Guiding Generative Models via Class Label Information

by

Jan S. Rudy

A Thesis
presented to
The University of Guelph

In partial fulfillment of requirements
for the degree of
Master of Applied Science
in
Engineering

Guelph, Ontario, Canada

©Jan S. Rudy, January, 2016
ABSTRACT

GUIDING GENERATIVE MODELS VIA CLASS LABEL INFORMATION

Jan S. Rudy
University of Guelph, 2016

Advisor:
Dr. Graham W. Taylor

Advisory Committee:
Dr. Rob Deardon

Given a finite number of samples from some high-dimensional distribution, the task of efficiently and accurately modeling the distribution can be challenging. Some datasets, however, provide additional information (e.g. categorical class labels) for each input. When class labels are available, can they be used to better model the data distribution? A conditional modeling and training procedure is introduced for a type of generative model (the generalized denoising autoencoder) and two methods of injecting class label information are presented (additive vs. multiplicative). When trained on natural images, models with access to class information generate samples of higher visual fidelity than those trained on images alone. Additionally, with higher dimensional data, multiplicative architectures outperform their additive counterparts. Finally, experimental results confirm recent findings that Parzen likelihood estimates are a poor measure of visual sample quality.
I would like to thank my advisor, Graham W. Taylor, for his seemingly unlimited supply of patience and dedication. Without his advice and encouragement this thesis would not have been completed.

The tireless work of School of Engineering I.T. Manager Joel Best was instrumental in completing the technical requirements of this thesis. Whether I needed a simple package installed on my workstation or a power failure took down the compute cluster, Joel was only an email away and always happy to help.

Lab members Ammar Abu Leil, Ethan Buchman, Daniel Im, Gavin Ding, Natalia Neverova, Griffin Lacey, and Matt Veres made the lab a pleasure to be a part of. Our lab discussions not only introduced me to new areas of research, but also helped clarify my own goals and interests.

My time in the city of Guelph would not have been the same without the friends I made there. Although my time there was far too short, I'd like to thank Handel Calendar, Martin Ford, Marco Lück and Gerardo Villagran for making me feel at home. Also, pizza club.

Most importantly, I would like to thank my wife, Magdalena Sobol, whose love and support made this work possible. Not only was she always available for encouragement and advice, but the unending stream of fresh baked brownies fueled much of the writing process.

This work was supported financially by both the Canadian and Ontario governments through the Ontario Graduate fellowship award and the NSERC Canadian Graduate Scholarship.
# Table of Contents

List of Tables vi

List of Figures vii

1 **Introduction** 1
   1.1 Objectives ................................................. 3
   1.2 Contributions .............................................. 6
   1.3 Organization ............................................... 7

2 **Background** 8
   2.1 Supervised vs. Unsupervised vs. Semi-supervised Learning .... 9
      2.1.1 Supervised Learning ................................. 9
      2.1.2 Unsupervised Learning ............................. 10
      2.1.3 Semi-supervised Learning ......................... 11
   2.2 Feed Forward Neural Networks ............................... 11
   2.3 Training procedure ......................................... 14
      2.3.1 Gradient Descent .................................. 14
      2.3.2 Backpropagation ................................. 15
   2.4 Autoencoders ............................................... 16
   2.5 Denoising Autoencoders .................................... 17
   2.6 Denoising autoencoders as generative models ............... 19
      2.6.1 Walkback training procedure ....................... 20
   2.7 Generative model evaluation ............................... 21
      2.7.1 Annealed importance sampling (AIS) ............... 22
      2.7.2 Reverse annealed importance sampling (RAISE) .... 23
      2.7.3 Indirect sampling likelihood (ISL) ................ 24
      2.7.4 Conservative Sampling-based Likelihood Estimator (CSL) 25
      2.7.5 Choosing an evaluation method .................... 27
3 Methodology

3.1 Models under review
3.1.1 Conditional and joint autoencoders
3.2 Additive (modulated bias) autoencoder
3.3 Multiplicative (modulated weight) autoencoder
3.4 Geometric interpretation
3.5 Class-conditional Sampling
3.6 Joint Sampling procedure
3.6.1 Logit noise
3.7 Datasets
3.7.1 MNIST database of handwritten digits
3.7.2 CIFAR-10 database
3.7.3 Street View House Numbers (SVHN) dataset
3.7.4 Data preprocessing
3.8 Experimental settings
3.9 Evaluation Criterion
3.10 Implementation

4 Results

4.1 MNIST
4.1.1 Quantitative Results
4.1.2 Qualitative Results
4.2 CIFAR-10
4.2.1 Quantitative results
4.2.2 Qualitative results
4.3 SVHN
4.3.1 Quantitative results
4.3.2 Qualitative results
4.4 Class conditional vs. joint samples
4.5 Disentangling factors of variation

5 Conclusions and Future Work

5.1 Future Work

References

A Model samples
A.1 MNIST Samples
A.2 CIFAR10 Samples
A.3 SVHN Samples
List of Tables

3.1 Models under review, organized over two axes of variation. .................. 31
3.2 Hyperparameter settings used for model training. ......................... 50

4.1 MNIST Parzen log-likelihood results in nats comparing 10,000 model samples to the test set. All samples were generated with a conditional sampling procedure. Results (1) are reported from Goodfellow et al. (2014), (2) are reported from Bengio and Thibodeau-Laufer (2013), (3) are from Sohl-Dickstein et al. (2015) ........................................... 58
4.2 CIFAR-10 Parzen log-likelihood results in nats comparing 10,000 model samples to the test set. ................................. 59
4.3 SVHN Parzen log-likelihood results in nats comparing 10,000 samples to a held out test set. All samples were generated via the conditional sampling procedure. ................................. 65
4.4 Parzen likelihood estimates for both joint and conditional sampling procedures for models trained on MNIST, CIFAR-10, and SVHN. ......... 71
## List of Figures

1.1 Left: samples from some unknown distribution $p(x)$. Right: With class information (i.e. colour) the underlying structure becomes apparent. 5

2.1 Example of decision process for choosing a evaluation method for a generative model. 29

3.1 Illustration of a typical DAE (a) and two methods of incorporating a class label into the model: an additive (modulated bias) autoencoder and (b) a multiplicative (modulated weight) autoencoder (a.k.a. a gated autoencoder) (c). All depicted models have a single hidden layer and biases have been omitted for clarity. 33

3.2 The effect of the $\beta$ parameter on logit-noise on a 10 dimensional one-hot vector. (a) Simulated average entropy vs. $\beta$. The red horizontal line indicates the theoretical maximum of the uniform distribution. (b) Simulated entropy heat map with logit noise applied to a single one-hot vector with increasing $\beta$ from over the range $[0,2]$. Notice that the distribution smoothly transitions from a one-hot (zero-entropy) to a uniform (max-entropy) distribution. 43

3.3 Examples from the MNIST database of handwritten digits. Each row contains examples from a single class. From top to bottom: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. 45

3.4 Examples from the CIFAR-10 dataset. Each row contains examples from a single class. From top to bottom: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. 46

3.5 Examples from the SVHN dataset. Each row contains examples from a single class. From top to bottom: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Although similar to MNIST, the SVHN dataset offers images with greater variation. 48
4.1 Non-consecutive samples from the three top MNIST models using conditional sampling for classes 2, 4, 6, and 8. Top row is a cDAE, middle row is a cGAE with a conditional objective and the bottom row is a jGAE. All samples were generated via a conditional sampling procedure. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample. ........................................... 57

4.2 Samples from two separate generative DAEs trained on MNIST with a reconstruction objective. The model on the left has 258 \text{ tanh} hidden units while the model on the right has 2000. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample. ........................................... 58

4.3 Non-consecutive samples from a cDAE trained on CIFAR-10 using a conditional sampling procedure. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample. 62

4.4 Non-consecutive samples from a cGAE trained on CIFAR-10 using a conditional sampling procedure. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample. 63

4.5 Samples from a generative DAE trained on CIFAR-10 with a reconstruction objective. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample. ............... 64

4.6 Non-consecutive samples from a cDAE trained on SVHN using a conditional sampling procedure. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample. . . 67

4.7 Non-consecutive samples from a cGAE trained on SVHN using a conditional sampling procedure. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample. . . 68

4.8 Samples from a generative DAE trained on SVHN with a reconstruction objective. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample. ............... 69

4.9 Comparison of conditional vs joint sampling procedures for a jDAE trained on MNIST. Samples are consecutive with the $x$ input of the Markov chain initialized to the zero vector and the $y$ input initialized to a single class for each row. In the case of joint sampling, the $y$ corruption and reconstruction allows the samples for a given row to wander between classes. ........................................... 72

4.10 Comparison of conditional vs joint sampling procedures for a jDAE and jGAE trained on CIFAR-10. Samples are consecutive with the Markov chain initialized to the zero vector. ........................................... 73

4.11 Comparison of conditional vs joint sampling procedures for a jDAE and jGAE trained on SVHN. Samples are consecutive with the Markov chain initialized to the zero vector. ........................................... 74
4.12 Analogies created by the cDAE and cGAE. A test example is provided as a seed (leftmost column), and the image is reconstructed with an alternate class label. ................................................................. 77

4.13 Multi-step analogies created by a cGAE trained on MNIST. With each step, the model approaches finds a point attractor on the conditional manifold closest to the original test image (leftmost column). ............... 78

4.14 Multi-step analogies created by cGAE trained on CIFAR-10 and SVHN. A test example is provided as a seed (leftmost column), and the image is reconstructed with an alternate class label. ...................... 80

A.1 Samples from a traditional DAE with 258 tanh hidden units trained on MNIST ................................................................. 93
A.2 Samples from a traditional DAE with 2000 tanh hidden units trained on MNIST ................................................................. 94
A.3 Samples from a GAE trained on MNIST .................................................. 95
A.4 Consecutive samples from additive DAE with conditional objective function. 96
A.5 Consecutive samples from multiplicative GAE with conditional objective function. ......................................................... 97
A.6 Consecutive samples from additive DAE with joint objective function and conditional sampling procedure. ................... 98
A.7 Consecutive samples from an additive DAE with joint objective function and joint sampling procedure. ...................... 99
A.8 Consecutive samples from jGAE and conditional sampling procedure. .. 100
A.9 Consecutive samples from an jDAE and joint sampling procedure. ....... 101
A.10 Samples from a traditional DAE with 1026 tanh hidden units trained on CIFAR ................................................................. 102
A.11 Samples from a traditional GAE with 1026 factors and 1026 tanh hidden units trained on CIFAR ................................................. 103
A.12 Consecutive samples from additive DAE with conditional objective function. 104
A.13 Consecutive samples from multiplicative GAE with conditional objective function. ......................................................... 105
A.14 Consecutive samples from an additive DAE with joint objective function and conditional sampling procedure. ................... 106
A.15 Consecutive samples from an additive DAE with joint objective function and joint sampling procedure. ...................... 107
A.16 Consecutive samples from a multiplicative GAE with joint objective function and conditional sampling procedure. ................... 108
A.17 Consecutive samples from a multiplicative DAE with joint objective function and joint sampling procedure. ................... 109
A.18 Samples from a traditional DAE with 1026 tanh hidden units trained on SVHN ................................................................. 110
A.19 Samples from a traditional GAE with 1026 factors and 1026 tanh hidden units trained on SVHN ............................... 111
A.20 Consecutive samples from additive DAE with conditional objective function. 112
A.21 Consecutive samples from multiplicative GAE with conditional objective function. ................................. 113
A.22 Consecutive samples from an additive DAE with joint objective function and conditional sampling procedure. 114
A.23 Consecutive samples from an additive DAE with joint objective function and joint sampling procedure. 115
A.24 Consecutive samples from a multiplicative GAE with joint objective function and conditional sampling procedure. 116
A.25 Consecutive samples from a multiplicative GAE with joint objective function and joint sampling procedure. 117
Chapter 1

Introduction

Much of the present interest in deep learning was spurred by its successes in the supervised setting. Specifically, discriminative tasks based on perception as object recognition applied to visual data (Krizhevsky et al., 2012; Szegedy et al., 2014; He et al., 2015) and speech recognition for audio (Hinton et al., 2012; Graves et al., 2013; Hannun et al., 2014) have seen dramatic gains in accuracy over the last few years. In addition, deep learning techniques have also been applied to bilingual text translation (Sutskever et al., 2014) and even paired with reinforcement learning to achieve human level performance across a variety of Atari video games (Mnih et al., 2015).

Although the basic ideas behind these recent record-breaking approaches have been around for decades (Rumelhart et al., 1985; Le Cun et al., 1990), the current wave of success can be attributed to the coinciding of a number of events. First, the availability of relatively cheap and fast computer hardware in the form of the graphics processing
units (GPUs) led an order of magnitude reduction in model training times (LeCun et al., 2015). Second, labelled datasets have vastly grown in both the number of dimensions and the number of examples. For example, the MNIST dataset released in 1998 contains only 60,000 training examples, where each example is a 28x28 greyscale image of a single handwritten digit (LeCun and Cortes, 1998). In contrast, the training data from the 2015 ImageNet classification challenge contains 1.2 million full-colour images, each labeled from one of 1000 distinct classes (Russakovsky et al., 2015). Finally, advances in regularization techniques such as Dropout have allowed researchers to train large and extremely powerful models while avoiding overfitting to the training data (Srivastava et al., 2014).

Discriminative models have garnered much of the current attention, as their applicability to real world problems is inherent in their formulation. Their objective is to learn an unknown function based on a dataset consisting of input/output pairs. The trained models can then be applied to real world data to answer useful questions, such as “What traffic signs are in view?” or “What does Où est la piscine? mean in English?”.

Although deep generative models and unsupervised training have attracted some interest from researchers, the focus has been predominantly theoretical. As such the commercial interest and applications have fallen short of the widespread adoption of supervised models. However, such models show potential promise as they can make use of the effectively unlimited amounts of unlabeled data freely available online. In 2006, it was a method for unsupervised generative pretraining of discriminative models
that initiated the current interest in neural networks (Hinton and Salakhutdinov, 2006). In this work, Hinton and Salakhutdinov (2006) introduced a method of layerwise generative pretraining using stacks of RBMs to initialize the weights of a discriminative model. Since this time, many deep generative models have been introduced, including Deep Boltzmann machines (Salakhutdinov and Hinton, 2009), generalized denoising autoencoders (Bengio et al., 2013), generative stochastic networks (Bengio and Thibodeau-Laufer, 2013), variational autoencoders (Kingma and Welling, 2014) and generative adversarial networks (Goodfellow et al., 2014).

1.1 Objectives

This thesis aims to answer two related questions:

1. Does the incorporation of class label information improve the quality of samples from a generative model?

2. What architectures and/or sampling procedures are best suited for such models?

As already mentioned, generative pretraining showed initial promise as a means of initializing discriminative models – especially when labeled training data is scarce. Deep learning models, in essence, are a means of learning a hierarchy of feature extractors. When performing classification, training labels typically have far fewer bits of information than the data we wish to classify. With limited labeled training data, it may be advisable to make use of generative pretraining to learn useful features from the data
itself. Much of the field of semi-supervised learning aims to explore how unsupervised approaches can aid in the training of supervised discriminative models. Such approaches make the assumption that the input distribution $p(x)$ and the conditional distribution learned by supervised training $p(y|x)$ share useful information (Lasserre et al., 2006).

This thesis aims to explore a related question: can label information be used to inform generative models? Where the data dimensionality is high, it is often difficult to train a generative model to capture the data distribution. Under the manifold hypothesis, we assume that the input data is concentrated along lower dimensional, yet highly non-linear manifolds in the input space (Cayton, 2005). Semi-supervised learning approaches take this idea one step further and posit that when the input data can be separated into distinct classes, these classes form a family of separable sub-manifolds. It is this separation which makes classification possible (Chapelle et al., 2006).

Where semi-supervised approaches explore this idea as a means of improving classification performance, this thesis investigates the use of class information to improve generative performance. In other words, can learning about $p(x|y)$ inform a generative model of $p(x)$? Specifically, we examine the special case where $y$ is a class label and $x$ is from some high dimensional unknown distribution. For example, this may be the distribution over images of handwritten digits. Where $x$ may be complex and difficult to model, the class labels $y$ can be modeled via a simple multinomial distribution whose parameters can be easily estimated via the training data. Under the assumption that each data point $x$ is drawn from a categorical mixture model over the class labels, sam-
Figure 1.1: Left: samples from some unknown distribution $p(x)$. Right: With class information (i.e. colour) the underlying structure becomes apparent.

Examples of $x$ can be generated by first sampling from the simple distribution $p(y)$, and then from $p(x|y)$.

Figure 1.1 provides a simplified visual example where each point in the left plot represents an example from some unknown input distribution $p(x)$. Attempting to model this distribution may not be trivial, as the underlying structure may not be apparent from the samples alone. However, if we are also provided class label $y$ for each point (as depicted as colour in the plot on the right), it is apparent that each class has been generated by a simpler distribution – in this case a multivariate Gaussian.

The real world data we wish to model will be far more complex than this trivial example. As such, we will require a far more expressive class of models to capture
the underlying structure of each class. A number of such models have been introduced above, such as the generative stochastic networks, variational autoencoders or generative adversarial networks.

The analysis contained in this thesis will focus solely on generalized denoising autoencoders as generative models (Bengio et al., 2013). By holding the overall class of generative models fixed, the current analysis will attempt to experimentally determine the most effective means of incorporating the class label information. A number of models will be trained, each with different connectivity and/or training objective.

1.2 Contributions

The contributions of this thesis include:

- This work demonstrates quantitatively and qualitatively that generative models with access to class label information produce higher quality samples than those without access to class labels.

- This work also shows that models which incorporate class label information via multiplicative interaction between hidden units outperform models with additive interaction.

- In work that parallels Theis et al. (2015), this thesis provides further evidence that Parzen likelihood estimates are a poor measure of visual sample quality.
1.3 Organization

Chapter 2 begins with a brief overview of unsupervised and supervised learning, followed by an introduction to feed forward neural networks, gradient descent, and back-propagation. Next, a review of denoising autoencoders is provided along with an introduction to their training and sampling procedures put forth by Bengio et al. (2013). The chapter is concluded with an overview of evaluation metrics for generative models.

Chapter 3 describes the experimental setup with which our hypothesis will be tested. The chapter contains a description of the types of models under review, along with the accompanying sampling procedures. Finally, three benchmark datasets are presented, followed by an overview of hyperparameter selection used in the experiments.

Chapter 4 presents quantitative and qualitative results. Visual samples are provided along with Parzen likelihood estimates from each model. Where possible, comparison to models in the literature are also provided. The thesis is concluded in Chapter 5 with an overview of the findings and directions for future work.
Chapter 2

Background

The following chapter places the current work into a greater research context. We begin by defining supervised, unsupervised and semi-supervised learning. Next, we introduce feed forward neural networks and stochastic gradient descent in terms of updating the parameters of such models. As our experimental analysis will be focused on denoising autoencoders as generative models, we next introduce such models, along with their training and sampling procedure. Finally, we provide a summary of methods used for evaluation of generative models.
2.1 Supervised vs. Unsupervised vs. Semi-supervised Learning

2.1.1 Supervised Learning

Machine learning methods typically fall into one of two regimes: \textit{supervised} and \textit{unsupervised} (Murphy, 2012, pg. 2). Supervised methods aim to learn a function which maps between an input domain $\mathcal{X}$ and an output domain $\mathcal{Y}$. This function is learned via a \textit{labeled} training set $\mathcal{D} = \{x_i, y_i\}$ (Murphy, 2012, pg. 2). Here, the term \textit{labeled} refers to the fact that each input $x_i$ in the training set is paired with output (or label) $y_i$. When $y_i$ represents a class label (i.e. $y_i = y_i \in \{1, \ldots, C\}$), such supervised methods are also known as \textit{classification} (Murphy, 2012, pg. 3). Where $y_i$ is real-valued, on the other hand, such models fall under the scope of \textit{regression} (Murphy, 2012, pg. 8). In either case, the training set $\mathcal{D}$ provides the training signal for the model.

However, learning a simple mapping between the inputs and labels of the training data is not sufficient – the true aim of training such models is generalization to held out test data. For this generalization to even by possible, all supervised methods rely on certain assumptions regarding the $x$ and $y$ distributions. Namely, that if two inputs $x_1, x_2$ are similar, then their corresponding labels $y_1, y_2$ should also be similar (Chapelle et al., 2006). Chapelle et al. (2006) refers to this as the \textit{supervised smoothness assumption}. 
2.1.2 Unsupervised Learning

By contrast, unsupervised learning does not aim to learn a mapping between an input \(x_i\) and an output \(y_i\). Instead, unsupervised methods attempt to learn some internal structure of a training set \(x_i \in \mathcal{D}\) and often involve learning a generative model of the data (Murphy, 2012, pg. 9). By generative, we refer to a model which aims to capture the density of the data generating distribution. Note that generative models are \textit{not} synonymous with unsupervised learning, as generative approaches can also be used in the supervised setting. For example, by modeling distribution \(p(x|y)\) and applying Bayes theorem, we can construct a generative classifier of the form \(p(y|x) \propto p(x|y)p(y)\) (Murphy, 2012, pg.65).

The assumptions made regarding the nature of the dataset \(\mathcal{D}\) guide our choice of model. Clustering models assume that each data point is associated with exactly one of \(k\) groups. Mixture models, on the other hand, are probabilistic models which assume that the data was sampled from a convex combination of \(k\) base distribution, each of which typically has the same form. Regardless of the type of model, unsupervised learning invariably projects an input vector \(x\) onto some unknown (or \textit{latent}) space \(h\). In the case of clustering, \(h\) is the single categorical variable denoting the cluster centre, whereas mixture models project \(x\) onto a \(k\) dimensional vector of responsibilities of each component.
2.1.3 Semi-supervised Learning

Semi-supervised models occupy a middle ground between these two regimes. Where a discriminative supervised model can be seen as modeling $p(y|x)$, an unsupervised generative approach models $p(x)$. As mentioned in Chapter 1, semi-supervised learning is applicable when the information gained from modeling $p(x)$ is useful for modeling $p(y|x)$ (Lasserre et al., 2006).

While supervised models rely on the supervised smoothness assumption, semi-supervised training makes an additional assumption regarding the data. Recall that under the manifold assumption, we assume that the input distribution lies in a lower dimensional subspace of the input space. The semi-supervised smoothness assumption posits that not only does the data lie along a manifold, but that each class lies along its own sub-manifold. Additionally, we assume that each of these sub-manifolds is separated by lower density regions (Chapelle et al., 2006). As such, if we can learn a better model of the data manifold (via unsupervised methods), we should have a clearer picture of these low density boundaries between classes. In this way, we can use generative approaches to learning $p(x)$ on unlabeled data to inform a classification model of $p(y|x)$.

2.2 Feed Forward Neural Networks

Feed forward neural networks (also known as a multilayer perceptrons or MLPs) are a class of learned function approximators which are composed of a series of layers. Each
layer is stacked upon the previous layer, with the previous layer’s output acting as the
input to the next layer. The layers all have the same basic form and define a non-linear
function of their input. Such models are extremely expressive and with enough capacity,
it has been shown that feed forward networks can approximate any measurable function
to an arbitrary degree of accuracy (Hornik et al., 1989).

Let us assume some domain $\mathcal{X}$, some codomain $\mathcal{Y}$, and some unknown mapping
$\mathcal{F} : x \mapsto y$ for all $x \in \mathcal{X}$ such that $y \in \mathcal{Y}$. Let $\mathcal{T}$ be some finite subset of the do-
main/codomain pairing, i.e. let $\mathcal{T} = \{(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)\}$ such that $x_i \in \mathcal{X}$,
$\mathcal{F}(x_i) = y_i$ and each $x_i$ is an independent and identically distributed sample from $\mathcal{X}$.
Since we assume that $\mathcal{T}$ can be summarized by some unknown function $\mathcal{F}$, the neural
network is thus trained to be an approximation of this true function, modulo any sam-
pling noise present in the subset $\mathcal{T}$. In other words, $f(x_i) = \hat{y}_i$ such that $\hat{y}_i \approx y_i$ for all
$i$.

Consider a feed forward neural network with $L$ hidden layers where we define
$h^{(0)} = x$ and $h^{(L+1)} = y$. Define $h^{(l)}$ as the output of the non-linear function described
by hidden layer $l$. Next, define the model parameters: the weights $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and
the biases $b^{(l)} \in \mathbb{R}^{n_l}$ for layer $l$ where $n_i$ is the size of the $i^{th}$ hidden layer. The value of
each hidden layer is computed by

$$h^{(l)} = \sigma^{(l)} \left( W^{(l)} h^{(l-1)} + b^{(l)} \right)$$

(2.1)
where $\sigma^{(t)}$ is an element-wise non-linear function. The choice of non-linearity (as well as the number and size of the hidden layers) is a model hyper-parameter and is at the discretion of the model architect. Sigmoidal functions have traditionally been used (such as the hyperbolic tangent or the logistic function), however models making use of piecewise linear activation functions (such as the rectified linear function (ReLU)) have exhibited state of the art performance across a number of domains (Krizhevsky et al., 2012; Goodfellow et al., 2013; He et al., 2015; Breuel, 2015).

Similarly, the model’s output layer is defined as

$$ f(x) = \hat{y} = \sigma^{(y)} \left( W^{(L+1)} h^{(L)} + b^{(L+1)} \right) $$

(2.2)

where $\sigma^{(y)}$ is the output activation function, the choice of which is dependent on the nature of the codomain $\mathcal{Y}$. For example, for real valued output $\sigma^{(y)}$ may be the identity and where $y$ is a “one-hot” encoding of a class label $\sigma(y)$ may be the softmax function.

All weight matrices and bias vectors are (randomly) initialized and then learned via minimization of a cost (or error) function over the training set $\mathcal{T}$. The range and nature of the random initializations must be chosen with care, as the initialization schemes has been shown to have an effect on both learning dynamics and quality of the learned solution (Glorot and Bengio, 2010; Sutskever et al., 2013; Saxe et al., 2013).
2.3 Training procedure

2.3.1 Gradient Descent

First, let us define $f(x; \theta)$ as the model output for input $x$, where $\theta$ represents a particular setting of the all the model parameters (i.e. weights and biases). To train such models, we define some smooth loss function, $L(f(x; \theta), y)$ over each input/output pair in the training set. The training object is to minimize this loss function, and this can be done via gradient descent strategies. That is, we define an iterative procedure which evaluates the gradient of our loss (for each data point) with respect to $\theta$, and then takes a small step in the direction of steepest downhill direction. (Murphy, 2012, pg.262)

Formally, we first take the gradient with respect to each of the model parameters

$$
\delta_i = \sum_T \frac{\partial L(f(x; \theta_i), y)}{\partial \theta_i}
$$

(2.3)

where $\theta_i$ are the model parameters at training epoch $i$. The parameters are then adjusted as follows:

$$
\theta_{i+1} = \theta_i - \eta \delta_i
$$

(2.4)

where $\eta$ the learning rate. The learning rate is a hyperparameter and is either set to some small fixed value, annealed during training, or dynamically adapted at training time (Murphy, 2012, pg.263).

Due to the size of the datasets used in practice, the true gradient is often prohibitively
expensive to compute. Thus, the true gradient is typically approximated by the gradient at a specific data point or, more commonly, over a small subset of points in the training set. The former is known as stochastic gradient descent (when the data point is randomly sampled), while the latter is known as mini-batch gradient descent. Matrix computations used in mini-batch gradient descent can be vectorized, offering computational advantages when performed on parallel hardware, e.g. graphics processing units (GPUs). In addition to the computational advantages, these approximate gradients also lead to faster convergence to better solutions than online methods (LeCun et al., 2012; Breuel, 2015).

### 2.3.2 Backpropagation

The expressive power of MLPs is a function of both number of layers and the number of hidden units per layer. Recent theoretical results support that which is often seen in practice: that deep networks are far more efficient – i.e. they require far fewer parameters to model complex functions. It was shown in Cohen et al. (2015) that shallow models of exponential size are equivalent to deep models of only polynomial size.

Backpropagation is an algorithm for efficiently calculating the gradient in deep models. The errors are propagated backwards through the model, from the output layer back towards the input layer in a manner analogous to that of the signal flow from input to output in the forward pass. As both the backward and forward passes share the same computation complexity, the learning step adds only a constant factor to the training
2.4 Autoencoders

An autoencoder is a machine learning model which aims to minimize the reconstruction error of an input data vector via a latent representation, thus falling into the category of unsupervised models. Autoencoders can be interpreted as the composition of two learned functions, an encoder function $f$ and the decoder function $g$. The encoder function $f$ is a mapping from input space onto a representation space. Both $f$ and $g$ are typically parameterized as feed forward neural networks and, as such, we will assume this parameterization for the remainder of this thesis. A single layer encoder function $f$ can be formalized as

$$f(x) = \sigma^{(enc)}(Wx + b)$$ (2.5)

where $x \in \mathbb{R}^{n_X}$ is the input vector, $W \in \mathbb{R}^{n_H \times n_X}$ is the weight matrix, $b \in \mathbb{R}^{n_H}$ is the hidden bias vector, $n_H$ is the dimension of the hidden representation, and $\sigma^{(enc)}$ is an activation function. As mentioned in Section 2.2, the activation is a non-linear function often from the sigmoidal family (e.g. the logistic or hyperbolic tangent functions) or a piecewise linear function (e.g. rectified linear).

The decoder function $g$ projects this representation $h = f(x)$ back onto the input
space. Again, we parameterize the decoder as single layer feed forward neural network

\[
\hat{x} = g(h) = \sigma^{\text{dec}}(W'h + b')
\]  

(2.6)

where \(\hat{x}\) is the reconstruction of the input vector, \(\sigma^{\text{dec}}\) is the output activation function, \(W' \in \mathbb{R}^{n_X \times n_H}\) are the output weights and \(b' \in \mathbb{R}^{n_X}\) are the output biases. In order to restrict the number of free parameters of the model and to aid in learning, the input and output weight matrices are often ‘tied’ such that \(W' = W^T\).

The model parameters (i.e. weights and biases) are updated via a gradient-based optimization in order to minimize a loss function \(L(x)\) based on reconstruction error. The choice of loss function depends on the data domain. When dimensions of \(x\) are real-valued, a typical choice of \(L(x)\) is the squared error, i.e. \(L(x) = \sum_{i=1}^{n_X} (x_i - \hat{x}_i)^2\). When \(x\) is binary, a more appropriate loss function is the cross-entropy loss, i.e. \(L(x) = \sum_{i=1}^{n_X} x_i \log \hat{x}_i + (1 - x_i) \log(1 - \hat{x}_i)\).

### 2.5 Denoising Autoencoders

When the dimension of the hidden representation \(n_H\) is smaller than the dimension of the data space \(n_X\), the learning procedure encourages the model to learn the underlying structure of the data. The data representation can exploit structure in order to compress the data to fewer dimensions than the original space. As such, each dimension of the representation space is interpretable as a useful feature of the data. For example,
when trained on images of handwritten digits these features are interpretable as strokes.

However, when $n_H \geq n_X$, the autoencoder can achieve perfect reconstruction without learning useful features in the data by simply learning the identity function. In this so-called “overcomplete” setup, regularization is essential. Among the various kinds of regularized autoencoders, the denoising autoencoder (DAE) (Vincent et al., 2008) is among the most popular. Instead of reconstructing the data from the actual input vector $x$, the DAE attempts to reconstruct the original input from an encoding of a corrupted version, $f(\tilde{x})$. This effectively prohibits the model from learning a trivial solution while learning robust features of the data.

The corruption procedure is defined as a sample from the conditional distribution $\tilde{x} \sim C(\tilde{x}|x)$. Here, $\tilde{x}$ is a noisy version of $x$ where the type of noise is defined by the distribution $C(\tilde{x}|x)$. Vincent et al. (2010) provide the following typical choices for the noise distribution:

1. Gaussian noise - adding $\epsilon_i \sim N(0, \sigma)$ to each dimension.

2. Masking noise - setting $x_i = 0$ with probability $\rho$,

3. Salt-and-pepper noise - similar to masking noise, but corrupting $x_i$ with probability $\rho$ and each corrupted dimension is set to $x_i = 0$ or $x_i = 1$ with probability 0.5.

Apart from preventing the model from learning a trivial representation of the input by acting as a form of regularization, the DAE can be interpreted as a means of learning
the data manifold (Vincent et al., 2008). The corruption procedure moves a given input into a nearby low density region. In order to successfully denoise the corrupted input, the model must learn to project the corrupted input back onto the manifold, thus learning the boundaries of the high density region of the data generating distribution.

### 2.6 Denoising autoencoders as generative models

Although DAEs are useful as a means of pre-training discriminative models, especially when stacked to form deep models (Vincent et al., 2010), recent work by Bengio et al. (2013) has shown that DAEs and their variants locally characterize the data generating density. This provides an important link between DAEs and probabilistic generative models.

We define observed data $\mathbf{x}$ such that $\mathbf{x} \sim \mathcal{P}(\mathbf{x})$ where $\mathcal{P}(\mathbf{x})$ is the true data distribution and we define $C(\tilde{\mathbf{x}}|\mathbf{x})$ as the conditional distribution of the corruption process. When such models are trained using a loss function that can be interpreted as log-likelihood, by predicting $\mathbf{x}$ given $\tilde{\mathbf{x}}$, the model is learning the conditional distribution $\mathcal{P}_\theta(\mathbf{x}|\tilde{\mathbf{x}})$ (where $\theta$ represents the parameters of the model). (Bengio et al., 2013)

In order to generate samples from the model, one simply forms a Markov chain which alternately samples from learned distribution and the corruption distribution (Bengio et al., 2013). Where $\mathbf{x}_t$ is the state of the Markov chain at time $t$, then $\mathbf{x}_t \sim \mathcal{P}_\theta(\mathbf{x}|\tilde{\mathbf{x}}_{t-1})$ and $\tilde{\mathbf{x}}_t \sim C(\tilde{\mathbf{x}}|\mathbf{x}_t)$. In other words, samples can be generated by alternating between the corruption process and the denoising function learned by the autoencoder.
Bengio et al. (2013) stress that this is not a true Gibbs sampling procedure, as \((x_t, \tilde{x}_{t-1})\) may not share the same asymptotic distribution as \((x_t, \tilde{x}_t)\). Regardless, theoretical results indicate that under some conditions elaborated on in section 3.5, the asymptotic distribution of the generated samples converges to the data-generating distribution (Bengio et al., 2013).

### 2.6.1 Walkback training procedure

Although the above procedure does produce convincing samples, it will also generate spurious examples that lie far from any of the training data. This is because the training procedure does not sufficiently explore the input space. Under the manifold interpretation described above, the corruption procedure defines a region around each example that is explored during training, the size of which is determined by the amount of the corruption (e.g. \(\sigma\) in the case of Gaussian noise and \(\rho\) for masking or salt and pepper noise). This can leave much of the input space unexplored, allowing the model to place measurable amounts of probability mass (i.e. spurious modes) in regions of the space that lie far from any training example (Bengio et al., 2013).

One solution involves using large or increasing amounts of noise during training, however this results in a naive search of the space. A more efficient procedure called “walkback training” is described by Bengio et al. (2013) and bears a resemblance to the contrastive divergence training used to train restricted Boltzmann machines (RBMs) (Hinton, 2002). Instead of defining the loss as the reconstruction cost of a single step...
of corruption and reconstruction chain, walkback training defines a series of $k$ recon-
structions via a random walk originating at the training example. Each reconstruction
is corrupted and subsequently reconstructed, where the final cost is defined as the sum
of the reconstruction costs of each intermediate reconstruction. Since the training pro-
cEDURE mimics that of the sampling procedure, walkback training is a means of seeking
out these spurious modes and redistributing their probability mass back towards the
manifold.

2.7 Generative model evaluation

Evaluating generative models presents a unique set of challenges. In the supervised
setting, the input/output pairs in the training and test data provided a natural means
evaluating performance. For example, when training a discriminative models the efficacy
can be measured via classification performance on a held out test set. Unfortunately no
such analog exists for generative models. If one is attempting to learn an unknown dis-
tribution based solely on set of examples, how is one to measure how well the generated
samples match that unknown distribution?

Despite these challenges, a number of methods have been proposed which attempt
to measure the likelihood of held out test data under the learned model. However, these
methods are typically dependent on the architecture and statistical properties of the
generative model. As such, each have their own set of drawbacks and limitations which
impede their wide spread acceptance as a performance metric.
Very coarsely, these methods can be divided into two main categories:

1. Methods which evaluate the likelihood of a test set under a trained model.

2. Methods which evaluate the likelihood of a test set under a set of samples from a trained model.

In the following sub-sections, we will examine two metrics from each category: annealed importance sampling (AIS) and reverse annealed importance sampling (RAISE) from the first and indirect sampling likelihood (ISL) and conservative sampling-based likelihood estimator (CSL) from the second.

### 2.7.1 Annealed importance sampling (AIS)

So called energy-based models, such as restricted Boltzmann machines, deep belief networks and Markov random fields are trained via the minimization of an energy function. This energy function is related to the probability function as follows:

\[
p(x) = \frac{\exp(-E(x))}{Z} \tag{2.7}
\]

where \(E(x)\) is the energy at data point \(x\) and \(Z\) is the model’s partition function (Bishop, 2006, pg.554). Where the likelihood of the training set can be maximized under the model via a minimization of the energy function, the exact likelihood cannot be computed since the partition function \(Z\) is typically intractable for even moderately sized models.
However, this partition function can often be estimated via the (AIS) algorithm (Neal, 2001). This is done by first choosing a tractable initial distribution \( p_0(x) \), and gradually transforming (or annealing) it into the intractable target distribution \( p_K(x) \) via a series of \( K \) steps.

Although the AIS estimate is unbiased, it can be shown that it will, on average, be an underestimation of the log partition function (Salakhutdinov and Murray, 2008). When this estimate is applied to equation 2.7, this leads to an over-estimation of the probability. When calculating the probability of held out test data, this over-estimation is problematic as it gives an overly optimistic estimate of the model’s performance.

### 2.7.2 Reverse annealed importance sampling (RAISE)

As with AIS, reverse annealed importance sampling (Burda et al., 2014) is an algorithm which is used to estimate the partition function in energy based models. However, RAISE was proposed to address the fact that AIS tends to report overly optimistic performance of generative models. Where AIS estimates the partition function via a series of \( K \) intermediate distributions, RAISE estimates the likelihood of a single \( x_i \) under the trained model via a similar chain of \( K \) interpolated distributions.

Where AIS tends to underestimate the partition function (thus overestimating the likelihood), the RAISE estimate is designed in such a way as to underestimate the likelihood of a given sample, providing a conservative estimate of model performance (Burda et al., 2014). Although the likelihood estimates will converge to the true likelihood as
$K \to \infty$ (just as with AIS), the RAISE estimate is not strictly guaranteed to underestimate the test likelihood for finite $K$. This may be the case if, for example, the intermediate distribution is better at modeling the data than the original model itself (Burda et al., 2014).

In both AIS and RAISE, the estimates converge to the true likelihood as $K \to \infty$. Interestingly, experiments on a small selection of datasets has shown that for sufficiently large $K$ the two estimates will indeed converge. In Burda et al. (2014), $K = 100,000$ showed convergence between AIS and RAISE to within 0.1 nats on most dataset/model combinations.

### 2.7.3 Indirect sampling likelihood (ISL)

One of the most general methods for the evaluation for generative models, ISL, was proposed by Breuleux et al. (2010). In order to compute the ISL, one only requires a held out test set, as well as samples from the trained model. As such, the method makes no assumptions about the model in question. As ISL only requires samples from the model, the procedure can also be used to evaluate different sampling procedures for a single model.

Given a trained generative model, a set of samples $S$ from that model, Breuleux et al. (2010) describes the ISL computation as follows:

1. Samples $S$ are generated from our trained model.

2. Fit a tractable density model $P$ to generated samples $S$. 
3. Compute the likelihood of the test set under $\mathcal{P}$.

For continuous data, the model $\mathcal{P}$ is typically a mixture of Gaussians where each is centred at a sample in $S$ with diagonal covariance (i.e. Parzen density estimator):

$$
\mathcal{P}(x) = \frac{1}{N} \sum_{\hat{x} \in S} \mathcal{N}(x; \mu = \hat{x}, \sigma I)
$$

(2.8)

where $|S| = N$ and $\sigma$ is the spreading parameter used to control the amount of probability density that is redistributed to points near each sample.

When data is binary, a factorized Bernoulli distribution is often used:

$$
\mathcal{P}(x) = \frac{1}{N} \sum_{\hat{x} \in S} \prod_{j=1}^{d} \beta^{1_{x_{j} = \hat{x}_{j}}} (1 - \beta)^{1_{x_{j} = \hat{x}_{j}}}
$$

(2.9)

where the function for the $\beta$ hyper-parameter is an analogous to the $\sigma$ hyper-parameter used in the Gaussian case – it is used to spread out probability mass to neighbors of the generated samples. As the likelihood estimate is dependent on this parameter, it can be chosen via log-likelihood on the validation set.

### 2.7.4 Conservative Sampling-based Likelihood Estimator (CSL)

Despite the promise of both AIS and RAISE as evaluation criterion for generative models, especially when used in conjunction, both AIS and RAISE are only applicable to energy based models where the model learns an unnormalized log probability of the data. As such, they are not directly applicable to the models like those described in 2.6.
Bengio and Yao (2013) proposed the conservative sampling-based likelihood to address the problem of AIS overestimating the likelihood while also being applicable to models where AIS and RAISE are not suitable (i.e. the model formulation does not supply an unnormalized probability function).

In order to calculate the CSL estimator, the generative model must define a Markov chain that alternately samples from the input space and some latent state, as is the case with both restricted Boltzmann machines and the denoising autoencoder described in section 2.6. Where ISL is calculated via a set of samples from the input distribution, CSL instead utilizes a set of samples $H$ from the latent state and is calculated as follows:

$$
\log(f(x)) = \log \left( \frac{1}{N} \sum_{\hat{h} \in H} P(x|\hat{h}) \right).
$$

(2.10)

By relying on samples from the hidden state, the estimator has lower variance than ISL. Also, CSL does away with the need to choose some tractable distribution along with any associated hyper-parameters. Finally, Bengio and Yao (2013) provide proof that the CSL estimator is both conservative in average and asymptotically unbiased.

Unfortunately, the work of Burda et al. (2014) found that the estimates obtained by CSL can be too conservative to be useful as a means of model comparison. Across a number of datasets and models, the CSL estimator was found to underestimate the log probability by 2 - 60 nats, where AIS and RAISE agreed within 1 nat. This gap makes CSL unreliable as a means of model comparison.
2.7.5 Choosing an evaluation method

Based on the strengths and weaknesses of the evaluation methods described in section 2.7, we can now summarize the decision process that may go into the choice of a performance metric for a generative model. The considerations involved are visualized in figure 2.1.

First, if we wish to not only make comparisons between competing models, but also between sampling methods from a single model, we must choose a metric from second class of metrics: those which estimate the likelihood of a test set under a set of samples from the model. If the choice of sampling method is not applicable or not of interest and the model defines $P(x)$ (or it can be approximated via AIS and/or RAISE) we may choose to calculate the likelihood of the test set under the model directly.

When deciding between ISL or CSL, recall that CSL requires that the model define $P(x|h)$ and $P(h|x)$ and alternately samples between the two distributions. If not, only ISL is applicable. Note that even in cases where CSL is applicable, we may still choose to apply ISL. Despite it’s high variance, the ISL (or more specifically, the Parzen likelihood estimate) has been the most widely accepted method for comparison of the latest generation of generative models, such as generative adversarial nets (Goodfellow et al., 2014), generative stochastic networks (Bengio and Thibodeau-Laufer, 2013), and generative models based on nonequilibrium thermodynamics (Sohl-Dickstein et al., 2015). This makes ISL attractive, as new models can readily be compared to existing work.

However, a recently published pre-print by Theis et al. (2015) warn of the dangers
of relying on such quantitative measures when comparing generative models trained on high dimensional data. Not only do they find that log-likelihood and Parzen estimates (ISL with an isotropic Gaussian kernel) are independent of each other but that they are also independent of visual sample quality. In addition, Theis et al. (2015) also argue that Parzen window estimates are extremely inefficient as well as misleading. As such, the authors suggest that an evaluation criteria should be chosen with the intended task in mind. When the task is synthesis of images, for example, a visual inspection of generated samples is preferred to other numerical metrics.
Figure 2.1: Example of decision process for choosing an evaluation method for a generative model.
Chapter 3

Methodology

This thesis aims to investigate two related problems with respect to semi-supervised
generative models:

1. Does the incorporation of class information improve the quality of samples from a
generative model?

2. What architectures and/or sampling procedures are best suited for class condi-
tional sampling?

As mentioned in previous sections, there are numerous classes of generative models.
Under the umbrella of representation learning alone, this includes RBMs, generalized
denoising autoencoders, adversarial networks, variational autoencoders and more. Each
of these classes of model comes with its own set of hyperparameters and choices to be
made during training, making it infeasible to do an exhaustive search across each of the
model types. As such, the experimental investigations of this thesis will be limited in scope to generalized denoising autoencoders as generative models.

By limiting the scope in this manner, we can isolate variables in which we are most interested while leaving the model class fixed. These variables can be organized along two orthogonal axes: the distribution which we are attempting to model and the types of interactions between hidden units in the model.

Along the first axis, the models can be divided into two categories: those that model the joint distribution \( P(x, y) \) and those that model the conditional distribution \( P(x|y) \). As a control, we will also examine the type of model that most often comes to mind when considering generative models: those that model the data generating distribution \( P(x) \).

Along the second axis, we can further divide the models into those which allow only additive interaction between units and those which also incorporate multiplicative interactions.

<table>
<thead>
<tr>
<th>Distribution ( \mid ) Interaction</th>
<th>Additive</th>
<th>Multiplicative</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(x, y) )</td>
<td>Joint denoising</td>
<td>Joint gated</td>
</tr>
<tr>
<td>• ( x, y ) as input</td>
<td>autoencoder (jGAE)</td>
<td>autoencoder (jGAE)</td>
</tr>
<tr>
<td>• joint objective</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( P(x</td>
<td>y) )</td>
<td>Conditional denoising</td>
</tr>
<tr>
<td>• ( x, y ) as input</td>
<td>autoencoder (cDAE)</td>
<td>autoencoder (cGAE)</td>
</tr>
<tr>
<td>• ( x ) reconstruction objective</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( P(x) )</td>
<td>Denoising</td>
<td>Covariance</td>
</tr>
<tr>
<td>• ( x ) as input</td>
<td>autoencoder (DAE)</td>
<td>autoencoder (CovAE)</td>
</tr>
<tr>
<td>• ( x ) reconstruction objective</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Models under review, organized over two axes of variation.
3.1 Models under review

Next we describe in detail the specific models which will be examined. These are summarized in Table 3.1 along with the abbreviated naming conventions that will be used for the remainder of this thesis.

3.1.1 Conditional and joint autoencoders

Before delving into the models in Table 3.1, we will introduce two autoencoder variants: conditional and joint autoencoders. Just as in a traditional autoencoder, all models in Table 3.1 consist of a composition of two functions: an encoder function $f$ and a decoder function $g$. However, the encoder and/or decoder are defined as functions over both $x$ and $y$. Specifically, we examine the special case where $y$ is a class label corresponding to input $x$.

Where a denoising autoencoder reconstructs $x$ from a corrupted $\tilde{x}$, we define a conditional autoencoder as a model which reconstructs $x$ from a corrupted $\tilde{x}$ and some conditioning variable $y$. A joint autoencoder, on the other hand, corrupts both $x$ and $y$ and the training objective is based on a reconstruction of both inputs.

In the following sections, we describe the models listed in Table 3.1 and then describe sampling procedures based on the method for denoising autoencoders described in Section 2.6. The two columns of Table 3.1 divide the models in question into two classes: additive and multiplicative. This refers to the parametrization of the encoder and decoder functions the model and, as a result, the means by which the class label is
incorporated into the model. In an additive model, the hidden layer biases become a function of $y$. With a multiplicative model, the model weights become a function of $y$ (see Figure 3.1).

### 3.2 Additive (modulated bias) autoencoder

The additive autoencoder defines the encoder as a function over $x$ and $y$ as

$$h = f(x, y) = \sigma^{(enc)}(Wx + Vy + b)$$ (3.1)
where \(x \in \mathbb{R}^{n_X}, y \in \mathbb{R}^{n_Y}\) are the inputs, \(W \in \mathbb{R}^{n_H \times n_X}, V \in \mathbb{R}^{n_H \times n_Y}\) are the weight matrices, \(b_h \in \mathbb{R}^{n_H}\) is the hidden bias vector, \(n_H\) is the dimension of the hidden representation, and \(\sigma^{(enc)}\) is an activation function.

Equation 3.1 closely resembles the encoder function of a traditional autoencoder (Equation 2.5), but with the additional term \(V y\). Upon completion of training, the hidden bias in Equation 2.5 is fixed. In the additive model, this additional \(V y\) term has the effect of making the hidden layer bias a linear function of the conditioning input \(y\).

Formally, if we define \(b(y) = V y + b\), then eq. 3.1 becomes

\[
h = f(x, y) = \sigma^{(enc)}(W x + b(y))
\]

where \(b(y)\) is our modulated bias function.

The decoder is equivalent to that of a traditional denoising autoencoder:

\[
\hat{x} = g(h) = \sigma^{(dec)}(W'h + b')
\]

Note that the output bias \(b'\) could also be parameterized as a linear function of the input \(x\), however an examination of this variant is beyond the scope of this thesis.

The model bares architectural similarities to the conditional restricted Boltzmann machine (CRBM) (Taylor et al., 2006). However, the additive autoencoder is deterministic, where the CRBM is probabilistic.
3.3 Multiplicative (modulated weight) autoencoder

Next, we describe an autoencoder in which the model weights are a function of the conditioning input $y$. These models, as described in Memisevic (2013), are known as gated autoencoders (GAEs) or relational autoencoders. They are an extension of the autoencoder framework which learn relations on input-output pairs $x \in \mathbb{R}^{n_X}$ given $y \in \mathbb{R}^{n_Y}$ (Memisevic, 2013). Instead of defining a fixed weight matrix $W \in \mathbb{R}^{n_H \times n_X}$, the GAE learns a function $w(y)$ where the model weights are modulated by the value of the conditioning variable $y$. In order to remain consistent with current literature, such models will be referred to as GAEs for the remainder of this thesis.

The naive implementation involves constructing a weight tensor $W \in \mathbb{R}^{n_H \times n_X \times n_Y}$ and defining

$$w_{ij}(y) = \sum_{k=1}^{n_Y} W_{ijk}y_k$$

(3.4)

where the subscripts indicate indexing. Under this model, the encoder in Equation 2.5 becomes

$$h = f(x, y) = \sigma^{(enc)}(w(y)x + b)$$

(3.5)

However, this requires storing and learning $(n_X \times n_Y \times n_H)$ model parameters. In practice, this is infeasible for all but the smallest models as the number of weights is cubic in the number of units (assuming $n_X$, $n_Y$ and $n_H$ are roughly of the same dimensionality).

However, we can restrict the types of interactions allowed between $x$, $y$, and $h$ by first
projecting each onto factors $F^X, F^Y, F^H \in \mathbb{R}^{n_F}$ (Memisevic, 2013). In this manner, we can allow only element-wise interaction between the factors. Instead of a cubic number of weights needed in the naive method above, the factored model is parameterized via three weight matrices: $W^X \in \mathbb{R}^{n_X \times n_F}, W^Y \in \mathbb{R}^{n_Y \times n_F}$ and $W^H \in \mathbb{R}^{n_F \times n_H}$. The hidden representation under the factored model is defined as

$$h = f(x, y) = \sigma^{(\text{enc})}\left((W^H)^T (W^X x \odot W^Y y) + b^H\right) \quad (3.6)$$

where $\odot$ denotes element-wise multiplication and $b^H \in \mathbb{R}^{n_H}$ is the hidden bias.

Notice that the encoder is a function over both $x$ and $y$. Similarly, the decoder function will also be over two input variables, the choice of which is dependent on which of $x$ and $y$ are to be reconstructed. When learning a conditional model, one of the input variables will be held fixed. For example, the reconstruction of $x$ given $y$ is defined as

$$\hat{x} = g^x(h, y) = \sigma^{(\text{dec}^x)}\left((W^X)^T (W^H h \odot W^Y y) + b^X\right) \quad (3.7)$$

where $b^H \in \mathbb{R}^{n_X}$ is the output bias and $\sigma^{(\text{dec}^x)}$ is the output activation function. The $y$ decoder is symmetric to Equation 3.7:

$$\hat{y} = g^y(h, x) = \sigma^{(\text{dec}^y)}\left((W^Y)^T (W^H h \odot W^X x) + b^Y\right) \quad (3.8)$$

Equations 3.6, 3.7 and 3.8 describe a symmetric model where the encoder and de-
coder share the same set of weight matrices. However, this is not a hard requirement and various regimes of tied vs untied weights for gated models have been explored by Alain and Olivier (2013).

Like the traditional denoising autoencoder, the GAE is typically trained with a denoising criterion where the noise is applied to both \( x \) and \( y \) inputs. Yet where a denoising autoencoder learns features of the input, a GAE can learn relationships between its two inputs. For example, when trained on pairs of images where one is translated version of the other, the filters learned by the input factors resemble phase-shifted Fourier components (Memisevic, 2013). The GAE can also be trained such that the same input example is provided to both the \( x \) and \( y \) inputs. In this case, noise is independently applied to both copies of the input. This model can learn covariance between dimensions and is thus referred to as a covariance autoencoder (CovAE) (Memisevic, 2013).

The loss function is defined in much the same way as the classical autoencoder described above, using squared error on real-valued inputs and cross entropy loss where the input values are binary. As all operations are differentiable, the model can be trained via stochastic gradient descent, making use of the many optimization techniques that have been developed for neural network training.

### 3.4 Geometric interpretation

Recall that although the training data exists in some high dimensional space \( x \in \mathbb{R}^{n_x} \), we assume that it is concentrated in small regions of \( \mathbb{R}^{n_x} \), bounded by regions of
low density (with respect to our target distribution $P(x)$). These high density regions form a non-linear manifold, which the DAE training procedure attempts to capture.

Geometrically, both additive and multiplicative models can be seen as learning a conditional manifold of $x$ indexed by class label $y$. Like the DAE, if $x$ lies on this manifold, the application of noise pushes $\tilde{x}$ into the surrounding low density region, with the noise parameters defining how far $\tilde{x}$ lies from the original $x$. The model must learn to correct the noisy input by pushing it back towards the data manifold.

However, the additive or multiplicative models can make use of the class labels in order to learn a separate manifold for each class label. In the additive model, the modulated biases push a hidden unit’s pre-activation into different ranges depending on the class label. In effect, the model may learn to use a different subset of hidden units for each class. In the multiplicative model, the $F^Y$ factors, via their multiplicative interactions, can make use of the class label to scale the features learned by the $F^X$ factors.

In either case, these conditional models are akin to learning a DAE for each class, yet the models can make use of cross-class structure by sharing weights between each of the class-specific models. For example, features representing the stem of a 1, 4, or 7 may be shared between classes and those corresponding to the curved lower-right region of a 3, 6, 5 or 8 by other classes.
3.5 Class-conditional Sampling

The sampling procedure proposed by Bengio et al. (2013) for classical denoising autoencoders can also be applied to conditional autoencoders. Here, we define the true data distribution as the conditional distribution $P(x|y)$ where $x \in \mathbb{R}^n$ is an input data point and $y \in \mathbb{R}^m$ is the associated class label in a ‘one-hot’ encoding.

GAE training typically applies noise to both $x$ and $y$. Similarly, in training the conditional GAE and DAE we could also apply noise to both the $x$ and $y$ inputs. However, in generating class conditional samples we would like the model to be able to make use of a reliable class label during the sampling procedure. Thus, under conditional training we will apply the corruption procedure to $x$ only. In this manner, we hope to encourage the model to make use of any information it can from the $y$ input.

Thus, the corruption distribution is the same as in the DAE framework, namely the noise procedure draws samples from $C(\tilde{x}|x)$, a fixed distribution of our choosing. When training under a loss function interpretable as log-likelihood, the GAE and conditional DAE learn the conditional distribution $P_\theta(x|\tilde{x}, y)$

Like the sampling procedure for DAEs, the Markov chain formed by alternating samples from $x_t \sim P_\theta(x|\tilde{x}_{t-1}, y)$ and $\tilde{x}_t \sim C(\tilde{x}|x)$ will generate samples from the true distribution $P(x|y)$. During training, we can also apply a class-conditional version of the walkback training algorithm to seek out and squash any spurious modes of our model.

Bengio et al. (2013) provide a proof of the following theorem (Bengio et al., 2013,
Theorem 3.1. If $P_\theta(x|\tilde{x})$ is a consistent estimator of the true conditional distribution $P(x|\tilde{x})$ and $T_n$ defines an ergodic Markov chain, then as the number of examples $n \to \infty$, the asymptotic distribution $\pi_n(x)$ of the generated samples converges to the data-generating distribution $P(x)$.

For the conditional case of the conditional DAE or GAE the same arguments hold, where each of the following substitutions are made: $P_\theta(x|\tilde{x})$ with $P_\theta(x|\tilde{x},y)$, $P(x|\tilde{x})$ with $P(x|\tilde{x},y)$ and $P(x)$ with $P(x|y)$:

Theorem 3.2. If $P_\theta(x|\tilde{x},y)$ is a consistent estimator of the true conditional distribution $P(x|\tilde{x},y)$ and $T_n$ defines an ergodic Markov chain, then as the number of examples $n \to \infty$, the asymptotic distribution $\pi_n(x)$ of the generated samples converges to the data-generating distribution $P(x|y)$.

The proof of Theorem 3.1 provided by Bengio et al. (2013) can be trivially extended to 3.2 by modifying the conditionals in all relevant distributions. Bengio et al.’s arguments for consistency and ergoticity can also be adapted in the same manner.

3.6 Joint Sampling procedure

As with conditional sampling, we wish to sample the joint model in a manner as similar as possible to the training procedure. Recall that in joint training, noise is applied to both the $x$ and $y$ inputs. We can define the trivial joint distribution $C(\tilde{x},\tilde{y}|x,y)$
where the corruption procedures for $x$ and $y$ are independent noise distributions $\tilde{x} \sim C_x(\tilde{x}|x)$ and $\tilde{y} \sim C_y(\tilde{y}|y)$ for $x$ and $y$ respectively. The model is trained to approximate $\mathcal{P}(x, y|\tilde{x}, \tilde{y})$.

Samples from the true joint distribution can be generated in the same manner as in Section 3.5, where samples are alternately drawn from $x_t \sim P_\theta(x, y|x_{t-1}, \tilde{y}_{t-1})$ and $C(\tilde{x}, \tilde{y}|x_{t-1}, y_{t-1})$. Justification follows the same arguments as Section 3.5.

### 3.6.1 Logit noise

There has been much research regarding the application of noise to input data during training (Bishop, 1991, 1995; Vincent et al., 2008; Rifai et al., 2011). However, for training a joint DAE or GAE we require a method for adding noise to the class labels. Although there has been some work in training discriminative models with noisy labels (Natarajan et al., 2013), the work is not applicable to our setting. Natarajan et al. (2013) examine how modify on objective function in the presence of random noise on binary class labels (class flipping).

For our walkback training algorithm, we provide a noisy label and noisy image as input and the model learns to output both the denoised label and image. As such, we require a stochastic noise procedure $\text{noise}(y|\beta) = \hat{y}$ that satisfies the following criteria:

- The noising procedure is tunable with a single smoothing parameter $\beta$.

- When $\beta = 0$, the application of noise leaves $y$ unchanged (i.e. $\text{noise}(y|0) = y$)
• When $\beta \to \infty$, the distribution of noisy class label approaches a maximum entropy distribution (i.e. uniform).

• The noisy label $\hat{y}$ represents a valid probability distribution over the class labels (i.e. $\sum_i \hat{y}_i = 1$).

To satisfy these criteria, a logit noise procedure is proposed. The procedure is related to the softmax function commonly used as the output activation for neural networks trained for multi-class classification. The softmax function is defined as

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$ \hspace{1cm} (3.9)

We can ‘invert’ the softmax function by taking the logarithm of a one-hot encoded class label. Note that this is not a true inverse, as we have lost the scale information of the original softmax input. However, we can use this ‘pseudo-inverse’ to create a noise procedure satisfying the criterion above. In this vein, we propose our noise procedure as follows:

$$\text{logit\_noise}(y|\beta) = \text{softmax}(\log (y + u))$$ \hspace{1cm} (3.10)

where $u \sim \mathcal{U}(0, \beta)$. When $\beta = 0$, the noise procedure leaves $y$ unchanged as $u_i = 0$ for all $i$. As $\beta$ is increased, we generate noisy labels $\hat{y}$ from an increasingly smoothed distribution. Figure 3.2 plots the average entropy of a noisy one-hot class label (over 10 classes) as a function of $\beta$. 
Figure 3.2: The effect of the $\beta$ parameter on logit-noise on a 10 dimensional one-hot vector. (a) Simulated average entropy vs. $\beta$. The red horizontal line indicates the theoretical maximum of the uniform distribution. (b) Simulated entropy heat map with logit noise applied to a single one-hot vector with increasing $\beta$ from over the range $[0,2]$. Notice that the distribution smoothly transitions from a one-hot (zero-entropy) to a uniform (max-entropy) distribution.
3.7 Datasets

Experiments will be performed on three standard benchmark datasets: the MNIST database of handwritten digits (LeCun and Cortes, 1998), the CIFAR-10 dataset (Krizhevsky and Hinton, 2009), and the Street View House Numbers (SVHN) dataset (Netzer et al., 2011).

3.7.1 MNIST database of handwritten digits

The MNIST (mixed National Institute of Standards and Technology) database has historically been the standard benchmarking dataset for machine learning and image recognition research. The dataset comprises 10 classes, one for each digit from 0 to 9. As shown in Figure 3.3, each example is a 28x28 pixel 8-bit greyscale image. The dataset consists of 60,000 training examples and 10,000 test examples, and is made up of writing samples from approximately 250 individuals.

In its present form, MNIST was constructed from two earlier NIST datasets of handwritten digits, SD-3 (for training) and SD-1 (for testing). However, the training set was gathered solely from census employees, while the test contained samples from high-school students – making the learning task needlessly difficult. To address this issue, MNIST combined SD-3 and SD-1, such that both are equally represented in the training and test subsets. It should be noted that the training set and test sets were divided in such a way as to ensure that there is no overlap in writer identity between training and test sets (LeCun and Cortes, 1998). For model selection, we have further divided the
Figure 3.3: Examples from the MNIST database of handwritten digits. Each row contains examples from a single class. From top to bottom: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

training set into 50,000 used solely for model training and 10,000 for model validation.

### 3.7.2 CIFAR-10 database

The CIFAR-10 (Canadian Institute for Advanced Research) database (Krizhevsky and Hinton, 2009) contains natural images from 10 object classes. Each image is 32x32 pixels with 3 colour channels, resulting in 3,072 input dimensions. There are a total of 60,000 images divided into 50,000 training images and 10,000 test images. For the purpose of this thesis, 10,000 training images were set aside for validation purposes, leaving only 40,000 images for model training. In total, the dataset contains 6,000 images per class. The object classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and
Figure 3.4: Examples from the CIFAR-10 dataset. Each row contains examples from a single class. From top to bottom: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck.

truck. A sample of images from the training set are provided in Figure 3.4.

Both CIFAR-10 and its sister dataset CIFAR-100 were assembled from a much larger database of 80 million ‘tiny’ images (Torralba et al., 2008). Of these, a subset of 72,062 images were accompanied by a single non-abstract noun. Although the dataset is vast in scope, the lack of reliable labels made it unsuitable for training and benchmarking object recognition systems. To this end, Krizhevsky and Hinton (2009) created a small yet reliably labeled subset of the data.
3.7.3 Street View House Numbers (SVHN) dataset

Like MNIST, SVHN (Netzer et al., 2011) consists of images of from 10 digit classes. However, SVHN aims to investigate character recognition challenges in the context of real world images. Accordingly, the dataset contains colour images of digits extracted from house numbers in Google Street View images. Although the dataset is provided in two formats, this thesis will make use of only the cropped version of the dataset. Thus, any reference to SVHN for the remainder of this thesis will assume the cropped data. Like MNIST, SVHN contains images of digits from 0-9. However, each image in cropped SVHN is a 32x32 colour image.

There are a total of 73,257 training images and 10,000 test images. Again, for validation purposes the training images were further subdivided into 63,257 image training set and a 10,000 validation set. The dataset provides an extra 531,131 images which are not as challenging from the perspective of classification. However, these were not used for the current analysis. As seen in Figure 3.5, the dataset, although superficially similar to MNIST, offers a greater modeling challenge due to colour, background clutter, and variable image quality.

3.7.4 Data preprocessing

As is typical for the MNIST dataset, no preprocessing was performed beyond adjusting the scale of the values to be in the range $[0, 1]$. Both CIFAR-10 and SVHN, however,
Figure 3.5: Examples from the SVHN dataset. Each row contains examples from a single class. From top to bottom: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Although similar to MNIST, the SVHN dataset offers images with greater variation.
contain natural images. Smooth spatial gradients in such images result in neighbouring pixel values being highly correlated. As decorrelated input dimensions aid in training, it is common to perform principal components analysis (PCA) as preprocessing step and projecting the original input onto the principal components. It should be noted that prior to performing PCA, the data was zero-centred along each dimension- i.e. the mean was subtracted from each data dimension. Although it is typical to discard low variance PCA components as a means of reducing the dimensionality of the data, all principal components were retained for our experiments. In addition, the decorrelated dimensions were also whitened to ensure that variance across each dimension is equal (i.e. the covariance matrix of the data is the identity matrix $I$).

\section{3.8 Experimental settings}

Neural network training involves the selection of a number of hyperparameters. These include (but are not limited to) the learning rate, optimization algorithm, weight initialization strategy, number of hidden layers, number of hidden units per layer, activation functions, and regularization strategies. Searching for optimal settings of each hyperparameter quickly becomes infeasible as the search space grows increasingly large with each additional hyperparameter.

For this reason, we will limit our search by first choosing \textit{sensible} values for many of these hyperparameters and then holding them fixed for all models. These choices were made via informal exploration of various hyperparameter settings and based on
<table>
<thead>
<tr>
<th><strong>Hyperparameter</strong></th>
<th><strong>Value/Setting</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td># of non-linear hidden layers</td>
<td>1</td>
</tr>
<tr>
<td>Non-linear hidden activation</td>
<td>Hyperbolic tangent</td>
</tr>
<tr>
<td>Weight initialization</td>
<td>He weight initialization</td>
</tr>
<tr>
<td>Optimization algorithm</td>
<td>Adam or Nesterov accelerated Gradient</td>
</tr>
<tr>
<td>Learning rate</td>
<td>Cross validated based on reconstruction cost</td>
</tr>
<tr>
<td># of hidden units</td>
<td>All models constructed to have roughly the same # of free parameters</td>
</tr>
<tr>
<td>Regularization strategy</td>
<td>Denoising</td>
</tr>
<tr>
<td>Noise type</td>
<td>Salt and pepper for binary data ($p = 0.5$)</td>
</tr>
<tr>
<td></td>
<td>Gaussian for real valued data ($\mu = 0, \sigma = 1.0$)</td>
</tr>
<tr>
<td></td>
<td>Logit noise for class labels ($\beta = 0.2$)</td>
</tr>
<tr>
<td># Walkback steps</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3.2: Hyperparameter settings used for model training.

both qualitative inspection of generated samples and Parzen likelihood estimates on a validation set. These decisions are summarized in Table 3.2.

All model weights were initialized via the weight initialization scheme proposed by He et al. (2015). For the SGD optimization algorithm, experiments were performed with either Nesterov accelerated gradient (Sutskever et al., 2013) or Adam (Kingma and Ba, 2014). The best performing model was chosen via reconstruction cost on the validation set.

### 3.9 Evaluation Criterion

For comparison of the models enumerated in Table 3.1, we will use a single evaluation metric. Section 2.7 reviews the common criteria for evaluating generative models and considerations for their selection (Figure 2.1). As the models under review do not define
an unnormalized probability function, neither AIS nor RAISE are applicable. The models do, however, define both $P(x|h)$ and $P(h|x)$, and the model samples are generated via a Markov chain that alternates between the two distributions. As such, CSL estimates can be calculated.

However, for the purpose of this thesis we will employ the ISL estimator with a Gaussian distribution (i.e. a Parzen likelihood estimator) to compare samples from the model and a held out test set. This choice is motivated mainly by the wide spread adoption of this metric in evaluating generative models, allowing for the ready comparison of our results to those of the wider research community.

Recall that in utilizing the Parzen estimator, one must choose an appropriate value of the spreading parameter (i.e. the standard deviation of the isotropic Gaussian distribution). In keeping with the literature, this parameter will be chosen to maximize likelihood on a held out validation set. Example code for calculation of Parzen log-likelihood estimates was provided by Goodfellow et al. (2014).

All estimates provided were calculated on 10,000 consecutive samples from the model. Each Markov chain was initialized to the zero vector and the first 10 samples were discarded before the estimates were calculated. In the case of our conditional models, the samples were generated via 10 parallel Markov chains - one for each class label. The number of samples for each class was generated as a sample from a multinomial distribution with 10,000 trials. The parameters of the multinomial were fit via the statistics of the training set. Visual examples of samples from the each model are
provided in this chapter’s appendix.

3.10 Implementation

All experiments are implemented in the Python programming language using the Theano mathematics and optimization library (Bastien et al., 2012; Bergstra et al., 2010) and the Lasagne deep learning framework. Experiments were performed on a GPU compute cluster consisting on 10 servers. Each server has three Nvidia GTX Titan Black GPU video cards. Each card has 6GB of on-board memory and 2880 CUDA cores running at a base clock rate of 889MHz.
Chapter 4

Results

A quantitative and qualitative evaluation of each of the models considered is provided below. Model performance is evaluated on each dataset separately, beginning with MNIST (sec. 4.1), followed by CIFAR-10 (sec. 4.2) and SVHN (sec. 4.3). Sample images for the top 3 models will be provided for each dataset, as determined via Parzen likelihood estimates. In section 4.4 joint and conditional sampling procedures are compared, both in terms of Parzen likelihood estimates and visual quality. Finally in section 4.5, analogy making is explored as an indication of the model’s ability to disentangle factors of variation. Note that all experiments performed throughout the chapter make use of the ‘best’ model as found in sections 4.1, 4.2 and 4.3. Additionally, appendix A contains consecutive sample images for all trained models.
4.1 MNIST

4.1.1 Quantitative Results

Quantitative results for generative models trained on MNIST are provided in table 4.1, with all likelihood estimates provided in nats. For all Parzen log-likelihood calculations, a spreading parameter of $\sigma = 0.16681$ was chosen via performance on the validation set. Three models, the cDAE (273.48), cGAE (274.96) and the jDAE (275.50), outperform a comparatively sized DAE (218.82) by a significant margin. In addition to the comparatively sized models, a larger DAE was also trained containing a single hidden layer of 2000 tanh units. This model has roughly 8 times the parameters of the other models tested, yet samples from this model produce a Parzen estimate of only 243.69.

The two remaining models, the jGAE and the CovAE failed to converge to a good solution during training. As such, the Markov chain sampling procedure failed to generate any recognizable samples. Although training failed to converge to an adequate solution, it is possible that there exist hyperparameter settings outside of the current search space which would lead to stable training dynamics. This is left for future investigation.

Additionally, a comparison with other generative models from the literature is provided. It should be noted that not all details were available for the models in question, so there may be some discrepancy in how the Parzen scores were calculated by the other researchers (e.g. setting of the spreading parameter, binarized samples/test set,
Therefore, hard comparisons may not be applicable. As an additional baseline, Parzen scores were computed on 10,000 examples from the training set, resulting in an estimate of $243.69 \pm 2.01$. Interestingly, this score is on par with the DAE with 2000 hidden units and lower than our top performing models.

### 4.1.2 Qualitative Results

Figure 4.1 presents samples from the three top models. Note that the samples provided are non-consecutive in order to present samples from the entire length of the Markov chain. For an indication of mixing properties of the sampling procedures, please refer to the images in appendix A. In order to verify that the model is not merely memorizing examples from the test set, the rightmost column of each figure contains the nearest neighbour (via Euclidean distance) from the training set to the sample immediately to its left. Most samples are not as well defined as those from the training set, due likely to the nature of the squared error training objective. It is interesting to note that, on occasion, a sample’s nearest neighbour hails from a different class, indicating that the models may indeed be sharing information between classes.

With the conditional sampling procedure, each model is able to explore the manifold of a given class with each set of samples showing much intra-class variation. In addition, the models rarely stray from a given class, especially with the conditional training objective. However, this is somewhat less evident with the joint training objective. Recall that the joint objective applies noise to both the $x$ and the $y$ inputs and the model is
trained to reconstruct each variable conditional on the other. Even with a conditional sampling procedure (keeping the class label fixed during sampling), the jointly trained model appears to be more prone to wander into nearby classes. The noisy labels during training act as a regularizer, encouraging the model rely less on the class label.

Although samples from the joint GAE model outperformed other models in terms of Parzen likelihood estimates, the generated samples are of lower visual quality. Compared to the two models trained with the conditional objective, the jointly trained model generates more fuzzy amorphous samples which appear to lie outside the true data generating distribution. This suggests that the Parzen likelihood estimates may not be a good indication of visual sample quality.

Finally, figure A.2 provides samples from small and large DAE models. Again, the samples are non-consecutive with the rightmost column containing a nearest neighbour from the training set. The samples from the non-conditional models contain many spurious examples compared to the conditionally and jointly trained models from figure 4.1. This appears to support our hypothesis, that by injecting class information into the generative process the models are able to more easily capture the input data distribution. The DAE with 258 hidden units contains a comparable number of free parameters as the conditional and joint models, yet the visual quality of samples from models with access to class information is arguably superior.
Figure 4.1: Non-consecutive samples from the three top MNIST models using conditional sampling for classes 2, 4, 6, and 8. Top row is a cDAE, middle row is a cGAE with a conditional objective and the bottom row is a jGAE. All samples were generated via a conditional sampling procedure. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample.
Figure 4.2: Samples from two separate generative DAEs trained on MNIST with a reconstruction objective. The model on the left has 258 tanh hidden units while the model on the right has 2000. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample.

<table>
<thead>
<tr>
<th>Model</th>
<th>objective</th>
<th># hidden</th>
<th># params</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAE</td>
<td>$P(x</td>
<td>\tilde{x})$</td>
<td>258</td>
<td>203,314</td>
</tr>
<tr>
<td>DAE</td>
<td>$P(x</td>
<td>x)$</td>
<td>2000</td>
<td>1,570,784</td>
</tr>
<tr>
<td>CovAE</td>
<td>$P(x</td>
<td>x)$</td>
<td>258</td>
<td>269,878</td>
</tr>
<tr>
<td>cDAE</td>
<td>$P(x</td>
<td>x, y)$</td>
<td>256</td>
<td>203,520</td>
</tr>
<tr>
<td>cGAE</td>
<td>$P(x</td>
<td>x, y)$</td>
<td>204</td>
<td>203,796</td>
</tr>
<tr>
<td>jDAE</td>
<td>$P(x, y</td>
<td>x, \tilde{y})$</td>
<td>256</td>
<td>203,520</td>
</tr>
<tr>
<td>jGAE</td>
<td>$P(x, y</td>
<td>x, \tilde{y})$</td>
<td>204</td>
<td>203,796</td>
</tr>
<tr>
<td>Adversarial net</td>
<td>-</td>
<td>100/1200</td>
<td>2503984</td>
<td>225 ± 2.00</td>
</tr>
<tr>
<td>GSN</td>
<td>$P(x</td>
<td>\tilde{x})$</td>
<td>1200/1200</td>
<td>3800000</td>
</tr>
<tr>
<td>Diffusion</td>
<td>-</td>
<td>not provided</td>
<td>not provided</td>
<td>219.73 ± 1.87</td>
</tr>
</tbody>
</table>

10k training examples:  - - - 244.57 ± 2.11

Table 4.1: MNIST Parzen log-likelihood results in nats comparing 10,000 model samples to the test set. All samples were generated with a conditional sampling procedure. Results (1) are reported from Goodfellow et al. (2014), (2) are reported from Bengio and Thibodeau-Laufer (2013), (3) are from Sohl-Dickstein et al. (2015).
4.2 CIFAR-10

4.2.1 Quantitative results

Parzen log-likelihood estimates for models trained on CIFAR-10 are provided in table 4.2. Samples from the cGAE received the highest Parzen log-likelihood estimates (870.58), followed by the non-conditional DAE obtaining the next highest score (812.97), the condition cDAE (786.61), the jDAE (747.33) and the joint GAE (667.76). Again, training of the CovAE was faced with stability issues. As such, the sampling procedure was unable to generate samples and calculation of a Parzen estimate was not possible.

No Parzen estimates for external models are provided for comparison. Although generative models have been trained on this dataset and visual samples have been published (Goodfellow et al., 2014; Sohl-Dickstein et al., 2015), Parzen log-likelihoods were not provided by the authors. However, a Parzen estimate for 10,000 training examples is given in table 4.2 (884.01).

<table>
<thead>
<tr>
<th>Model</th>
<th>objective</th>
<th># hidden</th>
<th># params</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAE</td>
<td>( P(x</td>
<td>\tilde{x}) )</td>
<td>1026</td>
<td>3,155,970</td>
</tr>
<tr>
<td>CovAE</td>
<td>( P(x</td>
<td>\tilde{x}) )</td>
<td>1026</td>
<td>4,208,646</td>
</tr>
<tr>
<td>cDAE</td>
<td>( P(x</td>
<td>\tilde{x}, y) )</td>
<td>1024</td>
<td>3,156,992</td>
</tr>
<tr>
<td>cGAE</td>
<td>( P(x</td>
<td>\tilde{x}, y) )</td>
<td>811</td>
<td>3,158,034</td>
</tr>
<tr>
<td>jDAE</td>
<td>( P(x, y</td>
<td>\tilde{x}, \tilde{y}) )</td>
<td>1024</td>
<td>3,156,992</td>
</tr>
<tr>
<td>jGAE</td>
<td>( P(x, y</td>
<td>\tilde{x}, \tilde{y}) )</td>
<td>811</td>
<td>3,158,034</td>
</tr>
<tr>
<td>10k training examples</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>884.01 ± 7.74</td>
</tr>
</tbody>
</table>

Table 4.2: CIFAR-10 Parzen log-likelihood results in nats comparing 10,000 model samples to the test set.
4.2.2 Qualitative results

Samples generated with the cGAE, cDAE, and DAE are provided in figures 4.4, 4.3 and 4.5 respectively. The dataset provides an extremely challenging modeling task, as the training set is both highly variable and relatively small in size. Samples from both the cDAE (figure 4.3) and the DAE (figure 4.5) indicate that models are able to capture the general colour palette of natural images, although the samples do tend to be less saturated than those in the training set (figure 3.4). However, the conditional sampling procedure of the cDAE fails to capture the inter-class variability. Samples from the cDAE are nearly indistinguishable from those of the DAE despite the fact that the cDAE had access to the class information during training while the DAE did not. All images are extremely similar in their (lack of) fidelity, containing mostly vague patches of colour with no discernible shape. This suggests that for high dimensional data of high variability, additively incorporating the class label as a modulated bias is an ineffective means of integrating class information.

This is in stark contrast with samples from the cGAE (figure 4.4). The conditional sampling procedure produces distinct styles of samples when conditioning on each class with respect to both colour palette and content. Conditioning on the airplane class produces images with a smooth blue background with straight dark lines in the foreground, while frog samples are predominantly earth toned with a round shape in the centre.

Although the samples are by no means of the same quality as those in the training set, they do contain recognizable images. Certain classes are easier to model than others,
for example the horse and automobile samples are readily recognizable. Others, such as the cat and deer classes are less defined. This is likely due to the variability of the images in the training set. Horses and cars are more likely to be photographed from a limited range of viewpoints, making the modeling task much easier. This can be seen in the small sample of training images in figure 3.4. Many of the images from the horse class are full profiles while images from the car class are typically 45 degree frontal views.

Despite the higher quality samples, the model does tend to generate samples from limited regions of the input space. Even from the small number of samples provided in figure 4.4, there are a number of duplicate images in the test set nearest neighbour column, indicating that the model may be preferentially sampling from a few modes and/or failing to fully capture the full distribution.

As with models trained on MNIST, the Parzen likelihood estimate are a poor measure of visual quality. Although samples generated via the cGAE were also the highest scoring with respect to the Parzen likelihood, the samples were a mere 14 nats shy of the training set (870.58 vs 884.01) and the standard error of the two measures are in fact overlapping. In terms of visual quality, however, the samples are of far lower fidelity than those in the test set.
Figure 4.3: Non-consecutive samples from a cDAE trained on CIFAR-10 using a conditional sampling procedure. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample.
Figure 4.4: Non-consecutive samples from a cGAE trained on CIFAR-10 using a conditional sampling procedure. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample.
Figure 4.5: Samples from a generative DAE trained on CIFAR-10 with a reconstruction objective. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample.
4.3 SVHN

4.3.1 Quantitative results

Although of equivalent dimensionality to the CIFAR-10 dataset, the SVHN dataset provides a middle ground between MNIST and CIFAR-10 in terms of modeling difficulty. Parzen log-likelihood estimates are provided in table 4.3. Surprisingly, the highest Parzen likelihood was measured against samples from a DAE - a model which does not make use of class label information (2047.42). Samples from this model were roughly 100 nats higher than the next best model, the cDAE (1941.27), followed by the cGAE (1866.68) and the jGAE. Again, the CovAE exhibited training stability issues.

<table>
<thead>
<tr>
<th>Model</th>
<th>objective</th>
<th># hidden</th>
<th># params</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAE</td>
<td>( P(x</td>
<td>\tilde{x}) )</td>
<td>1026</td>
<td>3,155,970</td>
</tr>
<tr>
<td>CovAE</td>
<td>( P(x</td>
<td>\tilde{x}) )</td>
<td>1026</td>
<td>4,208,646</td>
</tr>
<tr>
<td>cDAE</td>
<td>( P(x</td>
<td>\tilde{x}, y) )</td>
<td>1024</td>
<td>3,156,992</td>
</tr>
<tr>
<td>cGAE</td>
<td>( P(x</td>
<td>\tilde{x}, y) )</td>
<td>811</td>
<td>3,158,034</td>
</tr>
<tr>
<td>jDAE</td>
<td>( P(x, y</td>
<td>\tilde{x}, \tilde{y}) )</td>
<td>1024</td>
<td>3,156,992</td>
</tr>
<tr>
<td>jGAE</td>
<td>( P(x, y</td>
<td>\tilde{x}, \tilde{y}) )</td>
<td>811</td>
<td>3,158,034</td>
</tr>
</tbody>
</table>

Table 4.3: SVHN Parzen log-likelihood results in nats comparing 10,000 samples to a held out test set. All samples were generated via the conditional sampling procedure.

4.3.2 Qualitative results

Visualization of samples from the top three SVHN models are provided in figures 4.6 (cDAE), 4.7 (cGAE), and 4.8 (DAE). From a qualitative perspective, an examination
of the SVHN samples yields similar results as those from models trained on CIFAR-10. As with CIFAR-10, the samples overall tend to have a less saturated colour palette than those of the training set. Also, both the cDAE (fig. 4.6) and the DAE (fig. 4.8) generate similar low quality samples. Although the samples contain an occasional recognizable digit, overall they consist of undefined patches of colour.

Again, the cGAE model is able to successfully capture the class conditional data distributions while the cDAE is not. Not only do samples in figure 4.7 rarely stray from a given class, but they also contain very few spurious samples. In addition, when comparing generated samples to their nearest neighbours from the training set, the model appears to not simply be memorizing training examples but generating novel examples from the learned space.

Unlike CIFAR-10, nearly all samples generated via the cGAE are readily recognizable with respect to their class membership. The overall quality of samples is high and they appear to exhibit a fair amount of intra-class variation. Despite their higher visual quality, the cGAE samples received a lower Parzen likelihood estimate than samples from both the DAE and cDAE. Again, this highlights the unreliability of the Parzen metric for assessing visual sample quality.
Figure 4.6: Non-consecutive samples from a cDAE trained on SVHN using a conditional sampling procedure. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample.
Figure 4.7: Non-consecutive samples from a cGAE trained on SVHN using a conditional sampling procedure. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample.
Figure 4.8: Samples from a generative DAE trained on SVHN with a reconstruction objective. The rightmost column of each figure contains the nearest neighbour from the training set of the preceding sample.
4.4 Class conditional vs. joint samples

Next, an investigation the relative effects of conditional vs joint sampling is provided. Recall that with condition sampling, the class label \( y \) is held fixed while the Markov chain of noising and reconstruction is traversed with respect to the \( x \) input only. In joint sampling, both the \( x \) and the \( y \) are allowed to wander. A single step of the joint Markov chain consists of corrupting both \( x \) and \( y \) to obtain noisy \( \tilde{x} \) and \( \tilde{y} \). The model then reconstructs \( x \) conditional \( \tilde{y} \) and \( y \) conditional on \( \tilde{x} \). The joint sampling procedure only applies to the jDAE and jGAE, as only the joint models are trained to reconstruct the \( y \) input.

Parzen likelihood estimates for all joint models on all datasets are shown in table 4.4. On MNIST, the joint sampling produces lower likelihood samples than conditional sampling. On CIFAR-10 and SVHN, the joint sampling receive higher likelihood estimates with the jDAE. However, the trend is reversed with the jGAE on the same datasets with conditional sampling generating samples with higher Parzen likelihood estimates.

Figure 4.9 contains samples from a jDAE trained on MNIST. The samples from both sampling procedures appear to be of comparable visual quality. As expected, however, the joint sampling procedure is more prone to stray from a given class. Samples for models trained on CIFAR-10 and SVHN are presented in figures 4.10 and 4.11 respectively. Compared to the cGAE samples provided in the previous sections, the visual quality of the samples from jDAE and jGAE is poor. The jDAE models fail to generate recognizable samples while the jGAE models produce samples with little variability. This may be due
to the balance between generative and discriminative costs used to train the model. In our experiments, both were given equal weight. If the discriminative cost (y reconstruction) dominates, the model may begin to overfit before it is able to fully capture the x distribution. In future work, this hypothesis may be explored by adjusting the relative balance of the two costs.

### 4.5 Disentangling factors of variation

High dimensional data such as natural images exhibit many so-called factors of variation. Not only can images vary as a function of content, but also with respect to 3D orientation, translation, lighting, etc. All these factors can have a significant effect on
Figure 4.9: Comparison of conditional vs joint sampling procedures for a jDAE trained on MNIST. Samples are consecutive with the $x$ input of the Markov chain initialized to the zero vector and the $y$ input initialized to a single class for each row. In the case of joint sampling, the $y$ corruption and reconstruction allows the samples for a given row to wander between classes.
Figure 4.10: Comparison of conditional vs joint sampling procedures for a jDAE and jGAE trained on CIFAR-10. Samples are consecutive with the Markov chain initialized to the zero vector.
Figure 4.11: Comparison of conditional vs joint sampling procedures for a jDAE and jGAE trained on SVHN. Samples are consecutive with the Markov chain initialized to the zero vector.
the resulting pixel values of a given image. For object recognition, it would be beneficial to ignore certain factors (e.g. lighting, orientation, background) while focusing only on others (e.g. content). Unfortunately these factors interact in incredibly complex and non-linear ways, making disentangling of such factors difficult.

Despite these difficulties, a handful of researchers have attempted addressing this question of disentangling – mostly under the umbrella of semi-supervised learning. Kingma et al. (2014) incorporate class information into a variational autoencoder by treating the class label vector $y$ as a latent variable of their model and adding a discriminative (i.e. prediction) term to the cost function. A similar approach was applied by Cheung et al. (2014), where the hidden space of a traditional autoencoder is partitioned into a set of free variables and an additional set of units corresponding to the class labels. Finally, Reed et al. (2014) train a higher-order (i.e. gated, multiplicative) RBM architecture to incorporate class label information into the learned representation.

The present work can also be interpreted as a means of disentangling factors of variation. One such factor is encoded in the class label. Recall that the conditional sampling procedure involves a Markov chain which stochastically explores the manifold of a given class. The directions on the manifold in which the model is freely available to explore can be interpreted as encoding different factors. In order to generate high quality samples, the model must learn which factors are free to vary (e.g. colour, lighting, style) and those which may not (i.e. overall shape, class identity).

In order to test this hypothesis, each model’s ability to make analogies is explored.
Some of the earliest work created visual analogies was explored by Memisevic and Hinton (2007), where a gated model was able to successfully recreate an artistic image filter and apply it to unseen test images. In the current problem setting, we can attempt to reconstruct a specific example from the test set while replacing its class label with that of an alternate class, thus creating analogies of given test example. A similar approach was employed in the works of Kingma et al. (2014), Cheung et al. (2014), and Reed et al. (2014).

Under the models in question, given an image and class label which do not correspond, the model will attempt to ‘denoise’ the image by pushing it closer to the manifold corresponding to the new class label. It is hypothesized that this will correspond to a point which is close in visual similarity to the original image, with only the class content affected. Unlike the sampling procedure, noise is not applied before reconstructing, the goal is not exploration of the space of the manifold.

Figure 4.12 shows analogies created by the cDAE and cGAE models trained on MNIST. Although both models were able to generate class conditional samples of similar quality, the cDAE is unable to ‘correct’ the test examples in order to create analogies from different classes. The cGAE fairs better, projecting the test examples onto alternate classes while retaining some of the style information. Unfortunately, the analogies are blurry and often are not entirely legible.

However, the denoising need not stop at a single reconstruction. The recurrent nature of the walkback training procedure (and the Markov chain sampling algorithm) suggests
Figure 4.12: Analogies created by the cDAE and cGAE. A test example is provided as a seed (leftmost column), and the image is reconstructed with an alternate class label.

A simple means of cleaning up these analogies by repeatedly applying the denoising model to the previous reconstruction. Again, additional noise is not added, allowing the model instead to clean up the noise left over by the previous reconstruction.

If the procedure is stable, this will converge to point attractors on the class conditional manifold. Figure 4.13 shows that this is indeed the case for the cGAE model. Each subsequent reconstruction moves toward an attractor on the conditional manifold. Occasionally this attractor ends up being a spurious mode (see class 9 analogy in row 4). For the most part, however, the reconstructions are of high visual quality and do indeed retain some style information. For example, all digits in top row are nearly perfectly vertical, the second row has a distinct right slant, and all digits in the third row use a heavier stroke.
Figure 4.13: Multi-step analogies created by a cGAE trained on MNIST. With each step, the model approaches finds a point attractor on the conditional manifold closest to the original test image (leftmost column).
The same technique can be applied to both CIFAR-10 and SVHN, although for CIFAR-10 the intuition is less clear. There is less of a notion of an image of a horse being of the same style as an airplane. Regardless, 3-step analogies of both CIFAR-10 and SVHN are presented in figure 4.14. Analogies of CIFAR-10 images occasionally retain some of the colour information of the test input, but the quality of the samples in general make further interpretation fruitless.

SVHN analogies are indeed projected onto the class conditional manifolds, however much of the style information is lost. Reconstruction rarely resemble the original test input, and often the color palettes are inverted for images in a single row (i.e. dark foreground/light background vs. light foreground/dark background).

Regardless, the success of the cGAE analogies on MNIST warrants further investigation of the model as a means of disentangling factors of variation, despite the fact that the model was not explicitly trained to do so. Although the analogies are not of the same visual quality as those in other work, the cGAE models tested are relatively small (with respect to both depth and the number of model parameters) compared to those used by Kingma et al. (2014), Cheung et al. (2014) and Reed et al. (2014). Both larger and deeper models, as well as modifications to the training or testing procedure, may improve the analogies.
Figure 4.14: Multi-step analogies created by cGAE trained on CIFAR-10 and SVHN. A test example is provided as a seed (leftmost column), and the image is reconstructed with an alternate class label.
Chapter 5

Conclusions and Future Work

There has been an incredible amount of momentum in both research and industry regarding the training and application of deep discriminative models. Generative models of high dimensional data, on the other hand, have been slower to gain traction. Although traditional methods of training generative models for use in image synthesis are purely unsupervised, this thesis has examined how class label information can help guide both the training and sampling procedures of a single class of generative models: the generalized denoising autoencoder. Specifically, it is shown that class labels can be used during training to greatly improve the visual fidelity of generated samples.

While Chapter 1 painted the research landscape and problem description in broad strokes, Chapter 2 provided a more detailed look at both the current literature and concepts relevant to the task at hand. An overview of supervised, unsupervised and semi-supervised training regimes was presented before providing mathematical defini-
tions for feed forward neural networks and their training procedure. Next, autoencoders were introduced, including their interpretation as generative models. The chapter was concluded with an exploration of various methods of evaluating generative models.

Experimental methods were outlined in Chapter 3. Two methods of incorporating class label information into the denoising autoencoder are described. The sampling procedure first introduced by Bengio et al. (2013) was extended to both a conditional and joint setting. The chapter was concluded with a description of the datasets with which the experiments were performed, as well as the hyperparameter search space and evaluation criteria.

Chapter 4 presented both quantitative and qualitative results for all three datasets. From a visual examination of the generated samples, the results confirmed those of Theis et al. (2015): that Parzen likelihood estimates do not always provide a good indication of sample visual quality. For example, of all models trained on SVHN, the highest likelihood estimate is given to samples from the DAE model. From a visual inspection, the samples contain mostly diffuse colour patches with little in the way of recognizable digits. The conditional GAE, however, produced immediately recognizable images from each digit class. Additionally, the samples appear to mix well between modes of the distribution, generating images with a range of foreground and background color combinations. Despite their greater visual fidelity, the cGAE samples received a Parzen estimate nearly 200 nats lower than samples from the DAE.

With regard to the main aims of this thesis, class label information is shown to greatly
improve sample quality – especially when the dimensionality of the data is high. Semi-supervised training assumes that each class lies on a distinct manifold in the input space. Instead of applying this assumption to a discriminative task, it was hypothesized that the learning task could be greatly simplified by providing the class information during training. By providing class information, the model may be encouraged to focus on learning each class manifold separately, carving the space into more manageable slices. From the experimental results presented, this does appear to be the case.

On all datasets, the visual quality of samples is greatly improved when class information is made available at training time. In addition, choosing an appropriate model architecture is essential to make use of this information – especially when the dimensionality is high. For MNIST, both the multiplicative and additive models generated samples of similar quality. On both CIFAR-10 and SVHN, however, the additive models failed to generate samples of even passable quality. Multiplicative models of comparable size are indeed able to carve the data distribution into class conditional slices. Class conditional sampling rarely generates samples from the wrong class and the overall quality of samples is much improved over the other models.

Finally, the multiplicative models also showed promise in making visual analogies (sec. 4.5). Although the model was not explicitly trained in this capacity, the denoising criteria is well suited for this task. Additionally, the recurrent nature of the walkback training procedure provides a natural means of ‘cleaning up’ analogy predictions. Although the multiplicative model is able to generate successful analogies of MNIST digits,
it fails to do so for both CIFAR-10 or SVHN.

5.1 Future Work

The current thesis is limited in scope to a single class of generative model. The insights gained can (and should) be applied to a wider range of models such as generative stochastic networks, generative adversarial networks, and variational autoencoders. In addition, the difficulties in training the CovAE as a generative model should be explored via an expansion of the hyperparameter search space and an analysis of the stability issues encountered.

From a high level, the present work explores methods of sampling from conditional distributions. The current scope, however, focuses solely on the special case where the conditioning variable is a class label. Future work may explore cases where the conditioning variable is from a far richer distribution. For example, the work may be extended to models which sample a frame of video conditional on the previous frame or sampling an image conditional on a text-based description.

The success of the cGAE in creating visual analogies on MNIST warrants further investigation. The models under consideration are relatively small in terms of both size (i.e. number of parameters) and depth (i.e. number of layers). Larger and deeper models may improve results on SVHN. It should be noted that the present work uses the more challenging subset of SVHN, containing only 73,257 examples. An additional dataset of 531,131 less challenging examples is also provided which may improve performance on
this task.

Finally, exploring new quantitative measures of generative models should be prioritized. Parzen log-likelihood estimates have been the predominant metric for the comparison of such models. As argued by Theis et al. (2015) and supported by the experiments presented, Parzen scores are a poor proxy for visual sample quality. As generative models gain in popularity, alternate means of evaluating generative models will be essential for comparing new and existing approaches.
References


Appendix A

Model samples

A.1 MNIST Samples
Figure A.1: Samples from a traditional DAE with 258 tanh hidden units trained on MNIST
Figure A.2: Samples from a traditional DAE with 2000 tanh hidden units trained on MNIST
Figure A.3: Samples from a GAE trained on MNIST
Figure A.4: Consecutive samples from additive DAE with conditional objective function.
Figure A.5: Consecutive samples from multiplicative GAE with conditional objective function.
Figure A.6: Consecutive samples from additive DAE with joint objective function and conditional sampling procedure.
Figure A.7: Consecutive samples from an additive DAE with joint objective function and joint sampling procedure.
Figure A.8: Consecutive samples from jGAE and conditional sampling procedure.
Figure A.9: Consecutive samples from an jDAE and joint sampling procedure.
A.2 CIFAR10 Samples

Figure A.10: Samples from a traditional DAE with 1026 tanh hidden units trained on CIFAR
Figure A.11: Samples from a traditional GAE with 1026 factors and 1026 tanh hidden units trained on CIFAR
Figure A.12: Consecutive samples from additive DAE with conditional objective function.
Figure A.13: Consecutive samples from multiplicative GAE with conditional objective function.
Figure A.14: Consecutive samples from an additive DAE with joint objective function and conditional sampling procedure.
Figure A.15: Consecutive samples from an additive DAE with joint objective function and joint sampling procedure.
Figure A.16: Consecutive samples from a multiplicative GAE with joint objective function and conditional sampling procedure.
Figure A.17: Consecutive samples from a multiplicative DAE with joint objective function and joint sampling procedure.
A.3 SVHN Samples

Figure A.18: Samples from a traditional DAE with 1026 tanh hidden units trained on SVHN
Figure A.19: Samples from a traditional GAE with 1026 factors and 1026 tanh hidden units trained on SVHN
Figure A.20: Consecutive samples from additive DAE with conditional objective function.
Figure A.21: Consecutive samples from multiplicative GAE with conditional objective function.
Figure A.22: Consecutive samples from an additive DAE with joint objective function and conditional sampling procedure.
Figure A.23: Consecutive samples from an additive DAE with joint objective function and joint sampling procedure.
Figure A.24: Consecutive samples from a multiplicative GAE with joint objective function and conditional sampling procedure.
Figure A.25: Consecutive samples from a multiplicative GAE with joint objective function and joint sampling procedure.