiming Yang and Jaime Carbonell, and my Master's advisor, Jamie Callan, for their guidance, support and help during my journey of always challenging and surpassing myself for the last couple years. I will always be grateful for what I learned from you.

I would also like to thank my thesis committee, Pradeep Ravikumar and John Paisley for generously agreeing to serve on my committee and offering valuable discussions and feedbacks.

Thank my fellow group members, past and present: Chenyan Xiong, Shriphani Palakodety, Siddharth Gopal, Hanxiao Liu, Andrew Hsi, Wanli Ma, Adams Wei Yu, Keerthiram Murugesan, Jay-Yoon Lee, Ruochen Xu, Wei-Cheng Chang, Guokun Lai, Yuexin Wu, Jingzhou Liu, Bohan Li, Zihang Dai; and other CMU friends: Hector Liu, Di Wang, Zi Yang, Max Ma, Xiaohua Yan, Tom Vu, Collin McCormack, Qizhe Xie, Zhuyun Dai, Diyi Yang, Zichao Yang, Zhiting Hu and many other friends whose names I haven't listed here. Without all of you, the days at CMU couldn't have been so enjoyable.

To my family - my parents and younger brother, thank you for supporting my decision to pursue graduate studies in the U.S. and always having my back while I am away from home.

Lastly, I would like to thank Ni, without you getting a PhD would not mean as much.

ABSTRACT

Probabilistic models are powerful tools in understanding real world data from various domains, including images, natural language texts, audios and temporal time series. Often more flexible and expressive probabilistic models are preferred for accurate modeling, however the difficulty for effective model learning and inference arises accordingly due to increasingly more complex probabilistic model architectures. Meanwhile, recent advances in deep neural networks for both supervised and unsupervised learning have shown prominent advantages in learning flexible deterministic mappings, compared to traditional shallow models. Integrating deep neural networks into probabilistic modeling thus becomes an important research direction. Though existing works have opened the door of modeling stochasticity in data with deep neural networks, they may still suffer from limitations, such as a) the family of distributions that can be captured for both modeling and inference is limited, b) probabilistic models for some important discrete structures, such as permutations, have not yet been extensively studied; and c) applications to discrete and continuous sequential data modeling, such as natural language and time series, could still use significant improvements.

In this thesis, we propose simple yet effective methods to address the above limitations of incorporating stochastic deep neural networks for probabilistic modeling. Specifically, we propose: a) to enrich the family of distributions used for probabilistic modeling and inference, b) to define probabilistic models over certain important discrete structures and to demonstrate how learning and inference could be performed over them; and c) to develop significantly better probabilistic models in both discrete and continuous sequential data domains, such as natural languages and continuous time series. Experimental results have demonstrated the effectiveness of the proposed approaches.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

In this chapter we firstly introduce and explain background materials of probabilistic modeling with deep neural networks, identify limitations of existing work and raise a set of research questions to explore and answer in this thesis.

1.1 OVERVIEW

Probabilistic modeling are powerful tools in modeling real world data from various domains, such as natural languages, images, time series, which embraces rich flexibility, accurate prediction and meaningful interpretations from the probabilistic models. Probabilistic modeling involves the problems of capturing uncertainty, (or randomness, stochasticity) in the data with statistical tools, inferring the probabilistic distributions of the quantity of interest, which could be either the data itself or latent information underlying the data, and making predictions in a probabilistic manner. Often expressive, yet efficient probabilistic models are preferred for accurately modeling stochasticity in the data, particularly for data domains with rich structures, such as images, natural languages, time series, etc., however the difficulties for model learning and inference arise accordingly due to the more complex architectures of the probabilistic models, high computational and design cost associated with them.

Meanwhile, recent advances with deep neural networks in supervised learning tasks have shown prominent advantages in terms empirical performances over traditional shallow models and has become required components for any successful methods in various tasks, including but not limited to text classification (Glorot et al., 2011b; Joulin et al., 2016; Kim, 2014; Lai et al., 2015; Liu et al., 2017; Yang et al., 2016; Zhang et al., 2015), machine translation (Bahdanau et al., 2014; Cho et al., 2014a,b; Chung et al., 2014; Sutskever et al., 2014; Wu et al., 2016; Zoph et al., 2016; Zou et al., 2013), image recognition and segmentation (Badrinarayanan et al., 2017; Chollet, 2017; Donahue et al., 2014; Donahue et al., 2015; Glorot et al., 2011a; He et al., 2016; Huang et al., 2017; Ioffe and Szegedy, 2015; Noh et al., 2015; Simonyan and Zisserman, 2014; Wu et al., 2015) and speech recognition (Amodei

et al., 2016; Bahdanau et al., 2016; Dahl et al., 2012; Graves et al., 2013a,b; Hannun et al., 2014; Hinton et al., 2012). They also shed light on improving probabilistic modelings. Integrating deep neural networks into probabilistic modeling thus becomes an important research direction.

Due to the complex architectures of deep neural networks and the requirement of large scale data to train them, modeling, reasoning and inference about uncertainty with deep neural networks brings new challenges in probabilistic modeling. The main obstacles for fully empowering probabilistic modeling and inference with deep neural networks results from two fundamental characteristics of stochastic modeling with deep neural networks :

a) Deep neural networks are best known for its ability to learn and approximate arbitrary *deterministic* functions mapping from their input to their output (Cybenko, 1989; Goodfellow et al., 2016; Hornik, 1991), however there is only a handful ways to inject randomness into the network to model uncertainty about the quantity of interest and once uncertainty is introduced to the architecture, it becomes a challenge to do effective reasoning and inference with them;

b) For the cases where uncertainty can indeed be injected and modeled, the family of probabilistic distributions that allows feasible, let alone efficient, learning is still limited, which further hampers its use for general purpose probabilistic modeling and inference.

For example, variational autoencoders (VAE) (Kingma and Welling, 2013) is one representative work that tries to incorporate and model uncertainly in data, which have shown success in probabilistic modeling data from various domains. However, there are limitations in its ability to represent arbitrarily complex probabilistic models. VAE injects randomness in its network architecture, and in order to use stochastic gradient descent (Bottou, 2010) for model training, often over-simplified distributions about the randomness is assumed, such as Gaussians.

Though existing works have opened the door of modeling stochasticity in data with deep neural networks (Kingma and Welling, 2013; Larochelle and Murray, 2011; Maaløe et al., 2016; Miao et al., 2016; Sønderby et al., 2016), they may still suffer from limitations, such as a) the family of distributions that can be captured for both modeling and inference is limited, b) probabilistic models for some important discrete structures, such as permutations, have not yet been extensively studied; and c) applications to discrete and continuous sequential data modeling, such as natural language and time series, could still use significant improvements.

1.2 THESIS STATEMENTS AND CONTRIBUTIONS

In this thesis, we aim to address the above challenges of probabilistic modeling with deep neural networks. Particularly, we emphasize that we are far from fully harnessing the power of deep neural networks to manipulate stochasticity for probabilistic modeling and inference, and we propose to make a step forward in this direction, by asking and answering the following research questions:

> **Research question 1:** (On stochastic neural variational inference) *The original variational autoencoder is a representing work for variational inference with deep neural networks.It relies on the reparameterization trick to construct inference model for variational inference, hence the family of variational posterior it can model is quite limited. Can we make the inference model of VAE more flexible, to accommodate variational families that might not admit reparameterization tricks?*
>
> **Proposed solution:** We propose to cover a much richer variational family q for VAE, via two methods. One is to incorporate auxiliary variables into the VAE framework, and we term the resulting model as Asymmetric Variational Autoencoders (Chapter 2); the other is to propose a new family of neural network layer for density transformation to capture complex posterior families based on efficient 1-d convolutions (Chapter 3).
>
> **Research question 2:** (On neural probabilistic modeling of discrete structures) *Certain discrete structures, including permutations, which are important to many machine learning tasks, haven't been extensively studied in the context of neural generative modeling. Taking permutations as an example, can we model, capture and compute them with (stochastic) deep neural networks?*
>
> **Proposed solution:** To this end, we first propose to model and learn permutations with adversarial training for the unpaired setting; then for the unsupervised setting, we construct probabilistic models over permutations and propose to learn such latent permutations from the data in a fully unsupervised manner (Chapter 4).
>
> **Research question 3:** (On probabilistic modeling of discrete sequences)
>
> *Neural probabilistic models have achieved great success on continuous i.i.d data, such as images. Though there are efforts to model discrete sequences, such as natural language texts, can we build much more effective probabilistic models to capture the dynamics of such data?*
>
> **Proposed solution:** We propose a novel neural language model based on insights of natural language generation. The proposed method is shown to be more effective compared to existing state-of-the-arts (Chapter 5).

Research question 4: (On probabilistic modeling of continuous sequences) *Continuous sequential data has been known to to more difficult to model with neural probabilistic models, due to more complex dynamics and volatility underlying the data. Existing best generative modeling performances are achieved by recurrent latent variable models, however model learning and inference turn to be more challenging and expensive due to the introduced latent variables. Can we build neural probabilistic models retaining the good performance, i.e., better capturing the dynamics in continuous sequential data, without compromising the efficiency in model learning and inference?*

Proposed solution: To encounter complex uncertainty in temporal data domains and retaining efficient model learning and inference, we propose a new probabilistic model by constructing expressive data generative distribution with context aware normalizing flow per step. We identify that the context awareness is key to successful continuous sequences modeling. As no latent variables are introduced, no additional inference network is required thus efficient model learning and inference are retained (Chapter 6).

1.3 THESIS RELATED PUBLICATIONS

Our two solutions to enrich the variational family to improve stochastic neural variational inference in Chapter 2 and 3, i.e., via auxiliary variables and convolutional normalizing flows, were published as two workshop papers at ICML 2018 (Zheng et al., 2017a,b) respectively. The new neural probabilistic language model in Chapter 5 is under submission to ICLR 2019. Lastly, the newly proposed generative model for continous sequences in Chapter 6 has been submitted to AAAI 2019.

This thesis also lead to related work not listed as individual chapters, including a shift-invariant dictionary learning framework for time series data published at KDD 2016 (Zheng et al., 2016), and a variational variant of WaveNet (Oord et al., 2016b) for sequeunces modeling published as another workshop paper at ICML 2018 (Lai et al., 2018).

2

ASSYMETRIC VARIATIONAL AUTOENCODERS

Variational inference for latent variable models is prevalent in various machine learning problems, typically solved by maximizing the Evidence Lower Bound (ELBO) of the true data likelihood with respect to a variational distribution. However, freely enriching the family of variational distribution is challenging since the ELBO requires variational likelihood evaluations of the latent variables. In this paper, we propose a novel framework to enrich the variational family by incorporating auxiliary variables to the variational family. The resulting inference network doesn't require density evaluations for the auxiliary variables and thus complex implicit densities over the auxiliary variables can be constructed by neural networks. It can be shown that the actual variational posterior of the proposed approach is essentially modeling a rich probabilistic mixture of simple variational posterior indexed by auxiliary variables, thus a flexible inference model can be built. Empirical evaluations on several density estimation tasks demonstrates the effectiveness of the proposed method.

2.1 INTRODUCTION

Estimating posterior distributions is the primary focus of Bayesian inference, where we are interested in how our belief over the variables in our model would change after observing a set of data. Predictions can also be benefited from Bayesian inference as every prediction will be equipped with a confidence interval representing how sure the prediction is. Compared to the maximum a posteriori (MAP) estimator of the model parameters, which is a point estimator, the posterior distribution provides richer information about model parameters and hence more justified prediction.

Among various inference algorithms for posterior estimation, variational inference (VI) (Blei et al., 2017) and Markov Chain Monte Carlo (MCMC) (Geyer, 1992) are the most wisely used ones. It is well known that MCMC suffers from slow mixing time though asymptotically the chained samples will approach the true posterior. Furthermore, for latent variable models (LVMs) (Wainwright and

Jordan, 2008) where each sampled data point is associated with a latent variable, the number of simulated Markov Chains increases with the number of data points, making the computation too costly. VI, on the other hand, facilitates faster inference because it optimizes an explicit objective function and its convergence can be measured and controlled. Hence, VI has been widely used in many Bayesian models, such as the mean-field approach for the Latent Dirichlet Allocation (Blei et al., 2003), etc. To enrich the family of distributions over the latent variables, neural network based variational inference methods have also been proposed, such as Variational Autoencoder (VAE) (Kingma and Welling, 2013), Importance Weighted Autoencoder (IWAE) (Burda et al., 2015) and others (Kingma et al., 2016; Mnih and Gregor, 2014; Rezende and Mohamed, 2015). These methods outperform the traditional mean-field based inference algorithms due to their flexible distribution families and easy-to-scale algorithms, therefore becoming the state of the art for variational inference.

The aforementioned VI methods are essentially maximizing the evidence lower bound (ELBO), i.e., the lower bound of the true marginal data likelihood, defined as

$$\log p_\theta(x) \geqslant \mathbb{E}_{z\sim q_\phi(z|x)} \log \frac{p(z,x)}{q(z|x)} \tag{1}$$

where x, z are data point and its latent code, p and q denote the generative model and the variational model, respectively. The equality holds if and only if $q_\phi(z|x) = p_\theta(z|x)$ and otherwise a gap always exists. The more flexible the variational family $q(z|x)$ is, the more likely it will match the true posterior $p(z|x)$. However, arbitrarily enriching the variational model family q is non-trivial, since optimizing Eq. 1 always requires evaluations of $q(z|x)$. Most of existing methods either make over simplified assumptions about the variational model, such as simple Gaussian posterior in VAE (Kingma and Welling, 2013), or resort to implicit variational models without explicitly modeling $q(z|x)$ (Dumoulin et al., 2016).

In this paper we propose to enrich the variational distribution family, by incorporating auxiliary variables to the variational model. *Most importantly, density evaluations are not required for the auxiliary variables and thus complex implicit density over the auxiliary variables can be easily constructed, which in turn results in a flexible variational posterior over the latent variables*. We argue that the resulting inference network is essentially modeling a complex probabilistic mixture of different variational posteriors indexed by the auxiliary variable, and thus a much richer and flexible family of variational posterior distribution is achieved. We conduct empirical evaluations on several density estimation tasks, which validate the effectiveness of the proposed method.

The rest of the paper is organized as follows: We briefly review two existing approaches for inference network modeling in Section 2.2, and present our proposed

framework in the Section 2.3. We then point out the connections of the proposed framework to related methods in Section 2.4. Empirical evaluations and analysis are carried out in Section 2.5, and lastly we conclude this paper in the Section 2.6.

2.2 PRELIMINARIES

In this section, we briefly review several existing methods that aim to address variational inference with stochastic neural networks.

2.2.1 *Variational Autoencoder (VAE)*

Given a generative model $p_\theta(x, z) = p_\theta(z)p_\theta(x|z)$ defined over data x and latent variable z, indexed by parameter θ, variational inference aims to approximate the intractable posterior $p(z|x)$ with $q_\phi(z|x)$, indexed by parameter ϕ, such that the ELBO is maximized

$$\mathcal{L}_{\text{VAE}}(x) \equiv \mathbb{E}_q \log p(x, z) - \mathbb{E}_q \log q(z|x) \leqslant \log p(x) \tag{2}$$

Parameters of both generative distribution p and variational distribution q are learned by maximizing the ELBO with stochastic gradient methods.[1] Specifically, VAE (Kingma and Welling, 2013) assumes both the conditional distribution of data given the latent codes of the generative model and the variational posterior distribution are Gaussians, whose means and diagonal covariances are parameterized by two neural networks, termed as generative network and inference network, respectively. Model learning is possible due to the re-parameterization trick (Kingma and Welling, 2013) which makes back propagation through the stochastic variables possible.

2.2.2 *Importance Weighted Autoencoder (IWAE)*

The above ELBO is a lower bound of the true data log-likelihood $\log p(x)$, hence (Burda et al., 2015) proposed IWAE to directly estimate the true data log-likelihood with the presence of the variational model[2], namely

$$\log p(x) = \log \mathbb{E}_q \frac{p(x, z)}{q(z|x)} \geqslant \log \frac{1}{m} \sum_{i=1}^{m} \frac{p(x, z_i)}{q(z_i|x)} \equiv \mathcal{L}_{\text{IWAE}}(x) \tag{3}$$

where m is the number of importance weighted samples. The above bound is tighter than the ELBO used in VAE. When trained on the same network structure

1 We drop the dependencies of p and q on parameters θ and ϕ to prevent clutter.

2 The variational model is also referred to as the inference model, hence we use them interchangeably.

as VAE, with the above estimate as training objective, IWAE achieves considerable improvements over VAE on various density estimation tasks (Burda et al., 2015) and similar idea is also considered in (Mnih and Rezende, 2016).

2.3 THE PROPOSED METHOD

2.3.1 *Variational Posterior with Auxiliary Variables*

Consider the case of modeling binary data with classic VAE and IWAE, which typically assumes that a data point is generated from a multivariate Bernoulli, conditioned on a latent code which is assumed to be from a Gaussian prior, it's easy to verify that the Gaussian variational posterior inferred by VAE and IWAE will not match the non-Gaussian true posterior.

To this end, we propose to introduce an auxiliary random variable τ to the inference model of VAE and IWAE. Conditioned on the input x, the inference model equipped with auxiliary variable τ now defines a joint density over (τ, z) as

$$q(z, \tau|x) = q(\tau|x)q(z|\tau, x) \tag{4}$$

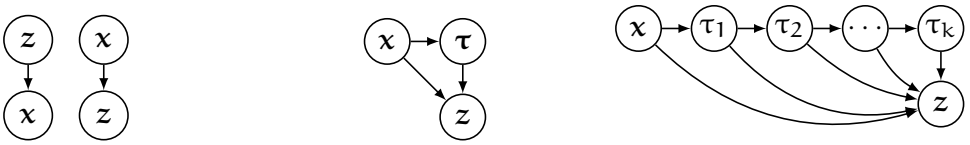
where we assume τ has proper support and both $q(\tau|x)$ and $q(z|\tau, x)$ can be parameterized. Accordingly the marginal variational posterior of z given x turns to be

$$\begin{aligned} q(z|x) &= \int_\tau q(z, \tau|x)d\tau = \int_\tau q(z|\tau, x)q(\tau|x)d\tau \\ &= \mathbb{E}_{q(\tau|x)}\, q(z|\tau, x) \end{aligned} \tag{5}$$

which essentially models the posterior $q(z|x)$ as a probabilistic mixture of different densities $q(z|\tau, x)$ indexed by τ, together with $q(\tau|x)$ as the mixture weights. This allows complex and flexible posterior $q(z|x)$ to be constructed, even when both $q(\tau|x)$ and $q(z|\tau, x)$ are from simple density families. Due to the presence of auxiliary variables τ, the inference model is trying to capture more sources of stochasticity than the generative model, hence we term our approach as Asymmetric Variational Autoencoder (AVAE). Figure 1a and 1b present a comparison of the inference models between classic VAE and the proposed AVAE.

In the context of of VAE and IWAE, the proposed approach includes two instantiations, AVAE and IW-AVAE, with loss functions

$$\begin{aligned} &\mathcal{L}_{\text{AVAE}}(x) \equiv \mathbb{E}_{q(z|x)}[\log p(x, z) - \log q(z|x)] \\ =&\mathbb{E}_{q(z|x)}\Big(\log p(x, z) - \log \mathbb{E}_{q(\tau|x)}\, q(z|\tau, x)\Big) \end{aligned} \tag{6}$$



(a) Generative model (left) and inference model (right) for VAE

(b) Inference model for AVAE (Generative model is the same as in VAE)

(c) Inference model for AVAE with k auxiliary variables

Figure 1: Inference models for VAE, AVAE and AVAE with k auxiliary random variables (The generative model is fixed as shown in Figure 1a). Note that multiple arrows pointing to a node indicate one stochastic layer, with the source nodes concatenated as input to the stochastic layer and the target node as stochastic output. One stochastic layer could consist of multiple deterministic layers. (For detailed architecture used in experiments, refer to Section 2.5.)

and

$$\begin{aligned}\mathcal{L}_{\text{IW-AVAE}}(x) &\equiv \log \mathbb{E}_{q(z|x)} \frac{p(x,z)}{q(z|x)} \\ &= \log \mathbb{E}_{q(z|x)} \frac{p(x,z)}{\mathbb{E}_{q(\tau|x)} q(z|\tau,x)}\end{aligned} \tag{7}$$

respectively.

AVAE enjoys the following properties:

- **VAEs are special cases of AVAE**. Conventional variational autoencoders can be seen as special cases of AVAE with no auxiliary variables τ assumed;
- **No density evaluations for τ are required**. One key advantage brought by the auxiliary variable τ is that both terms inside the inner expectations of $\mathcal{L}_{\text{AVAE}}$ and $\mathcal{L}_{\text{IW-AVAE}}$ do not involve $q(\tau|x)$, hence no density evaluations are required when Monte Carlo samples of τ are used to optimize the above bounds.
- **Flexible variational posterior.** To fully enrich variational model flexibility, we use a neural network f to *implicitly* model $q(\tau|x)$ by sampling τ given x and a random Gaussian noise vector ϵ as

  $$\tau = f(x, \epsilon) \text{ with } \epsilon \sim \mathcal{N}(0, I) \tag{8}$$

  Due to the flexible representative power of f, the implicit density $q(\tau|x)$ can be arbitrarily complex. Further we assume $q(z|\tau, x)$ to be Gaussian with its mean and variance parameterized by neural networks. Since the actual variational posterior $q(z|x) = \mathbb{E}_\tau q(z|x,\tau)$, complex posterior can be achieved

even a simple density family is assumed for $q(z|x,\tau)$, due to the possibly flexible family of *implicit* density of $q(\tau|x)$ defined by $f(x,\epsilon)$. (Illustration can be found in Section 2.5.1)

For completeness, we briefly include that

Proposition 1 *Both* $\mathcal{L}_{\text{AVAE}}(x)$ *and* $\mathcal{L}_{\text{IW-AVAE}}(x)$ *are lower bounds of the true data log-likelihood, satisfying* $\log p(x) = \mathcal{L}_{\text{IW-AVAE}}(x) \geqslant \mathcal{L}_{\text{AVAE}}(x)$.

Proof is trivial from Jensen's inequality, hence it's omitted.

Remark 1 Though the first equality holds for any choice of distribution $q(\tau|x)$ (whether τ depends on x or not), for practical estimation with Monte Carlo methods, it becomes an inequality $(\log p(x) \geqslant \hat{\mathcal{L}}_{\text{IW-AVAE}}(x))$ and the bound tightens as the number of importance samples is increased (Burda et al., 2015). The second inequality always holds when estimated with Monte Carlo samples.

Remark 2 The above bounds are only concerned with one auxiliary variable τ, in fact τ can also be a set of auxiliary variables. Moreover, with the same motivation, we can make the variational family of AVAE even more flexible by defining a series of k auxiliary variables, such that

$$q(z,\tau_1,...,\tau_k|x) = q(\tau_1|x)q(\tau_2|\tau_1,x)...q(\tau_k|\tau_{k-1},x)q(z|\tau_1,...,\tau_k,x) \tag{9}$$

with sample generation process for all τs defined as

$$\begin{aligned}\tau_1 &= f_1(x,\epsilon_1)\\ \tau_i &= f_i(\tau_{i-1},\epsilon_k) \text{ for } i = 2,3,...,k\end{aligned} \tag{10}$$

where all ϵ_i are random noise vectors and all f_i are neural networks to be learned. Accordingly, we have

Proposition 2 *The AVAE with* k *auxiliary random variables* $\{\tau_1,\tau_2,...,\tau_k\}$ *is also a lower bound to the true log-likelihood, satisfying* $\log p(x) = \mathcal{L}_{\text{IW-AVAE-k}} \geqslant \mathcal{L}_{\text{AVAE-k}}$, *where*

$$\begin{aligned}&\mathcal{L}_{\text{AVAE-k}}(x) \equiv \mathbb{E}_{q(z|x)}[\log p(x,z) - \log q(z|x)]\\ =&\mathbb{E}_{q(z|x)}\Big(\log p(x,z) - \log \mathbb{E}_{q(\tau_1,\tau_2,...,\tau_k|x)} q(z|\tau_1,...,\tau_k,x)\Big)\end{aligned} \tag{11}$$

and

$$\begin{aligned}&\mathcal{L}_{\text{IW-AVAE-k}}(x) \equiv \log \mathbb{E}_{q(z|x)} \frac{p(x,z)}{q(z|x)}\\ =&\log \mathbb{E}_{q(z|x)} \frac{p(x,z)}{\mathbb{E}_{q(\tau_1,\tau_2,...,\tau_k|x)} q(z|\tau_1,...,\tau_k,x)}\end{aligned} \tag{12}$$

Figure 1c illustrates the inference model of an AVAE with k auxiliary variables.

2.3.2 *Learning with Importance Weighted Auxiliary Samples*

For both AVAE and IW-AVAE, we can estimate the corresponding bounds and its gradients of $\mathcal{L}_{\text{AVAE}}$ and $\mathcal{L}_{\text{IW-AVAE}}$ with ancestral sampling from the model. For example, for AVAE with one auxiliary variable τ, we estimate

$$\hat{\mathcal{L}}_{\text{AVAE}}(x) = \frac{1}{m}\sum_{i=1}^{m}\left(\log p(x, z_i) - \log \frac{1}{n}\sum_{j=1}^{n} q(z_i|\tau_j, x)\right) \tag{13}$$

and

$$\hat{\mathcal{L}}_{\text{IW-AVAE}}(x) = \log \frac{1}{m}\sum_{i=1}^{m} \frac{p(x, z_i)}{\frac{1}{n}\sum_{j=1}^{n} q(z_i|\tau_j, x)} \tag{14}$$

where n is the number of τs sampled from the current $q(\tau|x)$ and m is the number of zs sampled from the *implicit* conditional $q(z|x)$, which is by definition achieved by first sampling from $q(\tau|x)$ and subsequently sampling from $q(z|\tau, x)$. The parameters of both the inference model and generative model are jointly learned by maximizing the above bounds. Besides back propagation through the stochastic variable z (typically assumed to be a Gaussian for continuous latent variables) is possible through the re-parameterization trick, and it is naturally also true for all the auxiliary variables τ since they are constructed in a generative manner.

The term $\frac{1}{n}\sum_{j=1}^{n} q(z_i|\tau_j, x)$ essentially is an n-sample importance weighted estimate of $q(z|x) = \mathbb{E}_\tau q(z|\tau, x)$, hence it is reasonable to believe that more samples of τ will lead to less noisy estimate of $q(\tau|x)$ and thus a more accurate inference model q. It's worth pointing out for AVAE that additional samples of τ comes almost at no cost when multiple samples of z are generated ($m > 1$) to optimize $\mathcal{L}_{\text{AVAE}}$ and $\mathcal{L}_{\text{IW-AVAE}}$, since sampling a z from the inference model will also generate intermediate samples of τ, thus we can always reuse those samples of τ to estimate $q(z|x) = \mathbb{E}_\tau q(z|\tau, x)$. For this purpose, in our experiments we always assume $n = m$ so that no separate process of sampling τ is needed in estimating the lower bounds. This also ensures that the forward pass and backward pass time complexity of the inference model are the same as conventional VAE and IWAE. In fact, as we will show in all our empirical evaluations that if $n = 1$ AVAE performs similarly to VAE and while $n > 1$ IW-AVAE always outperforms IWAE, i.e., its counterpart with no auxiliary variables.

2.4 CONNECTION TO RELATED METHODS

Before we proceed to the experimental evaluations of the proposed methods, we highlight the relations of AVAE to other similar methods.

2.4.1 *Other methods with auxiliary variables*

Relation to Hierarchical Variational Models (HVM) (Ranganath et al., 2016) and Auxiliary Deep Generative Models (ADGM) (Maaløe et al., 2016) are two closely related variational methods with auxiliary variables. HVM also considers enriching the variational model family by placing a prior over the latent variable for the variational distribution $q(z|x)$. While ADGM takes another way to this goal, by placing a prior over the auxiliary variable on the generative model, which in some cases will keep the marginal generative distribution of the data invariant. It has been shown that HVM and ADGM are mathematically equivalent by (Brümmer, 2016).

However, our proposed method doesn't add any prior on the generative model and thus doesn't change the structure of the generative model. We emphasize that our proposed method makes the least assumption about the generative model and that the proposal in our method is orthogonal to related methods, thus it can can be integrated with previous methods with auxiliary variables to further boost the performance on accurate posterior approximation and generative modeling.

2.4.2 *Adversarial learning based inference models*

Adversarial learning based inference models, such as Adversarial Autoencoders (Makhzani et al., 2015), Adversarial Variational Bayes (Mescheder et al., 2017), and Adversarially Learned Inference (Dumoulin et al., 2016), aim to maximize the ELBO without any variational likelihood evaluations at all. It can be shown that for the above adversarial learning based models, when the discriminator is trained to its optimum, the model is equivalent to optimizing the ELBO. However, due to the minimax game involved in the adversarial setting, practically at any moment it is not guaranteed that they are optimizing a lower bound of the true data likelihood, thus no maximum likelihood learning interpretation can be provided. Instead in our proposed framework, we don't require variational density evaluations for the flexible auxiliary variables, while still maintaining the maximum likelihood interpretation.

2.5 EXPERIMENTS

2.5.1 *Flexible Variational Family of AVAE*

To test the effect of adding auxiliary variables to the inference model, we parameterize two unnormalized 2D target densities $p(z) \propto \exp(U(z))$[3] with

$$\begin{aligned} U_1(z) =& \frac{1}{2}\left(\frac{\|z\|-2}{4}\right)^2 \\ & -\log\left(e^{-\frac{1}{2}\left[\frac{z_1-2}{0.6}\right]^2} + e^{-\frac{1}{2}\left[\frac{z_1+2}{0.6}\right]^2}\right) \end{aligned}$$

$$\text{and } U_2(z) = \frac{1}{2}\left[\frac{z_2 - w_1(z)}{0.4}\right]^2 \quad \text{where } w_1(z) = \sin\left(\frac{\pi z_1}{2}\right)$$

We construct inference model[4] to approximate the target density by minimizing the KL divergence

$$\begin{aligned} \mathrm{KL}(q(z)\|p(z)) &= \mathbb{E}_{z\sim q(z)}\big(\log q(z) - \log p(z)\big) \\ &= \mathbb{E}_{z\sim q(z)}\big(\log \mathbb{E}_\tau q(z|\tau) - \log p(z)\big) \end{aligned} \tag{15}$$

Figure 2 illustrates the target densities as well as the ones learned by VAE and AVAE, respectively. It's unsurprising to see that standard VAE with Gaussian stochastic layer as its inference model will only be able to produce Gaussian density estimates (Figure 2(b)). While with the help of introduced auxiliary random variables, AVAE is able to match the non-Gaussian target densities (Figure 2(c)), even the last stochastic layer of the inference model, i.e., $q(z|\tau)$, is also Gaussian.

2.5.2 *Handwritten Digits and Characters*

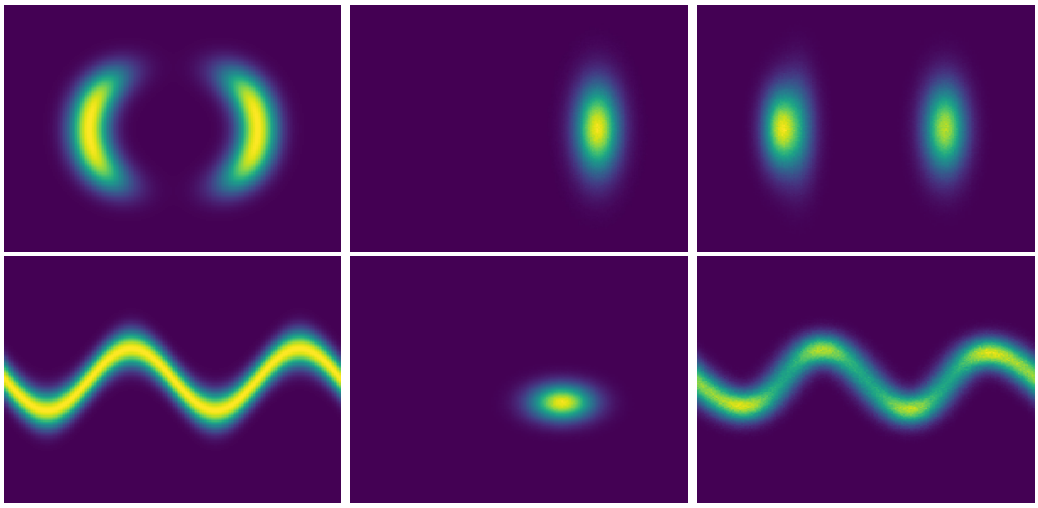
To test AVAE for variational inference we use standard benchmark datasets MNIST[5] and OMNIGLOT[6] (Lake et al., 2013). Our method is general and can be applied to any formulation of the generative model $p_\theta(x, z)$. For simplicity and fair comparison, in this paper we focus on $p_\theta(x, z)$ defined by stochastic neural networks, i.e., a family of generative models with their parameters defined by neural networks. Specifically, we consider the following two types of generative models:

3 Sample densities originate from (Rezende and Mohamed, 2015)

4 Inference model of VAE defines a conditional variational posterior $q(z|x)$, to match the target density $p(z)$ which is independent of x, we set x to be fixed. In this synthetic example, x is set to be an all one vector of dimension 10.

5 http://www.cs.toronto.edu/~larocheh/public/datasets/binarized_mnist/

6 https://github.com/yburda/iwae/raw/master/datasets/OMNIGLOT/chardata.mat



(a) (b) (c)

Figure 2: (a) True density; (b) Density learned by VAE; (c) Density learned by AVAE.

G_1 : $p_\theta(x,z) = p_\theta(z)p_\theta(x|z)$ with single Gaussian stochastic layer for z with 50 units. In between the latent variable z and observation x there are two deterministic layers, each with 200 units;

G_2 : $p_\theta(x,z_1,z_2) = p_\theta(z_1)p_\theta(z_2|z_1)p_\theta(x|z_2)$ with two Gaussian stochastic layers for z_1 and z_2 with 50 and 100 units, respectively. Two deterministic layers with 200 units connect the observation x and latent variable z_2, and two deterministic layers with 100 units connect z_2 and z_1.

A Gaussian stochastic layer consists of two fully connected linear layers, with one outputting the mean and the other outputting the logarithm of diagonal covariance. All other deterministic layers are fully connected with tanh nonlinearity. The same network architectures for both G_1 and G_2 are also used in (Burda et al., 2015)

For G_1, an inference network with the following architecture is used by AVAE with k auxiliary variables

$$\tau_i = f_i(\tau_{i-1}\|\epsilon_i) \text{ where } \epsilon_i \sim \mathcal{N}(0,I) \text{ for } i = 1,2,...,k$$
$$q(z|x,\tau_1,...,\tau_k) = \mathcal{N}\Big(\mu(x\|\tau_1\|...\|\tau_k), \text{diag}\big(\sigma(x\|\tau_1\|...\|\tau_k)\big)\Big)$$

where τ_0 is defined as input x, all f_i are implemented as fully connected layers with tanh nonlinearity and $\|$ denotes the concatenation operator. All noise vectors ϵs are set to be of 50 dimensions, and all other variables have the corresponding dimensions in the generative model. Inference network used for G_2 is the same, except that the Gaussian stochastic layer is defined for z_2. An additional Gaussian

stochastic layer for z_1 is defined with z_2 as input, where the dimensions of variables aligned to those in the generative model G_2. Further, Bernoulli observation models are assumed for both MNIST and OMNIGLOT. For MNIST, we employ the static binarization strategy as in (Larochelle and Murray, 2011) while dynamic binarization is employed for OMNIGLOT.

Our baseline models include VAE and IWAE. Since our proposed method involves adding more layers to the inference network, we also include another enhanced version of VAE with more deterministic layers added to its inference network, which we term as VAE+[7] and its importance sample weighted variant IWAE+. To eliminate discrepancies in implementation details of the models reported in the literature, we implement all models and carry out the experiments under the same setting: All models are implemented in PyTorch[8] and parameters of all models are optimized with Adam (Kingma and Ba, 2014) for 2000 epochs, with an initial learning rate of 0.001, cosine annealing for learning rate decay (Loshchilov and Hutter, 2016), exponential decay rates for the 1st and 2nd moments at 0.9 and 0.999, respectively. Batch normalization (Ioffe and Szegedy, 2015) is applied to all fully connected layers, except for the final output layer for the generative model, as it has been shown to improve learning for neural stochastic models (Sønderby et al., 2016). Linear annealing of the KL divergence term between the variational posterior and the prior in all the loss functions from 0 to 1 is adopted for the first 200 epochs, as it has been shown to help training stochastic neural networks with multiple layers of latent variables (Sønderby et al., 2016). Code to reproduce all reported results will be made publicly available.

2.5.2.1 *Generative Density Estimation*

For both MNIST and OMNIGLOT, all models are trained and tuned on the training and validation sets, and estimated log-likelihood on the test set with 128 importance weighted samples are reported. Table 1 presents the performance of all models with for both G_1 and G_2.

Firstly, VAE+ achieves slightly higher log-likelihood estimates than vanilla VAE due to the additional layers added in the inference network, implying that a better Gaussian posterior approximation is learned. Second, AVAE achieves lower NLL estimates than VAE+, more so with increasingly more samples from auxiliary variables (i.e., larger m), which confirms our expectation that: a) adding auxiliary variables to the inference network leads to a richer family of variational distributions; b) more samples of auxiliary variables yield a more accurate estimate of variational posterior $q(z|x)$. We also point out that there's a trade-off between the

7 VAE+ is a restricted version of AVAE with all the noise vectors ϵs set to be constantly 0, but with the additional layers for fs retained.

8 http://pytorch.org/

Table 1: MNIST and OMNIGLOT test set NLL with generative models G_1 and G_2 (*Lower is better; for VAE+, k is the number of additional layers added and for AVAE it is the number of auxiliary variables added. For each column, the best result for each k of both type of models (VAE based and IWAE based) are printed in bold.)*

	MNIST		OMNIGLOT	
Models	$-\log p(x)$ on G_1	$-\log p(x)$ on G_2	$-\log p(x)$ on G_1	$-\log p(x)$ on G_2
VAE (Burda et al., 2015)	88.37	85.66	108.22	106.09
VAE+ $(k = 1)$	88.20	85.41	108.30	106.30
VAE+ $(k = 4)$	88.08	85.26	108.31	106.48
VAE+ $(k = 8)$	87.98	85.16	108.31	106.05
AVAE $(k = 1)$	88.20	85.52	108.27	106.59
AVAE $(k = 4)$	88.18	85.36	108.21	106.43
AVAE $(k = 8)$	88.23	85.33	108.20	106.49
AVAE $(k = 1, m = 50)$	**87.21**	**84.57**	**106.89**	**104.59**
AVAE $(k = 4, m = 50)$	**86.98**	**84.39**	**106.50**	**104.76**
AVAE $(k = 8, m = 50)$	**86.89**	**84.36**	**106.51**	**104.67**
Models (Importance weighted)				
IWAE $(m = 50)$ (Burda et al., 2015)	86.90	84.26	106.08	104.14
IW-AVAE $(k = 1, m = 5)$	86.86	84.47	106.80	104.67
IW-AVAE $(k = 4, m = 5)$	86.57	84.55	106.93	104.87
IW-AVAE $(k = 8, m = 5)$	86.67	84.44	106.57	105.06
IWAE+ $(k = 1, m = 50)$	86.70	84.28	105.83	**103.79**
IWAE+ $(k = 4, m = 50)$	86.31	83.92	105.81	**103.71**
IWAE+ $(k = 8, m = 50)$	86.40	84.06	105.73	**103.77**
IW-AVAE $(k = 1, m = 50)$	**86.08**	**84.19**	**105.49**	103.84
IW-AVAE $(k = 4, m = 50)$	**86.02**	**84.05**	**105.53**	103.89
IW-AVAE $(k = 8, m = 50)$	**85.89**	**83.77**	**105.39**	103.97

model expressiveness and complexity, as we found that increasing the number of auxiliary variables to 16, 32, the performances saturate and also model training is slowed down. In practice, we found that use 4 to 8 auxiliary variables works fairly well balancing the two. Lastly, with more importance weighted samples from both τ and z, i.e., IW-AVAE variants, the best data density estimates are achieved. Overall, on MNIST AVAE outperforms VAE by 1.5 nats on G_1 and 1.3 nats on G_2; IW-AVAE outperforms IWAE by about 1.0 nat on G_1 and 0.5 nats on G_2. Similar trends can be observed on OMNIGLOT, with AVAE and IW-AVAE outperforming conventional VAE and IWAE in all cases, except for G_2 IWAE+ slightly outperforms IW-AVAE.

Compared with previous methods with similar settings, IW-AVAE achieves a best NLL of 83.77, significantly better than 85.10 achieved by Normalizing Flow (Rezende and Mohamed, 2015). Best density modeling with generative modeling on statically binarized MNIST is achieved by Pixel RNN (Oord et al., 2016a; Salimans et al., 2017) with autoregressive models and Inverse Autoregressive Flows (Kingma et al., 2016) with latent variable models, however it's worth noting that much more sophisticated generative models are adopted in those methods and that AVAE enhances standard VAE by focusing on enriching inference model flexibility, which pursues an orthogonal direction for improvements. Therefore, AVAE can be integrated with above-mentioned methods to further improve performance on latent generative modeling.

2.5.2.2 *Latent Code Visualization*

We visualize the inferred latent codes z of digits in the MNIST test set with respect to their true class labels in Figure 3 from different models with tSNE (Maaten and Hinton, 2008). We observe that on generative model G_2, all three models are able

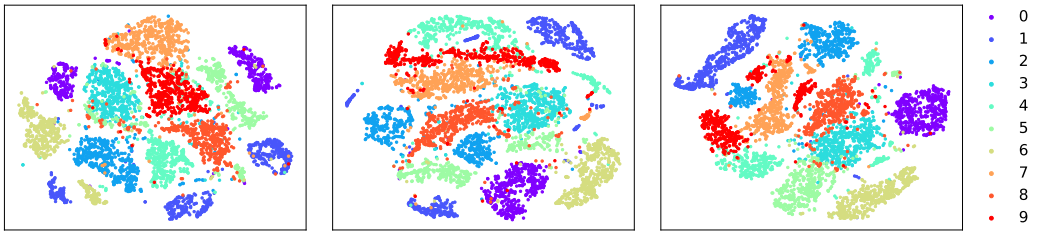
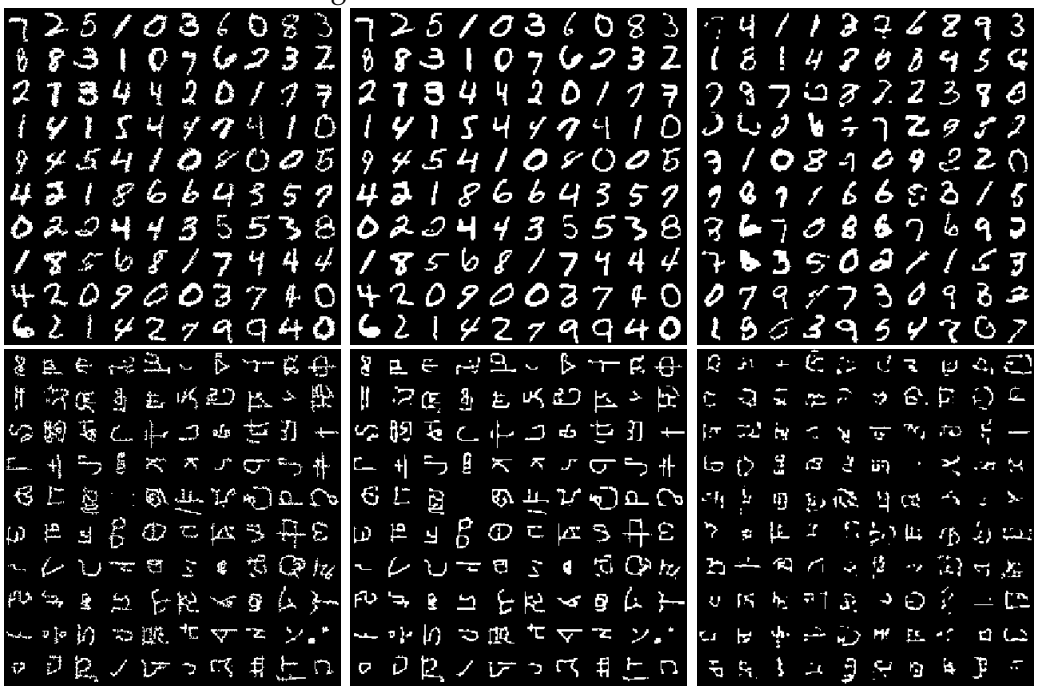


Figure 3: **Left:** VAE, **Middle:** VAE+, **Right:**AVAE. Visualization of inferred latent codes for 5000 MNIST digits in the test set (best viewed in color)

to infer latent codes of the digits consistent with their true classes. However, VAE and VAE+ still shows disconnected cluster of latent codes from the same class (both class 0 and 1) and latent code overlapping from different classes (class 3 and 5), while AVAE outputs clear separable latent codes for different classes (notably for class 0,1,5,6,7).

2.5.2.3 *Reconstruction and Generated Samples*

Generative samples can be obtained from trained model by feeding $z \sim N(0, I)$ to the learned generative model G_1 (or $z_2 \sim N(0, I)$ to G_2). Since higher log-likelihood estimates are obtained on G_2, Figure 4 shows real samples from the dataset, their reconstruction, and random data points sampled from AVAE trained on G_2 for both MNIST and OMNIGLOT. We observe that the reconstructions align well with the input data and that random samples generated by the models are visually consistent with the training data.



(a) Data (b) Reconstruction (c) Random samples

Figure 4: Training data, its reconstruction and random samples. (Upper: MNIST, Lower: OMNIGLOT)

2.6 CONCLUSIONS

This paper presents AVAE, a new framework to enrich variational family for variational inference, by incorporating auxiliary variables to the inference model. It can be shown that the resulting inference model is essentially learning a richer probabilistic mixture of simple variational posteriors indexed by the auxiliary variables. We emphasize that *no density evaluations are required for the auxiliary variables*, hence neural networks can be used to construct complex implicit distribution for the auxiliary variables. Empirical evaluations of two variants of AVAE demonstrate

the effectiveness of incorporating auxiliary variables in variational inference for generative modeling.

3

CONVOLUTIONAL NORMALIZING FLOWS

Bayesian posterior inference is prevalent in various machine learning problems. Variational inference provides one way to approximate the posterior distribution, however its expressive power is limited and so is the accuracy of resulting approximation. Recently, there has a trend of using neural networks to approximate the variational posterior distribution due to the flexibility of neural network architecture. One way to construct flexible variational distribution is to warp a simple density into a complex by normalizing flows, where the resulting density can be analytically evaluated. However, there is a trade-off between the flexibility of normalizing flow and computation cost for efficient transformation. In this chapter, we propose a simple yet effective architecture of normalizing flows, *ConvFlow*, based on convolution over the dimensions of random input vector. Experiments on synthetic and real world posterior inference problems demonstrate the effectiveness and efficiency of the proposed method.

3.1 INTRODUCTION

Posterior inference is the key to Bayesian modeling, where we are interested to see how our belief over the variables of interest change after observing a set of data points. Predictions can also benefit from Bayesian modeling as every prediction will be equipped with confidence intervals representing how sure the prediction is. Compared to the maximum a posterior estimator of the model parameters, which is a point estimator, the posterior distribution provide richer information about the model parameter hence enabling more justified prediction.

Among the various inference algorithms for posterior estimation, variational inference (VI) and Monte Carlo Markov chain (MCMC) are the most two widely used ones. It is well known that MCMC suffers from slow mixing time though asymptotically the samples from the chain will be distributed from the true posterior. VI, on the other hand, facilitates faster inference, since it is optimizing an explicit objective function and convergence can be measured and controlled, and it's been widely used in many Bayesian models, such as Latent Dirichlet

Allocation (Blei et al., 2003), etc. However, one drawback of VI is that it makes strong assumption about the shape of the posterior such as the posterior can be decomposed into multiple independent factors. Though faster convergence can be achieved by parameter learning, the approximating accuracy is largely limited.

The above drawbacks stimulates the interest for richer function families to approximate posteriors while maintaining acceptable learning speed. Specifically, neural network is one among such models which has large modeling capacity and endows efficient learning. (Rezende and Mohamed, 2015) proposed normalization flow, where the neural network is set up to learn an invertible transformation from one known distribution, which is easy to sample from, to the true posterior. Model learning is achieved by minimizing the KL divergence between the empirical distribution of the generated samples and the true posterior. After properly trained, the model will generate samples which are close to the true posterior, so that Bayesian predictions are made possible. Other methods based on modeling random variable transformation, but based on different formulations are also explored, including NICE (Dinh et al., 2014), the Inverse Autoregressive Flow (Kingma et al., 2016), and Real NVP (Dinh et al., 2016).

One key component for normalizing flow to work is to compute the determinant of the Jacobian of the transformation, and in order to maintain fast Jacobian computation, either very simple function is used as the transformation, such as the planar flow in (Rezende and Mohamed, 2015), or complex tweaking of the transformation layer is required. Alternatively, in this paper we propose a simple and yet effective architecture of normalizing flows, based on convolution on the random input vector. Due to the nature of convolution, bi-jective mapping between the input and output vectors can be easily established; meanwhile, efficient computation of the determinant of the convolution Jacobian is achieved linearly. We further propose to incorporate dilated convolution (Oord et al., 2016b; Yu and Koltun, 2015) to model long range interactions among the input dimensions. The resulting convolutional normalizing flow, which we term as *Convolutional Flow (ConvFlow)*, is simple and yet effective in warping simple densities to match complex ones.

The remainder of this paper is organized as follows: We briefly review the principles for normalizing flows in Section 3.2, and then present our proposed normalizing flow architecture based on convolution in Section 3.3. Empirical evaluations and analysis on both synthetic and real world data sets are carried out in Section 3.4, and we conclude this paper in Section 3.5.

3.2 PRELIMINARIES

3.2.1 *Transformation of random variables*

Given a random variable $z \in \mathbb{R}^d$ with density $p(z)$, consider a smooth and invertible function $f : \mathbb{R}^d \to \mathbb{R}^d$ operated on z. Let $z' = f(z)$ be the resulting random variable, the density of z' can be evaluated as

$$p(z') = p(z) \left| \det \frac{\partial f^{-1}}{\partial z'} \right| = p(z) \left| \det \frac{\partial f}{\partial z} \right|^{-1} \tag{16}$$

thus

$$\log p(z') = \log p(z) - \log \left| \det \frac{\partial f}{\partial z} \right| \tag{17}$$

3.2.2 *Normalizing flows*

Normalizing flows considers successively transforming z_0 with a series of transformations $\{f_1, f_2, ..., f_K\}$ to construct arbitrarily complex densities for $z_K = f_K \circ f_{K-1} \circ ... \circ f_1(z_0)$ as

$$\log p(z_K) = \log p(z_0) - \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial z_{k-1}} \right| \tag{18}$$

Hence the complexity lies in computing the determinant of the Jacobian matrix. Without further assumption about f, the general complexity for that is $\mathcal{O}(d^3)$ where d is the dimension of z. In order to accelerate this, (Rezende and Mohamed, 2015) proposed the following family of transformations that they termed as *planar flow*:

$$f(z) = z + \mathbf{u} h(\boldsymbol{w}^\top z + b) \tag{19}$$

where $\boldsymbol{w} \in \mathbb{R}^d, \mathbf{u} \in \mathbb{R}^d, b \in \mathbb{R}$ are parameters and $h(\cdot)$ is a univariate non-linear function with derivative $h'(\cdot)$. For this family of transformations, the determinant of the Jacobian matrix can be computed as

$$\det \frac{\partial f}{\partial z} = \det(\mathbf{I} + \mathbf{u}\psi(z)^\top) = 1 + \mathbf{u}^\top \psi(z) \tag{20}$$

where $\psi(z) = h'(\boldsymbol{w}^\top z + b)\boldsymbol{w}$. The computation cost of the determinant is hence reduced from $\mathcal{O}(d^3)$ to $\mathcal{O}(d)$.

Applying f to z can be viewed as feeding the input variable z to a neural network with only one single hidden unit followed by a linear output layer which has the same dimension with the input layer. Obviously, because of the bottleneck caused by the single hidden unit, the capacity of the family of transformed density is hence limited.

3.3 A NEW TRANSFORMATION UNIT

In this section, we first propose a general extension to the above mentioned planar normalizing flow, and then propose a restricted version of that, which actually turns out to be convolution over the dimensions of the input random vector.

3.3.1 *Normalizing flow with* d *hidden units*

Instead of having a single hidden unit as suggested in planar flow, consider d hidden units in the process. We denote the weights associated with the edges from the input layer to the output layer as $\boldsymbol{W} \in \mathbb{R}^{d \times d}$ and the vector to adjust the magnitude of each dimension of the hidden layer activation as $\mathbf{u}$, and the transformation is defined as

$$f(\boldsymbol{z}) = \mathbf{u} \odot h(\boldsymbol{W}\boldsymbol{z} + \mathbf{b}) \tag{21}$$

where $\odot$ denotes the point-wise multiplication. The Jacobian matrix of this transformation is

$$\frac{\partial f}{\partial \boldsymbol{z}} = \text{diag}(\mathbf{u} \odot h'(\boldsymbol{W}\boldsymbol{z} + b))\boldsymbol{W} \tag{22}$$

$$\det \frac{\partial f}{\partial \boldsymbol{z}} = \det[\text{diag}(\mathbf{u} \odot h'(\boldsymbol{W}\boldsymbol{z} + b))] \det(\boldsymbol{W}) \tag{23}$$

As $\det(\text{diag}(\mathbf{u} \odot h'(\boldsymbol{W}\boldsymbol{z} + \mathbf{b})))$ is linear, the complexity of computing the above transformation lies in computing $\det(\boldsymbol{W})$. Essentially the planar flow is restricting $\boldsymbol{W}$ to be a vector of length d instead of matrices, however we can relax that assumption while still maintaining linear complexity of the determinant computation based on a very simple fact that the determinant of a triangle matrix is also just the product of the elements on the diagonal.

3.3.2 *Convolutional Flow*

Since normalizing flow with a fully connected layer may not be bijective and generally requires $\mathcal{O}(d^3)$ computations for the determinant of the Jacobian even it is, we propose to use 1-d convolution to transform random vectors.

Figure 5(a) illustrates how 1-d convolution is performed over an input vector and outputs another vector. We propose to perform a 1-d convolution on an input random vector $\boldsymbol{z}$, followed by a non-linearity and necessary post operation after activation to generate an output vector. Specifically,

$$f(\boldsymbol{z}) = \boldsymbol{z} + \mathbf{u} \odot h(\text{conv}(\boldsymbol{z}, \boldsymbol{w})) \tag{24}$$

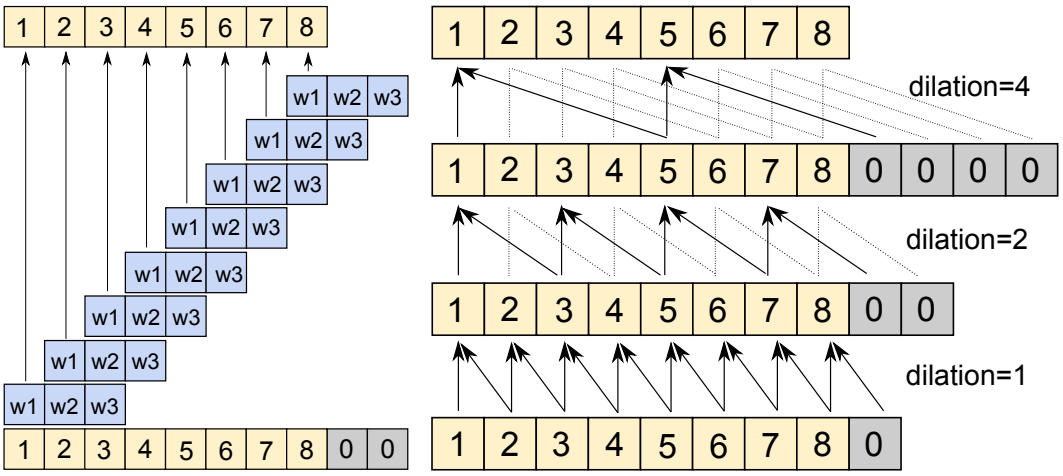


Figure 5: (a) Illustration of 1-D convolution, where the dimensions of the input/output variable are both 8 (the input vector is padded with 0), the width of the convolution filter is 3 and dilation is 1; (b) A block of ConvFlow layers stacked with different dilations.

where $\boldsymbol{w} \in \mathbb{R}^k$ is the parameter of the 1-d convolution filter (k is the convolution kernel width), $\mathrm{conv}(\boldsymbol{z}, \boldsymbol{w})$ is the 1d convolution operation as shown in Figure 5(a), $h(\cdot)$ is a monotonic non-linear activation function[1], $\odot$ denotes point-wise multiplication, and $\mathbf{u} \in \mathbb{R}^d$ is a vector adjusting the magnitude of each dimension of the activation from $h(\cdot)$. We term this normalizing flow as *Convolutional Flow (ConvFlow)*.

ConvFlow enjoys the following properties

- Bi-jectivity can be easily achieved with standard and fast 1d convolution operator if proper padding and a monotonic activation function with bounded gradients are adopted (Minor care is needed to guarantee strict invertibility, see Appendix 3.6 for details);
- Due to local connectivity, the Jacobian determinant of ConvFlow only takes $\mathcal{O}(d)$ computation independent from convolution kernel width k since

$$\frac{\partial \mathbf{f}}{\partial \boldsymbol{z}} = \mathrm{I} + \mathrm{diag}(w_1 \mathbf{u} \odot h'(\mathrm{conv}(\boldsymbol{z}, \boldsymbol{w}))) \tag{25}$$

1 Examples of valid $h(x)$ include all conventional activations, including sigmoid, tanh, softplus, rectifier (ReLU), leaky rectifier (Leaky ReLU) and exponential linear unit (ELU).

where w_1 denotes the first element of $\boldsymbol{w}$.
For example for the illustration in Figure 5(a), the Jacobian matrix of the 1d convolution $\text{conv}(\boldsymbol{z}, \boldsymbol{w})$ is

$$\frac{\partial\, \text{conv}(\boldsymbol{z}, \boldsymbol{w})}{\partial \boldsymbol{z}} = \begin{bmatrix} w_1 & w_2 & w_3 & & & & & \\ & w_1 & w_2 & w_3 & & & & \\ & & w_1 & w_2 & w_3 & & & \\ & & & w_1 & w_2 & w_3 & & \\ & & & & w_1 & w_2 & w_3 & \\ & & & & & w_1 & w_2 & w_3 \\ & & & & & & w_1 & w_2 \\ & & & & & & & w_1 \end{bmatrix} \tag{26}$$

which is a triangular matrix whose determinant can be easily computed;

- ConvFlow is much simpler than previously proposed variants of normalizing flows. The total number of parameters of one ConvFlow layer is only $d + k$ where generally $k < d$, particularly efficient for high dimensional cases. Notice that the number of parameters in the planar flow in (Rezende and Mohamed, 2015) is 2d and one layer of Inverse Autoregressive Flow (IAF) (Kingma et al., 2016) and Real NVP (Dinh et al., 2016) require even more parameters. In Section 3.3.3, we discuss the key differences of ConvFlow from IAF in detail.

A series of K ConvFlows can be stacked to generate complex output densities. Further, since convolutions are only visible to inputs from adjacent dimensions, we propose to incorporate dilated convolution (Oord et al., 2016b; Yu and Koltun, 2015) to the flow to accommodate interactions among dimensions with long distance apart. Figure 5(b) presents a block of 3 ConvFlows stacked, with different dilations for each layer. Larger receptive field is achieved without increasing the number of parameters. We term this as a ConvBlock.

From the block of ConvFlow layers presented in Figure 5(b), it is easy to verify that dimension $i\,(1 \leqslant i \leqslant d)$ of the output vector only depends on succeeding dimensions, but not preceding ones. In other words, dimensions with larger indices tend to end up getting little warping compared to the ones with smaller indices. Fortunately, this can be easily resolved by a *Revert Layer*, which simply outputs a reversed version of its input vector. Specifically, a Revert Layer g operates as

$$g(\boldsymbol{z}) := g([z_1, z_2, ..., z_d]^\top) = [z_d, z_{d-1}, ..., z_1]^\top \tag{27}$$

It's easy to verify a Revert Layer is bijective and that the Jacobian of g is a $d \times d$ matrix with 1s on its anti-diagonal and 0 otherwise, thus $\log \left|\det \frac{\partial g}{\partial z}\right|$ is 0. Therefore, we can append a Revert Layer after each ConvBlock to accommodate warping for dimensions with larger indices without additional computation cost for the Jacobian as follows

$$z \to \underbrace{\text{ConvBlock} \to \text{Revert} \to \text{ConvBlock} \to \text{Revert} \to ... \to}_{\text{Repetitions of ConvBlock+Revert for K times}} f(z) \tag{28}$$

3.3.3 *Connection to Inverse Autoregressive Flow*

Inspired by the idea of constructing complex tractable densities from simpler ones with bijective transformations, different variants of the original normalizing flow (NF) (Rezende and Mohamed, 2015) have been proposed. Perhaps the one most related to ConvFlow is Inverse Autoregressive Flow (Kingma et al., 2016), which employs autoregressive transformations over the input dimensions to construct output densities. Specifically, one layer of IAF works as follows

$$f(z) = \mu(z) + \sigma(z) \odot z \tag{29}$$

where

$$[\mu(z), \sigma(z)] \leftarrow \text{AutoregressiveNN}(z) \tag{30}$$

are outputs from an autoregressive neural network over the dimensions of z. There are two drawbacks of IAF compared to the proposed ConvFlow:

- The autoregressive neural network over input dimensions in IAF is represented by a Masked Autoencoder (Germain et al., 2015), which generally requires $\mathcal{O}(d^2)$ parameters per layer, where d is the input dimension, while each layer of ConvFlow is much more parameter efficient, only needing $k + d$ parameters (k is the kernel size of 1d convolution and $k < d$).
- More importantly, due to the coupling of $\sigma(z)$ and z in the IAF transformation, in order to make the computation of the overall Jacobian determinant $\det \frac{\partial f}{\partial z}$ linear in d, the Jacobian of the autoregressive NN transformation is assumed to be *strictly* triangular (Equivalently, the Jacobian determinants of μ and σ w.r.t z are both always 0. This is achieved by letting the ith dimension of μ and σ depend only on dimensions $1, 2, ..., i-1$ of z). In other words, *the mappings from z onto $\mu(z)$ and $\sigma(z)$ via the autoregressive NN are always singular, no matter how their parameters are updated, and because of this, μ and σ will only be able to cover a subspace of the input space z belongs to*, which is obviously less

desirable for a normalizing flow.[2] Though these singularity transforms in the autoregressive NN are somewhat mitigated by their final coupling with the input z, IAF still performs slightly worse in empirical evaluations than ConvFlow as no singular transform is involved in ConvFlow.

- Lastly, despite the similar nature of modeling variable dimension with an autoregressive manner, ConvFlow is much more efficient since the computation of the flow weights w and the input z is carried out by fast native 1-d convolutions, where IAF in its simplest form needs to maintain a masked feed forward network (if not maintaining an RNN). Similar idea of using convolution operators for efficient modeling of data dimensions is also adopted by PixelCNN (Oord et al., 2016a).

3.4 EXPERIMENTS

We test performance the proposed ConvFlow on two settings, one on synthetic data to infer unnormalized target density and the other on density estimation for hand written digits and characters.

3.4.1 *Synthetic data*

We conduct experiments on using the proposed ConvFlow to approximate an unnormalized target density of z with dimension 2 such that $p(z) \propto \exp(-U(z))$. We adopt the same set of energy functions $U(z)$ in (Rezende and Mohamed, 2015) for a fair comparison, which is reproduced below

$$U_1(z) = \frac{1}{2}\left(\frac{\|z\| - 2}{4}\right)^2 - \log\left(e^{-\frac{1}{2}\left[\frac{z_1-2}{0.6}\right]^2} + e^{-\frac{1}{2}\left[\frac{z_1+2}{0.6}\right]^2}\right)$$

$$U_2(z) = \frac{1}{2}\left[\frac{z_2 - w_1(z)}{0.4}\right]^2$$

where $w_1(z) = \sin\left(\frac{\pi z_1}{2}\right)$ r. The target density of z are plotted as the left most column in Figure 6, and we test to see if the proposed ConvFlow can transform a

2 Since the singular transformations will only lead to subspace coverage of the resulting variable μ and σ, one could try to alleviate the subspace issue by modifying IAF to set both μ and σ as free parameters to be learned, the resulting normalizing flow of which is exactly a version of planar flow as proposed in (Rezende and Mohamed, 2015).

two dimensional standard Gaussian to the target density by minimizing the KL divergence

$$\begin{aligned} &\mathrm{KL}(q_K(z_k)||p(z)) = \mathbb{E}_{z_k} \log q_K(z_k)) - \mathbb{E}_{z_k} \log p(z_k) \\ =&\mathbb{E}_{z_0} \log q_0(z_0)) - \mathbb{E}_{z_0} \log \left| \det \frac{\partial f}{\partial z_0} \right| + \mathbb{E}_{z_0} U(f(z_0)) + \text{const} \end{aligned} \tag{31}$$

where all expectations are evaluated with samples taken from $q_0(z_0)$. We use a 2-d standard Gaussian as $q_0(z_0)$ and we test different number of ConvBlocks stacked together in this task. Each ConvBlock in this case consists a ConvFlow layer with kernel size 2, dilation 1 and followed by another ConvFlow layer with kernel size 2, dilation 2. Revert Layer is appended after each ConvBlock, and tanh activation function is adopted by ConvFlow. The Autoregressive NN in IAF is implemented as a two layer masked fully connected neural network (Germain et al., 2015).

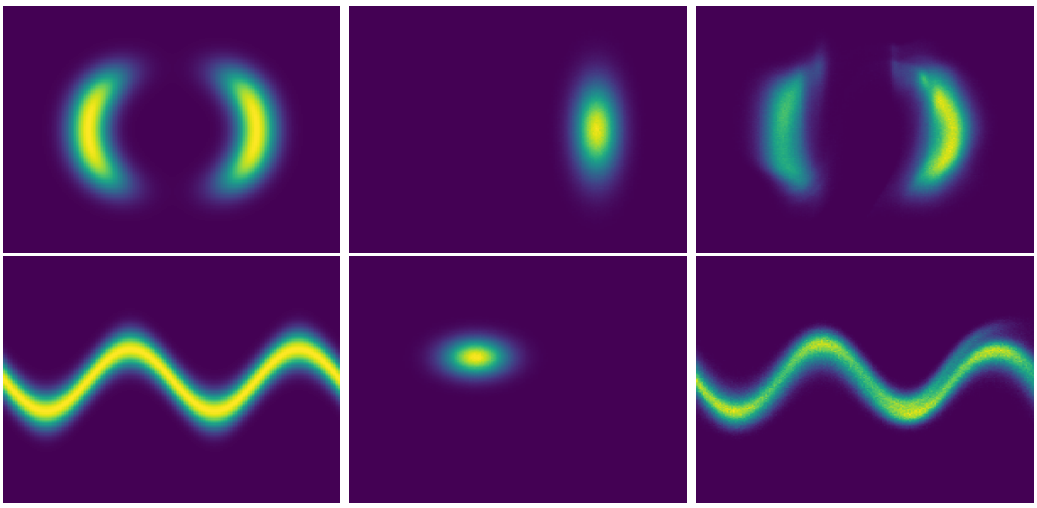


Figure 6: (a) True density; (b) Density learned by IAF (16 layers); (c) Density learned by ConvFlow. (8 blocks with each block consisting of 2 layers)

Experimental results are shown in Figure 6 for IAF (middle column) and ConvFlow (right column) to approximate the target density (left column). Even with 16 layers, IAF puts most of the density to one mode, confirming our analysis about the singular transform problem in IAF: As the data dimension is only two, the subspace modeled by $\mu(z)$ and $\sigma(z)$ in Eq. (29) will be lying on a 1-d space, i.e., a straight line, which is shown in the middle column. The effect of singular transform on IAF will be less severe for higher dimensions. While with 8 layers of ConvBlocks (each block consists of 2 1d convolution layers), ConvFlow is already

approximating the target density quite well despite the minor underestimate about the density around the boundaries.

3.4.2 *Handwritten digits and characters*

3.4.2.1 *Setups*

To test the proposed ConvFlow for variational inference we use standard benchmark datasets MNIST[3] and OMNIGLOT[4] (Lake et al., 2013). Our method is general and can be applied to any formulation of the generative model $p_\theta(x, z)$; For simplicity and fair comparison, in this paper, we focus on densities defined by stochastic neural networks, i.e., a broad family of flexible probabilistic generative models with its parameters defined by neural networks. Specifically, we consider the following two family of generative models

$$G_1 : p_\theta(x, z) = p_\theta(z)p_\theta(x|z) \tag{32}$$

$$G_2 : p_\theta(x, z_1, z_2) = p_\theta(z_1)p_\theta(z_2|z_1)p_\theta(x|z_2) \tag{33}$$

where $p(z)$ and $p(z_1)$ are the priors defined over z and z_1 for G_1 and G_2, respectively. All other conditional densities are specified with their parameters θ defined by neural networks, therefore ending up with two stochastic neural networks. This network could have any number of layers, however in this paper, we focus on the ones which only have one and two stochastic layers, i.e., G_1 and G_2, to conduct a fair comparison with previous methods on similar network architectures, such as VAE, IWAE and Normalizing Flows.

We use the same network architectures for both G_1 and G_2 as in (Burda et al., 2015), specifically shown as follows

G_1 : A single Gaussian stochastic layer z with 50 units. In between the latent variable z and observation x there are two deterministic layers, each with 200 units;

G_2 : Two Gaussian stochastic layers z_1 and z_2 with 50 and 100 units, respectively. Two deterministic layers with 200 units connect the observation x and latent variable z_2, and two deterministic layers with 100 units are in between z_2 and z_1.

where a Gaussian stochastic layer consists of two fully connected linear layers, with one outputting the mean and the other outputting the logarithm of diagonal

3 Data downloaded from http://www.cs.toronto.edu/~larocheh/public/datasets/binarized_mnist/

4 Data downloaded from https://github.com/yburda/iwae/raw/master/datasets/OMNIGLOT/chardata.mat

covariance. All other deterministic layers are fully connected with tanh nonlinearity. Bernoulli observation models are assumed for both MNIST and OMNIGLOT. For MNIST, we employ the static binarization strategy as in (Larochelle and Murray, 2011) while dynamic binarization is employed for OMNIGLOT.

The inference networks $q(z|x)$ for G_1 and G_2 have similar architectures to the generative models, with details in (Burda et al., 2015). ConvFlow is hence used to warp the output of the inference network $q(z|x)$, assumed be to Gaussian conditioned on the input x, to match complex true posteriors. Our baseline models include VAE (Kingma and Welling, 2013), IWAE (Burda et al., 2015) and Normalizing Flows (Rezende and Mohamed, 2015). Since our propose method involves adding more layers to the inference network, we also include another enhanced version of VAE with more deterministic layers added to its inference network, which we term as VAE+.[5] With the same VAE architectures, we also test the abilities of constructing complex variational posteriors with IAF and ConvFlow, respectively. All models are implemented in PyTorch. Parameters of both the variational distribution and the generative distribution of all models are optimized with Adam (Kingma and Ba, 2014) for 2000 epochs, with a fixed learning rate of 0.0005, exponential decay rates for the 1st and 2nd moments at 0.9 and 0.999, respectively. Batch normalization (Ioffe and Szegedy, 2015) and linear annealing of the KL divergence term between the variational posterior and the prior is employed for the first 200 epochs, as it has been shown to help training multi-layer stochastic neural networks (Sønderby et al., 2016). Code to reproduce all reported results will be made publicly available.

For inference models with latent variable z of 50 dimensions, a ConvBlock consists of following ConvFlow layers

$$
\begin{aligned}
&[\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 1),\\
&\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 2),\\
&\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 4),\\
&\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 8),\\
&\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 16),\\
&\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 32)]
\end{aligned}
\tag{34}
$$

5 VAE+ adds more layers before the stochastic layer of the inference network while the proposed method is add convolutional flow layers after the stochastic layer.

and for inference models with latent variable z of 100 dimensions, a ConvBlock consists of following ConvFlow layers

$$\begin{aligned}
&[\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 1),\\
&\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 2),\\
&\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 4),\\
&\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 8),\\
&\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 16),\\
&\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 32),\\
&\text{ConvFlow}(\text{kernel size} = 5, \text{dilation} = 64)]
\end{aligned} \tag{35}$$

A Revert layer is appended after each ConvBlock and leaky ReLU with a negative slope of 0.01 is used as the activation function in ConvFlow. For IAF, the autoregressive neural network is implemented as a two layer masked fully connected neural network.

3.4.2.2 *Generative Density Estimation*

For MNIST, models are trained and tuned on the 60,000 training and validation images, and estimated log-likelihood on the test set with 128 importance weighted samples are reported. Table 2 presents the performance of all models, when the generative model is assumed to be from both G_1 and G_2.

Firstly, VAE+ achieves higher log-likelihood estimates than vanilla VAE due to the added more layers in the inference network, implying that a better posterior approximation is learned (which is still assumed to be a Gaussian). Second, we observe that VAE with ConvFlow achieves much better density estimates than VAE+, which confirms our expectation that warping the variational distribution with convolutional flows enforces the resulting variational posterior to match the true non-Gaussian posterior. Also, adding more blocks of convolutional flows to the network makes the variational posterior further close to the true posterior. We also observe that VAE with Inverse Autoregressive Flows (VAE+IAF) improves over VAE and VAE+, due to its modeling of complex densities, however the improvements are not as significant as ConvFlow. The limited improvement might be explained by our analysis on the singular transformation and subspace issue in IAF. Lastly, combining convolutional normalizing flows with multiple importance weighted samples, as shown in last row of Table 2, further improvement on the test set log-likelihood is achieved. Overall, the method combining ConvFlow and importance weighted samples achieves best NLL on both settings, outperforming IWAE significantly by 7.1 nats on G_1 and 5.7 nats on G_2. Notice that, ConvFlow combined with IWAE achieves an NLL of 79.11, comparable to the best published

result of 79.10, achieved by PixelRNN (Oord et al., 2016a) with a much more sophisticated architecture. Also it's about 0.8 nat better than the best IAF result of 79.88 reported in (Kingma et al., 2016), which demonstrates the representative power of ConvFlow compared to IAF[6].

Table 2: MNIST test set NLL with generative models G_1 and G_2 (lower is better K is number of ConvBlocks)

MNIST (static binarization)	$-\log p(x)$ on G_1	$-\log p(x)$ on G_2
VAE (Burda et al., 2015)	88.37	85.66
IWAE ($IW = 50$) (Burda et al., 2015)	86.90	84.26
VAE+NF (Rezende and Mohamed, 2015)	-	$\leqslant$ 85.10
VAE+ ($K = 1$)	88.20	85.41
VAE+ ($K = 4$)	88.08	85.26
VAE+ ($K = 8$)	87.98	85.16
VAE+IAF ($K = 1$)	87.70	85.03
VAE+IAF ($K = 2$)	87.30	84.74
VAE+IAF ($K = 4$)	87.02	84.55
VAE+IAF ($K = 8$)	86.62	84.26
VAE+ConvFlow ($K = 1$)	86.91	85.45
VAE+ConvFlow ($K = 2$)	86.40	85.37
VAE+ConvFlow ($K = 4$)	84.78	81.64
VAE+ConvFlow ($K = 8$)	83.89	81.21
IWAE+ConvFlow ($K = 8, IW = 50$)	79.78	79.11

Results on OMNIGLOT are presented in Table 3 where similar trends can be observed as on MNIST. One observation different from MNIST is that, the gain from IWAE+ConvFlow over IWAE is not as large as it is on MNIST, which could be explained by the fact that OMNIGLOT is a more difficult set compared to MNIST,

6 The result in (Kingma et al., 2016) are not directly comparable, as their results are achieved with a much more sophisticated VAE architecture and a much higher dimension of latent code ($d = 1920$ for the best NLL of 79.88). However, in this paper, we only assume a relatively simple VAE architecture compose of fully connected layers and the dimension of latent codes to be relatively low, 50 or 100, depending on the generative model in VAE. One could expect the performance of ConvFlow to improve even further if similar complex VAE architecture and higher dimension of latent codes are used.

Table 3: OMNIGLOT test set NLL with generative models G_1 and G_2 (lower is better, K is number of ConvBlocks)

OMNIGLOT	$-\log p(x)$ on G_1	$-\log p(x)$ on G_2
VAE (Burda et al., 2015)	108.22	106.09
IWAE ($IW = 50$) (Burda et al., 2015)	106.08	104.14
VAE+ ($K = 1$)	108.30	106.30
VAE+ ($K = 4$)	108.31	106.48
VAE+ ($K = 8$)	108.31	106.05
VAE+IAF ($K = 1$)	107.31	105.78
VAE+IAF ($K = 2$)	106.93	105.34
VAE+IAF ($K = 4$)	106.69	105.56
VAE+IAF ($K = 8$)	106.33	105.00
VAE+ConvFlow ($K = 1$)	106.42	105.33
VAE+ConvFlow ($K = 2$)	106.08	104.85
VAE+ConvFlow ($K = 4$)	105.21	104.30
VAE+ConvFlow ($K = 8$)	104.86	103.49
IWAE+ConvFlow ($K = 8, IW = 50$)	104.21	103.02

as there are 1600 different types of symbols in the dataset with roughly 20 samples per type. Again on OMNIGLOT we observe IAF with VAE improves over VAE and VAE+, while doesn't perform as well as ConvFlow.

3.4.2.3 *Latent Code Visualization*

We visualize the inferred latent codes z of 5000 digits in the MNIST test set with respect to their true class labels in Figure 7 from different models with tSNE (Maaten and Hinton, 2008). We observe that on generative model G_2, all three models are able to infer latent codes of the digits consistent with their true classes. However, VAE and VAE+IAF both show disconnected cluster of latent codes from the same class (e.g., digits 0 and digits 1). Latent codes inferred by VAE for digit 3 and 5 tend to mix with each other. Overall, VAE equipped with ConvFlow produces clear separable latent codes for different classes while also maintaining high in-class density (notably for digit classes 0, 1, 2, 7, 8, 9 as shown in the rightmost figure).

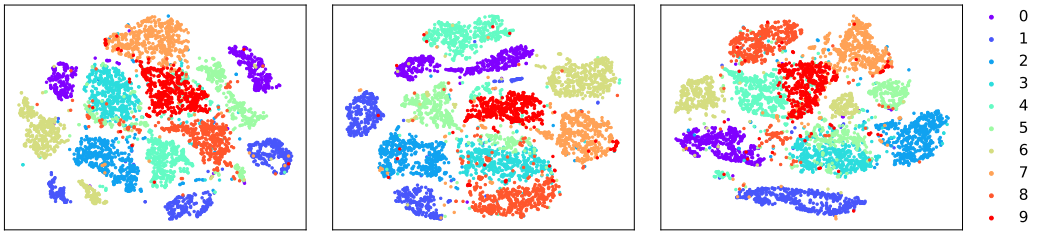
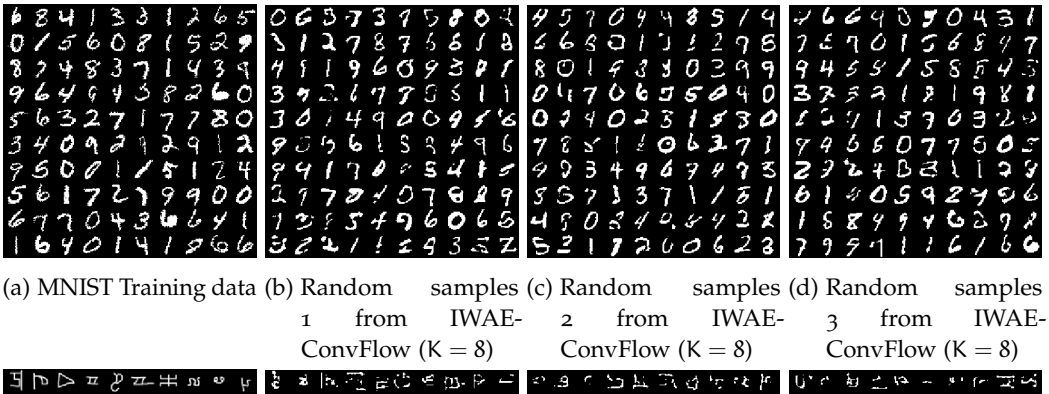


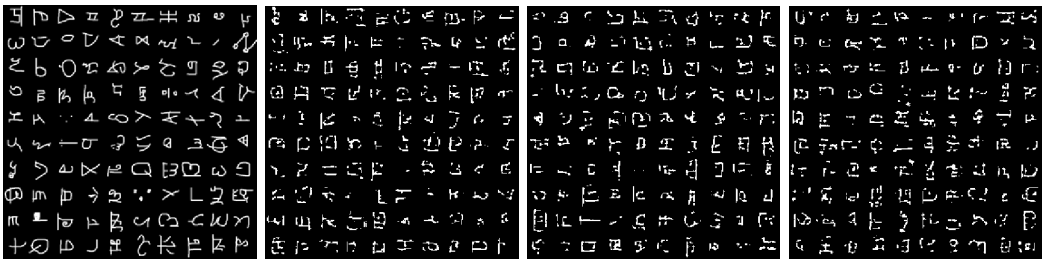
Figure 7: **Left:** VAE, **Middle:** VAE+IAF, **Right:**VAE+ConvFlow. (best viewed in color)

3.4.2.4 *Generation*

After the models are trained, generative samples can be obtained by feeding $z \sim N(0, I)$ to the learned generative model G_1 (or $z_2 \sim N(0, I)$ to G_2). Since higher log-likelihood estimates are obtained on G_2, Figure 8 shows three sets of random generative samples from our proposed method trained with G_2 on both MNIST and OMNIGLOT, compared to real samples from the training sets. We observe the generated samples are visually consistent with the training data.



(a) MNIST Training data (b) Random samples 1 from IWAE-ConvFlow (K = 8) (c) Random samples 2 from IWAE-ConvFlow (K = 8) (d) Random samples 3 from IWAE-ConvFlow (K = 8)



(e) OMNIGLOT Training data (f) Random samples from IWAE-ConvFlow (K = 8) (g) Random samples from IWAE-ConvFlow (K = 8) (h) Random samples from IWAE-ConvFlow (K = 8)

Figure 8: Training data and generated samples

3.5 CONCLUSIONS

This paper presents a simple and yet effective architecture to compose normalizing flows based on 1d convolution on the input vectors. ConvFlow takes advantage of the effective computation of convolution to warp a simple density to the possibly complex target density, as well as maintaining as few parameters as possible. To further accommodate long range interactions among the dimensions, dilated convolution is incorporated to the framework without increasing model computational complexity. A Revert Layer is used to maximize the opportunity that all dimensions get as much warping as possible. Experimental results on inferring target complex density and density estimation on generative modeling on real world handwritten digits data demonstrates the strong performance of ConvFlow. Particularly, density estimates on MNIST show significant improvements over state-of-the-art methods, validating the power of ConvFlow in warping multivariate densities. It remains an interesting question to see how ConvFlows can be directly combined with powerful observation models such as PixelRNN to further advance generative modeling with tractable density evaluation. We hope to address these challenges in future work.

3.6 CONDITIONS FOR INVERTIBILITY

The ConvFlow proposed in Section 3.3 is invertible, as long as every term in the main diagonal of the Jacobian specified in Eq. (25) is non-zero, i.e., for all $i = 1, 2, ..., d$,

$$w_1 u_i h'(\text{conv}(z, w)) + 1 \neq 0 \tag{36}$$

where u_i is the i-th entry of the scaling vector $\mathbf{u}$. When using $h(x) = \tanh(x)$, since $h'(x) = 1 - \tanh^2(x) \in [0, 1]$, a sufficient condition for invertibility is to ensure $w_1 u_i > -1$. Thus a new scaling vector $\mathbf{u}'$ can be created from free parameter $\mathbf{u}$ to satisfy the condition as

$$\mathbf{u}' = \begin{cases} \mathbf{u} & \text{if } w_1 = 0 \\ -\frac{1}{w_1} + \text{softplus}(\mathbf{u}) & \text{if } w_1 > 0 \\ -\frac{1}{w_1} - \text{softplus}(\mathbf{u}) & \text{if } w_1 < 0 \end{cases} \tag{37}$$

where $\text{softplus}(x) = \log(1 + \exp(x))$. The above sufficient condition works readily for other non-linearity functions h , including sigmoid, softplus, rectifier(ReLU), leaky rectifier (Leaky ReLU) and exponential linear unit (ELU), as all their gradients are bounded in $[0, 1]$.

4

NEURAL GENERATIVE PERMUTATION LEARNING

Permutations and matchings are fundamental building blocks for many machine learning applications, as they can be used to align, organize, match, and sort data. A permutation shuffles the indices of a data point to form another data point and permutation learning aims to recover this unknown mapping. Supervised permutation learning, where the pairs of (original object, permuted object) are available , has been extensively studied in the literature and effective end-to-end approaches based on deep neural networks have been proposed to tackle the problem. Prior work for supervised permutation learning includes (Caetano et al., 2009; Petterson et al., 2009; Tang et al., 2015). Meanwhile, there's little to no exploration for the unpaired permutation learning setting and unsupervised permutation setting for permutation learning.

In this chapter, we first study the problem of *unpaired* permutation learning, i.e., only samples of original objects and that of permuted objects are observed while no paired link between the two is given. We propose to tackle the unpaired permutation learning under the adversarial training framework; specifically, a permutation generative network is trained to generate aproximated permutations conditioned on the permuted object, by which the permuted object can be transformed back to the original space, and a discriminative network is trained to disdinguish real objects from the original space and the recovered ones. Empirical experiments on sorting numbers and recovering scrambled images demonstrates the effectiveness of the proposed method.

Next we address the problem of unsupervised generative permutation learning, where we only observe the data samples and the permutation information is hidden. We treat the permutations as latent variables which govern the generation process of the data samples. We propose an efficient way to construct approximate permutation matrices and also tractable density functions over them. With the permutation variable treated as latent variables, a framework to maximize the data log-likelihood is proposed. We conduct experiments on image generation, where both the permutation and the image generators are learned. Comparable results

in terms of data likelihood are obtained compared to state-of-the-art probabalistic gerative models for images.

4.1 PERMUTATION LEARNING PRELIMINARIES

4.1.1 *What is a permutation*

A permutation matrix is a square integer matrix $\mathbf{P} \in \{0,1\}^{N \times N}$, satisfying all rows summed up to 1 and all columns summed upto 1. In other words, each row (or column) of $\mathbf{P}$ is one of the N basis vectors for $\mathbb{R}^N$. By applying this permutation matrix to a N-dimensional vector is equivalent to shuffling the indices of that vector. For example, consider the following 3×3 permutation matrix and a 3-d data vector $\mathbf{x}$, applying the permutation to the vector yields

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ x_1 \end{bmatrix} \tag{38}$$

A closely related concept to permutation matrix is the so called doubly-stochastic matrix or bistochastic matrix (BSM) $\mathbf{B} \in [0,1]^{N \times N}$, also satisfying row summed to 1 and column summed to 1. The difference between a BSM and a permutation matrix is that the entires in the BSM can be values in the range $[0,1]$. For example, consider the following BSM applying to the same example data vector above,

$$\begin{bmatrix} 0.2 & 0.8 & 0 \\ 0 & 0.1 & 0.9 \\ 0.8 & 0.1 & 0.1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.2x_1 + 0.8x_2 \\ 0.1x_2 + 0.9x_3 \\ 0.8x_1 + 0.1x_2 + 0.1x_3 \end{bmatrix} \tag{39}$$

The output vector is no longer a shuffled version of the input; instead, each entry in the output vector is a weighted average of all the entries from the input. In fact, the Birkhoff–von Neumann theorem states that family of BSMs, denoted by $\mathcal{B}_N$, is a convex polytope called Birkhoff polytope, with all the permutation matrices as its vertices, i.e.

$$\mathcal{P}_N \subset \mathcal{B}_N \tag{40}$$

Because of the discrete nature of the permutation matrices, directly solving for permutations is a complex combinatorial optimization problem where gradient based methods cannot be applied, thus the BSMs comes in handy to approximate permutation matrices.

4.1.2 *Biostochastic Matrix Construction: Sinkhorn Operator*

It has been shown that the Sinkhorn perator (Sinkhorn and Knopp, 1967) can be used to construct a BSM from any positive square matrix. Given a positive matrix $A \in \mathbb{R}_+^{N\times N}$, iterate

- Normalize each row to sum 1
- Normalize each column to sum 1

The Sinkhorn operator is ensured to converge to a bistochastic matrix, guaranteed by the following theorem:

Lemma 3 *(Sinkhorn and Knopp, 1967) If A is an $n \times n$ matrix with strictly positive elements, then there exist diagonal matrices D_1 and D_2 with strictly positive diagonal elements such that D_1AD_2 is doubly stochastic. The matrices D_1 and D_2 are unique modulo multiplying the first matrix by a positive number and dividing the second one by the same number.*

It's worth noting that the Sinkhorn operator is a continuous operation and thus gradient can be propagated back if the Sinkhorn operator is plugged in as a part of a computation graph. In practice, often k, e.g., $k = 10, 20$, iterations of the Sinkhorn operator is performed, which in the language of deep neraul networks is equivalent to stacking k layers of row-sum normalization and column-sum normalization.

4.1.3 *Gumbel-Sinkhorn Networks*

The Sinkhorn operator is guaranteed to output a bistochastic matrix, hence it's natural to wonder if there is a procedure to construct an exact permutation matrix. A recent research by (Mena et al., 2018) proposes the Gumbel-Sinkhorn Networks: For a square matrix A, a (hard) permutation matrix can be obtained asymptotically by solving

$$M(A) = \arg\max_{P\in\mathcal{P}_N} \langle P, A\rangle_F = \arg\max_{P\in\mathcal{P}_N} \operatorname{trace}(P^T A) \tag{41}$$

Gumbel-Sinkhorn operator with temperature parameter $\tau > 0$

$$S^0(A/\tau) = \exp(A/\tau) \tag{42}$$

$$S^l(A/\tau) = \mathcal{T}_c(\mathcal{T}_r(S^{l-1}(A/\tau))) \tag{43}$$

$$S(A/\tau) = \lim_{l\to\infty} S^l(A/\tau) \tag{44}$$

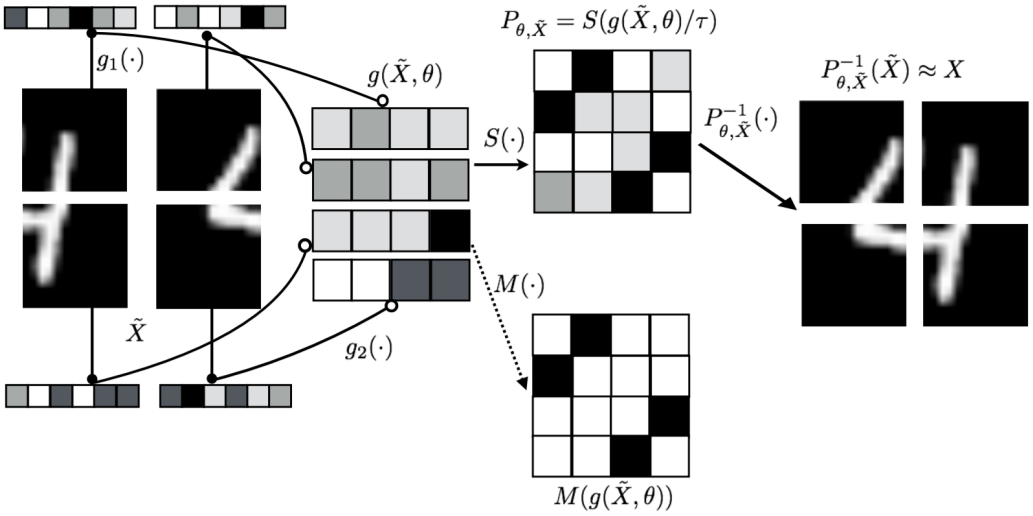


Figure 9: Example of Gumbel-Sinkhorn operator used to learn to recover shuffled image patches. (reproduced from (Mena et al., 2018))

Lemma 4 *(Mena et al., 2018) For a bistochastic matrix B, define its entropy as* $h(B) = -\sum_{i,j} B_{i,j} \log B_{i,j}$. *Then one has*

$$S(A/\tau) = \arg\max_{B \in \mathcal{B}_N} \langle B, A \rangle_F + \tau h(B)$$

And under mild conditions,

$$M(A) = \lim_{\tau \to 0^+} S(A/\tau)$$

The Gumbel-Sinkhorn operator ensures convergence to permutation matrices when $\tau \to 0^+$. It is essentially solving a relaxed, and entropy regularized version of Eq. 41, where the solution becomes a hard permutation matrix when the termperature hyper-parameter $\tau \to 0$, because any biostochastic matrix has to be a permutation matrix if its entropy is 0. To let the solution approximate closer to a permutation matrix, one might attempp to set τ to a very small value. However, note that there is a tradeoff between a higher temperature and lower one, as a larger τ will ensure the feasible space more smooth thus gradient based optimization techniques can be used but the solution will be less closer to a permutation matrix and a smaller τ will make the problem almost indifferentiable at every B.

4.1.4 *Supervised permutation laerning*

The problem of supervised (visual) permutation learning can be defined in two differnt manners:

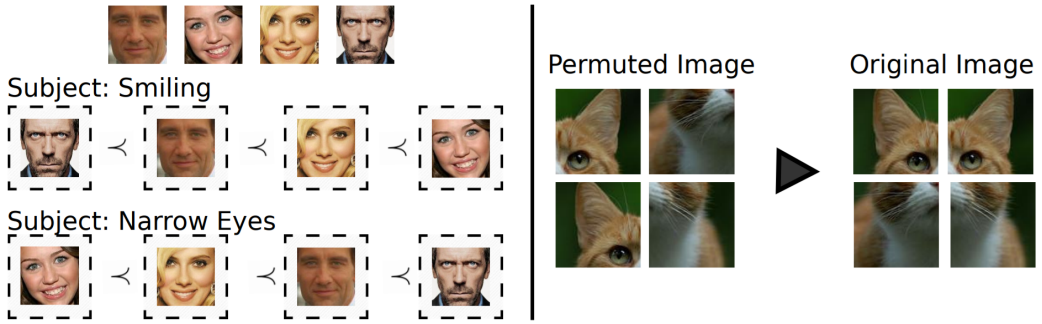


Figure 10: Example of supervised visual permutation learning (reproduced from (Cruz et al., 2017))

- **Supervision on permuted images**: Given pairs of original image X and permuted image $\tilde{X}$, we can formulate to solve for the permutation such that the residuar error in recovering the original image after applying the permutation is minimized

$$\min_{\theta} \sum_{i=1}^{M} \|X_i - P_{\theta,\tilde{X}_i} \tilde{X}_i\|^2 \tag{45}$$

  Since P is difficult to parameterize and optimize directly, we relax it to the family of BSMs

$$\min_{\theta} \sum_{i=1}^{M} \|X_i - B_{\theta,\tilde{X}_i} \tilde{X}_i\|^2 \tag{46}$$

- **Supervision on permutaion matrix itself**: If we have knowledge of how each data point are permuted, i.e., the groundtruth of the permutation matrix, we can formulate the supervised permutation learning problem as

$$\min_{\theta} \sum_{i=1}^{M} \|P_i - B_{\theta,\tilde{X}_i}\| \tag{47}$$

4.2 UNPAIRED PERMUTATION LEARNING WITH ADVERSARIAL NETS

We are now ready to investigate the problem of *unpaired* permutation learning, where only samples of original objects and that of permuted objects are observed while no paired link betwen the two is present. This is intrinsically a harder task

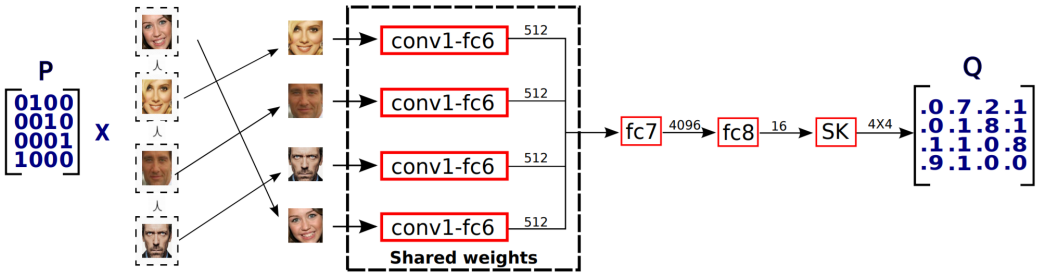


Figure 2. DeepPermNet Architecture. It receives a permuted sequence of images as input. Each image in the sequence goes trough a different branch that follows the AlexNet [24] architecture from *conv1* up to *fc6*. Then, the outputs of *fc6* are concatenated and passed as input to *fc7*. Finally, the model predictions are obtained by applying the Sinkhorn Layer on the outputs of *fc8* layer.

Figure 11: The DeepPermNet (Cruz et al., 2017)

compared to supervised permutation learning and existing methods on paired permutation learning cannot be direclty applied. We propose to address this problem under the adversarial training framework. Specifically, a permutation generative network and a discriminative network are jointly trained. The permutation generative newtork is trained to generate approximated permutations conditioned on the permuted object, such that after the generated permutation, the permuted object can be transformed back to the original space; while the discriminative newtork is trained to tell apart the real objects from the original space and the recovered ones. The two networks are adversarially trained.

4.2.1 *Unpaired Permutation Learning with Adversarial Net*

Due to the nature of unpaired info between permuted input and original input, it's infeasible to construct a permutation and measure the reconstruction loss per indivisual sample. Instead, we address this problem with the idae of adversarial training. Specifically, two networks are trained to achieve this goal: a permutation generative network is trained by taking a permuted objbect as input and emiting an approximated permutation, hence a reconstruction of the permuted object can be obatined and a discriminative network is trained to distinguish the reconstructed sample and original sample (note that they might not necessarily be about the same object as they are unpaired). The two networks are trained adversarially to achieve an equillibrium. We term the proposed architecture UPLAN and an illustration of the architecture can be referred to at Figure 12.

UPLAN consists of two network, a BSM generator and a discriminator . The BSM generator is designed to extract features from a shuffled image, to construct a input dependent BSM via the Sinkhorn operator and to re-permute the input image. Specifically, the BSM generator G_θ consists of two components:

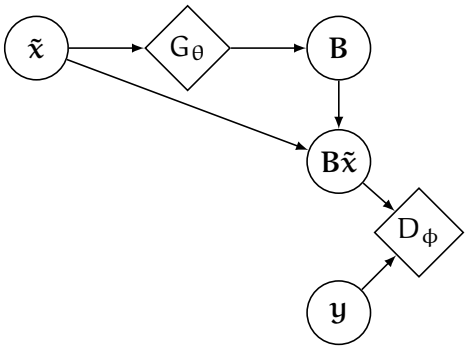


Figure 12: Unpaired permutation learning with adversarial nets ($\tilde{\mathbf{x}}$ is one shuffled image, $\mathbf{y}$ is another original image)

- Feature extractor, outputs a $d \times d$ nonegative matrix A
- Sinkhorn Network, normalize A onto a BSM B

The discriminator D_ϕ is designed to tell apart real images from reconstructed images, which is essentially a binary image classifier with the real images labeled as 1 and the reconstructed images labeled as -1.

The optimization objective of UPLAN is the following minimax problem

$$\min_\theta \max_\phi \mathbb{E}_{\tilde{\mathbf{x}}} \log(1 - D_\phi(G_\theta(\tilde{\mathbf{x}})\tilde{\mathbf{x}})) + \mathbb{E}_{\mathbf{y}} \log D_\phi(\mathbf{y}) \tag{48}$$

Analogous to standard GAN reasoning, given infinite model capacities for both G_θ and D_ϕ, UPLAN is able to recover the BSM used to shuffle the original real images.

4.2.2 *Experiments*

We evaluate UPLAN for its ability to recover permutations on two settings: sorting numbers and solving jagsaw problems. Quantitative comparison against supervised permutation learning methods is provided.

4.2.2.1 *Sorting numbers*

Similar to the supervised settings in (Mena et al., 2018), we sample two sets of random vectors with dimension N, where each entry is uniformly sampled from the $[0, 1]$ interval, $\mathbf{x}$ and $\mathbf{y}$. we only sort the entries of $\mathbf{y}$ and keep $\mathbf{x}$ as it is. This ensures $\mathbf{x}$ is the shuffled version of some sorted vector $\mathbf{x}_0$ which the model doesn't have access to. We train our network with these two sets of vectors and evaluate if the model is able to learn to sort $\mathbf{x}$ to $\mathbf{x}_0$ based on another sorted vector $\mathbf{y}$.

Following the setting in (Mena et al., 2018), the BSM generator in UPLAN network has two fully connected layers with 32 hidden units and the discriminator is also implemented as a two-layer fully connected layers with 32 hidden units with a sigmoid output non-linearity for binary classification. We report the proportion of sequences where there was at least one error entry in the sorted sequence as the evaluation measure. Table 4 presents the performance of sorting N numbers compared to existing work. We observe that with the more difficult unpaired

	N = 5	N = 10	N = 15	N = 80	N = 100	N = 120
(Mena et al., 2018) - Gumbel-Sinkhorn	.0	.0	.0	.0	.0	.01
(Ours) - UPLAN	.0	.002	.005	.012	.015	.02

Table 4: Error rates on sorting different size of numbers (lower is better)

setting, there is little to no different in terms of sorting number compared to the supervised permutation learning method.

4.2.2.2 *Jigsaw puzzles*

Figure 13: (a) Scrambled images; (b) Real images; (c) Recovered images

Next we evaluate UPLAN on solving jigsaw puzzles of real words images, a more complex scenario for learning permutations to reconstruct an image from a collection of scrambled jigsaw pieces from it. We use the CelebA human face dataset (Liu et al., 2015) and chop each image to 4×4 patches. Both the BSM generator and the discriminator uses the same CNN architecture as DCGAN (Radford et al., 2015).

We use Kendall's tau to measure the correlation of recovered sequence of image patches to the true sequence of patches from un-scrambled image. UPLAN achieves a Kendall's tau of 0.73, while the supervised permutation state-of-the-art (Mena

et al., 2018) scored 0.88, which demonstrates that without the paired information of true images and scrambled images, UPLAN is still capable of doing a reasonable good job in recovering the correct ordering of image patches.

Figure 13 presents qualitative results, showing a batch of scrambled images, real images and recovered images. Note that images in Figure 13(a) are not scrambled from images in Figure 13(b), hence they are unpaired.

4.3 GENERATIVE VARIATIONAL PERMUTATION LEARNING

Compared to the unpaired setting, which we still have access to a set of permuted objects and a set of original objects. It's reasonble to ask the question of whether permutation underlying only one set of observed objects is still possible. Consider the problem of density estimation for data $\mathbf{x} \in \mathbb{R}^d$, where the joint data distribution can be decomposed in an autoregressive manner on its dimensions as

$$p(\mathbf{x}) = p(\mathbf{x}_1) \prod_{i=2}^{d} p(\mathbf{x}_i | \mathbf{x}_{<i}) \tag{49}$$

Specifically, PixelRNN (Oord et al., 2016a) is one of these models to learn the conditional density of pixel generation with a Recurrent Neural Network and achieves the strongest likelihood estimates so far. One implicit assumption made by PixelRNN is that image pixels are generated in a raster manner, i.e., pixels in the upper left corner of an image should be generated before pixels in the lower right. This essentially makes a strong assmption about the conditoinal dependencies among the pixels based their order. However, for any probabalistic model with finite model capacity, this parcitular order might not be optimal for every images. In this section, we try to address this aspect by treating the order of conditional dependencies of an image as a latent variable and models the density of an image based on it. Exact inference on this discrete variable turns out intractable, therefore we propose a continuous relaxation to it.

4.3.1 *Probablistic modeling with latent permutations*

As stated in previous sections, applying a permutation matrix $\mathbb{P}$ to a data vector $\mathbf{x}$ will permutate the dimensions accordingly

$$\log p(\mathbf{x}) = \log \sum_{\mathbb{P}} p(\mathbf{x}, \mathbb{P}) \tag{50}$$

Computing the summation over all possbile $\mathbb{P}$ is infeasible as the support size of $\mathbb{P}$ is $d!$. We resort to a continuous relaxation of the set of all permutation matrices, the

set of all doubly stochastic matrices $\mathbb{B}$, which is also known as Birkhoff polytope whose limit points are eactly the permutation matrices. Formally

$$\log p(\mathbf{x}) = \log \mathbb{E}_B \frac{p(\mathbf{x}, B)}{q(B|\mathbf{x})} \tag{51}$$

$$\geqslant \mathbb{E}_B \log p(B) + \mathbb{E}_B \log p(\mathbf{x}|B) - \mathbb{E}_B \log q(B|\mathbf{x}) \tag{52}$$

where $p(x|B)$ models the density with a given (soft) permutation B of the data $\mathbf{x}$, i.e., $p(\mathbf{x}|B) \equiv p(B\mathbf{x})$. Existing density estimates methods including PixelRNN can be plugged into $p(x|B)$, estimating the density of a softly permutated version of $B\mathbf{x}$, instead of $\mathbf{x}$ itself.

To facilitate the stochastic optimization over the expectations of B, we need to define a propoer probabalistic measure over the Birkhoff polytope, which also allows for efficient sampling and density evaluations.

4.3.2 *Construction of doubly stochastic matrices*

For any given square A matrix with non-negative entries, a doubly stochastic matrix B can be obtained from the following procecedure of alternatively normalizing its rows to of sum 1 and its columns to be of sum 1.

This procedure will take A as input and ultimately output a doubley stochastic matrix B. However this process is not invertible, as any multiples of A will lead to the same B. To devise an invertible construction of doubly stochastic matrix, we assume the colomn sum of A is of 1. Hence one iteration of the procedure can be written as

$$A^{(i+1)} = D_1 A^{(i)} D_2 \tag{53}$$

wherr $D_1 = \text{diag}(\text{rowsum}(A))$ and $D_2 = \text{diag}(\text{colsum}(D_1 A))$.

Assume A is column-normalized and B is the output of applying row normalization to A, i.e.,

$$B = [\text{diag}(\boldsymbol{\lambda})]^{-1} A \Leftrightarrow A = \text{diag}(\boldsymbol{\lambda}) B \tag{54}$$

where $\boldsymbol{\lambda}$ is the row sum vector of A. Since A is column-normalized, we also have $B^\top \boldsymbol{\lambda} = \mathbf{1}$, which gives $\boldsymbol{\lambda} = (B^\top)^{-1}\mathbf{1}$, meaning that $\boldsymbol{\lambda}$ is uniquely determined by the rows sum of $(B^\top)^{-1}$, hence column sums of B^{-1}. This suggests one step of row normalizing a column-normalized matrix is a bijective operation, with its Jacobian specified as $\det(J) = \prod_{i=1}^{d} \det(J_i)$ where J_i is the Jacobian of normalizing row i of A, each of which takes $\mathcal{O}(d^3)$ computation, resulting in an overall computation complexity of Jacobian determinant computation to $\mathcal{O}(d^4)$. However, we will show

that the determinant computation is not required in defining and learning the distributions over the bistochastic matrices.

Readers could refer to (Linderman et al., 2017) for constructing a doubly stochastic matrices with the stick breaking process, however in this chapter, the proposed construction is a lot simpler and more efficient. Most importantly, reparametering the Birkhoff polytope is not required.

4.3.3 *Efficient Computation for Determinant, Inverse of bistochastic matrices*

Given two bistochastic matrix A and B with size $n \times n$, a larger bistochastic matrix with size $2n \times 2n$ can be constructed via

$$S = \begin{bmatrix} aA & (1-a)A \\ (1-a)B & aB \end{bmatrix} \tag{55}$$

and $\det(S) = \det\Big((2a-1)AB\Big)$, or equivalently $\log|\det(S)| = n\log|2a-1| + \log|\det(A)| + \log|\det(B)|$. Futher, the inverse of S can be efficiently computed in a recursive way as

$$S^{-1} = \frac{1}{1-2a}\begin{bmatrix} -aA^{-1} & (1-a)B^{-1} \\ (1-a)A^{-1} & -aB^{-1} \end{bmatrix} \tag{56}$$

with the inverse of a 2×2 stochastic matrix computed as $\begin{bmatrix} a & 1-a \\ 1-a & a \end{bmatrix}^{-1} = \frac{1}{1-2a}\begin{bmatrix} -a & 1-a \\ 1-a & -a \end{bmatrix}$.

Suppose biostochastic matrices S_1 and S_2 are of size $m \times m$ and $n \times n$, respectively, and without loss of generality, $m \leqslant n$. we can construct a new bistochastic matrix S with size $(m+n) \times (m+n)$ as follows

$$S = a\begin{bmatrix} S_1 & \\ & S_2 \end{bmatrix} + (1-a)\begin{bmatrix} & S_1 \\ S_2 & \end{bmatrix} \tag{57}$$

and its determinant is

$$\det(S) = \det(A+B) = \det(A)\det(I_{m+n} + A^{-1}B) \tag{58}$$

$$= \det(A)\det\left(I_{m+n} + \frac{1-a}{a}\begin{bmatrix} S_1^{-1} & \\ & S_2^{-1} \end{bmatrix}\begin{bmatrix} & S_1 \\ S_2 & \end{bmatrix}\right) \tag{59}$$

$$= \det(A)\det\left(I_{m+n} + \frac{1-a}{a}\begin{bmatrix} & I_m \\ I_n & \end{bmatrix}\right) \tag{60}$$

$$= \det(S_1)\det(S_2)\prod_{j=0}^{m+n-1}\left(a + (1-a)\exp\left\{\frac{2jm\pi i}{m+n}\right\}\right) \tag{61}$$

$$= \det(S_1)\det(S_2)\prod_{j=0}^{m+n-1}\sqrt{a^2 + (1-a)^2 + 2a(1-a)\cos\left(\frac{2jm\pi}{m+n}\right)} \tag{62}$$

where we used the determinant of a circulant matrix. This gives $\mathcal{O}((m+n)\log(m+n))$ complexity if $m \neq n$ and $\mathcal{O}(m+n)$ if $m = n$.

The inverse of S can also be computed efficiently as

$$S^{-1}S = I_{m+n} \tag{63}$$

$$\Rightarrow \left(a\begin{bmatrix} S_1 & \\ & S_2 \end{bmatrix} + (1-a)\begin{bmatrix} & S_1 \\ S_2 & \end{bmatrix}\right)\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix} = I_{m+n} \tag{64}$$

$$\Rightarrow a\begin{bmatrix} S_1A_1 & S_1A_2 \\ S_2A_3 & S_2A_4 \end{bmatrix} + (1-a)\begin{bmatrix} S_1B_3 & S_1B_4 \\ S_2B_1 & S_2B_2 \end{bmatrix} = I_{m+n} \tag{65}$$

$$\Rightarrow a\begin{bmatrix} S_1A_1 & S_1A_2 \\ S_2A_3 & S_2A_4 \end{bmatrix} + (1-a)\begin{bmatrix} S_1B_3 & S_1B_4 \\ S_2B_1 & S_2B_2 \end{bmatrix} = I_{m+n} \tag{66}$$

4.3.4 *Learning global permutation for a dataset*

For a data set $\mathcal{X}$, we aim to find a fixed permutation approximation B by generative modeling, such that

$$\log p(\mathbf{x}) = \log p(B\mathbf{x}) + \log|\det B| \tag{67}$$

is maximized. In generation, a sample image can be constructed as $B^{-1}\tilde{\mathbf{x}}$, where $\tilde{\mathbf{x}}$ is a sample output from the autoregressive generator p.

4.3.5 *Distribution on doubly stochastic matrices*

Given a random matrix with row sums of 1, we can construct follow the alternatively normalizing procedure, which in the limit will return a doubly stochastic

matrix. It turns out that running the procedure for K steps will suffice to give a sufficiently accurate approximated doubly stochastic matrix. Thus we readily obtain a way to construct doubly stochastic matrices as well as evaluate density likelihoods of the resulting doubly stochastic matrix, since the above procedure is bijective given that the initial random matrix is of row 1. To this end, one way to construct such matrix is to sample a set of d random vectors from a logistic Gaussian (also for the ease for reparameterization). For both the prior and variational posterior of B, we set

$$\begin{aligned} p(\alpha_i) &= \text{LogisticGaussian}(0, I) \ \text{ for } i = 1, .., d \\ q(\alpha_i|x) &= \text{LogisticGaussian}(\mu(x), \Sigma(x)) \ \text{ for } i = 1, ..., d \end{aligned} \tag{68}$$

where the parameter of the variational posterior for A conditioned on x denotes the inference dependency on x, which can be implemented by a neural network.

Then the data likelihood equipped with dimension ordering turns to be

$$\begin{aligned} &\log p(x) \geqslant \mathbb{E}_B \log p(B) + \mathbb{E}_B \log p(x|B) - \mathbb{E}_B \log q(B|x) \\ =&\mathbb{E}_\alpha \log p(\alpha) + \mathbb{E}_\alpha \log p(x|f(\alpha)) - \mathbb{E}_\alpha \log q(\alpha|x) \\ =&\mathbb{E}_\alpha \log p(\alpha) - \mathbb{E}_\alpha \log q(\alpha|x) \\ &+ \mathbb{E}_\alpha \log p(f(\alpha)x|\alpha) + \mathbb{E}_\alpha \log |\det f(\alpha)| \\ =&\mathbb{E}_\alpha \log p(f(\alpha)x|\alpha) + \mathbb{E}_\alpha \log |\det f(\alpha)| \\ &- \mathsf{KL}(q(\alpha|x)||p(\alpha)) \end{aligned} \tag{69}$$

where f is the operator mapping vector α to a bistochastic matrix B as defined above, and $p(\cdot|\cdot)$ is a conditional generator, such as conditional PixelCNN where α can be treated as the permutation embeddings for data x.

4.3.6 *Experiments*

We conduct experiments on generative modeling for CIFAR-10 images. Specifically, we incorporate the permutation learning framework with the PixelCNN image generator to evaluate the effects of adding the components for permutation learning.

It can be seen that incorporating pixel-level permutation learning into PixelCNN is a diffcult task, in thes sense that in a fully unsupervised setting without any knowledge about the annotations of the images, any individual pixel doesn't carry much information about the content of the image. Modeling permutation on blocks of pixels, such as image regions would be a meaningful direction to purse, however that may require access to information about the regions in an image.

Model	Test set
(Oord et al., 2016a) - PixelCNN	2.996
PixelCNN with reversed ordering	2.992
PixelCNN (deterministic perm. learning)	3.656
PixelCNN (variational perm. learning)	5.628

Table 5: Negative log-likelihood on CIFAR-10 in bits/dim (lower is better)

4.4 CONCLUSIONS

In this chapter, we explore neural generative and probabilistic modeling of permutations, one of the key discrete structures for various machine learning tasks. To this end, we first propose to model and learn permutations with adversarial training for the unpaired setting; then for the unsupervised setting, we construct probabilistic models over permutations and propose to learn such latent permutations from the data in a fully unsupervised manner. Experiments for the unpaired setting demonstrate that learning permutations without supervision is possbile, however for the fully unsupervised settings, pixel-level permutation learning still encounters technical challenges.

5

NEURAL PROBABLISTIC LANGUAGE MODELING

The task of language modeling aims to address the question of how tokens in language sequences can be modeled in a principled way. It aims to learn the probability of language sequences, by modeling and how language tokens, such as words, can be generated based on its context . Specifically, given a piece of language text consisting T words, $w_1, w_2, ..., w_T$, a language model tries to model and maximize the joint density $p(w_1, w_2, ..., w_T)$. The way to approach the joint likelihood is to decompose the joint modeling problem into a series of sub-problems of estimating the conditional generative distribution of a single word given its context as

$$p(w_1, w_2, ..., w_T) = \prod_{i=1}^{T} p(w_i | w_{<i}) \tag{70}$$

where $w_{<i}$ represents the words preceding w_i. Classical language models, such as N-grams, applies the above strategy to estimate the conditionals $p(w_i|w_{<i})$ by counting the frequencies and computing the ratios of all sub-sequences of length n. N-grams are simple and easy to implement, however they are prone to overfit as it's simply memorizing the frequency ratios.

Recently, deep neural network based language models have attracted much attention, due to their ability to represent and learn complex conditional densities embedded in human languages. Significant improvement over perplexity has been achieved with state-of-the-art language models based on recurrent neural networks, by directly modeling the next word generative distribution $p(w_i|w_{<i})$ with RNN. For example, on the PTB benchmark data set, the AWD-LSTM achieves an impressively low perplexity of 68.6 on the validation set and 65.8 on the test set, respectively. Despite the great success of language modeling with complex RNN architectures, the underlying principle of language modeling, i.e. using multiple conditional densities of a single next token as a surrogate for joint density of the entire sequence, hasn't changed. We argue that with any finite model capacity, this is sub-optimal as the resulting model is only trying to predict the best **next word** given a context, rather than the best **sequence**. As greedily picking the most

probable words at each step with a language model does not necessarily grantee the most likely sequence. In fact, beam search is widely used in the sampling process from a trained language model, however the training process is not aware of such guidance to maximize the likelihood of completing the sequence.

In this chapter, we propose to address this problem from a fundamental perspective for language modeling , i.e., trying to model how the language sequence can be completed based on the context. In contrast to only modeling the next word generative distribution $p(w_{k+1}|w_1, w_w, ..., w_k)$ by previous approaches, this corresponds to learning and maximizing the conditional generative density of the remaining of the sequence

$$p(w_{k+1}, w_{k+2}, ..., w_T|w_1, w_2, ..., w_k) \tag{71}$$

where k ranges through $[0, T-1]$. The above formulation is explicitly emphasizing the predictive distribution of completing a whole sequence of words based on the context. By doing so, the model is guided to generate words such that it's more likely for them to compose a natural language sequence in the long run.

Specifically, we propose to model the conditional sequence generative distribution with two recurrent neural networks, with one to encode and digest the context $w_1, w_2, ..., w_k$ and the other to decode the context and generate the target sequence $w_{k+1}, w_{k+2}, ..., w_T$. We conduct empirical evaluations against strong state-of-the-art RNN language models on several benchmark datasets and our model significantly outperforms most of the baseline language models, achieving a test set perplexity of 57.0 on PTB and 64.8 on WikiText2, respectively.

The remaining of the chapter is organized as follows: we briefly highlight the essence of neural language modeling and indentify the connection of the proposed approach to neural machine translation in Section 5.1. We describe the proposed model and learning algorithm in Section 5.2. Experiments and analysis against state-of-the-art language models are presented in 5.3 and we conclude the chapter in 5.4.

5.1 PRELIMINARIES

5.1.1 *Neural language modeling*

A neural language model aims to model the next word generative distribution $p(w_{k+1}|w_{<k})$ by a neural network and the model is trained to maximize the average log-likelihood of all possible context and word pairs. Namely, the data

sequence $w_1, w_2, ..., w_T$ is decomposed to a set of pairs $\{(w_{<i}, w_i)\}$ with $i = 1, ..., T$ and the learning objective is to

$$\max \frac{1}{T} \sum_{i=1}^{T} \log p_\theta(w_k | w_{<i}) \tag{72}$$

where θ indexes the parameters of the neural network. For example, an RNN language model takes all previous words as input, and produces a hidden vector as digest for the context. The output from the RNN is fed into a softmax layer to produce probabilities over the entire vocabulary.

5.1.2 *Neural machine translation*

Meanwhile, the huge success of neural machine translation (NMT) methods is witnessed by the great performance of fitting parallel corpus to translate a sentence from one language to that from another. Figure 14 depicts a classic encoder-decoder architecture for machine translation

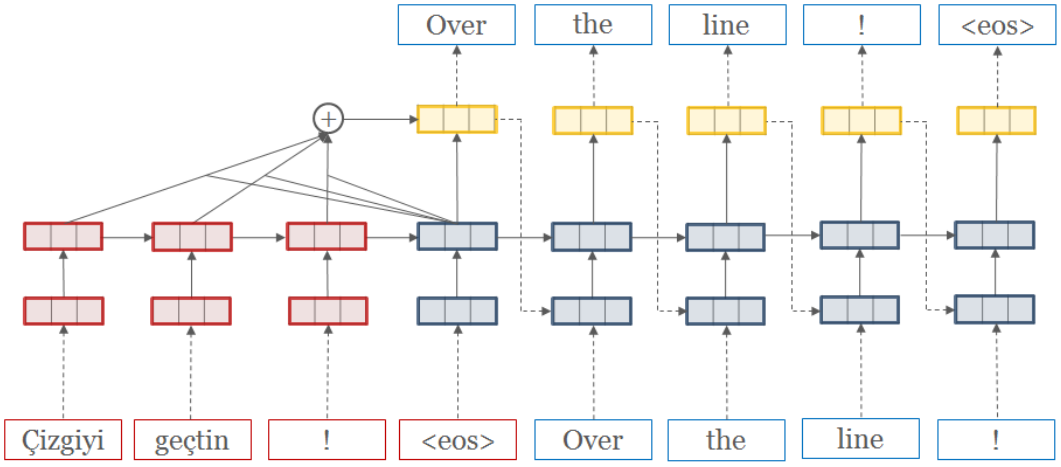


Figure 14: A simple NMT system with attention (reproduced from http://opennmt.net/)

Essentially, the neural MT system aims to maximize the conditional generative log-likelihood of a sentence $\mathbf{y}$ in the target language from the parallel sentence in the source language $\mathbf{x}$, i.e.

$$\max \log p(\mathbf{y}|\mathbf{x}) \equiv \log p(y_1, y_2, ..., y_m | x_1, x_2, ..., x_n) \tag{73}$$

$$= \sum_{i=1}^{m} \log p(y_i | x_1, x_2, ..., x_n, y_{<i}) \tag{74}$$

where m, n are the number of tokens in the target and source sentences, respectively. When multiple pairs of source and target sentences are present, the conditional generative log-likelihood is normalized by target sentence length and set as the training objective.

5.1.3 *Beam search for language generation*

Beam search is a greedy strategy in language generation to generate a sequence of tokens such that the likelihood of the generated sentence could be maximized. It is greedy in the sense that in each step of the generation, a set of K (a user specified parameter) most probable words is kept and generation for the next step is conditioned on this seed set and again a set of K most probable words are kept to the generated sequences. Beam search is essentially trying to maximize the long term probability of the full generated sequence in an ad-hoc manner. The greedy strategy does alleviate the problem of only focusing maximizing the per-step likelihood of a single word, however this is still a compromise to accurately model the joint generative likelihood of the entire sequence. Meanwhile, the proposed method explicitly models the desired objective, i.e., the probability of completing the full sequence, hence a better language model can be obtained.

5.2 PROPOSED METHOD

We propose to direcly model the conditional generative distribution of completing a language sequence given a context, i.e, $p(w_{k+1}, w_{k+2}, ..., w_T | w_1, w_2, ..., w_k)$. Specifically, two recurrent neural networks are used. The encoder RNN takes the context $w_1, w_2, ..., w_k$ as input and produces a digest as the context; the decoder RNN takes the context vector and tries to correctly decode it to the target sub-sequence $w_{k+1}, w_{k+2}, ..., w_T$.

Compared with existing language models, which are all based on modeling the per-step predictive distribution of the best next word, the proposed method essentially enlarges the prediction horizon such that the predicted sequence is more probable in the long run. The success of RNN based language models to capture a longer context history demonstrates the need to model long term dependencies over the history; the same idea should also help language modeling on the prediction horizon. A longer prediction horizon is more likely to capture the dynamics in natural languages, as the meaning of a sentence is reflected by the sequence, not any individual word.

Basic sequence unit. There are various ways to define a list of word tokens as a complete sequence for language modeling, e.g., a complete sequence can be a sentence, a paragraph or even an entire document. In this chapter, we choose to

treat a sentence as the basic unit for language sequence, for both its semantic to express some meaning and also its relatively short length compared to a paragraph or even a document for ease of model training.

Stochastic sentence slicing. With a bit of notation abuse, consider a sentence with T words, $w_1, w_2, ..., w_T$. There are T different ways to slice the sentence into the input and the target sub-sequences. Hence to process a sentence, a total of T training pairs can be extracted and fed sequentially to the encoder and decoder respectively, i.e.

$$\max_{\theta,\phi} \frac{1}{T} \sum_{k=0}^{T-1} \frac{1}{T-k} \log p_{\theta,\phi}(w_{k+1}, w_{k+2}, ..., w_T | w_1, w_2, ..., w_k) \tag{75}$$

where θ and ϕ denote the parameters for the encoder RNN and decoder RNN, respectively.

In analogy to stochastic gradient descent, we propose to sample an integer k uniformly from $0, 1, ..., T-1$ for each sentence, and optimize

$$\max_{\theta,\phi} \frac{1}{T-k} \log p_{\theta,\phi}(w_{k+1}, w_{k+2}, ..., w_T | w_1, w_2, ..., w_k)$$
$$k \sim \mathtt{Uniform}\{0, 1, ..., T-1\} \tag{76}$$

For a context sub-sequence $w_1, w_2, ..., w_k$ and the target sub-sequence $w_{k+1}, w_{k+2}, ..., w_T$, the encoder RNN encodes the context sequence and feeds it to the decoder; the decoder RNN takes the most recent word as input, together with the context vector and tries to maximize the likelihood of generating the next word. The process continues until all target words are evaluated by the decoder. In other words, the conditional generative distribution is computed as

$$\log p_{\theta,\phi}(w_{k+1}, w_{k+2}, ..., w_T | w_1, w_2, ..., w_k) \tag{77}$$
$$= \sum_{j=k+1}^{T} \log p(w_j | g_\phi(c, w_{k+1}, w_{k+2}, ..., w_{j-1})) \tag{78}$$

where $c = f_\theta(w_1, ..., w_k)$ is the digest of the input produced by the encoder RNN.

Regularization. We apply similar regularization techniques to both the encoder and decoder as commonly used by state-of-the-art RNN based language models (Merity et al., 2017).

5.3 EXPERIMENTS

We implemented the MT based language models with LSTM and conducted experiments on two widely used language modeling data sets, Penn Tree Bank (PTB) (Mikolov and Zweig, 2012) and WikiText-2 (WT2) (Merity et al., 2016).

5.3.1 *Test set perplexity*

Generative modeling results compared to state of the art methods are presented in Table 6 for PTB and in Table 7 for WikiText2. To enable fair comparisons with state-of-the-art methods, we control the number of parameters to be roughly the same as theirs. The proposed model is implemented in PyTorch[1] and all experiments are conducted on an Nvidia GeForce GTX 1080Ti graphics card.

Model	#Params	Validation	Test
(Kalchbrenner et al., 2014) – RNN-LDA + KN-5 + cache	9M	-	92.0
(Dieng et al., 2016) - TopicRNN	-	99.6	97.3
(Zaremba et al., 2014) – LSTM	20M	86.2	82.7
(Gal and Ghahramani, 2016) – Variational LSTM (MC)	20M	-	78.6
(Sennrich et al., 2015) – CharCNN	19M	-	78.9
(Merity et al., 2016) – Pointer Sentinel-LSTM	21M	72.4	70.9
(Grave et al., 2016) – LSTM + continuous cache pointer	-	-	72.1
(Inan et al., 2016) – Tied Variational LSTM + augmented loss	24M	75.7	73.2
(Zilly et al., 2016) – Variational RHN	23M	67.9	65.4
(Melis et al., 2017) – 2-layer skip connection LSTM	24M	60.9	58.3
(Merity et al., 2017) - AWD-LSTM	24M	60.7	58.8
(Yang et al., 2017) - AWD-LSTM-MoS	22M	58.1	55.9
(Zoph and Le, 2016) – NAS Cell	25M	-	64.0
(Liu et al., 2018) - DARTS	23M	58.3	56.1
(Ours) EncDecLM	22M	58.9	57.0

Table 6: Perplexity on Penn Tree Bank (lower is better). Baseline results obtained from (Merity et al., 2017), (Yang et al., 2017) and (Liu et al., 2018).

From both sets of results, it's evidently clear that with the aim to model the remaining part of a sentence given a context is significantly better than just modeling the next word given the same context. Note that the performance of the proposed methods are not extensively tuned, which again demonstrates the effectiveness of the motivation for the proposed model. Also, the improvement of the proposed approach is orthogonal to using mixture of softmaxes (Yang et al., 2017), hence the performance can be further boosted if combined with MoS.

1 https://pytorch.org/

Model	#Params	Validation	Test
(Inan et al., 2016) – Variational LSTM + augmented loss	28M	91.5	87.0
(Grave et al., 2016) – LSTM + continuous cache pointer	-	-	68.9
(Melis et al., 2017) – 2-layer skip connection LSTM	24M	69.1	65.9
(Merity et al., 2017) - AWD-LSTM	33M	69.1	66.0
(Yang et al., 2017) - AWD-LSTM-MoS	35M	66.0	63.3
(Ours) EncDecLM	32M	67.5	64.8

Table 7: Perplexity on WikiText-2 (lower is better). Baseline results obtained from (Merity et al., 2017) and (Yang et al., 2017).

5.3.2 *Generated language samples*

We show case several language samples of length 300 words from the best model trained on PTB.

Sample 1:

```
on frankfurt sterling values as oil prices closed <unk> at a lower level as
     well as on tuesday 's market decline <eos>
after the current resignation but wo n't ranged in today 's high <eos>
N percentage point <eos>
the dollar 's N plunge worried about the <unk> of selling <eos>
in light of the past several factors <eos>
the dow jones market plunged N points to N pence friday <eos>
wall street was <unk> by gainers by others <eos>
investors located in sydney southwest <eos>
friday <eos>
late friday <eos>
executing major trading were respected in landmark <eos>
by potentially low ranges from most major currencies <eos>
the board 's move <eos>
british petroleum co. 's ford financial report <eos>
friday <eos>
of comparison warned the federal reserve 's coffee ratio it does n't
    decline <eos>
the most defensive pursuit of current drilling industry is published <eos>
in this article <eos>
stock prices went back friday <eos>
the dollar closed last year <eos>
early trading in london and manila say it is n't likely to <unk> its
    intermediate ceiling <eos>
coming killing the interstate 's monetary policy <eos>
```

```
after the rule income <eos>
much <eos>
further and economic improvement <eos>
fiscal N <eos>
safe growth of scoring high income <eos>
taxes <eos>
showed reserves ending early september <eos>
the N N increase issued <eos>
the second leading seasonally adjusted for the u.s. economy <eos>
only N growth <eos>
the imf said <eos>
N <eos>
some economists had never been in <eos>
at the past week <eos>
she has focuses on the domestic earnings and down the ceiling <eos>
<unk> the bank <eos>
a N N increase in margin <eos>
proportion <eos>
income from usual period <eos>
in the year-earlier period <eos>
in houston <eos>
he said <eos>
```

Sample 2:

```
kravis universal foods corp. announced a $ N million labor-management bid
    for the federal bankruptcy code in in an attempt to limit the goal of a
    license for expenditures <eos>
the waiting period <eos>
its full terms at which it does n't fail to pay for the ual deal <eos>
every three years <eos>
he exists on the robins carrier <eos>
with its adrs and <unk> cost customers to focus on debt <eos>
good notes <eos>
when the million redemption amount of six other properties may lead to the
    pay stage <eos>
cars <eos>
the company 's position only <eos>
the government 's defaulted value would be for sale <eos>
N million <eos>
revenue <eos>
it is in an outsider <eos>
to join prudential-bache <eos>
<unk> power units <eos>
the facility <eos>
insiders owned by the underwriters <eos>
grocery waste co <eos>
```

```
armstrong ’s chief operating partners <eos>
atlanta <eos>
palm <unk> union needs to sell the company ’s assets <eos>
the trust stock in other areas <eos>
but would do n’t have any plans to handle it <eos>
offices <eos>
<eos>
the case <eos>
N <eos>
the person outlawed george <unk> chief executive will take act on a
    placement of assets <eos>
at a <unk> price <eos>
advised mccaw based in an interview <eos>
wpp group <eos>
norfolk tests <eos>
the high-yield fund that the transaction were seeking to deal with <eos>
the majority of the company ’s assets <eos>
it <unk> <eos>
filing <eos>
the company took place monday <eos>
john robertson chairman and chief executive officer <eos>
georgia-pacific <eos>
cigna and executive as well as a director of qintex entertainment ’ sale <
    eos>
the agreement <eos>
the british company <eos>
it has the debt offering after potential increases and details of its
    transactions <eos>
generation study patent
```

Sample 3:

```
but many will have less discounting <unk> but able to come at monday before
     contemporary party will be at the crucial time <eos>
vicar <unk> president in the financial-services market at lebanon <eos>
the credit institute <eos>
the <unk> for businesses <eos>
the rapid consortium of <unk> east bloc here <eos>
the region <eos>
bankers <unk> <unk> adviser and trying to ease their official <unk> <eos>
the public management contractor we consider credibility <eos>
a cost-of-living tube <eos>
more than people near the country or cambodia <eos>
the provider of only all the <unk> issues <eos>
for some positive <unk> but indeed after the soviet union <eos>
why those <unk> were closed but were learning <eos>
their massive <unk> did n’t open even intense <eos>
```

```
a shipbuilding industry for <unk> <eos>
they halls <eos>
<unk> <eos>
the regular and <unk> <unk> of inspection <eos>
of the almost $ N million code with hispanics to the site of japan <eos>
sailing operations <eos>
which it expects to deliver about $ N million <eos>
to improve these costs <eos>
reached and in the last two years <eos>
efforts to require <unk> efficiency and consumer sales <eos>
the collapse of sudden increases in dual analysis <eos>
and palace <unk> costs <eos>
flew from the steel operations <eos>
recycled savings bank <eos>
the federal local government <eos>
no significant accounts <eos>
putting both real coverage <eos>
<eos>
effective to any <unk> soon <eos>
dividends <eos>
<eos>
the strong growth of the guerrilla process violates how much of the nih
    struck or <unk> corp. would continue to be described as a result of
    amended a restricting interest since N <eos>
later <eos>
these rates have been short-lived <eos>
<eos>
at least a bad one hands of efficient sales <eos>
and is under
```

Sample 4:

```
up the concern <eos>
guests in tokyo entered the first <unk> <unk> and its <unk> business
    program <eos>
<unk> as the N <unk> of arkansas <unk> that jointly chosen to register a <
    unk> to N chinese <unk> in <unk> cambria county living in new york <eos>
pushing money $ N N on <unk> 's N winter games in the suburban new england
    <eos>
the virgin islands <eos>
between the soviet union and the national institutes of congress <eos>
office <eos>
an <unk> development candidate <eos>
an <unk> bill to staffers support comeback <eos>
after a lobbying estimate <eos>
at least against abc was <unk> and less expensive centers <eos>
leaving into <unk> <unk> <eos>
```

```
a student <unk> <eos>
the mandatory age of series <eos>
refuge as a <unk> <eos>
such a <unk> <unk> revamped <eos>
the <unk> against the <unk> <eos>
<unk> n.m. <eos>
because this have been <unk> la <unk> mayor dai-ichi rockefeller <eos>
N hours <eos>
break the <unk> <unk> tell questions <eos>
weigh on a major <unk> <eos>
row noting that he was <unk> <eos>
a republic scotland new york legislature on its <unk> center <eos>
fate <eos>
the moore show for just another newspaper <eos>
black troops <eos>
is another respective life <eos>
everything from <unk> <eos>
terry core school and urge it along to the french <unk> <eos>
class of the movie program being <unk> <eos>
the home system in <unk> with a <unk> that disappointment that the <unk>
    can make data with the <unk> of the <unk> or the <unk> <unk> of which
    the <unk> is <unk> out <eos>
the <unk> or interest james <unk> 's environmental clearance on inner land
    an official in los angeles where a atlantic island investigation the
    picture even any <unk> tally by gov. kean cities/abc corp. and the
```

It can be seen that the generated text samples resemble well to natural sentences with the same theme as sentences from the data set. Meanwhile, the logic and story flows are reasonably good. Note that in the absence of any grammatical intervention and regularizations, grammatical errors occur less than we would have expected.

5.4 CONCLUSIONS

In this chapter, we propose to address the language modeling problem from a fundamental perspective for language modeling , i.e., trying to model how the language sequence can be completed based on the context. Specifically we propose to model the conditional sequence generative distribution with two recurrent neural networks, with one to encode and digest the history and the other to decode the context and generate the target sequence to complete the sequence. We conduct empirical evaluations against state-of-the-art RNN language models on several benchmark data sets and our model improves over several baseline language models, achieving comparable performances with the best method so far.

6

CONTINUOUS SEQUENCES MODELING WITH CONTEXT-AWARE NORMALIZING FLOWS

Learning probabilistic generative models of sequential data is a long standing research problem. For discrete sequences, such as natural language and text, recurrent auto-regressive geneartive models without latent varibles, such as PixelRNN, WaveNet, have been dominant in terms of performance in data lilkelihood. However for continuous sequential data, such as speech signals and time series, the state-of-the-art methods so far are all based on recurrent latent variable modelds. Due to the high volatility and varying dynamics of high dimensional time series data, the introduction of latent variables per time step seems inevitable for successful generative modeling. Though recurrent latent variable models outperform those without latent variables, it brings up two major difficulties: a) an inference model is required for model learning and b) exact data likelihood is infeasible since approximate inference has to be used in model learning.

In the area of multivariate temporal data analysis, infering the correlations, or latent structures among the different signal dimensions is another critical problem to better understand the interactions among the variables, with the potential to leading to deeper and more accurate data understanding and prediction. Traditional methods in such direction often relies on parametric assumptions about the joint distribution of the multivariate temporal signals, a widely used one of which is Gaussian (Friedman et al., 2010). The parametric assumption often limits the application scope of such methods. In this chapter, we propse to model multivariate temporal data with a novel framework which supports not only fast inference for unseen data series, but also complex or even non-parametric distributions among the different signal dimensions.

In this chapter, we propose to model continuous sequential data with recurrent network, equipped with conditional normalizing flow which enables exact likelihood evaluation and effective learning due to the absence of an inference model. The proposed methods significantly outperforms state-of-the-art recurrent latent variable models on two major speech audio datasets.

6.1 INTRODUCTION

Learning generative models of sequences has long been an important research problem. Because of the temporal dynamics underlying the data, essential care has be to taken of capturing temporal dependencies among different time stamps. In prior to the emergence of deep neural networks, dynamic Bayesian networks (DBN) (Murphy, 2002) has been one of most widely used methods for modeling sequences, including hidden Markov models (HMM) to model speech signals and dynamic topic models (DTM) (Blei and Lafferty, 2006) to model natural language texts. The key component of DBNs is the latent variables associated with each time stamp, i.e., hidden states in HMM and latent topics in DTM, which is designed and empirically demonstrated to capture to underlying structure of the data.

Recently, deep neural networks have been recognized for their flexibility and capability to approximate universal functions, and particulary recurrent neural networks (RNN) has been proposed to capture and model sequential data. RNN has been shown to succeed in various probablistic generative modeling tasks on discrete sequential data, including machine translation (Bahdanau et al., 2014; Luong et al., 2015; Sutskever et al., 2014), language modeling (Melis et al., 2017; Yang et al., 2017) and image generation[1] (Oord et al., 2016a). It's worth noting that all these generative modeling methods don't assume latent variables associated with each timestamp and yet they still produce state-of-the-art results in terms of data likelihood measures, outperforming their counterparts with latent variables, i.e., those based on DBN. Though no latent variables are used and thus the family of data generative distribution could be somewhat limited, the massive number of parameters and RNN's representing power to capture long term dependencies compensates the lack of latent variables. This verifies that powerful generative modeling without latent variables is possible.

However, the story is totally different for continuous sequential data, where the succeeding methods in terms of data likelihood are those based on recurrent latent variable models. For example, on modeling raw speech signals, significantly higher data likelihoods are obtained by incorporating latent variables to RNN, e.g., variational recurrent neural networks (VRNN) (Chung et al., 2015), SRNN (Fraccaro et al., 2016), z-forcing (Kim et al., 2018) and Stochastic WaveNet (Lai et al., 2018). It is argued that due to the high volatility and varying dynamics underlying continuous sequential data, latent variables are introduced for each timestamp and a more flexible family of data generating distribution for each time stamp can be constructed.

Though recurrent latent variable models achieve state-of-the-art in modeling continuous sequential data, they bring up two major difficulties:

1 Each pixel in each of the RGB channel of an image is treated as discrete values from $[0, 255]$

- An additional inference model is required for variational inference model model learning. This could be more difficult if the recurrent generative model is complex;
- No exact likelihood evaluation can be provided as variational inference is only approximately optimizing the data likelihood.

In this chapter, we try to address the performance gap between the RNN-based methods and recurrent latent variable models, by arguing that the key lies in enriching the per-step data distribution in a context aware manner. We propose to model continuous sequential data without latent variables by incorporating RNN with conditional normalizing flows. The proposed model admits exact inference thus avoiding the above two difficulties. Importantly, we show that in empirical generative modeling of continuous sequential data, the proposed model significantly outperforms all state-of-the-art methods in terms of data likelihood.

The rest of the chapter is organized as follows: we firstly briefly review the basics of neural generative modeling of continuous sequences and then propose to improve over existing approaches. Empirical evaluations and analysis are conducted and lastly we conclude the chapter.

6.2 PRELIMINARIES

The problem of generative modeling of continuous sequential data is, given a sequence of data vectors $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_T\}$ associated with timestamps from 1 to T, where each vector is d-dimensional continuous vector $\mathbf{x}_i \in \mathbb{R}^d$, to learn a probabilistic model by maximizing the joint data likelihood

$$\max_\theta p_\theta(\mathbf{x}_1, ..., \mathbf{x}_T) = \max_\theta \prod_{i=1}^{T} p_\theta(\mathbf{x}_i | \mathbf{x}_1, ..., \mathbf{x}_{i-1}) \tag{79}$$

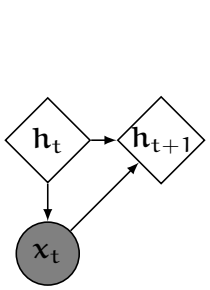
Thus the essence of generative modeling is to learn the conditional density $p(x_t|x_1, ..., x_{t-1})$. Prior to the deep learning era, to model such temporal dependencies, often historical data with a fixed window size s is considered, potentially assuming that the data sequence is a Markov chain with order s. For continuous sequences with an infinite support, a widely used family for the conditional density $p(x_t|x_1, ..., x_{t-1})$ is the Gaussians. A couple of existing methods can be categorized to this setting; for example, the vector auto-regressive model (VAR) essentially assumes a Gaussian conditional for $p(\mathbf{x}_i|\mathbf{x}_{i-1}, ..., \mathbf{x}_{i-s})$ and the model is learned by minimizing the mean squared error (MSE) between the predicted value and true one, equivalent to maximizing the Gaussian log-likelihoods.

6.2.1 *Recurrent generative models*

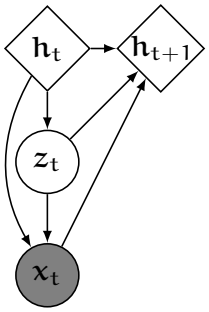
To fully leverage long term dependencies from the sequence history without exploding the number of parameters, recurrent generative models aims to capture the conditional density $p(x_t|x_1,...,x_{t-1})$ via recurrent neural networks (RNN, including all its variants such as LSTM (Hochreiter and Schmidhuber, 1997), GRU (Cho et al., 2014a)). Specifically, the RNN hidden state reads in the history and produce a digest, and then the conditional density is defined for the current time stamp with the hidden state as a parameter

$$p(x_t|x_1,...,x_{t-1}) \equiv p_\theta(x_t|h_{t-1}) \tag{80}$$

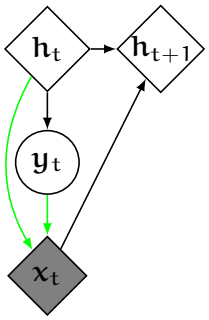
where $h_{t-1} = \text{RNN}(\mathbf{x}_1,...,x_{t-1})$ is the RNN hidden state encoding the data history up to timestamps $t-1$. Again for continuous sequences, a widely used distribution family for $p_\theta(x_t|h_{t-1})$ is the Gaussians. With RNN in the setting, the mean and variance for the Gaussian are defined as functions of the RNN hidden state, hence all parameters can be learned by back propagation over the loss function in Eq. (79). An illustration of recurrent generative models can be found in Figure 15(a).



(a) Recurrent generative models (RNN)



(b) Recurrent latent variable models (VRNN)



(c) Recurrent generative models with normalizing flows (proposed, the context aware NF is highlighted in green)

Figure 15: Illustration of various generative models (diamonds denote deterministic variables while circles denot random variables. Gray color denotes observed variables.)

6.2.2 *Recurrent latent variable models*

Besides directly modeling the data vector $\mathbf{x}_t$ at each time stamp, in many cases we often believe that there is some underlying structure governing the generation of the actual data. Since the structure is not observable to us, we associate a latent

variable z_t with each time stamp. With the latent variable taken into consideration, the following generative modeling process is can be defined

$$p(\mathbf{X}, \mathbf{Z}) = \prod_{i=1}^{N} p(z_i|x_1, ..., x_{i-1}) p(x_i|z_i, x_1, ..., x_{i-1}) \tag{81}$$

where $\mathbf{Z}$ is the collection of all z_t. Figure 15(b) shows an illustration of such model. An example of the above model is Gaussian-Markov Model if all conditionals are defined as Gaussians.

In the context of deep neural networks, if one models the per-step joint density $p(x_t, z_t)$ with a recurrent neural network, this essentially turns to be a recurrent latent variable model, as a recurrent sequence over the latent variables is defined as

$$p(z_i|x_1, ..., x_{i-1}) \equiv p_\theta(z_t|h_{t-1}) \tag{82}$$

$$p(x_i|z_i, x_1, ..., x_{i-1}) \equiv p_\phi(x_i|z_i, \mathbf{h}_{t-1}) \tag{83}$$

for $t \in \{1, 2, ..., , T\}$, where h_{t-1} is the RNN hidden state summarizing the entire context and for continuous data and latent variables, Gaussians are again assumed and parameterized for both $p_\theta(z_t|h_{t-1})$ and $p_\phi(x_i|z_i, \mathbf{h}_{t-1})$.

The model is learned by maximizing the log-likelihood of observed data; however due to the introduction of the latent variables, exact evaluation of the marginal data likelihood in Eq. (79) is infeasible. Approximate inference learning methods has to be applied, such as variational inference. Specifically to facilitate model inference, an inference model parameterized from neural network is proposed

$$q(\mathbf{Z}|\mathbf{X}) = \prod_{t=1}^{T} q(z_i|x_i, z_{i-1}) \tag{84}$$

where similar constructions for p are used as for q. To this end, both the generative model p and inference model q is joint learned by maximizing the ELBO

$$\log p(\mathbf{X}) = \log \mathbb{E}_{q(\mathbf{Z}|\mathbf{X})} \frac{p(\mathbf{X}, \mathbf{Z})}{q(\mathbf{Z}|\mathbf{X})} \tag{85}$$

$$\geqslant \mathbb{E}_{q(\mathbf{Z}|\mathbf{X})} \log p(\mathbf{X}, \mathbf{Z}) - \mathbb{E}_{q(\mathbf{Z}|\mathbf{X})} \log q(\mathbf{Z}|\mathbf{X}) \tag{86}$$

The ELBO is the key to success of many recurernt latent variables for sequences, including (Chung et al., 2015; Fraccaro et al., 2016; Kim et al., 2018), however this brings difficulties to manually design and specify an inference network and also that no exact likelihood evaluations can be provided from the learned model.

6.3 CONTEXT-AWARE NORMALIZING FLOWS

It's well received that for continuous sequential data, such as speech signals and time series, vanilla recurrent generative models based on RNN don't have enough representing power to model the temporal data generative distribution, thus latent variables are all incorporated in state-of-the-art generative models for continuous sequence modeling and significant performance boost have been observed (Chung et al., 2015; Fraccaro et al., 2016; Kim et al., 2018). However, due to the fact that all these recurrent latent variable models rely on variational inference for model learning, significant amount of efforts must be devoted to designing and specifying the inference networks, and deriving the ELBO for model learning. Besides, since only a lower bound of the actual data likelihood is obtained, no exact evaluations about data likelihood can be performed.

In this chapter, we take a step back by first pointing out that the expressive power of current state-of-the-art recurrent latent variable models comes from the context dependent prior for the latent variables at each timestamp. Based on that, we further propose to enrich per-step data generative distribution with normalizing flow (Rezende and Mohamed, 2015) in a *context-aware manner*. The resulting framework, which we termed as "NF-RNN", is not only able to capture complex and flexible per-step data generative distribution underlying continuous sequences without relying on latent variables and but also importantly, offering exact data likelihood evaluations, entirely eliminating the need for approximate variational inference.

6.3.1 *Essence of recurrent latent variable models*

For a vanilla recurrent generative model based on RNN, it parameterizes directly the per-step data distribution $p(x_t|x_1, ..., x_{t-1}) \equiv p_\theta(x_t|h_{t-1})$. Though RNN provides a flexible way to fit various distributions $p_\theta(x_t|h_{t-1})$ based on the data, the distribution family it can capture is still limited. For example, assuming $p_\theta(x_t|h_{t-1})$ is a Gaussian with its mean and co-variance matrix parameterized as functions of $(\mathbf{x}_t, \mathbf{h}_{t-1})$, although hypothetically it is able to fit any Gaussians with arbitrary mean and co-variances given sufficient number of parameters in the RNN (Hornik, 1991), however it will never to able to fit perfectly to a distribution with more than one modes (e.g., a mixture of 2 Gaussians).

To this end, recurrent latent variable models defines a hierarchical structure of how data at each step is generated on the one hand; on the other hand, it essentially defines a broader family of data distributions compared to the vanilla RNN case.

Consider instead the case where latent variables are assumed for each time stamps, from Eq. (83) the following per-step joint density over $(\mathbf{x}_t, \mathbf{z}_t)$ is defined

$$p(\mathbf{x}_t, \mathbf{z}_t|\mathbf{h}_{t-1}) \equiv p_\theta(\mathbf{z}_t|\mathbf{h}_{t-1}) p_\phi(\mathbf{x}_t|\mathbf{z}_t, \mathbf{h}_{t-1}) \tag{87}$$

Hence the actual conditional data generative distribution for $\mathbf{x}_t$ becomes

$$p(\mathbf{x}_t|\mathbf{h}_{t-1}) = \int_{\mathbf{z}_t} p(\mathbf{x}_t, \mathbf{z}_t|\mathbf{h}_{t-1}) d\mathbf{z}_t \tag{88}$$

$$= \int_{\mathbf{z}_t} p(\mathbf{z}_t|\mathbf{h}_{t-1}) p(\mathbf{x}_t|\mathbf{z}_t, \mathbf{h}_{t-1}) d\mathbf{z}_t \tag{89}$$

$$= \mathbb{E}_{\mathbf{z}_t \sim p(\mathbf{z}_t|\mathbf{h}_{t-1})} p(\mathbf{x}_t|\mathbf{z}_t, \mathbf{h}_{t-1}) \tag{90}$$

which is essentially an infinite mixture of $p(\mathbf{x}_t|\mathbf{z}_t, \mathbf{h}_{t-1})$. Though similar to vanilla recurrent generative models without latent variables, $p(\mathbf{x}_t|\mathbf{z}_t, \mathbf{h}_{t-1})$ might be of simple forms, an infinite mixture of these simple densities could still constitute a rather expressive generative model. Indeed the success of various recurrent latent variables demonstrates the power of this technique (Chung et al., 2015; Fraccaro et al., 2016; Kim et al., 2018; Lai et al., 2018)[2], similar idea has been successfully applied to the non-sequential data settings (Maaløe et al., 2016; Ranganath et al., 2016).

It is worth pointing out that the history (context) dependent mixture weights, $p(\mathbf{z}_t|\mathbf{h}_{t-1})$ is the key to successful enriching the per-step data generative distribution for sequential data, as also empirically validated by various state-of-the-art recurrent latent variable models (Chung et al., 2015; Fraccaro et al., 2016; Kim et al., 2018).

6.3.2 *Context-aware normalizing flows*

Besides introducing latent variables, another approach to enrich a distribution family is to apply normalizing flow (Rezende and Mohamed, 2015) to transform a random variable with a known density to a complex one, where the transformation function is a bijective function. Suppose that $\mathbf{y} \in \mathbb{R}^d$ is a random variable with density $p(\mathbf{y})$, and assume that the data $\mathbf{x}$ can be transformed from $\mathbf{y}$ by $\mathbf{x} = g(\mathbf{y})$, where g is a normalizing flow. Thus $\mathbf{y} = g^{-1}(\mathbf{x}) \equiv f(\mathbf{x})$ is also a NF and the distribution of $\mathbf{x}$ can be analytically written as

$$\log p(\mathbf{x}) = \log p(\mathbf{y}) - \log \left| \det \frac{\partial \mathbf{x}}{\partial \mathbf{y}} \right| \tag{91}$$

$$= \log p(f(\mathbf{x})) + \log |\det f'(\mathbf{x})| \tag{92}$$

2 WaveNet (Oord et al., 2016b) and Stochastic WaveNet (Lai et al., 2018) are not based on RNNs, however they are still modeling the causal relationship between the past and the present, just with CNN to replace RNN for better model training and testing efficiencies

where $\det \frac{\partial x}{\partial y}$ denotes the determinant of the Jacobian matrix. With normalizing flow, the resulting data generative distribution is enriched and its flexibility is controlled by the capacity of the NF network.

However, we argue in this chapter that directly applying the NFs here will lead to sub-optimal performances, as the NF is not aware of the history in the RNN. In analogy to the history dependent prior distribution of latent variables in recurrent latent variable models , we propose to employ history dependent normalizing flows on top of the plain recurrent generative models to enrich the representation power for recurrent generative models. Specifically, the NF for the data distribution at time stamp t is assumed to be a function indexed by the data history up to that point, i.e.

$$p(\mathbf{y}_t|\mathbf{x}_1,...,\mathbf{x}_{t-1}) \equiv p_\theta(\mathbf{y}_t|\mathbf{h}_{t-1}) \tag{93}$$

$$\mathbf{x}_t = f_{\phi,\mathbf{h}_{t-1}}(\mathbf{y}_t) \tag{94}$$

where h_{t-1} is the hidden state encoding the context from an RNN and p_θ is a simple data generation density, such as Gaussian or mixture of Gaussians for continuous data, and f is a normalizing flow. The free parameters are $\{\theta, \phi\}$ and we emphasize the dependence of f on $\mathbf{h}_{t-1}$. An illustration of NF-RNN compared with recurrent generative models and recurrent latent variables can be found in Figure 15.

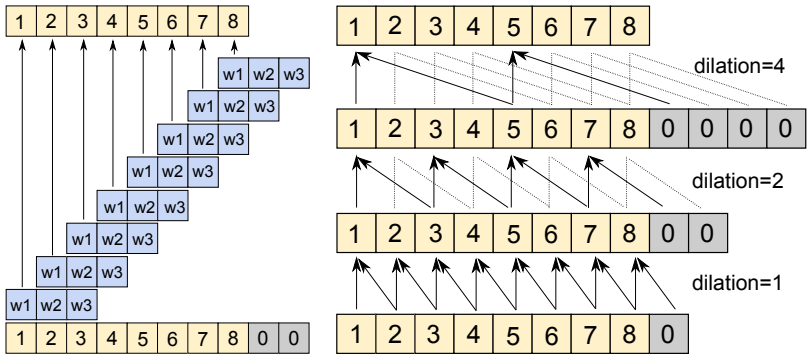
Standard normalizing flows in the form of $\mathbf{x}_t = f(\mathbf{y}_t)$ are context agnostic, fortunately we can build context-aware normalizing flows easily by conditioning the parameters of f on the context $\mathbf{h}_{t-1}$. There are various NFs we can choose from to model f, including planar flow (Rezende and Mohamed, 2015), inverse auto-regressive flow (Kingma et al., 2016), masked auto-regressive flow (Papamakarios et al., 2017). In this chapter we adopt the convolutional normalizing flows (Zheng et al., 2017b) for their simplicity to train and the ease to extend to the context-aware setting.

6.3.3 *Context Aware Convolutional Normalizing Flows*

As proposed in (Zheng et al., 2017b), a convolutional normalizing flow (ConvFlow) transforms an input random vector $\mathbf{x}$ with 1-d convolution, followed by a non-linearity and necessary post operation after activation to generate an output vector. Specifically,

$$f(\mathbf{x}) = \mathbf{x} + \mathbf{u} \odot h(\text{conv}(\mathbf{x}, \boldsymbol{w})) \tag{95}$$

where $\boldsymbol{w} \in \mathbb{R}^k$ is the parameter of the 1-d convolution filter (k is the convolution kernel width), $\text{conv}(z, \boldsymbol{w})$ is the 1d convolution operation with proper zero padding



(a) (b)

Figure 16: (a) Example of 1-d convolution, assuming the dimensions of the input/output variable are both 8 (the input vector is padded with 0), the convolution kernel size 3 and dilation is 1; (b) A ConvBlock, i.e., a stack of ConvFlow layers with different dilations from $1, 2, 4$, up to the nearest powers of 2 smaller than the vector dimension.

to ensure the output vector has the same dimension as the input, shown in Figure 16(a), $h(\cdot)$ is a monotonic non-linear activation function, $\odot$ denotes point-wise multiplication, and $\mathbf{u} \in \mathbb{R}^d$ is a vector adjusting the magnitude of each dimension of the activation from $h(\cdot)$. Refer to (Zheng et al., 2017b) for a detailed coverage of ConvFlow.

One appealing property about ConvFlow is its ability to flexibly warp the input random vector with a simple density to a complex one, by stacking a series of k ConvFlows to generate complex output densities. Furthermore, dilated convolutions (Oord et al., 2016b; Yu and Koltun, 2015) are incorporated to the flow to promote interactions among dimensions with long distance apart: different dilations for each ConvFlow increase the receptive field without exploding the number of model parameters, as illustrated by the ConvBlock in Figure 16(b). Most importantly after applying one layer of ConvFlow, the Jacobian matrix is triangular,

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \mathrm{I} + \mathrm{diag}(w_1 \mathbf{u} \odot h'(\mathrm{conv}(\mathbf{x}, \boldsymbol{w}))) \tag{96}$$

hence its determinant can be effectively computed in time linear to the input dimension, which is critical to computing the density of the resulting vector from a normalizing flow. The Jacobian matrix of applying a ConvBLock or more is straightforward by simply multiplying the Jacobians of each individual transformation.

Context aware normalizing flow with ConvBlocks. Note that in the original ConvFlow, both $\boldsymbol{w}$ and $\mathbf{u}$ are free paramters to learn. To make the normalizing

flow context aware, we propose to extend $\mathbf{u}$ to be dependent on the hidden states of the RNN, i.e. for time stamp t,

$$\mathbf{u}_t = l(\mathbf{h}_{t-1}) \tag{97}$$

where g is a function, e.g. represented by a neural network, whose parameters can be learned by back propagation. Thus the per-step data generative distribution with one layer of context aware convolutional normalizing flow is

$$\begin{aligned} &\log p(\mathbf{x}_t|\mathbf{h}_{t-1}) = \log p_\theta(f(\mathbf{x}_t)|\mathbf{h}_{t-1}) + \log|\det f'(\mathbf{x}_t)| \\ &\text{where } \; f(\mathbf{x}_t) = \mathbf{x}_t + l(\mathbf{h}_{t-1}) \odot h(\text{conv}(\mathbf{x}_t, \boldsymbol{w}) \end{aligned} \tag{98}$$

The representative power of convolutional normalizing flow increases by stacking multiple layers of convolutional flow layers with different dilations, i.e., a ConvBlock as shown in Figure 16(b). In practice, we can stack multiple ConvBlocks to construct a more expressive normalizing flow. Denote the normalizing flow with k ConvBlocks as F_k (with all the $\mathbf{u}$ vectors parameterized similar to Eq. (97)), the resulting per-step data generative distribution is

$$\log p(\mathbf{x}_t|\mathbf{h}_{t-1}) = \log p_\theta(F_k(\mathbf{x}_t)|\mathbf{h}_{t-1}) + \log|\det F_k'(\mathbf{x}_t)| \tag{99}$$

To this end, with the proposed recurrent generative model with k layers of context-aware ConvBlocks, the final objective function to optimize is

$$\begin{aligned} &\max \log p(\mathbf{x}_1, ..., \mathbf{x}_T) = \max \sum_{t=1}^{T} \log p(\mathbf{x}_t|\mathbf{h}_{t-1}) \\ &= \max \sum_{t=1}^{T} \Big(\log p_\theta(F_k(\mathbf{x}_t)|\mathbf{h}_{t-1}) + \log|\det F_k'(\mathbf{x}_t)| \Big) \end{aligned} \tag{100}$$

6.4 EXPERIMENT

We conduct evaluations of generative modeling of continuous sequences of the proposed method on a set of benchmark data sets, against state-of-the-art recurrent latent variable models.

6.4.1 *Data sets and setups*

We mainly evaluate on two widely evaluated speech signal data sets: TIMIT[3] and Blizzard[4]. TIMIT contains 6,300 spoken English sentences recorded by 630 speakers

3 https://catalog.ldc.upenn.edu/ldc93s1
4 http://www.cstr.ed.ac.uk/projects/blizzard/

Model	Blizzard	TIMIT
(Chung et al., 2015) - RNN	7413	26643
(Oord et al., 2016b) - WaveNet	−5777	26074
(Chung et al., 2015) - VRNN	$\geqslant$ 9392	$\geqslant$ 28982
(Fraccaro et al., 2016) - SRNN	$\geqslant$ 11991	$\geqslant$ 60550
(Goyal et al., 2017) - Z-forcing(+kla)	$\geqslant$ 14226	$\geqslant$ 68903
(Goyal et al., 2017) - Z-forcing(+kla, aux)	$\geqslant$ 15024	$\geqslant$ 70469
(Lai et al., 2018) - Stochastic WaveNet	$\geqslant$ 15708	$\geqslant$ 72463
(Ours) NF-RNN ($k = 4$)	**15983**	**72619**

Table 8: Test set log-likelihood on natural speech modeling (higher is better, k is the number of ConvBlocks in NF-RNN)

with each reading ten sentences. The Blizzard dataset (Prahallad et al., 2013) is from the Blizzard Challenge 2013, which contains 300 hours of spoken English by a single speaker.

For both datasets, we follow the same pre-processing procedures as in previous works (Chung et al., 2015; Fraccaro et al., 2016; Kim et al., 2018). Models are trained on the training set; best model is picked by best likelihood on the validation set and the corresponding likelihood on the test set is reported.

We also evaluate on one handwriting data set, the IAM-OnDB data set[5], which contains 13,040 handwritten lines by 500 writers (Liwicki and Bunke, 2005). Each line is a sequence of 3-dimensional vectors, with two dimensions recording the coordinates and the other one indicating pen-up or pen-down. The data set is pre-processed and split in the same manner as previous works (Graves, 2013).

To enable fair comparisons with existing state-of-the-arts, we use the following setups for model parameters:

- **TIMIT**: The data is mapped to a 512-dimensional embedding space and we use a one-layer LSTM for the RNN, with 1024 hidden units and a mixture of 20 Gaussians is used for the per-step generative distribution for the intermediate variable $\mathbf{y}_t$. Following previous work, we report the average log likelihood per sequence for the test set.

- **Blizzard**: The data is mapped to a 1024-dimensional embedding space and a one-layer LSTM is implemented for the RNN, with 2048 hidden units and

5 http://www.fki.inf.unibe.ch/databases/iam-on-line-handwriting-database

a mixture of 20 Gaussians is used for the per-step generative distribution for the intermediate variable $\mathbf{y}_t$. Similar to previous works, we report the average log-likelihood on half-second sequences for the test set.

- **IAM-OnDB**: The 3-dimensional data is mapped to a 200-dimensional embedding space before feeding to a one-layer LSTM, with 200 hidden units. Note that only 2 our of the three data dimensions is continuous and the third one is binary indicating whether the pen is up or down, hence the proposed normalizing flow only applies to this two dimensions and the results is concatenated with the third binary dimension to form a complete normalizing flow transformation. Similar to previous works, we report the average log-likelihood for the validation set[6].

On all three data sets, the context awareness reflected by Eq. (97) are all implemented by MLPs with 2 hidden layers with rectifier nonlinearities. The setups ensures comparable number of parameters used with existing state-of-the-arts. We use Adam (Kingma and Ba, 2014) with the cosine learning rate scheduling (Loshchilov and Hutter, 2016) for all experiments on three data sets and initial learning rates are tuned based on pilot exploratory analysis. For convolutional normalizing flows, the 1-d convolutional kernel size is set to 100 for TIMIT and Blizzard, and 2 for IAM-OnDB. The proposed model is implemented in PyTorch[7] and all experiments are conducted on an Nvidia GeForce GTX 1080Ti graphics card. Training is performed for 500, 500, and 200 epochs on TIMIT, Blizzard and IAM-OnDB, respectively and models with the best performance on validation sets are retained. Code to reproduce the reported numbers will be made publicly available.

6.4.2 *Generative modeling on speech signals*

Table 8 presents the log-likelihoods of the proposed method as well as those of previous state-of-the-art models on both TIMIT and Blizzard.

We can observe that the proposed NF-RNN provides exact log-likelihoods, similar to RNN and WaveNet, however the generative modeling performance is significantly better than these baseline methods. Secondly, compared to state-of-art recurrent latent variables, the log-likelihoods are significantly higher than those without KL regularization and auxiliary losses in training (VRNN, SRNN). Against baseline methods applying KL regularization and/or auxiliary losses in training, NF-RNN performs still slightly higher than them (Z-forcing and Stochastic WaveNet), or at least comparable to them since only lower bounds of log-likelihoods can be reported by those recurrent latent variable model baselines. This verifies

6 Existing state-of-the-arts split the entire data set into a training set and a validation set only.

7 https://pytorch.org/

Model	IAM-OnDB
(Chung et al., 2015) - RNN	1016
(Chung et al., 2015) - VRNN	$\geqslant$ 1334
(Lai et al., 2018) - Stochastic WaveNet	$\geqslant$ 1301
(Oord et al., 2016b) - WaveNet	1021
(Ours) - NF-RNN	**1359**

Table 9: Log-likelihood on IAM-OnDB (higher is better)

that for recurrent latent variables to work, additional regularization strategies need to be considered due to the large variance occurred in stochastic variational inference methods; in contrast, learning and inference with NF-RNN is much simpler and efficient as no additional inference network design and stochastic variational learning is needed to achieve these results. Plus, due to the analytical likelihood evaluations from the model, exact data likelihood evaluations are easily obtained.

6.4.3 *Generative modeling on human handwritings*

Table 9 presents the log-likelihoods of all methods on the validation data set for human handwriting modeling. NF-RNN achieves the best log-likelihoods among all existing methods, indicating that even when the data dimension is small (3 in the handwriting case), using normalizing flows will still significantly enrich the data generative distribution. Additionally, NF-RNN provides exact likelihood inference, another advantage not possessed by recurrent latent variable models.

6.4.4 *Ablation studies*

We conduct ablation studies on TIMIT and Blizzard where the data dimensions (200) are much larger than that of IAM-OnDB (3). We modify and test the proposed model in three aspects.

Importance of context awareness. We first test the role of context awareness plays in the proposed model. To do this, we fixed the number of ConvBlocks used and all other components of the architectures, but uses only the original context agnostic normalizing flow (as formulated in Eq.(95) where $\mathbf{u}$ is a free parameter independent from the RNN hidden state $\mathbf{h}_t$). Table 10 presents the results of the two settings. It can be observed that on TIMIT, without context awareness, NF-RNN

is still able to achieve significantly higher log-likelihoods compared to recurrent generative model baseline (RNN with 7413) and two recurrent latent variable model baselines (VRNN with 9392). After including context awareness, the performance is further gained, hence demonstrating the importance of incorporating historical information when enriching the data generative distribution per step. Similar trends can also be observed on Blizzard.

Model	Blizzard	TIMIT
(Ours) NF-RNN ($k = 4$, context independent)	10228	51387
(Ours) NF-RNN ($k = 4$, context aware)	**15983**	***72619***

Table 10: Test set log-likelihood on natural speech modeling with context independent NF and context awaren NF (higher is better)

Effects on the number of ConvBlocks. We next test NF-RNN with varying number of ConvBlocks for $k = 1, 2, 4$. Note that by increasing k, the number of parameters increases for NF-RNN, but only in a rather small amount since the convoluation window size is relatively small (100 for both TIMIT and Blizzard, 2 for IAM-OnDB) compared to the number of hidden units used in the RNN (1024 for TIMIT, 2048 for Blizzard and 200 for IAM-OnDB), thus the number of parameters is still comparable. Table 11 presents the full results, indicating that slightly better results can be obtained with larger k, which verifies that a more expressive data generative distribution can be obtained by stacking multiple ConvBlocks.

Model	Blizzard	TIMIT
(Ours) NF-RNN ($k = 1$)	14372	64630
(Ours) NF-RNN ($k = 2$)	15417	69328
(Ours) NF-RNN ($k = 4$)	**15983**	***72619***

Table 11: Test set log-likelihood on natural speech modeling with different number of ConvBlocks (higher is better)

Effects on the convolutional kernel size of ConvBlocks. Lastly we test the effects of using different convolution kernel size in ConvBlocks for NF-RNN. On TIMIT and Blizzard, we test on two different settings, convolution kernel size with 40 and 100, corresponding to 1/5 and half of the data size, respectively. Again tuning the kernel size only changes the total number of parameters by a small amount, since they are relatively small compared to the number of hidden units used in RNN. Table 12 presents the log-likelihoods for both settings. (The number

of ConvBlocks is set to 4). The results aligns with the observation that a longer convolution filter will model multiple dimensions from input more effectively, thus leading to a more expressive data generative distribution.

Model	Blizzard	TIMIT
(Ours) NF-RNN (kernel size 40)	15162	69077
(Ours) NF-RNN (kernel size 100)	**15983**	**72619**

Table 12: Test set log-likelihood on natural speech modeling (higher is better)

6.5 CONCLUSIONS

In this chapter, we present a simple yet effective method for generative modeling of continuous sequences with exact inference. We point out that the key ingredient to expressive probabilistic modeling for continuous sequences modeling lies in constructing a flexible data generative distribution. All existing state-of-the-arts rely on recurrent latent variables models to achieve better generative modeling performances, however that comes at the costs of manually designing complex inference network for necessary stochastic variational inference. Instead, we propose an exact method, termed NF-RNN, to construct expressive data generative distributions via convolutional normalizing flows and emphasize that context awareness is important to accommodate the rich dynamics underlying the sequential data. Empirical evaluations on two natural speech data data sets and one human handwriting data set clearly demonstrate the advantages of NF-RNN over recurrent latent variable models for continuous sequences modeling.

7

CONCLUSIONS AND FUTURE WORK

In this thesis, we aim to address the limitations of existing work in incorporating stochastic deep neural networks for probabilistic modeling. Specifically,

a) To facilitate stochastic variational inference for complex generative models defined by deep neural networks, we propose to enrich the family of variational posterior distributions for stochastic variational inference with two different strategies. One is to incorporate auxiliary random variables into the inference network for variational inference, which essentially results in an expressive variational posterior with an infinite mixture of simple densities; the other is to compose flexible and expressive densities for variational inference by warping simple densities with a novel and effective type of normalizing flows based on 1-d convolutions;

b) Certain discrete structures, including permutations, which are important to many machine learning tasks, haven't been studied in the context of neural probabilistic modeling. To this end, we propose to model and learn permutations with adversarial training for the unpaired setting; for the unsupervised setting, we construct probabilistic models over permutations and propose to learn such latent permutations from the data in a fully unsupervised manner;

c) Probabilistic modeling for sequential data domains, such as natural languages and continuous time series, haven't been fully explored and significantly more effective and elegant models could still be expected. For natural language modeling and continuous time series modeling, we propose two new probabilistic models respectively, based on simple insights from the task. Empirical evaluations have demonstrated the clear advantage of the proposed approaches over existing state-of-the-arts.

This thesis also shed light on understanding and manipulating stochasticity with deep neural networks for probabilistic modeling. A non-exhaustive list of possible interesting directions for future work includes:

- A general framework for stochastic architecture search to automatically design stochastic deep neural networks to better cater individual problems with uncertainty involved;
- A theoretical understanding of where randomness should be injected in stochastic deep neural networks to model uncertainty. Currently, there are models which place randomness on the input end, output end, and also in between; however, a thorough understanding of the differences among these ways of handling stochasticity is still lacking;
- Incorporating the generative permutation learning framework with language generation is also an interesting direction to pursue, where we can model the process of language generation as firstly generating a set of words and then permuting them to the correct order to make a natural sentence.

BIBLIOGRAPHY

Amodei, Dario, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. (2016). "Deep speech 2: End-to-end speech recognition in english and mandarin." In: *International Conference on Machine Learning*, pp. 173–182.

Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla (2017). "Segnet: A deep convolutional encoder-decoder architecture for image segmentation." In: *IEEE transactions on pattern analysis and machine intelligence* 39.12, pp. 2481–2495.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural machine translation by jointly learning to align and translate." In: *arXiv preprint arXiv:1409.0473*.

Bahdanau, Dzmitry, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio (2016). "End-to-end attention-based large vocabulary speech recognition." In: *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, pp. 4945–4949.

Blei, David M and John D Lafferty (2006). "Dynamic topic models." In: *Proceedings of the 23rd International Conference on Machine Learning*. ACM, pp. 113–120.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan (2003). "Latent Dirichlet Allocation." In: *Journal of Machine Learning Research* 3, pp. 993–1022.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe (2017). "Variational inference: A review for statisticians." In: *Journal of the American Statistical Association* 112.518, pp. 859–877.

Bottou, Léon (2010). "Large-scale machine learning with stochastic gradient descent." In: *Proceedings of COMPSTAT'2010*. Springer, pp. 177–186.

Brümmer, Niko (2016). "Note on the equivalence of hierarchical variational models and auxiliary deep generative models." In: *arXiv preprint arXiv:1603.02443*.

Burda, Yuri, Roger Grosse, and Ruslan Salakhutdinov (2015). "Importance Weighted Autoencoders." In: *arXiv preprint arXiv:1509.00519*.

Caetano, Tibério S, Julian J McAuley, Li Cheng, Quoc V Le, and Alex J Smola (2009). "Learning graph matching." In: *IEEE transactions on pattern analysis and machine intelligence* 31.6, pp. 1048–1058.

Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014a). "Learning phrase representations using RNN encoder-decoder for statistical machine translation." In: *arXiv preprint arXiv:1406.1078*.

Cho, Kyunghyun, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio (2014b). "On the properties of neural machine translation: Encoder-decoder approaches." In: *arXiv preprint arXiv:1409.1259*.

Chollet, François (2017). "Xception: Deep learning with depthwise separable convolutions." In: *arXiv preprint*, pp. 1610–02357.

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling." In: *arXiv preprint arXiv:1412.3555*.

Chung, Junyoung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio (2015). "A recurrent latent variable model for sequential data." In: *Advances in neural information processing systems*, pp. 2980–2988.

Cruz, Rodrigo Santa, Basura Fernando, Anoop Cherian, and Stephen Gould (2017). "Visual Permutation Learning." In: *CVPR*.

Cybenko, George (1989). "Approximation by superpositions of a sigmoidal function." In: *Mathematics of control, signals and systems* 2.4, pp. 303–314.

Dahl, George E, Dong Yu, Li Deng, and Alex Acero (2012). "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition." In: *IEEE Transactions on audio, speech, and language processing* 20.1, pp. 30–42.

Dieng, Adji B, Chong Wang, Jianfeng Gao, and John Paisley (2016). "Topicrnn: A recurrent neural network with long-range semantic dependency." In: *arXiv preprint arXiv:1611.01702*.

Dinh, Laurent, David Krueger, and Yoshua Bengio (2014). "NICE: Non-linear independent components estimation." In: *arXiv preprint arXiv:1410.8516*.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2016). "Density estimation using Real NVP." In: *arXiv preprint arXiv:1605.08803*.

Donahue, Jeff, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell (2014). "Decaf: A deep convolutional activation feature for generic visual recognition." In: *International conference on machine learning*, pp. 647–655.

Donahue, Jeffrey, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell (2015). "Long-term recurrent convolutional networks for visual recognition and description." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2625–2634.

Dumoulin, Vincent, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville (2016). "Adversarially learned inference." In: *arXiv preprint arXiv:1606.00704*.

Fraccaro, Marco, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther (2016). "Sequential neural models with stochastic layers." In: *Advances in neural information processing systems*, pp. 2199–2207.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2010). "A note on the group lasso and a sparse group lasso." In: *arXiv preprint arXiv:1001.0736*.

Gal, Yarin and Zoubin Ghahramani (2016). "A theoretically grounded application of dropout in recurrent neural networks." In: *Advances in neural information processing systems*, pp. 1019–1027.

Germain, Mathieu, Karol Gregor, Iain Murray, and Hugo Larochelle (2015). "MADE: masked autoencoder for distribution estimation." In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 881–889.

Geyer, Charles J (1992). "Practical markov chain monte carlo." In: *Statistical science*, pp. 473–483.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011a). "Deep sparse rectifier neural networks." In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323.

— (2011b). "Domain adaptation for large-scale sentiment classification: A deep learning approach." In: *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 513–520.

Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio (2016). *Deep learning*. Vol. 1. MIT press Cambridge.

Goyal, Anirudh, Alessandro Sordoni, Marc-Alexandre Côté, Nan Rosemary Ke, and Yoshua Bengio (2017). "Z-Forcing: Training stochastic recurrent networks." In: *Advances in Neural Information Processing Systems*, pp. 6713–6723.

Grave, Edouard, Armand Joulin, and Nicolas Usunier (2016). "Improving neural language models with a continuous cache." In: *arXiv preprint arXiv:1612.04426*.

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

Graves, Alex, Navdeep Jaitly, and Abdel-rahman Mohamed (2013a). "Hybrid speech recognition with deep bidirectional LSTM." In: *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, pp. 273–278.

Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton (2013b). "Speech recognition with deep recurrent neural networks." In: *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, pp. 6645–6649.

Hannun, Awni, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. (2014). "Deep speech: Scaling up end-to-end speech recognition." In: *arXiv preprint arXiv:1412.5567*.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hinton, Geoffrey, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. (2012). "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." In: *IEEE Signal processing magazine* 29.6, pp. 82–97.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory." In: *Neural computation* 9.8, pp. 1735–1780.

Hornik, Kurt (1991). "Approximation capabilities of multilayer feedforward networks." In: *Neural networks* 4.2, pp. 251–257.

Huang, Gao, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger (2017). "Densely Connected Convolutional Networks." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 2261–2269.

Inan, Hakan, Khashayar Khosravi, and Richard Socher (2016). "Tying word vectors and word classifiers: A loss framework for language modeling." In: *arXiv preprint arXiv:1611.01462*.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 448–456.

Joulin, Armand, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov (2016). "Bag of tricks for efficient text classification." In: *arXiv preprint arXiv:1607.01759*.

Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom (2014). "A convolutional neural network for modelling sentences." In: *arXiv preprint arXiv:1404.2188*.

Kim, Yoon (2014). "Convolutional neural networks for sentence classification." In: *arXiv preprint arXiv:1408.5882*.

Kim, Yoon, Sam Wiseman, Andrew C Miller, David Sontag, and Alexander M Rush (2018). "Semi-Amortized Variational Autoencoders." In: *arXiv preprint arXiv:1802.02550*.

Kingma, Diederik P and Max Welling (2013). "Auto-Encoding Variational Bayes." In: *arXiv preprint arXiv:1312.6114*.

Kingma, Diederik P., Tim Salimans, Rafal Józefowicz, Xi Chen, Ilya Sutskever, and Max Welling (2016). "Improving Variational Autoencoders with Inverse Autoregressive Flow." In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 4736–4744.

Kingma, Diederik and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization." In: *arXiv preprint arXiv:1412.6980*.

Lai, Guokun, Bohan Li, Guoqing Zheng, and Yiming Yang (2018). "Stochastic WaveNet: A Generative Latent Variable Model for Sequential Data." In: *arXiv preprint arXiv:1806.06116*.

Lai, Siwei, Liheng Xu, Kang Liu, and Jun Zhao (2015). "Recurrent Convolutional Neural Networks for Text Classification." In: *AAAI*. Vol. 333, pp. 2267–2273.

Lake, Brenden M., Ruslan Salakhutdinov, and Joshua B. Tenenbaum (2013). "One-shot learning by inverting a compositional causal process." In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural*

Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States. Pp. 2526–2534.

Larochelle, Hugo and Iain Murray (2011). "The Neural Autoregressive Distribution Estimator." In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pp. 29–37.

Linderman, Scott W., Gonzalo E. Mena, Hal Cooper, Liam Paninski, and John P. Cunningham (2017). "Reparameterizing the Birkhoff Polytope for Variational Permutation Inference." In: *arXiv preprint arXiv:1710.09508*.

Liu, Hanxiao, Karen Simonyan, and Yiming Yang (2018). "DARTS: Differentiable Architecture Search." In: *arXiv preprint arXiv:1806.09055*.

Liu, Jingzhou, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang (2017). "Deep learning for extreme multi-label text classification." In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, pp. 115–124.

Liu, Ziwei, Ping Luo, Xiaogang Wang, and Xiaoou Tang (2015). "Deep Learning Face Attributes in the Wild." In: *Proceedings of International Conference on Computer Vision (ICCV)*.

Liwicki, Marcus and Horst Bunke (2005). "IAM-OnDB-an on-line English sentence database acquired from handwritten text on a whiteboard." In: *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*. IEEE, pp. 956–961.

Loshchilov, Ilya and Frank Hutter (2016). "SGDR: stochastic gradient descent with restarts." In: *arXiv preprint arXiv:1608.03983*.

Luong, Minh-Thang, Hieu Pham, and Christopher D Manning (2015). "Effective approaches to attention-based neural machine translation." In: *arXiv preprint arXiv:1508.04025*.

Maaløe, Lars, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther (2016). "Auxiliary Deep Generative Models." In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1445–1453. URL: http://jmlr.org/proceedings/papers/v48/maaloe16.html.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.Nov, pp. 2579–2605.

Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey (2015). "Adversarial autoencoders." In: *arXiv preprint arXiv:1511.05644*.

Melis, Gábor, Chris Dyer, and Phil Blunsom (2017). "On the state of the art of evaluation in neural language models." In: *arXiv preprint arXiv:1707.05589*.

Mena, Gonzalo, David Belanger, Scott Linderman, and Jasper Snoek (2018). "Learning Latent Permutations with Gumbel-Sinkhorn Networks." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Byt3oJ-0W.

Merity, Stephen, Caiming Xiong, James Bradbury, and Richard Socher (2016). "Pointer sentinel mixture models." In: *arXiv preprint arXiv:1609.07843*.

Merity, Stephen, Nitish Shirish Keskar, and Richard Socher (2017). "Regularizing and optimizing LSTM language models." In: *arXiv preprint arXiv:1708.02182*.

Mescheder, Lars, Sebastian Nowozin, and Andreas Geiger (2017). "Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks." In: *arXiv preprint arXiv:1701.04722*.

Miao, Yishu, Lei Yu, and Phil Blunsom (2016). "Neural variational inference for text processing." In: *International Conference on Machine Learning*, pp. 1727–1736.

Mikolov, Tomas and Geoffrey Zweig (2012). "Context dependent recurrent neural network language model." In:

Mnih, Andriy and Karol Gregor (2014). "Neural Variational Inference and Learning in Belief Networks." In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pp. 1791–1799.

Mnih, Andriy and Danilo Jimenez Rezende (2016). "Variational Inference for Monte Carlo Objectives." In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 2188–2196.

Murphy, Kevin Patrick (2002). "Dynamic bayesian networks: representation, inference and learning." In:

Noh, Hyeonwoo, Seunghoon Hong, and Bohyung Han (2015). "Learning deconvolution network for semantic segmentation." In: *Proceedings of the IEEE international conference on computer vision*, pp. 1520–1528.

Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu (2016a). "Pixel recurrent neural networks." In: *arXiv preprint arXiv:1601.06759*.

Oord, Aaron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu (2016b). "Wavenet: A generative model for raw audio." In: *arXiv preprint arXiv:1609.03499*.

Papamakarios, George, Iain Murray, and Theo Pavlakou (2017). "Masked autoregressive flow for density estimation." In: *Advances in Neural Information Processing Systems*, pp. 2338–2347.

Petterson, James, Jin Yu, Julian J McAuley, and Tibério S Caetano (2009). "Exponential family graph matching and ranking." In: *Advances in Neural Information Processing Systems*, pp. 1455–1463.

Prahallad, Kishore, Anandaswarup Vadapalli, Naresh Elluru, G Mantena, B Pulugundla, P Bhaskararao, HA Murthy, S King, V Karaiskos, and AW Black (2013). "The blizzard challenge 2013–indian language task." In: *Blizzard challenge workshop*. Vol. 2013.

Radford, Alec, Luke Metz, and Soumith Chintala (2015). "Unsupervised representation learning with deep convolutional generative adversarial networks." In: *arXiv preprint arXiv:1511.06434*.

Ranganath, Rajesh, Dustin Tran, and David M. Blei (2016). "Hierarchical Variational Models." In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 324–333.

Rezende, Danilo Jimenez and Shakir Mohamed (2015). "Variational Inference with Normalizing Flows." In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 1530–1538.

Salimans, Tim, Andrej Karpathy, Xi Chen, and Diederik P Kingma (2017). "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications." In: *arXiv preprint arXiv:1701.05517*.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (2015). "Neural machine translation of rare words with subword units." In: *arXiv preprint arXiv:1508.07909*.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556*.

Sinkhorn, Richard and Paul Knopp (1967). "Concerning nonnegative matrices and doubly stochastic matrices." In: *Pacific Journal of Mathematics* 21.2, pp. 343–348.

Sønderby, Casper Kaae, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther (2016). "Ladder Variational Autoencoders." In: *Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3738–3746.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks." In: *Advances in neural information processing systems*, pp. 3104–3112.

Tang, Kui, Nicholas Ruozzi, David Belanger, and Tony Jebara (2015). "Bethe learning of conditional random fields via map decoding." In: *arXiv preprint arXiv:1503.01228*.

Wainwright, Martin J, Michael I Jordan, et al. (2008). "Graphical models, exponential families, and variational inference." In: *Foundations and Trends® in Machine Learning* 1.1–2, pp. 1–305.

Wu, Ren, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun (2015). "Deep image: Scaling up image recognition." In: *arXiv preprint arXiv:1501.02876*.

Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. (2016). "Google's neural machine translation system: Bridging the gap between human and machine translation." In: *arXiv preprint arXiv:1609.08144*.

Yang, Zhilin, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen (2017). "Breaking the softmax bottleneck: A high-rank RNN language model." In: *arXiv preprint arXiv:1711.03953*.

Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy (2016). "Hierarchical attention networks for document classification." In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1480–1489.

Yu, Fisher and Vladlen Koltun (2015). "Multi-scale context aggregation by dilated convolutions." In: *arXiv preprint arXiv:1511.07122*.

Zaremba, Wojciech, Ilya Sutskever, and Oriol Vinyals (2014). "Recurrent neural network regularization." In: *arXiv preprint arXiv:1409.2329*.

Zhang, Xiang, Junbo Zhao, and Yann LeCun (2015). "Character-level convolutional networks for text classification." In: *Advances in neural information processing systems*, pp. 649–657.

Zheng, Guoqing, Yiming Yang, and Jaime G. Carbonell (2016). "Efficient Shift-Invariant Dictionary Learning." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 2095–2104. DOI: 10.1145/2939672.2939824. URL: http://doi.acm.org/10.1145/2939672.2939824.

Zheng, Guoqing, Yiming Yang, and Jaime Carbonell (2017a). "Asymmetric Variational Autoencoders." In: *arXiv preprint arXiv:1711.02255*.

— (2017b). "Convolutional Normalizing Flows." In: *arXiv preprint arXiv:1711.02255*.

Zilly, Julian Georg, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber (2016). "Recurrent highway networks." In: *arXiv preprint arXiv:1607.03474*.

Zoph, Barret and Quoc V Le (2016). "Neural architecture search with reinforcement learning." In: *arXiv preprint arXiv:1611.01578*.

Zoph, Barret, Deniz Yuret, Jonathan May, and Kevin Knight (2016). "Transfer learning for low-resource neural machine translation." In: *arXiv preprint arXiv:1604.02201*.

Zou, Will Y, Richard Socher, Daniel Cer, and Christopher D Manning (2013). "Bilingual word embeddings for phrase-based machine translation." In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1393–1398.