Université de Montréal

Improved Training of Generative Models

par Anirudh Goyal

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

July, 2018

Résumé

Cette thèse explore deux idées différentes:

— **Une méhode améliorée d'entraînement de réseaux de neurones récurrents.** Communément, l'entraînement des réseaux de neurones récurrents se fait à l'aide d'une méthode connue sous le nom de 'teacher forcing'. Cette méthode consiste à utiliser les valeurs de la séquence observée en tant qu'entrées du réseau pendant la phase d'entraînement, alors que l'on utilise la séquence des valeurs prédites par le modèle lors de la phase de génération. Nous présentons ici un algorithme appelé 'professor forcing' qui utilise l'adaptation de domaine adversaire pour encourager la dynamique du réseau récurrent à être la même lors de la phase d'entraînement et lors de la phase de génération. Ce travail a été accepté a la session de posters de la conférence NIPS 2016.

— **Un nouveau modèle pour l'entraînement de modèles génératifs**. Un obstacle connu lors de l'entraînement de modèles graphiques non orientés avec variables latentes, tels que les machines de Boltzmann, est que la procédure d'entraînement par maximum de vraisemblance necéssite une chaîne de Markov pour échantillonner. Or le temps de mixage de la chaîne de Markov dans la boucle interne de l'entraînement peut être très long. Dans cette thèse, nous proposons d'abord l'idée qu'il suffit de découper localement la fonction dénergie de sorte que son gradient pointe dans la bonne direction (cest-a-dire vers la génération des données). Cela correspond à une nouvelle procédure d'apprentissage qui s'éloigne d'abord des données en suivant l'opérateur de transition du modèle, et qui ensuite entraîne cet opérateur à revenir en arrière à chaque étape, en revenant vers les données. Ce travail a été accepté en tant que poster à la conference NIPS 2017.

Dans le premier chapître, je présente quelques notions élémentaires sur les modèles génératifs (en particulier les modèles graphiques orientés et non orientés). Je montre en quoi la méthode proposée dans le chapître 3 est liée à ces modèles.

Dans le deuxième chapître, je décris notre méthode proposée (appelée 'professor forcing') pour améliorer l'entraînement des réseaux de neurones récurrents.

Dans le troisième chapître, je décris notre méthode proposée pour entraîner un modèle génératif en paramétrant directement un opérateur de transition.

Mots clés : modèles génératifs, réseaux de neurones récurrents.

Summary

This thesis explores ideas along 2 different directions:

— **Improved Training of Recurrent Neural Networks** - Recurrent Neural Networks are trained using teacher forcing which works by supplying observed sequence values as inputs during training, and using the network's own one-step ahead predictions to do multi-step sampling. We introduce the Professor Forcing algorithm, which uses adversarial domain adaptation to encourage the dynamics of the recurrent network to be the same when training the network and when sampling from the network over multiple time steps. This work was accepted as a conference poster at NIPS 2016.

— **Training iterative generative models** A recognized obstacle to training undirected graphical models with latent variables such as Boltzmann machines is that the maximum likelihood training procedure requires sampling from Monte-Carlo Markov chains which may not mix well, in the inner loop of training, for each example. In this thesis, we first propose the idea that it is sufficient to locally carve the energy function everywhere so that its gradient points in the right direction (i.e., towards generating the data). This corresponds to a new learning procedure that first walks away from data points by following the model transition operator and then trains that operator to walk backwards for each of these steps, back towards the training example. This work was accepted as a conference poster at NIPS 2017.

Chapter One is dedicated to background knowledge about generative models. This covers directed and undirectored graphical models and how the proposed method in Chapter 3 are related to these. In the following chapter, I will describe our proposed method to improve training of recurrent neural networks using Professor Forcing Goyal et al. [2016]. The third chapter describes the Variational Walkback [Goyal et al., 2017a] algorithm. This is an algorithm for training an iterative generative model by directly learns a parameterized transition operator.

Keywords: unsupervised learning, generative models, recurrent neural networks

Table des matières

Résumé . . . . **ii**

Summary . . . . **iii**

Contents . . . . **iv**

List of Figures . . . . **vii**

List of Tables . . . . **xi**

List of Abbreviations . . . . **xii**

Acknowledgments . . . . **xiv**

1 Introduction **1**
- 1.1 Generative Modelling 1
- 1.2 What is a stochastic process ? 2
- 1.3 What is a Markov chain ? 2
  - 1.3.1 What is a stationary distribution of a Markov chain ? 2
  - 1.3.2 Reversibility of a markov chain 3
  - 1.3.3 Sufficient statistic 3
- 1.4 What is Markov Chain Monte Carlo ? 4
- 1.5 Latent variable models 4
- 1.6 Graphical Model 5
- 1.7 Directed and Undirected Graphical Models 5
- 1.8 What is a energy function ? 6
- 1.9 Energy Based Models 7
  - 1.9.1 Maximum Likelihood Training of Undirected Graphical Models 7
- 1.10 Variational Autoencoder 8
  - 1.10.1 The approximate inference model 8
  - 1.10.2 Generative Model of the VAE 9
  - 1.10.3 Reparametrization Trick 9
  - 1.10.4 Variational Inference and Learning 10
- 1.11 Learning a Transition Operator 11
  - 1.11.1 Denoising Autoencoders 11

1.11.2 Walkback Training for Denoising Autoencoders 12
1.11.3 Generative Stochastic Networks 12
1.11.4 Non Equillibrium Thermodynamics 13

2 Professor Forcing: A New way of Training RNN's 14
2.1 Introduction 15
2.2 Proposed Approach: Professor Forcing 17
2.2.1 Definitions and Notation 17
2.2.2 Training Objective 18
2.3 Related Work 19
2.4 Experiments 20
2.4.1 Networks Architecture and Professor Forcing Setup 20
2.4.2 Character-Level Language Modeling 22
2.4.3 Sequential MNIST 23
2.4.4 Handwriting Generation 24
2.4.5 Music Synthesis on Raw Waveforms 25
2.4.6 Negative Results on Shorter Sequences 28
2.5 Conclusion 28

3 Variational Walkback: Learning a Transition Operator as a Stochastic Recurrent Net 29
3.1 Variational Walkback - Connections to Directed and Undirected Graphical Models 30
3.2 Introduction 33
3.3 The Variational Walkback Training Process 35
3.4 Variational Derivation of Walkback 38
3.4.1 Tightness of the variational lower bound 40
3.4.2 Estimating log likelihood via importance sampling 42
3.4.3 VW transition operators and their convergence 42
3.5 Related Work 43
3.6 Experiments 44
3.7 Discussion and Future Work 46
3.8 VW transition operators and their convergence 47
3.9 Additional Results 47
3.10 VW on Toy Datasets 48
3.11 VW chains 48
3.12 Walkback Procedure Details 49
3.13 Higher Lower Bound: not always better samples 54
3.14 Reversibility of transition operator 54
3.15 Some Minor Points 55

3.16 Inception Scores on CIFAR 55

4 Conclusion **64**

Bibliography **67**

Table des figures

2.1 Figure 1: Architecture of the Professor Forcing - Learn correct one-step predictions such as to to obtain the same kind of recurrent neural network dynamics whether in open loop (teacher forcing) mode or in closed loop (generative) mode. An open loop generator that does one-step-ahead prediction correctly. Recursively composing these outputs does multi-step prediction (closed-loop) and can generate new sequences. This is achieved by train a classifier to distinguish open loop (teacher forcing) vs. closed loop (free running) dynamics, as a function of the sequence of hidden states and outputs. Optimize the closed loop generator to fool the classifier. Optimize the open loop generator with teacher forcing. The closed loop and open loop generators share all parameters 21

2.2 Figure 2: Penn Treebank Likelihood Curves in terms of the number of iterations. Training Negative Log-Likelihood (left). Validation BPC (Right) 23

2.3 Figure 3: T-SNE visualization of hidden states, left: with teacher forcing, right: with professor forcing. Red dots correspond to teacher forcing hidden states, while the gold dots correspond to free running mode. At t = 500, the closed-loop and open-loop hidden states clearly occupy distinct regions with teacher forcing, meaning that the network enters a hidden state region during sampling distinct from the region seen during teacher forcing training. With professor forcing, these regions now largely overlap. We computed 30 T-SNEs for Teacher Forcing and 30 T-SNEs for Professor Forcing and found that the mean centroid distance was reduced from 3000.0 to 1800.0, a 40% relative reduction. The mean distance from a hidden state in the training network to a hidden state in the sampling network was reduced from 22.8 with Teacher Forcing to 16.4 with Professor Forcing (vocal synthesis). 24

2.4 Figure 4: Samples with Teacher Forcing (left) and Professor Forcing (right). 25

2.5 Figure 5: Handwriting samples with teacher forcing (top) and professor forcing (bottom). Note that in both cases the model is only trained on observed sequences of length 50 steps (a few letters) but is used to do conditional generation for 1000 steps. 26
2.6 Figure 6: Left: training likelihood curves. Right: validation likelihood curves. 26
2.7 Figure 7: Human evaluator ratings for vocal synthesis samples (higher is better). The height of the bar is the mean of the ratings and the error bar shows the spread of one standard deviation. 27

3.1 Variational WalkBack framework. The generative process is represented in the blue arrows with the sequence of $p_{T_t}(\boldsymbol{s}_{t-1}|\boldsymbol{s}_t)$ transitions. The destructive forward process starts at a datapoint (from $q_{T_0}(\boldsymbol{s}_0)$) and gradually heats it through applications of $q_{T_t}(\boldsymbol{s}_t|\boldsymbol{s}_{t-1})$. Larger temperatures on the right correspond to a flatter distribution, so the whole destructive forward process maps the data distribution to a Gaussian and the creation process operates in reverse. 35
3.2 Generating MNIST-like samples using a Bernoulli likelihood in the transition operator. Left: VW iteratively generates images starting from a noise prior. For intermediate steps we display samples and for the final step (right hand side image) we display the transition operator's mean. 44
3.3 VW inpainting in CelebA images. Images on the left are the ground truth images corrupted for their bottom half (which is the starting point of the chain). The goal is to fill in the bottom half of each face image given an observed top half of an image (drawn from test set). Images on the right show the inpainted lower halves for all these images. 48
3.4 VW samples on MNIST using Gaussian noise in the transition operator. The model is trained with 30 steps of walking away, and samples are generated using 30 annealing steps. 49
3.5 VW samples on CelebA dataset using Gaussian noise in the transition operator. Model is trained using 30 steps to walk away and samples are generated using 30 annealing steps. 50
3.6 VW samples on Cifar10 using Gaussian noise in the transition operator. Model is trained using 30 steps to walk away and samples are generated using 30 annealing steps. 51
3.7 VW samples on SVHN dataset using Gaussian noise in the transition operator. Model is trained using 30 steps to walk away and samples are generated using 30 annealing steps. 52

3.8 The proposed modeling framework trained on 2-d swiss roll data. This algorithm was trained on 2D swiss roll for 30 annealing steps using annealing schedule increasing temperator by 1.1 each time. We have shown every 5th sample (ordering is row wise, and within each row it is column-wise. 53
3.9 The proposed modeling framework trained on circle data. This algorithm was trained on circle for 30 annealing time steps using annealing schedule increasing temperature by factor 1.1 each time. We have shown every 5th sample (ordering is row wise, and within each row it is column-wise. 53
3.10 VW sample chain (vertically, going down) starting from pure noise. Model trained using $K = 30$ steps to walk away and samples are generated using 30 steps of annealing. The figure shows every 3rd sample of the chain in each column. 56
3.11 VW sample chain. Each coloumn above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps at temperature = 1, to demonstrate that even over very long chain, the transition operator continues to generate good samples. 57
3.12 VW sample chain. Each column above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps at temperature = 1, to demonstrate that even over very long chain, the transition operator continues to generate good samples. 58
3.13 VW sample chain. Each column above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps temperature = 1. 59
3.14 VW sample chain. Each column above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps at temperature = 1, to demonstrate that even over very long chain, the transition operator continues to generate good samples. 60
3.15 VW sample chain. Each column above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps at temperature = 1, to demonstrate that even over very long chain, the transition operator continues to generate good samples. 61
3.16 VW sample chain. Each column above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps at temperature = 1, to demonstrate that even over very long chain, the transition operator continues to generate good samples. 62

3.17 Samples from two VW models (left and right) which have a higher lower bound than the one whose samples are shown in Figure 5 (and comparable but slightly better importance sampling estimators of the log-likelihood): yet, the generated samples are clearly not as good, suggesting that either the bound is sometimes not tight enough or that the log-likelihood is not always a clear indicator of sample quality. 63

Liste des tableaux

2.1 Test set negative log-likelihood evaluations 25
2.2 Human Evaluation Study Results for Handwriting 26

3.1 Comparative log-likelihoods estimated by importance sampling. . . 45

List of Abbreviations

VAE	Variational Auto-Encoder
GAN	Generative Adversarial Networks
RNN	Recurrent Neural Networks
VW	Variational Walkback
XE	Cross Entropy

To my grandfather!

Acknowledgments

I am extremely grateful for an amazing set of family, friends and advisors without whom my work would not be possible.

I'm grateful to Prof. Yoshua Bengio for accepting me as a graduate student, for encouraging me to have the right intuitions, and approaching the problem from different perspectives. His guidance has been very important in illuminating the importance of random exploration for research, and in my understanding of, how brains do credit assignment through time.

I'd also like to thank Rosemary Nan Ke, Alex Lamb and Benjamin Scellier for supporting me. They have played a very unique role in my graduate life.

To my many friends, co-authors, co-workers, and colleagues (random order): Adriana Romero, Alessandro Sordoni, Amjad Almahairi, Asja Fischer, Anna Huang, Ahmed Touati, Akram Erraqabi, Arnaud Bergeron, Bart van Merrienboer, Ben Poole, Benjamin Scellier, Caglar Gulcehre, Cesar Laurent, Daniel Jiwoong Im, David Krueger, David Warde-Farley, Devon Hjelm, Dmitriy Serdyuk, Dzmitry Bahdanau, Faruk Ahmed, Francesco Visin, Frederic Bastien, Gabriel Huang, Gaetan Marceau Caron, Guillaume Alain, Harm de Vries, Ishmael Belghazi, Krishna Kumar Singh, Iulian Vlad Serban, Joao Felipe Santos, Jonathan Binas, Jorg Bornschein, Jose Sotelo, Junyoung Chung, Joseph Paul Cohen, Marcin Moczulski, Mathieu Germain, Mehdi Mirza, Mohammad Pezeshski, Myriam Cote, Negar Rostamzadeh, Olexa Bilaniuk, Pascal Lamblin, Pierre-Luc Carrier, Philemon Brakel, Shawn Tan, Simon Lefrancois, Sina Honari, Soroush Mehri, Sungjin Ahn, Tim Cooijmans, Vincent Dumoulin ; Thank you for everything you do.

Thanks Celine Begin for all her patience during the thesis submission process and for her organizational help during my studies.

I would like to thank my undergrad research advisor Prof. C.V Jawahar, for accepting me as a research assitant, and teaching me how to do research. Thanks to Dr. Shailesh Kumar, for advising me during my final year of the undergrad. He helped to encourage me to think more, thanks to all his brainstorming sessions. I learned the importance of having good intuitions from him. I would like to thank Dr. Rahul Sukthankar, for showing faith in me when I was most uncertain about my future. Without him, I would not be able to come to MILA. Thanks to Dr. Ray Kurzweil for giving me a chance to intern in his team. He fueled up my curiosity to understand how the brain works. Thanks to all of you.

I also want to express an extra thanks to my current collaborators: Dr. Hugo Larochelle, Professor Sergey Levine, Professor Konrad Kording, Dr. Matthew Botvinick. Finally, the work reported in this thesis would not have been possible without the financial support from: Ubisoft, Google, Samsung, IBM, NSERC, Calcul Quebec, Compute Canada, the Canada Research Chairs and CIFAR.

Finally, I would like to thank my parents, my grandparents, and other members of my family.

1 Introduction

1.1 Generative Modelling

Given a data distribution X, a generative model refers to classes of models that model the distribution of data $P(X)$ so that samples can be drawn from the approximating distribution. This is different from discriminative models that models $P(Y|X)$ for distributions Y and X. Generative models are an important class of models for several reasons. One reason is that learning generative models can help to capture the salient features in the data, this can in turn help with many other tasks. Some examples of these tasks are classification, semi-supervised learning, denoising, impainting, structured predication and exploration in Reinforcement Learning. One other important reason is that learning generative models does not require human labor to hand annotate the data with labels. Hence, we have access to lot more unsupervised data (data without labels): if we can learn a good generative model from these and capture the salient features, this can help other tasks as we have mentioned earlier.

Given access to a set of samples from an unknown target distribution, a generative model provides a mechanism for producing samples from the distribution learned by the model. A generative model assumes data is created by a particular distribution which is defined by a set of parameters or non-parametric variants and it approximates that underlying distribution with particular algorithms. This gives the ability of generating new data and not only classify it from existing data..

Suppose you are a small child being asked ”What is a bicycle ?”. You can draw it onto a paper showing a primitive bicycle. This is a generative process. This is not a real bicycle, what you draw, but still a bicylce. This is a trace that we have a generative model in our heads. However, when we ask you ”Which one is the bicycle ?”, you are still able to point it versus a car. This is a classification process that we mimic by discriminative modelling.

1.2 What is a stochastic process?

A stochastic process is simply a collection of random variables indexed by time $\{X_u, u \in I\}$, where I is the index set. The set of all times we wish to define the particular process under consideration. At every new unit of time, the random variables could assume one of many possible values, and each sequence of these values has a probability associated with it. While we cannot know the exact path that those random variables will follow, we can make inferences about the path it might take based on those probabilities.

The index set I will be either a discrete or a continuous set. If it is discrete (e.g. $I = \{0, 1, 2, ...\}$) then we say that X is a discrete-time stochastic process. If it is continuous then we say X is a continuous-time stochastic process.

1.3 What is a Markov chain?

A stochastic process is said to be a Markov Chain if it has the Markov Property.

For any s, $i_0, ..., i_{n1} \in S$ and any $n \geq 1$.

$P(X_n = s|X_0 = i_0.........X_{n-1} = i_{n-1}) = P(X_n = s|X_{n-1} = i_{n-1})$

Here, we say that the distribution of X_n given the entire past of the process only depends on the immediate past. Note that we are not saying that, for example X_5 and X_1 are independent. They are not. However, given X_4, for example, X_5 is conditionally independent of X_1. Graphically, we may imagine being on a particle jumping around in the state space as time goes on to form a (random) sample path. The Markov property is that the distribution of where I go to next depends only on where I am now, not on where I have been. This property is a reasonable assumption for many (though certainly not all) real-world processes.

1.3.1 What is a stationary distribution of a Markov chain?

A stochastic process is stationary if for any points $i_1.....i_n$ and $m \geq 0$ the joint distribution of $X_{i_1}.......X_{i_n}$ is the same as the joint distribution of $X_{i_{1+m}}.......X_{i_{n+m}}$. So stationary refers to stationary in time. In particular, for a stationary process, the distribution of X_n is the same for all n.

So why do we care if our Markov chain is stationary ? Well, if it were stationary and we knew what the distribution of each X_n was then we would know a lot because we would know the long run proportion of time that the Markov chain was in any state. For example, suppose that the process was stationary and we knew that $P(X_n = 2) = \frac{1}{10}$ for every n. Then over 1000 time periods we should expect that roughly 100 of those time periods was spent in state 2, and over N time periods roughly $\frac{N}{10}$ of those time periods was spent in state 2. As N went to infinity, the proportion of time spent in state 2 will converge to 1/10 (this can be proved rigorously by some form of the Strong Law of Large Numbers). One of the attractive features of Markov chains is that we can often make them asymptotically stationary (for n sufficiently large) and there is a nice and neat characterization of the distribution of X_n when it is stationary.

1.3.2 Reversibility of a markov chain

A Markov chain is reversible if its transition probability is reversible with respect to its initial distribution. Reversibility implies stationarity, but not vice versa. A reversible Markov chain has the same laws running forward or backward in time, that is, for any i and k the distributions of $(X_{i+1}, ..., X_{i+k})$ and $(X_{i+k}, ..., X_{i+1})$ are the same.

Here is a nice hypothesis test that might be a good way to explain it. The hypothesis test is as follows: If I show you a movie of a time series of a stochastic process. Then I ask you: Is time going forwards in the movie or am I playing the movie backwards ? If you are not able to distinguish between the two, than the process is a reversible process.

1.3.3 Sufficient statistic

Sufficient statistics provide a summary of the data, and are often useful for learning tasks. A statistic is any function $T(X)$ of the data X. if θ parametrizes the class of underlying data-generating distributions, then for any statistic, we have the Markov chain $\theta \to X \to T(X)$ i.e. $\theta \perp T(X)|X$ and data processing inequality tells us that $I(\theta, T(X)) \leq I(\theta, X)$. A statistic is sufficient for a parameter θ if $\theta \perp X|T(X)$, i.e. we also have the Markov chain $\to T(X) \to X$. In words, once we know $T(X)$, the remaining randomness in X does not depend on θ. This implies

$p(X|T(X))$.

1.4 What is Markov Chain Monte Carlo ?

Monte Carlo methods are computational algorithms (simply sets of instructions) which randomly sample from some process under study. They are a way of estimating something which is too difficult or time consuming to find deterministically. They're basically a form of computer simulation of some mathematical or physical process. Monte Carlo integration works great on a high-dimensional functions by taking a random sample of points of the function and calculating some type of average at these various points. By increasing the sample size, the law of large numbers tells us we can increase the accuracy of our approximation by covering more and more of the function.

These two concepts can be put together to solve some difficult problems in areas such as Bayesian inference, computational biology, etc where multi-dimensional integrals need to be calculated to solve common problems. The idea is to construct a Markov Chain which converges to the desired probability distribution after a number of steps. We want to generate random draws from a target distribution. We then identify a way to construct a 'nice' Markov chain such that its equilibrium probability distribution is our target distribution of interest. If we can construct such a chain then we arbitrarily start from some point and iterate the Markov chain many times. Eventually, the draws we generate would appear as if they are coming from our target distribution.

1.5 Latent variable models

A latent variable model is a model that contains latent i.e unobserved variables. Given a data distribution, there might be very complex interdependencies between the dimensions. Modelling all of these interdependencies is costly, and makes training much harder in a high dimensional data space. A simple way to make this easier is to assume there is a (low-dimensional) latent variable z that induces global structure on the samples, thus freeing us from modelling complex interactions among the observed random variables in order to capture high level structure, since the observed random variables are conditionally independent given the latent variables. For example, to generate a natural image, the latent variable might encode hidden causes behind the image such as the object categories, their positions and orientations, etc., and then a generator only needs to decode this high level information into a set of pixels that visually represents it. The variable z is referred to as latent because we never see these variables in the training data, only the observed random variables x are seen. To deduce our model latent z corresponding to an observed x we need to perform inference. If we assume that the model is parameterized by θ, we would like to train the model such that on average, the samples produced by the model by sampling from a prior over z, $p(z)$, would match the data distribution. Formally, this would maximize $p(x) = \int p(x|z;)p(z)dz$ which is referred to as the maximum likelihood training principle for latent variable models.

1.6 Graphical Model

A graphical model is a probabilistic model for which a graph expresses the conditional dependence structure between random variables.

1.7 Directed and Undirected Graphical Models

Directed versions of such generative models can *synthesize* new sensory data through a sampling process which often converts a simple distribution over latent variables, modelling causes in the sensory data, into complex distributions over the data itself. They can also *analyze* sensory data by computing the posterior distribution over latent variables given sensory data. An early instantiation of this idea was the Helmholtz machine [Dayan et al., 1995b], in which the analysis was performed by a *recognition* model and the synthesis was performed by a separate generative model, and the two were trained together to maximize the marginal probability of the data. More recent work on directed generative modelling using a small number of generative steps involves variational auto-encoders (VAEs) [Kingma and Welling, 2013, Rezende et al., 2014], while non-probabilistic approaches to training include generative adversarial networks (GANs) [Goodfellow et al., 2014]. In the probabilistic VAE approach however, it can be difficult to learn very deep models, with multiple layers of abstraction intervening between latent causes and sensory data. The *credit assignment problem* stands as a major impediment to learning such deep models as it can be difficult to optimize parameters controlling latent variables far removed from the data.

A parallel strand of generative modeling, through undirected probabilistic models, involves modelling the data as the *stationary* distribution of a stochastic process (e.g. various Boltzmann machines [Salakhutdinov and Hinton, 2009]). Sampling under this method corresponds to a potentially powerful iterative process of repeatedly applying a fixed stochastic operator that can gradually turn simple initial distributions over data into complex stationary distributions over data. However a key impediment to this approach is the *mixing time problem*: if the stationary distribution has multiple modes, the sampling process can take a long time to mix, or reach the stationary distribution, due to the excessive time sampling methods can take to jump between modes. Moreover, these models are also difficult to train, as solving the credit assignment problem also involves sampling from the model, in order to match the model's stationary distribution to the data distribution. Thus the mixing time problem contributes to the difficulty of the credit assignment difficult in undirected models.

1.8 What is a energy function ?

The role of energy function in generative models is the same as that of probability. It is a property of the system in current configuration. Every configuration of variables correspond to an energy value.

1.9 Energy Based Models

Learning any distribution over data involves two fundamental problems:

— Model must place probability mass where the data is located.

— Model must avoid probability mass where the data is not located.

Probability modes of the model distribution where there is no data are known as spurious modes, and a fundamental goal of learning is to hunt down these spurious modes and remove them. In this thesis, we proposed a fundamental way of hunting down these spurious modes.

1.9.1 Maximum Likelihood Training of Undirected Graphical Models

Let $\boldsymbol{v}$ denote the vector of visible units which takes as value an observed data exampled and $\boldsymbol{h}$ denote the vector of hidden random variables, with the full state of the model being $\boldsymbol{s} = (\boldsymbol{v}, \boldsymbol{h})$. Let $p_{\boldsymbol{\theta}}$ denote the model distribution, with joint energy function $E_{\boldsymbol{\theta}}$ and parameter vector $\boldsymbol{\theta}$:

$$p_{\boldsymbol{\theta}}(\boldsymbol{s}) := \frac{e^{-E_{\boldsymbol{\theta}}(\boldsymbol{s})}}{Z_{\boldsymbol{\theta}}}, \tag{1.1}$$

where Z_θ is the partition function

$$Z_\theta := \int e^{-E_{\boldsymbol{\theta}}(\boldsymbol{s})} ds. \tag{1.2}$$

Let $p_{\mathcal{D}}$ be the unknown training distribution, from which a sample $\mathcal{D}$ is typically drawn to obtain the training set. The maximum likelihood parameter gradient is

$$\mathbb{E}_{\boldsymbol{v}\sim p_{\mathcal{D}}}\left[-\frac{\partial \log p_{\boldsymbol{\theta}}(\boldsymbol{v})}{\partial \boldsymbol{\theta}}\right] = \mathbb{E}_{\boldsymbol{v}\sim p_{\mathcal{D}}, \boldsymbol{h}\sim p_{\boldsymbol{\theta}}(\boldsymbol{h}|\boldsymbol{v})}\left[\frac{\partial E_{\boldsymbol{\theta}}(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}}\right] - \mathbb{E}_{\boldsymbol{s}\sim p_{\boldsymbol{\theta}}(\boldsymbol{s})}\left[\frac{\partial E_{\boldsymbol{\theta}}(\boldsymbol{s})}{\partial \boldsymbol{\theta}}\right] \tag{1.3}$$

which is zero when training has converged, with expected energy gradients in the positive phase (under $p_{\mathcal{D}}(\boldsymbol{v})p_{\boldsymbol{\theta}}(\boldsymbol{h}|\boldsymbol{v})$) matching those under the negative phase (under $p_{\boldsymbol{\theta}}(\boldsymbol{s})$). Training thus consists in matching the shape of two distributions, as captured by the sufficient statistics: the positive phase distribution $p_{\boldsymbol{\theta}}(\boldsymbol{h}|\boldsymbol{v})p_{\mathcal{D}}(\boldsymbol{v})$ (influenced by the data, via the visible) and the negative phase distribution $p_{\boldsymbol{\theta}}(\boldsymbol{h}|\boldsymbol{v})$ (where the model is free-running and generating configurations by itself).

A recognized obstacle to training undirected graphical models with latent variables such as Boltzmann machines is that the maximum likelihood training procedure requires sampling from Monte-Carlo Markov chains which may not mix well, in the inner loop of training.

1.10 Variational Autoencoder

The Variational Autoencoder (VAE) Kingma and Welling [2013] is a neural network-based approach to latent variable modeling where the richly-structured dependencies found in the data are disentangled into the relatively simple dependencies between a set of latent variables. Formally, let x be a random real-valued vector representing the observed data and let z be a random real-valued vector representing the latent variables that reflect the principle directions of variation in the input data.

1.10.1 The approximate inference model

In general, inference in directed graphical models with latent variables is intractable. Variational Autoencoders deal this issue by using a learned approximate posterior distribution $q_\phi(z|x)$ parameterized by another neural network ϕ called encoder network $g(z)$. The choice of approximate inference model is crucial in variational inference. The family of approximate posterior distributions should be rich enough, so that it could contain the true posterior distribution. Finding a way to better approximate posterior distribution is a hot research area. One possible solution towards improving the approximate distribution is using **normalizing flows** Rezende and Mohamed [2015]. It basically describes the transformation of a probability distribution through a sequence of invertible mappings. Now, the idea is you start with a valid probability distribution in the beginning like normal distribution and you apply these transformations, and at the end of this, you obtain a valid probability distribution.

1.10.2 Generative Model of the VAE

We specify the generative model over the pair (x, z) as $p_\theta(x, z) = p_\theta(x|z)p_\theta(z)$, where $p_\theta(z)$ is the prior distribution over the latent variables and $p_\theta(x|z)$ is the conditional likelihood of the data given the latents. θ represents the generative model parameters. As is typical in the VAE framework, we assume a standard Normal (Gaussian) prior distribution over z: $p_\theta(z) = N(z|0, I)$. For real-valued data such as natural images, by far the most common conditional likelihood is the Gaussian distribution: $p(x|z) = \mathcal{N}(x|\mu_\theta(z), diag(\sigma_\theta^2))$, where the mean $\mu_x(z)$ is a nonlinear function of the latent variables specified by a neural network, which following autoencoder terminology, we refer to as the decoder network $f(x)$.

Most of applications on real-valued data use a multivariate Gaussian with diagonal co-variance matrix as the conditional likelihood of the data given the latent variables, which corresponds to minimizing element wise reconstruction penalty. One problem with this is that, the element-wise distance metrics are a poor fit for human notions of similarity.

It is straightforward to see that having a more powerful $p(x|z)$ would boost the generative power of VAE's. This idea has been explored while applying VAE's to sequences, where the decoding distribution $p(x|z)$ is a RNN with strong autoregressive dependencies i.e $p(x|z) = \prod_i p(x_i|z, x_{<i})$

However, it has been empirically observed that it is difficult to take use of latent variables in VAE's when you have a strong decoder network like an autoregressive model as a decoder network, $p(x|z)$. Since, early in the training, your approximate inference model would be weak i.e it would carry little information about x i.e your data points, so what happens in practice is that model sets $q_\phi(z|x)$ to the prior so that to avoid paying any KL cost i.e $D_{KL}(q(z|x)||p(z))$.

1.10.3 Reparametrization Trick

One interesting thing to think about is, how to propagate gradients from the decoder network to the encoder network. Since, normally if z is sampled from the encoder network, you can't easily pass the gradients through this sampling procedure. With real valued z, we can exploit a reparameterization trick Kingma and Welling [2013] to propagate the gradient from the decoder network to the encoder network. Instead of sampling directly from $q_\phi(z|x)$, z is computed as a deterministic function of x and some noise term $\epsilon \in N(0, I)$ such that z has the desired distribution. For instance, if

$$q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), diag(\sigma_\phi^2(x))) \tag{1.4}$$

then we would express z as

$$z = \mu_\phi(x) + \sigma_\phi(x) * \epsilon \tag{1.5}$$

1.10.4 Variational Inference and Learning

Let x be a set of observed variables, z be a set of latent variables and let $p(x, z)$ be their joint distribution. Given a set of examples in a dataset $X_1, X_2, X_3 ......X_N$, we want to maximize the marginal likelihood of the parameters i.e to maximize

$$\log(p(\boldsymbol{X})) = \sum_{n=1}^{N} \log(p(\boldsymbol{X}_i)) \tag{1.6}$$

but in general, this marginal likelihood is intractable to compute for generative models parametrized by neural networks. A solution is to introduce an inference network $q_\phi(z)$ defined over latent variables and instead optimize the variational lower bound on the marginal log likelihood of each observation in the training dataset.

$$\log(p(\boldsymbol{X})) \leq \mathbb{E}_q(z|x)[\log(p(x, z)) - \log(q(z|x))] = L(x, \theta) \tag{1.7}$$

where θ indicates the parameters of p and q models.

$$L(x, \theta) = \log(p(X)) - D_{KL}(q(z|x)||p(z|x)) \tag{1.8}$$

Since the KL divergence is non negative, it is clear that $L(x, \theta)$ is a lower bound on the marginal log likelihood of X i.e $\log(p(X))$.

There are various ways to optimize the lower bound $L(x, \theta)$ for continuous z it can be done efficiently using a trick called reparametrization trick of $q(z|x)$, described above.

As can be seen from equation (1.3), maximizing $L(x, \theta)$ w.r.t. θ will also maximize log p(x) and minimize $D_{KL}(q(z|x)||p(z|x))$. The closer $D_{KL}(q(z|x)||p(z|x))$ is to 0, the closer $L(x,\theta)$ will be to $\log(p(x))$, and the better an approximation to our optimization objective $L(x,\theta)$ is to our true objective $\log(p(x))$. Also, minimization of $D_{KL}(q(z|x)||p(z|x))$ can be a goal in itself, if we are interested in using $q(z|x)$ for inference after optimization. In any case, the divergence $D_{KL}(q(z|x)||p(z|x))$ is a function of our parameters through both the inference model and the generative model, and increasing the flexibility of either is generally helpful towards our objective. We will discuss later how we can increase the flexibility of both the inference model and the generative model.

1.11 Learning a Transition Operator

A transition operator maps the previous-state distribution to a next-state distribution, and is implemented by a stochastic transformation which from the previous state of a Markov chain generates the next state.

1.11.1 Denoising Autoencoders

Denoising Autoencoders transform the problem of learning the true data distribution $P(X)$ as a supervised learning problem, which is a relatively easier problem to solve. The basic approach is as follows: given a data example X, obtain a corrupted version $\widetilde{X}$ by sampling from some corruption distribution $C(\widetilde{X}|X)$. For ex. we take a clean image, X, and add random white noise to produce $\widetilde{X}$.

We then use supervised learning methods to train a function to reconstruct, as accurately as possible, any X from the data set given only a noisy version $\widetilde{X}$. The reconstruction distribution $P(X|\widetilde{X})$ may often be much easier to learn than the data distribution P(X), because $P(X|\widetilde{X})$ tends to be dominated by a single or few major modes.

But how does learning the reconstruction distribution help us solve our original problem of modeling P(X) ? The two problems are clearly related, because if we knew everything about P(X), then with our knowledge of the $C(\widetilde{X}|X)$ would allow us to precisely specify the optimal reconstruction function via Bayes rule: $P(X|\widetilde{X}) = \frac{C(\widetilde{X}|X)P(X)}{z}$ where z is a normalizing constant that does not depend on X. As one might hope, the relation is also true in the opposite direction: once we pick a method of adding noise, $C(\widetilde{X}|X)$, knowledge of the corresponding reconstruction distribution $P(X|(\widetilde{X})$ is sufficient to recover the density of the data P(X).

1.11.2 Walkback Training for Denoising Autoencoders

One issue with using trained DAEs as generative models is that the region visited during training by a traditional DAE is limited to a close neighborhood of the true data distribution. Sampling in high-dimensional spaces using a simple local corruption process (such as Gaussian noise) suggests that if the corruption is too local, the DAEs behavior far from the training examples can create spurious modes in the regions insufficiently visited during training.

Walkback training exploits the knowledge of the currently learned model $P(X|\widetilde{X})$ to define the corruption, so as to pick values of $\widetilde{X}$ that would be obtained by following the generative chain: wherever the model would go if we sampled using the generative Markov chain starting at a training example X. In this way, walkback training can hunt down the spurious modes of the generative model and results in faster training.

1.11.3 Generative Stochastic Networks

The Generative Stochastic Networks (GSN) algorithm proposed by Bengio et al. [2013b] learns a transition operator by iteratively injecting noise and minimizing the reconstruction error after a number of transition operator steps starting at a data point, and back-propagating through all these steps. One thing in common is the idea of using the walkback intuition instead of isotropic noise in order to converge more efficiently. GSN training involves the minimization of overall reconstruction error (from the input data point x to the sampled reconstruction many steps later). This will tend to blur the learned distribution. Instead, the proposed algorithm in this thesis, minimizes reconstruction error one step at a time along the walk-away trajectory. In addition, the GSNs require back-propagating through all the iterated steps, like the DRAW algorithm (Gregor et al., 2015). Instead the proposed algorithm only requires back-propagating through a single step at a time of the transition operator. This should make it easier to train because we avoid having to optimize a highly non-linear transformation obtained by the composition of many transition operator steps.

1.11.4 Non Equillibrium Thermodynamics

Deep Unsupervised Learning using Nonequilibrium Thermodynamics (NET) introduces another way to train a generative model. They used an iterative diffusion process that slowly destroy the structure of the true data distribution. And then they learn a generator network (parameterized as neural network) to learn the reverse process that restructures the data from the unstructured noise. Learning is accomplished by maximizing a lower bound on the log-likelihood with a proposal distribution q used as diffusion process.

With NET, the destructive process corresponds to a fixed, finite time diffusion process in data space. In NET, since the diffusion process is fixed and therefore has no knowledge of the generative process, it is not at all clear that diffusion efficiently seeks out and thereby enables removal of the spurious modes of the reverse, generative process.

2 Professor Forcing: A New way of Training RNN's

Professor Forcing: A New Algorithm for Training Recurrent Networks. Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron Courville and Yoshua Bengio

This chapter presents a joint work with Alex Lamb, Ying Zhang, Saizhang Zhang, Aaron Courville, and Yoshua Bengio. It was published at Advances in Neural Information Processing Systems 29 (NIPS 2016) (Conference Track).

Contribution: I came up with the idea in a discussion with Alex Lamb. I was responsible for writing the initial version of the code, and having the proof of concept that the idea works. I did char level language modelling experiments on PTB and speech modelling experiments. Handwriting generation experiments were done by Alex Lamb. MNIST experiments were done by both Alex and me. Prof. Yoshua Bengio wrote the introduction as well as method description. Alex and me wrote the experimental section. Saizhang Zhang drew the figure associated in the paper. Ying Zhang did experiments on speech recognition but they are not included in this paper.

Affiliation

— Anirudh Goyal, MILA, University of Montreal

— Alex Lamb, MILA, University of Montreal

— Ying Zhang, MILA, University of Montreal

— Saizheng Zhang, MILA, University of Montreal

— Aaron C. Courville, University of Montreal

— Yoshua Bengio, MILA, University of Montreal

2.1 Introduction

Recurrent neural networks (RNNs) have become to be the generative models of choice for sequential data with impressive results in language modeling [Mikolov, 2010, Mikolov and Zweig, 2012], speech recognition [Bahdanau et al., 2015, Chorowski et al., 2015], Machine Translation [Cho et al., 2014b, Sutskever et al., 2014, Bahdanau et al., 2014], handwriting generation [Graves, 2013], image caption generation [Xu et al., 2015, Chen and Lawrence Zitnick, 2015], etc.

The RNN models the data via a fully-observed directed graphical model: it decomposes the distribution over the discrete time sequence $y_1, y_2, \dots y_T$ into an ordered product of conditional distributions over tokens

$$P(y_1, y_2, \dots y_T) = P(y_1) \prod_{t=1}^{T} P(y_t \mid y_1, \dots y_{t-1}).$$

By far the most popular training strategy is via the maximum likelihood principle. In the RNN literature, this form of training is also known as *teacher forcing* [Williams and Zipser, 1989], due to the use of the ground-truth samples y_t being fed back into the model to be conditioned on for the prediction of later outputs. These fed back samples force the RNN to stay close to the ground-truth sequence.

When using the RNN for prediction, the ground-truth sequence is not available conditioning and we sample from the joint distribution over the sequence by sampling each y_t from its conditional distribution given the previously generated samples. Unfortunately, this procedure can result in problems in generation as small prediction error compound in the conditioning context. This can lead to poor prediction performance as the RNN's conditioning context (the sequence of previously generated samples) diverge from sequences seen during training.

Recently, [Bengio et al., 2015a] proposed to remedy that issue by mixing two kinds of inputs during training: those from the ground-truth training sequence and those generated from the model. However, when the model generates several consecutive y_t's, it is not clear anymore that the correct target (in terms of its distribution) remains the one in the ground truth sequence. This is mitigated in various ways, by making the self-generated subsequences short and annealing the probability of using self-generated vs ground truth samples. However, as remarked by Huszár [2015], scheduled sampling yields a biased estimator, in that even as the number of examples and the capacity go to infinity, this procedure may not converge to the correct model. It is however good to note that experiments with scheduled sampling clearly showed some improvements in terms of the robustness of the generated sequences, suggesting that something indeed needs to be fixed (or replaced) with maximum-likelihood (or teacher forcing) training of generative RNNs.

In this paper, we propose an alternative way of training RNNs which explicitly seeks to make the generative behavior and the teacher-forced behavior match as closely as possible. This is particularly important to allow the RNN to continue generating robustly well beyond the length of the sequences it saw during training. More generally, we argue that this approach helps to better model long-term dependencies by using a training objective that is not solely focused on predicting the next observation, one step at a time.

Our work provides the following contributions regarding this new training framework:

— We introduce a novel method for training generative RNNs called Professor Forcing, meant to improve long-term sequence sampling from recurrent networks. We demonstrate this with human evaluation of sample quality by performing a study with human evaluators.

— We find that Professor Forcing can act as a regularizer for recurrent networks. This is demonstrated by achieving improvements in test likelihood on character-level Penn Treebank, Sequential MNIST Generation, and speech synthesis. Interestingly, we also find that training performance can also be improved, and we conjecture that it is because longer-term dependencies can be more easily captured.

— When running an RNN in sampling mode, the region occupied by the hidden states of the network diverges from the region occupied when doing teacher forcing. We empirically study this phenomenon using T-SNEs and show that it can be mitigated by using Professor Forcing.

— In some domains the sequences available at training time are shorter than the sequences that we want to generate at test time. This is usually the case in long-term forecasting tasks (climate modeling, econometrics). We show how using Professor Forcing can be used to improve performance in this setting. Note that scheduled sampling cannot be used for this task, because it still uses the observed sequence as targets for the network.

2.2 Proposed Approach: Professor Forcing

The basic idea of Professor Forcing is simple: while we do want the generative RNN to match the training data, we also want the behavior of the network (both in its outputs and in the dynamics of its hidden states) to be indistinguishable whether the network is trained with its inputs clamped to a training sequence (teacher forcing mode) or whether its inputs are self-generated (free-running generative mode). Because we can only compare the distribution of these sequences, it makes sense to take advantage of the generative adversarial networks (GANs) framework [Goodfellow et al., 2014] to achieve that second objective of matching the two distributions over sequences (the one observed in teacher forcing mode vs the one observed in free-running mode).

Hence, in addition to the generative RNN, we will train a second model, which we call the discriminator, and that can also process variable length inputs. In the experiments we use a bidirectional RNN architecture for the discriminator, so that it can combine evidence at each time step t from the past of the behavior sequence as well as from the future of that sequence.

2.2.1 Definitions and Notation

Let the training distribution provide $(\boldsymbol{x}, \boldsymbol{y})$ pairs of input and output sequences (possibly there are no inputs at all). An output sequence $\boldsymbol{y}$ can also be generated by the generator RNN when given an input sequence $\boldsymbol{x}$, according to the sequence to sequence model distribution $P_{\boldsymbol{\theta}_g}(\boldsymbol{y}|\boldsymbol{x})$. Let $\boldsymbol{\theta}_g$ be the parameters of the generative RNN and $\boldsymbol{\theta}_d$ be the parameters of the discriminator. The discriminator is trained as a probabilistic classifier that takes as input a behavior sequence derived from the generative RNN's activity (hiddens and outputs) when it either generates or is constrained by a sequence $\boldsymbol{y}$, possibly in the context of an input sequence $\boldsymbol{x}$ (often but not necessarily of the same length). The behavior sequence is either the result of running the generative RNN in teacher forcing mode (with $\boldsymbol{y}$ from a training sequence with input $\boldsymbol{x}$), or in free-running mode (with $\boldsymbol{y}$ self-generated according to $P_{\boldsymbol{\theta}_g}(\boldsymbol{y}|\boldsymbol{x})$, with $\boldsymbol{x}$ from the training sequence). The function $B(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}_g)$ outputs the behavior sequence (chosen hidden states and output values) given the appropriate data (where $\boldsymbol{x}$ always comes from the training data but $\boldsymbol{y}$ either comes from the data or is self-generated). Let $D()$ be the output of the discriminator, estimating the probability that was produced in teacher-forcing mode, given that half of the examples seen by the discriminator are generated in teacher forcing mode and half are generated in the free-running mode.

Note that in the case where the generator RNN does not have any conditioning input, the sequence $\boldsymbol{x}$ is empty. Note also that the generated output sequences could have a different length then the conditioning sequence, depending of the task at hand.

2.2.2 Training Objective

The discriminator parameters $\boldsymbol{\theta}_d$ are trained as one would expect, i.e., to maximize the likelihood of correctly classifying a behavior sequence:

$$C_d(\boldsymbol{\theta}_d|\boldsymbol{\theta}_g) = E_{(\boldsymbol{x},\boldsymbol{y})\sim \text{data}}[-\log D(B(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}_g), \boldsymbol{\theta}_d) + E_{\boldsymbol{y}\sim P_{\boldsymbol{\theta}_g}(\boldsymbol{y}|\boldsymbol{x})}[-\log(1-D(B(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}_g), \boldsymbol{\theta}_d)]]. \tag{2.1}$$

Practically, this is achieved with a variant of stochastic gradient descent with minibatches formed by combining N sequences obtained in teacher-forcing mode and N sequences obtained in free-running mode, with $\boldsymbol{y}$ sampled from $P_{\boldsymbol{\theta}_g}(\boldsymbol{y}|\boldsymbol{x})$. Note also that as $\boldsymbol{\theta}_g$ changes, the task optimized by the discriminator changes too, and it has to track the generator, as in other GAN setups, hence the notation $C_d(\boldsymbol{\theta}_d|\boldsymbol{\theta}_g)$.

The generator RNN parameters $\boldsymbol{\theta}_g$ are trained to (a) maximize the likelihood of the data and (b) fool the discriminator. We considered two variants of the latter. The negative log-likelihood objective (a) is the usual teacher-forced training criterion for RNNs:

$$NLL(\boldsymbol{\theta}_g) = E_{(\boldsymbol{x},\boldsymbol{y})\sim \text{data}}[-\log P_{\boldsymbol{\theta}_g}(\boldsymbol{y}|\boldsymbol{x})]. \tag{2.2}$$

Regarding (b) we consider a training objective that only tries to change the free-running behavior so that it better matches the teacher-forced behavior, considering the latter fixed:

$$C_f(\boldsymbol{\theta}_g|\boldsymbol{\theta}_d) = E_{\boldsymbol{x}\sim\text{data},\boldsymbol{y}\sim P_{\boldsymbol{\theta}_g}(\boldsymbol{y}|\boldsymbol{x})}[-\log D(B(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\theta}_g),\boldsymbol{\theta}_d)]. \tag{2.3}$$

In addition (and optionally), we can ask the teacher-forced behavior to be indistinguishable from the free-running behavior:

$$C_t(\boldsymbol{\theta}_g|\boldsymbol{\theta}_d) = E_{(\boldsymbol{x},\boldsymbol{y})\sim\text{data}}[-\log(1 - D(B(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\theta}_g),\boldsymbol{\theta}_d))]. \tag{2.4}$$

In our experiments we either perform stochastic gradient steps on $NLL + C_f$ or on $NLL + C_f + C_t$ to update the generative RNN parameters, while we always do gradient steps on C_d to update the discriminator parameters.

2.3 Related Work

Professor Forcing is an adversarial method for learning generative models that is closely related to Generative Adversarial Networks [Goodfellow et al., 2014] and Adversarial Domain Adaptation Ajakan et al. [2014], Ganin et al. [2015]. Our approach is similar to generative adversarial networks (GANs) because both use a discriminative classifier to provide gradients for training a generative model. However, Professor Forcing is different because the classifier discriminates between hidden states from sampling mode and teacher forcing mode, whereas the GAN's classifier discriminates between real samples and generated samples. One practical advantage of Professor Forcing over GANs is that Professor Forcing can be used to learn a generative model over discrete random variables without requiring to approximate backpropagation through discrete spaces Bengio et al. [2013].

The Adversarial Domain Adaptation uses a classifier to discriminate between the hidden states of the network with inputs from the source domain and the hidden states of the network with inputs from the target domain. However this method was not applied in the context of generative models, more specifically, was not applied to the task of improving long-term generation from recurrent networks.

Alternative non-adversarial methods have been explored for improving long-term generation from recurrent networks. The scheduled sampling method Bengio et al. [2015a], which is closely related to SEARN [Daumé et al., 2009] and DAGGER Ross et al. [2010], involves randomly using the network's predictions as its inputs (as in sampling mode) with some probability that increases over the course of training. This forces the network to be able to stay in a reasonable regime when receiving the network's predictions as inputs instead of observed inputs. While Scheduled Sampling shows improvement on some tasks, it is not a consistent estimation strategy. This limitation arises because the outputs sampled from the network could correspond to a distribution that is not consistent with the sequence that the network is trained to generate. This issue is discussed in detail in Huszár [2015]. A practical advantage of Scheduled Sampling over Professor Forcing is that Scheduled Sampling does not require the additional overhead of having to train a discriminator network.

Actor-critic methods have also been explored for improving modeling of long-term dependencies in generative recurrent neural networks Goyal et al. [2017b], Bahdanau et al. [2016].

Finally, the idea of matching the behavior of the model when it is generating in a free-running way with its behavior when it is constrained by the observed data (being clamped on the ”visible units”) is precisely that which one obtains when zeroing the maximum likelihood gradient on undirected graphical models with latent variables such as the Boltzmann machine. Training Boltzmann machines amounts to matching the sufficient statistics (which summarize the behavior of the model) in both ”teacher forced” (positive phase) and ”free-running” (negative phase) modes.

2.4 Experiments

2.4.1 Networks Architecture and Professor Forcing Setup

The neural networks and Professor Forcing setup used in the experiments is the following. The generative RNN has single hidden layer of gated recurrent units (GRU), previously introduced by [Cho et al., 2014a] as a computationally cheaper alternative to LSTM units [Hochreiter and Schmidhuber, 1997]. At each time step, the generative RNN reads an element $\boldsymbol{x}_t$ of the input sequence (if any) and an

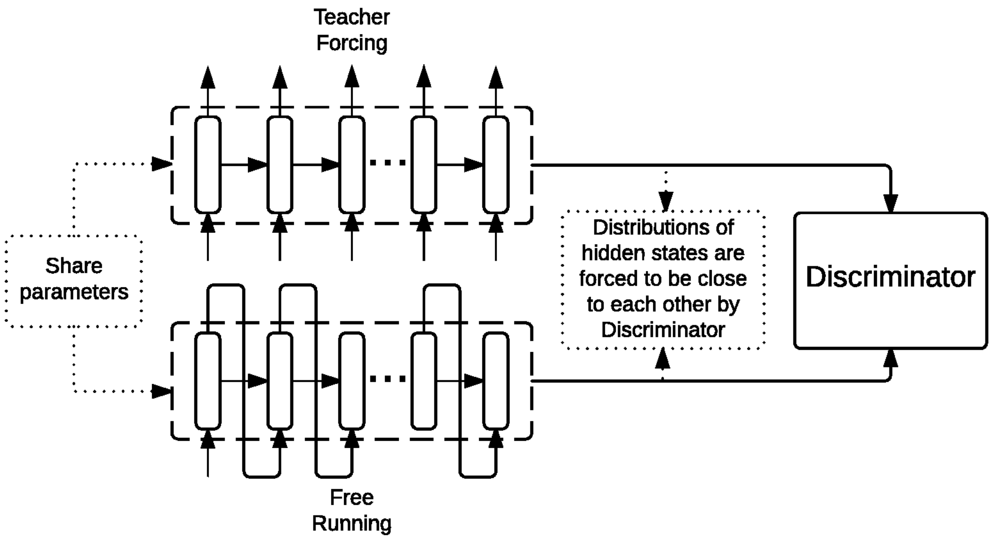
Teacher Forcing
Share parameters
Distributions of hidden states are forced to be close to each other by Discriminator
Discriminator
Free Running

Figure 2.1 – Figure 1: Architecture of the Professor Forcing - Learn correct one-step predictions such as to to obtain the same kind of recurrent neural network dynamics whether in open loop (teacher forcing) mode or in closed loop (generative) mode. An open loop generator that does one-step-ahead prediction correctly. Recursively composing these outputs does multi-step prediction (closed-loop) and can generate new sequences. This is achieved by train a classifier to distinguish open loop (teacher forcing) vs. closed loop (free running) dynamics, as a function of the sequence of hidden states and outputs. Optimize the closed loop generator to fool the classifier. Optimize the open loop generator with teacher forcing. The closed loop and open loop generators share all parameters

element of the output sequence $\boldsymbol{y}_t$ (which either comes from the training data or was generated at the previous step by the RNN). It then updates its state $\boldsymbol{h}_t$ as a function of its previous state $\boldsymbol{h}_{t-1}$ and of the current input $(\boldsymbol{x}_t, \boldsymbol{y}_t)$. It then computes a probability distribution $P_{\theta_g}(\boldsymbol{y}_{t+1}|\boldsymbol{h}_t) = P_{\theta_g}(\boldsymbol{y}_{t+1}|\boldsymbol{x}_1, \dots, \boldsymbol{x}_t, \boldsymbol{y}_1, \dots, \boldsymbol{y}_t)$ over the next element of the output. For discrete outputs this is achieved by a softmax / affine layer on top of $\boldsymbol{h}_t$, with as many outputs as the size of the set of values that $\boldsymbol{y}_t$ can take. In free-running mode, $\boldsymbol{y}_{t+1}$ is then sampled from this distribution and will be used as part of the input for the next time step. Otherwise, the ground truth $\boldsymbol{y}_t$ is used.

The behavior function B used in the experiments outputs the pre-tanh activation of the GRU states for the whole sequence considered, and optionally the softmax outputs for the next-step prediction, again for the whole sequence.

The discriminator architecture we used for these experiments is based on a bidirectional recurrent neural network, which comprises two RNNs (again, two

GRU networks), one running forward in time on top of the input sequence , and one running backwards in time, with the same input. The hidden states of these two RNNs are concatenated at each time step and fed to a multi-layer neural network shared across time (the same network is used for all time steps). That MLP has three layers, each composing an affine transformation and a rectifier (ReLU). Finally, the output layer composes an affine transformation and a sigmoid that outputs $D()$.

When the discriminator is too poor, the gradient it propagates into the generator RNN could be detrimental. For this reason, we back-propagate from the discriminator into the generator RNN only when the discriminator classification accuracy is greater than 75%. On the other hand, when the discriminator is too successful at identifying fake inputs, we found that it would also hurt to continue training it. So when its accuracy is greater than 99%, we do not update the discriminator.

Both networks are trained by minibatch stochastic gradient descent with adaptive learning rates and momentum determined by the Adam algorithm [Kingma and Ba, 2014]. All of our experiments were implemented using the Theano framework [Al-Rfou et al., 2016].

2.4.2 Character-Level Language Modeling

We evaluate Professor Forcing on character-level language modeling on Penn-Treebank corpus, which has an alphabet size of 50 and consists of 5059k characters for training, 396k characters for validation and 446k characters for test. We divide the training set into non-overlapping sequences with each length of 500. During training, we monitor the negative log-likelihood (NLL) of the output sequences. The final model are evaluated by bits-per-character (BPC) metric. The generative RNN implements an 1 hidden layer GRU with 1024 hidden units. We use Adam algorithm for optimization with a learning rate of 0.0001. We feed both the hidden states and char level embeddings into the discriminator. All the layers in the discriminator consists of 2048 hidden units. Output activation of the last layer is clipped between -10 and 10. We see that training cost of Professor Forcing network decreases faster compared to teacher forcing network. The training time of our model is 3 times more as compared to teacher forcing, since our model includes sampling phase, as well as

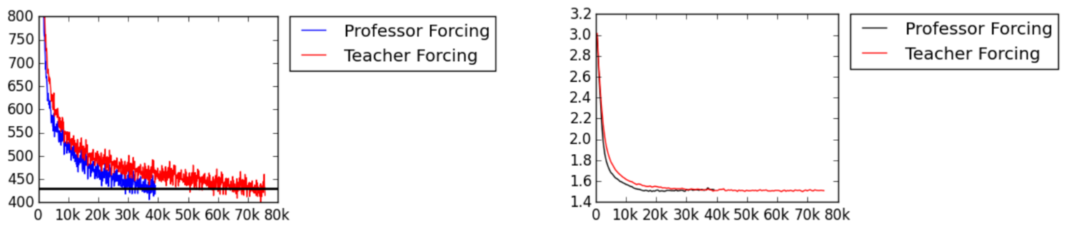

Figure 2.2 – Figure 2: Penn Treebank Likelihood Curves in terms of the number of iterations. Training Negative Log-Likelihood (left). Validation BPC (Right)

passing the hidden distributions corresponding to free running and teacher forcing phase to the discriminator. The final BPC on validation set using our baseline was 1.50 while using professor forcing it is 1.48.

On word level Penn Treebank we did not observe any difference between Teacher Forcing and Professor Forcing. One possible explanation for this difference is the increased importance of long-term dependencies in character-level language modeling.

2.4.3 Sequential MNIST

We evaluated Professor Forcing on the task of sequentially generating the pixels in MNIST digits. We use the standard binarized MNIST dataset Murray and Salakhutdinov [2009]. We selected hyperparameters for our model on the validation set and elected to use 512 hidden states and a learning rate of 0.0001. For all experiments we used a 3-layer GRU as our generator. Unlike our other experiments, we used a convolutional network for the discriminator instead of a bi-directional RNN, as the pixels have a 2D spatial structure. We note that our model achieves the second best reported likelihood on this task, after the PixelRNN, which used a significantly more complicated architecture for its generator van den Oord et al. [2016]. Combining Professor Forcing with the PixelRNN would be an interesting area for future research. However, the PixelRNN parallelizes computation in the teacher forcing network in a way that doesn't work in the sampling network. Because Professor Forcing requires running the sampling network during training, naively combining Professor Forcing with the PixelRNN would be very slow.

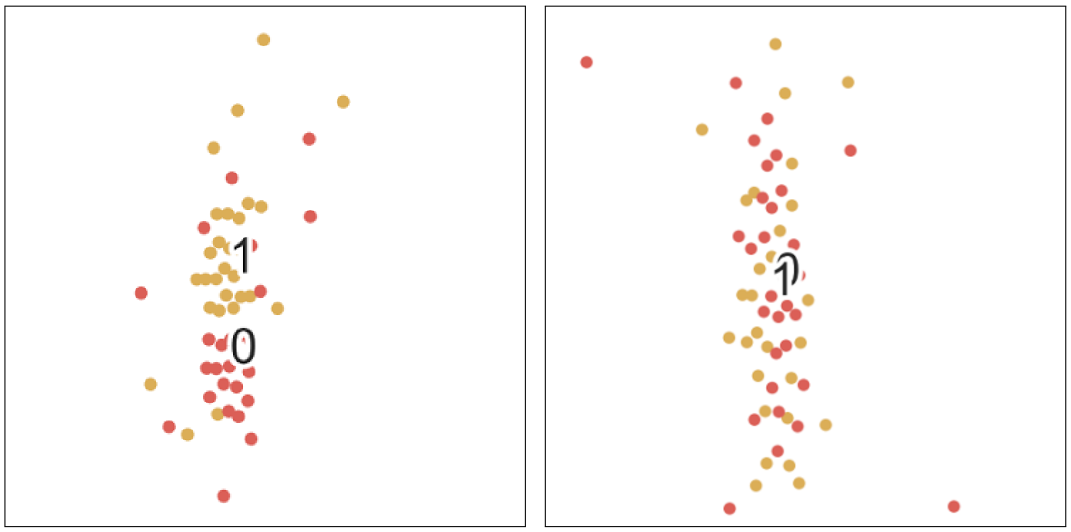

Figure 2.3 – Figure 3: T-SNE visualization of hidden states, left: with teacher forcing, right: with professor forcing. Red dots correspond to teacher forcing hidden states, while the gold dots correspond to free running mode. At t = 500, the closed-loop and open-loop hidden states clearly occupy distinct regions with teacher forcing, meaning that the network enters a hidden state region during sampling distinct from the region seen during teacher forcing training. With professor forcing, these regions now largely overlap. We computed 30 T-SNEs for Teacher Forcing and 30 T-SNEs for Professor Forcing and found that the mean centroid distance was reduced from 3000.0 to 1800.0, a 40% relative reduction. The mean distance from a hidden state in the training network to a hidden state in the sampling network was reduced from 22.8 with Teacher Forcing to 16.4 with Professor Forcing (vocal synthesis).

2.4.4 Handwriting Generation

With this task we wanted to investigate if Professor Forcing could be used to perform domain adaptation from a training set with short sequences to sampling much longer sequences. We train the Teacher Forcing model on only 50 steps of text-conditioned handwriting (corresponding to a few letters) and then sample for 1000 time steps . We let the model learn a sequence of (x, y) coordinates together with binary indicators of pen-up vs. pen-down, using the standard handwriting IAM-OnDB dataset, which consists of 13,040 handwritten lines written by 500 writers Liwicki and Bunke [2005]. For our teacher forcing model, we use the open source implementation Brebisson [2016] and use their hyperparameters which is based on the model in Graves [2013]. For the professor forcing model, we sample for 1000 time steps and run a separate discriminator on non-overlapping segments of length 50 (the number of steps used in the teacher forcing model). For both Teacher Forcing and Professor Forcing we sample with a bias of 0.5.

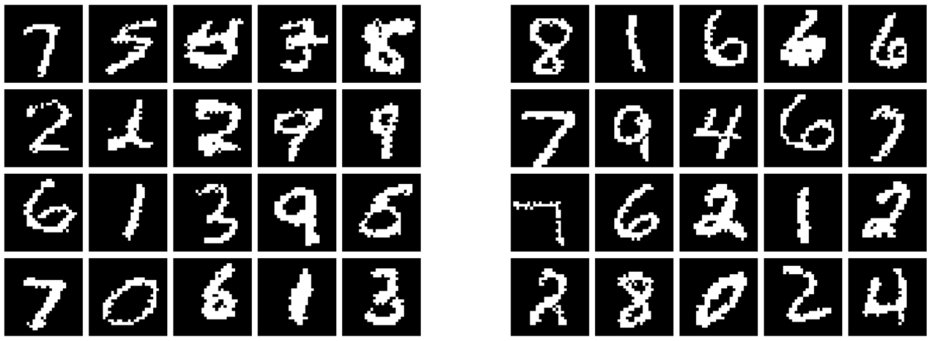

Figure 2.4 – Figure 4: Samples with Teacher Forcing (left) and Professor Forcing (right).

Method	MNIST NLL
DBN 2hl [Germain et al., 2015]	≈ 84.55
NADE [Larochelle and Murray, 2011]	88.33
EoNADE-5 2hl [Raiko et al., 2014]	84.68
DLGM 8 leapfrog steps [Salimans et al., 2014]	≈ 85.51
DARN 1hl [Gregor et al., 2015]	≈ 84.13
DRAW [Gregor et al., 2015]	≤ 80.97
Pixel RNN [van den Oord et al., 2016]	**79.2**
Professor Forcing (ours)	79.58

Table 2.1 – Test set negative log-likelihood evaluations

We performed a human evaluation study on handwriting samples. We gave 48 volunteers 16 randomly selected Prof. Forcing samples randomly paired with 16 Teacher Forcing samples and asked them to indicate which sample was higher quality and whether it was 'much better' or 'slightly better'. Both models had equal training time and samples were drawn using the same procedure. Volunteers were not aware of which samples came from which model.

.

2.4.5 Music Synthesis on Raw Waveforms

We considered the task of vocal synthesis on raw waveforms. For this task we used three hours of monk chanting audio scraped from YouTube (https://www.

Response	Percent	Count
Professor Forcing Much Better	19.7	151
Professor Forcing Slightly Better	57.2	439
Teacher Forcing Slightly Better	18.9	145
Teacher Forcing Much Better	4.3	33
Total	100.0	768

Table 2.2 – Human Evaluation Study Results for Handwriting

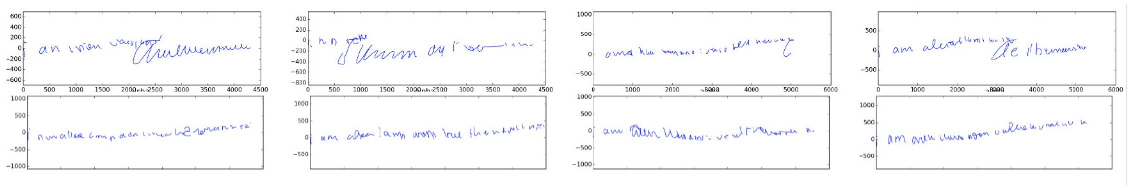

Figure 2.5 – Figure 5: Handwriting samples with teacher forcing (top) and professor forcing (bottom). Note that in both cases the model is only trained on observed sequences of length 50 steps (a few letters) but is used to do conditional generation for 1000 steps.

youtube.com/watch?v=9-pD28iSiTU). We sampled the audio at a rate of 1 kHz and took four seconds for each training and validation example. On each time step of the raw audio waveform we binned the signal's value into 8000 bins with boundaries drawn uniformly between the smallest and largest signal values in the dataset. We then model the raw audio waveform as a 4000-length sequence with 8000 potential values on each time step.

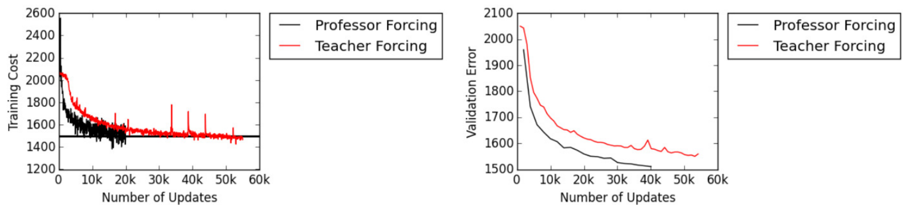

Figure 2.6 – Figure 6: Left: training likelihood curves. Right: validation likelihood curves.

We evaluated the quality of our vocal synthesis model using two criteria. First, we demonstrated a regularizing effect and improvement in negative log-likelihood. Second, we observed improvement in the quality of samples. We included a few randomly selected samples in the supplementary material and also performed human evaluation of the samples.

Visual inspection of samples is known to be a flawed method for evaluating generative models, because a generative model could simply memorize a small number of examples from the training set (or slightly modified examples from the training set) and achieve high sample quality. This issue was discussed in Theis et al. [2015]. However, this is unlikely to be an issue with our evaluation because our method also improved validation set likelihood, whereas a model that achieves quality samples by dropping coverage would have poorer validation set likelihood.

We performed human evaluation by asking 29 volunteers to listen to five randomly selected teacher forcing samples and five randomly selected professor forcing samples (included in supplementary materials and then rate each sample from 1-3 on the basis of quality. The annotators were given the samples in random order and were not told which samples came from which algorithm. The human annotators gave the Professor Forcing samples an average score of 2.20, whereas they gave the Teacher Forcing samples an average score of 1.30. .

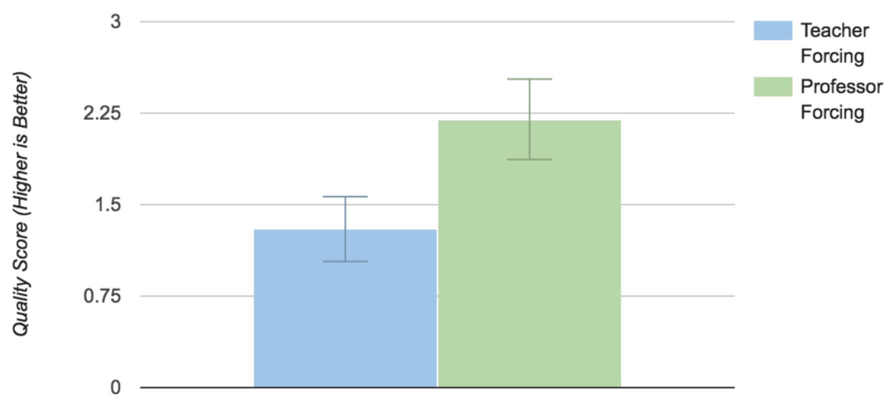

Figure 2.7 – Figure 7: Human evaluator ratings for vocal synthesis samples (higher is better). The height of the bar is the mean of the ratings and the error bar shows the spread of one standard deviation.

2.4.6 Negative Results on Shorter Sequences

On word level Penn Treebank we did not observe any difference between Teacher Forcing and Professor Forcing. One possible explanation for this difference is the increased importance of long-term dependencies in character-level language modeling. Also, for speech synthesis, we did not observe any difference between Teacher Forcing and Professor Forcing while training on sequences of length less than 100.

2.5 Conclusion

The idea of matching behavior of a model when it is running on its own, making predictions, generating samples, etc. vs when it is forced to be consistent with observed data is an old and powerful one. In this paper we introduce Professor Forcing, a novel instance of this idea when the model of interest is a recurrent generative one, and which relies on training an auxiliary model, the discriminator to spot the differences in behavior between these two modes of behavior. A major motivation for this approach is that the discriminator can look at the statistics of the behavior and not just at the single-step predictions, forcing the generator to behave the same when it is constrained by the data and when it is left generating outputs by itself for sequences that can be much longer than the training sequences. This naturally produces better generalization over sequences that are much longer than the training sequences, as we have found. We have also found that it helped to generalize better in terms of one-step prediction (log-likelihood), even though we are adding a possibly conflicting term to the log-likelihood training objective. This suggests that it acts like a regularizer but a very interesting one because it can also greatly speed up convergence in terms of number of training updates. We validated the advantage of Professor Forcing over traditional teacher forcing on a variety of sequential learning and generative tasks, with particularly impressive results in acoustic generation, where the training sequences are much shorter (because of memory constraints) than the length of the sequences we actually want to generate.

3 Variational Walkback: Learning a Transition Operator as a Stochastic Recurrent Net

Varitional Walkback: Learning a Transition Operator as a Stochastic Recurrent Net. Anirudh Goyal, Rosemary Nan Ke, Surya Ganguli, Yoshua Bengio

This chapter presents a joint work with Rosemary Nan Ke, Surya Ganguli and Yoshua Bengio. It was published at Advances in Neural Information Processing Systems 30 (NIPS 2017) (Conference Track).

Contribution: Prof. Yoshua Bengio got the original idea of Variational Walkback. I was responsible for writing the initial version of the code, and having the proof of concept that the idea works. Prof. Surya Ganguli connected it to non-equillibrium thermodynamics. Rosemary and I were responsible for showing that the proposed method works on toy tasks, which proved to be very insightful for the paper. I ran the code on all the real image datasets like MNIST, CIFAR, CelebA, SVHN. Initial version of the paper was writen by me and Rosemary, which was again rewritten by Prof. Yoshua, and which was again rewritten by Prof. Surya Ganguli and Prof. Yoshua Bengio. I wrote the appendix, as well as experimental section.

Affiliation

— Anirudh Goyal, MILA, University of Montreal

— Anirudh Goyal, MILA, University of Montreal

— Surya Ganguli, Stanford University

— Yoshua Bengio, MILA, University of Montreal

3.1 Variational Walkback - Connections to Directed and Undirected Graphical Models

Directed versions of probabilistic generative models can *synthesize* new sensory data through a sampling process which often converts a simple distribution over latent variables, modelling causes in the sensory data, into complex distributions over the data itself. They can also *analyze* sensory data by computing the posterior distribution over latent variables given sensory data. An early instantiation of this idea was the Helmholtz machine Dayan et al. [1995a], in which the analysis was performed by a *recognition* model and the synthesis was performed by a separate generative model, and the two were trained together to maximize the marginal probability of the data. More recent work on directed generative modelling using a small number of generative steps involves variational auto-encoders (VAEs) [Kingma and Welling, 2013, Rezende et al., 2014], while other approaches to training include generative adversarial networks (GANs) [Goodfellow et al., 2014]. In the probabilistic VAE approach however, it can be difficult to learn very deep models, with multiple layers of abstraction intervening between latent causes and sensory data. The *credit assignment problem* stands as a major impediment to learning such deep models as it can be difficult to optimize parameters controlling latent variables far removed from the data.

A parallel strand of generative modeling, through undirected probabilistic models, involves modelling the data as the *stationary* distribution of a stochastic process (e.g. various Boltzmann machines [Salakhutdinov and Hinton, 2009]). Sampling under this method corresponds to a potentially powerful iterative process of repeatedly applying a fixed stochastic operator that can gradually turn simple initial distributions over data into complex stationary distributions over data. However a key impediment to this approach is the *mixing time problem*: if the stationary distribution has multiple modes, the sampling process can take a long time to mix, or reach the stationary distribution, due to the excessive time sampling methods can take to jump between modes. Moreover, these models are also difficult to train, as solving the credit assignment problem also involves sampling from the model, in order to match the model's stationary distribution to the data distribution. Thus the mixing time problem contributes to the difficulty of the credit assignment difficult in undirected models.

In this thesis, we fuse the best ideas from directed (i.e. feedforward) generative models, and undirected (i.e. recurrent) generative models to provide novel methods to attack both the mixing time problem and credit assignment problems. A key insight is to note that in some sense, the analysis or recognition phase of directed generative models corresponds to a destructive process in which complex structure in the data distribution is replaced with a much simpler distribution over latent variables. Conversely, the synthesis or generative phase corresponds to a process of creation in which a simple distribution over latent variables is transformed into a complex distribution over visible data. The denoising autoencoder (DAE), as well as a more recent approach involving non-equilbirium thermodynamics (NET) [Sohl-Dickstein et al., 2015] both perform unsupervised learning based on training a creative process to reverse the effects of a process that destroys structure in data. In particular in NET, the destructive process corresponds to a fixed, finite time diffusion process in data space, which gradually turns a complex distribution over data into a simple distribution over the same space. The creative process corresponds to a feedforward neural network that is trained to reverse the flow of time in this diffusion process. The result is a very deep neural network that can turn simple distributions over data into the original complex data distribution. This method circumvents the mixing time problem by *not* modeling the data distribution as the stationary distribution of a stochastic process, but rather modeling it as the outcome of a non-equilibrium finite time process, and *demanding* during training that this process arrive at the data distribution in a limited time. Moreover, deep neural networks generically suffer from a credit assignment problem, partially solved via backpropagation. However, the NET approach circumvents the credit assignment problem because the past of the destructive process provides training targets for the future of the creation process ; thus each layer of the neural network reverses the flow of time by one-step and backpropagation over multiple layers is not required.

However, in NET, it is not clear that the fixed diffusion operation provides the best possible training targets for the generative neural network that underlies the creative process. In NET, since the diffusion process is fixed and therefore has no knowledge of the generative process, it is not at all clear that diffusion efficiently seeks out and thereby enables removal of the spurious modes of the reverse, generative process.

With this background in hand, we are now ready to describe the essential idea underlying our contribution in this paper. We learn *recurrent* dynamical systems by tying together the destruction and creation processes using the *same* stochastic dynamical system. In particular, the destructive process corresponds to a transient heating up of the dynamical system, which melts away structure in the model of the data distribution. This heating enables the destructive process to efficiently explore far away spurious modes of the generative process. The generative process in turn involves a reverse cooling process. The recurrent dynamical system is trained, via variational lower bound ideas that are employed in directed, or feed-forward generative models. The learning algorithm, derived in a principled manner via variational methods, trains the the cooling process to reverse the destruction of structure induced by the heating process. Intuitively, the learning algorithm punishes data trajectories which walk away from data during the heating process, and forces these trajectories to *WalkBack* to the data during the reverse cooling process. Hence we call our learning algorithm the Variational WalkBack Algorithm. This WalkBack intuition was already exploited in Bengio et al. [2013c] but without the firm mathematical grounding presented here.

Before presenting our framework in detail, we present one final, and important motivation for it, namely that it innovates in the rarely explored direction of directly parameterizing the generative model via a transition operator, rather than via an explicit probability function or energy function. This idea has already been discussed in the context Generative Stochastic Networks (GSNs) [Bengio et al., 2013b], a generalization of denoising auto-encoders (DAEs) [Vincent et al., 2008] which interprets the auto-encoder as estimating the gradient of an energy function [Alain and Bengio, 2014b] or as a transition operator [Bengio et al., 2013c]. From a machine learning perspective, as reviewed below, this removes the constraint of detailed balance, which is a constraint that is often invoked in stochastic processes which model data distributions as their stationary distributions. Removing this constraint yields access to a larger and potentially more powerful space of models. But more importantly, in neural network implementations of stochastic transition operators, the detailed balance constraint often translates into exact symmetry of the synaptic weight matrix [Bengio et al., 2015b]. However, such an exact constraint is biologically implausible. By removing the need for detailed balance, we thereby also remove one major impediment to biologically plausible generative modelling.

3.2 Introduction

A fundamental goal of unsupervised learning involves training generative models that can understand sensory data and employ this understanding to generate, or sample new data and make new inferences. In machine learning, the vast majority of probabilistic generative models that can learn complex probability distributions over data fall into one of two classes: (1) directed graphical models, corresponding to a finite time feedforward generative process (e.g. variants of the Helmholtz machine [Dayan et al., 1995b] like the Variational Auto-Encoder (VAE) [Kingma and Welling, 2013, Rezende et al., 2014]), or (2) energy function based undirected graphical models, corresponding to sampling from a stochastic process whose *equilibrium* stationary distribution obeys detailed balance with respect to the energy function (e.g. various Boltzmann machines [Salakhutdinov and Hinton, 2009]). This detailed balance condition is highly restrictive: for example, energy-based undirected models corresponding to neural networks require symmetric weight matrices and very specific computations which may not match well with what biological neurons or analog hardware could compute.

In contrast, biological neural circuits are capable of powerful generative dynamics enabling us to model the world and imagine new futures. Cortical computation is highly recurrent and therefore its generative dynamics cannot simply map to the purely feed-forward, finite time generative process of a directed model. Moreover, the recurrent connectivity of biological circuits is not symmetric, and so their generative dynamics cannot correspond to sampling from an energy-based undirected model.

Thus, the asymmetric biological neural circuits of our brain instantiate a type of stochastic dynamics arising from the repeated application of a transition operator[1] whose stationary distribution over neural activity patterns is a *non-equilibrium* distribution that does not obey detailed balance with respect to any energy function. Despite these fundamental properties of brain dynamics, machine learning approaches to training generative models currently lack effective methods to model complex data distributions through the repeated application a transition operator, that is not indirectly specified through an energy function, but rather is *directly*

1. A transition operator maps the previous-state distribution to a next-state distribution, and is implemented by a stochastic transformation which from the previous state of a Markov chain generates the next state

parameterized in ways that are inconsistent with the existence of *any* energy function. Indeed the lack of such methods constitutes a glaring gap in the pantheon of machine learning methods for training probabilistic generative models.

The fundamental goal of this paper is to provide a step to filling such a gap by proposing a novel method to learn such directly parameterized transition operators, thereby providing an empirical method to control the stationary distributions of non-equilibrium stochastic processes that do not obey detailed balance, and match these distributions to data. The basic idea underlying our training approach is to start from a training example, and iteratively apply the transition operator while gradually increasing the amount of noise being injected (i.e., temperature). This heating process yields a trajectory that starts from the data manifold and walks away from the data due to the heating and to the mismatch between the model and the data distribution. Similarly to the update of a denoising autoencoder, we then modify the parameters of the transition operator so as to make the *reverse* of this heated trajectory *more* likely under a reverse cooling schedule. This encourages the transition operator to generate stochastic trajectories that evolve towards the data distribution, by learning to walk back the heated trajectories starting at data points. This walkback idea had been introduced for generative stochastic networks (GSNs) and denoising autoencoders [Bengio et al., 2013d] as a heuristic, and without annealing. Here, we derive the specific objective function for learning the parameters through a principled variational lower bound, hence we call our training method variational walkback (VW). Despite the fact that the training procedure involves walking back a set of trajectories that last a finite, but variable number of time-steps, we find empirically that this yields a transition operator that continues to generate sensible samples for many more time-steps than are used to train, demonstrating that our finite time training procedure can sculpt the non-equilibrium stationary distribution of the transition operator to match the data distribution.

We show how VW emerges naturally from a variational derivation, with the need for annealing arising out of the objective of making the variational bound as tight as possible. We then describe experimental results illustrating the soundness of the proposed approach on the MNIST, CIFAR-10, SVHN and CelebA datasets. Intriguingly, we find that our finite time VW training process involves modifications of variational methods for training directed graphical models, while our potentially asymptotically infinite generative sampling process corresponds to non-equilibrium generalizations of energy based undirected models. Thus VW goes beyond the two disparate model classes of undirected and directed graphical models, while simultaneously incorporating good ideas from each.

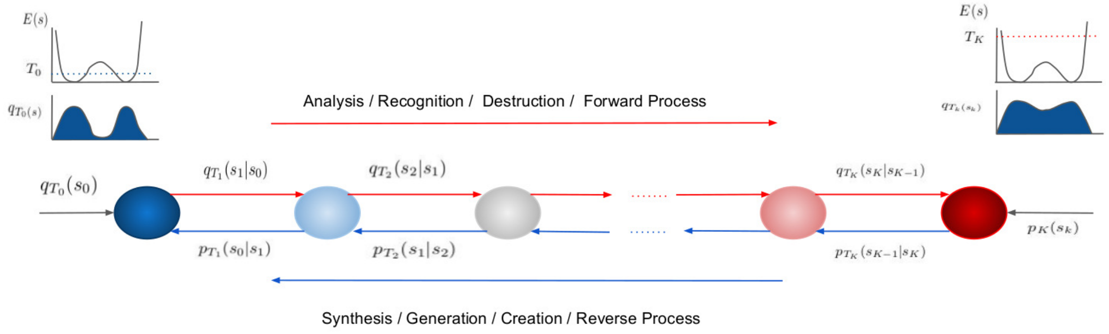

Figure 3.1 – Variational WalkBack framework. The generative process is represented in the blue arrows with the sequence of $p_{T_t}(\boldsymbol{s}_{t-1}|\boldsymbol{s}_t)$ transitions. The destructive forward process starts at a datapoint (from $q_{T_0}(\boldsymbol{s}_0)$) and gradually heats it through applications of $q_{T_t}(\boldsymbol{s}_t|\boldsymbol{s}_{t-1})$. Larger temperatures on the right correspond to a flatter distribution, so the whole destructive forward process maps the data distribution to a Gaussian and the creation process operates in reverse.

3.3 The Variational Walkback Training Process

Our goal is to learn a stochastic transition operator $p_T(\boldsymbol{s}'|\boldsymbol{s})$ such that its repeated application yields samples from the data manifold. Here T reflects an underlying temperature, which we will modify during the training process. The transition operator is further specified by other parameters which must be learned from data. When K steps are chosen to generate a sample, the generative process has joint probability $p(\boldsymbol{s}_0^K) = p(\boldsymbol{s}_K)\prod_{t=1}^{K} p_{T_t}(\boldsymbol{s}_{t-1}|\boldsymbol{s}_t)$, where T_t is the temperature at step t. We first give an intuitive description of our learning algorithm before deriving it via variational methods in the next section. The basic idea, as illustrated in Fig. 3.1 and Algorithm 1 is to follow a walkback strategy similar to that introduced in Alain and Bengio [2014a]. In particular, imagine a destructive process $q_{T_{t+1}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t)$ (red arrows in Fig. 3.1), which starts from a data point $\boldsymbol{s}_0 = \boldsymbol{x}$, and evolves it stochastically to obtain a trajectory $\boldsymbol{s}_0, \ldots, \boldsymbol{s}_K \equiv \boldsymbol{s}_0^K$, i.e., $q(\boldsymbol{s}_0^K) = q(s_0)\prod_{t=1}^{K} q_{T_t}(s_t|s_{t-1})$, where $q(s_0)$ is the data distribution. Note that the p and q chains will share the same parameters for the transition operator (one going backwards and one forward) but they start from different priors for their first step: $q(s_0)$ is the data distribution while $p(s_0)$ is a flat factorized prior (e.g. Gaussian). The training procedure trains the transition operator p_T to make reverse transitions of the destructive process more likely. For this reason we index time so the destructive process operates forward in time, while the reverse generative process operates backwards in time, with the data distribution occurring at $t = 0$. In particular, we need only train the transition operator to reverse time by 1-step at each step, making it unnecessary to solve a deep credit assignment problem by performing backpropagation through time across multiple walk-back steps. Overall, the destructive process generates trajectories that walk away from the data manifold, and the transition operator p_T learns to walkback these trajectories to sculpt the stationary distribution of p_T at $T = 1$ to match the data distribution.

Because we choose q_T to have the *same* parameters as p_T, they have the same transition operator but not the same joint over the whole sequence because of differing initial distributions for each trajectory. We also choose to increase temperature with time in the destructive process, following a temperature schedule $T_1 \leq \cdots \leq T_K$. Thus the forward destructive (reverse generative) process corresponds to a heating (cooling) protocol. This training procedure is similar in spirit to DAE's [Vincent et al., 2008] or NET [Sohl-Dickstein et al., 2015] but with one major difference: the destructive process in these works corresponds to the addition of random noise which knows nothing about the current generative process during training. To understand why tying together destruction and creation may be a good idea, consider the special case in which p_T corresponds to a stochastic process whose stationary distribution obeys detailed balance with respect to the energy function of an undirected graphical model. Learning any such model involves two fundamental goals: the model must place probability mass (i.e. lower the energy function) where the data is located, and remove probability mass (i.e. raise the energy function) elsewhere. Probability modes where there is no data are known as spurious modes, and a fundamental goal of learning is to hunt down these spurious modes and remove them. Making the destructive process *identical* to the transition operator to be learned is motivated by the notion that the destructive process should then efficiently explore the spurious modes of the current transition operator. The walkback training will then destroy these modes. In contrast, in DAE's and NET's, since the destructive process corresponds to the addition of unstructured noise that knows nothing about the generative process, it is not clear that such an agnostic destructive process will efficiently seek out the spurious modes of the reverse, generative process.

We chose the annealing schedule empirically to minimize training time. The generative process starts by sampling a state $\boldsymbol{s}_K$ from a broad Gaussian $p^*(\boldsymbol{s}_K)$, whose variance is initially equal to the total data variance $\sigma^2_{\max}$ (but can be later adapted to match the final samples from the inference trajectories). Then we sample from $p_{T_{\max}}(\boldsymbol{s}_{K-1}|\boldsymbol{s}_K)$, where $T_{\max}$ is a high enough temperature so that the resultant injected noise can move the state across the whole domain of the data. The injected noise to simulate temperature has variance linearly proportional to temperature. Thus if σ^2 is the equivalent noise injected by the transition operator p_T at $T = 1$, we choose $T_{\max} = \frac{\sigma^2_{\max}}{\sigma^2}$ to achieve the goal of the first sample s_{K-1} being able to move across the entire range of the data distribution. Then we successively cool the temperature as we sample "previous" states s_{t-1} according to $p_T(\boldsymbol{s}_{t-1}|\boldsymbol{s}_t)$, with T reduced by a factor of 2 at each step, followed by n steps at temperature 1. This cooling protocol requires the number of steps to be

$$K = \log_2 T_{\max} + n, \tag{3.1}$$

in order to go from $T = T_{\max}$ to $T = 1$ in K steps. We choose T from a random distribution. Thus the training procedure trains p_T to rapidly transition from a simple Gaussian distribution to the data distribution in a finite but variable number of steps. Ideally, this training procedure should then indirectly create a transition operator p_T at $T = 1$ whose repeated iteration samples the data distribution with a relatively rapid mixing time. Interestingly, this intuitive learning algorithm for a recurrent dynamical system, formalized in Algorithm 1, can be derived in a principled manner from variational methods that are usually applied to directed graphical models, as we see next.

Algorithm 1 VariationalWalkback($\boldsymbol{\theta}$)
Train a generative model associated with a transition operator $p_T(\boldsymbol{s}|\boldsymbol{s}')$ at temperature T (temperature 1 for sampling from the actual model), parameterized by $\boldsymbol{\theta}$. This transition operator injects noise of variance $T\sigma^2$ at each step, where σ^2 is the noise level at temperature 1.

```
Require: Transition operator $p_T(\boldsymbol{s}|\boldsymbol{s}')$ from which one can both sample and compute the gradient of $\log p_T(\boldsymbol{s}|\boldsymbol{s}')$ with respect to parameters $\theta$, given $\boldsymbol{s}$ and $\boldsymbol{s}'$.
Require: Precomputed $\sigma^2_{\max}$, initially data variance (or squared diameter).
Require: $N_1 > 1$ the number of initial temperature-1 steps of $q$ trajectory (or ending a $p$ trajectory).
  repeat
    Set $p^*$ to be a Gaussian with mean and variance of the data.
    $T_{\max} \leftarrow \frac{\sigma^2_{\max}}{\sigma^2}$
    Sample $n$ as a uniform integer between 0 and $N_1$
    $K \leftarrow \text{ceil}(\log_2 T_{\max}) + n$
    Sample $\boldsymbol{x} \sim$ data (or equivalently sample a minibatch to parallelize computation and process each element of the minibatch independently)
    Let $\boldsymbol{s}_0 = (\boldsymbol{x})$ and initial temperature $T = 1$, initialize $\mathcal{L} = 0$
    for $t = 1$ to $K$ do
      Sample $\boldsymbol{s}_t \sim p_T(\boldsymbol{s}|\boldsymbol{s}_{t-1})$
      Increment $\mathcal{L} \leftarrow \mathcal{L} + \log p_T(\boldsymbol{s}_{t-1}|\boldsymbol{s}_t)$
      Update parameters with log likelihood gradient $\frac{\partial \log p_T(\boldsymbol{s}_{t-1}|\boldsymbol{s}_t)}{\partial \theta}$
      If $t > n$, increase temperature with $T \leftarrow 2T$
    end for
    Increment $\mathcal{L} \leftarrow \mathcal{L} + \log p^*(\boldsymbol{s}_K)$
    Update mean and variance of $p^*$ to match the accumulated 1st and 2nd moment statistics of the samples of $\boldsymbol{s}_K$
  until convergence monitoring $\mathcal{L}$ on a validation set and doing early stopping
```

3.4 Variational Derivation of Walkback

The marginal probability of a data point $\boldsymbol{s}_0$ at the end of the K-step generative cooling process is

$$p(\boldsymbol{s}_0) = \sum_{\boldsymbol{s}_1^K} d\boldsymbol{s}_1^K p_{T_0}(\boldsymbol{s}_0|\boldsymbol{s}_1) \left(\prod_{t=2}^{K} p_{T_t}(\boldsymbol{s}_{t-1}|\boldsymbol{s}_t) \right) p^*(\boldsymbol{s}_K) \tag{3.2}$$

where $\boldsymbol{s}_1^K = (\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_K)$ and $\boldsymbol{v} = \boldsymbol{s}_0$ is a visible variable in our generative process, while the cooling trajectory that lead to it can be thought of as a latent, hidden variable $\boldsymbol{h} = \boldsymbol{s}_1^K$. Recall the decomposition of the marginal log-likelihood via a variational lower bound,

$$\ln p(v) \equiv \ln \sum_h p(v|h)p(h) = \underbrace{\sum_h q(h|v) \ln \frac{p(v,h)}{q(h|v)}}_{\mathcal{L}} + D_{KL}[q(h|v)||p(h|v)]. \tag{3.3}$$

Here $\mathcal{L}$ is the variational lower bound which motivates the proposed training procedure, and $q(h|v)$ is a variational approximation to $p(h|v)$. Applying this decomposition to $\boldsymbol{v} = \boldsymbol{s}_0$ and $\boldsymbol{h} = \boldsymbol{s}_1^K$, we find

$$\ln p(s_0) = \sum_{s_1^k} q(s_1^k|s_0) \ln \frac{p(s_0|s_1^k)p(s_1^k)}{q(s_1^k|s_0)} + D_{KL}[q(s_1^k|s_0) \,||\, p(s_1^k|s_0)] \tag{3.4}$$

Similarly to the EM algorithm, we aim to approximately maximize the log-likelihood with a 2-step procedure. Let θ_p be the parameters of the generative model p and θ_q be the parameters of the approximate inference procedure q. Before seeing the next example we have $\theta_q = \theta_p$. Then in the first step we update θ_p towards maximizing the variational bound $\mathcal{L}$, for example by a stochastic gradient descent step. In the second step, we update θ_q by setting $\theta_q \leftarrow \theta_p$, with the objective to reduce the KL term in the above decomposition. See Sec. 3.4.1 below regarding conditions for the tightness of the bound, which may not be perfect, yielding a possibly biased gradient when we force the constraint $\theta_p = \theta_q$. We continue iterating this procedure, with training examples s^0. We can obtain an unbiased Monte-Carlo estimator of $\mathcal{L}$ as follows from a single trajectory:

$$\mathcal{L}(s^0) \approx \sum_{t=1}^{K} \ln \frac{p_{T_t}(s_{t-1}|s_t)}{q_{T_t}(s_t|s_{t-1})} + \ln p^*(s_K) \tag{3.5}$$

with respect to p_θ, where s^0 is sampled from the data distribution $q_{T_0}(s^0)$, and the single sequence s_1^K is sampled from the heating process $q(s_1^K|s_0)$. We are making the reverse of heated trajectories more likely under the cooling process, leading to Algorithm 1. Such variational bounds have been used successfully in many learning algorithms in the past, such as the VAE [Kingma and Welling, 2013], except that they use an explicitly different set of parameters for p and q. Some VAE variants [Kingma et al., 2016] however mix the p-parameters implicitly in forming q, by using the likelihood gradient to iteratively form the approximate posterior.

3.4.1 Tightness of the variational lower bound

As seen in (3.4), the gap between $\mathcal{L}(s_0)$ and $\ln p(s_0)$ is controlled by $D_{KL}[q(s_1^k|s_0)||p(s_1^k|s_0)]$, and is therefore tight when the distribution of the heated trajectory, *starting* from a point s_0, matches the posterior distribution of the cooled trajectory *ending* at s_0. Explicitly, this KL divergence is given by

$$D_{KL} = \sum_{s_1^k} q(s_1^k|s_0) \ln \frac{p(s_0)}{p^*(s_K)} \prod_{t=1}^{K} \frac{q_{T_t}(s_t|s_{t-1})}{p_{T_t}(s_{t-1}|s_t)}. \tag{3.6}$$

As the heating process q unfolds forward in time, while the cooling process p unfolds backwards in time, we introduce the *time reversal* of the transition operator p_T, denoted by p_T^R, as follows. Under repeated application of the transition operator p_T, state s settles into a stationary distribution $\pi_T(s)$ at temperature T. The probability of observing a transition $s_t \to s_{t-1}$ under p_T in its stationary state is then $p_T(s_{t-1}|s_t)\pi_T(s_t)$. The time-reversal p_T^R is the transition operator that makes the reverse transition equally likely for all state pairs, and therefore obeys

$$P_T(s_{t-1}|s_t)\pi_T(s_t) = P_T^R(s_t|s_{t-1})\pi_T(s_{t-1}) \tag{3.7}$$

for all pairs of states s_{t-1} and s_t. It is well known that p_T^R is a valid stochastic transition operator and has the same stationary distribution $\pi_T(s)$ as p_T. Furthermore, the process p_T obeys detailed balance if and only if it is invariant under time-reversal, so that $p_T = p_T^R$.

To better understand the KL divergence in (3.6), at each temperature T_t, we use relation (3.7) to replace the cooling process P_{T_t} which occurs backwards in time with its time-reversal, unfolding forward in time, at the expense of introducing ratios of stationary probabilities. We also exploit the fact that q and p are the same transition operator. With these substitutions in (3.6), we find

$$D_{KL} = \sum_{s_1^k} q(s_1^k|s_0) \ln \prod_{t=1}^{K} \frac{p_{T_t}(s_t|s_{t-1})}{p_{T_t}^R(s_t|s_{t-1})} + \sum_{s_1^k} q(s_1^k|s_0) \ln \frac{p(s_0)}{p^*(s_K)} \prod_{t=1}^{K} \frac{\pi_{T_t}(s_t)}{\pi_{T_t}(s_{t-1})}. \tag{3.8}$$

The first term in (3.8) is simply the KL divergence between the distribution over heated trajectories, and the time reversal of the cooled trajectories. Since the heating (q) and cooling (p) processes are tied, this KL divergence is 0 if and only if $p_{T_t} = p_{T_t}^R$ for all t. This is equivalent to the transition operator p_T obeying detailed balance at all temperatures.

Now intuitively, the second term can be made small in the limit where K is large and the temperature sequence is annealed slowly. To see why, note we can write the ratio of probabilities in this term as,

$$\frac{p(s_0)}{\pi_{T_1}(s_0)}\frac{\pi_{T_1}(s_1)}{\pi_{T_2}(s_1)}\cdots\frac{\pi_{T_{K-1}}(s_{K-1})}{\pi_{T_{K-1}}(s_K)}\frac{\pi_{T_K}(s_K)}{p^*(s_K)}. \tag{3.9}$$

which is similar in shape (but arising in a different context) to the product of probability ratios computed for annealed importance sampling [Neal, 2001] and reverse annealed importance sampling [Burda et al., 2014]. Here it is manifest that we are comparing probabilities of states under slightly different distributions, so all ratios ≈ 1. For example, under many steps, with slow annealing, the generative process approximately reaches its stationary distribution, $p(s_0) \approx \pi_{T_1}(s_0)$.

This slow annealing to go from $p^*(s_K)$ to $p(s_0)$ corresponds to the quasistatic limit in statistical physics, where the work to perform is equal to the free energy difference between states. To go faster, one must perform excess work, above and beyond the free energy difference, dissipated as heat into the surrounding environment. By writing distributions in terms of energies and free energies: $\pi_{T_t}(s_t) \propto e^{-E(s_t)/T_t}$, $p^*(s_K) = e^{-[E_K(s_K)-F_K]}$, and $p(s_0) = e^{-[E_0(s_0)-F_0]}$, one can see that the second term in the KL divergence is closely related to average heat dissipation in a finite time heating process (see e.g. [Crooks, 2000]). This intriguing connection between the size of the gap in a variational lower bound, and the excess heat dissipation in a finite time heating process opens the door to exploiting a wealth of work in statistical physics for finding optimal thermodynamic paths that minimize heat dissipation [Schmiedl and Seifert, 2007, Sivak and Crooks, 2012, Gingrich et al., 2016], which may provide new ideas to improve variational inference. In summary, tightness of the variational bound can be achieved if: (1) The transition operator of p approximately obeys detailed balance, and (2) the temperature annealing is done slowly over many steps. And intriguingly, the magnitude of the looseness of the bound is related to two physical quantities: (1) the degree of irreversiblity of the transition operator p, as measured by the KL divergence between p and its *time reversal* p^R, and (2) is the physical work required to perform the heating trajectory.

To check, potential looseness of the variational lower bound, we can measure the degree of irreversibility of p_T by estimating the KL divergence

$D_{KL}(p_T(s'|s)\pi_T(s) \,||\, p_T(s|s')\pi_T(s'))$, which is 0 if and only if p_T obeys detailed balance and is therefore time-reversal invariant. This quantity can be estimated by $\frac{1}{K}\sum_{t=1}^{K} \ln \frac{p_T(s_{t+1}|s_t)}{p_T(s_t|s_{t+1})}$, where s_1^K is a long sequence sampled by repeatedly applying transition operator p_T from a draw $s_1 \sim \pi_T$. If this quantity is strongly positive (negative) then forward transitions are more (less) likely than reverse transitions, and the process p_T is not time-reversal invariant. This estimated KL divergence can be normalized by the corresponding entropy to get a relative value (with 3.6% measured on a trained model, as detailed in Appendix).

3.4.2 Estimating log likelihood via importance sampling

We can derive an importance sampling estimate of the negative log-likelihood by the following procedure. For each training example x, we sample a large number of destructive paths (as in Algorithm 1). We then use the following formulation to estimate the log-likelihood $\log p(\boldsymbol{x})$ via

$$\log \mathbb{E}_{\boldsymbol{x}\sim p_{\mathcal{D}}, q_{T_0}(\boldsymbol{x}) q_{T_1}(\boldsymbol{s}_1|\boldsymbol{s}_0(\boldsymbol{x},)) \left(\prod_{t=2}^{K} q_{T_t}(\boldsymbol{s}_t|\boldsymbol{s}_{t-1})\right)} \left[\frac{p_{T_0}(\boldsymbol{s}_0 = \boldsymbol{x}|\boldsymbol{s}_1) \left(\prod_{t=2}^{K} p_{T_t}(\boldsymbol{s}_{t-1}|\boldsymbol{s}_t)\right) p^*(\boldsymbol{s}_K)}{q_{T_0}(\boldsymbol{x}) q_{T_1}(\boldsymbol{s}_1|\boldsymbol{s}_0 = \boldsymbol{x}) \left(\prod_{t=2}^{K} q_{T_t}(\boldsymbol{s}_t|\boldsymbol{s}_{t-1})\right)} \right] \tag{3.10}$$

3.4.3 VW transition operators and their convergence

The VW approach allows considerable freedom in choosing transition operators, obviating the need for specifying them indirectly through an energy function. Here we consider Bernoulli and isotropic Gaussian transition operators for binary and real-valued data respectively. The form of the stochastic state update imitates a discretized version of the Langevin differential equation. The Bernoulli transition operator computes the element-wise probability as $\rho = \text{sigmoid}(\frac{(1-\alpha)*\boldsymbol{s}_{t-1}+\alpha*F_\rho(\boldsymbol{s}_{t-1})}{T_t})$. The Gaussian operator computes a conditional mean and standard deviation via $\mu = (1-\alpha)*\boldsymbol{s}_{t-1} + \alpha * F_\mu(\boldsymbol{s}_{t-1})$ and $\sigma = T_t \log(1 + e^{F_\sigma(\boldsymbol{s}_{t-1})})$. Here the F functions can be arbitrary parametrized functions, such as a neural net and T_t is the temperature at time step t.

A natural question is when will the finite time VW training process learn a transition operator whose stationary distribution matches the data distribution, so that repeated sampling far beyond the training time continues to yield data samples. To partially address this, we prove the following theorem: If p has enough capacity, training data and training time, with slow enough annealing and a small departure from reversibility so p can match q, then at convergence of VW training, the transition operator p_T at $T = 1$ has the data generating distribution as its stationary distribution. A proof can be found in the Appendix, but the essential intuition is that if the finite time generative process converges to the data distribution at multiple different VW walkback time-steps, then it remains on the data distribution for all future time at $T = 1$. We cannot always guarantee the preconditions of this theorem but we find experimentally that its essential outcome holds in practice.

3.5 Related Work

A variety of learning algorithms can be cast in the framework of Fig. 3.1. For ex. for directed graphical models like VAEs [Kingma and Welling, 2013, Rezende et al., 2014], DBNs [Hinton et al., 2006], and Helmholtz machines in general, q corresponds to a recognition model, transforming data to a latent space, while p corresponds to a generative model that goes from latent to visible data in a finite number of steps. None of these directed models are designed to learn transition operators that can be iterated *ad infinitum*, as we do. Moreover, learning such models involves a complex, deep credit assignment problem, limiting the number unobserved latent layers that can be used to generate data. Similar issues of limited trainable depth in a finite time feedforward generative process apply to Generative Adversarial Networks (GANs) [Goodfellow et al., 2014], which also further eschew the goal of specifically assigning probabilities to data points. Our method circumvents this deep credit assignment problem by providing training targets at each time-step ; in essence each past time-step of the heated trajectory constitutes a training target for the future output of the generative operator p_T, thereby obviating the need for backpropagation across multiple steps. Similarly, unlike VW, Generative Stochastic Networks (GSN) [Bengio et al., 2014] and the DRAW [Gregor et al., 2015] also require training iterative operators by backpropagating across multiple computational steps.

VW is similar in spirit to DAE [Bengio et al., 2013d], and NET approaches [Sohl-Dickstein et al., 2015] but it retains two crucial differences. First, in each of these frameworks, q corresponds to a very simple destruction process in which Gaussian noise is injected into the data. This agnostic destruction process has no knowledge of underlying generative process p that is to be learned, and therefore cannot be expected to efficiently explore spurious modes, or regions of space, unoccupied by data, to which p assigns high probability. VW has the advantage of using a high-temperature version of the model p itself as part of the destructive process, and so should be better than random noise injection at finding these spurious modes. A second crucial difference is that VW ties weights of the transition operator across time-steps, thereby enabling us to learn a *bona fide* transition operator than can be iterated well beyond the training time, unlike DAEs and NET. There's also another related recent approach to learning a transition operator with a denoising cost, developped in parallel, called Infusion training [Bordes et al., 2017], which tries to reconstruct the target data in the chain, instead of the previous step in the destructive chain.

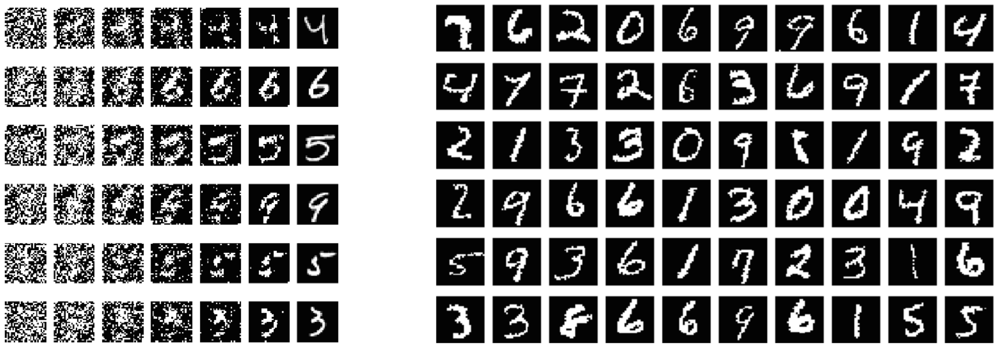

Figure 3.2 – Generating MNIST-like samples using a Bernoulli likelihood in the transition operator. Left: VW iteratively generates images starting from a noise prior. For intermediate steps we display samples and for the final step (right hand side image) we display the transition operator's mean.

3.6 Experiments

VW is evaluated on four datasets: MNIST, CIFAR10 [Krizhevsky and Hinton, 2009], SVHN [Netzer et al., 2011] and CelebA [Liu et al., 2015]. The MNIST, SVHN and CIFAR10 datasets were used as is except for uniform noise added to MNIST and CIFAR10, as per Theis et al. [2016], and the aligned and cropped version of CelebA was scaled from 218 x 178 pixels to 78 x 64 pixels and center-cropped at 64 x 64 pixels [Liu et al., 2015]. We used the Adam optimizer [Kingma and Ba, 2014] and the Theano framework [Al-Rfou et al., 2016]. More details are in Appendix and code for training and generation is at https://github.com/anirudh9119/walkback_nips17. Table 3.1a shows the effect of longer annealing while Table 3.1b compares the importance sampling log-likelihood estimates for VW with published AIS values for existing models, and Table 3.2 compares with published NET results on CIFAR.

Model	Train LL	Test LL
VW (5 steps, LB)	702	510
VW (10 steps, LB)	890	801
VW (15 step, LB)	1148	1018
VW (IS estimate)	1311	1243

(a) MNIST log-likehoods: VW Model using a Gaussian noise with diagonal covariance, and we applied lkboth the lower bound and importance sampling (15 steps) estimates.

Model	Train LL	Test LL
VAE-50 (AIS)	1272 ± 6.7	991 ± 6.5
GAN-50 (AIS)	620 ± 31	627 ± 8.8
GMMN-50 (AIS)	571 ± 31	593 ± 8.6
VAE-10 (AIS)	780 ± 19	705 ± 7.4
GAN-10 (AIS)	318 ± 22	328 ± 5.5
GMMN-10 (AIS)	345 ± 20	346 ± 5.9
VW (IS estimate)	793 ± 2.1	712 ± 3.4

(b) MNIST log-likehoods. VW with isotropic Gaussian output, importance sampling estimates. Log-likelihoods estimated by AIS reported in [Wu et al., 2016].

Table 3.1 – Comparative log-likelihoods estimated by importance sampling.

Model	bits/dim $\leq$
NET [Sohl-Dickstein et al., 2015]	5.40
VW	4.40
Deep VAE	< 4.54
DRAW [Gregor et al., 2015]	< 4.13
ResNet VAE with IAF [Kingma et al., 2016]	3.11

Table 3.2 – Comparisons on CIFAR10, test set average number of bits/data dimension(lower is better)

Image Generation. Figure 3, 5, 6, 7, 8 (see supplementary section) show VW samples on each of the datasets. For MNIST, real-valued views of the data are modeled. **Image Inpainting.** We clamped the bottom part of CelebA test images (for each step during sampling), and ran it through the model. Figure 1 (see Supplementary section) shows the generated conditional samples.

3.7 Discussion and Future Work

We introduced a new approach to learning non-energy-based transition operators which inherits advantages from several previous generative models, including a training objective that requires rapidly generating the data in a finite number of steps (as in directed models), re-using the same parameters for each step (as in undirected models), directly parametrizing the generator (as in GANs and DAEs), and using the model itself to quickly find its own spurious modes (the walk-back idea). We also anchor the algorithm in a variational bound and show how its analysis suggests to use the same process for the destruction, or inference, and creation, or generation, and to use a cooling (heating) schedule during generation (inference). Moreover, we connected the variational gap to physical notions like reversibility and heat dissipation. We verified empirically that the model converges towards an *approximately* reversible chain (see Appendix) making the variational bound tighter. We also found that samples are of good quality, and better than previous approaches at learning directly a transition operator (VAE, GSN and NET). Note that when the data is discrete and we consider the q inference chain and p generative sequence as two long sequences of latent states, VW avoids the need to backpropagate (which is not possible for discrete states) or use REINFORCE to obtain an update for the q parameters, thanks to sharing parameters between p and q, but at the price of a biased estimator of the lower bound gradient.

A fundamental aspect of our approach is that we can train stochastic processes that need not exactly obey detailed balance, yielding access to a larger and potentially more powerful space of models. In particular, this enables us to relax the weight symmetry constraint of undirected graphical models corresponding to neural networks, yielding a more brain like iterative computation characteristic of asymmetric biological neural circuits.

Many questions remain open to analyze and extend VW, in particular to incorporate latent layers. The state at each step would now include both visible $\boldsymbol{x}$ and latent $\boldsymbol{h}$ components. Essentially the same procedure can be run, except for the chain initialization, with $\boldsymbol{s}_0 = (\boldsymbol{x}, \boldsymbol{h}_0)$ where $\boldsymbol{h}_0$ a sample from the posterior distribution of $\boldsymbol{h}$ given $\boldsymbol{x}$. Overall, our work takes a step to filling a relatively open niche in the machine learning literature on *directly* training non-energy-based iterative stochastic operators, and we hope that the many possible extensions of this approach could lead to a rich new class of more powerful brain-like machine learning models.

3.8 VW transition operators and their convergence

If p has enough capacity, training data and training time, with slow enough annealing and a small departure from reversibility so p can match q, then at convergence of VW training, the transition operator p_T at $T = 1$ has the data generating distribution as its stationary distribution.

Démonstration. With these conditions $p(s_0^{K+n})$ match $q(s_0^{K+n})$, where $q(s_0)$ is the data distribution. It means that $p(s_0)$ (the marginal at the last step of sampling) is the data distribution when running the annealed (cooling) trajectory for $K + n$ steps, for n any integer between 0 and N_1, where the last $n + 1$ steps are at temperature 1. Since the last n steps are at temperature 1, they apply the same transition operator. Consider any 2 consecutive sampling steps among these last n steps. Both of these samples are coming from the same distribution (the data distribution). It means that the temperature 1 transition operator leaves the data distribution unchanged. This implies that the data distribution is an eigenvector of the linear operator associated with the temperature 1 transition operator, or that the data generating distribution is a stationary distribution of the temperature 1 transition operator. □

3.9 Additional Results

Image inpainting samples from CelebA dataset are shown in Fig 3.3, where each top sub-figure shows the masked image of a face (starting point of the chain), and the bottom sub-figure shows the inpainted image. The images are drawn from the test set.

The VW samples for CelebA, CIFAR10 and SVHN are shown in Fig 3.5, 3.6, 3.7.

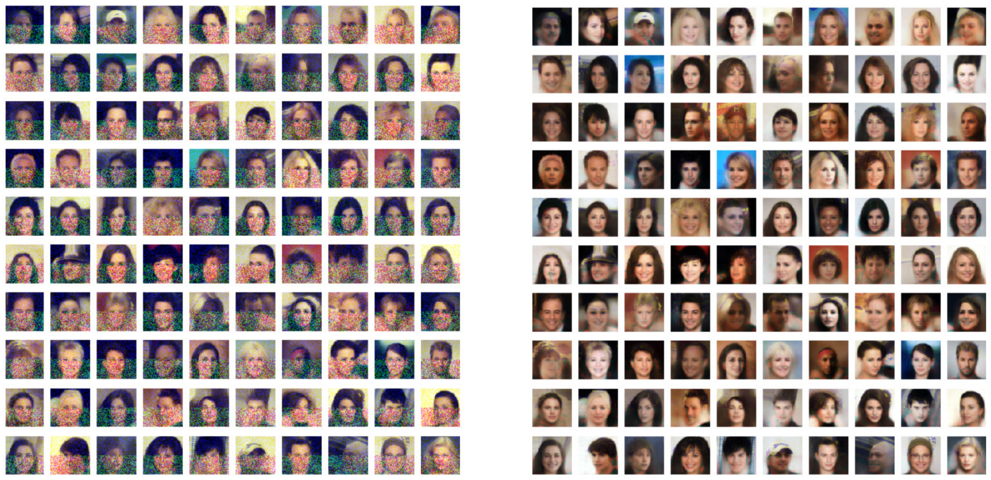
Figure 3.3 – VW inpainting in CelebA images. Images on the left are the ground truth images corrupted for their bottom half (which is the starting point of the chain). The goal is to fill in the bottom half of each face image given an observed top half of an image (drawn from test set). Images on the right show the inpainted lower halves for all these images.

3.10 VW on Toy Datasets

Fig. 3.8 and 3.9 shows the application of a transition operator applied on 2D datasets.

3.11 VW chains

Fig. 3.10, 3.11, 3.12, 3.13, 3.14, 3.15, 3.16 shows the model chains on repeated application of transition operator at temperature = 1. This is to empirically prove the conjecture mentioned in the paper (Preposition 1) that is, if the finite time generative process converges to the data distribution at multiple different VW

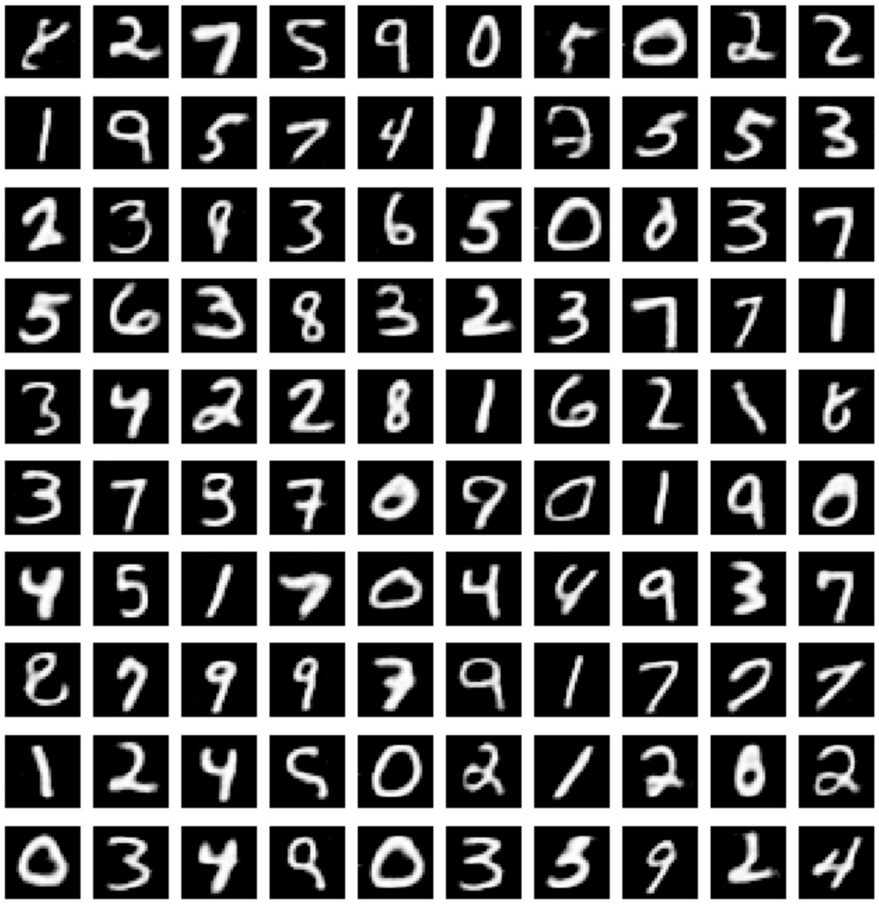

Figure 3.4 – VW samples on MNIST using Gaussian noise in the transition operator. The model is trained with 30 steps of walking away, and samples are generated using 30 annealing steps.

walkback time-steps, then it remains on the data distribution for all future time at T= 1

3.12 Walkback Procedure Details

The variational walkback algorithm has three unique hyperparameters. We specify the number of Walkback steps used during training, the number of extra Walk-

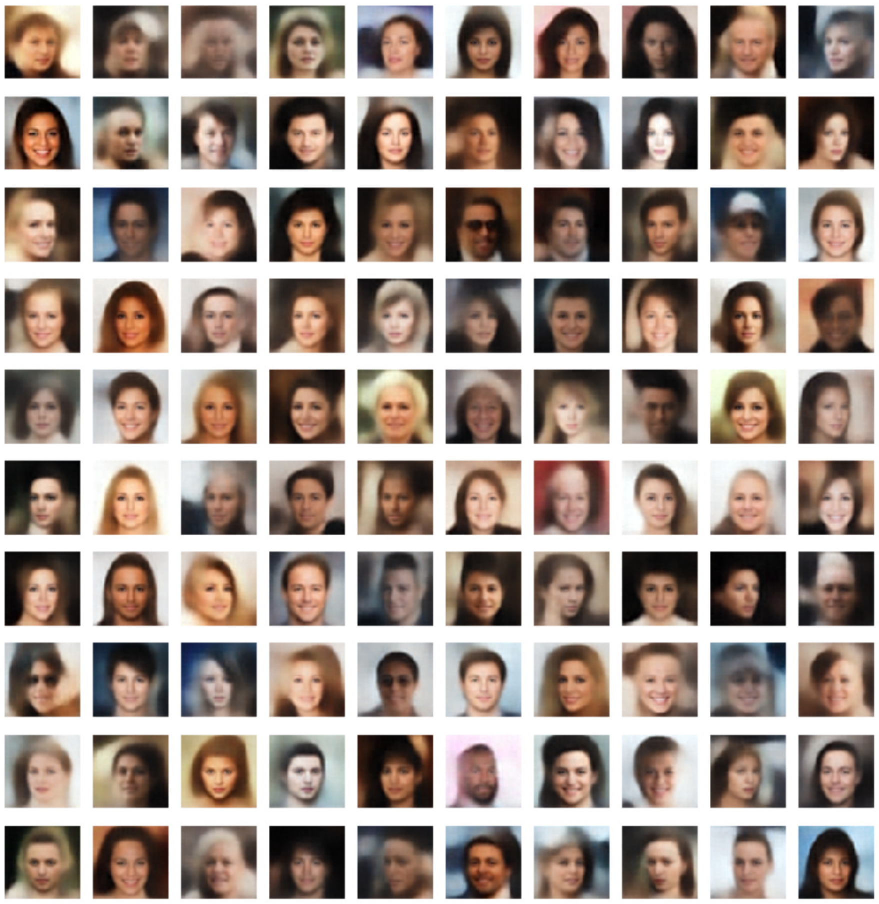

Figure 3.5 – VW samples on CelebA dataset using Gaussian noise in the transition operator. Model is trained using 30 steps to walk away and samples are generated using 30 annealing steps.

back steps used during sampling and also the temperature increase per step.

The most conservative setting would be to allow the model to slowly increase the temperature during training. However, this would require a large number of steps for the model to walk to the noise, and this would not only significantly slow down the training process, but this also means that we would require a large number of steps used for sampling.

There may exist a dynamic approach for setting the number of Walkback steps

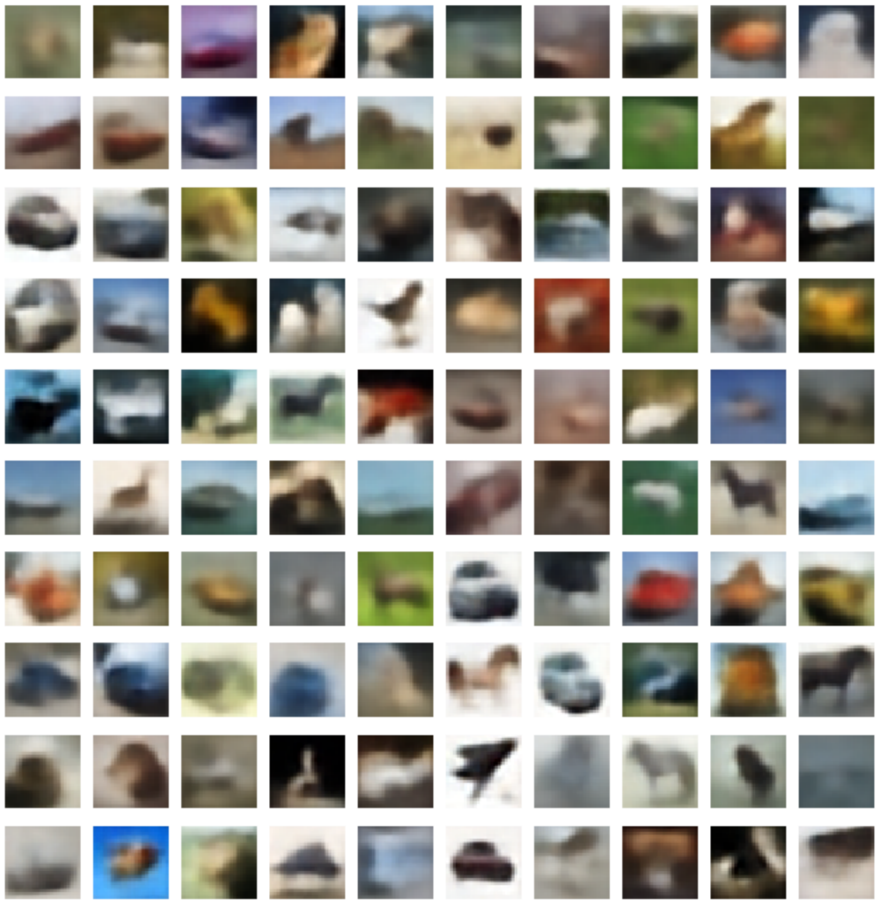

Figure 3.6 – VW samples on Cifar10 using Gaussian noise in the transition operator. Model is trained using 30 steps to walk away and samples are generated using 30 annealing steps.

and the temperature schedule. In our work, we set this hyperparameters heuristically. We found that a heating temperature schedule of $T_t = T_0\sqrt{2^t}$ at step t produced good results, where $T_0 = 1.0$ is the initial temperature. During sampling, we found good results using the exactly reversed schedule: $T_t = \frac{\sqrt{2^N}}{\sqrt{2^t}}$, where t is the step index and N is the total number of cooling steps.

For MNIST, CIFAR, SVHN and CelelbA, we use $K = 30$ training steps and $N = 30$ sampling steps. We also found that we could achieve better quality results if allow the model to run for a few extra steps with a temperature of 1 during

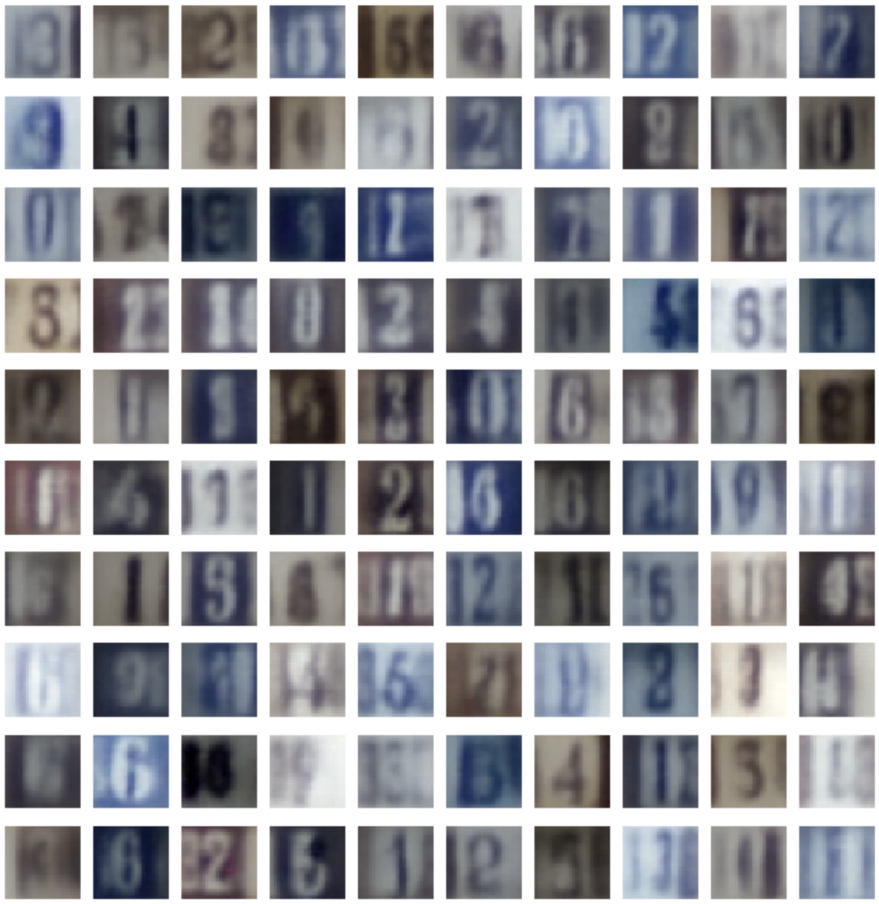

Figure 3.7 – VW samples on SVHN dataset using Gaussian noise in the transition operator. Model is trained using 30 steps to walk away and samples are generated using 30 annealing steps.

sampling. Finally, our model is able to achieve similar results compared to the NET model[Sohl-Dickstein et al., 2015]. Considering our model uses only 30 steps for MNIST and NET [Sohl-Dickstein et al., 2015] uses 1000 steps for MNIST.

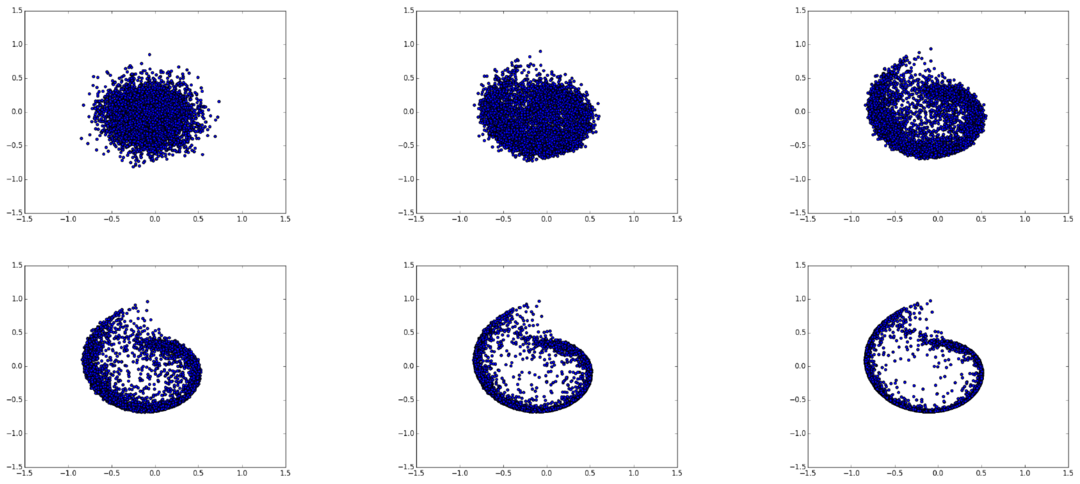

Figure 3.8 – The proposed modeling framework trained on 2-d swiss roll data. This algorithm was trained on 2D swiss roll for 30 annealing steps using annealing schedule increasing temperator by 1.1 each time. We have shown every 5th sample (ordering is row wise, and within each row it is column-wise.

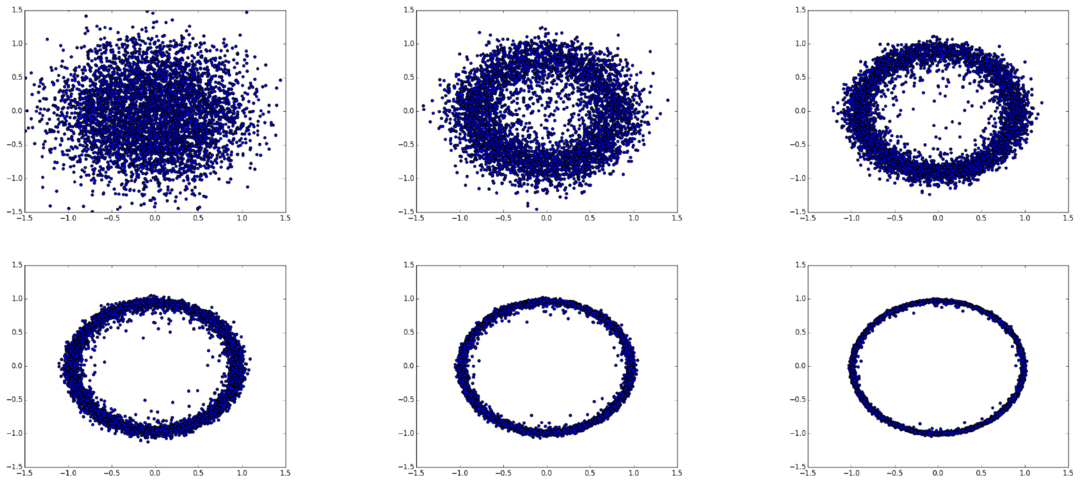

Figure 3.9 – The proposed modeling framework trained on circle data. This algorithm was trained on circle for 30 annealing time steps using annealing schedule increasing temperature by factor 1.1 each time. We have shown every 5th sample (ordering is row wise, and within each row it is column-wise.

3.13 Higher Lower Bound: not always better samples

We have observed empirically that the variational lower bound does not necessarily correspond to sample quality. Among trained models, higher value of the lower bound is not a clear indication of visually better looking samples. Our MNIST samples shown in Fig 3.17 is an example of this phenomenon. A model with better lower bound could give better reconstructions while not producing better generated samples. This resonates with the finding of [Theis et al., 2016]

3.14 Reversibility of transition operator

We measured the degree of reversibility of p_T by estimating the KL divergence $D_{KL}(p_T(s'|s)\pi_T(s)\,||\,p_T(s|s')\pi_T(s'))$, which is 0 if and only if p_T obeys detailed balance and is therefore time-reversal invariant by computing the Monte-Carlo estimator $\frac{1}{K}\sum_{t=1}^{K}\ln\frac{p_T(s_{t+1}|s_t)}{p_T(s_t|s_{t+1})}$, where s_1^K is a long sequence sampled by repeatedly applying transition operator p_T from a draw $s_1 \sim \pi_T$, i.e., taking samples after a burn-in period (50 samples).

To get a sense of the magnitude of this reversibility measure, and because it corresponds to an estimated KL divergence, we estimate the corresponding entropy (of the forward trajectory) and use it as a normalizing denominator telling us how much we depart from reversibility in nats relative to the number of nats of entropy. To justify this, consider that the minimal code length required to code samples from a distribution p is the entropy $H(p)$. But suppose we evaluate those samples from p using q instead to code them. Then the code length is $H(p) + D(p||q)$. So the fractional increase in code length due to having the wrong distribution is $D(p||q)/H(p)$, which is what we report here, with p being the forward transition probability and q the backward transition probability.

To compute this quantity, we took our best model (in terms of best lower bound) on MNIST, and ran it for 1000 time steps i.e ($T = 1000$), at a constant temperature.

We run the learned generative chain p for T time steps getting $s_0 \to s_1 \to s_2 \to \cdots s_T$ and computing $\log p(s_0 \to s_1 \to s_2 \to \cdots s_T)/p(s_T \to \cdots \to s2 \to s1)$

both under the same generative chain, divided by T to get the per-step average.

On the same set of runs, we compute $1/T * \log p(s_0 \to s_1 \to s_2 \to \cdots s_T)$ under the same generative chain. This is an estimate of the entropy per unit time of the chain. This is repeated multiple times to average over many runs and reduce the variance of the estimator.

The obtained ratio (nats/nats) is 3.6%, which seems fairly low but also suggests that the trained model is not perfectly reversible.

3.15 Some Minor Points

— In all the image experiments, we observed that by having different batchnorm papemeters for different steps, actually improves the result considerably. Having different batchnorm parameters was also necessery for making it work on mixture on gaussian. The authors were not able to make it work on MoG without different parameters. One possible way, could be to let optimizer know that we are on different step by giving the temperature information to the optimizer too.

— We observed better results while updating the parameters in online-mode, as compared to batch mode. (i.e instead of accumulating gradients across different steps, we update the parameters in an online fashion)

3.16 Inception Scores on CIFAR

We computed the inception scores using 50,000 samples generated by our model. We compared the inception scores with [Salimans et al., 2016] as the baseline model.

Model	Inception Score
Real Data	11.24
Salimans (semi-supervised)	8.09
Salimans (unsupervised)	4.36
Salimans (supervised training without minibatch features)	3.87
VW(20 steps)	3.72
VW(30 steps)	4.39 ±0.2

Table 3.3 – Inception scores on CIFAR

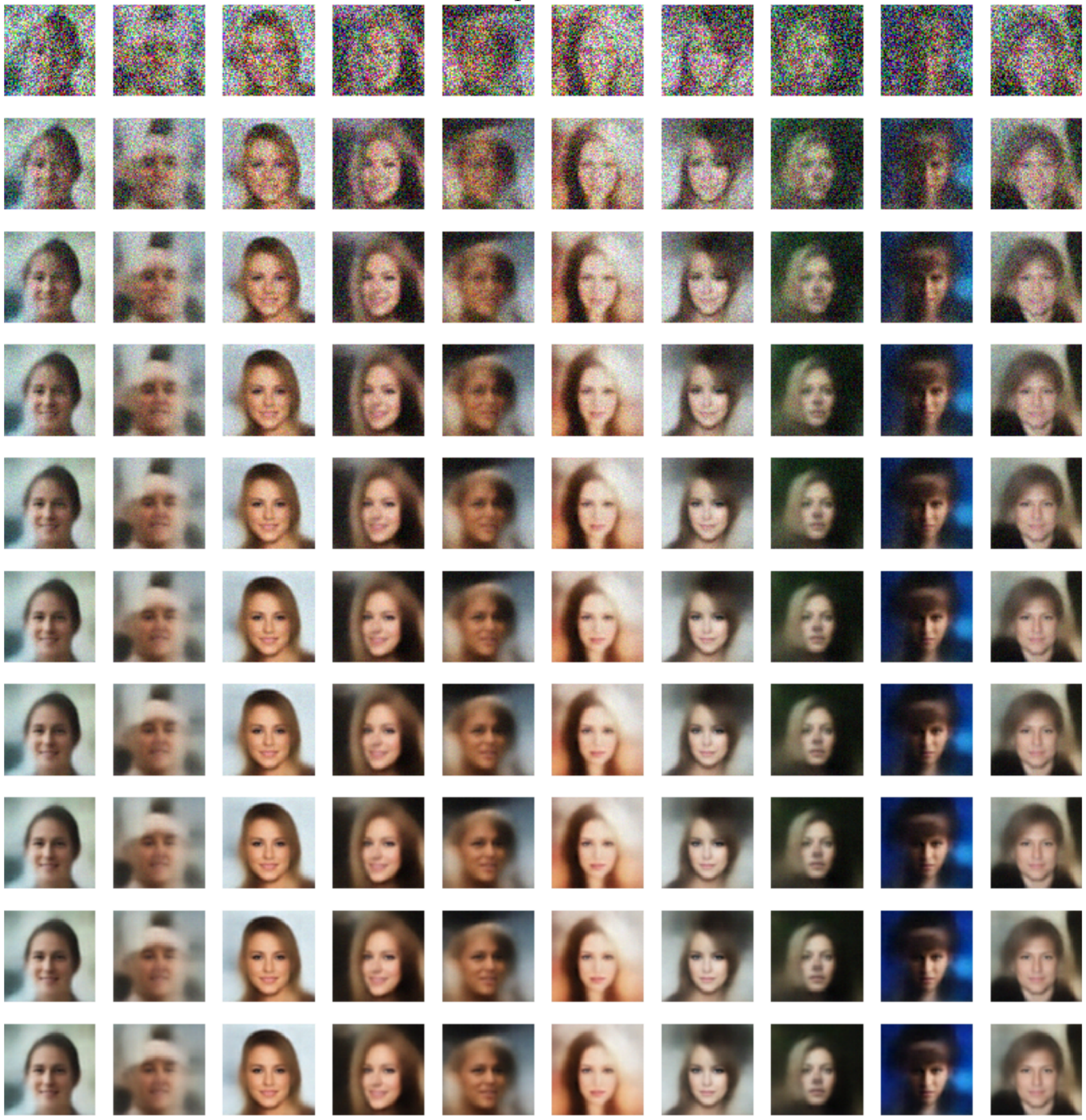

Figure 3.10 – VW sample chain (vertically, going down) starting from pure noise. Model trained using $K = 30$ steps to walk away and samples are generated using 30 steps of annealing. The figure shows every 3rd sample of the chain in each column.

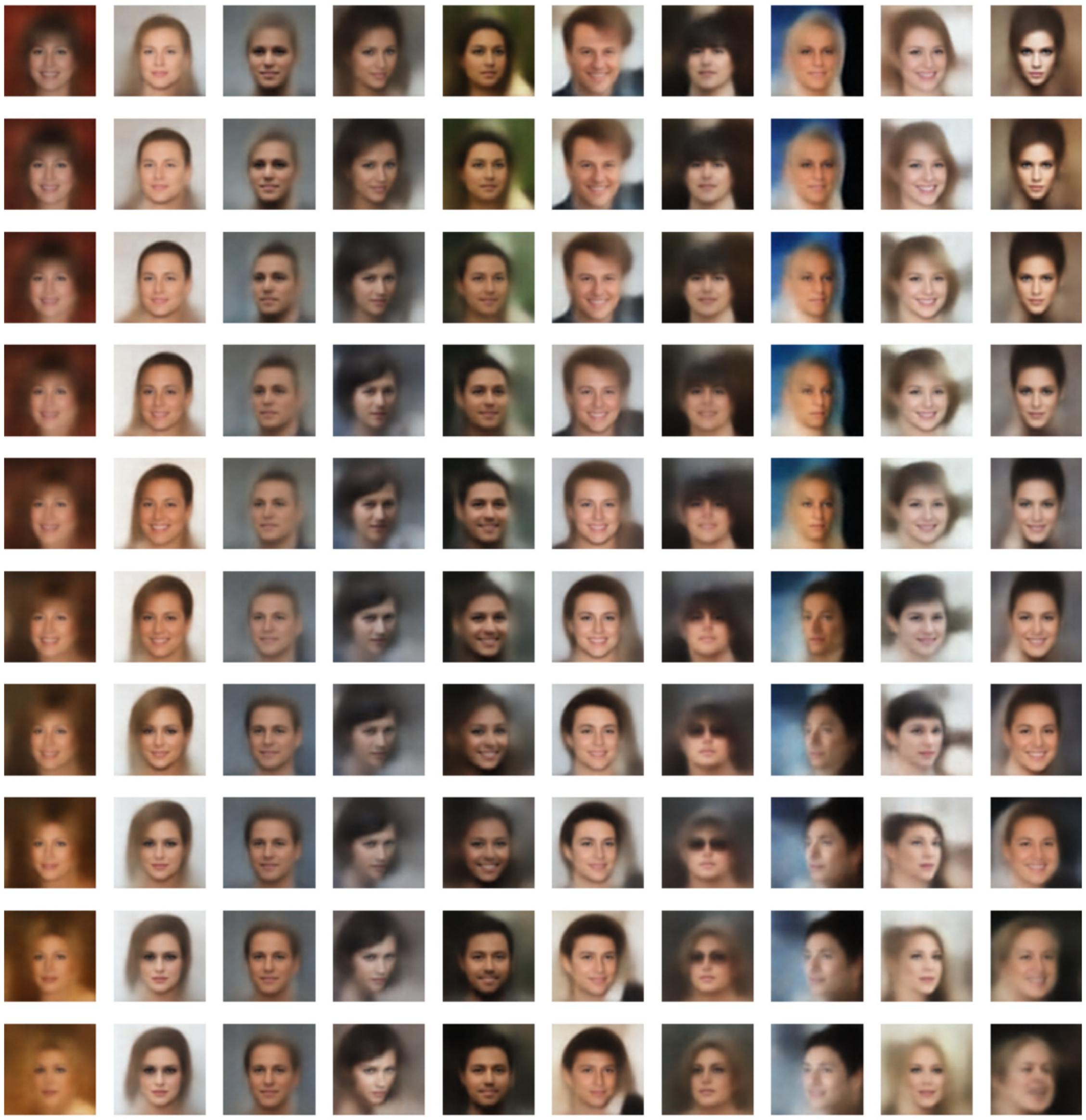

Figure 3.11 – VW sample chain. Each coloumn above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps at temperature = 1, to demonstrate that even over very long chain, the transition operator continues to generate good samples.

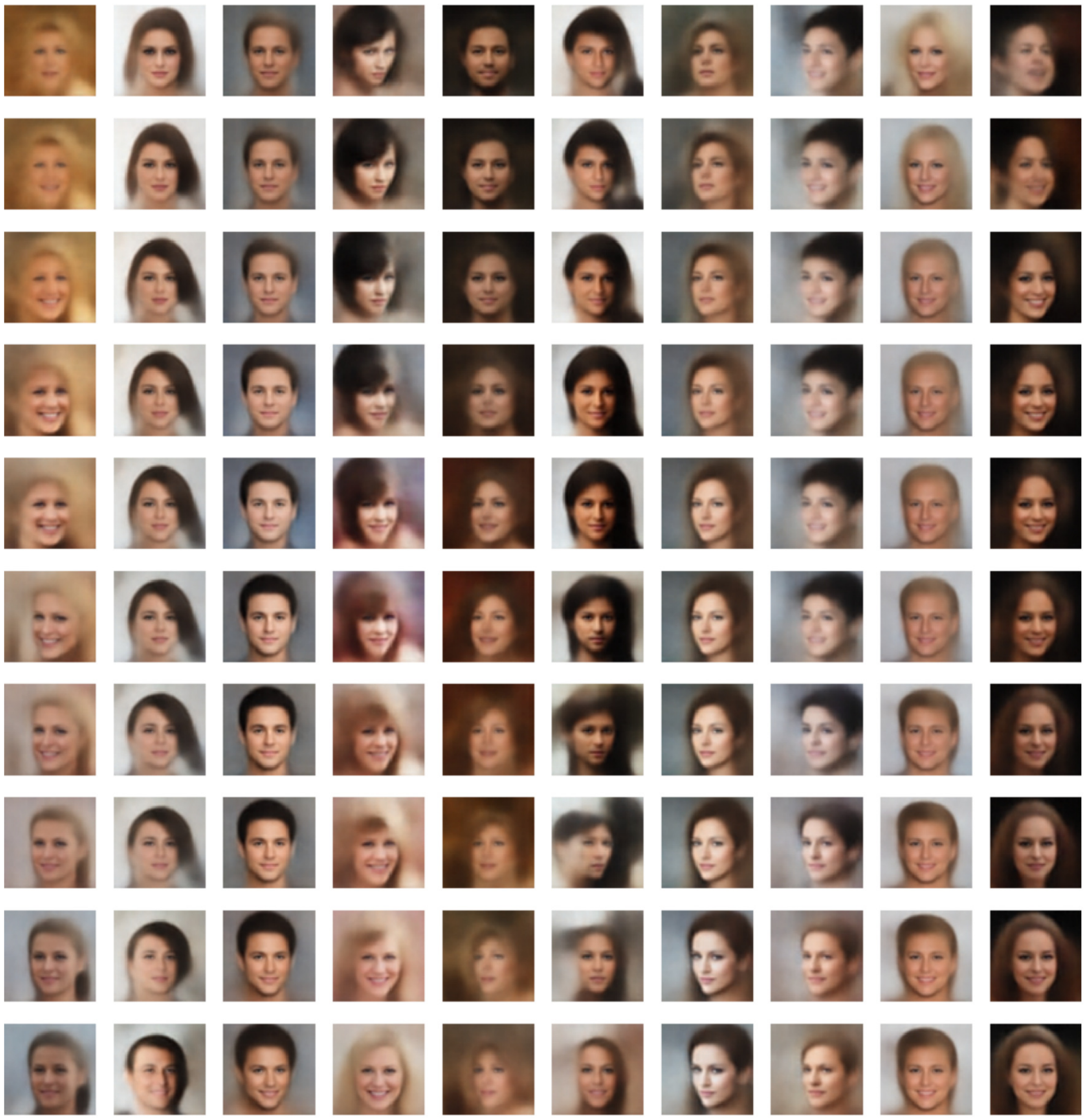

Figure 3.12 – VW sample chain. Each column above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps at temperature = 1, to demonstrate that even over very long chain, the transition operator continues to generate good samples.

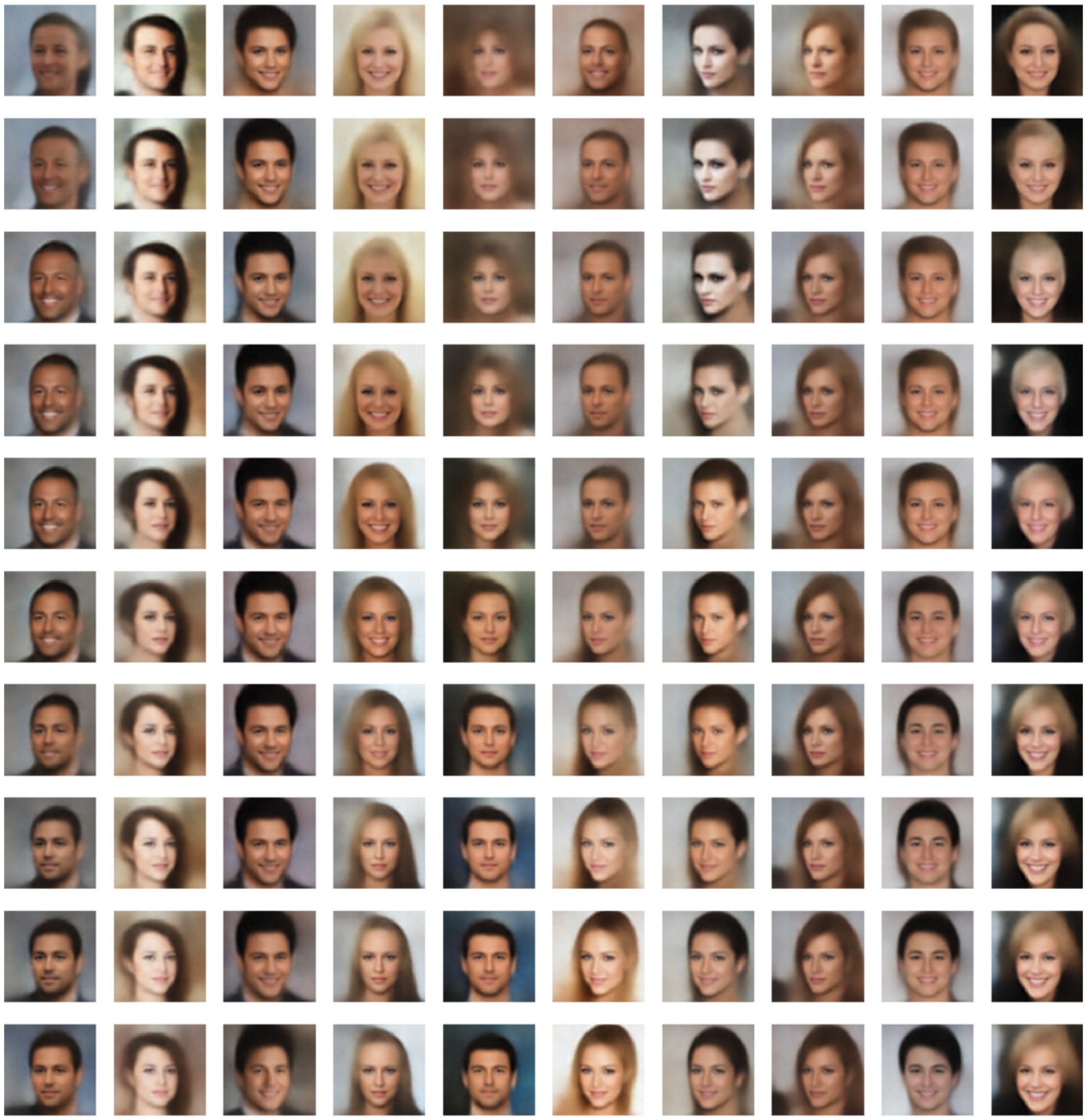

Figure 3.13 – VW sample chain. Each column above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps temperature = 1.

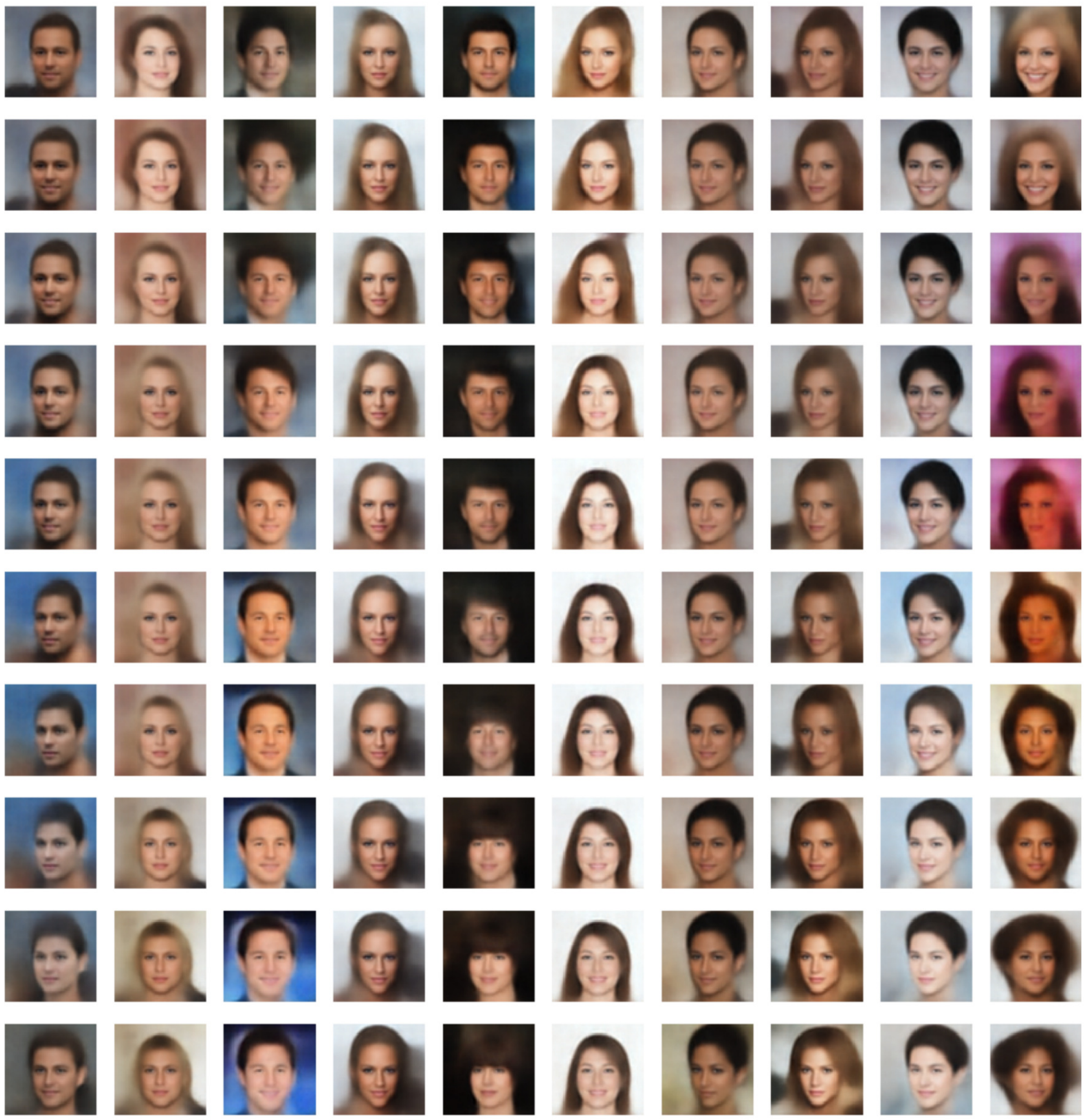

Figure 3.14 – VW sample chain. Each column above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps at temperature = 1, to demonstrate that even over very long chain, the transition operator continues to generate good samples.

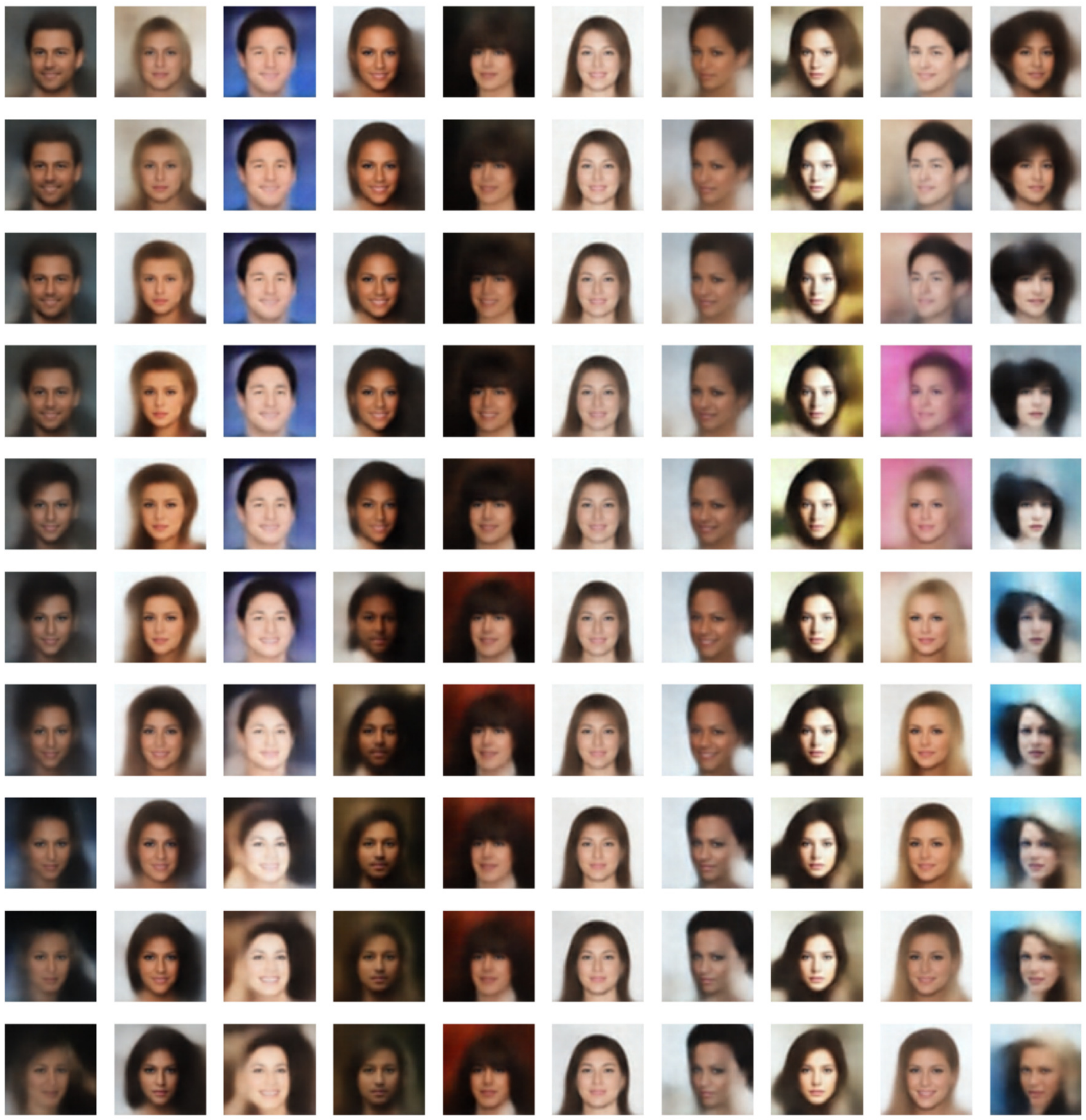

Figure 3.15 – VW sample chain. Each column above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps at temperature = 1, to demonstrate that even over very long chain, the transition operator continues to generate good samples.

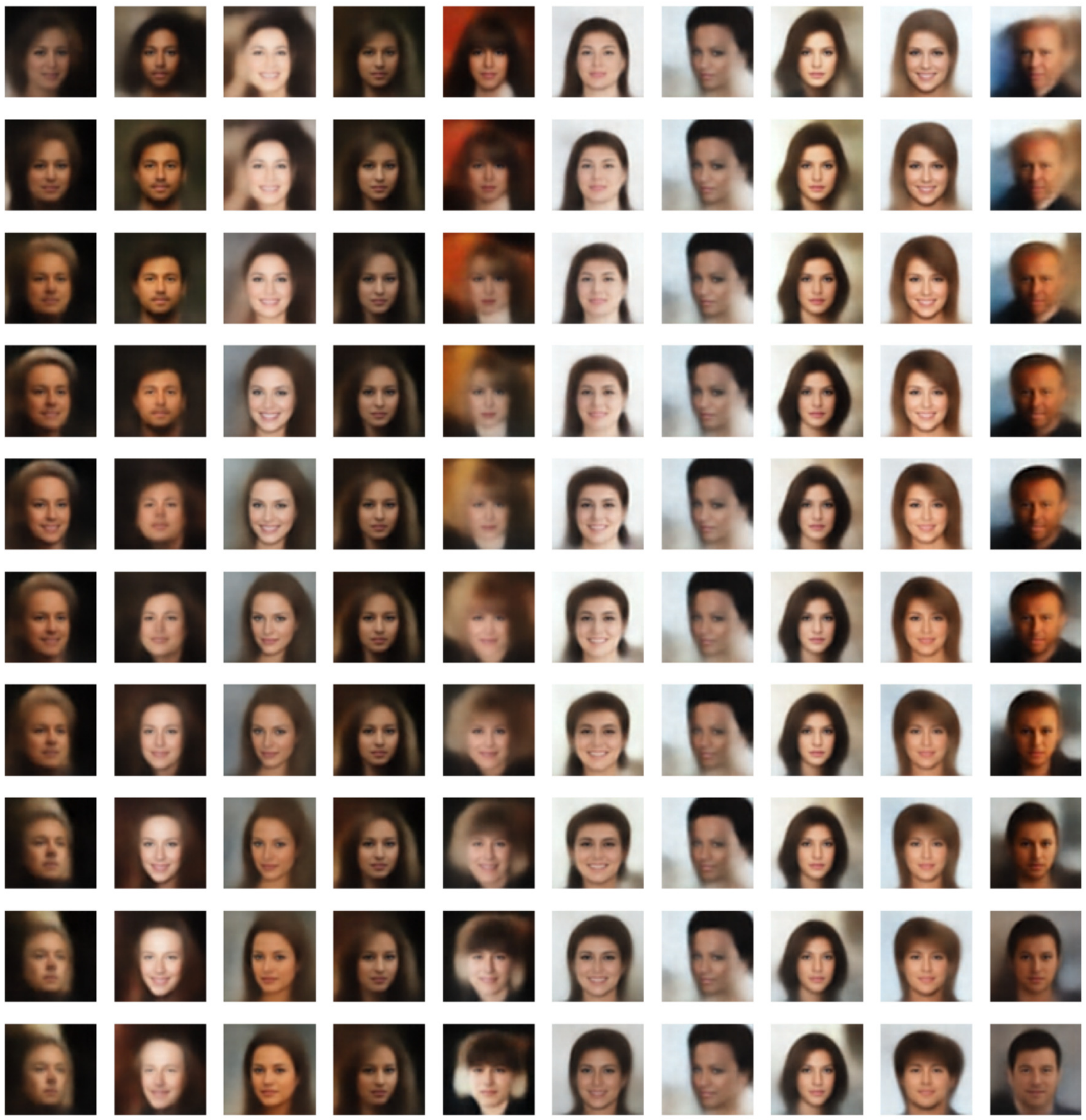

Figure 3.16 – VW sample chain. Each column above corresponds to one sampling chain. We have shown every 10th sample. We applied the transition operator for 5000 time-steps at temperature = 1, to demonstrate that even over very long chain, the transition operator continues to generate good samples.

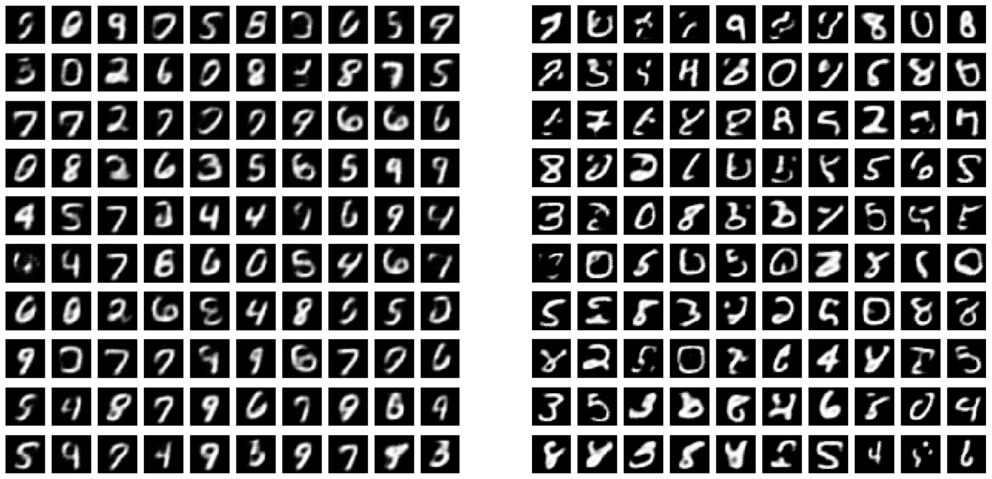

Figure 3.17 – Samples from two VW models (left and right) which have a higher lower bound than the one whose samples are shown in Figure 5 (and comparable but slightly better importance sampling estimators of the log-likelihood): yet, the generated samples are clearly not as good, suggesting that either the bound is sometimes not tight enough or that the log-likelihood is not always a clear indicator of sample quality.

4 Conclusion

The idea of matching behavior of a model when it is running on its own, making predictions, generating samples, etc. vs when it is forced to be consistent with observed data is an old and powerful one. In chap 2, we introduce Professor Forcing, an instance of this idea when the model of interest is a recurrent generative one, and which relies on training an auxiliary model, the discriminator to spot the differences in behavior between these two modes of behavior. A major motivation for this approach is that the discriminator can look at the statistics of the behavior and not just at the single-step predictions, forcing the generator to behave the same when it is constrained by the data and when it is left generating outputs by itself for sequences that can be much longer than the training sequences.

Future work would be to use the proposed method to model of the environment for reinforcement learning problems. Training control algorithms efficiently from interactions with the environment is a central issue in reinforcement learning (RL). Model-free RL methods, combined with deep neural networks, have achieved impressive results across a wide range of domains Lillicrap et al. [2015], Mnih et al. [2016], Silver et al. [2016]. However, existing model-free solutions lack *sample efficiency*, meaning that they require extensive interaction with the environment to achieve these levels of performance. As long as it is possible to scale up the number of available samples, progress can still be made. Increasing computational power, for example, allows for parallelism across simulated environments Mnih et al. [2016]. However, when we seek to deploy RL algorithms in the real world, limited sample efficiency can be problematic, since data is relatively slow and expensive to acquire.

Model-based methods in RL can mitigate this issue. These approaches learn an unsupervised model of the underlying dynamics of the environment, which does not necessarily require rewards, as the model observes and predicts state-to-state transitions. Since states are generally high-dimensional, this form of unsupervised learning provides a rich source of information to the learner. With a well-trained model, the algorithm can then simulate the environment and look ahead to future events to establish better value estimates. This allows the algorithm to explore various possibilities and contingencies without requiring expensive interactions with the environment. As a result, model-based methods can be more sample efficient than their model-free counterparts, but often do not achieve the same asymptotic performance Deisenroth and Rasmussen [2011], Nagabandi et al. [2017]. In complex domains for which an exact simulator is not available to the agent, the performance of model-based agents employing standard planning methods usually suffers from model bias resulting from inaccurate function approximation Ross and Bagnell [2012]. However, learning a world model involves taking a sample, its open-loop predictions may diverge from the manifold of latent states the model has seen during training. I think, the proposed method in chap 2, could be useful to mitigate this issue.

In chap 3, we introduced a new approach to learning non-energy-based transition operators which inherits advantages from several previous generative models, including a training objective that requires rapidly generating the data in a finite number of steps (as in directed models), re-using the same parameters for each step (as in undirected models), directly parametrizing the generator (as in GANs and DAEs), and using the model itself to quickly find its own spurious modes (the walk-back idea).

A fundamental aspect of our approach is that we can train stochastic processes that need not exactly obey detailed balance, yielding access to a larger and potentially more powerful space of models. In particular, this enables us to relax the weight symmetry constraint of undirected graphical models corresponding to neural networks, yielding a more brain like iterative computation characteristic of asymmetric biological neural circuits. This yields an intriguing connection to the neurobiology of dreams. As discussed in Bengio et al. [2015c], spike-timing dependent plasticity (STDP), a plasticity rule found in the brain [Markram and Sakmann, 1995], to increasing the probability of configurations towards which the network goes (i.e., remembering observed configurations), while reverse-STDP corresponds to forgetting or unlearning the states towards which the model goes (which is what happens with the VW update). If, as suggested, the neurobiological function of sleep involves re-organizing memories and in particular unlearning spurious modes through reverse-STDP, then the relative incoherence of dreams compared to reality is qualitatively consistent with heated destructive dynamics of VW, compared to the cooled transition operator in place during awake states.

In addition, consider energy-based models with energy terms of the form $E_{ij} = w_{ij} f_{ij}(s_i, s_j)$ linking unit i and unit j, like in Boltzmann machines or Hopfield nets. Inspecting the gradient of the energy w.r.t. s_i shows that the push on s_i to reduce the energy involves a term in w_{ij} as well as a term in w_{ji}. It means that updating s_i depends not just on the weight of the synapse from j to i but also on the weight of the synapse from i to j. This is called *weight transport* [Lillicrap et al., 2014] and is not biologically plausible. In the case where $f_{ij} = f_{ji}$ this leads to symmetry of the weights, i.e., the weight matrix is a symmetric matrix. With VW, this hard constraint goes away, although the training procedure itself may converge towards more symmetry. This is consistent with both empirical observations [Vincent et al., 2010] and theoretical analysis [Arora et al., 2015] of auto-encoders, for which symmetric weights are associated with minimizing reconstruction error.

Many questions remain open to analyze and extend VW, in particular to incorporate latent layers. The state at each step would now include both visible $\boldsymbol{x}$ and latent $\boldsymbol{h}$ components. Essentially the same procedure can be run, except for the chain initialization, with $\boldsymbol{s}_0 = (\boldsymbol{x}, \boldsymbol{h}_0)$ where $\boldsymbol{h}_0$ a sample from the posterior distribution of $\boldsymbol{h}$ given $\boldsymbol{x}$. Another interesting direction is to replace the log-likelihood objective at each step by a GAN-like objective, thus avoiding the need to inject noise independently on each of the pixels, during each transition step, and allowing latent variable sampling to inject the required high-level decisions associated with the transition. Based on the earlier results from [Bengio et al., 2013a], sampling in the latent space rather than in the pixel space should allow for better generative models and even better mixing between modes [Bengio et al., 2013a].

Overall, our work takes a step to filling a relatively open niche in the machine learning literature on *directly* training non-energy-based iterative stochastic operators, and we hope that the many possible extensions of this approach could lead to a rich new class of more powerful brain-like machine learning models.

Bibliographie

H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, and M. Marchand. Domain-Adversarial Neural Networks. *ArXiv e-prints*, December 2014.

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, and et al. Theano: A python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688, 2016. URL http://arxiv.org/abs/1605.02688.

Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *J. Mach. Learn. Res.*, 15(1):3563–3593, January 2014a. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2750359.

Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *Journal of Machine Learning Research*, 15(1): 3563–3593, 2014b.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. Why are deep nets reversible: A simple theory, with implications for training. *CoRR*, abs/1511.05653, 2015. URL http://arxiv.org/abs/1511.05653.

D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio. An Actor-Critic Algorithm for Sequence Prediction. *ArXiv e-prints*, July 2016.

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

Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. *arXiv preprint arXiv:1508.04395*, 2015.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015a.

Y. Bengio, N. Léonard, and A. Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *ArXiv e-prints*, August 2013.

Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations. In *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*. ACM, 2013a. URL http://icml.cc/2013/.

Yoshua Bengio, Eric Thibodeau-Laufer, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. *CoRR*, abs/1306.1091, 2013b. URL http://arxiv.org/abs/1306.1091.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. *CoRR*, abs/1305.6663, 2013c. URL http://arxiv.org/abs/1305.6663.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *NIPS'2013, arXiv:1305.6663*, 2013d.

Yoshua Bengio, É ric Thibodeau-Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–226–II–234. JMLR.org, 2014. URL http://dl.acm.org/citation.cfm?id=3044805.3044918.

Yoshua Bengio, Dong-Hyun Lee, Jörg Bornschein, and Zhouhan Lin. Towards biologically plausible deep learning. *CoRR*, abs/1502.04156, 2015b. URL http://arxiv.org/abs/1502.04156.

Yoshua Bengio, Thomas Mesnard, Asja Fischer, Saizheng Zhang, and Yuhai Wu. An objective function for STDP. *CoRR*, abs/1509.05936, 2015c. URL http://arxiv.org/abs/1509.05936.

Florian Bordes, Sina Honari, and Pascal Vincent. Learning to generate samples from noise through infusion training. *CoRR*, abs/1703.06975, 2017. URL http://arxiv.org/abs/1703.06975.

Alexandre Brebisson. Conditional handwriting generation in theano. https://github.com/adbrebs/handwriting, 2016.

Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. Accurate and conservative estimates of MRF log-likelihood using reverse annealing. *CoRR*, abs/1412.8566, 2014. URL http://arxiv.org/abs/1412.8566.

Xinlei Chen and C Lawrence Zitnick. Mind's eye: A recurrent visual representation for image caption generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2422–2431, 2015.

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

Kyunghyun Cho, Bart Van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014b. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1179.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585, 2015.

Gavin E Crooks. Path-ensemble averages in systems driven far from equilibrium. *Physical review E*, 61(3):2361, 2000.

H. Daumé, III, J. Langford, and D. Marcu. Search-based Structured Prediction. *ArXiv e-prints*, July 2009.

Peter Dayan, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel. The helmholtz machine. *Neural Computation*, 7(5):889–904, 1995a. doi: 10.1162/neco.1995.7.5.889. URL https://doi.org/10.1162/neco.1995.7.5.889.

Peter Dayan, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel. The helmholtz machine. *Neural Comput.*, 7(5):889–904, September 1995b. ISSN 0899-7667. doi: 10.1162/neco.1995.7.5.889. URL http://dx.doi.org/10.1162/neco.1995.7.5.889.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-Adversarial Training of Neural Networks. *ArXiv e-prints*, May 2015.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. *arXiv preprint arXiv:1502.03509*, 2015.

Todd R Gingrich, Grant M Rotskoff, Gavin E Crooks, and Phillip L Geissler. Near-optimal protocols in complex nonequilibrium transformations. *Proceedings of the National Academy of Sciences*, page 201606273, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron C. Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4601–4609, 2016. URL http://papers.nips.cc/paper/6099-professor-forcing-a-new-algorithm-for-training-recurrent-networks.

Anirudh Goyal, Nan Rosemary Ke, Surya Ganguli, and Yoshua Bengio. Variational walkback: Learning a transition operator as a stochastic recurrent net. In

Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, pages 4395–4405, 2017a. URL http://papers.nips.cc/paper/7026-variational-walkback-learning-a-transition-operator-as-a-stochastic-recur

Anirudh Goyal, Nan Rosemary Ke, Alex Lamb, R. Devon Hjelm, Chris Pal, Joelle Pineau, and Yoshua Bengio. Actual: Actor-critic under adversarial learning. *CoRR*, abs/1711.04755, 2017b. URL http://arxiv.org/abs/1711.04755.

A. Graves. Generating Sequences With Recurrent Neural Networks. *ArXiv e-prints*, August 2013.

Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013. URL http://arxiv.org/abs/1308.0850.

Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527. URL http://dx.doi.org/10.1162/neco.2006.18.7.1527.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

F. Huszár. How (not) to Train your Generative Model: Scheduled Sampling, Likelihood, Adversary ? *ArXiv e-prints*, November 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P. Kingma, Tim Salimans, and Max Welling. Improving variational inference with inverse autoregressive flow. *CoRR*, abs/1606.04934, 2016. URL http://arxiv.org/abs/1606.04934.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. 2011.

Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random feedback weights support learning in deep neural networks. *CoRR*, abs/1411.0247, 2014. URL http://arxiv.org/abs/1411.0247.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3730–3738, 2015.

M. Liwicki and H. Bunke. Iam-ondb - an on-line english sentence database acquired from handwritten text on a whiteboard. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 956–961 Vol. 2, Aug 2005. doi: 10.1109/ICDAR.2005.132.

H. Markram and B. Sakmann. Action potentials propagating back into dendrites triggers changes in efficacy. *Soc. Neurosci. Abs*, 21, 1995.

Tomas Mikolov. Recurrent neural network based language model. 2010.

Tomas Mikolov and Geoffrey Zweig. Context dependent recurrent neural network language model. 2012.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

Iain Murray and Ruslan R Salakhutdinov. Evaluating probabilities under high-dimensional latent variable models. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in*

Neural Information Processing Systems 21, pages 1137–1144. Curran Associates, Inc., 2009. URL http://papers.nips.cc/paper/3584-evaluating-probabilities-under-high-dimensional-latent-variable-models.pdf.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11 (2):125–139, April 2001. ISSN 0960-3174. doi: 10.1023/A:1008923215028. URL http://dx.doi.org/10.1023/A:1008923215028.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

Tapani Raiko, Li Yao, Kyunghyun Cho, and Yoshua Bengio. Iterative neural autoregressive distribution estimator NADE-k. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 325–333. Curran Associates, Inc., 2014.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

S. Ross, G. J. Gordon, and J. A. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *ArXiv e-prints*, November 2010.

Stéphane Ross and J. Andrew Bagnell. Agnostic system identification for model-based reinforcement learning. *CoRR*, abs/1203.1007, 2012.

Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial Intelligence and Statistics*, 2009.

Tim Salimans, Diederik P Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. *arXiv preprint arXiv:1410.6460*, 2014.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016. URL http://arxiv.org/abs/1606.03498.

Tim Schmiedl and Udo Seifert. Optimal finite-time processes in stochastic thermodynamics. *Physical review letters*, 98(10):108301, 2007.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David A Sivak and Gavin E Crooks. Thermodynamic metrics and optimal paths. *Physical review letters*, 108(19):190602, 2012.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015. URL http://arxiv.org/abs/1503.03585.

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

L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. *ArXiv e-prints*, November 2015.

L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, Apr 2016. URL http://arxiv.org/abs/1511.01844.

A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel Recurrent Neural Networks. *ArXiv e-prints*, January 2016.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1756006.1953039.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger B. Grosse. On the quantitative analysis of decoder-based generative models. *CoRR*, abs/1611.04273, 2016. URL http://arxiv.org/abs/1611.04273.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.