

Université de Montréal

**Learning to sample from noise with deep generative
models**

par

Florian Bordes

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

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

August 31, 2017

SOMMAIRE

L'apprentissage automatique et spécialement l'apprentissage profond se sont imposés ces dernières années pour résoudre une large variété de tâches. Une des applications les plus remarquables concerne la vision par ordinateur. Les systèmes de détection ou de classification ont connu des avancées majeures grâce à l'apprentissage profond. Cependant, il reste de nombreux obstacles à une compréhension du monde similaire aux êtres vivants. Ces derniers n'ont pas besoin de labels pour classifier, pour extraire des caractéristiques du monde réel. L'apprentissage non supervisé est un des axes de recherche qui se concentre sur la résolution de ce problème.

Dans ce mémoire, je présente un nouveau moyen d'entraîner des réseaux de neurones de manière non supervisée. Je présente une méthode permettant d'échantillonner de manière itérative à partir de bruit afin de générer des données qui se rapprochent des données d'entraînement. Cette procédure itérative s'appelle l'entraînement par infusion qui est une nouvelle approche permettant d'apprendre l'opérateur de transition d'une chaîne de Markov. Dans le premier chapitre, j'introduis des bases concernant l'apprentissage automatique et la théorie des probabilités. Dans le second chapitre, j'expose les modèles génératifs qui ont inspiré ce travail. Dans le troisième et dernier chapitre, je présente comment améliorer l'échantillonnage dans les modèles génératifs avec l'entraînement par infusion.

Mots clés: Apprentissage automatique, Apprentissage profond, Intelligence artificielle, Modèle Génératif, Infusion

SUMMARY

Machine learning and specifically deep learning has made significant breakthroughs in recent years concerning different tasks. One well known application of deep learning is computer vision. Tasks such as detection or classification are nearly considered solved by the community. However, training state-of-the-art models for such tasks requires to have labels associated to the data we want to classify. A more general goal is, similarly to animal brains, to be able to design algorithms that can extract meaningful features from data that aren't labeled. Unsupervised learning is one of the axes that try to solve this problem.

In this thesis, I present a new way to train a neural network as a generative model capable of generating quality samples (a task akin to imagining). I explain how by starting from noise, it is possible to get samples which are close to the training data. This iterative procedure is called Infusion training and is a novel approach to learning the transition operator of a generative Markov chain.

In the first chapter, I present some background about machine learning and probabilistic models. The second chapter presents generative models that inspired this work. The third and last chapter presents and investigates our novel approach to learn a generative model with Infusion training.

Keywords: Machine learning, Deep learning, Artificial Intelligence, Generative model, Infusion

CONTENTS

Sommaire	ii
Summary	iii
List of figures	vii
List of tables	ix
Acknowledgements	x
Introduction	xi
Chapter 1. Machine learning Basics	1
1.1. Introduction	1
1.2. Principle of compositionality and distributed representation	1
1.3. Tasks	2
1.3.1. Classification	2
1.3.2. Regression	2
1.3.3. Encoding and decoding	2
1.3.4. Denoising	2
1.3.5. Generative modeling for sampling	2
1.4. How to evaluate a model	3
1.5. Overfitting, underfitting and Capacity	3
1.6. Neural network	4
1.6.1. Perceptron	4
1.6.2. Multilayer Perceptron	5
1.6.3. Convolutional neural network	5
1.6.4. Autoencoder	6
1.6.5. Activation function	7
1.7. Training and cost function	7

1.8. KL divergence and Maximum likelihood	8
1.8.1. Gradient Descent	8
1.8.2. Regularization	9
1.8.3. Batch normalization	9
1.9. Markov chain	10
1.9.1. Markov chain Monte Carlo	11
Chapter 2. Generative models	12
2.1. Denoising Autoencoder	12
2.2. Variational Autoencoder	13
2.2.1. Variational inference	14
2.3. Generative stochastic networks	15
2.4. Diffusion approach	16
2.5. Variational Walkback	16
Chapter 3. Learning to generate samples from noise through infusion training	18
3.1. abstract	18
3.2. Introduction and motivation	18
3.3. Proposed approach	20
3.3.1. Setup	20
3.3.2. Generative model sampling procedure	20
3.3.3. Infusion training procedure	22
3.3.3.1. The infusion chain	23
3.3.3.2. Denoising-based Infusion training procedure	23
3.3.4. Stochastic log likelihood estimation	24
3.3.4.1. Lower-bound-based infusion training procedure	25
3.4. Relationship to previously proposed approaches	25
3.4.1. Markov Chain Monte Carlo for energy-based models	25
3.4.2. Variational auto-encoders	26
3.4.3. Sampling from autoencoders and Generative Stochastic Networks	26
3.4.4. Reversing a diffusion process in non-equilibrium thermodynamics	27

3.5. Experiments	27
3.5.1. Numerical results	28
3.5.2. Sample generation	30
3.5.3. Inpainting	30
3.6. Conclusion and future work	30
Appendix A. Details on the experiments	33
A.1. MNIST experiments	33
A.1.1. Infusion and model sampling chains on natural images datasets	34
Conclusion	41
Bibliography	42

LIST OF FIGURES

1.1	Plots showing how the capacity of a model influence its ability to fit data points. Train points are blue and test points are red.	4
1.2	A model of a Perceptron.....	5
1.3	A Multilayer Perceptron.....	6
1.4	An autoencoder with intermediate representation h	6
1.5	Examples of activation function.....	7
1.6	An example of Markov chain.....	10
2.1	A denoising autoencoder with intermediate representation h	12
2.2	Manifold learning with a DAE. The model learns to project the corrupted data to the manifold of the true data. (Figure from [36]).....	13
2.3	A Variational Autoencoder.....	14
2.4	A Generative stochastic networks.....	16
3.1	The <i>model sampling chain</i> . Each row shows a sample from $p(\mathbf{z}^{(0)}, \dots, \mathbf{z}^{(T)})$ for a model that has been trained on MNIST digits. We see how the learned Markov transition operator progressively denoises an initial unstructured noise sample. We can also see that there remains ambiguity in the early steps as to what digit this could become. This ambiguity gets resolved only in later steps. Even after a few initial steps, stochasticity could have made a chain move to a different final digit shape.	21
3.2	Training <i>infusion chains</i> , infused with target $\mathbf{x} = \mathbf{3}$. This figure shows the evolution of chain $q(\mathbf{z}^{(0)}, \dots, \mathbf{z}^{(30)} \mathbf{x})$ as training on MNIST progresses. Top row is after network random weight initialization. Second row is after 1 training epochs, third after 2 training epochs, and so on. Each of these images were at a time provided as the input part of the (<i>input</i> , <i>target</i>) training pairs for the network. The network was trained to denoise all of them into target 3. We see that as training progresses, the model has learned to pick up the	

	cues provided by target infusion, to move towards that target. Note also that a single denoising step, even with target infusion, is not sufficient for the network to produce a sharp well identified digit.	24
3.3	Training curves: lower bounds on average log-likelihood on MNIST as infusion training progresses. We also show the lower bounds estimated with the Parzen estimation method.	29
3.4	Parzen-window-based estimator of lower bound on average test log-likelihood on MNIST (in nats).	29
3.5	Mean predictions by our models on 4 different datasets. The rightmost column shows the nearest training example to the samples in the next-to last column.	31
3.6	Inpainting on CelebA dataset. In each row, from left to right: an image from the test set; the same image with bottom half randomly sampled from our factorial prior. Then several end samples from our sampling chain in which the top part is clamped. The generated samples show that our model is able to generate a varied distribution of coherent face completions.	32
A.1	Training curves on MNIST showing the log likelihood lower bound (nats) for different infusion rate schedules and different number of steps. We use an increasing schedule $\alpha^{(t)} = \alpha^{(t-1)} + \omega$. In each sub-figure for a fixed number of steps, we show the lower bound for different infusion rates.	35
A.2	Comparing samples of constant infusion rate versus an increasing infusion rate on infused and generated chains. The models are trained on MNIST in 15 steps. Note that having an increasing infusion rate with a small value for ω allows a slow convergence to the target distribution. In contrast having a constant infusion rate leads to a fast convergence to a specific point. Increasing infusion rate leads to more visually appealing samples. We observe that having an increasing infusion rate over many steps ensures a slow blending of the model distribution into the target distribution.	38
A.3	Infusion chains (Sub-Figure A.3a) and model sampling chains (Sub-Figure A.3b) on CIFAR-10.	39
A.4	Infusion chains (Sub-Figure A.4a) and model sampling chains (Sub-Figure A.4b) on CelebA.	40

LIST OF TABLES

3.1	Log-likelihood (in nats) estimated by AIS on MNIST test and training sets as reported in [38] and the log likelihood estimates of our model obtained by infusion training (last three lines). Our initial model uses a Gaussian output with diagonal covariance, and we applied both our lower bound and importance sampling (IS) log-likelihood estimates to it. Since [38] used only an isotropic output observation model, in order to be comparable to them, we also evaluated our model after replacing the output by an isotropic Gaussian output (same fixed variance for all pixels). Average and standard deviation over 10 repetitions of the evaluation are provided. Note that AIS might provide a higher evaluation of likelihood than our current IS estimate, but this is left for future work.....	29
3.2	Inception score (with standard error) of 50 000 samples generated by models trained on CIFAR-10. We use the models in Salimans, Goodfellow, Zaremba, Cheung, Radford, and Chen [31] as baseline. 'SP' corresponds to the best model described by Salimans et al. [31] trained in a semi-supervised fashion. 'L' corresponds to the same model after removing the label in the training process (unsupervised way), '-MBF' corresponds to a supervised training without minibatch features.....	30
A.1	Infusion rate impact on the lower bound log-likelihood (test set) and the samples generated by a network trained with different number of steps. Each sub-table corresponds to a fixed number of steps. Each row corresponds to a different infusion rate, where we show its lower bound and also its corresponding generated samples from the trained model. Note that for images, we show the mean of the Gaussian distributions instead of the true samples. As the number of steps increases, the optimal infusion rate decreases. Higher number of steps contributes to better qualitative samples, as the best samples can be seen with 15 steps using $\alpha = 0.01$	36

ACKNOWLEDGEMENTS

I would like to thank all people who helped me those last years. Especially, I would like to thank my supervisor Pascal Vincent who gave me the opportunity to work at MILA. I am extremely grateful for his continuous support. Also I would like to thank my colleagues Tom Bosc, Gauthier Gidel, Cesar Laurent, Thomas George, Tristan Sylvain, Margaux Luck, Akram Erraqabi, Hugo Bérard and Ahmed Touati for their great help and comments.

INTRODUCTION

Nowadays, numeric systems are collecting a huge amount of data. Dealing with big data, need systems that are able to read and extract relevant information very fast. It implies the ability to understand how those data are structured and how they can change. However, analyzing and extracting meaningful features is a complex task. Consider the following example, when we want to draw an animal, we have to think about the different parts of this animal. If we are able to make a model that can draw new data, it means that the model has an enough good understanding about the essence of those data. This is the principle behind generative model. For example, a model that generates images means that the model has learned some representation of the world i.e that an object is more likely to lay at the bottom of images than at the top. If we follow this principle, this model will be able to reconstruct images with only few bits of informations since it knows some basics about the world. With this kind of method, we can imagine a full virtual world compressed inside few bits of informations. However training a generative model is hard, and it is a current active field of research.

Chapter 1

MACHINE LEARNING BASICS

This chapter presents a basic overview of the machine learning notions that are mentioned in this thesis.

1.1. INTRODUCTION

Machine learning can be described as acquiring the ability to learn from data. Animal brains have the ability to generalize, to understand concepts they have never seen before. They learn a representation of the world such that, if they see another animal, they can recognize it in different contexts. One of the most fascinating question concerning intelligence can be "How a brain learns a representation of the world?". Some machine learning algorithms like neural networks aim to construct an understanding of sensory data by learning a higher level, more meaning-carrying representation. Once we get this representation of the data, this allows us to classify, to describe or to draw new data.

1.2. PRINCIPLE OF COMPOSITIONALITY AND DISTRIBUTED REPRESENTATION

One prior used in some machine learning algorithms is the Principle of compositionality, whereby a complex expression is determined by its constituent expressions and by the rules that combine them. It makes the assumption that any objects in the world can be explained by a subset of meaningful features. For example, if we think about a cat, we can recognize it because it's small, furry, it has a tail and it can meow. There is a vector of features that would enable us to recognize a cat. When we recognize a dog, there are some of those features that are common with the cat (small, tail). This can explain, when we see a new specie we have never seen, why we may think "it looks like a cat" because we detect those common features. Traditional machine learning algorithms such as k-means learn a local representation that doesn't allow the existence of any compositionality (a number n of classes implies a number n of representations). Neural networks assume a distributed representation that implies this

principle of compositionality. A recent study[16] provided evidence that brains use this kind of distributed representation to recognize faces.

1.3. TASKS

Examples of common tasks that can be resolved by machine learning algorithms are classification, detection, segmentation or generation. Another example of a successful application is AlphaGo that used neural networks to learn to play Go[32] beating human champions. Machine learning can be used in a wide range of applications that imply analyzing data.

1.3.1. Classification

The purpose of classification is to associate a label $y \in 1, \dots, K$ to a data x by learning a function f such that $y = f(x)$. Some well-known classification tasks are object detection, face recognition or fraud detection.

1.3.2. Regression

Similar to classification, in a regression task, we will try to learn a function f such that given an input x , we will predict a numeric value $y \in \mathbb{R}$ such that $y = f(x)$. An example of a regression task is to predict the fuel price given information about production level.

1.3.3. Encoding and decoding

When we have to deal with high dimensional data, it can be useful to learn a compressed representation of a data x . The task of encoding is to learn a representation z_x of x by using an encoding function f_{enc} such that $z_x = f_{enc}(x)$. Similarly, the task of decoding is to recover x from its compressed representation z_x by using a decoder f_{dec} such that $x = f_{dec}(z_x)$. We call an autoencoder a neural network that learns f_{enc} and f_{dec} .

1.3.4. Denoising

We get a data x , we apply some corruption with a function c on x , such that $\tilde{x} = c(x)$. We try to learn a function f that can recover x from \tilde{x} such that $x \approx f(\tilde{x})$. A denoising auto-encoder[37] is a neural network trained to perform such a denoising task, with the goal of extracting a useful intermediate representation in the process.

1.3.5. Generative modeling for sampling

Generative modeling with sampling can be considered one of the hardest tasks, it implies the ability to generate data that resemble the data used during training in the sense that they should ideally follow the same (unknown) true distribution. If we have data x that are generated from an unknown distribution p such that $x \sim p(x)$ we try to approximate p by

learning a distribution q - from which we can efficiently sample - that is close enough to p . This task is intimately related to probabilistic modeling (and probability density estimation), but the focus is on the ability to generate good samples efficiently, rather than obtaining a precise numerical estimation of the probability density at a given point.

1.4. HOW TO EVALUATE A MODEL

In order to measure the performance of a learned model, we should define a measure that computes whether the model is good for a given task of interest or not. In case of a model for a classification task, trained on a training set D_x , we want to evaluate if this model has learned to generalize well on new data. To measure this generalization performance, we define a validation set that contains data that are not in the training set and we observe the classification score over this validation set. Once the training is done (i.e we get the best score over the validation set), we use a test set that contains data that are neither in the training or validation set and we use this score to compare against other models. In supervised learning, assessing a model is easier because we have the ground-truth labels associated to the data and we can see on the test set, if the labels predicted by the model are correct or not. By contrast in unsupervised learning, it's very hard to tell if a neural net has learned good features or to measure to what degree it generates nice samples. In autoencoder-like models, we often look at the reconstruction error with respect to some metric such as the Euclidean metric. But the pixel-level Euclidean metric is a very poor measure of visual similarity. If the reconstruction is merely translated by one pixel compared to the input, the reconstruction error may be high even if the model had a good understanding of the object. A new approach that attempts to provide an objective numerical measure of the "quality" of sampled images is the inception score[31], with a classifier that is a pretrained model on a dataset called Imagenet[6], it implies looking at the classification error of the samples generated by the model under this classifier and see if it's close to the score of true data. This approach has several limits and the problem of evaluating generative model is still an active field of research.

1.5. OVERFITTING, UNDERFITTING AND CAPACITY

We can informally define the capacity of a model or a machine learning algorithm as the richness and flexibility of the set of function it is capable of representing. If the capacity is too big, the model will overfit on the data and will not be able to generalize well. It corresponds to the case where the model has learned all the training data «by heart» without being able to extract meaningful abstractions about those data. Small capacity implies that the model will not be able to learn a good enough representation of the data.

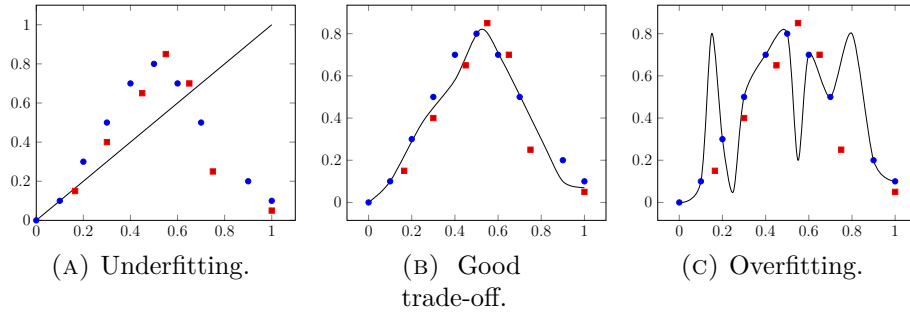


FIG. 1.1. Plots showing how the capacity of a model influence its ability to fit data points. Train points are blue and test points are red.

In Figure 1.1a, the curve doesn't fit all the data, given a new point it will not be able to make a good prediction. In Figure 1.1c, the learning curve fit all the training points however it doesn't generalize to test points. We get a good trade-off in Figure 1.1b between fitting the points and being able to make a good prediction on new points. Getting this kind of trade-off is essential in machine learning, we don't want to have a model that will get overrated good score on the training points and very poorly on test points. Neural networks are often models with high capacity. To avoid overfitting, we use a technique called regularization. An example of regularization is early stopping: if the error on the training points decrease and the error on validation points start to increase, we stop the training and keep the model that has the best score on the validation points.

1.6. NEURAL NETWORK

1.6.1. Perceptron

Inspired by a biological view (especially by neurons), Rosenblatt invented the perceptron[28]. Each neuron in the brain is connected to other neurons by synaptic connections. When a neuron receives a stimuli, it can fire (i.e send a signal to other neurons) or do nothing. The perceptron behavior is similar to a neuron, it is connected to many inputs with some weights (that symbolise the synapse) and it has a function that determines if the perceptron should send a signal or not given those inputs. We call this function an activation function. Figure 1.2 is an example of a perceptron that receives as input a vector $x = x_1, x_2$, and give an output o after being computed by a function f .

The output of a perceptron can be expressed as:

$$o = f\left(\sum_{i=0}^n x_i w_i\right)$$

with x that represents the inputs, w_i the synaptic weights and f an activation function that decides if the neuron should fire or not or "how much" it should fire. In the historic

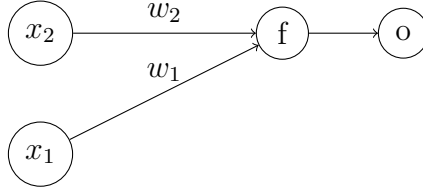


FIG. 1.2. A model of a Perceptron

Perceptron of Rosenblatt, f was taken as the heaviside function. This perceptron can learn very basic things like logical operation such as AND, OR. However it cannot learn a XOR because this problem isn't linearly separable (We cannot draw a line to cluster each input). This is one of the main arguments that has lead to the first AI Winter[23]. Since then, the heaviside function has been replaced by differentiable functions such as the sigmoid (see further below), which enable gradient computations. And the single-adaptive-neuron perceptron of Rosenblatt has been replaced by a network of adaptive neurons, simultaneously trained by gradient descent (see section 1.7.1), the simplest form of which is the multilayer perceptron.

1.6.2. Multilayer Perceptron

Multilayer Perceptron (MLP) is a generalization of the perceptron that can resolve non-linear problem like the XOR operation[28]. It is also called FeedForward network because the information is propagate from the input to the output. The advantage of a MLP is that we can stack layers of perceptrons (called hidden layers) to increase the depth of the network. Having multiple layers allows the MLP to exploit the principle of compositionality of simple elements to represent complicated functions and to learn different levels of abstract features. Figure 1.3 corresponds to a MLP with two hidden layers. MLPs were proven to be universal function approximators [11] i.e. if we have enough hidden units, we can approximate any function by an MLP however we have no proof about the true number of units we need to approximate such function. Each hidden unit can be expressed as:

$$h_i^d = f\left(\sum_{i=0}^n w_i^d h_i^{d-1}\right)$$

where h^0 corresponds to input x .

1.6.3. Convolutional neural network

The success of Convolutional neural networks[18] (CNN) is one reason of the recent popularity of neural networks. Instead of applying for each input the sum over all the parameters w_i , we consider the parameters as a small grid called a kernel that is shared across the input x_i . Those parameters will force the network to learn local features that correspond to some meaningful parts of an object (in case where the inputs are images). For

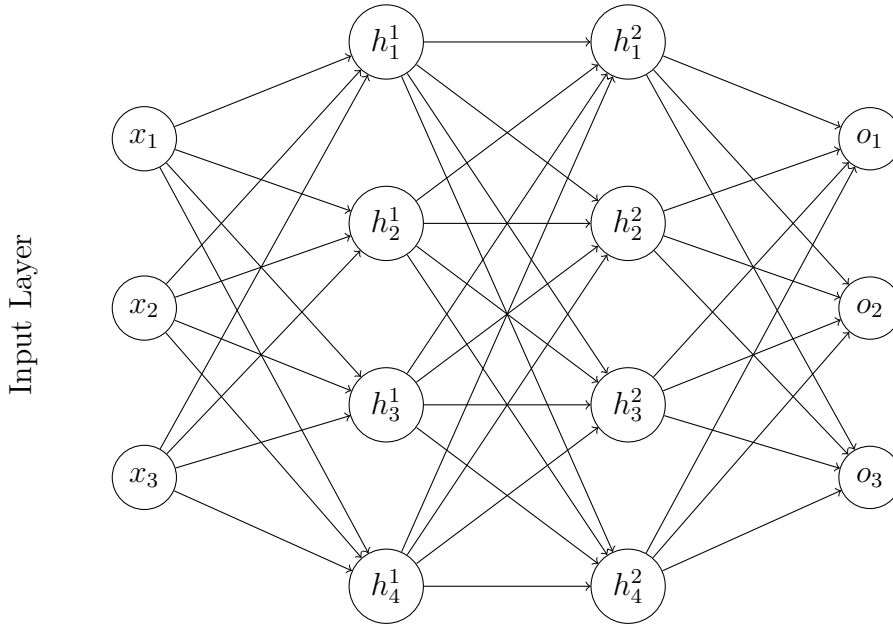


FIG. 1.3. A Multilayer Perceptron

example, first layers of a CNN will learn efficient edge detection and the last layers will learn more specific features.

1.6.4. Autoencoder

An autoencoder (AE) is a feedforward neural network that is trained in an unsupervised way. Its goal is to learn a function f that reproduces as best as it can its input such that $x \approx f(x)$. To avoid the autoencoder learning an identity function (Because we want the AE to be able to generalize, to extract useful features from the data), we use an intermediate representation called h . In Figure 1.4, we separate f as two function, f_{enc} called encoder that maps the inputs x to h and f_{dec} called decoder that maps the representation h to the input space of x . One key capability of the autoencoders is that they can perform

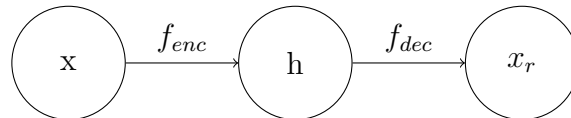


FIG. 1.4. An autoencoder with intermediate representation h

dimensionality reduction by specifying the size of the intermediate latent space h . If the dimension of h is lower than the input space, we qualify the autoencoder as undercomplete, then we expect it to extract some meaningful compact features about the data. However if the dimension of h is too high (i.e. superior to the input dimension), the AE can learn to copy its input and will not be able to generalize well, we call it an overcomplete autoencoder. In modern architectures, often space of the features we want to extract from the data can be

bigger than the original input space. In order to avoid learning the uninteresting solution of copying the input, one can use some regularization during the training of the autoencoder. The variational and denoising autoencoder presented in chapter 3 are two special cases of regularized autoencoders.

1.6.5. Activation function

The choice of an activation function is essential when building a neural net. Historically, the sigmoid and tanh function were the most used in neural networks. One of the disadvantages of those functions is that because they can be easily saturated, they don't give meaningful gradients in this regime. Rectifier Linear Unit[8] (Relu) address this issue by being saturate only when $x < 0$. Plots of those functions can be seen on Figure 1.5. One concern with Relu is that, it doesn't give any guarantee against exploding gradients (gradients getting too large because they are not bounded) which can lead to a hard task of optimization.

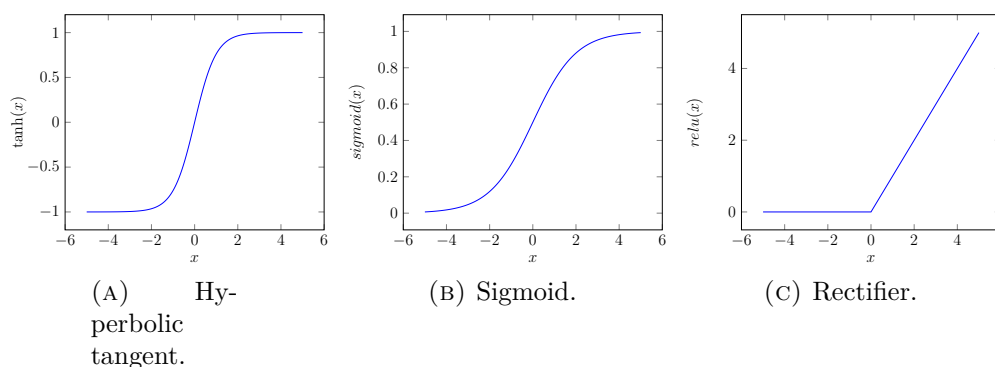


FIG. 1.5. Examples of activation function

1.7. TRAINING AND COST FUNCTION

A classifier or neural network needs to be trained to learn to extract features from data. The features learned by a neural network depend among other things on the cost function used during training. The cost function defines the task we want to solve. If we want the ability to classify, we will train our network to minimize the classification error over training points. Once we have defined the network architecture, i.e. the predictive function it computes, as well as the cost function, learning can be viewed as an optimization problem where we try to minimize the cost function. In order to minimize a function, we need to have an efficient algorithm. The most well-known and used algorithm to do that is gradient descent. Formally, we typically want to learn the set of parameters θ of a predictive function f_θ . For a neural network, θ is the set of synaptic weights. We have some measure of loss

$L(f_\theta(x), y)$ incurred when the network prediction is $f(x)$ while the true target was y . And we want to find the parameters θ that minimize the sum of losses over the training set:

$$J(\theta) = \sum_{(x,y) \in D_{train}} L(f_\theta(x), y)$$

1.8. KL DIVERGENCE AND MAXIMUM LIKELIHOOD

A cost usually taken in neural networks is the KL divergence. If we have a true but unknown distribution that generates data called $p_{data}(x)$, we would like to have a distribution $p_{model}(x; \theta)$ of parameters θ close enough of $p_{data}(x)$. To do so we need to minimize a divergence between both distributions. A divergence that is usually used is the KL divergence defined by:

$$D_{KL}(p_{data} \parallel p_{model}) = \mathbb{E}_{x \sim p_{data}} [\log p_{data}(x) - \log p_{model}(x, \theta)]$$

since p_{data} does not depend on the parameters θ , in order to minimize this divergence, we only need to minimize:

$$J(\theta) = -\mathbb{E}_{x \sim p_{data}} [\log p_{model}(x; \theta)]$$

which corresponds to the Maximum Likelihood estimation:

$$\theta_{ML} = \operatorname{argmax}_{\theta} \prod_{i=1}^m p_{model}(x^{(i)}; \theta)$$

when we take the log:

$$\theta_{ML} = \operatorname{argmax}_{\theta} \mathbb{E}_{x \sim p_{data}} \log p_{model}(x; \theta)$$

1.8.1. Gradient Descent

Gradient descent is a well-know algorithm in optimization. It is often explained as finding the steepest path to the minimum. It requires to compute the gradient (a vector composed by all partial derivatives of a function) at multiples points. If we have a function $J(\theta)$ to minimize, we start by computing gradients for a point θ and we find a new point θ' closer to the minimum such that:

$$\theta' = \theta - \lambda \nabla_{\theta} J(\theta)$$

We define λ as the learning rate and $\nabla_{\theta} J(\theta)$ as the gradient of J with respect to θ . In deep learning when we need to compute the gradients with respect to a large number of parameters, in order to minimize the loss, it can be very expensive to compute full batch gradient of J involving the sum over all the training points. To avoid memory issues, we use a stochastic version of gradient descent called Stochastic Gradient Descent (SGD). Instead of computing the gradient with respect to the entire training set, we use only a small part of the training set called mini-batches to compute the gradient. One of the reason of the recent deep learning breakthrough was the ability to use those mini-batches over a GPU in order to perform the gradients descent steps efficiently.

Backpropagation[19] is a method used to simplify the computation of the gradients over a network. In order to get the backpropagation, we need to use the chain rule:

$$\frac{\partial_z}{\partial_x} = \frac{\partial_z}{\partial_y} \frac{\partial_y}{\partial_x}$$

This allows us to compute gradients of the cost function with respect to the parameters of the network for a given mini-batch.

1.8.2. Regularization

In order to make the optimization easier, one can use some regularization. For example, it can consist to apply some penalty on the weights (Weight decay). One classical regularization is the L2 regularization, it can be expressed as:

$$J_{L2}(\theta) = J(\theta) + \lambda_2 \sum_k \theta_k^2$$

This regularization led to reduce the norm of the weights by constraint them to be close of 0. Another regularization is the L1, or Laplace regularization, it can be expressed as penalizing the norm L1 of the weights which will lead some of them to be exactly 0:

$$J_{L1}(\theta) = J(\theta) + \lambda_1 \sum_k |\theta_k|$$

1.8.3. Batch normalization

Another trick that can be seen as having a regularization effect is batch normalization[12]. One can see each layers inside a neural network as a probability distribution parametrized by the weights that samples the inputs of the next layer. However during training, since the parameters are updated, the distribution at each layers changes. We call the variation of those distribution the *Internal covariate shift*. The main idea behind batch normalization is to reduce this shift in order to make easier the training. For a given mini-batch \mathcal{B} of examples x^i we compute the mean $\mu_{\mathcal{B}}$ over the mini-batch such that:

$$\mu_{\mathcal{B}} = \frac{1}{N} \sum_{i=0} x^i$$

and the variance σ^2 such that:

$$\sigma_{\mathcal{B}}^2 = \frac{1}{N} \sum_{i=0} (x^i - \mu_{\mathcal{B}})^2$$

Once we have computed those two parameters, we compute the output of a batch normalized layer by applying the following formula on the output x_i of the previous layer:

$$o_{BN}(x_i) = \gamma \frac{(x_i - \mu_{\mathcal{B}})}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \beta$$

The parameters γ and β are learned during training.

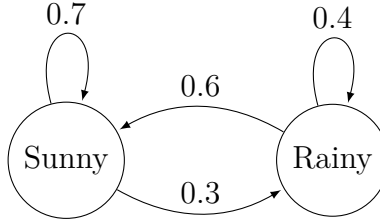


FIG. 1.6. An example of Markov chain

1.9. MARKOV CHAIN

A Markov chain is a sequence of random variables $X^0, X^1, X^2, \dots, X^n$ that have the property of Markov, i.e that any state X_i depends only of its previous state, such that:

$$P(X^t = x) | X^{t-1}, X^{t-2}, \dots, X^0 = P(X^t = x | X^{(t-1)})$$

It can be summarized as the ability to predict the future knowing the present state only. A useful tool to describe a Markov chain is a Stochastic matrix called transition matrix. It describes the probabilities to move to another state provided we are in a given state. A traditional example to explain Markov chains is the example in Figure 1.6. The transition matrix associated with this figure is:

$$P = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}$$

Since the rows describe a probability distribution, they sum to one. If we know that the weather at a state $X^{(0)}$ is sunny, we can express it as $P(X^{(0)} = \text{sunny}) = 1$ and $P(X^{(0)} = \text{rainy}) = 0$. It corresponds to a probability vector $[1, 0]$. Since we have the transition matrix P , we can find the weather for the next day by applying:

$$X^{(1)} = X^{(0)}P \tag{1.9.1}$$

$$= [1, 0] \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \tag{1.9.2}$$

$$= [0.7, 0.3] \tag{1.9.3}$$

Our contributed generative model presented in chapter 3, will be formalized as a Markov chain. However its state is not discrete as here, but continuous: $X^t \in \mathbb{R}^d$ i.e. the state space is a continuous high-dimensional space. In that case the transition from a state to another $P(X^t | X^{t-1})$ will no longer be represented by a simple transition matrix, but as a stochastic transition operator whose parameters θ will be learnt.

1.9.1. Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) are a class of algorithms to sample from a probability distribution by building a Markov chain that has the desired distribution as its equilibrium distribution. MCMC methods need to be run for a sufficient number of steps in order to get samples from the distribution we want to reach. MCMC methods are a vast field of study and are beyond the scope of this thesis. We simply point out here that while the generative model we will present in chapter 3 is defined as a Markov chain, it is one with a fixed number of steps and we will not care about an equilibrium distribution. So despite similarities the approach shouldn't be conceived as a MCMC method, and the transition operator doesn't have to respect the properties typically mandated for MCMC transitions.

Chapter 2

GENERATIVE MODELS

This chapter presents a brief overview of a few approaches that were developed for learning models capable of generating samples that "resemble" those in the training set, in the sense that they follow a distribution that is close to the unknown distribution that yielded the training data. These are called generative models. Developing algorithms to learn generative models is a very active research field. We restricted ourselves here to briefly presenting the approaches that are the more closely related to our novel contribution that will be presented in the next chapter. Many other approaches exist that won't be covered here, in particular Generative Adversarial Networks[9] and its many variants.

2.1. DENOISING AUTOENCODER

Denoising Autoencoder[37] (DAE, Figure 2.1) are autoencoders that receive a corrupted version \tilde{x} of the data x as input. From those noisy data \tilde{x} , the DAEs are trained to reconstruct the true data x . In order to train a DAE, we have to define a corruption process $C(\tilde{x} | x)$. In practical case, we often choose C as an additive Gaussian or uniform noise.

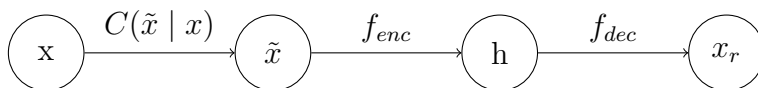


FIG. 2.1. A denoising autoencoder with intermediate representation h

If we see the autoencoder as a stochastic process where we are trying to maximise the probability $p_{dec}(x | h)$, where p_{dec} is a simple distribution whose parameters are given by $f_{dec}(h)$ e.g $p_{dec}(x | h) = \mathcal{N}(f_{dec}(h), \sigma^2 I)$, the criterion used for training a DAE is to maximize the expectation:

$$\mathbb{E}_{x \sim P(X)} \mathbb{E}_{\tilde{x} \sim C(\tilde{x}|x)} \log p_{dec}(x | f_{enc}(\tilde{x}))$$

One way to understand the DAE is to visualize the data as following a low dimensional manifold. The corrupted data will be likely to be further away from the data manifold. The DAE will try to learn a stochastic operator $P(X | \tilde{X})$ such that it will map the low

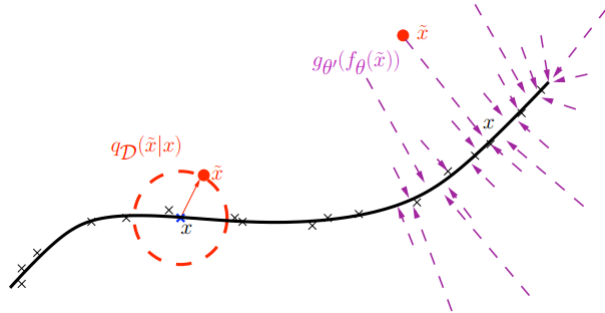


FIG. 2.2. Manifold learning with a DAE. The model learns to project the corrupted data to the manifold of the true data. (Figure from [36])

probability region of the corrupted sample to the high probability region of the true data. This operator learns to do small or big jump toward the manifold we want to reach. This is illustrated in Figure 2.2.

The algorithm of the DAE can be summarized as follow:

- Sample a data point x from the training set, $x \sim P(X)$
- Corrupt x with the corruption process C to get $\tilde{x} \sim C(\tilde{x} | x)$
- Use \tilde{x} as input to the encoder f_{enc}
- Use f_{dec} , to get $x_r = f_{dec}(f_{enc}(\tilde{x}))$
- Train the encoder and the decoder to match x_r and x

While the DAE was not initially developed to perform as a generative model, but to learn a good representation, there have been several proposals to use a DAE as a generative model. Starting with the "non-parametric" sampling procedure already proposed in [37], following up with [3]. These procedures have been generalized in the work on Generative Stochastic Networks [1] that we will present shortly. Another approach to learn a generative model, that also uses a stochastic autoencoder and is closely related to DAEs is the Variational Autoencoder (VAE)[13].

2.2. VARIATIONAL AUTOENCODER

The variational autoencoder (Figure 2.3) was introduced in [13]. It is a technique used to train a generative model $p(x)$ whose generative process can be defined as follows:

- A latent vector variable z is sampled following a simple distribution $p(z)$ (often a unit Gaussian): $z \sim p(z)$
- Observed variable x is then obtained by sampling from $p_\phi(x | z)$, which is also typically a simple distribution, such as a diagonal Gaussian, but whose parameters are obtained through a rich nonlinear function $f_\phi(z)$, e.g. $p_\phi(x | z) = \mathcal{N}(f_\phi(z), \sigma^2 I)$.

Training such a model by maximizing the exact likelihood is usually intractable, as computing $p(x)$ requires marginalizing over latent variable z , which in all but the simplest cases is

intractable. So instead, in VAEs, one maximizes a variational lower-bound on the log-likelihood. This involves using, in place of the true posterior $p(z | x)$ that we cannot compute efficiently, an approximate posterior $q_\theta(z | x)$. Under this view, the encoder can be considered as the parametrized proposal distribution $q_\theta(z | x)$ and the decoder as another parametrized distribution $p_\phi(x | z)$.

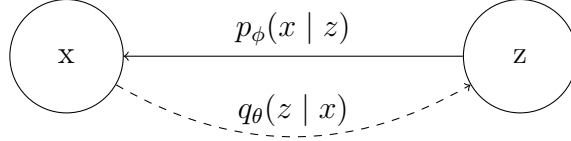


FIG. 2.3. A Variational Autoencoder

2.2.1. Variational inference

Given a dataset $X = x^1, x^2, \dots, x^n$ and x an observable variable, we want to maximize the log likelihood:

$$\log p(X) = \sum_{i=1}^N \log p(x^i)$$

However it requires to compute an integral over $p(x) = \int_z p(z)p(x | z)dz$ that is intractable. We can use the conditional probability rule to write:

$$\log p(x) = \mathbb{E}_{z \sim p(z|x)} \left[\log \frac{p(z, x)}{p(z | x)} \right]$$

Computing $p(z | x)$ is still intractable, so we use an approximate distribution called $q(z | x)$ to approximate $p(z | x)$. We want to train our network such that q becomes close of p , i.e. we want to minimize the divergence between those two distributions. We write the KL as follow:

$$D_{KL}(q(z | x) \| p(z | x)) = \int q(z | x) \log \frac{q(z | x)}{p(z | x)} dz \quad (2.2.1)$$

$$= \int q(z | x) \log \frac{q(z | x)p(x)}{p(z, x)} dz \quad (2.2.2)$$

$$= \mathbb{E}_{q(z|x)} [\log q(z | x) - \log p(z, x)] + \log p(x) \quad (2.2.3)$$

We can now express $\log p(x)$ as:

$$\log p(x) = D_{KL}(q(z | x) \| p(z | x)) - \mathbb{E}_{q(z|x)} [\log q(z | x) - \log p(z, x)]$$

Since the KL-divergence is positive, we can deduce that the second term is a lower bound on $\log p(x)$:

$$\log p(x) \geq -\mathbb{E}_{q(z|x)} [\log q(z | x) - \log p(z, x)] \quad (2.2.4)$$

$$\geq \mathbb{E}_{q(z|x)} [\log p(z, x) - \log q(z | x)] \quad (2.2.5)$$

To optimize this, in case of $q(z | x)$ is continue, we can use the reparametrization trick[13] i.e allowing to have a gradient that can backpropagate through the sampling steps.

We will thus choose as our optimization objective, to maximize the following lower-bound on the log-likelihood:

$$LBO = \mathbb{E}_{q(z|x)}[\log p(z, x) - \log q(z | x)] \quad (2.2.6)$$

$$= \mathbb{E}_{q(z|x)}[\log(p(x|z)p(z)) - \log q(z | x)] \quad (2.2.7)$$

$$= \mathbb{E}_{q(z|x)}[\log p(x|z) + \log p(z) - \log q(z | x)] \quad (2.2.8)$$

$$= \mathbb{E}_{q(z|x)}[\log p(x|z)] - \mathbb{E}_{q(z|x)}[\log q(z | x) - \log p(z)] \quad (2.2.9)$$

$$= \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z | x) || p(z)) \quad (2.2.10)$$

In VAE we consider p and q as part of a parametrized stochastic autoencoder, we get:

$$LBO = \mathbb{E}_{q(z|x)}[\log p_\phi(x|z)] - D_{KL}(q_\theta(z | x) || p(z))$$

Recall from above, minimizing this lower bound also corresponds to minimizing the KL divergence between $q(z | x)$ and $p(z | x)$. We can see the first term $\mathbb{E}_{q(z|x)}[\log p_\phi(x|z)]$ as trying to maximize the probability to get x with respect to the latent variable z . It's equivalent to a reconstruction loss of a stochastic auto-encoder. The second term can be seen as a regularization term where we want that our approximate posterior $q_\theta(z | x)$ stays close to the prior $p(z)$. In most of the practical cases, $p(z)$ is chosen to be Gaussian. The choice of the prior defines the latent space that will encode the features extracted from the data.

2.3. GENERATIVE STOCHASTICS NETWORKS

Generative stochastic networks[37] (GSN) (Figure 2.4) are a generalization of Denoising autoencoder to allow generating samples. It is based on the repeated application of a denoising-autoencoder-like stochastic transition operator. This yields a sequence of latent state that defines a Markov chain. It's based on learning the transition operator between each state of this chain. One inspiration for the GSN is that it might be easier to learn the conditional probability $P(X | \tilde{X})$ with $\tilde{x} = c(x)$ than the probability of the true data $P(X)$. One issue with using trained DAEs as generative models is that the number of region visited during training by a traditional DAE are limited to a close neighborhood of the true data. In order to increase the number of regions visited, i.e having a proposal distribution that is closer of the original $P(X)$, GSNs introduce a Markov chain whose stationary distribution π will have $P(X)$ as marginal density by alternatively adding noise to a sample and learning to denoise it in order to approximate the true $P(X | \tilde{X})$.

The procedure can be described as follows:

- $x^{(0)} \sim P(X), \dots, \tilde{x}^{(0)} \sim C(\tilde{x}^{(0)} | x^{(0)})$

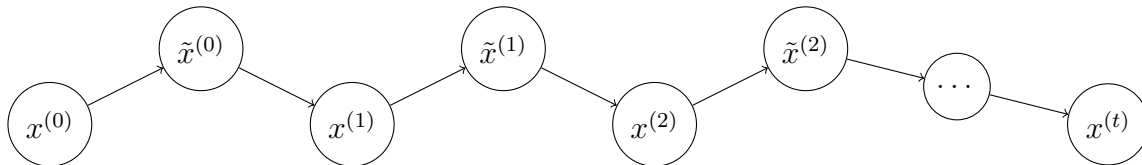


FIG. 2.4. A Generative stochastic networks

- $x^{(1)} \sim P_{\theta}(X^{(1)} | \tilde{x}^{(0)}), \dots, \tilde{x}^{(1)} \sim C(\tilde{x}^{(1)} | x^{(1)})$
- \dots
- $x^{(k)} \sim P_{\theta}(X^{(k)} | \tilde{x}^{(k-1)}), \dots, \tilde{x}^{(k)} \sim C(\tilde{x}^{(k)} | x^{(k-1)})$

In each step the neural network is trained to maximise the log probability $\log P(X | \tilde{X})$. Thus we maximize the loss (Called walk-back):

$$\mathbb{L} \approx \frac{1}{N} \sum_{i=1}^N \log P_{\theta}(X^i | \tilde{X}^{(i)})$$

Instead of using a fixed corruption function C , one can use a parametrized function that can be learned. The Markov chain becomes formulated as:

$$\begin{aligned} H^{(t)} &\sim P_{\theta_1}(H^{(t)} | H^{(t-1)}, X^{(t-1)}) \\ X^{(t)} &\sim P_{\theta_2}(X^{(t)} | H^{(t-1)}) \end{aligned}$$

2.4. DIFFUSION APPROACH

Deep Unsupervised Learning using Nonequilibrium Thermodynamics[33] introduces another way to train a generative model. Inspired by physics, they used an iterative diffusion process that slightly destroy the structure of the true data distribution. The main idea is to learn with a neural network 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 such that:

$$q(x^{(t)} | x^{(t-1)}) = \mathbb{T}_{\pi}(x^{(t)} | x^{t-1}; \beta_t)$$

where $x^{(0)}$ is an example taken from the training set and β_t a diffusion rate. To get a reverse process $p(x^{(t-1)} | x^{(t)})$ that has the same distribution form as the forward diffusion process $q(x^{(t)} | x^{(t-1)})$ (e.g a Gaussian) the diffusion needs to be very small between each steps. In the paper, they use chains with one thousand tiny steps. In this approach only the reverse process is learned, the diffusion process is fixed and doesn't change along the training.

2.5. VARIATIONAL WALKBACK

Another interesting approach is the Variational Walkback[10] (VW) that learns a transition operator of a Markov chain defined as a neural network. It is close to the diffusion

approach in the sense that there is a destruction process called heating process that moves gradually away from the data manifold towards a prior distribution. However in the approach of [33] the diffusion process is fixed while the diffusion process of Variational Walkback is done by sampling directly from the transition operator parametrized by a higher temperature. Then there is a *cooling* process that *walkbacks* gradually from the prior distribution towards the data distribution by sampling from the transition operator parametrized by a low temperature. Similarly to the denoising autoencoder, the network is trained to maximize the reconstruction at each step with respect to the previous step which corresponds to maximizing a variational lower bound of the log likelihood.

In parallel of the work on Variational Walkback, Pascal Vincent, Sina Honari and I have developed a similar approach, that is presented in the next chapter, in order to learn a transition operator. Starting from unstructured noise, we learn a stochastic transition operator that will progressively move toward the data manifold. One major difference with the Diffusion and VW approach is that we don't use or learn a destruction process. We only learn the reverse process that, when informed by the training set example, we call Infusion to contrast with the diffusion process of [33]. So we never posit that $p(x^{(t-1)} | x^{(t)})$ and $p(x^{(t)} | x^{(t-1)})$ have the same distribution form. In order to train this network we learn at each step to reconstruct a less noisy version closer to the target. Another difference with the Diffusion and VW approaches is that we didn't use a lower-bound of the log-likelihood as training criterion but a denoising criterion. This simple denoising criterion, while a heuristic, is local, requiring no backprop-through-time or reparametrization trick to flow gradients through the steps of the chain.

Chapter 3

LEARNING TO GENERATE SAMPLES FROM NOISE THROUGH INFUSION TRAINING

This chapter presents a joint work with Sina Honari and Pascal Vincent. It was published at the International Conference on Learning Representations 2017 (Conference Track).

Contribution My supervisor Pascal Vincent was the leader of the project. He got the original idea of Infusion Training. I proposed the idea to have an increasing schedule for the infusion rate and to use an adaptive batch normalization. I wrote the code and run all the experiments presented in the paper, generated all figures and tables. Pascal wrote the beginning of the paper, I wrote the experimental and appendix section. Sina Honari has provided a review of the paper and helped me to write the appendix. He launched experiments on keypoints detection but they are not included in this paper.

3.1. ABSTRACT

In this work, we investigate a novel training procedure to learn a generative model as the transition operator of a Markov chain, such that, when applied repeatedly on an unstructured random noise sample, it will denoise it into a sample that matches the target distribution from the training set. The novel training procedure to learn this progressive denoising operation involves sampling from a slightly different chain than the model chain used for generation in the absence of a denoising target. In the training chain we infuse information from the training target example that we would like the chains to reach with a high probability. The thus learned transition operator is able to produce quality and varied samples in a small number of steps. Experiments show competitive results compared to the samples generated with a basic Generative Adversarial Net.

3.2. INTRODUCTION AND MOTIVATION

To go beyond the relatively simpler tasks of classification and regression, advancing our ability to learn good generative models of high-dimensional data appears essential. There

are many scenarios where one needs to efficiently produce good high-dimensional outputs where output dimensions have unknown intricate statistical dependencies: from generating realistic images, segmentations, text, speech, keypoint or joint positions, etc..., possibly as an answer to the same, other, or multiple input modalities. These are typically cases where there is not just one right answer but a variety of equally valid ones following a non-trivial and unknown distribution. A fundamental ingredient for such scenarios is thus the ability to learn a good generative model from data, one from which we can subsequently efficiently generate varied samples of high quality.

Many approaches for learning to generate high dimensional samples have been and are still actively being investigated. These approaches can be roughly classified under the following broad categories:

- Ordered visible dimension sampling [35, 15]. In this type of auto-regressive approach, output dimensions (or groups of conditionally independent dimensions) are given an arbitrary fixed ordering, and each is sampled conditionally on the previous sampled ones. This strategy is often implemented using a recurrent network (LSTM or GRU). Desirable properties of this type of strategy are that the exact log likelihood can usually be computed tractably, and sampling is exact. Undesirable properties follow from the forced ordering, whose arbitrariness feels unsatisfactory especially for domains that do not have a natural ordering (e.g. images), and imposes for high-dimensional output a *long sequential* generation that can be slow.
- Undirected graphical models with *multiple* layers of *latent* variables. These make inference, and thus learning, particularly hard and tend to be costly to sample from [29].
- Directed graphical models trained as variational autoencoders (VAE) [13, 26]
- Adversarially-trained generative networks. (GAN)[9]
- Stochastic neural networks, i.e. networks with stochastic neurons, trained by an adapted form of stochastic backpropagation
- Generative uses of denoising autoencoders [37] and their generalization as Generative Stochastic Networks [1]
- Inverting a non-equilibrium thermodynamic slow *diffusion* process [33]
- Continuous transformation of a distribution by invertible functions (Dinh, Krueger, and Bengio [7], also used for variational inference in Rezende and Mohamed [25])

Several of these approaches are based on maximizing an explicit or implicit model log-likelihood or a lower bound of its log-likelihood, but some successful ones are not e.g. GANs. The approach we propose here is based on the notion of “denoising” and thus takes its root in denoising autoencoders and the GSN type of approaches. It is also highly related to the non-equilibrium thermodynamics inverse diffusion approach of Sohl-Dickstein et al. [33]. One key aspect that distinguishes these types of methods from others listed above is that

sample generation is achieved thanks to a learned stochastic mapping from input space to input space, rather than from a latent-space to input-space.

Specifically, in the present work, we propose to learn to generate high quality samples through a process of *progressive, stochastic, denoising*, starting from a simple initial “noise” sample generated in input space from a simple factorial distribution i.e. one that does not take into account any dependency or structure between dimensions. This, in effect, amounts to learning the transition operator of a Markov chain operating on input space. Starting from such an initial “noise” input, and repeatedly applying the operator for a small fixed number T of steps, we aim to obtain a high quality resulting sample, effectively modeling the training data distribution. Our training procedure uses a novel “target-infusion” technique, designed to slightly bias model sampling to move towards a specific data point during training, and thus provide inputs to denoise which are likely under the model’s sample generation paths. By contrast with Sohl-Dickstein et al. [33] which consists in inverting a slow and fixed diffusion process, our infusion chains make a few large jumps and follow the model distribution as the learning progresses.

The rest of this paper is structured as follows: Section 2 formally defines the model and training procedure. Section 3 discusses and contrasts our approach with the most related methods from the literature. Section 4 presents experiments that validate the approach. Section 5 concludes and proposes future work directions.

3.3. PROPOSED APPROACH

3.3.1. Setup

We are given a finite data set D containing n points in \mathbb{R}^d , supposed drawn i.i.d from an unknown distribution q^* . The data set D is supposed split into training, validation and test subsets D_{train} , D_{valid} , D_{test} . We will denote q_{train}^* the *empirical distribution* associated to the training set, and use \mathbf{x} to denote observed samples from the data set. We are interested in learning the parameters of a generative model p conceived as a Markov Chain from which we can efficiently sample. Note that we are interested in learning an operator that will display fast “*burn-in*” from the initial factorial “noise” distribution, but beyond the initial T steps we are not concerned about potential slow mixing or being stuck. We will first describe the sampling procedure used to sample from a trained model, before explaining our training procedure.

3.3.2. Generative model sampling procedure

The generative model p is *defined* as the following sampling procedure:

- Using a simple factorial distribution $p^{(0)}(\mathbf{z}^{(0)})$, draw an initial sample $\mathbf{z}^{(0)} \sim p^{(0)}$, where $\mathbf{z}^{(0)} \in \mathbb{R}^d$. Since $p^{(0)}$ is factorial, the d components of $\mathbf{z}^{(0)}$ are independent: p^0

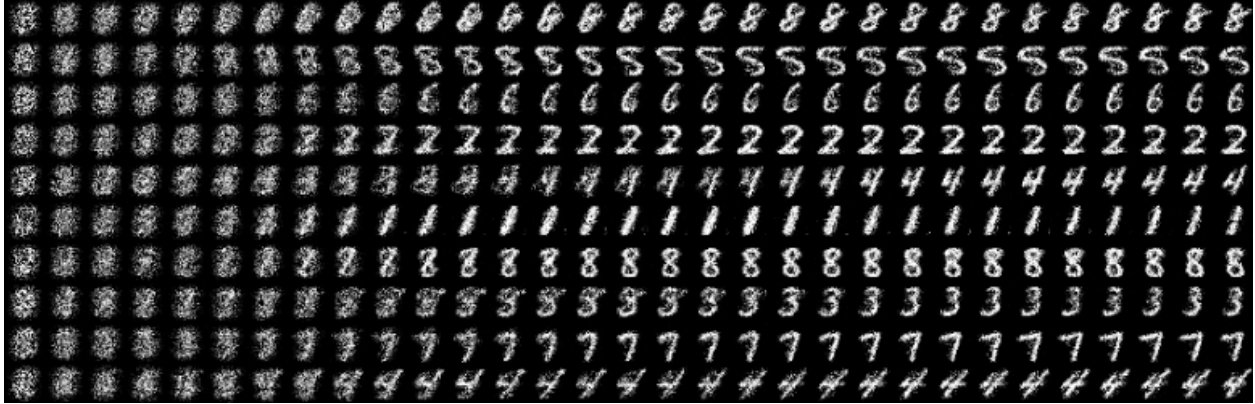


FIG. 3.1. The *model sampling chain*. Each row shows a sample from $p(\mathbf{z}^{(0)}, \dots, \mathbf{z}^{(T)})$ for a model that has been trained on MNIST digits. We see how the learned Markov transition operator progressively denoises an initial unstructured noise sample. We can also see that there remains ambiguity in the early steps as to what digit this could become. This ambiguity gets resolved only in later steps. Even after a few initial steps, stochasticity could have made a chain move to a different final digit shape.

cannot model any dependency structure. $\mathbf{z}^{(0)}$ can be pictured as essentially unstructured random noise.

- Repeatedly apply T times a stochastic transition operator $p^{(t)}(\mathbf{z}^{(t)}|\mathbf{z}^{(t-1)})$, yielding a more “denoised” sample $\mathbf{z}^{(t)} \sim p^{(t)}(\mathbf{z}^{(t)}|\mathbf{z}^{(t-1)})$, where all $\mathbf{z}^{(t)} \in \mathbb{R}^d$.
- Output $\mathbf{z}^{(T)}$ as the final generated sample. Our generative model distribution is thus $p(\mathbf{z}^{(T)})$, the marginal associated to joint $p(\mathbf{z}^{(0)}, \dots, \mathbf{z}^{(T)}) = p^{(0)}(\mathbf{z}^{(0)}) \left(\prod_{t=1}^T p^{(t)}(\mathbf{z}^{(t)}|\mathbf{z}^{(t-1)}) \right)$.

In summary, samples from model p are generated, starting with an initial sample from a simple distribution $p^{(0)}$, by taking the T^{th} sample along Markov chain $\mathbf{z}^{(0)} \rightarrow \mathbf{z}^{(1)} \rightarrow \mathbf{z}^{(2)} \rightarrow \dots \rightarrow \mathbf{z}^{(T)}$ whose transition operator is $p^{(t)}(\mathbf{z}^{(t)}|\mathbf{z}^{(t-1)})$. We will call this chain the *model sampling chain*. Figure 3.1 illustrates this sampling procedure using a model (i.e. transition operator) that was trained on MNIST. Note that we impose no formal requirement that the chain converges to a stationary distribution, as we simply read-out $\mathbf{z}^{(T)}$ as the samples from our model p . The chain also needs not be time-homogeneous, as highlighted by notation $p^{(t)}$ for the transitions.

The set of parameters θ of model p comprise the parameters of $p^{(0)}$ and the parameters of transition operator $p^{(t)}(\mathbf{z}^{(t)}|\mathbf{z}^{(t-1)})$. For tractability, learnability, and efficient sampling, these distributions will be chosen factorial, i.e. $p^{(0)}(\mathbf{z}^{(0)}) = \prod_{i=1}^d p_i^{(0)}(\mathbf{z}_i^{(0)})$ and $p^{(t)}(\mathbf{z}^{(t)}|\mathbf{z}^{(t-1)}) = \prod_{i=1}^d p_i^{(t)}(\mathbf{z}_i^{(t)}|\mathbf{z}_i^{(t-1)})$. Note that the conditional distribution of an individual component i , $p_i^{(t)}(\mathbf{z}_i^{(t)}|\mathbf{z}_i^{(t-1)})$ may however be multimodal, e.g. a mixture in which case $p^{(t)}(\mathbf{z}^{(t)}|\mathbf{z}^{(t-1)})$ would be a product of independent mixtures (conditioned on $\mathbf{z}^{(t-1)}$), one per dimension. In our experiments, we will take the $p^{(t)}(\mathbf{z}^{(t)}|\mathbf{z}^{(t-1)})$ to be simple diagonal Gaussian yielding a Deep Latent Gaussian Model (DLGM) as in Rezende et al. [26].

3.3.3. Infusion training procedure

We want to train the parameters of model p such that samples from D_{train} are likely of being generated under the *model sampling chain*. Let $\theta^{(0)}$ be the parameters of $p^{(0)}$ and let $\theta^{(t)}$ be the parameters of $p^{(t)}(\mathbf{z}^{(t)}|\mathbf{z}^{(t-1)})$. Note that parameters $\theta^{(t)}$ for $t > 0$ can straightforwardly be shared across time steps, which we will be doing in practice. Having committed to using (conditionally) *factorial* distributions for our $p^{(0)}(\mathbf{z}^{(0)})$ and $p^{(t)}(\mathbf{z}^{(t)}|\mathbf{z}^{(t-1)})$, that are both easy to learn and cheap to sample from, let us first consider the following greedy stagewise procedure. We can easily learn $p_i^{(0)}(\mathbf{z}^{(0)})$ to model the marginal distribution of each component \mathbf{x}_i of the input, by training it by gradient descent on a maximum likelihood objective, i.e.

$$\theta^{(0)} = \arg \max_{\theta} \mathbb{E}_{\mathbf{x} \sim q_{\text{train}}^*} [\log p^{(0)}(\mathbf{x}; \theta)] \quad (3.3.1)$$

This gives us a first, very crude unstructured (factorial) model of q^* .

Having learned this $p^{(0)}$, we might be tempted to then greedily learn the next stage $p^{(1)}$ of the chain in a similar fashion, after drawing samples $\mathbf{z}^{(0)} \sim p^{(0)}$ in an attempt to learn to “denoise” the sampled $\mathbf{z}^{(0)}$ into \mathbf{x} . Yet the corresponding following training objective $\theta^{(1)} = \arg \max_{\theta} \mathbb{E}_{\mathbf{x} \sim q_{\text{train}}^*, \mathbf{z}^{(0)} \sim p^{(0)}} [\log p^{(1)}(\mathbf{x}|\mathbf{z}^{(0)}; \theta)]$ makes no sense: \mathbf{x} and $\mathbf{z}^{(0)}$ are sampled independently of each other so $\mathbf{z}^{(0)}$ contains no information about \mathbf{x} , hence $p^{(1)}(\mathbf{x}|\mathbf{z}^{(0)}) = p^{(1)}(\mathbf{x})$. So maximizing this second objective becomes essentially the same as what we did when learning $p^{(0)}$. We would learn nothing more. It is essential, if we hope to learn a useful conditional distribution $p^{(1)}(\mathbf{x}|\mathbf{z}^{(0)})$ that it be trained on *particular* $\mathbf{z}^{(0)}$ containing some information about \mathbf{x} . In other words, we should not take our training inputs to be samples from $p^{(0)}$ but from a slightly different distribution, biased towards containing some information about \mathbf{x} . Let us call it $q^{(0)}(\mathbf{z}^{(0)}|\mathbf{x})$. A natural choice for it, if it were possible, would be to take $q^{(0)}(\mathbf{z}^{(0)}|\mathbf{x}) = p(\mathbf{z}^{(0)}|\mathbf{z}^{(T)} = \mathbf{x})$ but this is an intractable inference, as all intermediate $\mathbf{z}^{(t)}$ between $\mathbf{z}^{(0)}$ and $\mathbf{z}^{(T)}$ are effectively latent states that we would need to marginalize over. Using a workaround such as a variational or MCMC approach would be a usual fallback. Instead, let us focus on our initial intent of guiding a progressive stochastic denoising, and think if we can come up with a different way to construct $q^{(0)}(\mathbf{z}^{(0)}|\mathbf{x})$ and similarly for the next steps $q_i^{(t)}(\tilde{\mathbf{z}}_i^{(t)}|\tilde{\mathbf{z}}^{(t-1)}, \mathbf{x})$.

Eventually, we expect a sequence of samples from Markov chain p to move from initial “noise” towards a specific example \mathbf{x} from the training set rather than another one, primarily if a sample along the chain “resembles” \mathbf{x} to some degree. This means that the transition operator should learn to pick up a minor resemblance with an \mathbf{x} in order to transition to something likely to be even more similar to \mathbf{x} . In other words, we expect samples along a chain leading to \mathbf{x} to both have high probability under the transition operator of the chain $p^{(t)}(\mathbf{z}^{(t)}|\mathbf{z}^{(t-1)})$, *and* to have some form of at least partial “resemblance” with \mathbf{x} likely to increase as we progress along the chain. One highly inefficient way to emulate such a chain

of samples would be, for each step t , to sample many candidate samples from the transition operator (a conditionally factorial distribution) until we generate one that has some minimal “resemblance” to \mathbf{x} (e.g. for a discrete space, this resemblance measure could be based on their Hamming distance). A qualitatively similar result can be obtained at a negligible cost by sampling from a factorial distribution that is very close to the one given by the transition operator, but very slightly biased towards producing something closer to \mathbf{x} . Specifically, we can “infuse” a little of \mathbf{x} into our sample by choosing for each input dimension, whether we sample it from the distribution given for that dimension by the transition operator, or whether, with a small probability, we take the value of that dimension from \mathbf{x} . Samples from this biased chain, in which we slightly “infuse” \mathbf{x} , will provide us with the inputs of our input-target training pairs for the transition operator. The target part of the training pairs is simply \mathbf{x} .

3.3.3.1. The infusion chain

Formally we define an *infusion chain* $\tilde{\mathbf{z}}^{(0)} \rightarrow \tilde{\mathbf{z}}^{(1)} \rightarrow \dots \rightarrow \tilde{\mathbf{z}}^{(T-1)}$ whose distribution $q(\tilde{\mathbf{z}}^{(0)}, \dots, \tilde{\mathbf{z}}^{(T-1)} | \mathbf{x})$ will be “close” to the *sampling chain* $\mathbf{z}^{(0)} \rightarrow \mathbf{z}^{(1)} \rightarrow \mathbf{z}^{(2)} \rightarrow \dots \rightarrow \mathbf{z}^{(T-1)}$ of model p in the sense that $q^{(t)}(\tilde{\mathbf{z}}^{(t)} | \tilde{\mathbf{z}}^{(t-1)}, \mathbf{x})$ will be close to $p^{(t)}(\mathbf{z}^{(t)} | \mathbf{z}^{(t-1)})$, but will at every step be slightly biased towards generating samples closer to target \mathbf{x} , i.e. \mathbf{x} gets progressively “infused” into the chain. This is achieved by defining $q_i^{(0)}(\tilde{\mathbf{z}}_i^{(0)} | \mathbf{x})$ as a mixture between $p_i^{(0)}$ (with a large mixture weight) and $\delta_{\mathbf{x}_i}$, a concentrated unimodal distribution around \mathbf{x}_i , such as a Gaussian with small variance (with a small mixture weight)¹. Formally $q_i^{(0)}(\tilde{\mathbf{z}}_i^{(0)} | \mathbf{x}) = (1 - \alpha^{(0)})p_i^{(0)}(\tilde{\mathbf{z}}_i^{(0)}) + \alpha^{(0)}\delta_{\mathbf{x}_i}(\tilde{\mathbf{z}}_i^{(0)})$, where $1 - \alpha^{(0)}$ and $\alpha^{(0)}$ are the mixture weights². In other words, when sampling a value for $\tilde{\mathbf{z}}_i^{(0)}$ from $q_i^{(0)}$ there will be a small probability $\alpha^{(0)}$ to pick value close to \mathbf{x}_i (as sampled from $\delta_{\mathbf{x}_i}$) rather than sampling the value from $p_i^{(0)}$. We call $\alpha^{(t)}$ the *infusion rate*. We define the transition operator of the *infusion chain* similarly as: $q_i^{(t)}(\tilde{\mathbf{z}}_i^{(t)} | \tilde{\mathbf{z}}^{(t-1)}, \mathbf{x}) = (1 - \alpha^{(t)})p_i^{(t)}(\tilde{\mathbf{z}}_i^{(t)} | \tilde{\mathbf{z}}^{(t-1)}) + \alpha^{(t)}\delta_{\mathbf{x}_i}(\tilde{\mathbf{z}}_i^{(t)})$.

3.3.3.2. Denoising-based Infusion training procedure

For all $\mathbf{x} \in D_{\text{train}}$:

- Sample from the *infusion chain* $\tilde{\mathbf{z}} = (\tilde{\mathbf{z}}^{(0)}, \dots, \tilde{\mathbf{z}}^{(T-1)}) \sim q(\tilde{\mathbf{z}}^{(0)}, \dots, \tilde{\mathbf{z}}^{(T-1)} | \mathbf{x})$.
precisely so: $\tilde{\mathbf{z}}_0 \sim q^{(0)}(\tilde{\mathbf{z}}^{(0)} | \mathbf{x}) \dots \tilde{\mathbf{z}}^{(t)} \sim q^{(t)}(\tilde{\mathbf{z}}^{(t)} | \tilde{\mathbf{z}}^{(t-1)}, \mathbf{x}) \dots$

¹Note that $\delta_{\mathbf{x}_i}$ does not denote a Dirac-Delta but a Gaussian with small sigma.

²In all experiments, we use an increasing schedule $\alpha^{(t)} = \alpha^{(t-1)} + \omega$ with $\alpha^{(0)}$ and ω constant. This allows to build our chain such that in the first steps, we give little information about the target and in the last steps we give more informations about the target. This forces the network to have less confidence (greater incertitude) at the beginning of the chain and more confidence on the convergence point at the end of the chain.

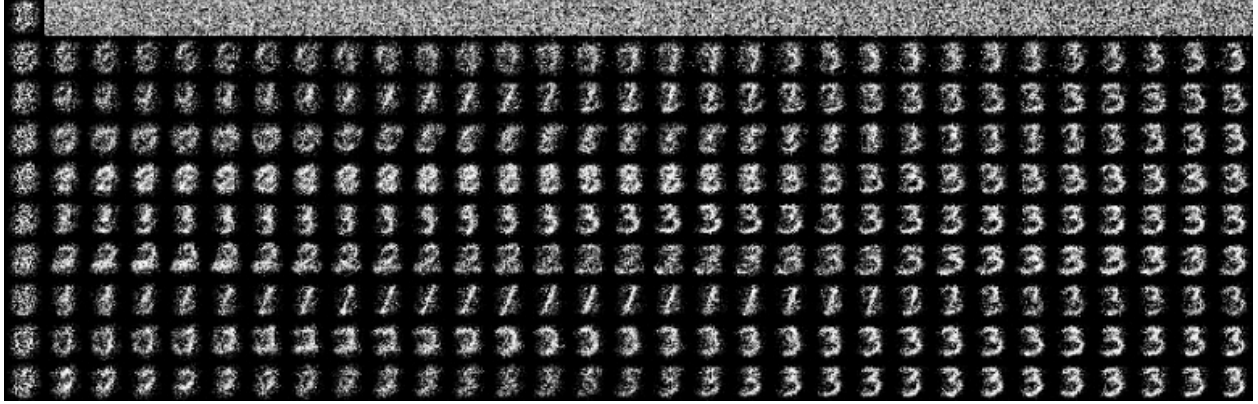


FIG. 3.2. Training *infusion chains*, infused with target $\mathbf{x} = \mathbf{3}$. This figure shows the evolution of chain $q(\mathbf{z}^{(0)}, \dots, \mathbf{z}^{(30)} | \mathbf{x})$ as training on MNIST progresses. Top row is after network random weight initialization. Second row is after 1 training epochs, third after 2 training epochs, and so on. Each of these images were at a time provided as the input part of the *(input, target)* training pairs for the network. The network was trained to denoise all of them into target 3. We see that as training progresses, the model has learned to pick up the cues provided by target infusion, to move towards that target. Note also that a single denoising step, even with target infusion, is not sufficient for the network to produce a sharp well identified digit.

- Perform a gradient step so that p learns to “denoise” every $\tilde{\mathbf{z}}^{(t)}$ into \mathbf{x} .

$$\theta^{(t)} \leftarrow \theta^{(t)} + \eta^{(t)} \frac{\partial \log p^{(t)}(\mathbf{x} | \tilde{\mathbf{z}}^{(t-1)}; \theta^{(t)})}{\partial \theta^{(t)}}$$

where $\eta^{(t)}$ is a scalar learning rate.³

As illustrated in Figure 3.2, the distribution of samples from the infusion chain evolves as training progresses, since this chain remains close to the model sampling chain.

3.3.4. Stochastic log likelihood estimation

The exact log-likelihood of the generative model implied by our model p is intractable. The log-probability of an example \mathbf{x} can however be expressed using proposal distribution q as:

$$\log p(\mathbf{x}) = \log \mathbb{E}_{q(\tilde{\mathbf{z}} | \mathbf{x})} \left[\frac{p(\tilde{\mathbf{z}}, \mathbf{x})}{q(\tilde{\mathbf{z}} | \mathbf{x})} \right] \quad (3.3.2)$$

Using Jensen’s inequality we can thus derive the following lower bound:

³Since we will be sharing parameters between the $p^{(t)}$, in order for the expected larger error gradients on the earlier transitions not to dominate the parameter updates over the later transitions we used an increasing schedule $\eta^{(t)} = \eta_0 \frac{t}{T}$ for $t \in \{1, \dots, T\}$.

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\tilde{\mathbf{z}}|\mathbf{x})} [\log p(\tilde{\mathbf{z}}, \mathbf{x}) - \log q(\tilde{\mathbf{z}}|\mathbf{x})] \quad (3.3.3)$$

where $\log p(\tilde{\mathbf{z}}, \mathbf{x}) = \log p^{(0)}(\tilde{\mathbf{z}}^{(0)}) + \left(\sum_{t=1}^{T-1} \log p^{(t)}(\tilde{\mathbf{z}}^{(t)}|\tilde{\mathbf{z}}^{(t-1)})\right) + \log p^{(T)}(\mathbf{x}|\tilde{\mathbf{z}}^{(T-1)})$ and $\log q(\tilde{\mathbf{z}}|\mathbf{x}) = \log q^{(0)}(\tilde{\mathbf{z}}^{(0)}|\mathbf{x}) + \sum_{t=1}^{T-1} \log q^{(t)}(\tilde{\mathbf{z}}^{(t)}|\tilde{\mathbf{z}}^{(t-1)}, \mathbf{x})$.

A stochastic estimation can easily be obtained by replacing the expectation by an average using a few samples from $q(\tilde{\mathbf{z}}|\mathbf{x})$. We can thus compute a lower bound estimate of the average log likelihood over training, validation and test data.

Similarly in addition to the lower-bound based on Eq.3.3.3 we can use the same few samples from $q(\tilde{\mathbf{z}}|\mathbf{x})$ to get an importance-sampling estimate of the likelihood based on Eq. 3.3.2⁴.

3.3.4.1. Lower-bound-based infusion training procedure

Since we have derived a lower bound on the likelihood, we can alternatively choose to optimize this stochastic lower-bound directly during training. This alternative lower-bound based infusion training procedure differs only slightly from the denoising-based infusion training procedure by using $\tilde{\mathbf{z}}^{(t)}$ as a training target at step t (performing a gradient step to increase $\log p^{(t)}(\tilde{\mathbf{z}}^{(t)}|\tilde{\mathbf{z}}^{(t-1)}; \theta^{(t)})$) whereas denoising training always uses \mathbf{x} as its target (performing a gradient step to increase $\log p^{(t)}(\mathbf{x}|\tilde{\mathbf{z}}^{(t-1)}; \theta^{(t)})$). Note that the same *reparametrization trick* as used in Variational Auto-encoders [13] can be used here to backpropagate through the chain’s Gaussian sampling.

3.4. RELATIONSHIP TO PREVIOUSLY PROPOSED APPROACHES

3.4.1. Markov Chain Monte Carlo for energy-based models

Generating samples as a repeated application of a Markov transition operator that operates on input space is at the heart of Markov Chain Monte Carlo (MCMC) methods. They allow sampling from an energy-model, where one can efficiently compute the energy or un-normalized negated log probability (or density) at any point. The transition operator is then *derived from an explicit energy function* such that the Markov chain prescribed by a specific MCMC method is guaranteed to converge to the distribution defined by that energy function, as the equilibrium distribution of the chain. MCMC techniques have thus been used to obtain samples from the energy model, in the process of learning to adjust its parameters.

⁴Specifically, the two estimates (lower-bound and IS) start by collecting k samples from $q(\tilde{\mathbf{z}}|\mathbf{x})$ and computing for each the corresponding $\ell = \log p(\tilde{\mathbf{z}}, \mathbf{x}) - \log q(\tilde{\mathbf{z}}|\mathbf{x})$. The lower-bound estimate is then obtained by averaging the resulting ℓ_1, \dots, ℓ_k , whereas the IS estimate is obtained by taking the log of the averaged $e^{\ell_1}, \dots, e^{\ell_k}$ (in a numerical stable manner as $\text{logsumexp}(\ell_1, \dots, \ell_k) - \log k$).

By contrast here we do not learn an explicit energy function, but rather learn directly a parameterized transition operator, and define an *implicit* model distribution based on the result of running the Markov chain.

3.4.2. Variational auto-encoders

Variational auto-encoders (VAE) [13, 26] also start from an unstructured (independent) noise sample and non-linearly transform this into a distribution that matches the training data. One difference with our approach is that the VAE typically maps from a lower-dimensional space to the observation space. By contrast we learn a stochastic transition operator from input space to input space that we repeat for T steps. Another key difference, is that the VAE learns a complex heavily parameterized approximate posterior proposal q whereas our *infusion based* q can be understood as a simple heuristic proposal distribution based on p . Importantly the specific heuristic we use to *infuse* \mathbf{x} into q makes sense precisely because our operator is a map from input space to input space, and couldn't be readily applied otherwise. The generative network in Rezende et al. [26] is a Deep Latent Gaussian Model (DLGM) just as ours. But their approximate posterior q is taken to be factorial, including across all layers of the DLGM, whereas our *infusion based* q involves an ordered sampling of the layers, as we sample from $q^{(t)}(\tilde{\mathbf{z}}^{(t)}|\tilde{\mathbf{z}}^{(t-1)}, \mathbf{x})$.

More recent proposals involve sophisticated approaches to sample from better approximate posteriors, as the work of Salimans, Kingma, and Welling [30] in which Hamiltonian Monte Carlo is combined with variational inference, which looks very promising, though computationally expensive, and Rezende & Mohamed [25] that generalizes the use of normalizing flows to obtain a better approximate posterior.

3.4.3. Sampling from autoencoders and Generative Stochastic Networks

Earlier works that propose to directly learn a transition operator resulted from research to turn autoencoder variants that have a stochastic component, in particular denoising autoencoders [37], into generative models that one can sample from. This development is natural, since a stochastic auto-encoder *is* a stochastic transition operator from input space to input space. Generative Stochastic Networks (GSN) [1] generalized insights from earlier stochastic autoencoder sampling heuristics [27] into a more formal and general framework. These previous works on generative uses of autoencoders and GSNs attempt to learn a chain whose *equilibrium distribution* will fit the training data. Because autoencoders and the chain are typically started from or very close to training data points, they are concerned with the chain mixing quickly between modes. By contrast our model chain is always restarted from unstructured noise, and is not required to reach or even have an equilibrium distribution. Our concern is only what happens during the T “burn-in” initial steps, and to make sure

that it transforms the initial factorial noise distribution into something that best fits the training data distribution. There are no mixing concerns beyond those T initial steps.

A related aspect and limitation of previous denoising autoencoder and GSN approaches is that these were mainly “local” around training samples: the stochastic operator explored space starting from and primarily centered around training examples, and learned based on inputs in these parts of space only. Spurious modes in the generated samples might result from large unexplored parts of space that one might encounter while running a long chain.

3.4.4. Reversing a diffusion process in non-equilibrium thermodynamics

The approach of Sohl-Dickstein et al. [33] is probably the closest to the approach we develop here. Both share a similar model sampling chain that starts from unstructured factorial noise. Neither are concerned about an *equilibrium distribution*. They are however quite different in several key aspects: Sohl-Dickstein et al. [33] proceed to invert an explicit *diffusion process* that starts from a training set example and very slowly destroys its structure to become this random noise, they then learn to reverse this process i.e. an *inverse diffusion*. To maintain the theoretical argument that the *exact* reverse process has the same distributional form (e.g. $p(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})$ and $p(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})$ both factorial Gaussians), the diffusion has to be infinitesimal by construction, hence the proposed approaches uses chains with *thousands* of tiny steps. Instead, our aim is to learn an operator that can yield a high quality sample efficiently using only a small number T of larger steps. Also our *infusion* training does not posit a fixed a priori diffusion process that we would learn to reverse. And while the distribution of diffusion chain samples of Sohl-Dickstein et al. [33] is fixed and remains the same all along the training, the distribution of our infusion chain samples closely follow the model chain as our model learns. Our proposed infusion sampling technique thus adapts to the changing generative model distribution as the learning progresses.

Drawing on both Sohl-Dickstein et al. [33] and the walkback procedure introduced for GSN in Alain et al. [1], a variational variant of the walkback algorithm was investigated by Goyal et al. [10] at the same time as our work. It can be understood as a different approach to learning a Markov transition operator, in which a “heating” diffusion operator is seen as a variational approximate posterior to the forward “cooling” sampling operator with the exact same form and parameters, except for a different temperature.

3.5. EXPERIMENTS

We trained models on several datasets with real-valued examples. We used as prior distribution $p^{(0)}$ a factorial Gaussian whose parameters were set to be the mean and variance for each pixel through the training set. Similarly, our models for the transition operators are factorial Gaussians. Their mean and elementwise variance is produced as the output

of a neural network that receives the previous $\mathbf{z}^{(t-1)}$ as its input, i.e. $p^{(t)}(\mathbf{z}_i^{(t)}|\mathbf{z}^{(t-1)}) = \mathcal{N}(\mu_i(\mathbf{z}^{(t-1)}), \sigma_i^2(\mathbf{z}^{(t-1)}))$ where μ and σ^2 are computed as output vectors of a neural network. We trained such a model using our *infusion training* procedure on MNIST [17], Toronto Face Database [34], CIFAR-10 [14], and CelebA [21]. For all datasets, the only preprocessing we did was to scale the integer pixel values down to range [0,1]. The network trained on MNIST and TFD is a MLP composed of two fully connected layers with 1200 units using batch-normalization [12]⁵. The network trained on CIFAR-10 is based on the same generator as the GANs of Salimans et al. [31], i.e. one fully connected layer followed by three transposed convolutions. CelebA was trained with the previous network where we added another transposed convolution. We use rectifier linear units [8] on each layer inside the networks. Each of those networks have two distinct final layers with a number of units corresponding to the image size. They use sigmoid outputs, one that predict the mean and the second that predict a variance scaled by a scalar β (In our case we chose $\beta = 0.1$) and we add an epsilon $\epsilon = 1e - 4$ to avoid an excessively small variance. For each experiment, we trained the network on 15 steps of denoising with an increasing infusion rate of 1% ($\omega = 0.01, \alpha^{(0)} = 0$), except on CIFAR-10 where we use an increasing infusion rate of 2% ($\omega = 0.02, \alpha^{(0)} = 0$) on 20 steps.

3.5.1. Numerical results

Since we can't compute the exact log-likelihood, the evaluation of our model is not straightforward. However we use the lower bound estimator derived in Section 3.3.4 to evaluate our model during training and prevent overfitting (see Figure 3.3). Since most previous published results on non-likelihood based models (such as GANs) used a Parzen-window-based estimator [5], we use it as our first comparison tool, even if it can be misleading [22]. Results are shown in Table 3.4, we use 10 000 generated samples and $\sigma = 0.17$. To get a better estimate of the log-likelihood, we then computed both the stochastic lower bound and the importance sampling estimate (IS) given in Section 3.3.4. For the IS estimate in our MNIST-trained model, we used 20 000 intermediates samples. In Table 3.1 we compare our model with the recent Annealed Importance Sampling results [38]. Note that following their procedure we add an uniform noise of 1/256 to the (scaled) test point before evaluation to avoid overevaluating models that might have overfitted on the 8 bit quantization of pixel values. Another comparison tool that we used is the Inception score as in Salimans et al. [31] which was developed for natural images and is thus most relevant for CIFAR-10. Since Salimans et al. [31] used a GAN trained in a semi-supervised way with some tricks, the comparison with our unsupervised trained model isn't straightforward. However, we can see in Table 3.2 that our model outperforms the traditional GAN trained without labeled data.

⁵We don't share batch norm parameters across the network, i.e for each time step we have different parameters and independent batch statistics.

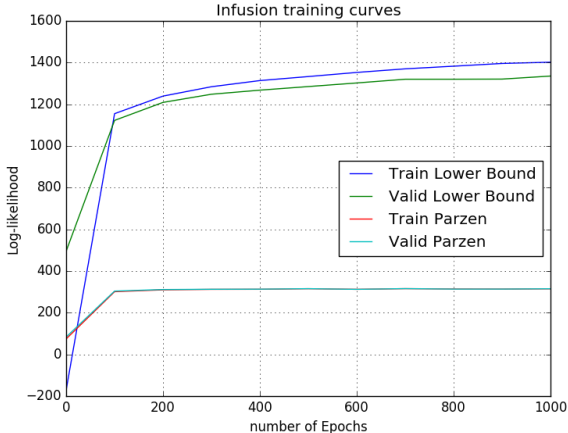


FIG. 3.3. Training curves: lower bounds on average log-likelihood on MNIST as infusion training progresses. We also show the lower bounds estimated with the Parzen estimation method.

TAB. 3.1. Log-likelihood (in nats) estimated by AIS on MNIST test and training sets as reported in [38] and the log likelihood estimates of our model obtained by infusion training (last three lines). Our initial model uses a Gaussian output with diagonal covariance, and we applied both our lower bound and importance sampling (IS) log-likelihood estimates to it. Since [38] used only an isotropic output observation model, in order to be comparable to them, we also evaluated our model after replacing the output by an isotropic Gaussian output (same fixed variance for all pixels). Average and standard deviation over 10 repetitions of the evaluation are provided. Note that AIS might provide a higher evaluation of likelihood than our current IS estimate, but this is left for future work.

Model	Test log-likelihood (1000ex)	Train log-likelihood (100ex)
VAE-50 (AIS)	991.435 ± 6.477	1272.586 ± 6.759
GAN-50 (AIS)	627.297 ± 8.813	620.498 ± 31.012
GMMN-50 (AIS)	593.472 ± 8.591	571.803 ± 30.864
VAE-10 (AIS)	705.375 ± 7.411	780.196 ± 19.147
GAN-10 (AIS)	328.772 ± 5.538	318.948 ± 22.544
GMMN-10 (AIS)	346.679 ± 5.860	345.176 ± 19.893
Infusion training + isotropic (IS estimate)	413.297 ± 0.460	450.695 ± 1.617
Infusion training (IS estimate)	1836.27 ± 0.551	1837.560 ± 1.074
Infusion training (lower bound)	1350.598 ± 0.079	1230.305 ± 0.532

Model	Test
DBM [2]	138 ± 2
SCAE [2]	121 ± 1.6
GSN [4]	214 ± 1.1
Diffusion [33]	220 ± 1.9
GANs (Goodfellow et al.)	225 ± 2
GMMN + AE (Li, Swersky, and Zemel)	282 ± 2
Infusion training (Our)	312 ± 1.7

FIG. 3.4. Parzen-window-based estimator of lower bound on average test log-likelihood on MNIST (in nats).

TAB. 3.2. Inception score (with standard error) of 50 000 samples generated by models trained on CIFAR-10. We use the models in Salimans et al. [31] as baseline. 'SP' corresponds to the best model described by Salimans et al. [31] trained in a semi-supervised fashion. '-L' corresponds to the same model after removing the label in the training process (unsupervised way), '-MBF' corresponds to a supervised training without minibatch features.

Model	Real data	SP	-L	-MBF	Infusion training
Inception score	$11.24 \pm .12$	$8.09 \pm .07$	$4.36 \pm .06$	$3.87 \pm .03$	$4.62 \pm .06$

3.5.2. Sample generation

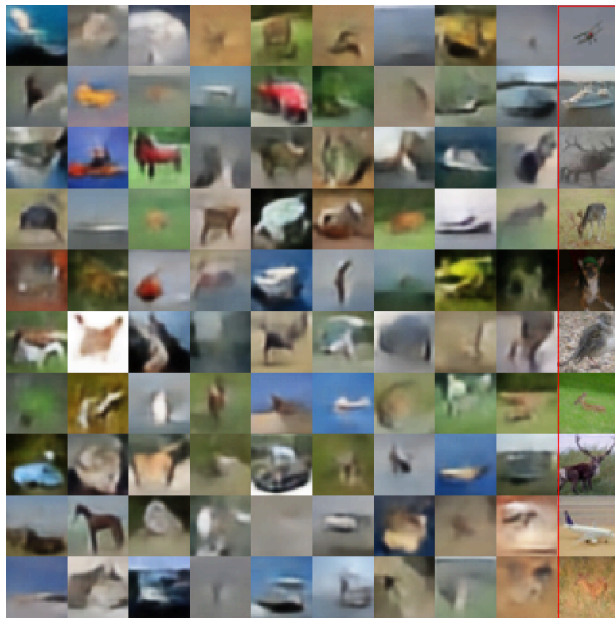
Another common qualitative way to evaluate generative models is to look at the quality of the samples generated by the model. In Figure 3.5 we show various samples on each of the datasets we used. In order to get sharper images, we use at sampling time more denoising steps than in the training time (In the MNIST case we use 30 denoising steps for sampling with a model trained on 15 denoising steps). To make sure that our network didn't learn to copy the training set, we show in the last column the nearest training-set neighbor to the samples in the next-to last column. We can see that our training method allow to generate very sharp and accurate samples on various dataset.

3.5.3. Inpainting

Another method to evaluate a generative model is *inpainting*. It consists of providing only a partial image from the test set and letting the model generate the missing part. In one experiment, we provide only the top half of CelebA test set images and clamp that top half throughout the sampling chain. We restart sampling from our model several times, to see the variety in the distribution of the bottom part it generates. Figure 3.6 shows that the model is able to generate a varied set of bottom halves, all consistent with the same top half, displaying different type of smiles and expression. We also see that the generated bottom halves transfer some information about the provided top half of the images (such as pose and more or less coherent hair cut).

3.6. CONCLUSION AND FUTURE WORK

We presented a new training procedure that allows a neural network to learn a transition operator of a Markov chain. Compared to the previously proposed method of Sohl-Dickstein et al. [33] based on inverting a slow diffusion process, we showed empirically that infusion training requires far fewer denoising steps, and appears to provide more accurate models. Currently, many successful generative models, judged on sample quality, are based on GAN architectures. However these require to use two different networks, a generator and a discriminator, whose balance is reputed delicate to adjust, which can be source of instability



(A) CIFAR-10



(B) CelebA

FIG. 3.5. Mean predictions by our models on 4 different datasets. The right-most column shows the nearest training example to the samples in the next-to last column.

during training. Our method avoids this problem by using only a single network and a simpler training objective.

Denosing-based infusion training optimizes a heuristic surrogate loss for which we cannot (yet) provide theoretical guarantees, but we empirically verified that it results in increasing



FIG. 3.6. Inpainting on CelebA dataset. In each row, from left to right: an image from the test set; the same image with bottom half randomly sampled from our factorial prior. Then several end samples from our sampling chain in which the top part is clamped. The generated samples show that our model is able to generate a varied distribution of coherent face completions.

log-likelihood estimates. On the other hand the lower-bound-based infusion training procedure does maximize an explicit variational lower-bound on the log-likelihood. While we have run most of our experiments with the former, we obtained similar results on the few problems we tried with lower-bound-based infusion training.

Future work shall further investigate the relationship and quantify the compromises achieved with respect to other Markov Chain methods including Sohl-Dickstein et al. [33], Salimans et al. [30] and also to powerful inference methods such as Rezende & Mohamed [25]. As future work, we also plan to investigate the use of more sophisticated neural net generators, similar to DCGAN's [24] and to extend the approach to a conditional generator applicable to structured output problems.

Appendix A

DETAILS ON THE EXPERIMENTS

A.1. MNIST EXPERIMENTS

We show the impact of the infusion rate $\alpha^{(t)} = \alpha^{(t-1)} + \omega$ for different numbers of training steps on the lower bound estimate of log-likelihood on the Validation set of MNIST in Figure A.1. We also show the quality of generated samples and the lower bound evaluated on the test set in Table A.1. Each experiment in Table A.1 uses the corresponding models of Figure A.1 that obtained the best lower bound value on the validation set. We use the same network architecture as described in Section 3.5, i.e two fully connected layers with Relu activations composed of 1200 units followed by two distinct fully connected layers composed of 784 units, one that predicts the means, the other one that predicts the variances. Each mean and variance is associated with one pixel. All of the the parameters of the model are shared across different steps except for the batch norm parameters. During training, we use the batch statistics of the current mini-batch in order to evaluate our model on the train and validation sets. At test time (Table A.1), we first compute the batch statistics over the entire train set for each step and then use the computed statistics to evaluate our model on the test test.

We did some experiments to evaluate the impact of α or ω in $\alpha^{(t)} = \alpha^{(t-1)} + \omega$. Figure A.1 shows that as the number of steps increases, the optimal value for infusion rate decreases. Therefore, if we want to use many steps, we should have a small infusion rate. These conclusions are valid for both increasing and constant infusion rate. For example, the optimal α for a constant infusion rate, in Figure A.1e with 10 steps is 0.08 and in Figure A.1f with 15 steps is 0.06. If the number of steps is not enough or the infusion rate is too small, the network will not be able to learn the target distribution as shown in the first rows of all subsection in Table A.1.

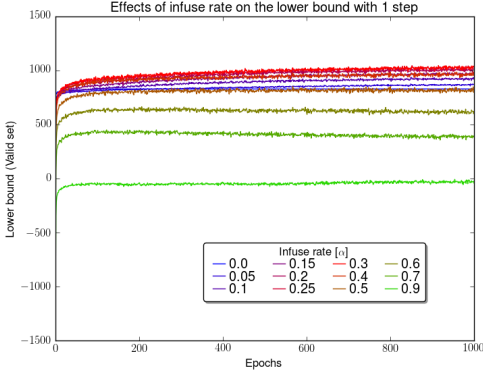
In order to show the impact of having a constant versus an increasing infusion rate, we show in Figure A.2 the samples created by infused and sampling chains. We observe that

having a small infusion rate over many steps ensures a slow blending of the model distribution into the target distribution.

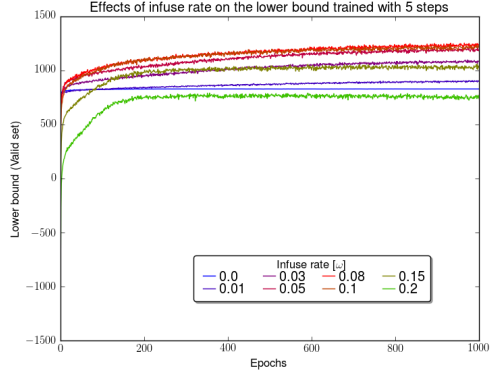
In Table 4, we can see high lower bound values on the test set with few steps even if the model can't generate samples that are qualitatively satisfying. These results indicate that we can't rely on the lower bound as the only evaluation metric and this metric alone does not necessarily indicate the suitability of our model to generated good samples. However, it is still a useful tool to prevent overfitting (the networks in Figure A.1e and A.1f overfit when the infusion rate becomes too high). Concerning the samples quality, we observe that having a small infusion rate over an adequate number of steps leads to better samples.

A.1.1. Infusion and model sampling chains on natural images datasets

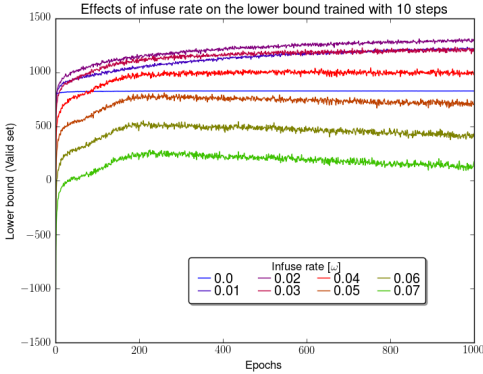
In order to show the behavior of our model trained by Infusion on more complex datasets, we show in Figure A.3 chains on CIFAR-10 dataset and in Figure A.4 chains on CelebA dataset. In each Figure, the first sub-figure shows the chains infused by some test examples and the second sub-figure shows the model sampling chains. In the experiment on CIFAR-10, we use an increasing schedule $\alpha^{(t)} = \alpha^{(t-1)} + 0.02$ with $\alpha^{(0)} = 0$ and 20 infusion steps (this corresponds to the training parameters). In the experiment on CelebA, we use an increasing schedule $\alpha^{(t)} = \alpha^{(t-1)} + 0.01$ with $\alpha^{(0)} = 0$ and 15 infusion steps.



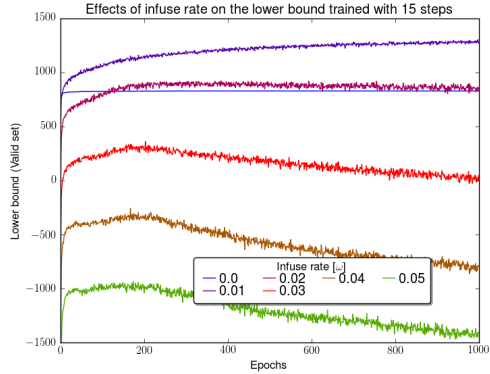
(A) Networks trained with 1 infusion step. Each infusion rate in the figure corresponds to $\alpha^{(0)}$. Since we have only one step, we have $\omega = 0$.



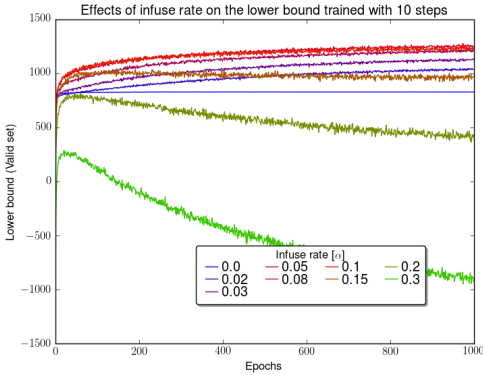
(B) Networks trained with 5 infusion steps. Each infusion rate corresponds to ω . We set $\alpha^{(0)} = 0$.



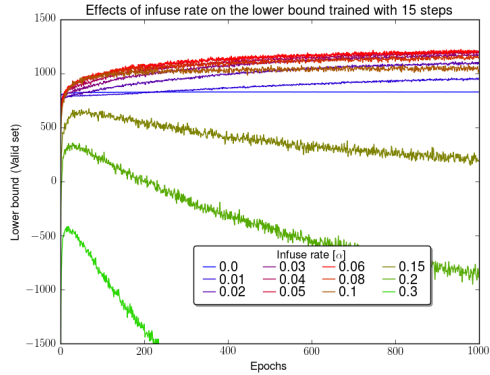
(C) Networks trained with 10 infusion steps. Each infusion rate corresponds to ω . We set $\alpha^{(0)} = 0$.



(D) Networks trained with 15 infusion steps. Each infusion rate corresponds to ω . We set $\alpha^{(0)} = 0$.



(E) Networks trained with 10 infusion steps. In this experiment we use the same infusion rate for each time step such that $\forall_t \alpha^{(t)} = \alpha^{(0)}$. Each infusion rate in the figure corresponds to different values for $\alpha^{(0)}$.



(F) Networks trained with 15 infusion steps. In this experiment we use the same infusion rate for each time step such that $\forall_t \alpha^{(t)} = \alpha^{(0)}$. Each infusion rate in the figure corresponds to different values $\alpha^{(0)}$.

FIG. A.1. Training curves on MNIST showing the log likelihood lower bound (nats) for different infusion rate schedules and different number of steps. We use an increasing schedule $\alpha^{(t)} = \alpha^{(t-1)} + \omega$. In each sub-figure for a fixed number of steps, we show the lower bound for different infusion rates.

TAB. A.1. Infusion rate impact on the lower bound log-likelihood (test set) and the samples generated by a network trained with different number of steps. Each sub-table corresponds to a fixed number of steps. Each row corresponds to a different infusion rate, where we show its lower bound and also its corresponding generated samples from the trained model. Note that for images, we show the mean of the Gaussian distributions instead of the true samples. As the number of steps increases, the optimal infusion rate decreases. Higher number of steps contributes to better qualitative samples, as the best samples can be seen with 15 steps using $\alpha = 0.01$.

(A) infusion rate impact on the lower bound log-likelihood (test set) and the samples generated by a network trained with 1 step.

infusion rate	Lower bound (test)	Means of the model
0.0	824.34	
0.05	885.35	
0.1	967.25	
0.15	1063.27	
0.2	1115.15	
0.25	1158.81	
0.3	1209.39	
0.4	1209.16	
0.5	1132.05	
0.6	1008.60	
0.7	854.40	
0.9	-161.37	


(B) infusion rate impact on the lower bound log-likelihood (test set) and the samples generated by a network trained with 5 steps

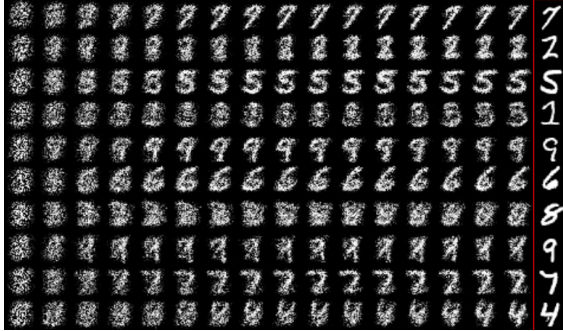
infusion rate	Lower bound (test)	
0.0	823.81	
0.01	910.19	
0.03	1142.43	
0.05	1303.19	
0.08	1406.38	
0.1	1448.66	
0.15	1397.41	
0.2	1262.57	

(c) infusion rate impact on the lower bound log-likelihood (test set) and the samples generated by a network trained with 10 steps

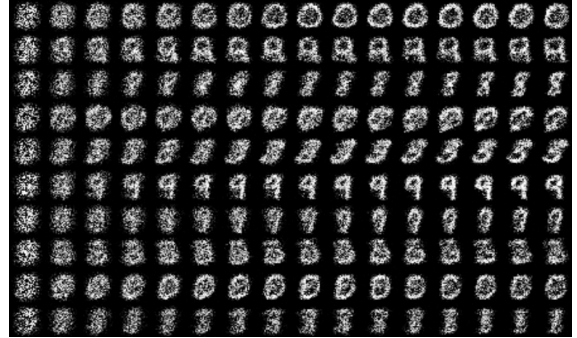
infusion rate	Lower bound (test)	
0.0	824.42	
0.01	1254.07	
0.02	1389.12	
0.03	1366.68	
0.04	1223.47	
0.05	1057.43	
0.05	846.73	
0.07	658.66	

(D) infusion rate impact on the lower bound log-likelihood (test set) and the samples generated by a network trained with 15 steps

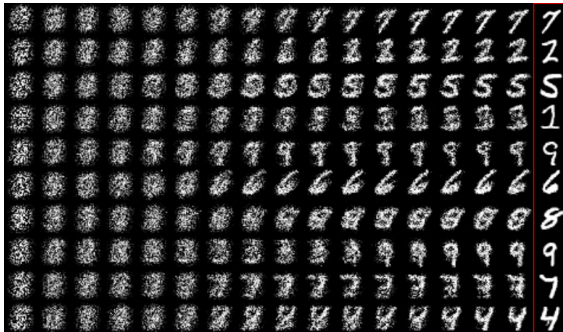
infusion rate	Lower bound (test)	
0.0	824.50	
0.01	1351.03	3 4 6 7 9 7 9 9 7 9 1 0 0 1 7 8 3 1 9 3
0.02	1066.60	5 5 2 1 6 2 9 6 9 9 7 6 7 1 4 6 9 9 9 8
0.03	609.10	6 9 1 1 7 9 7 1 6 7 5 5 1 0 9 9 9 3 0 8
0.04	876.93	4 1 4 1 5 8 6 7 1 4 7 1 3 8 0 6 0 3 5 7
0.05	-479.69	5 7 7 1 2 9 2 7 5 6 4 6 9 6 9 6 9 7 7 8
0.06	-941.78	9 4 4 8 1 9 1 1 2 7 9 6 4 2 4 2 9 8 9 9



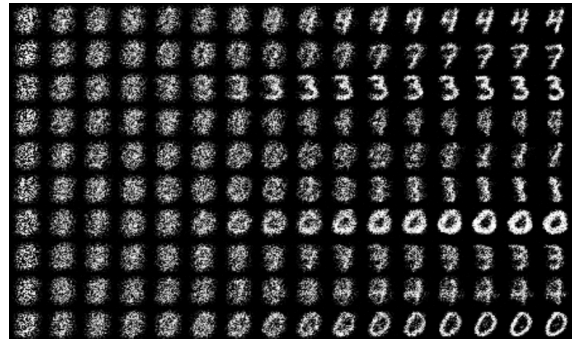
(A) Chains infused with MNIST test set samples by a constant rate ($\alpha^{(0)} = 0.05$, $\omega = 0$) in 15 steps.



(B) Model sampling chains on MNIST using a network trained with a constant infusion rate ($\alpha^{(0)} = 0.05$, $\omega = 0$) in 15 steps.



(C) Chains infused with MNIST test set samples by an increasing rate ($\alpha^{(0)} = 0.0$, $\omega = 0.01$) in 15 steps.

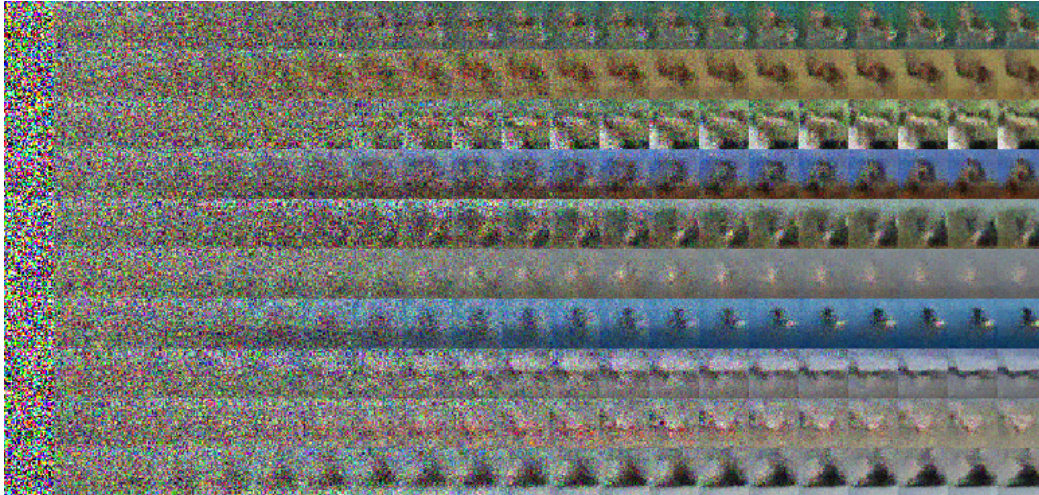


(D) Model sampling chains on MNIST using a network trained with an increasing infusion rate ($\alpha^{(0)} = 0.0$, $\omega = 0.01$) in 15 steps.

FIG. A.2. Comparing samples of constant infusion rate versus an increasing infusion rate on infused and generated chains. The models are trained on MNIST in 15 steps. Note that having an increasing infusion rate with a small value for ω allows a slow convergence to the target distribution. In contrast having a constant infusion rate leads to a fast convergence to a specific point. Increasing infusion rate leads to more visually appealing samples. We observe that having an increasing infusion rate over many steps ensures a slow blending of the model distribution into the target distribution.



(A) Infusion chains on CIFAR-10. Last column corresponds to the target used to infuse the chain.



(B) Model sampling chains on CIFAR-10

FIG. A.3. Infusion chains (Sub-Figure A.3a) and model sampling chains (Sub-Figure A.3b) on CIFAR-10.



(A) Infusion chains on CelebA. Last column corresponds to the target used to infuse the chain.



(B) Model sampling chains on CelebA

FIG. A.4. Infusion chains (Sub-Figure A.4a) and model sampling chains (Sub-Figure A.4b) on CelebA.

CONCLUSION

In this thesis, we present a new way to train a generative model. There is still a long way to go before having computers that understand the world in a similar way that brains do. However research those last years has shown very promising and exciting results. But a good metric to evaluate how good or bad the model has learned to extract meaningful features is still missing. Solving those problems will lead to system that are able to improve our understanding of the world. A perfect generative model for images would lead to perfect reconstruction, to increase or decrease resolution of images at will or be able to perform image segmentation at different level of details. Those methods could bring to a huge improvement of autonomous driving, images synthesis and medical analysis.

Bibliography

- [1] Guillaume Alain, Yoshua Bengio, Li Yao, Jason Yosinski, Eric Thibodeau-Laufer, Saizheng Zhang, and Pascal Vincent. GSNs: generative stochastic networks. *Information and Inference*, 2016. doi: 10.1093/imaiai/iaw003.
- [2] 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 2013)*, 2013.
- [3] Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*, pp. 899–907, 2013.
- [4] Yoshua Bengio, Eric Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, pp. 226–234, 2014.
- [5] Olivier Breuleux, Yoshua Bengio, and Pascal Vincent. Quickly generating representative samples from an rbm-derived process. *Neural Computation*, 23(8):2058–2073, 2011.
- [6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.
- [7] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.
- [8] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, pp. 275, 2011.
- [9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. Curran Associates, Inc., 2014.
- [10] Anirudh Goyal, Nan Rosemary Ke, Alex Lamb, and Yoshua Bengio. The variational walkback algorithm. Technical report, Université de Montréal, 2017. URL <https://openreview.net/forum?id=rkpdnIqlx>. On openreview.net.

- [11] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
- [12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of The 32nd International Conference on Machine Learning*, pp. 448–456, 2015.
- [13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014)*, 2014.
- [14] Alex. Krizhevsky and Geoffrey E Hinton. Learning multiple layers of features from tiny images. *Master’s thesis, Department of Computer Science, University of Toronto*, 2009.
- [15] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *AISTATS*, volume 1, pp. 2, 2011.
- [16] Doris Y. Tsao Le Chang. The code for facial identity in the primate brain. *Cell*, 169(6): 1013–1028.e14, June 2017.
- [17] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits, 1998.
- [18] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pp. 396–404, 1990.
- [19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [20] Yujia Li, Kevin Swersky, and Richard Zemel. Generative moment matching networks. In *International Conference on Machine Learning (ICML 2015)*, pp. 1718–1727, 2015.
- [21] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV 2015)*, December 2015.
- [22] Aäron van den Oord Lucas Theis and Matthias Bethge. A note on the evaluation of generative models. In *Proceedings of the 4th International Conference on Learning Representations (ICLR 2016)*, 2016.
- [23] M Minsky S Papert and ML Minsky. Perceptrons: an introduction to computational geometry. *Expanded Edition*, 1969.
- [24] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *International Conference on Learning Representations*, 2016.
- [25] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, pp. 1530–1538, 2015.
- [26] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the*

- 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pp. 1278–1286, 2014. URL <http://jmlr.org/proceedings/papers/v32/rezende14.html>.
- [27] Salah Rifai, Yoshua Bengio, Yann Dauphin, and Pascal Vincent. A generative process for sampling contractive auto-encoders. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*, 2012.
 - [28] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
 - [29] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *AISTATS*, volume 1, pp. 3, 2009.
 - [30] Tim Salimans, Diederik Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 1218–1226, 2015.
 - [31] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.
 - [32] 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.
 - [33] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *JMLR Proceedings*, pp. 2256–2265. JMLR.org, 2015.
 - [34] Josh M Susskind, Adam K Anderson, and Geoffrey E Hinton. The toronto face database. *Department of Computer Science, University of Toronto, Toronto, ON, Canada, Tech. Rep*, 3, 2010.
 - [35] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016)*, pp. 1747–1756, 2016.
 - [36] 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*, pp. 1096–1103. ACM, 2008.
 - [37] 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. *Journal of Machine Learning Research*, 11 (Dec):3371–3408, 2010.
 - [38] Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger B. Grosse. On the quantitative analysis of decoder-based generative models. *CoRR*, abs/1611.04273, 2016.