Spring 5-18-2018

Multi-Label Latent Spaces with Semi-Supervised Deep Generative Models

Rastin Rastgoufard
University of New Orleans, rrastgou@uno.edu

Multi-Label Latent Spaces with Semi-Supervised Deep Generative Models

A Dissertation

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Engineering and Applied Science

by

Rastin Rastgoufard

M.Sc. University of New Orleans, 2012
B.Sc. Tulane University, 2008

May, 2018

Acknowledgements

This was a long but exciting journey over the course of six years. I want to sincerely thank my advisor, AbdulRahman Alsamman, for adventuring with me throughout the entire process. We did not have a map, but we found our bearings nevertheless. I am grateful for all the times he grounded me in reality while still encouraging me to dream.

Many professors guided me along my path and helped me along the way. My committee – Vesselin Jilkov, Huimin Chen, Martin Guillot, and Vassil Roussev – gave me valuable feedback which strengthened this dissertation. The Power and Energy Research Lab including Ittiphong Leevongwat and Ebrahim Amiri supported me and gave me an escape plan for when things got tough.

My parents, Hilda and Parviz Rastgoufard, created a safe and loving home that allowed me to play and explore to the edges of my creative imagination.

Thank you all.

Contents

List of Figures

List of Tables

Abstract

Expert labeling, tagging, and assessment are far more costly than the processes of collecting raw data. Generative modeling is a very powerful tool to tackle this real-world problem. It is shown here how these models can be used to allow for semi-supervised learning that performs very well in label-deficient conditions.

The foundation for the work in this dissertation is built upon visualizing generative models' latent spaces to gain deeper understanding of data, analyze faults, and propose solutions. A number of novel ideas and approaches are presented to improve single-label classification. This dissertation's main focus is on extending semi-supervised Deep Generative Models for solving the multi-label problem by proposing unique mathematical and programming concepts and organization.

In all naive mixtures, using multiple labels is detrimental and causes each label's predictions to be worse than models that utilize only a single label. Examining latent spaces reveals that in many cases, large regions in the models generate meaningless results. Enforcing a priori independence is essential, and only when applied can multi-label models outperform the best single-label models. Finally, a novel learning technique called open-book learning is described that is capable of surpassing the state-of-the-art classification performance of generative models for multi-labeled, semi-supervised data sets.

Keywords: deep learning; semi-supervised learning; multi-label; generative models; a priori independence; open-book testing

Introduction

Advances in technology hardware – such as sensors, communication bandwidth, storage solutions – and technology software – such as social media and online services – are giving rise to big data. It is estimated [7] that 2.5 exabytes of data are created daily, a rate so fast that 90% of the data in the world today was created in the last two years.

Data science and machine learning are needed to deal with the enormous volume, rapid velocity, and variety of information being generated. Machine learning algorithms can be designed to make split-second decisions on high-dimensional data, therefore they are prime candidates for dealing with the volume and velocity of incoming data. However, they typically require a lot of *labeled* data for training; they require supervision to guide them to the correct answers.

Needing full supervision poses a severe problem in practice because of the complexity of real data. Real data captures infinitely many causes and effects with high correlations, and identifying even some of them can require precious, expert assessment. Big data is complex but is label-deficient.

Supervised learning leads to excellent results, but assigning true labels to training data is significantly more costly than collecting the data itself, and thus supervised learning alone is incapable of dealing with our ever-expanding world. **Unsupervised** algorithms have the potential to organize and cluster raw data, but they cannot make decisions because they do not account for labels.

Semi-supervised algorithms utilize unsupervised components that can leverage the incredible volume of data as well supervised elements to incorporate the precious labels. For a fixed amount of labeled data, semi-supervised algorithms outperform purely supervised ones because they extract information from the complex patterns in huge stores of additional unlabeled data.

The complexity of real data suggests that multiple experts could collaborate to make more-informed decisions than any one in isolation could make. Perhaps knowing the diagnosis of one expert can alter or influence that of another. For this reason coupled with label deficiency, we need semi-supervised algorithms that are capable of coordinating multiple labels.

Our world's overwhelming amounts of complex data provides a very practical motivation. Semi-supervised algorithms are key to bridging the discrepancy between the amount of data and the amount of experts, but being semi-supervised is only one characteristic that is necessary for an algorithm to help us with this task.

1.1 A Practical Motivation

The algorithms in this dissertation are guided by the following five design principles, all of which are motivated by challenges and complexity in big data.

1. Data driven.

   The systems proposed here do not require selection and hand-crafting of features nor do they require fine-tuning of the algorithms to data. Model parameters are learned from data.

2. Semi-supervised.

   Models need to be able to leverage both precious labeled data and abundant unlabeled data.

3. Steerable.

   The representations learned by unsupervised algorithms may not map to anything in the real world, but we need models that capture expert knowledge and information. Furthermore, because there are multiple experts in different but related fields, candidate models must be able to incorporate multiple tags or labels for each piece of data.

4. Generalize well.

   Generalization is tied to models' inference and generative capabilities. Models must be able to recognize (infer) information about new data, and they must be able to consider possible variations of existing data.

5. Adaptable.

   Our world changes constantly, and thus we need algorithms that can adapt accordingly.

This dissertation addresses all five of the motivating requirements. All of the models use deep learning in order to avoid hand-crafted feature selection. The problem formulations are semi-supervised so that models extrapolate from a very small selection of supervised data. Steering comes both from the semi-supervised nature of the models as well as the multi-label models we propose in the later chapters; we can embed little snippets of our real-world understanding by tagging data with multiple labels to serve as anchors for learning. Generalization is improved by creating deeper models and by enforcing a priori independence which strengthens models in a top-down fashion. And finally, the proposed concept of open-book testing tackles the issue of adaptation to new data.

All of the models in this dissertation share two fundamental components: an ability to generate data from imagination and an ability to recognize data in the context of a learned imagination. For an algorithm to

develop its generative and recognition components, it mimics its sensory input. It encounters a piece of data, tries to relate the data to its imagination, uses its understanding to attempt to recreate the data, examines its degree of success or failure, and uses that measure of accuracy to further develop its imagination powers. A model's understanding is directly visible in its latent spaces, and thus this dissertation focuses on the analysis and design of those spaces.

The forewords of the chapters that describe new models have scorecards that evaluate models in terms of the five motivating requirements. The models become progressively more successful through the dissertation, with the earliest ones achieving only 2.5 or 2.9 out of 5 and the final ones reaching near the full 5 out of 5. The scores are *highly subjective*, but I believe they provide reasonable summaries of the strengths and weaknesses of the different models.

1.2 Classes, Labels, Tasks, and Domains

Multi-class problems are classification problems in which each piece of data belongs to exactly one of several (more than two) classes. Binary classification algorithms such as support vector machines [26] can be extended to multi-class problems [38]. The MNIST handwritten digits are prototypical of this kind of problem as each data point is either a zero, a one, a two, etc., but cannot be both zero and one simultaneously.

Multi-label problems are classification problems in which each piece of data can be tagged with a (variable) number of labels [104]. As an example, consider a collection of books at a library. Each book can be categorized into one or multiple genres simultaneously resulting in combinations such as romantic science-fiction or nonfiction children's stories.

Multi-task problems use a single set of data on different tasks with the goal of exploiting commonalities and differences in shared representations [18, 99]. This is, in some sense, a generalization of both multi-class and multi-label learning. An example of this would be to predict a person's age and gender simultaneously from a photograph.

Distinguishing between multi-class, multi-label, and multi-task problems is difficult because all of them can be viewed as special cases of the others. For example, classification on MNIST's handwritten digits is a 10-class problem, but it can also be seen as ten related 1-vs-all tasks [98]. It also can be seen as a multi-label problem in which there is exactly one label present for any piece of data [104]. Furthermore, semi-supervised learning can be considered to be multi-task with one task being classification (the supervised part) and the other being data reconstruction (the unsupervised part).

In this dissertation, we consider MNIST to be **multi-class** and the NORB data set [55] to be **multi-label**. Technically, our interpretation of multi-label is closer to multi-multi-class, but we choose the name multi-label because each data point has associated with it several labels (overall class, elevation, and lighting). For

example, one data point can be a car seen at 45° under lighting condition L2 and another a human seen at 30° under lighting L5. We ultimately are interested in classifying a data point's overall class (car, human, truck, plane, animal), and we use the extra information of elevation and lighting to help classification. For this reason, we consider this to be a single-task problem. Similarly, we consider the semi-supervised labeling of points to be a single task.

Domains are complementary to classes, labels, and tasks. A domain consists of a coherent input set of data such as a set of images or a body of written documents. Specialized deep learning methods exist to leverage the structures of unique domains such as convolutional methods for images [47] and recurrent neural networks for sequential data like language or audio [31, 36]. Multi-domain mixtures can be used within any combination of single- or multi- class, label, or task problems [99]. Examples of multi-domain problems include identifying a person using both camera and microphone or creating text captions for images [72, 99].

Even though the data sets in this dissertation consist of images, no domain-specific knowledge is allowed into the algorithms. All of the methods could be extended to different domains by using the same fundamental algorithms and models but adding domain-specific machinery between the input data and the latent spaces.

1.3 Contributions

The following is a list of all novel contributions sorted by order of appearance in this thesis.

1. New nonlinearity called Nonplus with properties of both rectified linear units and sigmoid activations (Chapter 2). It performs better than sigmoid activations for the hybrid model of Chapter 2 and better than rectified linear units for the GPLVM of Chapter 3.

2. New objective function for successfully training a hybrid autoencoder and classifier (Chapter 2). Without this objective there is no information transfer between labeled and unlabeled data despite them sharing a common encoder.

3. Merger of Gaussian process latent variable models (GPLVM) with neural networks (Chapter 3). Gaussian processes require matrix inversion on all possible data pairs, limiting the number of points GPLVM can handle in practice. Neural networks amortize the matrix inverses in mini-batches thereby allowing GPLVM to scale to large data sets and significantly reducing training time.

4. Creation of lava floor distance metric with respect to data density in two-dimensional space (Chapter 3). This distance results in decision boundaries that are better than those found by strong support vector machines or neural network classifiers.

5. Three novel semi-supervised single-label DGMs, one of which is competitive with the best semi-supervised DGM while requiring fewer variables (Chapter 4).

6. Tensorized framework for DGMs (Chapter 5). Implementing DGMs *is possible* using standard matrices, but the tensorized framework trivializes the creation of complex models without which multi-label models would be infeasible.

7. New objective functions and training procedure for enforcing a priori independence (Chapter 6). API is critical in coordinating multiple labels, and it better organizes multiple classes in single-label models.

8. Novel learning paradigm called open-book testing (Chapter 7). State-of-the-art DGMs greatly improve their classification accuracies in this setting.

9. Five novel multi-label DGMs, one of which is nearly perfect (99.7%) and outperforms the best single-label DGM on its sole task (Chapter 7).

1.4 Historical Overview

Deep learning via artificial neural networks is currently the most promising solution to big data and artificial intelligence problems. Unlike other machine learning techniques, these networks do not require hand-crafted features, special prepping of data, or fine-tuned algorithms. Furthermore, once trained, these networks are capable of super-fast response at very low power per computation cycle [27, 49].

The "deep" part of deep learning is one of the essential components [11, 13, 37] for capturing information beyond what is locally available in adjacent data points. Increased depth compared to previous generations of neural networks has become possible due partly to advances in nonlinearities [33], initializations [12, 32], adaptive gradient descent [43, 101], hardware capabilities [21, 24, 53], software tools [8, 14, 25], and better understanding of objective functions to guide learning.

An enormous advantage of deep learning over other machine learning methods is the fact that deep learning automatically extracts features from data that optimize the objectives. For example, traditional (not deep-learning) supervised classification on images requires custom feature extractors – like SIFT [57], SURF [10], or ORB [79] – coupled with custom ways of transferring information from related points [69] and custom algorithms for expanding to large data sets [71]. By comparison, state-of-the-art supervised classification on images can be obtained using convolutional neural networks that require only data and the reasonable assumption that neighboring pixels in an image are connected [22, 47, 54].

Early mixtures of supervised and unsupervised deep learning used unlabeled data in autoencoders as pretraining for a supervised classification task [12, 37], but though successful such initialization was found

unnecessary with better nonlinearities [33]. Semi-supervised learning became the focal point instead of a secondary task with the formulation of three generic hypotheses: the distribution of input data and the conditional distribution of output data are related [50], real data in high dimensional spaces are likely to concentrate in lower-dimensional sub-manifolds [67], and points of different classes are likely to cluster in different sub-manifolds [77].

Semi-supervised embedding [94] and manifold tangent classifiers [77] leveraged those three hypotheses and were the state-of-the-art in semi-supervised deep learning in the early 2010s. The best semi-supervised algorithms outside of deep learning included transductive SVM [42] with low-density separation [20], atlas RBF [70], and probabilistic graphical models [46] using distributed versions of expectation maximization for better computational efficiency [96]. While deep learning methods were pushing the envelope, semi-supervised learning had much more room for improvement.

Two concepts significantly changed the landscape around 2014: deep generative models (DGM, also known as variational autoencoders, VAE) [45] and generative adversarial networks (GAN) [34]. Both were originally formulated as unsupervised methods but have simple extensions to semi-supervised learning [44, 80], and both rely on deep neural networks. While GANs have been applied successfully in many fields [28, 39, 68, 76, 82, 100], our focus is on DGMs in this dissertation. Structurally, the primary difference is that DGMs are autoencoders that contain both a generative network and an inference network whereas GANs have a similar generative network but instead use an adversarial discriminator that does not perform inference. Recently theoretical connections have been explored [62] between the two approaches.

Deep Generative Models result from the combination of probabilistic graphical models with deep learning. They are fully-Bayesian but avoid difficulties of inference in large data sets with complex models of non-conjugate distributions [45]. For generic semi-supervised learning [44], their graphical models contain at least three elements – an observed data variable, a partially-observed class variable, and a hidden latent variable that captures all variation in data not due to class. It was found that adding an additional auxiliary variable [59] improves classification accuracy. This addition is a point of examination in Chapter 4 and serves as a building block for our multi-label models in Chapter 7.

Research in DGMs is expanding rapidly, and most of it falls under one of two categories. The first category explores different model architectures and contains conditional variational autoencoders [83] that treat the class label as an output rather than an input in the generative model, [35] that adds a stochastic variable used in the generation of both data and classes, [84] that describes the ladder variational autoencoder which has a deep hierarchy of latent variables, and [97] that creates a separation of latent variables in order to segment foreground from background in images. The second category examines different objectives that can be used to train the models and contains the importance-weighted autoencoder [17] which uses repeated

sampling to tighten the training criteria's bounds and [64] which extends the multi-sample idea with a new gradient estimator.

The tensorized framework that we develop in Chapter 5 helps to bring the capabilities of graphical models into neural network toolsets. Conversely, automatic differentiation variational inference brings the power of neural networks into a Bayesian toolchain [48]. Tensors are standard in convolutional neural networks [2–4] and have recently been used in multi-task deep learning [98]. They can be incorporated into our tensorized framework despite our differing rationales.

The a priori independence criterion we develop is related to the concepts of fairness and disentangling sources of variation. A variational fair autoencoder [56] is a VAE in which the learned latent representation is invariant (fair) to what was originally described as either a sensitive variable or a nuisance variable [102]. The goals of fair VAEs are similar but the approach is different to our work in Chapter 6. The concept has been analyzed also in the context of GANs [15, 61, 89].

The NORB data set has very unique properties compared to other image databases [16]. It contains merely fifty objects, but each one is imaged under 1,944 different controlled conditions, whereas other sets tend to have more objects and thus generalization is simpler. Multi-label models built for databases like NORB tend to be fully supervised [29] or convolutional [72]. While not based on NORB, other works have similar approaches of separating modalities [87, 97] and create hierarchical models [88] like ours in Chapter 7. It has been shown that models trained partly on controlled object sets like NORB generalize well to other natural images [16].

Benchmarking on MNIST is much more common than NORB. Performance on the label-deficient semi-supervised MNIST problem now is almost identical to the fully-supervised MNIST case, leaving little room for improvement (Table 1.1). NORB is much more challenging (Table 1.2) and only recently has received increased attention [44]. The current state-of-the-art on semi-supervised NORB classification without using domain-specific knowledge is SDGM at 90.6% [59]. Full supervision and domain-specific knowledge of images can result in improvements to 97.5% [23]. Our implementation of SDGM reaches about 89% after 100 epochs, whereas the reported result of 90.6% uses 2000 epochs and network parameters tailored to NORB [59].

Comparing this dissertation to existing methods is difficult in the context of semi-supervised NORB due to our use of multiple labels as well as the novel open-book testing procedure. Open-book testing can drastically improve existing methods' classification accuracies with a prime example being SDGM reaching 99.3% (Table 7.2). There are no multi-label NORB results that are directly comparable to ours, so we create several model variations to serve as baselines. Using a priori independence and the best model we develop in this dissertation results in a nearly-perfect 99.7% (Table 7.3).

Table 1.1: Summary of MNIST Classification. All reported methods are trained using only 100 labeled data points and 60,000 unlabeled points, and none use convolutional neural networks or other domain-specific knowledge.

Method	Accuracy (%)
Neural Network [44]	74.2
Support Vector Machine [94]	76.6
Transductive SVM [94]	83.1
EmbedNN [94]	83.1
Contractive Autoencoder [78]	86.6
Manifold Tangent Classifier [77]	88.0
Deep Generative M2 [44]	88.1
Atlas RBF [70]	91.9
Deep Generative M1 + M2 [44]	96.7
Categorical GAN [85]	98.1
Virtual Adversarial Training [63]	98.6
Skip DGM [59]	98.6
Auxiliary DGM [59]	99.0
Ladder Network [73]	99.0
GAN with Feature Matching [80]	99.1

Table 1.2: Summary of NORB Classification. No consistent benchmark exists, though typical semi-supervised results are obtained by training on 1000 labeled data points and 24,300 unlabeled points. Some methods are convolutional. Open-Book testing is a product of this dissertation.

Method	Accuracy (%)
TSVM [44]	74.0
M1 + TSVM [44]	81.2
SST with Temporal Coherence [60]	82.0
Disentangling DCGAN + MLP [61]	86.5
Auxiliary Deep Generative Model [59]	89.9
Virtual Adversarial Training [63]	90.1
Skip Deep Generative Model (SDGM) [59]	90.6
Fully-Supervised Convolutional NN [23]	92.1
Fully-Supervised CNN with Domain Knowledge [23]	97.5
Open-Book SDGM	99.3
Open-Book StackedQP with J_{api}	99.7

1.5 Organization

Each chapter in this dissertation is based on a standalone paper. To bridge the technical content together, I have decided to include a foreword before each chapter's introduction.

The notation in each chapter is slightly different because each is sourced on a unique paper and focuses sharply on a specific topic. The differences in notation allow each chapter to ignore the bits that are irrelevant. A unified notation would have too many subscripts and Greek letters at all times.

This dissertation examines two data sets in depth: the MNIST handwritten digits and the NORB controlled object images. The earlier portions of the dissertation are focused on issues of semi-supervised learning and algorithm family; these chapters focus on the single-label MNIST data set in a label-deficient setting and show that even a simple data set can have complex issues. After understanding and solidifying the fundamentals, we move to the multi-label NORB data set in order to bring us closer to our motivations. The information contained in the multiple labels is very difficult to coordinate, and it turns out that naively adding the labels decreases performance. Analysis of this problem and its solutions are presented in Chapter 6 and Chapter 7.

Chapters in this dissertation present a sequential deepening of understanding. The first technical chapter, Chapter 2, contains my first interactions with deep neural networks [75]. This chapter is critical in order to understand neural networks, as every following topic builds upon and utilizes the basics here. The second technical chapter, Chapter 3, gives a noticeable improvement over the results of the previous chapter because it successfully merges neural networks with a probabilistic model [74]. All subsequent chapters go even farther in the direction of this merger. Both of these chapters contain results from middle to late 2015.

The next chapter provides a detailed analysis of three existing papers [44, 45, 59] that shattered the semi-supervised world. The results in those papers were so overwhelmingly successful, I had to try to understand them, so I spent the year of 2016 analyzing the math and implementing the algorithms. Chapter 4 is a collection of details and examinations into Deep Generative Models. There are a number of visualizations that I needed to create in order to understand the roles of the latent variables in different DGMs, work from 2017 which directly influenced the remaining chapters.

The two following chapters, Chapter 5 and Chapter 6, provide solutions to two of the most fundamental issues I found while examining DGMs. The first uses tensors instead of matrices in order to *significantly* reduce the mental burden of creating more intricate DGMs. The second identifies and fixes the issue of latent spaces learned by semi-supervised DGMs resulting in generative models that actively produce garbage. Both of these serve critical roles in the final technical chapter.

Chapter 7 culminates with a very large, hierarchical model that incorporates multiple labels. Without the tensorized framework and a priori independence, the results herein would not have been possible. And what results they are! To my knowledge, the classification accuracy that I obtained is the best in the world on the open-book, multi-label problem. I am very excited about this success, but of course I realize that, practically, the nine-times extra computation time and increased memory usage might not be worth the effort. This work is very fresh from the beginning of 2018 and as such has not yet had time to mature, but nevertheless, the fact that multiple labels can coordinate to improve the model's *understanding* gives me hope as well as future research directions.

The chapters build upon one another sequentially. The final chapter provides concluding remarks and presents several possible future research directions based on the work in this dissertation.

Hybrid Autoencoder and Classifier for Label-Deficient Semi-Supervised Learning

The entirety of this dissertation uses deep learning at its core with this chapter as an introduction that examines several basic questions, such "How deep is deep?" and "What are the effects of using different nonlinearities?" Additionally, the entirety of this dissertation focuses on semi-supervised learning with this chapter providing a very intuitive but primitive understanding of semi-supervised algorithms.

This chapter stays solely within the standard tools of semi-supervised neural network applications. It relies on autoencoders, classifiers, and gradient descent but has no probabilistic components. The next chapter introduces a probabilistic generative path to replace half of a standard autoencoder and discards the standard classifier entirely. The subsequent chapters are fully probabilistic in a Bayesian-inference setting. The models in the later chapters, including the fully-probabilistic algorithms, simply are fancy autoencoders.

The focal point of this chapter is, as the title suggests, a hybrid autoencoder and classifier. The autoencoder's latent space is called a bottleneck space or bottleneck layer in this chapter because it is constrained to two dimensions, thereby forcing an extreme dimension reduction, and is used both to reconstruct data and to classify labels. The autoencoding is purely unsupervised and finds underlying relationships in data. The knkowledge of some labels can be used to manipulate the encoding and improve the classification performance on the entire data set.

To accomplish this a single objective function that mixes several sub-objectives is used. The single objective function was found to be immensely more effective than training the system with two separate objectives for autoencoding and classification.

In terms of the five motivating requirements, the semi-supervised autoencoder-classifier hybrid model proposed in this chapter gets a score of 2.5 out of 5, shown in Table 2.1. It uses deep learning to avoid

Table 2.1: Hybrid Model Scores

	Data Driven	Semi-supervised	Steerable	Generalization	Adaptability	Total
Hybrid Model	1	1	0.5	0	0	2.5

hand-crafted features and finely-tuned classification algorithms. The model is semi-supervised using as little as 100 labels from the training data (0.2% of the training set). It is steerable but only marginally so, since it would have difficulty with multiple labels, thus half a point for this requirement. Examining the latent space in Figure 2.1(a), we see clearly that the model lacks a good understanding of the data, so no points here. At this primitive stage, whether or not it adapts to new data is irrelevant, so no points there either. Nevertheless, the hybrid model in this chapter is a good starting point.

2.1 Introduction

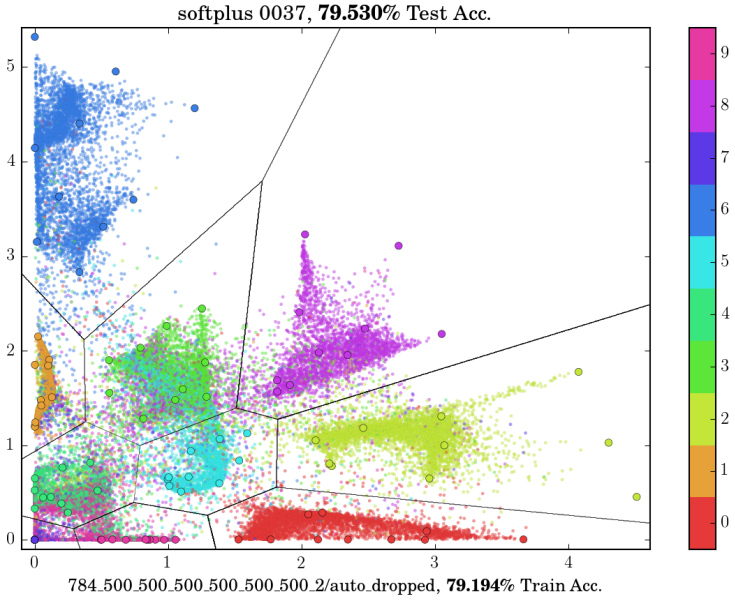
Label-deficient semi-supervised learning is a difficult variation of semi-supervised learning in which the amount of labeled data is minuscule, thus modeling many real world problems that involve large quantities of inexpensive (and unlabeled) sensor data and very small amounts of expensive human-assigned labels.

This chapter uses the MNIST handwritten digits data set. Each data point is a 28-by-28 pixel grayscale image containing one handwritten number between 0 and 9. We treat each point as a 784-dimensional vector and do not utilize the spatial structure inherent in images' neighboring pixels. For the purposes of label-deficient semi-supervised training only 100 labeled points – ten examples of each digit – are used.

Several semi-supervised deep networks have been proposed and have shown fantastic results on this challenging problem [44, 70, 77, 94]. The hybrid autoencoder and classifier neural networks in this chapter will force all results through a bottleneck of two dimensions for the sake of visualization. Figure 2.1 shows an example visualization of the bottleneck space and defines the meanings of the colors, circles, dots, and lines in all visualizations.

Deep neural networks are capable of dealing with the individual challenges inherent in semi-supervised problems; they are able to scale to large data sets [24], have been used in dimension reduction as a means of unsupervised learning [9, 37, 78], and have produced state-of-the-art classification results across multiple domains [86]. They have been used in fully-labeled semi-supervised settings by jointly training autoencoders and classifiers where the combination leads to better classification generalization on unseen data [12, 103]. We find similar results but show that a straight-forward combination is problematic when the amount of labeled data is very small, as depicted visually in Figure 2.2. We propose a novel objective function that can successfully train a hybrid system in this label-deficient setting.

This chapter is organized such that it formulates the hybrid network in section 2.2, describes the experiment and collects the results of the case study in section 2.3, and provides commentary in section 2.4. The formulation combines several objective functions in a novel way, culminating in the unified objective function of Equation 2.15. The novel nonlinearity, nonplus, also is defined in section 2.2, and discussion regarding its performance is in section 2.4. This case study provides insights into the effects of increasing network depth



(a)

(b)

Figure 2.1: (a) An example of the visualization of the two-dimensional bottleneck layer imposed onto every network. Each color corresponds to a different digit/class according to the colorbar to the right-hand side of the plot, the data points are plotted as dots colored according to their true classes, and the 100 labeled points, ten from each class and shown in (b), are plotted as larger circles with black outlines. The lines that separate clusters are the decision boundaries of a classifier trained in this bottleneck space, and the emboldened percentage in the title shows the accuracy of that classifier on unseen testing data. The 100 labeled points in (b) are constant throughout all experiments. No effort was made to select typical points or to remove outliers, and a support vector classifier trained just on those points in the full-dimensional image space achieves 70% accuracy on unseen testing data.

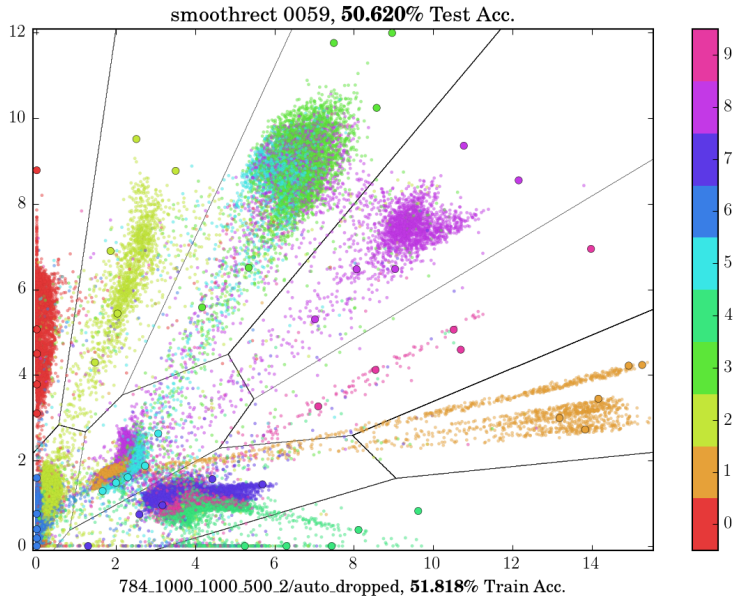


Figure 2.2: A straight-forward combination of autoencoder and classifier objectives yields poor results in a label-deficient setting. In this example, it is clear that the unlabeled 4s, 7s, and 9s are clustered very close to each other but that the labeled 4s and 9s are very far from their unlabeled clusters. The decision regions that classify points as either a 4 or a 9 are empty!

and nonlinearity choice in hybrid networks as well as the distinct ways that different nonlinearities fill space. Concluding remarks are summarized in section 2.5.

2.2 Formulation

The hybrid structure that we wish to train has three components: encoder, decoder, and classifier. All three are deep neural networks which stack together multiple layers. The combination of the encoder and the decoder produces an autoencoder, and the mixture of the autoencoder and classifier is intended to help the classifier better generalize to unseen data. The network is trained by optimizing via stochastic gradient descent an objective function with respect to the parameters of the network's layers.

Each layer has multiple identical units with different connection weights and internal biases. For the ith layer, the input is a matrix X_i with rows corresponding to data points and columns corresponding to dimensions of each point. The output of the ith layer is Y_i, the input multiplied by a weight matrix W_i, added to a bias vector b_i, and passed through a nonlinearity s_i.

$$Y_i = s_i(X_i W_i + b_i) \tag{2.1}$$

The nonlinearity s_i is known at design time, but the weights and biases are updated throughout the training process. W_i is sized appropriately based on the desired network structure, and b_i is a row vector whose length

is the number of columns of W_i. In a stack of layers, the output of layer i is the input of layer $i+1$.

$$Y_{i+1} = s_{i+1}(X_{i+1}W_{i+1} + b_{i+1}) \tag{2.2}$$

$$= s_{i+1}(s_i(X_iW_i + b_i)W_{i+1} + b_{i+1}) \tag{2.3}$$

We consider five nonlinearities for the units of the hidden layers. These are applied element-by-element to each element x.

$$\text{sigmoid}: \quad s(x) = \frac{1}{1+e^{-x}} \tag{2.4}$$

$$\text{nonplus}: \quad s(x) = \frac{1}{1+e^{-x}} + \frac{x}{20} - \frac{1}{2} \tag{2.5}$$

$$\text{softsign}: \quad s(x) = \frac{x}{1+|x|} \tag{2.6}$$

$$\text{rectify}: \quad s(x) = \max(0, x) \tag{2.7}$$

$$\text{smoothrect}: \quad s(x) = \log(e^x + 1) \tag{2.8}$$

The nonplus nonlinearity is novel, hoping to combine the characteristics of the sigmoid and rectify nonlinearities. Its performance and analysis are presented in section 2.3 and section 2.4.

In addition to the five nonlinearities is softmax, reserved for the single classification layer at the top of each network. Softmax is applied to a vector x.

$$\text{softmax}: \quad s(x) = \frac{e^x}{\sum e^x} \tag{2.9}$$

Note that e^x is the element-by-element exponential of the vector x such that the output of the softmax nonlinearity is a vector with the same shape as the input vector x.

The hybrid structure stacks layers to form an encoder $E(X)$ that takes as input raw images X and outputs an encoding Z as well as a decoder $D(Z)$ and classifier $C(Z)$ that take as input encodings Z. In this case study, the input of the encoder has $28 \times 28 = 784$ dimensions and outputs two dimensions with zero to seven hidden layers of 500 units in between. The decoder has the same structure as the encoder but in reverse and with free weights, and the chaining of the encoder and decoder results in an autoencoder $A(X) = D(E(X))$. The classifier takes a two-dimensional input and produces a ten-dimensional one-hot output, matching the ten classes in this case study. The classifier uses the softmax nonlinearity, and the decoder and encoder use one of the other five nonlinearities.

A combination of objective functions is needed to train the hybrid network. Consider an N by P input matrix X consisting of N images. The autoencoder objective L_{ae} measures the distance between X and reconstructions $A(X)$.

$$L_{\text{ae}}(X) = \frac{1}{NP} \sum_{n,p} \Big(X - A(X) \Big)^2_{n,p} \tag{2.10}$$

Corresponding to the images X is an N by 10 dimensional matrix of one-hot labels Y. The classifier objective L_{clf} minimizes the categorical cross entropy between the labels Y and the predicted labels $C(Y)$.

$$L_{\text{clf}}(X, Y) = -\frac{1}{N} \sum_{n} \sum_{c=0}^{9} Y_{n,c} \log(C(X)_{n,c}) \tag{2.11}$$

In addition to the character-defining objectives L_{ae} and L_{clf} are two generic objectives L_{norm} and L_{center}. Given that $E(X) = Z$ has rows z_n, then the norm penalty is L_{norm}.

$$L_{\text{norm}}(X) = \frac{1}{N} \sum_{n} z_n z_n^T \tag{2.12}$$

Further, given that $\text{argmax}_{\text{index}}\, C(z_n) = \hat{y}_n$ is the predicted class label of z_n and $\bar{z}_{\hat{y}_n}$ is the centroid in the bottleneck space of the points with known label equal to $\hat{y}_n$, then L_{center} is the penalty on being far from the class center.

$$L_{\text{center}}(X) = \frac{1}{N} \sum_{n} (z_n - \bar{z}_{\hat{y}_n})(z_n - \bar{z}_{\hat{y}_n})^T \tag{2.13}$$

The separate objective functions can be weighted and combined into a unified objective. The training data is split such that X and Y are separated into X_u, X_l and Y_u, Y_l corresponding to the unlabeled and labeled portions of the data. The true labels of the unlabeled data Y_u are not known, therefore X_u, X_l, and Y_l are available for training. The straight-forward combination of the objectives is L_{sf}.

$$\begin{aligned} L_{\text{sf}}(X_u, X_l, Y_l) = {} & \alpha_{\text{clf}} L_{\text{clf}}(X_l, Y_l) \\ & + \alpha_{\text{ae}} L_{\text{ae}}(X) \\ & + \alpha_{\text{norm}} L_{\text{norm}}(X) \\ & + \alpha_{\text{center}} L_{\text{center}}(X) \end{aligned} \tag{2.14}$$

Networks trained with the straight-forward objective function do not perform well when the amount of labeled data is very small, as seen in Figure 2.2. Our novel remedy is the objective function L_{hyb}.

$$\begin{aligned} L_{\text{hyb}}(X_u, X_l, Y_l) = \alpha_{\text{clf}} L_{\text{clf}}(X_l, Y_l) \\ + \alpha_{\text{ae},u} L_{\text{ae}}(X_u) \\ + \alpha_{\text{ae},l} L_{\text{ae}}(X_l) \\ + \alpha_{\text{norm}} L_{\text{norm}}(X) \\ + \alpha_{\text{center}} L_{\text{center}}(X) \end{aligned} \tag{2.15}$$

The novelty is the application of L_{ae} to the separate sets X_u and X_l with different weights $\alpha_{\text{ae},u}$ and $\alpha_{\text{ae},l}$. In order to successfully train a hybrid network with a small amount of labeled data, then $\alpha_{\text{ae},l}$ must be very large. Setting it too small causes L_{hyb} to degenerate back to L_{sf}.

2.3 Experiment and Results

The experiment uses only 100 labeled images, ten examples per class, and 49,900 unlabeled images, and its purpose is to clarify the effects of nonlinearity choice and varying network depth. The 100 labeled points are fixed and shown in Figure 2.1(b). A support vector classifier trained on these specific 100 images is able to achieve 70% correct classification accuracy on an unseen set of 10,000 testing images, so when using a hybrid network trained with our proposed objective, any classification accuracy greater than 70% can be considered a success that benefits from the inclusion of both labeled and unlabeled data.

During training, the autoencoder $A(X)$ may be truncated to $A_k(X)$. The truncation feeds the output of the first k layers of the encoder to the last k layers of the decoder. Varying k during training inherits the positive effects of greedy layer-wise training [12] to help initialization, of jointly training the layers [107] to ease the burden on the lower layers, and of embedding information into all layers [94] for improved performance.

The results in Table 2.2 show the best testing accuracy of every combination of network depth and nonlinearity type in the case study. The softplus nonlinearity outperforms all of the others by a significant margin, and almost all nonlinearities exhibit the trend of better performance with increasing network depth.

Figure 2.3 shows the encoding of all points X_u and X_l in the bottleneck space of two dimensions along with the class boundaries found by the classification layer at the top of each network. Column (a) shows the effect of increasing network depth by using the nonplus nonlinearity as an example. Column (b) shows the best results obtained by four nonlinearities, regardless of network depth. Column (c) shows a sequence of iterations during the training of a softplus network.

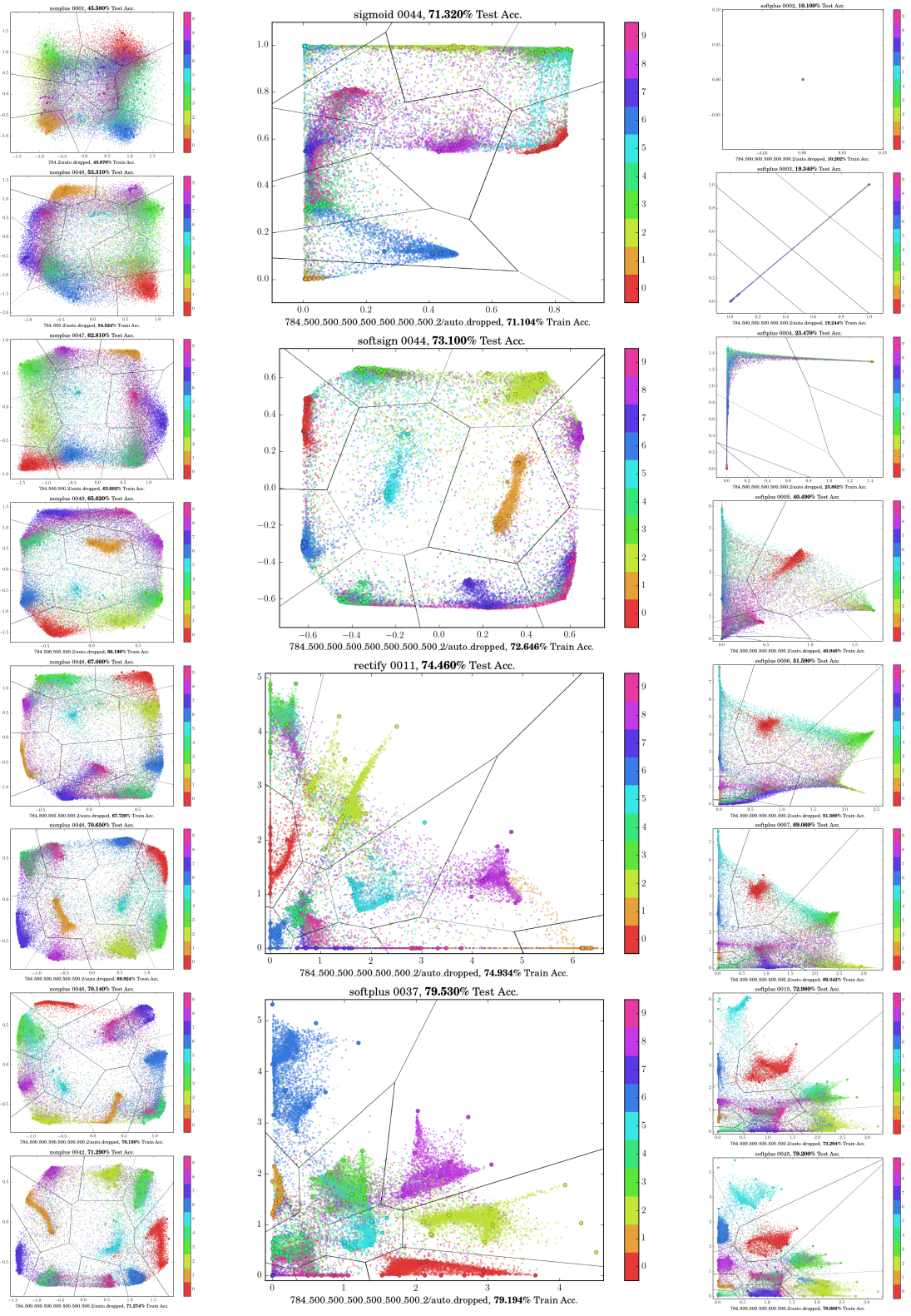


Figure 2.3: Bottleneck layer visualizations. (a) Nonplus with increasing network depth. (b) Best results of different nonlinearities. (c) Softplus with increasing iteration.

Table 2.2: Highest recorded test-data classification accuracy percentages. The peak accuracies are chosen regardless of training iteration but do not exceed the 50th iteration. Initializations of the network weights are random, causing some entries (such as softplus's four-hidden layer result) to seem inconsistent. Each ∘ represents a hidden layer of 500 units.

	nonplus	**rectify**	**sigmoid**	**softplus**	**softsign**
784,2,10	45.58	8.92	42.83	46.2	48.05
784,∘,2,10	53.31	63.55	62.26	64.75	52.76
784,∘,∘,2,10	62.81	64.87	65.9	65.31	61.15
784,∘,∘,∘,2,10	65.62	65.49	68.54	76.1	70.86
784,∘,∘,∘,∘,2,10	67.08	72.74	69.47	70.55	69.95
784,∘,∘,∘,∘,∘,2,10	70.65	74.46	70.4	79.2	72.85
784,∘,∘,∘,∘,∘,∘,2,10	70.14	72.24	69.79	79.53	70.25
784,∘,∘,∘,∘,∘,∘,∘,2,10	71.29	71.94	71.32	78.42	73.1

2.4 Discussion

The difference between the straight-forward objective function L_{sf} and our proposed objective function L_{hyb} is subtle. The reason why L_{sf} fails is because it allows the few labeled points to have very poor reconstructions, slightly decreasing L_{ae}, in order to improve their class' separations, significantly increasing L_{clf}. Even if α_{ae} is very large in L_{sf}, the subcost L_{ae} is dominated by X_u instead of X_l due to the disparity in the number of data points in each set. The proposed objective L_{hyb} ensures that the labeled points' reconstructions are not poor, and since there is an explicit smooth mapping back from the encoded space to the image space, the decoder, this has the effect of putting those points nearby other data points that would have similar reconstructions. The link between the labeled and unlabeled data points allows the classifier that is trained on labeled points to generalize well. Another way of creating a similar link is to preserve pairwise relationships between the labeled and unlabeled points using an unsupervised learning algorithm that directly uses pair information [94].

Figure 2.3 provides several interesting insights regarding the distinct ways that different nonlinearities fill the bottleneck space. The novel nonplus nonlinearity's usage of space looks similar to that of softsign, but the sigmoid nonlinearity has very pronounced corners and edges despite the fact that all cases used exactly the same norm penalty α_{norm}. Due to the random initialization, the sigmoid nonlinearity easily gets stuck in its saturation region, an issue that nonplus does not have due to its linear tails, and while the softsign nonlinearity is capable of saturating, it moves out of saturation faster than the sigmoid nonlinearity by design [32]. The best performance among these three nonlinearities is 73% accruacy achieved by the softsign nonlinearity. The two other nonlinearities, rectify and softplus, perform better. Both cluster data into similar shapes, but due to the flat-zero region of the rectify nonlinearity, it has comparatively more

collapsed clusters. The softplus nonlinearity consistently achieves the best performance, with a peak of 79% accuracy. Figure 2.3(c) shows that the softplus nonlinearity is capable of recovering from a collapsed initial state.

Based on Table 2.2 and Figure 2.3(a), it appears that increased network depth results in better classification performance. The sequence of images of the nonplus nonlinearity with varying number of hidden layers shows two effects as the number of layers increases: a decreasing number of points is left in the space between clusters, and the clusters become tighter with sharper edges. The rectify nonlinearity is the only exception to the trend probably because of its flat-zero region coupled with the increasing difficulty of recovering from a collapsed state as the number of parameters increasing.

2.5 Conclusion

In a label-deficient semi-supervised setting, a straight-forward combination of an autoencoder and classifier does not perform well, as can be seen in Figure 2.2. We formulate a hybrid network and provide a novel objective function that can successfully train the network even when the amount of labeled data is minuscule. The novel objective function links the labeled data and the unlabeled data by adding a heavily weighted sub-objective on the autoencoding of the labeled data, separate from the sub-objective dedicated to the autoencoding of the unlabeled data.

The case study provides valuable insights into the operation and behaviors of the hybrid network by depicting visually the effects of different nonlinearities and network depths. A thorough examination between five unique nonlinearities and eight network depths reveals that networks of increasing depths perform better in classification and that the nonlinearities have different capabilities regarding classification and the ways in which they fill space. The novel nonlinearity, nonplus, along with the sigmoid and softsign nonlinearities fill their bottleneck spaces in similar ways with the exception that the sigmoid nonlinearity shows more pronounced corners and edges. Among these three, the softsign performs the best at 73%, followed by nonplus and sigmoid at 71%. However, the two other nonlinearities outperform those three with the best rectify result at 74% and the best softplus result at 79%.

A standard support vector classifier trained on the 100 labeled points in the full-dimension space achieves 70% accuracy on unseen testing data. A deep network using the softplus nonlinearity trained by the hybrid objective function on the same set of 100 labeled points achieves 79% accuracy on unseen testing data. The proposed hybrid objective improves classification accuracy by incorporating both labeled and unlabeled data, and it is capable of creating successful classifiers in the label-deficient setting.

GPLVM and Lava Floor Distance for Label-Deficient Semi-Supervised Learning

One glaring problem with the hybrid model of the previous chapter is the amount of emptiness in the latent space, as seen in Figure 2.1(a). This emptiness is an indication that the hybrid model's imagination is limited. Large areas of possibility are collapsed into small parts of the latent space.

This chapter introduces Gaussian processes latent variable models (GPLVM). One interesting component of these models is the covariance function that determines how the latent space should be organized, and the parameters of the covariance function can force the latent space to be completely filled. GPLVM's latent space, Figure 3.5, is much more pleasing and natural than the hybrid model's latent space.

Intuitively, GPLVM performs dimension reduction in a fashion similar to flattening the globe onto a planar map. Points that are far apart in the original space, such as Louisiana and India, will remain far apart in the reduced space. But points that are nearby, such as Alaska and Russia, are not guaranteed to remain close in the reduced space. On many flat maps, Alaska ends up on the westernmost edge and Russia on the easternmost edge.

The version of GPLVM that we develop in this chapter uses a neural network for the encoder and a Gaussian process for the decoder. Both neural networks and Gaussian processes are universal function approximators in the sense that they can represent arbitrarily complex functions using an infinite number of parameters. Neural networks provide weights and biases as their parameters, but Gaussian processes use raw data directly, so in a sense, they have *no parameters*.

Because there are no parameters in the decoder of GPLVM, training models in this chapter is much more time-efficient than in the previous chapter. In fact, GPLVM with a neural network encoder trains in only a couple of minutes, much faster than any other model in this dissertation. Note that the lack of parameters, while significantly speeding training, forces testing to be computationally demanding, time-consuming, and memory intensive.

In this chapter, we maintain notation with GPLVM literature though we would have preferred the latent space to be Z and the raw data to be X. We reserve Y for class labels elsewhere in this dissertation.

In terms of the five motivating requirements, GPLVM with lava floor gets a score of 2.9 out of 5, shown in Table 3.1. As with the hybrid model of the previous chapter, this too uses deep learning, is semi-supervised, and would have difficulty with multiple labels, but we give it a slight bonus for the requirement of avoiding expert design because of the fact that the generative model is purely data driven and has no parameters, thus

Table 3.1: GPLVM Scores

	Data Driven	Semi-supervised	Steerable	Generalization	Adaptability	Total
GPLVM	1.1	1	0.5	0.3	0	2.9

making the model incredibly efficient in training time. Comparing Figure 3.5(h) with Figure 2.1(a) reveals that GPLVM's generalization of the data space is much better than the hybrid model's but less expressive than the following chapters' models, thus netting it 0.3 points for the fourth requirement. Like before, at this primitive state, adaptability is irrelevant.

GPLVM fails to satisfy the motivating requirements, but it is incredibly useful as a visualization technique. We use it extensively for that purpose in Chapter 6 and Chapter 7.

3.1 Introduction

We propose a two-stage method for applying Gaussian process latent variable models [51] (GPLVM) in a label-deficient setting. We first train a back-constrained GPLVM [52] using an objective function that includes discriminative GPLVM [92], and then we use the novel lava floor distance to extend the system to classification. The lava floor distance operates on the cluster assumption [20] that points of the same class ought to form clusters separated by regions of low density, and it leverages the capabilities of GPLVM to perform reductions down to two-dimensional spaces where the distance between two locations is the shortest path weighted by density and can be computed efficiently using Djikstra's algorithm. The lava floor distance is similar to the image forest transform [30] except that it is applied to a density histogram instead of an image.

Two related approaches in this label-deficient setting are EmbedNN [94] and the manifold tangent classifier [77]. EmbedNN, which augments a neural network classifier with an unsupervised learning algorithm, is similar to our combination of GPLVM with neural networks. The manifold tangent classifier, which uses tangent information first learned from a contractive autoencoder in a subsequent classifier, is similar to our two-stage process, first extracting density information in the form of the lava floor distance and then using it in a classifier. Our method differs from [77] in the usage of GPLVM instead of contractive autoencoders and lava floor distance instead of manifold tangents, and it differs from [94] in that it trains the network using a single objective function for the purpose of dimension reduction with classification at a later stage.

The rest of this chapter describes the formulation of our two-stage method, including the necessities and drawbacks of the individual components, in section 3.2, presents the experimental results of the case study in section 3.3, and provides discussion in section 3.4. The back-constraining neural network is trained using the objective function of Equation 3.15, and its extension to classification is through the lava floor distance described in subsection 3.2.2. In section 3.3, we compare the classification performance of our proposed lava floor distance against a standard support vector classifier (SVC) trained on the same GPLVM-trained networks, showing that the lava floor distance improves classification accuracy and successfully utilizes information from the unlabeled portion of the data. Finally, section 3.5 summarizes by providing concluding remarks.

3.2 Formulation

We propose a two-stage method for applying GPLVM in a label-deficient setting. The first stage trains a neural network using combined GPLVM, discriminative GPLVM, and back-constrained GPLVM objectives and is described in subsection 3.2.1. The second stage extends GPLVM to classification using the lava floor distance and is described in subsection 3.2.2.

3.2.1 GPLVM and Objective Functions

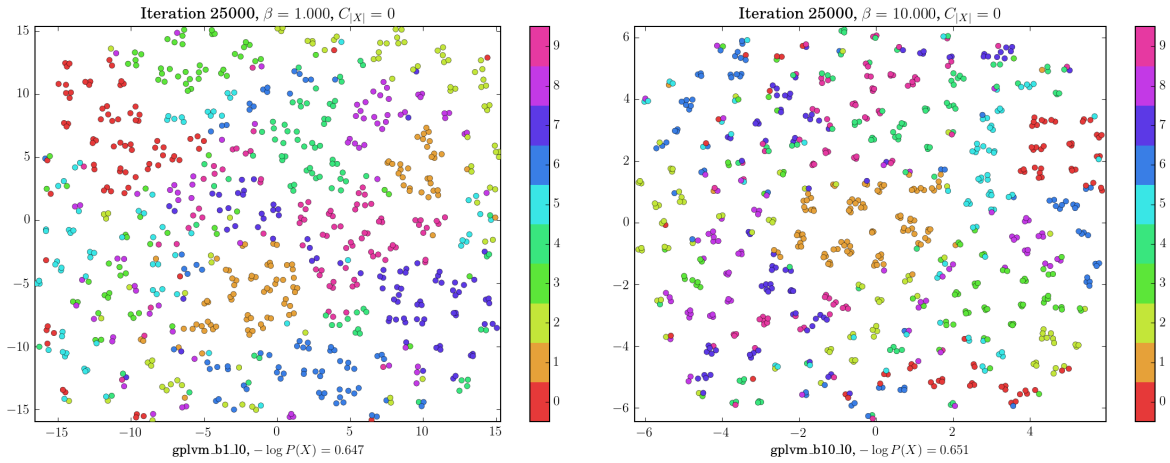
Data points are stored in a matrix Y with N rows, one row per data point, and D columns, one column per dimension. Corresponding to every high dimensional point is a point in a reduced-dimension latent space. Let the collection of corresponding latent points be X with N rows, one row per data point, and $P = 2$ columns, one column per dimension of the latent space. Determination of the matrix X is the goal of GPLVM.

The GPLVM likelihood function evaluates the likelihood of the observed data Y given a corresponding X and a chosen covariance matrix K.

$$P(Y|X,K) = \frac{1}{(2\pi)^{DN/2}|K|^{D/2}} \exp\left(-\frac{1}{2}\mathrm{tr}(K^{-1}YY^T)\right) \tag{3.1}$$

The exponentiated quadratic covariance K is a function of X and has a single parameter β. The elements of K are K_{ij}, corresponding to x_i and x_j, the ith and jth rows of X.

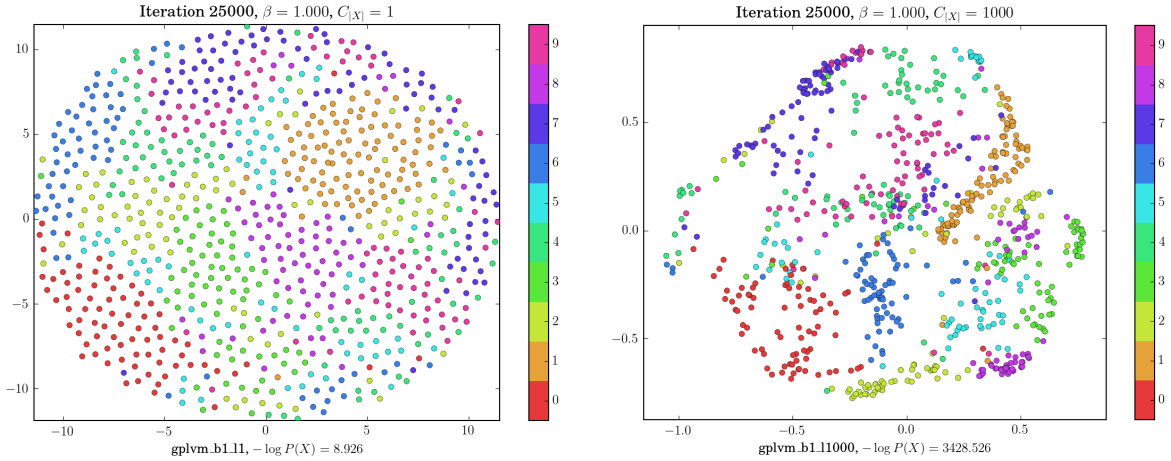
$$K_{ij} = k(x_i, x_j) = \exp\left(-\frac{|x_i - x_j|^2}{\beta}\right) \tag{3.2}$$



(a) $\beta = 1$ (b) $\beta = 10$

(c) $\beta = 100$ (d) $\beta = 1000$

Figure 3.1: Varying the length scale parameter β in the standard GPLVM objective L_G, Equation 3.4.



(a) $C_{|X|} = 1$ (b) $C_{|X|} = 1000$

Figure 3.2: Varying the norm penalty $C_{|X|}$ in $L_{G,X}$, Equation 3.6.

The matrix X is found by minimizing $-\log P(Y)$, the GPLVM negative log-likelihood with implied conditioning on X and β inside of the matrix K.

$$-\log P(Y) = \frac{DN}{2}\ln(2\pi) + \frac{D}{2}\ln|K| + \frac{1}{2}\text{tr}(K^{-1}YY^T) \tag{3.3}$$

The first two terms are safe to ignore, leaving the GPLVM objective function L_G.

$$L_G = \text{tr}(K^{-1}YY^T) \tag{3.4}$$

Figure 3.1 shows the result of minimizing L_G with varying values of β using the first $N = 1000$ images from the MNIST data set as Y. The most noticeable outcome is that high dimensional data points are shy or anti-social. In order to force more interaction between points, we add a norm penalty term, $L_{|X|}$, onto the objective function.

$$L_{|X|} = \sum_n x_n x_n^T \tag{3.5}$$

As before, x_n is the nth row of X. The combined objective is $L_{G,X}$.

$$L_{G,X} = L_G + C_{|X|}L_{|X|} \tag{3.6}$$

The value of $C_{|X|}$ controls the strength of the penalty, and Figure 3.2 shows the effect of its increase when $\beta = 1$.

The discriminative GPLVM variant adds label information to the standard GPLVM. DGPLVM originally [92] adds an objective to keep points within the same class together, S_w, and another objective to create space between classes, S_b. We use the knowledge that our data set is anti-social, so S_b can be ignored. The data set needs to be separated into two pieces, Y_u and Y_l, such that the labeled points are in Y_l and the remaining points are in Y_u. The corresponding latent locations are similarly separated into X_u and X_l. Each $x \in X_l$ belongs to the class $c(x)$, which is known beforehand, and the mean of class $c(x)$ is $\bar{x}_{c(x)}$. The discriminative objective, L_D, is applicable only to the labeled latent locations X_l.

$$L_D = \sum_{x \in X_l} (x - \bar{x}_{c(x)})(x - \bar{x}_{c(x)})^T \tag{3.7}$$

Combining the discriminative objective with $L_{G,X}$ gives $L_{G,X,D}$.

$$L_{G,X,D} = L_G + C_{|X|} L_{|X|} + S_w L_D \tag{3.8}$$

Figure 3.3 shows the result of applying the discriminative objective to 100 labeled points, and a quick comparison to Figure 3.1 and Figure 3.2 reveals that not only are our points anti-social, but the unlabeled are afraid of the labeled! Fortunately, Figure 3.3 (c) shows a reasonable arrangement of points once a significant norm penalty is imposed.

Back-constraining neural networks allow GPLVM to scale to $N > 1000$ data points. The back-constraining function f has parameters θ such that the latent locations of the data points are $f(Y|\theta)$ instead of X, thereby changing the goal of the optimization to finding the parameters θ. The new objective function, L_{BC}, is the same as that of GPLVM

$$L_{BC} = \mathrm{tr}(K^{-1} Y Y^T) \tag{3.9}$$

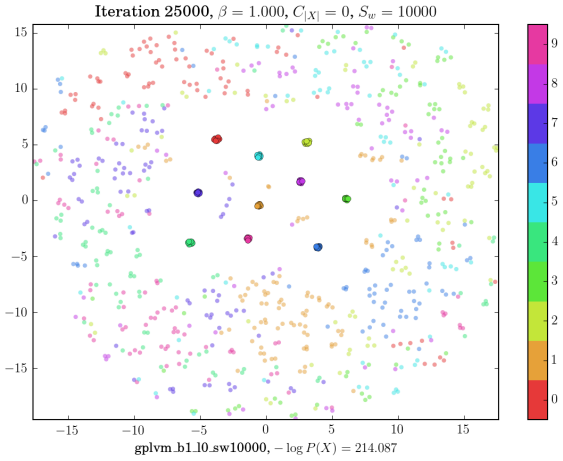
except that the entries of the covariance matrix K are computed using $f(Y)$ instead of X.

$$K_{ij} = k(f(y_i), f(y_j)) = \exp\left(-\frac{|f(y_i) - f(y_j)|^2}{\beta}\right) \tag{3.10}$$

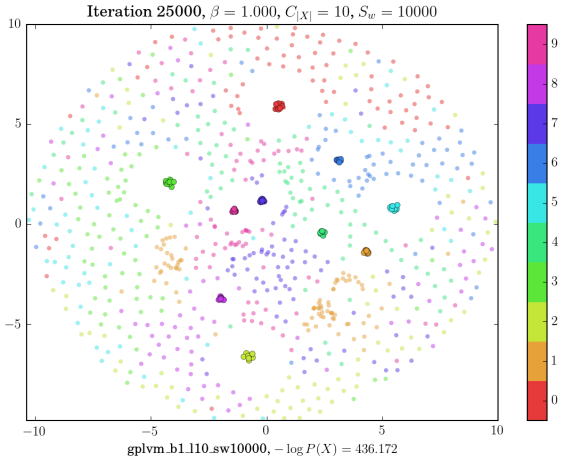
The matrix K has elements K_{ij}, and y_i, y_j are the ith and jth rows of Y.

The function f is a neural network constructed by a stack of layers with a chosen nonlinearity. Each layer i has an input X_i, an output Y_i, weights W_i, biases b_i, and nonlinearity s_i.

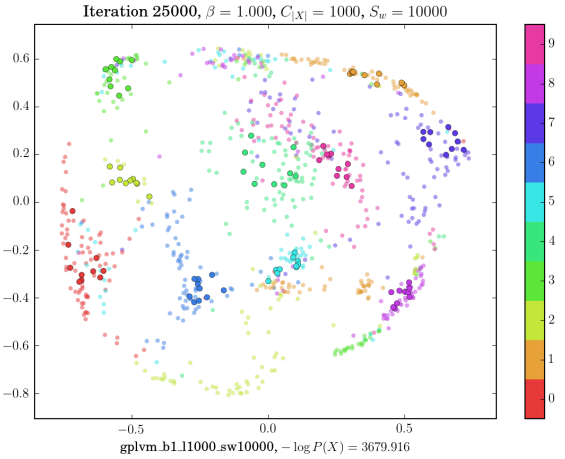
$$Y_i = s_i(X_i W_i + b_i) \tag{3.11}$$



(a) $C_{|X|} = 0$



(b) $C_{|X|} = 10$



(c) $C_{|X|} = 1000$

Figure 3.3: Unlabeled data points are afraid of labeled points until they are forced to play nice together. All three images come from Equation 3.8 with $S_w = 10000$ but with varying $C_{|X|}$.

The dimensions of W_i and b_i are chosen to match the desired input and output shapes of that layer, and the layers are stacked such that the output of one layer, Y_i, is the input to the next layer, X_{i+1}. In this case study, the nonlinearity s_i is one of three element-by-element functions.

$$\text{softsign}: \quad s(x) = \frac{x}{1 + |x|} \tag{3.12}$$

$$\text{nonplus}: \quad s(x) = \frac{1}{1 + e^{-x}} + \frac{x}{20} - \frac{1}{2} \tag{3.13}$$

$$\text{sigmoid}: \quad s(x) = \frac{1}{1 + e^{-x}} \tag{3.14}$$

The input to the first layer of the stack always has $784 = 28 \times 28$ dimensions, the output always has two dimensions, and there are zero to five hidden layers of 500 units in between. All layers contribute W_i and b_i to the set of parameters θ.

Combining L_{BC} with the previous norm-penalty and discriminative objectives gives $L_{BC,X,D}$.

$$L_{BC,X,D} = L_{BC} + C_{|X|} L_{|X|} + S_w L_D \tag{3.15}$$

Compared to Equation 3.8, L_{BC} is a drop-in replacement of L_G with two exceptions: the optimization is with respect to θ as opposed to X, and the network f can be trained using stochastic gradient descent on a large data set by subdividing the data into minibatches.

3.2.2 Lava Floor Distance

Networks trained with Equation 3.15 perform dimension reduction and are influenced by label information, but they are not classifiers. A way to convert them is to train a standard support vector classifier in the reduced-dimension space. SVCs learned in this fashion perform reasonably well, as can be seen in Table 3.2 and Figure 3.5 (a) - (c), but they ignore knowledge gained from unlabeled data. Figure 3.4 (a) shows that the decision boundaries of a standard SVC often cross high density regions, thus violating the cluster assumption.

The lava floor distance captures information about the distribution $P(f(Y))$ and makes it available for use in a classifier. It is a distance d between any location (r, c) and any other location (r', c') in the two-dimensional space. The distance is computed according to the following six steps.

1. Divide the two-dimensional space into a grid of R rows and C columns.
2. Count the number of points $f(Y)$ that fall into each cell to create a histogram.
3. Convolve a Gaussian kernel of desired σ with the $R \times C$ histogram. The result is a kernel density estimate $P(r, c)$ that can be evaluated at any location (r, c).

4. Create G, a square $R \times C$ by $R \times C$ sparse graph.

5. For every cell (r, c) in the space, add an edge between it and every neighboring cell (r_n, c_n). The weight (or cost) of the edge is L_{LF}.

$$L_{LF}(r, c, r_n, c_n|P) = \left(1 - \frac{P(r_n, c_n)}{\max(P)}\right)^8 \tag{3.16}$$

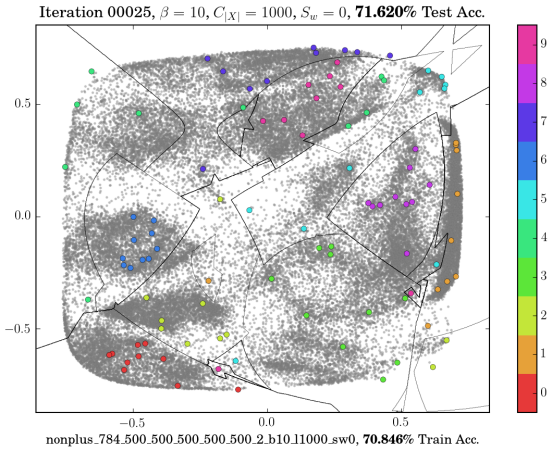
6. The distance $d(r, c, r', c'|P)$ is the length of the shortest path through G between (r, c) and (r', c'). Shortest paths are computed using Dijkstra's algorithm.

Two points y and y' in the high-dimensional space, reduced to $f(y)$ and $f(y')$ in the two-dimensional space, are considered as belonging to cells (r, c) and (r', c'), and the distance between them is the lava floor distance $d(r, c, r', c'|P)$. The distance d contains the lengths of the shortest paths, and Figure 3.4(d) shows visualizations of the shortest paths between all 100 labeled points and a randomly chosen location inside of the blue cluster of sixes. Notice that the paths tend to favor traveling through regions of high density and only cross low-density regions when necessary. Figure 3.4(b) shows the decision boundaries found by using the lava floor distance in a k-nearest neighbor classifier (LFKNN). The classification boundaries run along low-density regions whenever possible.

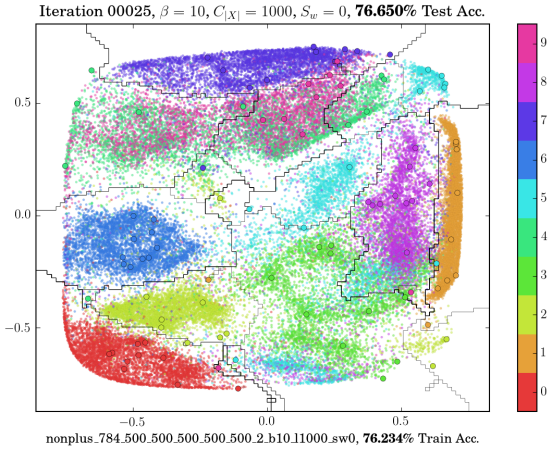
3.3 Experiment and Results

The objective functions are minimized using gradient descent. Optimization of the objective functions L_G, $L_{G,X}$, and $L_{G,X,D}$ first begins with random initializations for the elements of X, and the combined objective $L_{BC,X,D}$ begins with a random initialization of the parameters θ. Theano [14] is capable of symbolically obtaining the gradients of an objective function with respect to X or θ. Note that the pairwise differences needed in the covariance matrix K can be obtained symbolically using Theano's broadcasting rules (derived from NumPy's broadcasting rules) while avoiding loops.

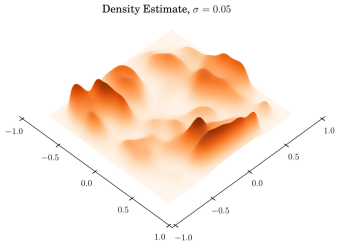
The results of optimizing the combined objective function $L_{BC,X,D}$ are shown in Table 3.2 and Figure 3.5 for fixed β, S_w, $C_{|X|}$, network depth, and nonlinearity. Table 3.2 shows how frequently unseen testing images are classified correctly, and Figure 3.5 shows the mapping of all training images along with decision boundaries. The results of SVC and lava floor k-nearest neighbor (LFKNN) are shown separately, but the underlying networks are identical for the same sets of parameters; Figure 3.5 (f) and (h) are direct counterparts. For reference, a support vector classifier trained on the 100 labeled points in the original image space achieves 70% accuracy on unseen test data.



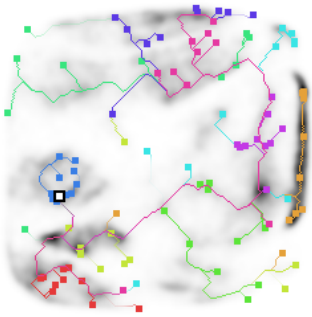
(a) SVC Boundaries



(b) LFKNN Boundaries



(c) Density Estimate



(d) Shortest Paths

Figure 3.4: Various views of the lava floor distance given a fixed $f(Y)$. (a) Decision boundaries obtained using a standard support vector classifier utilizing the (colored) labeled data while ignoring information about (gray) unlabeled data. (b) Decision boundaries using lava floor k-nearest neighbors (LFKNN) utilizing both labeled and unlabeled data are a full five percent better than (a). (c) Kernel density estimate of unlabeled points from (a) and (b). (d) Shortest paths between all (colored squares) labeled points to a specific (black-outlined square) location while taking into account low-density penalties. The lava floor name comes from the game many of us played as children where we hopped from chair to chair while pretending the ground is covered in lava. Moving atop a piece of furniture is free, but touching the ground is costly.

Table 3.2: Classification accuracies on unseen testing data of BC-GPLVM with and without label information across varying length scales β, network sizes, and nonlinearities. Each cell has two accuracies side-by-side: the number to the left shows the result of ignoring label information by setting $S_w = 0$, and the number to the right forces points of the same class together by setting $S_w = 10000$. Each $\circ$ is a hidden layer of 500 units. All accuracies less than 70% are grayed out, and accuracies greater than 75% are bolded.

NN Size	$\beta = 1$	$\beta = 10$	$\beta = 100$	$\beta = 1000$
784,2	48.34 46.22	33.08 44.39	37.56 45.10	28.96 36.51
784,∘,2	60.98 62.17	61.81 59.18	40.57 41.92	31.76 38.84
784,∘,∘,2	59.27 65.58	62.36 70.86	44.07 42.88	36.07 31.17
784,∘,∘,∘,2	69.09 64.31	73.21 70.22	37.05 42.24	24.93 34.56
784,∘,∘,∘,∘,2	71.54 47.56	65.57 55.95	32.72 37.59	31.04 45.14
784,∘,∘,∘,∘,∘,2	66.86 48.87	70.02 43.44	39.60 49.48	22.01 28.57

(a) softsign with SVC, $C_{|X|} = 0$

NN Size	$\beta = 1$	$\beta = 10$	$\beta = 100$	$\beta = 1000$
784,2	43.85 45.82	39.35 45.71	38.17 41.67	18.25 29.39
784,∘,2	57.17 60.16	56.22 67.49	34.67 43.71	20.90 28.74
784,∘,∘,2	61.60 63.96	58.68 64.65	43.46 49.64	22.93 27.12
784,∘,∘,∘,2	60.37 67.48	66.32 **76.67**	35.83 46.37	22.10 31.47
784,∘,∘,∘,∘,2	62.63 66.70	72.92 **77.33**	39.38 52.98	25.13 47.89
784,∘,∘,∘,∘,∘,2	62.61 73.18	71.62 75.14	32.28 40.96	26.11 34.98

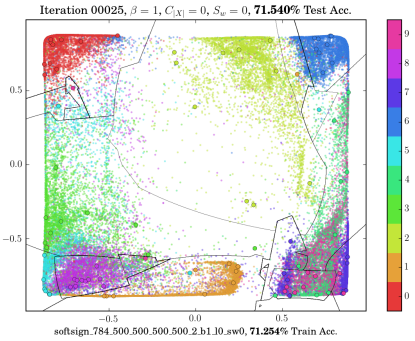
(b) nonplus with SVC, $C_{|X|} = 1000$

NN Size	$\beta = 1$	$\beta = 10$	$\beta = 100$	$\beta = 1000$
784,2	44.75 46.28	46.66 47.61	44.78 39.20	21.67 31.14
784,∘,2	59.15 63.80	62.59 75.38	65.23 65.29	21.95 40.17
784,∘,∘,2	67.84 70.28	75.09 **79.34**	66.33 72.49	35.34 44.05
784,∘,∘,∘,2	68.26 72.89	75.36 **76.64**	65.73 68.25	22.51 48.00
784,∘,∘,∘,∘,2	68.86 75.04	20.29 74.20	10.32 63.14	15.87 13.56
784,∘,∘,∘,∘,∘,2	10.09 24.33	10.09 11.17	18.23 10.09	10.09 16.94

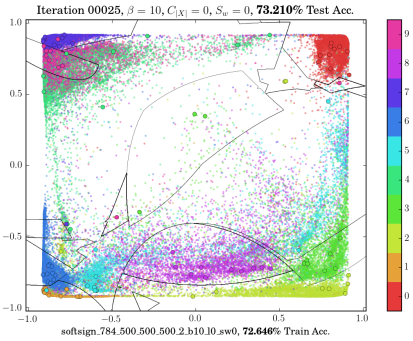
(c) sigmoid with SVC, $C_{|X|} = 0$

NN Size	$\beta = 1$	$\beta = 10$	$\beta = 100$	$\beta = 1000$
784,2	46.01 44.35	48.39 47.49	45.23 37.29	20.87 34.19
784,∘,2	63.67 64.71	62.78 74.85	63.87 65.73	19.78 40.87
784,∘,∘,2	66.80 66.25	71.81 **81.30**	64.35 74.58	25.60 43.00
784,∘,∘,∘,2	68.99 75.06	**78.20 80.18**	65.95 71.38	19.81 42.62
784,∘,∘,∘,∘,2	65.97 75.44	14.42 **80.20**	10.28 63.05	11.76 12.06
784,∘,∘,∘,∘,∘,2	10.28 22.72	10.28 12.99	17.51 10.28	10.28 10.16

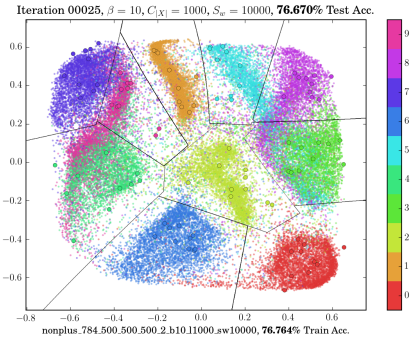
(d) sigmoid with LFKNN, $C_{|X|} = 0$



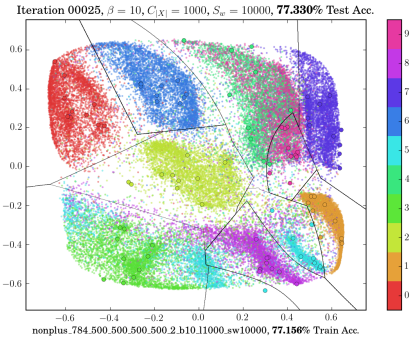
(a) softsign with SVC, 71.54%



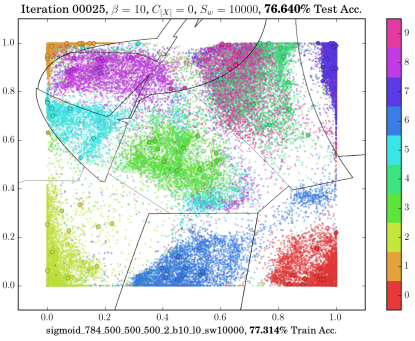
(b) softsign with SVC, 73.21%



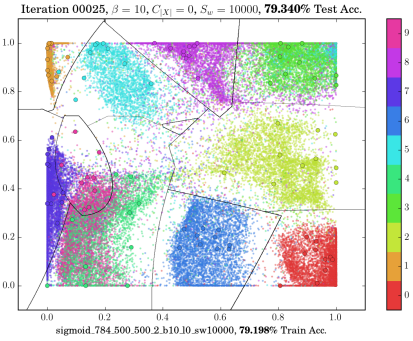
(c) nonplus with SVC, 76.67%



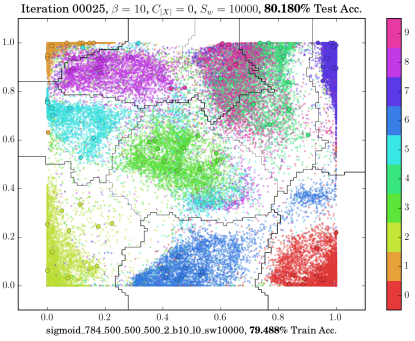
(d) nonplus with SVC, 77.33%



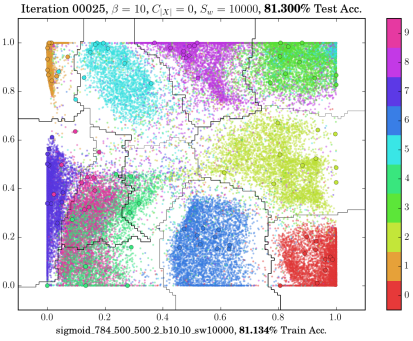
(e) sigmoid with SVC, 76.64%



(f) sigmoid with SVC, 79.34%



(g) sigmoid with LFKNN, 80.18%



(h) sigmoid with LFKNN, 81.30%

Figure 3.5: Visualizations of the encodings of all training images in learned spaces along with classification accuracies and boundaries found by SVC and LFKNN. The two best results from each of the four subtables of Table 3.2 are shown.

All of the results coming from $L_{BC,X,D}$ have the norm penalty $C_{|X|}$ set to zero except for the cases in which the nonplus nonlinearity is involved. The softsign and sigmoid nonlinearities are bounded functions, whereas the nonplus nonlinearity is unbounded, and thus softsign and sigmoid do not need a norm penalty term in the objective function. Values of $C_{|X|} < 1000$ consistently led to poor classification results for the nonplus nonlinearity.

3.4 Discussion

The results of the case study highlight five areas of discussion: the length scale β, the formation or lack of clusters caused by varying $C_{|X|}$, the utilization of space by different nonlinearities, the effects of network depth, and the value of the discriminative objective.

The length scale parameter β seems to have very little impact on the standard GPLVM objective function L_G for high dimensional data, as can be seen in Figure 3.1. The latent positions move far enough apart that, regardless of β, no two points in the latent space have strong interactions with one another. For the norm-penalized and the back-constrained versions of the objective functions, β needs to be selected appropriately in order to obtain the best results, as evidenced by the last two columns of Table 3.2.

We find it surprising that GPLVM and DGPLVM do not produce "clusters" unless points are forced to intermingle. In the latent space, GPLVM is known to preserve the dissimilarity of data points as opposed to the distances between points [52], but the ballooning of the space is unusual. Thanks to the dissimilarity preservation and despite the ballooning, GPLVM organizes points well, as can be seen best in Figure 3.2 (a) where even though all of the points are separated, the zeros (colored red) are in the bottom left and all of the ones (colored orange) are near the top right. What surprises us even more is the effect, the fear that unlabeled points have of labeled points, of forcing together known instances of each class by adding the discriminative objective L_D, as seen in Figure 3.3. This issue appears only in the label-deficient setting and not in the fully-labeled setting in which DGPLVM was originally proposed [92]. Neither the ballooning nor the fear are relevant with large $C_{|X|}$ or with bounded back-constraints.

Among the three nonlinearities that we tested, sigmoid consistently achieved the best performance. We think this is because of two reasons. The primary reason is that the exponentiated quadratic covariance function, the Gaussian likelihood, and the sigmoid function all have exponential components, thus allowing data points to move and to be placed freely in the entire usable space. Comparing Figure 3.5(b) and (e), we can see clearly that the learned encoder in the softsign case cannot use the center, and thus almost all of the encoded data points are forced to the edges of the space, whereas the encoder in the sigmoid case has no trouble filling the space naturally. The second reason is that the sigmoid nonlinearity, due to the fact that it is a bounded function, does not need $L_{|X|}$, allowing the learning process to focus more on the

remaining sub-objectives. Comparing Figure 3.5(d) and (f), we can see that the usage of space in both cases is very similar with the exception of the edges and corners that are less sharp in the case of the nonplus nonlinearity. When the nonplus nonlinearity is forced to have a large norm penalty $C_{|X|}$ in order to have reasonable results, it basically becomes like sigmoid, but ever so slightly worse, as evidenced by Table 3.2. We originally wanted to use rectifying linear units and the softplus nonlinearities [65], but their shapes led to the matrix K becoming singular and thus non-invertible.

The best classification results, according to Table 3.2, come from networks with two to four hidden layers. Networks with fewer layers are not powerful enough to freely control positions in the latent space according to GPLVM's guidance. Networks with too many layers suffer the traditional problems associated with deep networks, mainly that deep networks tend to be degenerate when randomly initialized, and they are not necessarily capable of recovering from all initializations. We are unsure if deeper networks will yield better results for the MNIST dataset, but we see that even shallow networks can produce good results.

The inclusion of the discriminative objective L_D improves classification performance. In each cell of Table 3.2, the accuracies to the right, corresponding to the discriminative case, are larger than the accuracies to the left. The softsign results are an exception but largely irrelevant because softsign's performance is very poor overall in this setting. The LFKNN classifier outperforms SVC in the best cases, cases in which the classes are well separated and the discriminative objective guided all instances of the same class to be very near each other. In those few cases, using unlabeled points' locations to inform the classification boundaries' locations works well. We are unsure if the labeled points are typical or outliers of their respective classes. Typical data points would more naturally fall into their classes' clusters even without the addition of L_D, but outliers would force an informative warping of the space with the addition of L_D.

3.5 Conclusion

GPLVM can be used in a label-deficient setting by following the proposed two-stage method. We have shown how to train neural networks using GPLVM objectives, and the proposed lava floor distance successfully makes the learned knowledge available for classification. The case study reveals the behaviors of the individual components of the two-stage method, and it also provides results that highlight the behaviors of the entire method. Visualizations made possible by bottlenecking all networks through a two-dimensional space provide valuable insights.

For the standard GPLVM objective L_G, Figure 3.1 shows that high-dimensional data points are anti-social in the sense that none of them rest close together or are clustered in the reduced-dimension space regardless of the covariance's length scale parameter β. The norm-penalizing objective $L_{|X|}$ forces points together, with Figure 3.2 showing that large penalties lead to the formation of clusters.

The objective L_D captures label information to help guide the dimension reduction, but Figure 3.3 shows that even though the labeled points in each class form tight clusters, the unlabeled points stay far away from the labeled points. Using a strong weighting on $L_{|X|}$ forces all points to interact and leads to the expected formation of clusters.

The back-constrained objective L_{BC} extends the capabilities of GPLVM to handle large data sets. Based on Table 3.2 and Figure 3.5, the sigmoid nonlinearity is particularly well-matched to GPLVM and the exponentiated quadratic covariance function. By comparison, the softsign nonlinearity is not able to fully utilize the reduced dimension space, leading to many points being degenerately forced to the edges. Deep networks are not required in order to get good results; the best classification accuracies come from networks with as few as two hidden layers.

The lava floor distance, coupled with simple k-nearest neighbor classifiers, successfully extends networks trained with the GPLVM objectives to label-deficient settings. Comparing Table 3.2 (c) and (d) shows that the best LFKNN test-data accuracies are consistently better than the best SVC accuracies. In both cases, the discriminative objective L_D improves performance. The proposed method outperforms the baseline standard support vector classifier trained only on the 100 labeled points in the full-dimension image space.

Deep Generative Models

Deep Generative Models (DGMs) were first proposed in [45] as a variational autoencoder (VAE) using a single continuous latent variable and would be referred to as the model M1 in [44]. Building upon M1, the model M2 is a semi-supervised model that uses both a continuous latent variable and a discrete latent variable whose purpose is to incorporate classification labels.

The performance of the semi-supervised model M2 in complex and multi-labeled data sets falls short of expectation due to the fact that the inference network is not expressive enough to capture complex data distributions. In [59] the authors proposed the addition of a third continuous, auxiliary, latent variable to be used as an aid in inferring the discrete variable. Two models are proposed there: the Auxiliary Deep Generative Model (ADGM) and the Skip Deep Generative Model (SDGM).

A GitHub repository[1] that hosts my implementation of ADGM and SDGM is publicly available. It uses a fully-tensorized framework that *significantly* reduces the complexity of designing and routing large probabilistic models, as will be demonstrated in Chapter 5.

This chapter contains derivations of the semi-supervised objective functions and fills in the gaps between [45] and [44]. It contains experimental results for M1, M2, and SDGM that attempt to provide insight into their operations. It also performs several experiments on six novel models: Seq Z, Split Z, Full Z, M2 Split, 2M, and 2M Full, all of which are defined graphically in Figure 4.1 and Figure 4.2. This is the only chapter in the dissertation that is not based on a published or submitted research paper; it serves as background for understanding the following chapters.

In terms of the five motivating requirements, the three existing models M1, M2, and SDGM have scores of 1.3, 3.1, and 4.1, respectively, shown in Table 4.1. M1 is completely unsupervised, so it cannot be steered at all. M2, by comparison, can be steered a little but would have trouble with multiple labels. Its generalization capabilities are slightly better than M1's because of the fact that it incorporates a single label. SDGM is the best of the published models. We have not yet seen how it adapts to new data, but it performs excellently with open-book testing in Chapter 7. Its generalization is a weakness that will be examined and corrected in Chapter 6. Its other weakness is the fact that it does not readily incorporate multiple labels, thus failing to achieve full marks for steering. The models proposed in this chapter (Seq Z, Split Z, Full Z, M2 Split, 2M,

[1] `http://github.com/rrastgoufard/sdgm`

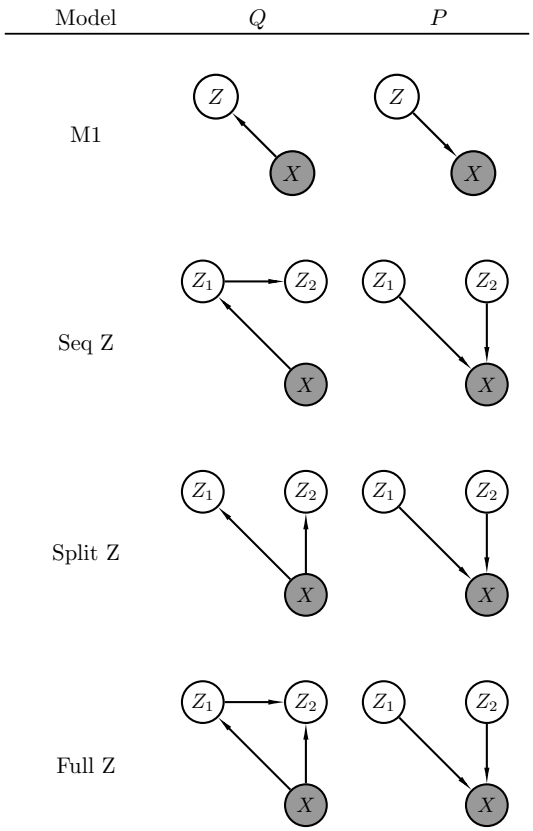


Figure 4.1: Unupervised variational DGMs showing their inference models Q and their generative models P.

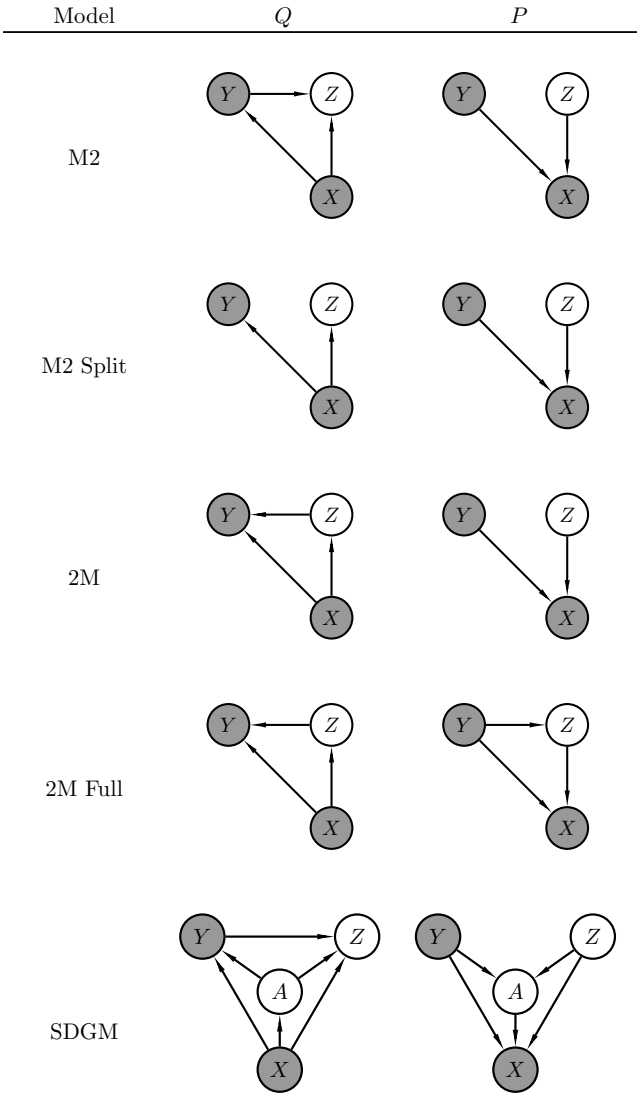


Figure 4.2: Supervised variational DGMs showing their inference models Q and their generative models P. The node X always is observed, and the node Y is observed only for a small number of labeled data points.

Table 4.1: Existing Models' Scores

	Data Driven	Semi-supervised	Steerable	Generalization	Adaptability	Total
M1	1	0	0	0.3	0	1.3
M2	1	1	0.5	0.6	0	3.1
SDGM	1	1	0.5	0.6	1	4.1

and 2M Full) are not serious contenders to satisfy the motivating requirements. All of them are designed solely for the sake of understanding various aspects of the existing models M1, M2, and SDGM.

Of the three existing models, SDGM is by far the best performing. It is only marginally more complicated than M2 but has significantly better accuracy and understanding. SDGM is the reference against which all other models need to be compared.

4.1 Evidence Lower Bound

We refer readers to [44, 45, 58, 59] for the theory behind deep generative models (DGMs) as well as the derivations of specific models. Here we will show the derivation of the simplest deep generative model, M1, first introduced in [45], then detail how to organize the computation of its objective function using tensors, and finally show how to extend the tensor representation to the more complex M2 and SDGM models.

In the specific deep generative model M1, first define a joint distribution $p(x, z)$ where x is the observed data and z is an unobserved latent variable that is responsible for generating x. The generative model factors into $p(x|z)p(z)$ and is marginalized to obtain $p(x)$.

$$p(x) = \int p(x|z)p(z)\ dz \tag{4.1}$$

Instead of solving the integral directly, introduce an inference network $q(z|x)$

$$p(x) = \int p(x|z)p(z)\frac{q(z|x)}{q(z|x)}\ dz \tag{4.2}$$

and recognize that the integral is an expectation over $q(z|x)$.

$$p(x) = \int q(z|x) \frac{p(x|z)p(z)}{q(z|x)} \, dz \tag{4.3}$$

$$= E_{q(z|x)} \left[\frac{p(x|z)p(z)}{q(z|x)} \right] \tag{4.4}$$

Taking the log of both sides and using Jensen's inequality creates a bound on the log-likelihood.

$$\log p(x) = \log E_{q(z|x)} \left[\frac{p(x|z)p(z)}{q(z|x)} \right] \tag{4.5}$$

$$\geq E_{q(z|x)} \left[\log \left(\frac{p(x|z)p(z)}{q(z|x)} \right) \right] \tag{4.6}$$

$$\log p(x) \geq -J(x) \tag{4.7}$$

The objective function $J(x)$ is an upper bound on the negative log-likelihood and has a negative sign for the sake of consistency with many machine learning algorithms that perform gradient descent to optimize the objectives.

Note that the difference between the ELBO and the log-likelihood is the KL divergence between the the inference distribution $q(z|x)$ and generative posterior distribution $p(z|x)$.

$$\mathrm{KL}\left(q(z|x) \middle\| p(z|x) \right) = E_{q(z|x)} \left[\log \frac{q(z|x)}{p(z|x)} \right] \tag{4.8}$$

$$= E_{q(z|x)} \Big[\log q(z|x) - \log p(x, z) \Big] + \log p(x) \tag{4.9}$$

Rearranging gives a view of the log-likelihood.

$$\log p(x) = \mathrm{KL} + E_{q(z|x)} \Big[\log p(x, z) - \log q(z|x) \Big] \tag{4.10}$$

$$= \mathrm{KL} + \int q(z|x) \left[\log \frac{p(x, z)}{q(z|x)} \right] dz \tag{4.11}$$

Because the KL divergence is always non-negative, the evidence can be lower-bounded.

$$\log p(x) \geq \int q(z|x) \left[\log \frac{p(x, z)}{q(z|x)} \right] dz = -J(x) \tag{4.12}$$

In order to evaluate the objective $J(x)$ and approximate the integral in its expectation, draw K_z samples from the distribution $q(z|x)$.

$$J(x) \approx \frac{1}{K_z} \sum_{i=1}^{K_z} J(x, z_i) \tag{4.13}$$

The objective $J(x)$ can be evaluated at each sample z_i as $J(x, z_i)$.

$$J(x, z_i) = \log \left(\frac{q(z_i|x)}{p(x|z_i)p(z_i)} \right) \tag{4.14}$$

The inference distribution $q(z|x)$ in M1 is Gaussian with mean $\mu_{qz}(x)$ and log-standard deviation $\sigma_{qz}(x)$ where both $\mu_{qz}(x)$ and $\sigma_{qz}(x)$ are neural network functions with weight matrices and bias vectors collectively called ϕ. Similarly, the generative distribution $p(x|z)$ in M1 has statistics that are obtained as neural network functions of z with weights and biases collectively called θ. The prior generative distribution $p(z)$ is assumed to be standard normal.

The reparameterization trick [45] is used to obtain samples z_i.

$$z_i \sim \mathcal{N}(\mu_{qz}(x), \sigma_{qz}(x)) \tag{4.15}$$

$$= \mu_{qz}(x) + e_{zi} \exp \sigma_{qz}(x) \tag{4.16}$$

$$e_{zi} \sim \mathcal{N}(0, 1) \tag{4.17}$$

The noise e_{zi} is drawn from a standard normal distribution and is scaled and shifted by $\exp \sigma_{qz}$ and μ_{qz} so that it transforms into observations z_i from the distribution $q(z|x)$.

With samples z_i drawn, the objective $J(x)$ in Equation 4.13 can be obtained symbolically, and automatic differentiation tools such as Theano can obtain the derivatives of $J(x)$ with respect to the neural network parameters θ and ϕ. It is at this point that existing literature stops providing detail under the (rightful) assumption that readers can use the provided information to implement and train their own DGMs. However, we wish to clarify and simplify some of the remaining steps by using tensors to organize computations.

4.2 Tensors for DGMs

The implementation of M1 requires some design parameters to be specified. First, assume that each data point x_n is a vector of length D_x. Next, assume that there are N data points in each minibatch. We choose the latent space z to have dimension D_z, and the expectation is approximated using K_z samples.

Each data point x_n will produce K_z samples drawn from the inference distribution $q(z|x_n)$. These samples z_i will then pass through neural network functions to obtain the statistics of the generative distribution $p(x_n|z_i)$. Note however that due to the sampling, each x_n is paired against K_z sets of statistics. If the distribution of x is characterized by a mean, for example, then there will be K_z means $\mu_{px}(z_i)$ paired against each x_n, thus each x_n needs to be replicated K_z times in order to obtain all of the necessary probabilities. To easily handle the replication, we introduce tensors.

First define the shapes of each tensor, beginning by collecting the set of N data points x_n into a third-order tensor X. Equation 4.18 shows the definition of the shape of X.

$$X : [N, 1, D_x] \tag{4.18}$$

The third-order tensor X has three axes, as will all other tensors in the implementation of M1. The first axis of X has dimension N, the second has dimension one, and the third has dimension D_x, and in general, we will call the first axis Axis N, the second axis Axis Z, and the third Axis D. Ordinarily, X would be a second-order tensor (more commonly known as a matrix) with dimensions N by D_x, but here we promote the order by inserting a new axis into the tensor. The set of points X passes through neural network functions to create the statistics of Z.

$$\mu_{qz}(X) : [N, 1, D_z] \tag{4.19}$$

$$\sigma_{qz}(X) : [N, 1, D_z] \tag{4.20}$$

Note that each x_n generates exactly one $\mu_{qz}(x_n)$ and $\sigma_{qz}(x_n)$. It is not until after the noise e_{zi} is added that z_i are created and replicated. Collect all of the noise into a tensor E_z.

$$E_z : [N, K_z, D_z] \tag{4.21}$$

The set of latent points drawn from $q(z|X)$, or more precisely transformed from E_z, are stored in the tensor Z.

$$Z : [N, K_z, D_z] \tag{4.22}$$

Statistics of X are obtained by passing Z through the appropriate neural network functions.

When implementing M1, only tensors X and E_z need to be explicitly defined. All others will be shaped appropriately based on the rules of tensor *broadcasting*. Two broadcasting rules are required to implement DGMs.

1. Operations that require two tensors of compatible shape allow for any axis of dimension one to be replicated along that axis.

2. Operations that require two tensors of compatible shape allow lower-order tensors to be promoted to higher-order tensors by adding axes of dimension one to the left.

The first rule comes into play when the statistics of a distribution are used to transform samples from another distribution. In M1, the (element-by-element) multiplication of $\exp \sigma_{qz}(X)$ and E_z followed by the addition of $\mu_{qz}(X)$ creates the replicated set Z.

$$Z = \mu_{qz}(X) + E_z \exp \sigma_{qz}(X) \tag{4.23}$$

Recall that both $\mu_{qz}(X)$ and $\sigma_{qz}(X)$ have shape $[N, 1, D_z]$ but the noise E_z and the resulting Z have shape $[N, K_z, D_z]$.

The second rule is used whenever a tensor passes through a neural network function. Each layer of a neural network has input X, output Y, weight matrix W, bias vector b, and activation function s.

$$Y = s(X \cdot W + b) \tag{4.24}$$

The dot-product $X \cdot W$ would be ordinary matrix multiplication if both X and W were matrices. Thinking of them as second-order tensors, if X is N by D_x and W is D_x by D_z, then the dot product's result is N by D_z. It transforms the dimension of the last axis of X into the dimension of the last axis of W. The general tensor dot-product is the same when W remains a matrix – the dot-product transforms the dimension of the last axis of X from D_x to D_z.

$$X : [N, 1, D_x] \tag{4.25}$$

$$W : [D_x, D_z] \tag{4.26}$$

$$X \cdot W : [N, 1, D_z] \tag{4.27}$$

The bias vector b must have D_z elements in order to be of the appropriate dimension. However, when it is added to the dot-product result, it is broadcasted into higher-order tensor based on the second broadcasting

rule.

$$b : [1, 1, D_z] \tag{4.28}$$

$$X \cdot W + b : [N, 1, D_z] \tag{4.29}$$

The activation s is applied element-by-element typically, but in special cases such as for the softmax activation, it may perform some computations on the *last* axis (axis farthest to the right) as a whole.

The implementation of M1 can be completed with these two broadcasting rules in mind. The tensor X is given, and the tensor E_z is filled with samples from a standard normal distribution. Given those two tensors and the neural network functions μ_{qz} and σ_{qz}, Z can be calculated from Equation 4.23. Then, supposing that the data X has a distribution characterized by a mean function, obtain the mean of X given Z by passing Z through an appropriate neural network function $\mu_{px}(Z)$.

$$X : [N, 1, D_x] \tag{4.30}$$

$$E_z \sim \mathcal{N}([N, K_z, D_z]) \tag{4.31}$$

$$Z = \mu_{qz}(X) + E_z \exp \sigma_{qz}(X) : [N, K_z, D_z] \tag{4.32}$$

$$\mu_{px}(Z) : [N, 1, D_x] \tag{4.33}$$

All that remains is the reduction of the tensors down to a scalar objective.

The objective shown in Equation 4.13 can have all K_z values of $J(x, z_i)$ calculated simultaneously using tensors. There are three terms that need to be computed in Equation 4.14 and summed together.

$$J(X, Z) = \log q(Z|X) - \log p(X|Z) - \log p(Z) \tag{4.34}$$

Consider first $\log p(Z)$. For M1, the prior distribution of Z in the generative model is standard normal.

$$\log p(Z) = -\frac{1}{2} \sum_{\text{Axis D}} \left(\log 2\pi + Z^2 \right) \tag{4.35}$$

$$= \text{loggauss}(Z)$$

$$\log p(Z) : [N, K_z, 1] \tag{4.36}$$

The log probability $p(Z)$ is itself a tensor, but the last axis has been summed over so that it has dimension one. The log probability $q(Z|X)$ has the same shape and similarly has a sum over Axis D.

$$\log q(Z|X) = -\frac{1}{2} \sum_{\text{Axis D}} \left(\log 2\pi + \sigma_{qz}(X) + \frac{(Z - \mu_{qz}(X))^2}{\exp 2\sigma_{qz}(X)} \right) \tag{4.37}$$

$$= \text{loggauss}(Z, \mu_{qz}(X), \sigma_{qz}(X)) \tag{4.38}$$

$$\log q(Z|X) : [N, K_z, 1] \tag{4.39}$$

The final probability, $\log p(X|Z)$, also is a tensor with shape $[N, K_z, 1]$ and is computed appropriately based on the statistics of X. Summing these three tensors yields Equation 4.34, and collapsing to a scalar simply involves averaging over Axis Z and Axis N.

$$J(X) = \frac{1}{NK_z} \sum_{\text{Axis N}} \sum_{\text{Axis Z}} J(X, Z) \tag{4.40}$$

$$J(X) : [1, 1, 1] \tag{4.41}$$

This third-order tensor is in reality a single number, and automatic differentiation tools such as Theano can compute its gradients.

4.3 Complex DGMs

Any model more complex than M1 will require higher-order tensors. M2 utilizes fourth-order tensors, and SDGM needs fifth-order tensors. The general formula is that the order of the tensor needed for a DGM is one plus the number of variables in the model. SDGM has X, Y, Z, and A as variables, and a batch of N data points is processed simultaneously, hence the necessity for a fifth-order tensor.

In addition to higher-order tensors, more complex DGMs will require neural networks with multiple inputs. Consider, for example, the neural network function in SDGM that generates the mean of Z in the inference network, $\mu_{qz}(X, Y, A)$. A single point x, a single one-hot class label y, and a single auxiliary variable a require the input to μ_{qz} to be a vector of length $D_x + D_y + D_a$ whose contents are the concatenation of the individual vectors. According to Equation 4.24, the input vector needs to be dotted against the first layer's weight matrix W which has dimensions D_{in} by D_{out}. A block structure emerges from the expansion of the

dot product.

$$x : [1, D_x] \tag{4.42}$$

$$y : [1, D_y] \tag{4.43}$$

$$a : [1, D_a] \tag{4.44}$$

$$W : [D_{in}, D_{out}] \tag{4.45}$$

$$(x, y, a) : [1, D_x + D_y + D_z] = [1, D_{in}] \tag{4.46}$$

$$(x, y, a) \cdot W : [1, D_{out}] \tag{4.47}$$

$$(x, y, a) \cdot W = x \cdot W_x + y \cdot W_y + a \cdot W_a \tag{4.48}$$

The matrices W_x, W_y, and W_a form W when stacked vertically. This block structure allows *tensors* to be replicated appropriately when serving as inputs to a neural network function. Consider the following case.

$$X : [N, 1, 1, 1, D_x] \tag{4.49}$$

$$Y : [1, 1, 1, D_y, D_y] \tag{4.50}$$

$$A : [N, K_a, 1, 1, D_a] \tag{4.51}$$

$$X \cdot W_x + Y \cdot W_y + A \cdot W_a : [N, K_a, 1, D_y, D_{out}] \tag{4.52}$$

Notice that the output has the correct dimensions along all axes. After the first layer, all other layers behave normally, and any number of layers can be stacked to form deeper networks.

M2 and SDGM are semi-supervised DGMs, so they have three sub-objectives: $J_l(X_l, Y_l)$ for labeled data, $J_u(X_u)$ for unlabeled data, and $J_a(X_l, Y_l)$ to ensure that the networks that are shared between J_l and J_u are properly linked. The objective $J_l(X_l, Y_l)$ is almost identical to the objective from M1 with the exception that the neural network functions have multiple inputs. The objective $J_u(X_u)$ requires replication across all possible classes because the true labels are not know for unlabeled data. $J_a(X_l, Y_l)$ is a simple neural network classifier objective. We will describe how to compute the objectives for SDGM, because, as a model, M2 is a subset of SDGM, so its details are easy to extract from the description of SDGM.

For SDGM's J_l, first define all of the necessary tensors. SDGM uses fifth-order tensors with axes labeled Axis N, Axis A, Axis Z, Axis Y, and Axis D. The labeled data X_l and Y_l in each minibatch are given, and Y_l is a class label encoded as one-hot with D_y total classes. We assume that both A and Z are Gaussian variables with means and log-deviations given by neural network functions. To approximate the expectations, draw K_a samples of A and K_z samples of Z. As before, scale and shift noise E_{al} and E_{zl} from standard

normal to Gaussians with the desired means and variances of A_l and Z_l.

$$X_l : [N_l, 1, 1, 1, D_x] \tag{4.53}$$

$$Y_l : [N_l, 1, 1, 1, D_y] \tag{4.54}$$

$$E_{al} : [N_l, K_a, 1, 1, D_a] \tag{4.55}$$

$$E_{zl} : [N_l, K_a, K_z, 1, D_z] \tag{4.56}$$

With these defined, obtaining samples of A_l and Z_l is straightforward. In the following, we notate μ_{qa} as a neural network function and μ_{qal} as a tensor that has been obtained by passing X_l through μ_{qa}.

$$\mu_{qal} = \mu_{qa}(X_l) \tag{4.57}$$

$$\sigma_{qal} = \sigma_{qa}(X_l) \tag{4.58}$$

$$A_l = \mu_{qal} + E_{al}\sigma_{qal} \tag{4.59}$$

$$\mu_{qzl} = \mu_{qz}(X_l, Y_l, A_l) \tag{4.60}$$

$$\sigma_{qzl} = \sigma_{qz}(X_l, Y_l, A_l) \tag{4.61}$$

$$Z_l = \mu_{qzl} + E_{zl}\sigma_{qzl} \tag{4.62}$$

At this point, all of the latent variables are inferred, and all of the statistics necessary to compute the inference log-probabilities have been obtained.

$$\log q(A_l|X_l) = \text{loggauss}(A_l, \mu_{qal}, \sigma_{qal}) \tag{4.63}$$

$$\log q(Z_l|X_l, Y_l, A_l) = \text{loggauss}(Z_l, \mu_{qzl}, \sigma_{qzl}) \tag{4.64}$$

Compute μ_{pal}, σ_{pal}, and any statistics necessary for $\log p(Y_l)$ and $\log p(X_l)$ on the generative side.

$$\mu_{pal} = \mu_{pa}(Y_l, Z_l) \tag{4.65}$$

$$\sigma_{pal} = \sigma_{pa}(Y_l, Z_l) \tag{4.66}$$

$$\log p(A_l|Y_l, Z_l) = \text{loggauss}(A_l, \mu_{pal}, \sigma_{pal}) \tag{4.67}$$

$$\log p(Z_l) = \text{loggauss}(Z_l) \tag{4.68}$$

With these in hand, the objective is easy to express. For the sake of notational brevity, we drop the dependencies because they can be inferred from the graphical model – that is, use $q(Z_l)$ instead of $q(Z_l|X_l, Y_l, A_l)$.

$$\begin{aligned} J_l(X, Y, Z, A) = &\log q(A_l) + \log q(Z_l) \\ &- \log p(A_l) - \log p(X_l) \\ &- \log p(Y_l) - \log p(Z_l) \end{aligned} \tag{4.69}$$

$$J_l = \frac{1}{NK_aK_z} \sum_{\text{Axis N}} \sum_{\text{Axis A}} \sum_{\text{Axis Z}} J_l(X, Y, Z, A) \tag{4.70}$$

In J_l, the class labels Y_l are known, so always the dimension of Axis Y is one, but in J_u, however, the dimension will be D_y because all possible classes need to be assessed. The tensor Y_u is in fact a normal D_y by D_y identity matrix that has been promoted to a fifth-order tensor by adding axes of dimension one to the left.

$$X_u : [N_u, 1, 1, 1, D_x] \tag{4.71}$$

$$Y_u : [1, 1, 1, D_y, D_y] \tag{4.72}$$

$$E_{au} : [N_u, K_a, 1, 1, D_a] \tag{4.73}$$

$$E_{zu} : [N_u, K_a, K_z, 1, D_z] \tag{4.74}$$

Note that the tensor Y_u has dimension one in Axis N because the same set of D_y possibilities is tested against all N_u data points, so the identity matrix that created Y_u will be replicated N_u times due to the tensor broadcasting rules as soon as an operation requires it. Furthermore, an expectation over Axis Y must be performed, so the results of the D_y possible classes must be weighted according to the likelihood of each class. The likelihoods of all classes are given as the output of a softmax neural network π_{qy}.

$$\pi^T_{qyu} = \pi_{qy}(X_u, A_u) \tag{4.75}$$

$$\pi^T_{qyu} : [N_u, K_a, 1, 1, D_y] \tag{4.76}$$

$$\pi : [N_u, K_a, 1, D_y, 1] \tag{4.77}$$

In order to obtain π from the output π^T_{qyu}, the last two axes need to be transposed. (Theano calls this operation a dimshuffle.) The remainder of J_u proceeds the same as J_l.

$$\mu_{qau} = \mu_{qa}(X_u) \tag{4.78}$$

$$\sigma_{qau} = \sigma_{qa}(X_u) \tag{4.79}$$

$$A_u = \mu_{qau} + E_{au}\sigma_{qau} \tag{4.80}$$

$$\mu_{qzu} = \mu_{qz}(X_u, Y_u, A_u) \tag{4.81}$$

$$\sigma_{qzu} = \sigma_{qz}(X_u, Y_u, A_u) \tag{4.82}$$

$$Z_u = \mu_{qzu} + E_{zu}\sigma_{qzu} \tag{4.83}$$

Note that the neural network functions μ_{qz}, σ_{qz}, μ_{qa}, and σ_{qa} are the same as the ones used to compute J_l; this is what allows semi-supervised learning. The log probabilities of $q(A_u)$, $q(Z_u)$, $p(A_u)$, $p(Z_u)$, $p(X_u)$ are obtained in a similar fashion to their labeled-data counterparts, finally resulting in $J_u(X, Y, Z, A)$ and J_u.

$$\begin{aligned} J_u(X, Y, Z, A) = &\log q(A_u) + \log q(Z_u) + \log q(Y_u) \\ &- \log p(A_u) - \log p(X_u) \\ &- \log p(Y_u) - \log p(Z_u) \end{aligned} \tag{4.84}$$

$$J_u = \frac{1}{NK_aK_z} \sum_{\text{Axis N}} \sum_{\text{Axis A}} \sum_{\text{Axis Y}} \sum_{\text{Axis Z}} \pi J_u(X, Y, Z, A) \tag{4.85}$$

In both J_l and J_u, the log-probability $\log p(Y)$ is simply $-\log D_y$. In J_l, $\log q(Y_l) = 0$ because Y_l is known, but in J_u, $\log q(Y_u) = \log \pi$. Finally, the distribution of $\log p(X)$ depends on the characteristics of the input data X.

The last sub-objective is the categorical cross-entropy J_a whose purpose is to ensure that the network π_{qy} is trained using labeled data. This reuses the computation of A_l along with the given labeled data X_l and Y_l. Remember that Y_l is represented with a one-hot encoding, so multiplying the probabilities π^T_{qyl} by Y_l and summing over Axis D will select the probability associated with the correct class label.

$$\pi^T_{qyl} = \pi_{qy}(X_l, A_l) \tag{4.86}$$

$$J_a(X_l, Y_l, A_l) = \log \sum_{\text{Axis D}} Y_l \pi^T_{qyl} \tag{4.87}$$

There is one class prediction for each of N data points and for each of K_a latent variable samples, so the corresponding axes need to be averaged in order to obtain a scalar objective function.

$$J_a = \frac{1}{N_l K_a} \sum_{\text{Axis N}} \sum_{\text{Axis A}} J_a(X_l, Y_l, A_l) \tag{4.88}$$

The final objective function for SDGM is J. The weight α is a reflection of how much labeled data versus unlabeled data is available. For further details on α, refer to [44].

$$J = J_l + J_u + \alpha J_a \tag{4.89}$$

As was the case with M1, this objective is a scalar (or a tensor with all axes' dimensions equal to one), and its gradients with respect to the parameters of the neural networks used to generate all ps and qs can be obtained via automatic differentiation.

4.4 Visualizations of Existing Models

The simplest DGM is M1, an unsupervised model that has a single latent variable Z in addition to the observed data X. Because M1 is unsupervised, no class labels are used in its training. To understand what it does, we train the model with the latent variable Z designed to exist in a two-dimensional space $D_z = 2$ so that it may be visualized. Each data point x results in an entire distribution $q(z|x)$ that is chosen to be Gaussian with mean $\mu_{qz}(x)$ and standard deviation $\sigma_{qz}(x)$ obtained as the outputs of neural network functions of x. To visualize the model, we plot the means $\mu_{qz}(x)$ in the two-dimensional latent Z space of all 60,000 MNIST training data points and color each point based on its true class label, even though the label was never used in training. This plot is shown in Figure 4.3.

In addition to the scatter plot of latent means, we show random selections of nine actual images x that fall into different regions of the latent space. To refer to specific bundles of nine images, we give coordinates in square brackets

[C,R]

that index columns from the left and rows from the top such that, for example, the portion of the zeros closest to the center of the Z space encompasses [d,3] and [e,3]. The cell referred to as [d,3] in image bundle coordinates is located approximately at $(-0.5, 2)$ in the Z space.

Figure 4.3 shows many regions of pure, non-overlapping color, indicating that a large portion of the latent Z space is dedicated to separating the data by class. Zeros and ones form large, well-separated clusters at

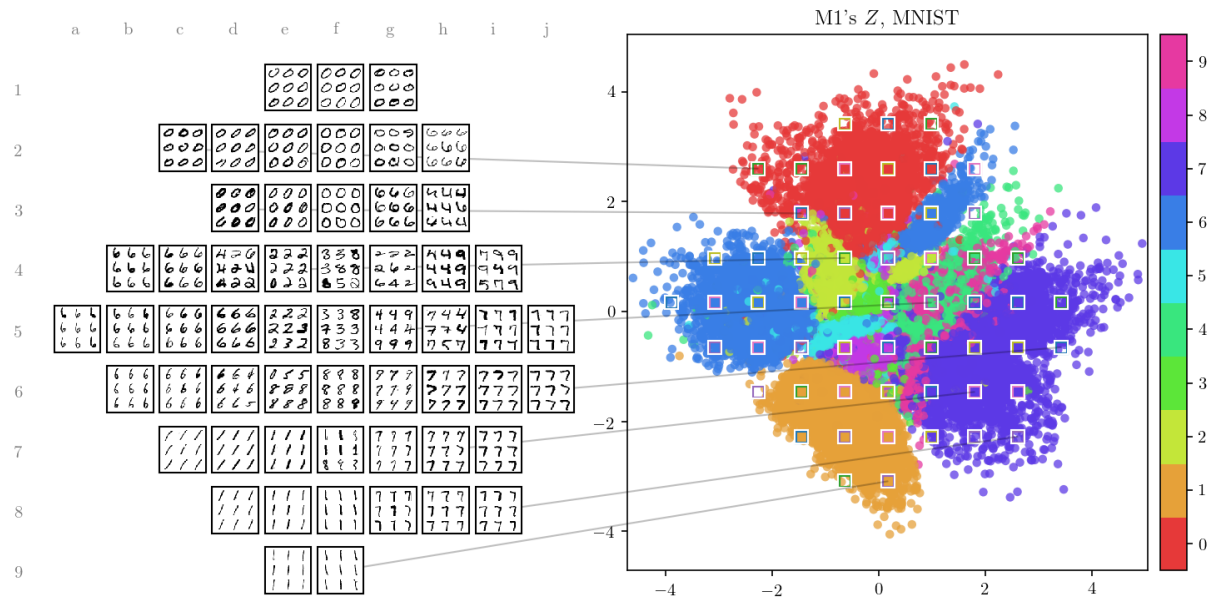


Figure 4.3: Visualization of M1's latent space Z when trained on MNIST.

the top and bottom of the space, respectively. Also, some classes, such as twos and sixes, are split into two distinct clusters. The sixes in particular take up a lot of space on the left side of the Z space, but a sizable cluster of them forms at [g,3] located near $(1, 2)$ in the Z space. Comparing [g,3] to [h,3], we see that the sixes in [g,3] are fairly wide and differ from some of the fours that appear in [h,3] only because they are missing a few pixels that would form the handle of the character four. Similarly, the sixes at [g,3] could almost appear to be zeros at [f,3] if a few pixels were removed from the part of the character six that extends upward or if a few pixels were added to close the extension.

By comparison, Figure 4.4 shows an inseparable mixture of colors, indicating that the learned space carries no information about class label. Figure 4.4 is the visualization of M2's latent Z space when all of the class labels Y are known. Again for the sake of visualization, the latent space Z is designed to be two-dimensional with $D_z = 2$, and the distribution of Z is chosen to be Gaussian with mean $\mu_{qz}(x, y)$ and standard deviation $\sigma_{qz}(x, y)$. Like before, the scatter plot shows the means of every latent z corresponding to each training data point (x, y) colored by the true class label. Because the class labels are known, the Z space captures information that relates to the *styles* of each handwritten digit. For example, the lower-left regions near [b,7] and [b,8] show collections of digits which are very thin. By comparison, the upper-right region near [h,2] shows digits that are very wide. Furthermore, the lower-right region captures digits that are heavily slanted to the right as if italicized, and emboldened digits appear near the center around [f,4].

Figure 4.5 shows visualizations of SDGM's auxiliary A and latent Z spaces. In this case, both A and Z are designed with dimensions $D_a = D_z = 2$. Again, the variables are chosen to be Gaussian distributed.

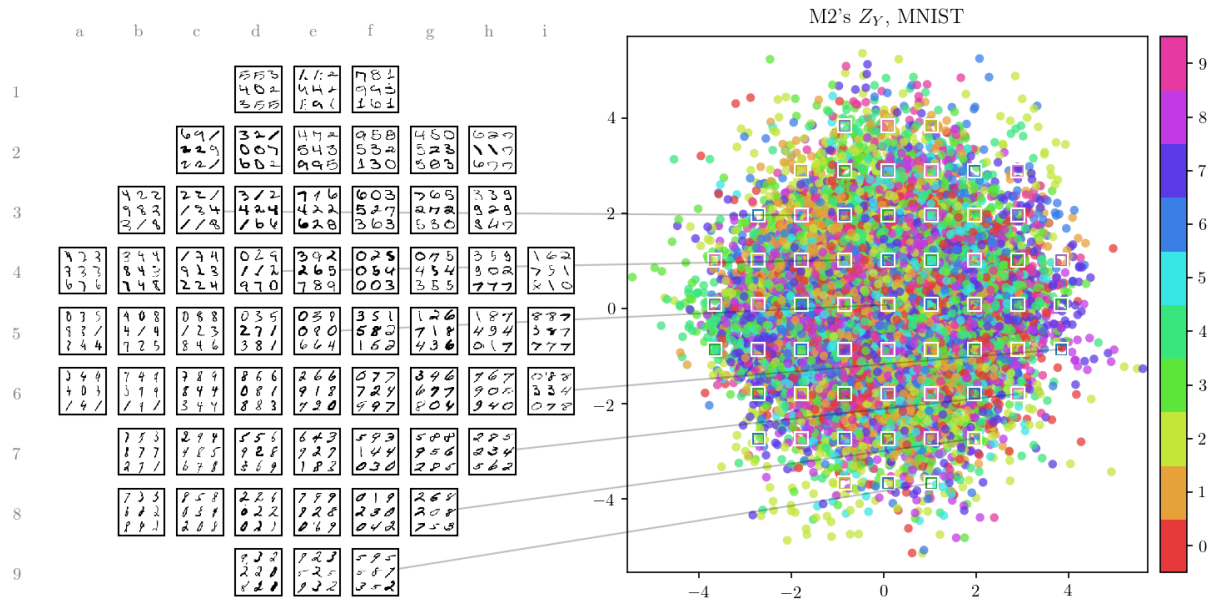
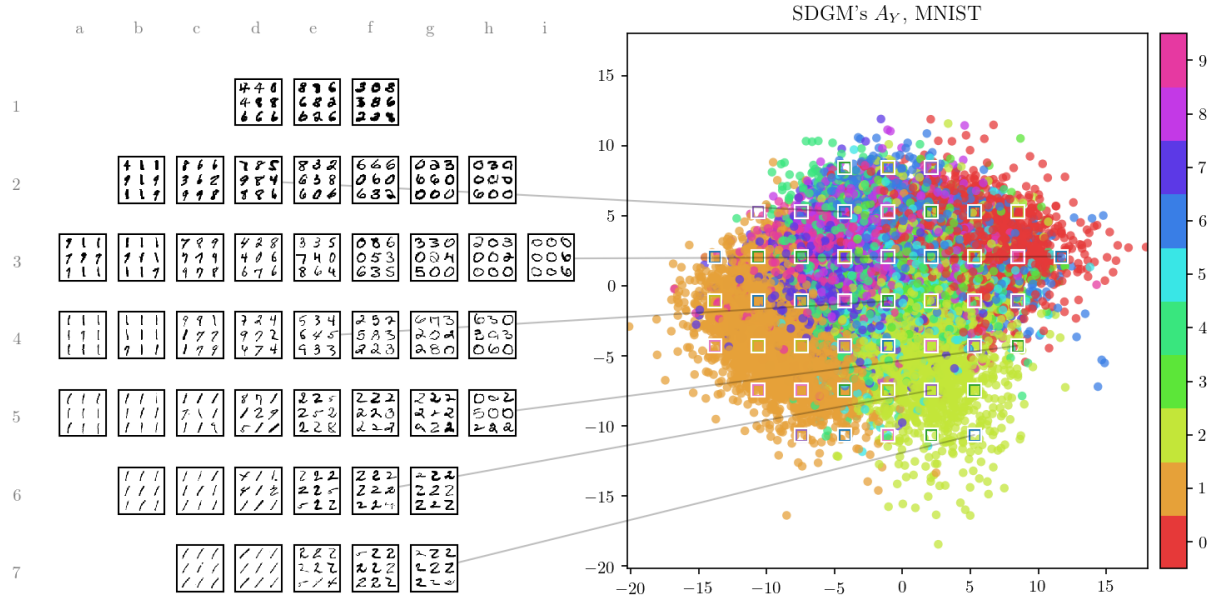


Figure 4.4: Visualization of M2's latent space Z when trained on MNIST.

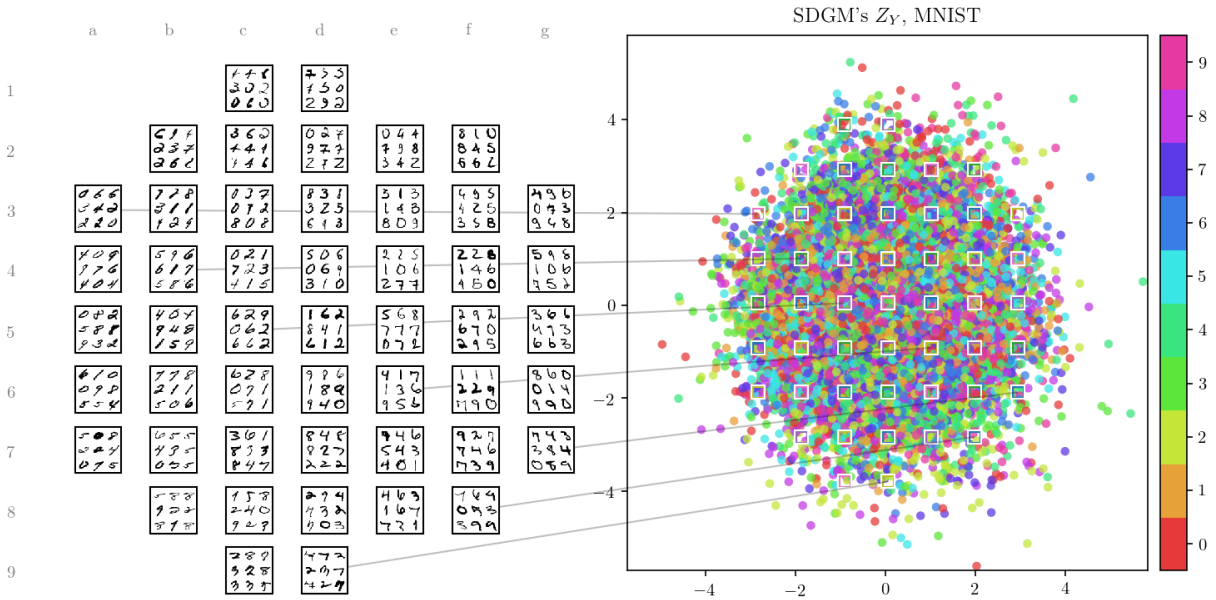
Their mean functions, visualized for every data point x, are $\mu_{qa}(x)$ and $\mu_{qz}(a, x, \hat{y})$, respectively, where $a = \mu_{qa}(x)$ and $\hat{y} = \text{argmax}\ \mu_{qy}(a, x)$. That is, a data point x obtains a single value a (located at the mean of its distribution), both of those predict the class label $\hat{y}$, and three obtain the mean of the latent variable z given the data point x.

Several interesting behaviors can be understood based on Figure 4.5(a). The visualization of the auxiliary variable A shows a large portion of the space is reserved for ones. We believe this is because ones that are slanted to the right and ones that are more plumb share very few pixels in common, but the auxiliary variable captures the fact that a continuous distribution of slants is present in the observed data X, and therefore A links the two extreme variations. Additionally, there is a concentrated region of emboldened digits, indicating that removing the thickness of a handwritten digit helps to classify it. Both of these behaviors cement the original authors' [59] intuition that the auxiliary variable A captures rich properties of the data that are useful for classification.

Several additional interesting behaviors can be understood based on Figure 4.5(b), a visualization of SDGM's latent space Z. The visualization is visually identical to M2's Z, shown in Figure 4.4. Upon further inspection of the image bundles, however, some minor differences between the two. The primary difference is that SDGM's Z captures the slant of digits but shows no real clustering of digit thicknesses. This is because, as noted previously, a lot of a digit's thickness is captured in SDGM's A with the explanation that removing a digit's thickness emphasizes structural information that is useful for predicting the class label.



(a)



(b)

Figure 4.5: Visualizations of SDGM's latent spaces (a) A and (b) Z when trained on MNIST.

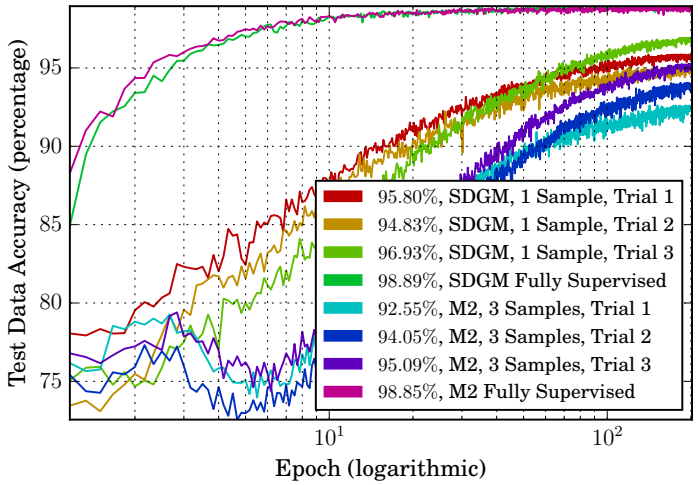


Figure 4.6: Classification performance of M2 and SDGM with 100 labeled examples for each trial. The fully-labeled cases are performance upper bounds.

4.5 Classifier Performances

Switching gears, let us examine the classifier performances that are depicted in Figure 4.6 through Figure 4.10. First, to serve as baseline performances, M2 and SDGM are trained on the same data set but with different selections of the 100 labeled images, always with the requirement of ten labeled examples from each of the ten digits. In these trials, the latent variable Z has dimension $D_z = 100$, and the auxiliary variable A also has dimension $D_a = 100$ in SDGM. All of the neural network functions have two hidden layers of size 500 and use the rectifying linear units (ReLUs) as their hidden layer nonlinearities. Each iteration of training uses $N_l = 100$ labeled data points and $N_u = 100$ unlabeled data points per minibatch, and training lasts a total of 200 epochs, where one epoch is a pass through all 60,000 training images. As expected, SDGM outperforms M2, and both sets of classification performances are reported for three separate trials in Figure 4.6. The number of samples in the legend of Figure 4.6 tells how many samples were used to approximate the expectations shown in Equation 4.70 and Equation 4.85.

We report three trials' worth of results in Figure 4.6 to provide a sense of the consistency of each algorithm. We find that SDGM almost always achieves 95% or better (often going as high as 97.7% in our experience). Contrary to reports by the original authors of M2 [44], M2 on its own can achieve performance as high as 95% with the worst case at 92.5%, higher than what was originally reported. Curiously, in all three trials, M2's classification performance reaches a minimum after around eight epochs and only then begins to increase. SDGM's performance seems to improve monotonically.

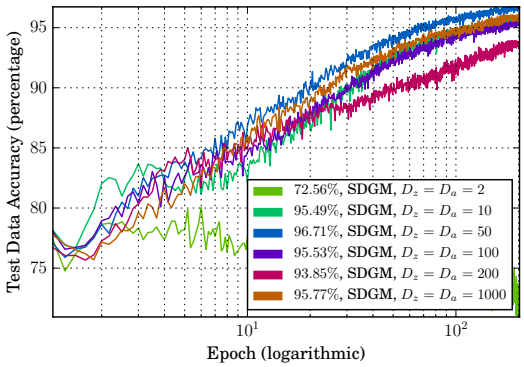


Figure 4.7: Classification performance of SDGM using a fixed seed but varying latent variable dimensions D_a and D_z.

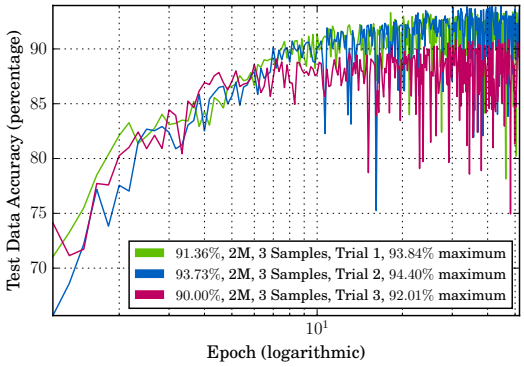


Figure 4.8: Classification performance of 2M. The performance is promising but highly unstable.

To determine whether or not the dimensions of the latent space D_z and auxiliary space D_a affect SDGM's classification performance, we trained SDGM on a fixed set of 100 labeled images and varied D_z and D_a. The results are shown in Figure 4.7. Empirically, it seems that barring the extremely difficult scenario of small dimensions $D_z = D_a = 2$, there is no dependence on the sizes of the spaces.

The model M2 has a latent variable Z that is dependent on the class label Y in the inference network, shown graphically in Figure 4.2. We wanted to know what effect this dependence has on the properties of the model, so we tested two variations – 2M and M2 Split, also shown in Figure 4.2 – over three trials each. Note that all three DGMs, M2, 2M, and M2 Split, have identical generative models P and differ only in their inference models Q. Figure 4.8 shows that reversing the direction of the arrow in Q makes the model very unstable. Surprisingly, the maximum test-data classification accuracies are competitive with the normal M2, but the fit of the model changes wildly on each epoch. We terminated their training regiments after only 50 epochs instead of the usual 200 due to stability. The DGM M2 Split does not have a direct connection between y and z in the inference model Q. Its classification results are presented in Figure 4.9. While M2

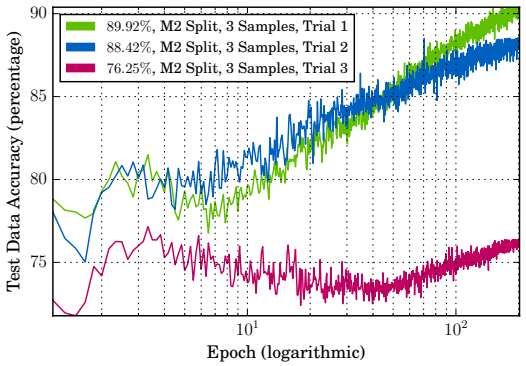


Figure 4.9: Classification performance of M2 Split. Performance is stable but significantly worse than M2 and 2M.

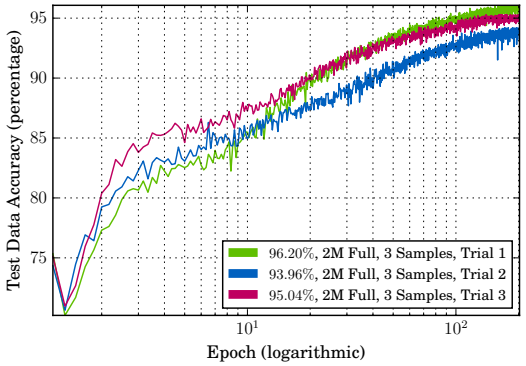


Figure 4.10: Classification performance of 2M Full. The performance is both stable and better than M2, rivaling the more complex model SDGM.

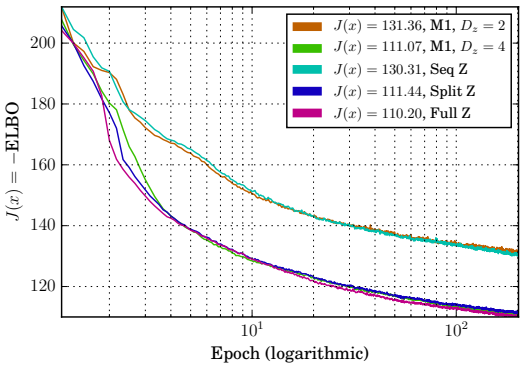


Figure 4.11: Evidence lower bound comparison for different unsupervised models.

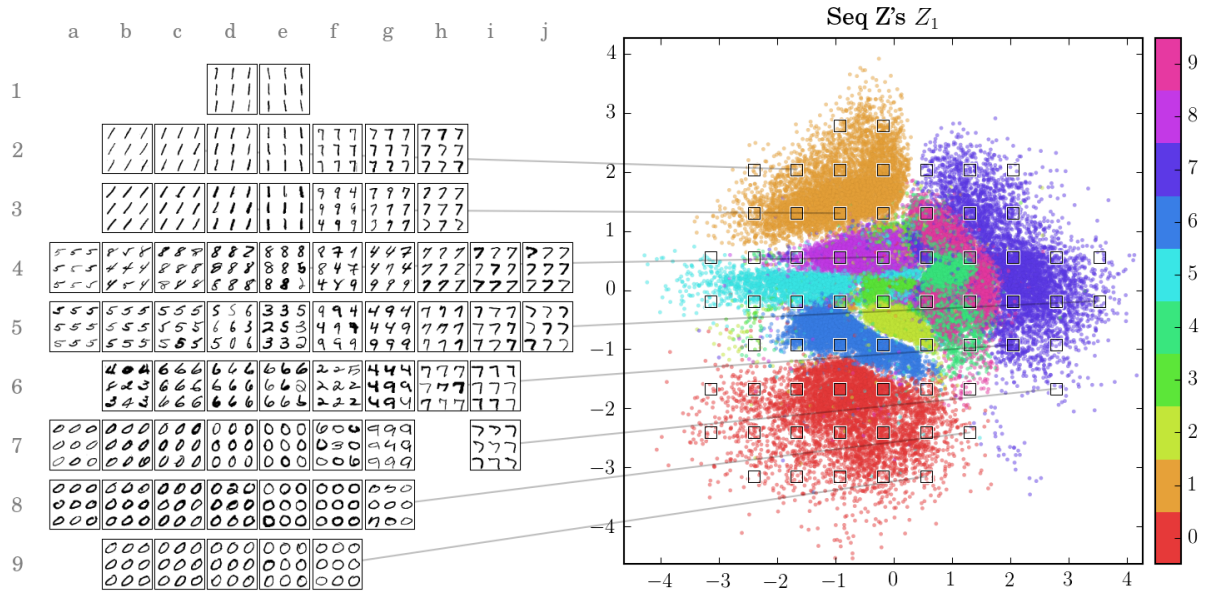
Split is stable in all three trials, its best performance is several percentage points lower than M2 and 2M, indicating that a fully-connected inference model is more powerful.

The model 2M can be interpreted as SDGM with its latent variable Z removed; the latent variable of 2M can be thought of as SDGM's auxiliary variable A. In this light, it is not surprising that SDGM's performance is the best of all of the DGMs we examined. SDGM is a combination of two DGMs, M2 and 2M, each of which can achieve good classification results with wildly different learning characteristics. SDGM's performance is very stable, unlike 2M, and the only difference between SDGM and the combination of M2 and 2M is the set of dependencies of 2M's latent variable. Is it possible that linking Y to Z in 2M's generative model can stabilize its performance? In fact, the answer is yes! Not only does linking Y to Z in the generative model of 2M add stability, it also significantly improves classification performance, as seen in Figure 4.10, putting it above M2 and almost on par with SDGM. The graphical model for this configuration is 2M Full in Figure 4.2.

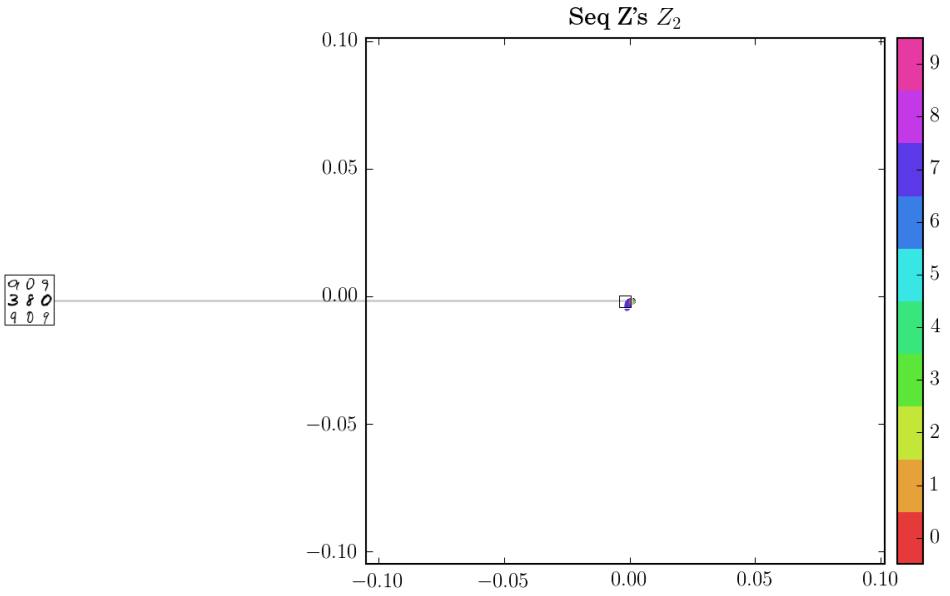
4.6 Analysis of Unsupervised Models

Analogous to the differences between the semi-supervised classifiers M2, 2M, and M2 Split, we tested unsupervised learners Split Z, Sequential Z, and Full Z in order to further identify effects of varying the inference model Q while keeping a fixed generative model P. In order to understand how they operate, we trained them on the completely unlabeled 60,000 images, designed all latent variables Z_1 and Z_2 to be two-dimensional and Gaussian-distributed, plotted the likelihood of unseen test data in Figure 4.11, and visualized the two-dimensional latent spaces in Figure 4.12 to Figure 4.16.

Figure 4.11 shows the objective function J_{UL} evaluated on unseen test data over the course of training five separate DGMs. There are two instances of the M1 model, one of which has its latent space at two dimensions $D_z = 2$ and the other which has $D_z = 4$. The two-dimensional case was previously visualized and analyzed in Figure 4.3. The other three DGMs are Split Z, Sequential Z, and Full Z whose models are



(a)



(b)

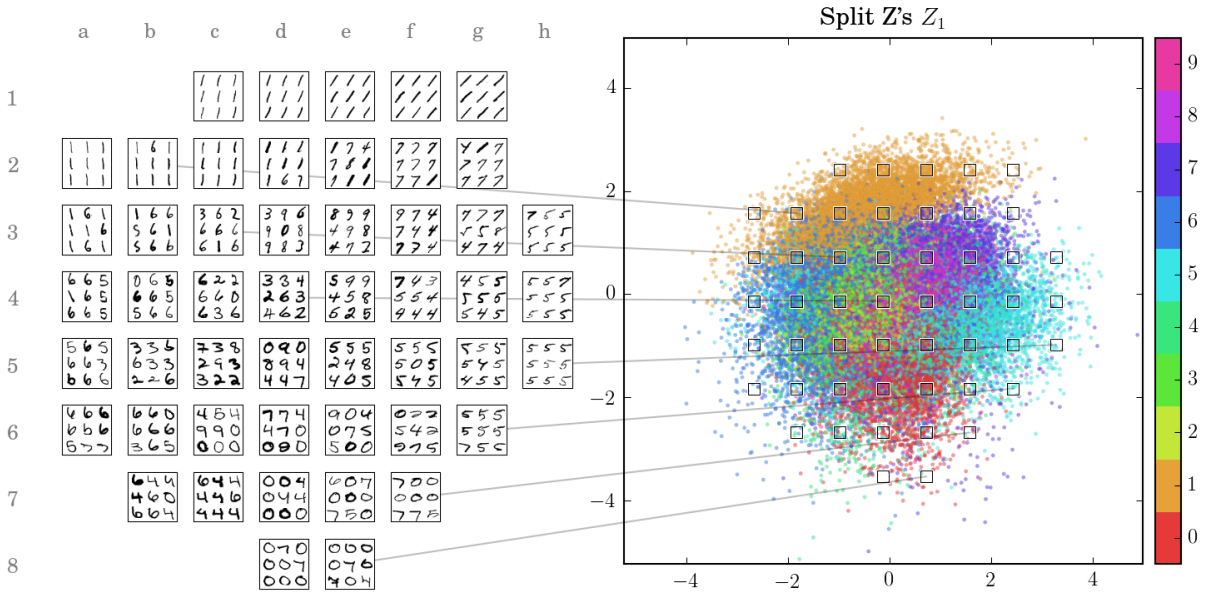
Figure 4.12: Visualizations of Seq Z's latent spaces (a) Z_1 and (b) Z_2 when trained on MNIST.

shown in Figure 4.1. We notice two very tight performance groupings: M1 with $D_z = 2$ is almost identical to Sequential Z, and M1 with $D_z = 4$, Split Z, and Full Z are very similar to each other. Obviously, M1 with $D_z = 4$ performs better than M1 with $D_z = 2$ because of the extra space allocated. The behaviors of the other cases are not so self-evident. Note that M1 with $D_z = 4$ is equivalent to Split Z where two dimensions of M1's latent Z are labeled Z_1 and the other two Z_2.

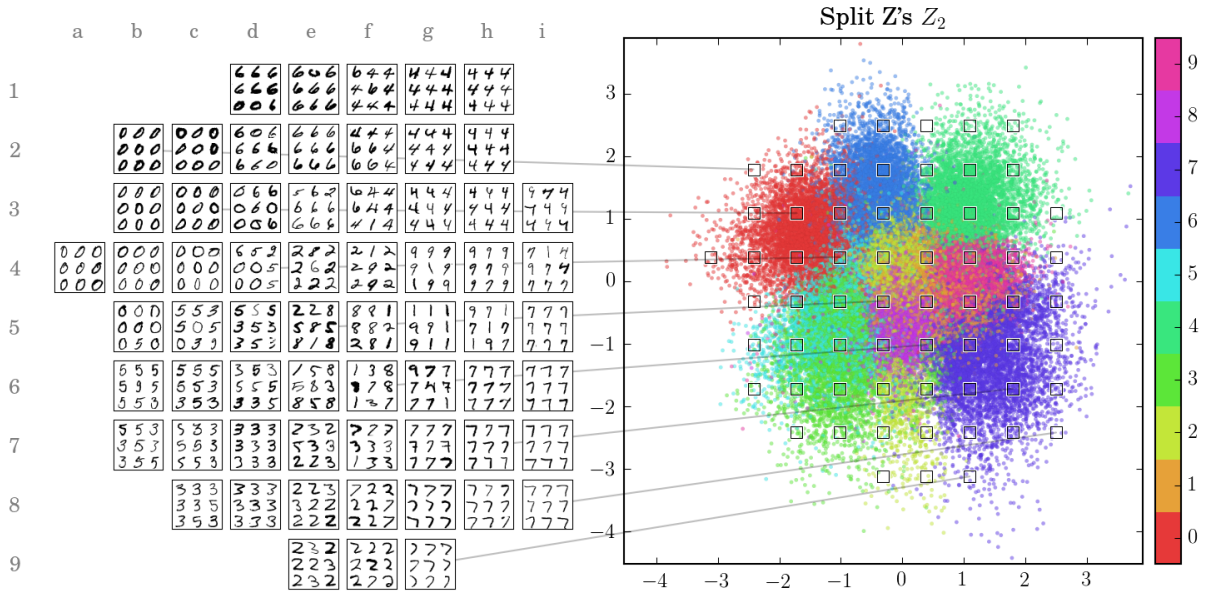
The performances of Seq Z and M1 are indistinguishable when the latent spaces are constrained to two dimensions. Figure 4.12 shows visualizations of the latent spaces of Seq Z. Notice that (a) is basically the same as Figure 4.3 and that (b) is completely collapsed onto the origin of the space. Seq Z uses just Z_1 to explain all of the data, and no information can pass between Z_1 and Z_2. The impermeability of the link between Z_1 and Z_2 may change in higher-dimensional spaces.

The most interesting cases are Split Z and Full Z. Whereas M2 Split and M2 exhibit huge performance differences, here Split Z and Full Z result in no difference. Figure 4.13 visualizes the latent spaces of Split Z, and Figure 4.15 visualizes the latent spaces of Full Z. Figure 4.14 and Figure 4.16 show visualizations of Z_1 given that Z_2 is held constant for Split Z and Z_2 given that Z_1 is held constant for Full Z. Full Z's Z_1 space shows classes that are almost well-separated, except in many cases two classes are intertwined in the same cluster. Full Z's Z_1 at image bundle [d,7] near Z_1-coordinates $(-1, -1.5)$ shows an overlapping cluster of zeros and sixes. Using Z_1 alone is not enough to separate the classes. However, Figure 4.16(a) shows that the data points that end up near $(-1, -1.5)$ in Z_1 are very well-separated in Z_2. This behavior persists almost everywhere. Split Z, similarly, exhibits the same behavior when Z_2 is held fixed and Z_1 is visualized, as can be seen in Figure 4.14.

The roles of Z_2 in Full Z and Z_1 in Split Z are similar to the role of Z in M2. In Figure 4.13(a) and Figure 4.15(b), all of the classes are mixed together, but the image bundles show that the data are ordered by slantedness and width. We find it surprising that even though a direct connection between Z_1 and Z_2 does not exist in Split Z, it still learns the same pattern. Starting from different initializations may reverse the roles of Z_1 and Z_2 in Split Z, but we expect Full Z's explicit dependence to preserve the roles. The fact that Split Z and Full Z learn representations of the data that capture class and style structures is a testament to the power of DGMs.



(a)



(b)

Figure 4.13: Visualizations of Split Z's latent spaces (a) Z_1 and (b) Z_2 when trained on MNIST.

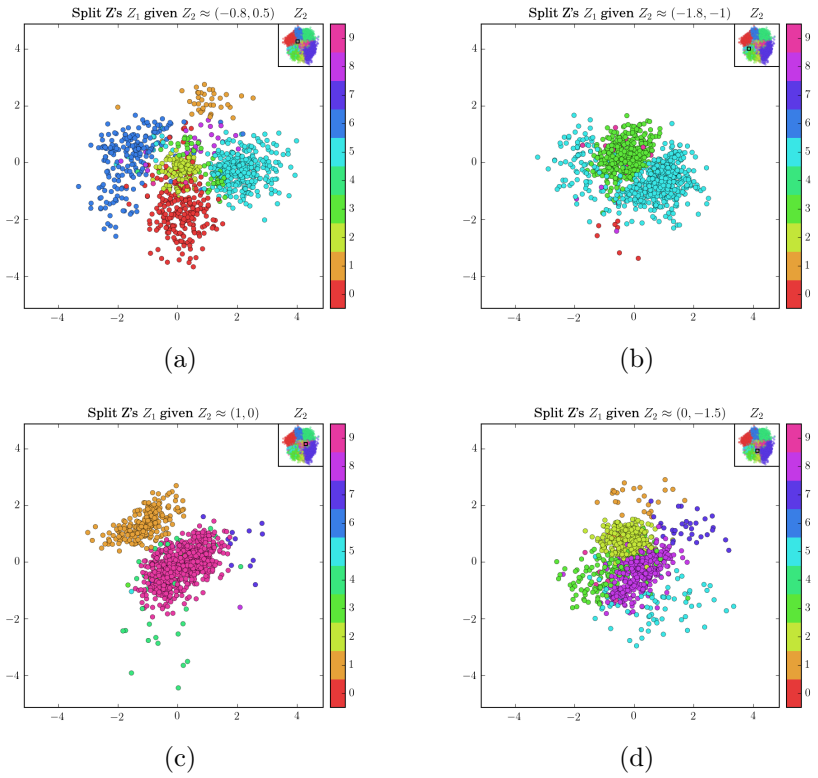
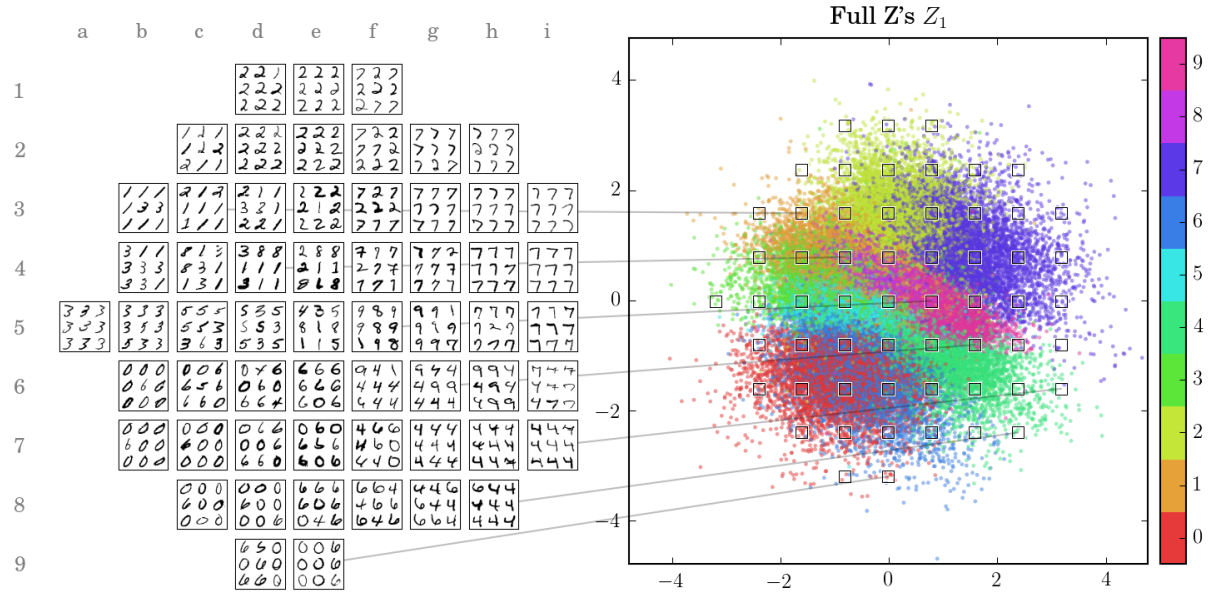
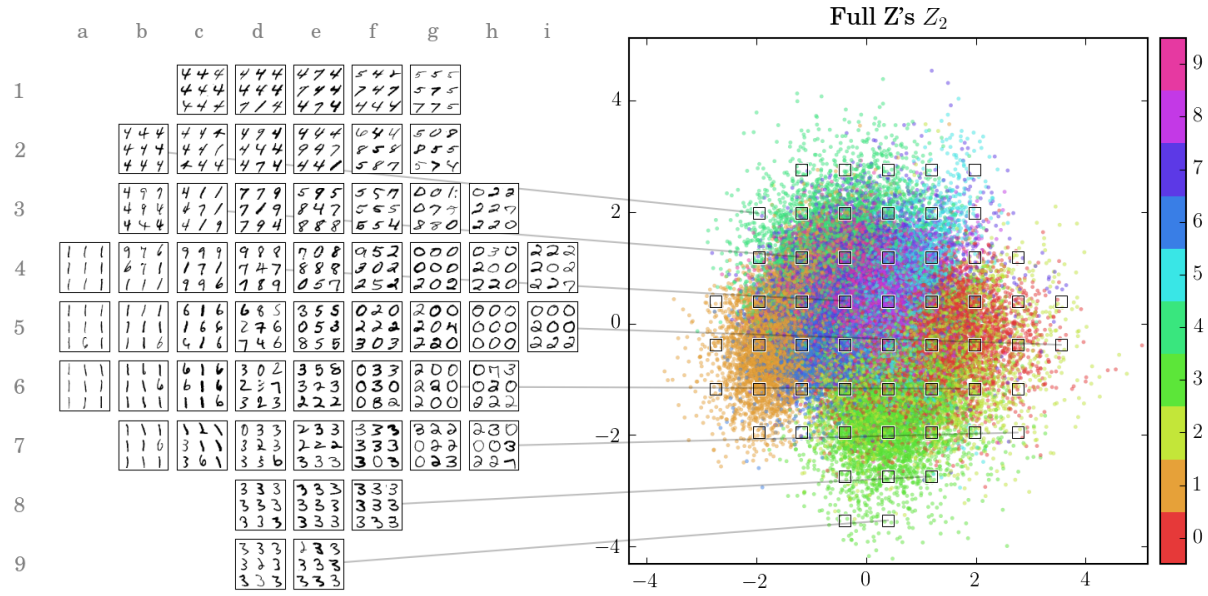


Figure 4.14: Closeups of Split Z's latent space Z_1 given various chosen values of Z_2 when trained on MNIST.

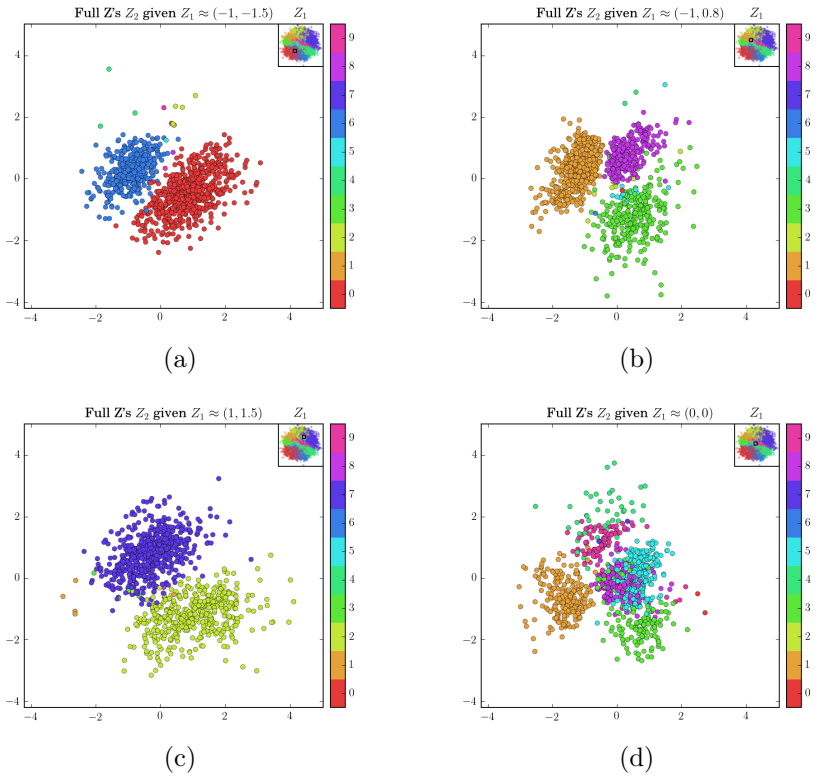


(a)



(b)

Figure 4.15: Visualizations of Full Z's latent spaces (a) Z_1 and (b) Z_2 when trained on MNIST.



(a) (b) (c) (d)

Figure 4.16: Closeups of Full Z's latent space Z_2 given various chosen values of Z_1 when trained on MNIST.

Tensorized Deep Generative Models

The previous chapter detailed how to implement DGMs using tensors, but the purpose of the current chapter is to crystallize the advantage of using the tensorized framework. Here is a comparison between the computation of a single term of M2's objective using standard matrices to the equivalent computation performed with tensors. It will be shown that the matrix version is more difficult and error-prone, and even this simple, single-label model's objective requires 34 equations compared to 20 when using the tensorized framework.

The computations *are possible* using matrices but become incredibly difficult as the size of the model increases. The Stacked model in Chapter 7 would not be feasible to implement without the tensorized framework.

While examining other papers and implementations online [1, 5, 6, 17, 35, 44, 59, 66, 90, 97, 105, 106], we noticed that none of them use tensors or a fully-tensorized framework for variational inference, and most of them either suffer through the repetitions needed with matrices or use poorer approximations to the integrals using a single sample, thus bypassing the need for tensors. (Convolutional models on images use tensors for other purposes – to maintain multi-color data and pixel layout.) It should be noted that the use of a single sample will not work for multi-label models as there is no way to avoid enumeration. Hopefully this chapter shows the simplicity of using the tensorized framework for Deep Generative Models in a neural network toolchain so that future implementations will not need to endure such difficulty.

5.1 Introduction

Deep Generative Models are probabilistic models that contain several neural networks. Inference in them requires calculating expectations over their latent variables, which for the case of continuous variables, take the form of integrals, typically approximated via sampling, and for discrete variables, take the form of enumerated summations. The neural networks each have multiple inputs, and because of the samples and enumerations created in inference, the amount of data present at the input of each network varies significantly, leading to great book-keeping challenges for aligning different input streams.

DGMs combine deep neural networks with probabilistic graphical models. Figure 5.1 shows the graphical models governing three existing DGMs. The nodes in the graphical models have user-specified probability distributions with parameters that need to be learned from data, and the edges that describe dependencies between nodes are neural network functions whose parameters also need to be learned from data. Every DGM has a generative model P and an inference model Q, each with its own set of neural networks.

Graphical models themselves and DGMs can be fit to data using variational inference where one optimizes the evidence lower bound (ELBO) of data given a generative model and an inference model [45, 48]. To train a DGM, one needs machinery capable of training neural networks as well as performing variational inference. One practical approach to merging the two is to use automatic differentiation variational inference (ADVI) [48] or its implementation in a library such as pymc3 [81] and then to add onto it the neural network components necessary to extend to deep learning [95], or even to use a language specifically designed for this combination [91]. ADVI eases the implementation burden for model creators and users, allowing the power of graphical models to reach even wider audiences. While this approach is an interesting direction for future research as the entire solution is rooted in a fully-Bayesian framework, it can be expensive computationally and is, currently, very niche.

Alternatively, one can begin with a well-developed toolchain for handling deep neural networks and then extend it to incorporate graphical models. This approach was taken by the original creators of DGMs [44, 45], others who followed in the same line of work [58, 59], and the framework developed in this chapter. This approach leads to state-of-the-art results, but implementing it can be very difficult. We develop a tensorized framework that, when paired with a modern neural network stack of tools, trivializes the extension to incorporate graphical models.

In practice, a significant difficulty of implementing DGMs revolves around the handling of samples or direct enumeration needed to evaluate expectations in the ELBO. Modern neural network tools, such as Theano [14] and TensorFlow [8], perform automatic differentiation on a symbolic computational graph, and thus a vectorized solution (or in this case, tensorized) is needed to avoid looping so that the compiler can calculate gradients. Existing implementations repeat and reshape matrices, but the amount of coercing grows considerably as the model increases in complexity. For example, [59]'s implementation of the skip deep generative model (SDGM, shown graphically in Figure 5.1) needs 29 reshapes and five repeats for a semi-supervised model that has just one class label and two latent variables.

We describe a tensorized framework that needs *zero* reshapes and repeats, regardless of how many variables are in the model. This is made possible because of NumPy's [93] broadcasting rules for tensors as well as the fact that Theano, TensorFlow, and other modern symbolic computational packages adopt the same rules. In terms of the graph-optimized computational complexity and performance, the tensorized framework is identical to existing implementations. However, the cognitive difficulty faced by model builders is significantly alleviated, thereby making it trivial to explore different model architectures.

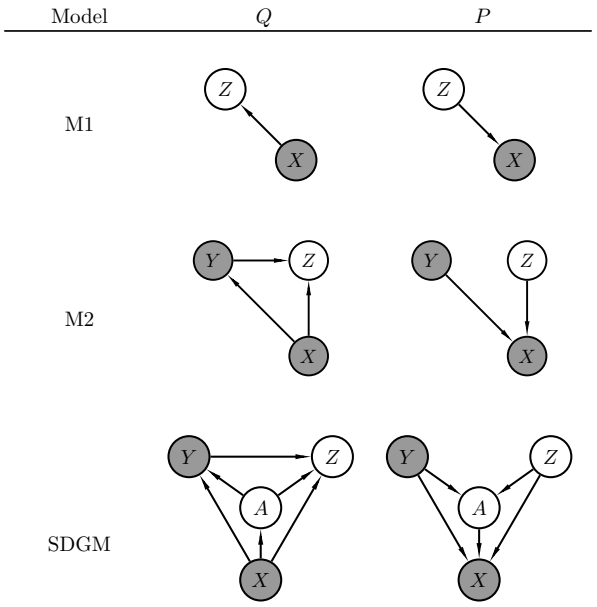


Figure 5.1: Three examples of variational deep generative models showing both the generative network P and the inference network Q.

5.2 Single Term in M2's Objective

Figure 5.1 shows the graphical model of M2, a semi-supervised classifier [44]. The inference distribution is Q, and the generative distribution is P.

$$Q(y, z|x) = q(z|x, y)\ q(y|x) \tag{5.1}$$

$$P(x, y, z) = p(x|y, z)\ p(y)\ p(z) \tag{5.2}$$

The objective function $J(x)$ will contain at least one term which involves $p(x|y, z)$. In the case when the class label y is unknown for a given data point x, then all possible class labels are evaluated and then weighted according to their likelihoods. This leads to an expectation that is computed as a summation over all possible y. Focusing on the specific term that contains $p(x|y, z)$, we get the sub-objective $J_{u,x}(x)$ that must be computed.

$$J_{u,x}(x) = \sum_{y} \sum_{z}\ p(x|y, z)\ q(y|x) \tag{5.3}$$

Note that because z is required to evaluate $p(x|y, z)$, samples of z need to be drawn from the distribution $q(z|x, y)$.

Suppose there are N data points in a batch, each data point has dimension D_x, there are K_y possible classes, and K_z samples of z, each of dimension D_z, are drawn simultaneously. We assume that z is a Gaussian-distributed latent variable that requires a mean and a standard deviation and that x contains vectors of Bernoulli random variables, thus its distribution is parameterized by just a mean vector. (This is a configuration for the MNIST dataset.)

In the following, we will describe the details involved in evaluating Equation 5.3 by coercing all calculations into matrices using stacking and reshaping functions, and then we will show how to obtain the same using our tensorized framework. Both of the detailed procedures are written independently such that each procedure is complete on its own and makes no reference to the other procedure. A direct comparison of the two procedures comes afterward.

5.2.1 Matrix Coercion

We begin with a batch of data X which consists of N points each with dimension D_x. The colon indicates the dimensions of the matrix.

$$X : [N, D_x] \tag{5.4}$$

Each of these points needs to be paired against K_y possible class labels. The matrix $Y = \text{eye}(K_y)$ is the set of all possible class labels in one-hot form.

$$Y : [K_y, K_y] \tag{5.5}$$

To obtain Z, we first need to obtain the mean and standard deviation of Z

$$(\mu_z, \sigma_z) = X_z \cdot W_{xz} + Y_z \cdot W_{yz} \tag{5.6}$$

where W_{xz} is a matrix simplification of a neural network that converts D_x into D_z, and W_{yz} converts D_y into D_z. However, using X and Y in place of X_z and Y_z does not work as the shapes do not align.

$$X \cdot W_{xz} : [N, D_z] \tag{5.7}$$

$$Y \cdot W_{yz} : [K_y, D_z] \tag{5.8}$$

The first tricky coercion is to create K_y copies of X so that each X is paired against K_y possible combinations and to create N copies of Y so that each Y is paired against N data points.

$$X_z = \text{vstack}([X] * K_y) \tag{5.9}$$

$$Y'_z = \text{vstack}([Y] * N) \tag{5.10}$$

This coercion makes the shapes align properly, but the ordering of the rows of Y'_z is incorrect. Let us use $N = 2$ and $K_y = 2$ for a brief examination.

$$X = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}, \; Y = \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}$$

$$X_z = \begin{pmatrix} x_0 \\ x_1 \\ x_0 \\ x_1 \end{pmatrix}, \; Y'_z = \begin{pmatrix} y_0 \\ y_1 \\ y_0 \\ y_1 \end{pmatrix}$$

Note that data point x_0 is paired against class y_0 twice and never against y_1. The correct pairing would be as follows.

$$X = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}, \; Y = \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}$$

$$X_z = \begin{pmatrix} x_0 \\ x_1 \\ x_0 \\ x_1 \end{pmatrix}, \; Y_z = \begin{pmatrix} y_0 \\ y_0 \\ y_1 \\ y_1 \end{pmatrix}$$

The operation shown in Equation 5.10 is intuitive but yields the incorrect Y'_z. The correct operation to give the desired Y_z is less intuitive.

$$Y_z = \text{hstack}([Y] * N).\text{reshape}(N * K_y, K_y) \tag{5.11}$$

With X_z and Y_z, the mean and variance of Z can be obtained using Equation 5.6. Each has $N * K_y$ rows and D_z columns.

$$\mu_z : [N * K_y, D_z] \tag{5.12}$$

$$\sigma_z : [N * K_y, D_z] \tag{5.13}$$

K_z samples of Z are drawn using the reparameterization trick [45], but first μ_z and σ_z need to be replicated K_z times.

$$\mu_{zx} = \text{vstack}([\mu_z] * K_z) \tag{5.14}$$

$$\sigma_{zx} = \text{vstack}([\sigma_z] * K_z) \tag{5.15}$$

$$Z_x = \mu_{zx} + \sigma_{zx}\epsilon \tag{5.16}$$

In this case, ϵ is a matrix with shape $[N * K_y * K_z, D_z]$ where each element is drawn from $\mathcal{N}(0, 1)$, and the multiplication $\sigma_{zx}\epsilon$ is not a matrix product but rather an elementwise multiplication.

At this point, all variables have been inferred. Now the likelihood of the data X needs to be evaluated. According to the generative distribution, the mean of X depends on Y and Z.

$$\mu_x = Y_x \cdot W_{yx} + Z_x \cdot W_{zx} \tag{5.17}$$

Z_x was obtained by stacking K_z copies together, and thus Y_x follows the same pattern.

$$Y_x = \text{vstack}([Y_z] * K_z) \tag{5.18}$$

The dimensions of the resulting μ_x are not the same as the original X, so X itself must be tiled to obtain X_x.

$$X : [N, D_x] \tag{5.19}$$

$$\mu_x : [N * K_y * K_z, D_x] \tag{5.20}$$

$$X_x = \text{vstack}([X] * K_y * K_z) \tag{5.21}$$

$$X_x : [N * K_y * K_z, D_x] \tag{5.22}$$

Suppose (for example) that the likelihood function $f(x, \mu)$ for a single data point x is the sum-of-squares difference from its mean μ across all D_x dimensions.

$$f(x, \mu) = \sum_{D_x} (x - \mu)^2 \tag{5.23}$$

Then for the batch X_x and set of means μ_x, we have $f(X_x, \mu_x)$.

$$X' = (X_x - \mu_x)^2 \tag{5.24}$$

$$f(X_x, \mu_x) = \tilde{X} = X'.\text{sum}(\text{axis} = -1) \tag{5.25}$$

The .sum() over axis=-1 means that the sum is taken once per row, or equivalently, that all columns are summed together. Therefore, $\tilde{X}$ is a column vector.

$$\tilde{X} : [N * K_y * K_z, 1] \tag{5.26}$$

The likelihood $p(x|y, z)$ is a scalar manipulation away from $\tilde{X}$.

Weighting by $q(y|x)$ requires a second very tricky coercion. Each data point x has a K_y-dimensional vector of probabilities, one probability associated to each possible class. The whole batch X has probabilities π.

$$\pi = X \cdot W_{xy} \tag{5.27}$$

$$\pi : [N, K_y] \tag{5.28}$$

Recall that each possible class Y_z was paired against every data point X_z, thus π needs to be unrolled to match up with Y_z. Performing a straight-forward reshaping will yield π'_z,

$$\pi'_z = \pi.\text{reshape}(N * K_y, 1) \tag{5.29}$$

however the elements of π'_z do not line up with their corresponding classes in the replicated Y_z. To obtain the correct alignment, the matrix π must first be transposed.

$$\pi_z = \pi^T.\text{reshape}(N * K_y, 1) \tag{5.30}$$

Then to actually perform the weighting, π_z must be replicated K_z times.

$$\pi_x = \text{vstack}([\pi_z] * K_z) \tag{5.31}$$

This column vector π_x can be element-by-element multiplied by $p(x|y,z)$ or $\tilde{X}$ to obtain $J_{u,xyz}$.

$$J_{u,xyz} = p(x|y,z)\pi_x \tag{5.32}$$

The intermediate variable $J_{u,xyz}$ needs to be unraveled such that the K_z samples of z can be summed out, and then it needs to be reordered again such that the K_y enumerations of y can be summed out. First $J_{u,xyz}$ is reshaped to allow z to be removed, and we give this variable a superscript z to denote this arrangement.

$$J^z_{u,xyz} = J_{u,xyz}.\text{reshape}(K_z, N * K_y) \tag{5.33}$$

$$J_{u,xy} = \sum_z J^z_{u,xyz} \tag{5.34}$$

$$= J^z_{u,xyz}.\text{sum}(\text{axis} = 0) \tag{5.35}$$

The intermediate variable $J_{u,xy}$ has had z removed and is now a matrix of shape $(1, N * K_y)$. The .sum() over axis=0 is performed vertically over the matrix resulting in one sum per column of data. Then $J_{u,xy}$ must again be reshaped so that the sum over y can be performed.

$$J^y_{u,xy} = J_{u,xy}.\text{reshape}(K_y, N) \tag{5.36}$$

$$J_{u,x} = \sum_y J^y_{u,xy} \tag{5.37}$$

$$= J^y_{u,xy}.\text{sum}(\text{axis} = 0) \tag{5.38}$$

Like the previous .sum(), the .sum() over axis=0 in this case results in one sum per column, thus $J_{u,x}$ is left as a row vector. It can be transposed to obtain the values corresponding to each point in the batch X.

5.2.2 Tensorized Framework

We begin with a batch of data X_2 which consists of N points each with dimension D_x. The colon indicates the shape of the tensor.

$$X_2 : [N, D_x] \tag{5.39}$$

We immediately upgrade X_2 to a fourth-order tensor X by inserting two empty axes into it.

$$X : [N, 1, 1, D_x] \tag{5.40}$$

These two empty axes will be used for replications of Y and Z. For convenience, let us label all axes from left to right as follows: Axis N, Axis Z, Axis Y, and Axis D. From this notation, it is clear that X is spread across Axis N and Axis D and contains no replications of y or z.

Each of the points needs to be paired against K_y possible class labels. The matrix $Y_2 = \text{eye}(K_y)$ is the set of all possible class labels in one-hot form, and we immediately promote it to a fourth-order tensor Y.

$$Y_2 : [K_y, K_y] \tag{5.41}$$

$$Y : [1, 1, K_y, K_y] \tag{5.42}$$

From the labeling of the axes, we see that Y is spread across Axis Y and Axis D.

The mean μ_z and standard deviation σ_z of Z are the outputs of a neural network function, represented here as vector dot products (equivalent to matrix multiplication), with inputs X and Y.

$$(\mu_z, \sigma_z) = X \cdot W_{xz} + Y \cdot W_{yz} \tag{5.43}$$

The weight matrix W_{xz} has shape $[D_x, D_z]$ and W_{yz} has shape $[K_y, D_z]$. Both μ_z and σ_z have replications along Axis N and Axis Y as desired, and there is no need to alter or reshape X or Y for this computation to succeed.

$$\mu_z : [N, 1, K_y, D_z] \tag{5.44}$$

$$\sigma_z : [N, 1, K_y, D_z] \tag{5.45}$$

To replicate K_z times along Axis Z, simply allocate enough noise samples ϵ

$$\epsilon : [N, K_z, K_y, D_z] \tag{5.46}$$

and compute Z directly.

$$Z = \mu_z + \sigma_z \epsilon \tag{5.47}$$

Each element of ϵ is drawn from $\mathcal{N}(0,1)$, and the multiplication $\sigma_z\epsilon$ is performed element-by-element.

At this point, all variables have been inferred. Now the likelihood of the data X needs to be evaluated. According to the generative distribution, the mean of X depends on Y and Z.

$$\mu_x = Y \cdot W_{yx} + Z \cdot W_{zx} \tag{5.48}$$

The matrices W_{yx} and W_{zx} have shapes $[K_y, D_x]$ and $[D_z, D_x]$ respectively. There is no need to alter or reshape Y or Z for this computation to succeed.

Suppose (for example) that the likelihood function $f(x, \mu)$ for a single data point x is the sum-of-squares difference from its mean μ across all D_x dimensions.

$$f(x, \mu) = \sum_{D_x} (x - \mu)^2 \tag{5.49}$$

Then for the batch X and set of means μ_x, we have $f(X, \mu_x)$.

$$X' = (X - \mu_x)^2 \tag{5.50}$$

$$f(X, \mu_x) = \tilde{X} = X'.\text{sum}(\text{axis} = \text{Axis D}) \tag{5.51}$$

The .sum() over Axis D results in the collapsing of Axis D.

$$\tilde{X} : [N, K_z, K_y, 1] \tag{5.52}$$

The likelihood $p(x|y,z)$ is a scalar manipulation away from $\tilde{X}$.

Weighting by $q(y|x)$ is simple. First compute the probabilities π'.

$$\pi' = X \cdot W_{xy} \tag{5.53}$$

$$\pi' : [N, 1, 1, K_y] \tag{5.54}$$

Notice that π' is spread across Axis N and Axis D, but we need to weight the likelihoods across Axis Y instead of Axis D. Swapping the axes in an operation analogous to a matrix transposition solves the alignment.

$$\pi = \pi'.\text{swap}(\text{Axis D}, \text{Axis Y}) \tag{5.55}$$

Then actually performing the weighting is a simple element-by-element multiplication.

$$J_{u,xyz} = p(x|y,z)\pi \tag{5.56}$$

To obtain $J_{u,x}$ from $J_{u,xyz}$, simply sum over the necessary axes.

$$J_{u,xy} = J_{u,xyz}.\text{sum}(\text{axis} = \text{Axis Z}) \tag{5.57}$$

$$J_{u,x} = J_{u,xy}.\text{sum}(\text{axis} = \text{Axis Y}) \tag{5.58}$$

This now contains the values corresponding to each point in the batch X.

$$J_{u,x} : [N, 1, 1, 1] \tag{5.59}$$

5.3 Direct Comparison

The graphical model for M2 is very small; aside from the data x, the only variables are class label y and a single latent variable z. Nevertheless, Equations 5.4-5.38 are needed to describe and reason about the objective function that trains M2 when coercing all intermediate calculations into matrices. Equations 5.39-5.59 calculate the same objective using the tensorized framework. Even for a model as small as M2, the tensorized framework significantly shortens the implementation of deep generative models.

Notice the subscript on the batch of data X in Equation 5.6 does not exist in the analogous Equation 5.43 from the tensorized framework. Every time the data X is used in the computation of a variable that requires replication, it too must be replicated. Similarly, Y needs to be replicated based on the other variables' replications. However, all replications are trivial in the tensorized framework, so there is no need to differentiate X from X_z and X_x – all three of these cases use the original tensor X. This becomes more and more useful as the complexity of the model increases.

The properties of tensors that allow trivial replication come from the *broadcasting* rules developed originally by NumPy. There are two rules that are relevant in deep generative models, and both can be seen in practice in Equation 5.43, for example.

1. Operations that require two tensors of compatible shape allow lower-order tensors to be promoted to higher-order tensors by adding axes of dimension one to the left.

2. Operations that require two tensors of compatible shape allow for any axis of dimension one to be replicated along that axis.

In Equation 5.43, two terms are summed together: $X \cdot W_{xz}$ and $Y \cdot W_{yz}$. Both matrices W are second-order tensors (which are immediately promoted), and the dot-product serves the function of changing D_x and D_y to D_z so that, before summing, the tensors' shapes are as follows.

$$X \cdot W_{xz} : [N, 1, 1, D_z] \tag{5.60}$$

$$Y \cdot W_{yz} : [1, 1, K_y, D_z] \tag{5.61}$$

The first broadcasting rule was used to allow the matrices W to interact with the higher-order tensors X and Y, and the second broadcasting rule allows the summation to obtain the proper replications. The first axis farthest to the left in $Y \cdot W_{yz}$ is one-dimensional, and therefore it can be replicated N times. Similarly, $X \cdot W_{xz}$ is one-dimensional in the third axis, and therefore it can be replicated K_y times. Unlike the matrix coercion method, however, it is not necessary to explicitly perform the replications or to even know how each variable will be replicated – the tensorized framework automatically infers the correct replications.

In addition to shortening implementations, not explicitly replicating and reshaping matrices significantly decreases the probability of making mistakes in implementation. Consider the matrix transpose in Equation 5.30, just before the reshape, compared to Equation 5.55, the much more straight-forward equivalent in the tensorized framework. It is clear in the tensorized version that the probabilities need to weight the individual classes, and thus moving them to the axis that contains all of the class labels is intuitive. More problematically in the matrix version, it is critically important to know the details of how the reshape operation is handled. Consider this simple example to see just how error-prone reshaping can be by following a row vector r with six elements.

$$r = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \end{pmatrix} \tag{5.62}$$

It can be reshaped to have shape $[3, 2]$ or $[2, 3]$.

$$r_{32} = r.\text{reshape}(3, 2) \tag{5.63}$$

$$= \begin{pmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{pmatrix} \tag{5.64}$$

$$r_{23} = r.\text{reshape}(2, 3) \tag{5.65}$$

$$= \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{pmatrix} \tag{5.66}$$

However, notice that $r_{23} \neq r_{32}^T$ even though both r_{23} and r_{32}^T have the same shape. This difference is very subtle and yet can result in severe implications if handled incorrectly, as was seen in Equation 5.30.

Even more reshapes can be avoided after all likelihoods are calculated and the latent variables need to be summed out. In the matrix version, unrolling the stack of calculations allows each variable to be summed out in order; Equation 5.33 first removes z and then Equation 5.36 removes y. Conversely, the tensorized framework allows any variable to be summed out at any time; Equation 5.57 and Equation 5.58 remove z and y, but nothing prevents the two sums to be performed in any other arbitrary order. Allocating one entire axis of all tensors to each latent variable makes the bookkeeping trivial.

5.4 Summary of Tensorized Framework

The tensorized framework uses tensors to store all intermediate variables. For a specific deep generative model, all tensors have a fixed order that depends on the number of variables in the DGM. In the case of M2, we saw that fourth-order tensors were sufficient. For the slightly more complex model SDGM, shown in Figure 5.1, the tensors are fifth-order.

In general, the order is one plus the number of variables in the model. SDGM, for example, has data x, class label y, latent variable z, and auxiliary variable a, and thus the tensors will have the following shapes.

$$X : [N, 1, 1, 1, D_x] \tag{5.67}$$

$$Y : [1, 1, 1, K_y, K_y] \tag{5.68}$$

$$A : [N, K_a, 1, 1, D_a] \tag{5.69}$$

$$Z : [N, K_a, K_z, K_y, D_z] \tag{5.70}$$

It is immediately obvious from the graphical model the order in which dependent latent variables will be replicated, and because each variable has its own axis, weighting and summing out a given variable can be performed along its unique axis easily.

For the variables that are to be sampled using the reparameterization trick, the amount of noise needs to match the tensor shape. SDGM's A and Z would need ϵ_a and ϵ_z, respectively.

$$\epsilon_a : [N, K_a, 1, 1, D_a] \tag{5.71}$$

$$\epsilon_z : [N, K_a, K_z, K_y, D_z] \tag{5.72}$$

If Gaussian, then each element of ϵ_a and ϵ_z would be drawn independently from $\mathcal{N}(0, 1)$.

For the variables that are to be completely enumerated, the identity matrix needs to be promoted from a second-order tensor to the higher order of the other tensors. It may occasionally be necessary to swap two axes of a higher-order tensor in order to spread the matrix to the correct axes or to weight a tensor along the correct axis. In SDGM, for example, the probabilities π' are obtained using X and A, but two axes need to be swapped to obtain the correct weights π.

$$\pi' = X \cdot W_{xy} + A \cdot W_{ay} \tag{5.73}$$

$$\pi' : [N, K_a, 1, 1, K_y] \tag{5.74}$$

$$\pi = \pi'.\text{swap}(\text{Axis D}, \text{Axis Y}) \tag{5.75}$$

$$\pi : [N, K_a, 1, K_y, 1] \tag{5.76}$$

Here, Axis D is the axis farthest to the right and Axis Y is the axis immediately to the its left. The operation of swapping two (or more) axes can be performed using a `dimshuffle` in Theano.

In summary, there are three guidelines needed to use the tensorized framework for a deep generative model.

1. Allocate tensors of the correct order based on the number of latent variables.
2. Create enough noise for the latent variables that use the reparameterization trick.
3. Promote the identity matrix and swap axes as needed for the latent variables that are enumerated.

Following the guidelines allows existing neural network toolchains to add probabilistic graphical models in a remarkably simple fashion.

5.5 Conclusion

We developed a tensorized framework for extending neural network toolchains to deep generative models by adding in probabilistic graphical models. We showed a typical term in the computation of the objective function of the deep generative model M2, then we detailed the process of computing it using both a traditional matrix-coerced version as well as our novel tensorized framework. Next we compared the two versions point-to-point along very specific challenges in the implementations of the objective. The tensorized framework significantly simplifies or trivializes each and every one of the difficulties of the traditional, matrix implementation.

To verify that the tensorized framework performs well with existing tools, we implemented two semi-supervised deep generative models M2 and SDGM in Theano. Both models were tested on the MNIST data set with a mere 100 labeled data points per trial for the semi-supervised cases. By using all 60,000 unlabeled data points in the training and validation sets, the two models matched the state-of-the-art results as expected. Note that, for simplicity, we did not implement batch normalization [40], and therefore the semi-supervised cases exhibit accuracy growth even at the end of the 200 epochs of training and would perform slightly better with longer training time. Our complete implementation of SDGM[1] using the tensorized framework is publicly available online.

[1] `http://github.com/rrastgoufard/sdgm`

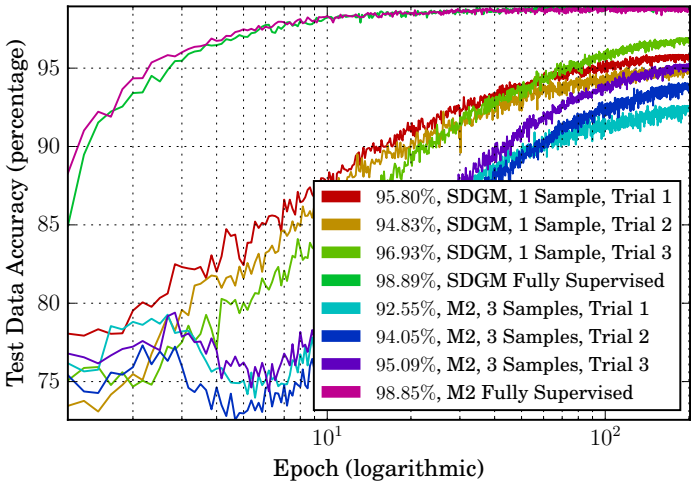


Figure 5.2: Classification performance of M2 and SDGM using the tensorized framework over multiple trials for both semi-supervised and fully supervised cases on the MNIST dataset. Both the M2 performances and the SDGM performances are competitive with the state-of-the-art in the respective authors' original implementations. M2 used $K_z = 3$ samples, and SDGM used $K_a = K_z = 1$ sample. Both enumerated all classes with $K_y = 10$.

A Priori Independence for Deep Generative Models

Deep Generative Models are wonderfully expressive. They utilize an autoencoder structure for generating (or imagining) data and for inferring (or recognizing) details about the data. The most basic version, called a variational autoencoder (or M1 in the previous chapter), has a single latent variable that encodes training data in an unsupervised manner. An extension to semi-supervised learning, the model M2, combines two variables, one is a discrete class label and the other is a continuous latent variable whose purpose is to provide variations in the data that are not caused by the class label.

The structure of the two-variable generative model should imply that both the class label and the continuous variable are free parameters that can be chosen to generate data, that is, the two are a priori independent. We examined this behavior very closely in Figure 4.4 and Figure 4.5. The behavior we saw is *not guaranteed*, though it did happen for the MNIST data set. It does not happen when using the NORB data set. The purpose of this chapter is to understand and enforce a priori independence.

This chapter and the next focus on the NORB data set. NORB is a multi-label set of images that consists of 50 plastic toys that are imaged under varying lightings L, camera elevations E, and rotations R, and each of the toys falls under one of five classes Y: animal, human, car, truck, or plane. In this chapter we examine the lighting label.

Based on the construction of the data set, we know that every toy/elevation/rotation is imaged under every possible lighting, thus we expect to be able to generate (or imagine) every lighting for any choice of toy. Experimentally, we find that some of the lighting conditions create strong shadows, and the latent space groups the shadowed data in one place and the remaining data in another. This separation of lightings conflicts with our understanding of the two-variable generative model.

A priori independence (API) is a concept that can be applied to any semi-supervised generative model. We examine its effects on M2 and SDGM in this chapter, and SDGM with API expresses very good characteristics, as shown in Table 6.1. M2 also improves, as seen visually in Figure 6.4, but we always use SDGM as a superior version of M2. In Chapter 7, API is used to strengthen a novel multi-label model. The model performs worse than the single-label SDGM unless API is enforced.

Autoencoders make an information round-trip that begins with data, goes to latent variables, and then comes back to data, hence autoencoders have a bottom-up character with a focus on the raw, low-level data. A priori independence does the reverse by beginning with latent variables, generating data, and re-inferring

Table 6.1: SDGM with API Scores

	Data Driven	Semi-supervised	Steerable	Generalization	Adaptability	Total
SDGM	1	1	0.5	0.6	1	4.1
SDGM with API	1	1	0.5	0.9	1	4.4

the latent variables. This flow is top-down and focuses on high-level information. For a multi-label model, the top-down flow is essential for coordinating the different labels, as we see in the following chapter.

6.1 Introduction

Machine learning algorithms benefit greatly from having large data sets for training, but fully labeling a data set in order to train a supervised algorithm can be very costly. Semi-supervised algorithms excel at using partially-labeled data sets, even when the portion of labeled data is minuscule, and they can leverage large bodies of unlabeled data to drastically improve performance. The label-deficient MNIST handwritten digit benchmark allows a mere 100 labeled images and 60,000 unlabeled images for training, and strong support vector machines and neural network classifiers achieve only 70% to 75% accuracy using just labeled data [75]. Incorporating unlabeled data can increase accuracy to 81% [74], 84% [19], 88% [77], 90% [94], or, more recently, 98% [44, 58, 59, 63, 73]. Deep Generative Models (DGMs) are among the state-of-the-art for unsupervised and semi-supervised learning [44, 45, 58, 59] capable of attaining 98% using 100 labels, close to the best results obtained by using full supervision.

As is suggested in the name, DGMs define generative models that describe the data creation process, and, for semi-supervised learning, they typically utilize a discrete latent variable for generating class information coupled with a continuous latent variable for generating all of the variation in the data that is not caused by the class. When trained on the NORB data set, for example, the discrete component could correspond to lighting conditions and the continuous variable to object type. In Figure 6.1(b), the continuous variable corresponds specifically to an elephant with a fixed camera azimuth and elevation, and the discrete component is varied over the possible lightings imaged in the data set.

Excluded from their name is the fact that DGMs are comprised of two networks – a generative network P that generates data from latent variables and an inference network Q that recognizes the values of latent variables given a set of observed data. The two networks are linked together probabilistically using a variational

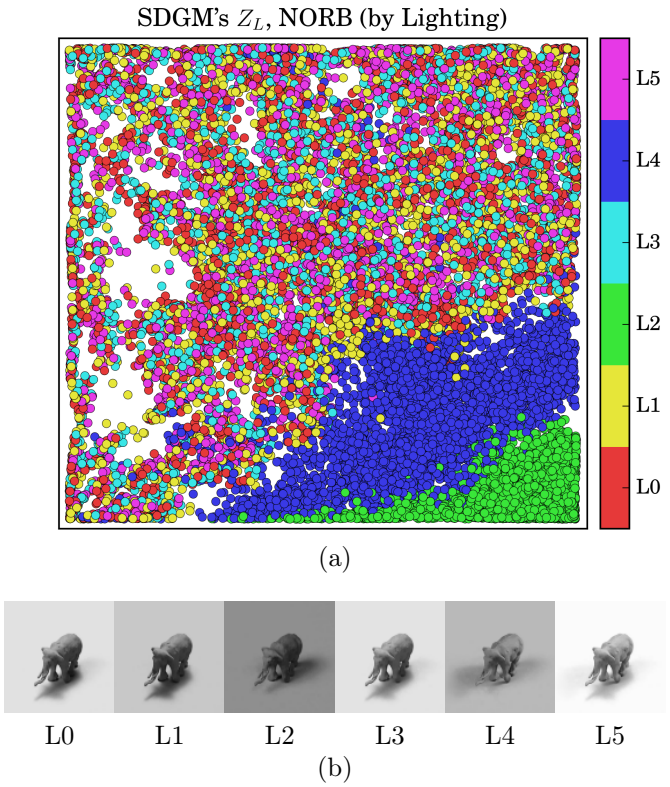


Figure 6.1: (a) Visualization of a learned latent space Z_L showing that the class label L and the latent variable Z_L are not naturally a priori independent, and (b) the same elephant imaged under six different lighting conditions (L0 to L5). Without a priori independence, the model *cannot* recognize that the elephant is the same in these six images.

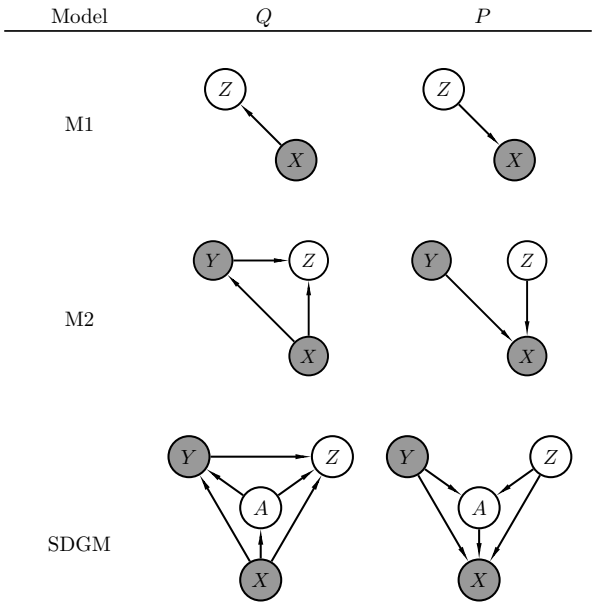


Figure 6.2: Three Deep Generative Models showing both the generative network P and the inference network Q. In semi-supervised training, the data X always is observed, and the class label Y sometimes is observed. Our proposed objectives can be applied to M2, SDGM, or any other semi-supervised DGMs.

Bayes objective that optimizes the evidence lower bound (ELBO) of the data. Figure 6.2 shows the graphical models of the original, unsupervised Variational Autoencoder M1 [45], a semi-supervised classifier M2 [44], and a more modern semi-supervised classifier known as the Skip Deep Generative Model (SDGM) [59].

Deep Generative Models are wonderfully expressive, but we notice the problem that some nonzero-probability regions of the latent spaces can generate gibberish data. This is evident by examining SDGM's learned latent space Z_L when trained on NORB's lighting label L, as shown in Figure 6.1(a). In the lower right corner of the latent space, there are two pronounced clusters, one corresponding to $L = 2$ and the other to $L = 4$. Choosing a Z_L in the region that corresponds to $L = 4$, for example, can reconstruct the L4 elephant in Figure 6.1(b), but keeping that specific value of Z_L and using any other value of L cannot possibly reconstruct the other elephants, even though the other elephants were present during training. More importantly, the data that would be generated with that specific Z_L and any other value of L is not reminiscent of any of the training data and thus must be considered gibberish. In this chapter, we aim to remedy the problem of generating garbage data by enforcing the latent space Z_L and the discrete label L to be a priori independent.

Research in DGMs is expanding rapidly, and most of it falls under one of two categories. The first category explores different model architectures and contains Conditional Variational Autoencoders [83] that treat the class label Y as an output rather than an input in the generative model, [35] that adds a stochastic

variable used in the generation of both X and Y, [84] that describes the Ladder Variational Autoencoder which has a deep hierarchy of Zs, and [97] that creates a separation of Zs in order to segment foreground from background in images. The second category examines different objectives that can be used to train the models and contains the Importance-Weighted Autoencoder [17] which uses repeated sampling to tighten the ELBO and [64] which extends the multi-sample idea with a new gradient estimator. Tangential to DGMs is the very popular family of Generative Adversarial Models (GANs) [34], and Adversarial Variational Bayes [62] unifies GANs with DGMs, thus putting it into both the alternative architecture and the alternative objective categories of research.

This chapter falls under the alternative objectives category of research. We do not modify the architectures of DGMs, but instead we propose two new objectives to be used in addition to the existing semi-supervised DGM objectives. The objectives are model-independent and thus can be applied to any variations of DGMs, but the training procedure, while still using the standard tools such as gradient descent, is unique for these objectives. Spiritually, the methodology that we describe here is similar to that of a GAN in the sense that we want to eliminate garbage generated data. The biggest difference is that we leverage the existing inference network in DGMs instead of using an adversarial classifier. It is possible to mix our objectives with a GAN variant extended to semi-supervised learning, and this could be a future research direction.

The remainder of this chapter is organized as follows. First we briefly describe Deep Generative Models and show results from the MNIST and NORB data sets in order to build intuition. Next we detail the two novel objectives along with their unique training procedure. Afterwards we apply the novel objectives to M2 and SDGM and present their successes on the NORB data set. Finally we provide discussion on the methodology and conclude this chapter.

6.2 Brief Overview of Deep Generative Models

Deep Generative Models for semi-supervised learning define two joint distributions $p(x, y, z)$ and $q(y, z|x)$ where x is the data, y is a discrete class label, and z is a continuous latent variable. The distribution p describes the generative model of the data, and the recognition distribution q describes how the latent variables can be inferred given a data point x. The likelihood of the data $p(x)$ can be found as a function of

the joint distributions.

$$p(x) = \sum_y \int p(x, y, z)\ dz \tag{6.1}$$

$$= \sum_y \int \frac{p(x, y, z)\ q(y, z|x)}{q(y, z|x)}\ dz \tag{6.2}$$

$$= E_{q(y,z|x)} \left[\frac{p(x, y, z)}{q(y, z|x)} \right] \tag{6.3}$$

The log-likelihood of the data is bounded to obtain the evidence lower bound (ELBO).

$$\log p(x) \geq E_{q(y,z|x)} \log \left[\frac{p(x, y, z)}{q(y, z|x)} \right] \tag{6.4}$$

In practice, the ELBO need not be simplified further and is evaluated empirically by enumerating over all possible y and approximating the integral over z with random samples.

The two distributions p and q are assumed to factorize according to chosen generative and inference models. For example, M2's graphical models, shown in Figure 6.2, factorize as follows.

$$p(x, y, z) = p(x|y, z)\ p(y)\ p(z) \tag{6.5}$$

$$q(y, z|x) = q(y|x)\ q(z|y, x) \tag{6.6}$$

SDGM has an additional continuous latent variable a, but its ELBO and factors are similar.

$$p(x, a, y, z) = p(x|y, z, a)\ p(a|y, z)\ p(y)\ p(z) \tag{6.7}$$

$$q(a, y, z|x) = q(a|x)\ q(y|x, a)\ q(z|x, a, y) \tag{6.8}$$

The prior distributions $p(y)$ and $p(z)$ are very simple multinomial and standard normal, but the remainder are more complicated. The conditional factors $p(x|...)$, $q(y|...)$, $q(z|...)$, $p(a|...)$, and $q(a|...)$ are distributions parameterized by neural networks whose inputs are the conditioning variables. In our case (for the NORB data set), $p(x|...)$ is Gaussian, as are $q(z|...)$, $p(a|...)$, and $q(a|...)$. The remaining distribution $q(y|...)$ is multinomial. We name the neural networks corresponding to the distributions as f_{px}, f_{qz}, f_{pa}, f_{qa}, and f_{qy}. The weights and biases that form the parameter sets of the networks are θ_{fpx}, θ_{fqz}, θ_{fpa}, θ_{fqa}, and θ_{fqy}.

The negative of the bound in Equation 6.4 serves as the objective to be minimized, but because both labeled and unlabeled data are present, separate sub-objectives are needed. The unlabeled objective J_u uses unlabeled data $x \in X_u$, enumerates over all possible y, and samples randomly from a and z in order to

evaluate the expectation. The labeled objective J_l uses labeled data $x \in X_l$ along with the known labels $y \in Y_l$, so no summation over y is necessary, and the expectation is over samples of a and z. The classifier f_{qy} is used only in J_u whereas it ought to be trained using the labeled data pairs (X_l, Y_l), so a third objective J_a, the log-cross-entropy between $f_{qy}(x)$ and y, is added to ensure that the outputs of the network f_{qy} match the true labels y for $(x, y) \in (X_l, Y_l)$. The combination of these three sub-objectives forms the standard semi-supervised objective J_{ssl}.

$$J_{ssl} = J_u + J_l + \alpha_a J_a \tag{6.9}$$

$$\theta_{ssl} = \theta_{fpx} \cup \theta_{fqz} \cup \theta_{fqa} \cup \theta_{fpa} \cup \theta_{fqy} \tag{6.10}$$

The mixing weight α_a determines the importance of J_a. Optimizing J_{ssl} involves performing gradient descent on the parameters θ_{ssl}.

This overview of DGMs is *very* brief, and we refer readers to [44, 45, 58, 59] for more detail. Our notation is slightly different than the existing literature because we need to differentiate the existing objective J_{ssl} and its sub-objectives J_u, J_l, and J_a from the objectives J_{api} and J_{ccv} that we propose in this chapter. We also need to separate the neural network fs and their parameter θs for our novel training procedure, hence their individual namings.

6.3 A Priori Independence, or Lack Thereof

The generative models in M2 and SDGM imply that Y and Z can be selected independently, and the combination of the two is necessary to generate data X. Before now [44, 83] there was no reason to believe otherwise, as experiments on the MNIST data set yield the expected result, reproduced here in Figure 6.3 which visualizes the latent space Z_Y that is learned when training SDGM on the MNIST data set. The values of Z_Y that correspond to all 60,000 training data points are plotted in the scatter plot on the right side of the figure, and the colors indicate the true class label Y associated with each data point. The left side of the figure shows actual data points and gives a good indication of the handwriting style information that is stored in Z_Y. The continuous latent variable Z_Y captures all of the variation in the data that is not caused by the class label Y. Notice that the space of Z_Y is completely independent of the class labels Y, that is $p(Z_Y|Y) = p(Z_Y)$.

Contrast the space of Z in Figure 6.3 with that in Figure 6.1(a) which is trained on the NORB data set using only the lighting label L as the discrete label instead of the class Y. In Figure 6.1, it is clear that there are distinct clusters in the Z_L space for the specific labels $L = 2$ and $L = 4$, meaning that $p(Z_L|L) \neq p(Z_L)$.

The lack of a priori independence has two significant and negative implications.

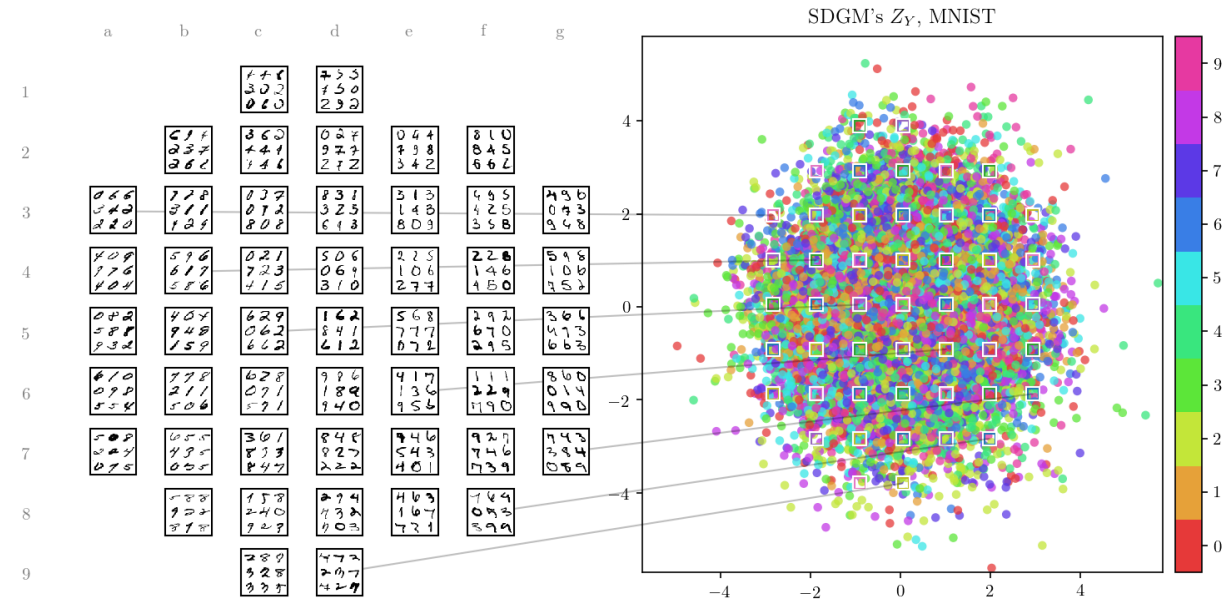


Figure 6.3: Visualization of SDGM's latent space Z_Y when trained on the MNIST data set, and the colors correspond to the inferred data points' true class labels Y. Notice that the space of Z_Y shows no structure regarding the class label (all of the ones, twos, threes, etc. are completely mixed together), but there is clear structure in the space that captures variation in handwriting style. The images to the left of the scatter plot are real samples from the training data, and their positioning indicates where their corresponding inferred Z_Ys are situated.

1. Generated data from the region $Z_{L=2}$ is gibberish when $L \neq 2$. The generative model *cannot* function properly in large portions of its latent space.

2. The same real-world object has completely different inferred Z_L based on the inferred L. The L1 elephant and the L2 elephant in Figure 6.1(b) *cannot* be recognized as the same elephant even though the variation in the images is due solely to a difference in lighting and not to a difference in elephant.

Only with a priori independence can the model possibly hope to overcome the two negative implications.

We now define two objective functions J_{api} and J_{ccv}. The first is a measure of a priori independence. The second is a measure of conditional covariance.

6.3.1 Measuring A Priori Independence

To measure a priori independence, we leverage the fact that choosing specific values of the continuous latent variable Z and pairing them with specific values of the discrete label Y will result in garbage outputs from the generative model. The class label that would be inferred from such an output does not match the true class label that would have been used to generate it. The objective J_{api} captures the error between the inferred class label and the true label.

Computing J_{api} requires a top-down roundtrip that is similar (but reversed in direction) to the bottom-up roundtrip of an autoencoder. The process requires four steps.

1. Begin with a real data point x.
2. Use the inference network of a DGM to infer $\hat{y}$ and $\hat{z}$.
3. Discard the inferred label $\hat{y}$ and instead use all possible labels y_i paired with the single $\hat{z}$ to generate several data reconstructions $\hat{x}_i$.
4. Use the inference network again to infer all of the $\hat{y}_i$ corresponding to the generated $\hat{x}_i$.

The objective J_{api} is the log-cross-entropy between the inferred $\hat{y}_i$ and the known y_i.

Ordinarily, training a DGM means optimizing the objective J_{ssl} using gradient descent on the parameters of the DGM's neural networks θ_{ssl}. To utilize J_{api}, we cannot simply combine it with J_{ssl} using a mixing weight. Doing so results in a catastrophic failure where the DGM aggressively learns to generate garbage and to classify garbage. Instead, we need to mix J_{api} and J_{ssl} on a network-by-network basis.

Our proposed training procedure pairs each network f and θ_f with its own mixture of objectives. We demonstrate on SDGM's networks f_{px}, f_{qy}, f_{qa}, f_{pa}, and f_{qz} along with their parameters. Here, the notation $\theta : J$ indicates that the cost function J is to be used in gradient descent for updating the parameters of θ.

$$\theta_{px} : J_{ssl}$$

$$\theta_{qy} : J_{ssl}$$

$$\theta_{qa} : J_{ssl} + \alpha_{api} J_{api}$$

$$\theta_{pa} : J_{ssl} + \alpha_{api} J_{api}$$

$$\theta_{qz} : J_{ssl} + \alpha_{api} J_{api}$$

For a general DGM, θ_{px} and θ_{qy} are *not allowed* to help with the optimization of J_{api}, but all other θs should interact with J_{api}. This training procedure will be extended when incorporating J_{ccv}.

6.3.2 Measuring Conditional Covariance

The conditional covariance objective J_{ccv} aims to force $p(Z|Y) = p(Z)$ for all possible Y. We choose $p(Z)$ to be standard normal, so we need to guide the inferred $\hat{z}$ to have a diagonal covariance matrix equal to the identity matrix for each possible class label Y.

Computing the conditional covariance matrices for a batch of data points is tricky because the data's true class labels are not known in our semi-supervised setting. Therefore, all data points in the batch contribute

to all conditional covariance matrices, but the contribution amounts depend on each individual data point's soft class assignments.

A handful of inferences need to be obtained before computing the conditional covariance matrices. Begin with a batch of data points x stored in a matrix that has N rows, one per data point, and D_x columns, one per data dimension. Infer the best class predictions $\hat{y}$, an $N \times D_y$ matrix where D_y is the number of possible classes such that $\hat{y}$ is in a one-hot form. Also infer soft class assignments π, an $N \times D_y$ matrix where each row sums to one but without the one-hot restriction. Finally, infer the latent $\hat{z}$ corresponding to x and $\hat{y}$ but ignoring (for now) π. The matrix $\hat{z}$ has N rows and D_z columns, where choice of D_z, the dimension of the latent space, is in the hands of the model designer. Note that the j, ith element of $\hat{y}$ is one if and only if $\pi_{j,i}$ is the maximum probability in the row π_j.

With $\hat{z}$, π, and D_z, computation of J_{ccv} is straight-forward. The ith conditional covariance matrix is C_i, the (soft) number of data points contributing to that covariance is N_i, and the soft assignments to class i are π_i.

$$\pi_i = \pi_{:,i} \tag{6.11}$$

$$\hat{z}_i = \pi_i \hat{z} \tag{6.12}$$

$$C_i = \hat{z}_i^T \hat{z}_i \tag{6.13}$$

$$N_i = \sum \pi_i \tag{6.14}$$

In Equation 6.12, the multiplication is performed element-by-element, but in Equation 6.13, the multiplication is a matrix product. The summation in Equation 6.14 is over the elements of the column vector π_i. The resulting C_i is a $D_z \times D_z$ matrix regardless of how many data points N were in the original data batch. The ith objective $J_{ccv,i}$ is a measure of how far C_i is from an identity matrix.

$$J_{ccv,i} = \text{mean } (C_i - N_i I)^2 \tag{6.15}$$

The identity matrix I has dimension $D_z \times D_z$, the squaring operation is performed element-by-element, and the mean is taken over all elements such that $J_{ccv,i}$ is a scalar. The overall objective J_{ccv} is then the average of the individual conditional covariance costs.

$$J_{ccv} = \frac{1}{D_y} \sum_i J_{ccv,i} \tag{6.16}$$

Just like with J_{api}, only certain sets of parameters are allowed to interact with J_{ccv}. As before, we reuse the $\theta : J$ notation to indicate the correspondences. For SDGM, the only network that should be guided by J_{ccv} is f_{qz}.

$$\theta_{px} : J_{ssl}$$

$$\theta_{qy} : J_{ssl}$$

$$\theta_{qa} : J_{ssl}$$

$$\theta_{pa} : J_{ssl}$$

$$\theta_{qz} : J_{ssl} + \alpha_{ccv} J_{ccv}$$

The same is true for general DGMs.

Of course, J_{api} and J_{ccv} can be trained simultaneously.

$$\theta_{px} : J_{ssl}$$

$$\theta_{qy} : J_{ssl}$$

$$\theta_{qa} : J_{ssl} + \alpha_{api} J_{api}$$

$$\theta_{pa} : J_{ssl} + \alpha_{api} J_{api}$$

$$\theta_{qz} : J_{ssl} + \alpha_{api} J_{api} + \alpha_{ccv} J_{ccv}$$

Several DGMs using different combinations of objectives are examined in the next section.

6.4 Experimental Results

We perform unconventional experiments on the NORB data set in order to highlight the faults of not having a priori independence in a DGM's latent space. Ordinarily, classification would aim to predict the class labels Y of the objects in the testing set – namely, does a given image contain a car, a truck, a plane, a human, or an animal? Each image in the NORB data set is labeled by class, but also present are the true camera azimuths and elevations as well as the lighting conditions L for $L \in \{0, 1, 2, 3, 4, 5\}$. See Figure 6.1(b) for examples of the six different lightings. We use L as the target of our classification experiments.

The data set is separated into a training set X_u, X_l, and L_l, and the testing set is a separate collection of data X_t and lighting labels L_t. In our semi-supervised setting, the unlabeled portion X_u of the training data encompasses all 24,300 images in the training set, and the labeled portion X_l consists of only 1000 images along with their true lighting labels L_l. The labeled data X_l is a subset of the entire pool of training data

Table 6.2: Sub-objective mixing weights for three different experiments using M2 and three using SDGM along with their classification accuracies (in % correct class predictions on testing data). The standard objective is $J_{ssl} = J_u + J_l + \alpha_a J_a$, and our proposed objectives are J_{api} and J_{ccv} weighted by α_{api} and α_{ccv}, respectively.

	α_a	α_{api}	α_{ccv}	%
M2	50	0	0	92.80
M2 with J_{api}	50	25	0	92.27
M2 with J_{api} and J_{ccv}	50	25	1000	93.73
SDGM	50	0	0	93.73
SDGM with J_{ccv}	50	0	25	92.73
SDGM with J_{api} and J_{ccv}	50	25	25	93.67

X_u. Corresponding to lighting labels L_l and L_t are true class labels Y_l and Y_t, but they are never used in either training or testing in this chapter.

We trained six DGMs, three instances of M2 and three instances of SDGM, each with different combinations of objective functions. The sub-objective mixing weights are listed in Table 6.2 along with their lighting label classification accuracies when tested against X_t and L_t. The classification accuracies indicate that no significant losses or gains are obtained by including or excluding J_{api} and J_{ccv}, implying that generalization performance is not adversely affected. However, classification accuracy is not the only way to assess the success of the proposed objectives.

From the visualizations of the latent spaces Z_L learned by all six DGMs in Figure 6.4 and Figure 6.5, it is clear that J_{api} and J_{ccv} together produce the desired a priori independence. Figure 6.4(b) suggests that J_{api} is insufficient on its own to produce the mixing of the lighting labels even though it increases the mixed region from two labels (L0 and L3) to three labels (L0, L1, and L3). Figure 6.5(b) suggests that J_{ccv} in isolation can produce a fully mixed space, however the value of J_{api}, a measure of how much garbage the generative model produces, is not minimized when optimizing J_{ccv} alone. Subjectively and objectively in the cases of both M2 and SDGM, including J_{api} and J_{ccv} is essential for producing a priori independence.

6.5 Discussion

Even though the lighting label L is the target of our classification experiments, ultimately we are interested in how a priori independence of L and Z_L could affect a DGM's understanding of the objects it is tasked with recognizing. Figure 6.4 and Figure 6.5 show visualizations of the latent space Z_L, and the data points are colored both by their true lighting labels L and the their class labels Y. Notice that the class clusters in M2 and SDGM are very fragmented until J_{api} and J_{ccv} are included. This improvement in clustering of Y means

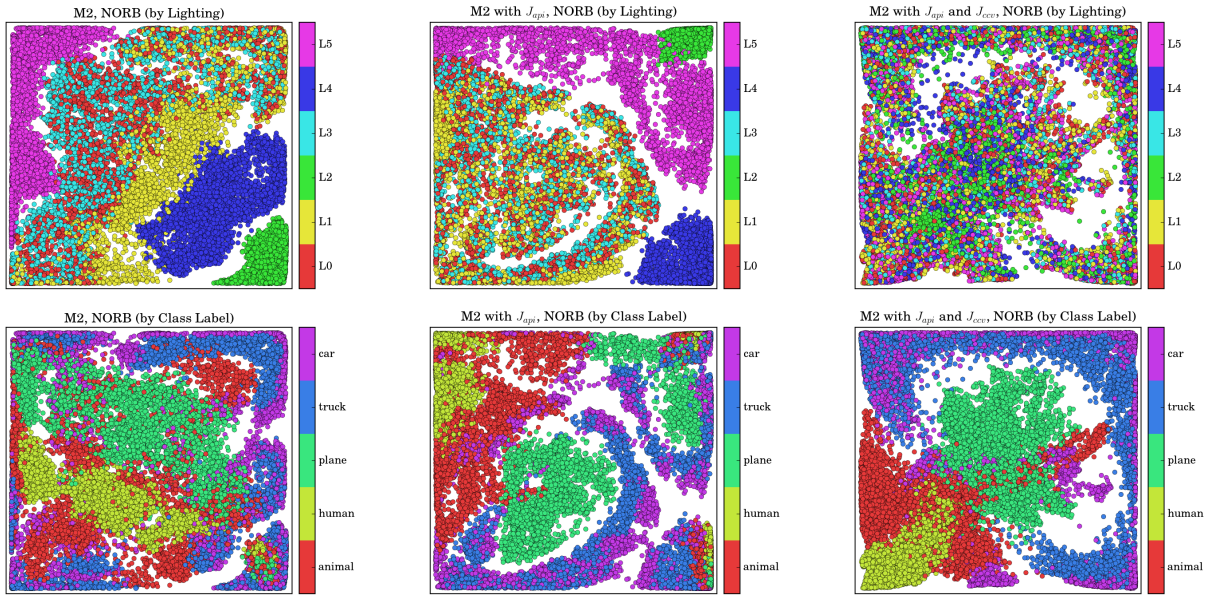


Figure 6.4: Visualizations of M2's Z_L with (a) no proposed objectives added, (b) only J_{api} added, and (c) both J_{api} and J_{ccv} included. The first row colors the latent space by the true lighting labels L and the second row by the true class labels Y. The standard M2 latent space combines only L0 and L3; including J_{api} additionally merges L1 but is otherwise insufficient on its own to merge all of the labels; utilizing both J_{api} and J_{ccv} merges all labels successfully.

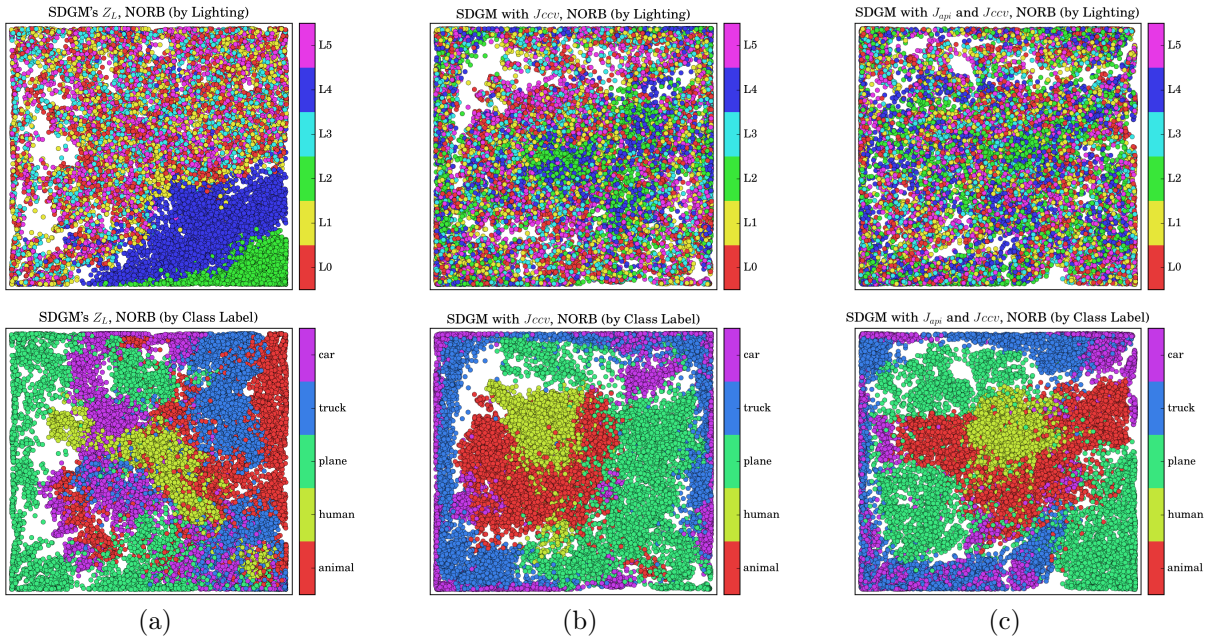


Figure 6.5: Visualizations of SDGM's Z_L with (a) no proposed objectives added, (b) J_{ccv} added, and (c) both J_{api} and J_{ccv} included. The first row colors the latent space by the true lighting labels L and the second row by the true class labels Y. It seems that J_{ccv} is sufficient for merging all of the labels together, but including both J_{api} and J_{ccv} objectively produces a better generative model because J_{api} is not optimized by J_{ccv} alone. Enforcing a priori independence better clusters the data according to true class Y even though Y was never used in training or testing.

that a priori independence could possibly be used to remove nuisance variables such as lighting L when a DGM tries to understand the contents of an image. Further research is needed in a multi-label environment.

The GPLVM [74] dimension reduction we use for plotting is non-deterministic and highly non-linear, thus the cluster locations and shapes are different for each visualization. In all six cases, Z_L is standard normal and would look like Figure 6.3 when using a linear reduction. GPLVM gives us more certainty that a mixture of labels in Z_L is really caused by a priori independence instead of an unlucky linear projection of high-dimensional data.

Choosing the mixing weights of the sub-objectives is relatively simple. We suggest using α_{api} roughly equal to half of α_a, with a rule of thumb provided for α_a in [44]. Both J_a and J_{api} are log-cross-entropies, so setting them to the same order of magnitude seems reasonable. We found the weight α_{ccv} to not be critically important, though setting it too high will prevent the DGM from learning anything at all. We used $\alpha_{ccv} = 25$ for SDGM and $\alpha_{ccv} = 1000$ for M2, though we suggest the smaller of the two to reduce stiffness and sensitivity to small changes in label predictions.

6.6 Conclusion

Without a priori independence, large portions of Deep Generative Models' latent spaces could generate garbage data. We propose two objectives, the objective of a priori independence J_{api} and the conditional covariance objective J_{ccv}, for minimizing the amount of gibberish. We show experimentally that without these objectives, two semi-supervised DGMs M2 and SDGM show clustering in the latent space when trained on the NORB data set even though we expect otherwise based on the intuition of the generative model's construction as well as results from the MNIST data set.

The interactions between the parameters of different parts of a DGM's inference and generative networks and the sub-objectives need to be controlled carefully. Without consideration, the DGM could actively learn to produce garbage and to classify garbage. With our novel training procedure, we showed experimentally that it is possible to train latent spaces that have the desired a priori independence without sacrificing generalization performance.

The top-down construction of J_{api} ensures that generated data from a DGM is inferred correctly, whereas the bottom-up approach of the standard J_{ssl} ensures that observed data is reconstructed faithfully. The two complement each other, and possible future research directions involve mixing J_{api} with GANs as well as testing on new data sets, including those that have multiple labels.

Open-Book Testing and Multi-Label Deep Generative Models

The goal of this chapter is to see if additional information in the NORB data set, specifically elevation information and lighting information, can improve classification accuracy on class labels. Intuitively, having more information should make classification better, but experimentally we find that naively incorporating the additional labels causes performance degradation. It turns out that the task of simultaneously identifying the three labels is more complex than simply identifying a single label.

The NORB data set is chosen not only because it is multi-label but also because of the way the training and testing sets are separated. NORB consists of 50 toys, each of which has been imaged under various lightings, elevations, and rotations. The training set contains all possible variations of 25 toys, and the testing set consists of all possible variations of the remaining toys. This means that the testing set is guaranteed to be statistically different from the training set. SDGM, one of the best non-convolutional classifiers, achieves only about 90% when trying to classify the testing data even when fully supervised.

The unique split between the training and testing sets leads us to question whether the training procedure needs to be separate from the testing procedure. In our semi-supervised framework, merely 1000 images out of the training set's 24300 are labeled. We propose adding the testing set's 24300 images to the training procedure but not including any of the testing set's true labels. This is analogous to a student attending an exam with textbook in hand and with the capability of continuing to learn on the spot. The goal remains to attempt to classify the testing data. We call this proposal open-book testing.

Models that succeed in open-book testing have the potential to adapt and learn in the real world. In this chapter, we examine SDGM and propose five multi-label models: a multi-label extension to SDGM called ML-SDGM, a Stacked model, and three variants of the Stacked model including StackedQP. Of these, SDGM is very successful in open-book testing, but all of the other models perform worse. However, the novel StackedQP model with the previous chapter's API objective outperforms SDGM.

Table 7.1 shows the multi-label models' evaluations regarding the five motivating requirements. The multi-label models have improved scores over the single-label models with regard to steerability. Upon examining SDGM and ML-SDGM's latent spaces, we find that their understandings of the data are not so useful compared to the Stacked model variants; the imagination space contains a lot of gibberish, something that the API objective could remedy. The original Stacked model is the only one to receive full marks for the understanding category because it can incorporate meta-information about the relationships between labels

Table 7.1: Multi-Label Models' Scores

	Data Driven	Semi-supervised	Steerable	Generalization	Adaptability	Total
SDGM	1	1	0.5	0.6	1	4.1
ML-SDGM	1	1	1	0.6	0.5	4.1
Stacked Model	1	1	1	1	0	4.0
StackedQP with API	1	1	1	0.9	1	4.9

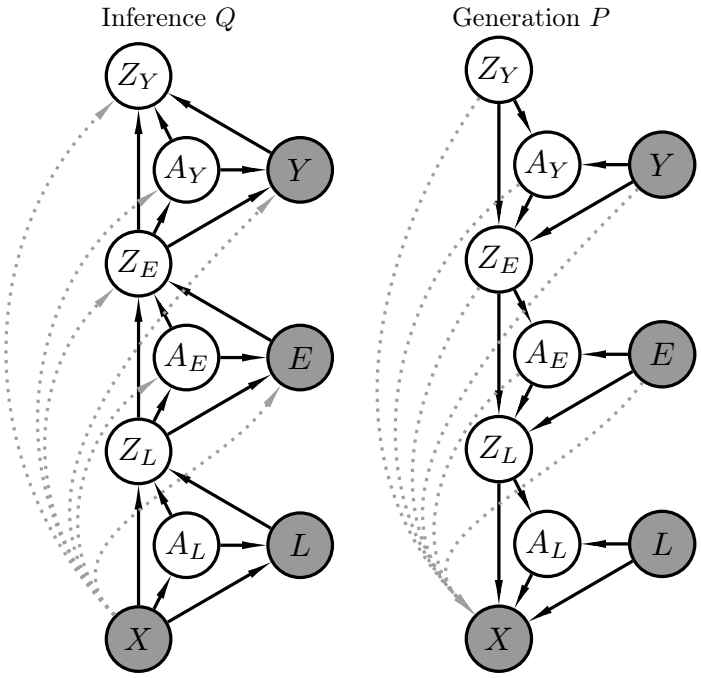
and also because its latent spaces do not need the API objective in order to exhibit the desired independence. However, the Stacked model fails completely at open-book testing. StackedQP with API receives a lower score for understanding because the short-circuit connections harm the model's interpretability, but it outperforms even SDGM in open-book testing, so it receives full marks for adaptability.

Open-book testing and a priori independence demand further exploration. Future work should see which algorithms are and are not successful open-book, and the results should be compared with and without the presence of API.

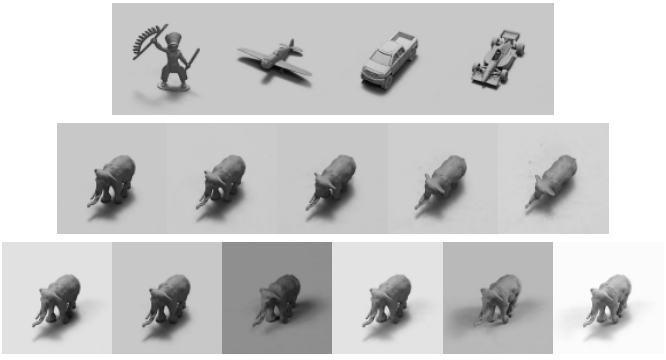
7.1 Introduction

As suggested by the name, DGMs define generative models that describe the data creation process, and, for semi-supervised learning, they typically utilize a discrete latent variable for generating class information coupled with a continuous latent variable for generating all of the variation in the data that is not caused by the class. We extend the typical generative model to incorporate multi-labeled data and specifically experiment on the NORB data set. The NORB data set contains images of objects belonging to different classes Y, imaged under varying camera elevations E, and lit using one of several lighting conditions L. Figure 7.1(b) shows example images from different classes, elevations, and lightings. Our primary motivation is to assess whether or not knowledge of elevation and lighting can improve class label prediction.

Despite the name Deep Generative Models, DGMs are comprised of two networks – a generative network P that generates data from latent variables and an inference network Q that recognizes the values of latent variables given a set of observed data. The two networks are linked together probabilistically using a variational Bayes objective that optimizes the evidence lower bound (ELBO) of the data. Figure 7.1 shows the Stacked model that we propose and analyze in this chapter, and Figure 7.2 shows the graphical models of



(a)



(b)

Figure 7.1: (a) The Stacked model's inference network Q and generation network P are designed to capture information about lighting L, elevation E, and class Y in the NORB data set. (b) Samples from NORB's training images showing four classes, five elevations, and six different lightings. In the networks Q and P, the black arrows define the Stacked model, including the dotted connections in Q yields the StackedQ model, using the dotted connections in P results in the StackedP model, and combining both gives the StackedQP model. The data X always is observed, and in the semi-supervised setting the labels L, E, and Y are observed only for a small portion of the training data.

an existing semi-supervised classifier known as the Skip Deep Generative Model (SDGM) [59] in addition to our proposed multi-label extension ML-SDGM.

Research in DGMs is expanding rapidly, and most of it falls under one of two categories. The first category explores different model architectures and contains Conditional Variational Autoencoders [83] that treat the class label Y as an output rather than an input in the generative model, [35] that adds a stochastic variable used in the generation of both X and Y, [84] that describes the Ladder Variational Autoencoder which has a deep hierarchy of Zs, and [97] that creates a separation of Zs in order to segment foreground from background in images. The second category examines different objectives that can be used to train the models and contains the Importance-Weighted Autoencoder [17] which uses repeated sampling to tighten the ELBO and [64] which extends the multi-sample idea with a new gradient estimator. Tangential to DGMs is the very popular family of Generative Adversarial Models (GANs) [34], and Adversarial Variational Bayes [62] unifies GANs with DGMs, thus putting it into both the alternative architecture and the alternative objective categories of research.

This chapter falls under the alternative model architecture category of research because we are interested in utilizing extra information in the form of additional labels on some of our data points. Classification is about predicting a target Y, which for the NORB data set would typically be the class label – is this an image of an animal, human, car, plane, or truck? We believe that a model that is aware of camera elevation E and lighting L should be able to improve accuracy in predicting Y. For example, an elephant when seen from above may be hard to identify, but its shadow on the floor may serve as a key feature for recognizing it. The Stacked model shown in Figure 7.1(a) is custom-tailored to capture a generation process that incorporates Y, E, and L. Another model, ML-SDGM in Figure 7.2, utilizes the same information in a generic, black-box fashion without making any assumptions about the relationships between the labels.

One significant problem with deep hierarchies like the Stacked model is overfitting. To combat this, we propose open-book testing. Allow the testing data to be used during training, but do not provide any true labels about the test data. The amount of information used in training is analogous to a textbook: semi-supervised training provides answers to only some of the end-of-chapter problems, and open-book testing announces in advance which of the unanswered problems are going to appear on the test. Because test data is included during training, the model can overfit with no bounds as long as it gets the correct answers on the test questions.

The remainder of this chapter is organized as follows. First we briefly describe Deep Generative Models and note the burden that is added for multi-label data. Next we demonstrate the effect of open-book testing on SDGM, a single-label model. Afterwards we design the multi-label extension ML-SDGM as well as the

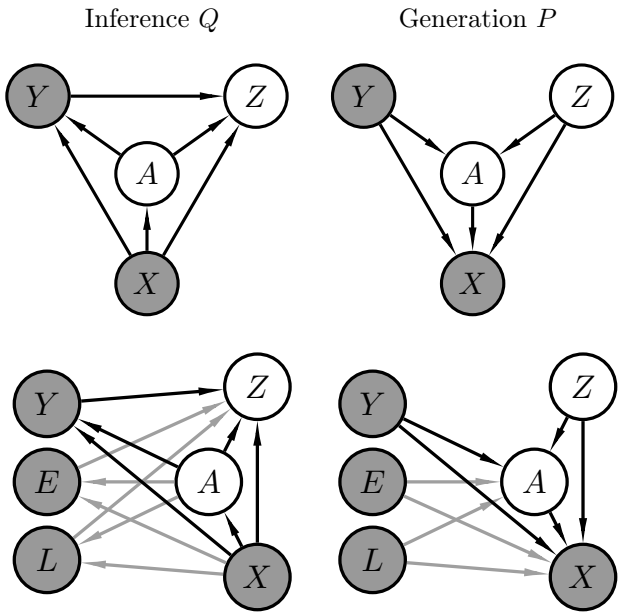


Figure 7.2: Inference network Q and generative network P of two DGMs, the existing Skip Deep Generative Model (SDGM, above) and our proposed multi-label extension ML-SDGM, below. In ML-SDGM, the black arrows are the connections that are present in the standard SDGM and the gray arrows are the dependencies that incorporate the additional labels for elevation E and lighting L.

custom-tailored Stacked model. Finally we provide experimental results along with a discussion on our findings.

7.2 Brief Overview of Deep Generative Models

Deep Generative Models define probabilistic models P that aim to capture the generation process of observed data X using a set of latent variables $\boldsymbol{Z}$. A simple semi-supervised classifier would produce data using $\boldsymbol{Z} = \{Y, Z\}$ where Y is a discrete class label and Z is a continuous latent variable. Additionally, DGMs define a recognition model Q that helps to infer the values of $\boldsymbol{Z}$ given X. The parameters of the generative joint-distribution $p(X, \boldsymbol{Z})$ and the inference conditional-distribution $q(\boldsymbol{Z}|X)$ are learned by optimizing the evidence lower bound (ELBO) of the data.

$$\log p(X) \geq E_{q(\boldsymbol{Z}|X)} \log \frac{p(X, \boldsymbol{Z})}{q(\boldsymbol{Z}|X)} \tag{7.1}$$

The two distributions p and q are assumed to factorize according to the probabilistic graph of a DGM. For example, ML-SDGM's graphs, shown in Figure 7.2, are written out as p_{ML} and q_{ML}.

$$\begin{aligned} p_{ML}(X, \boldsymbol{Z}) &= p(L)\ p(E)\ p(Y)\ p(Z) \\ &\quad p(A|L, E, Y, Z) \\ &\quad p(X|L, E, Y, A, Z) \end{aligned} \tag{7.2}$$

$$\begin{aligned} q_{ML}(\boldsymbol{Z}|X) &= q(A|X) \\ &\quad q(L|A, X)\ q(E|A, X)\ q(Y|A, X) \\ &\quad q(Z|L, E, Y, A, X) \end{aligned} \tag{7.3}$$

Each factored distribution is either Gaussian (for the continuous variables X, A, and Z) or multinomial (for the discrete variables L, E, and Y), and the statistics of each distribution are the outputs of neural network functions. The ELBO serves as the cost function for guiding gradient descent on the weights and biases of the neural networks.

In the semi-supervised setting, the ELBO is evaluated in two different cases, one for labeled data and another for unlabeled data. When L, E, and Y are known corresponding to a given data point X, the labeled objective J_l evaluates the expectation in Equation 7.1 as integration over A and Z. The objective J_u does not know any of the labels corresponding to a given point X, so in this case the expectation requires a double integral over A and Z as well as a triple summation that enumerates all possible L, E, and Y. This triple enumeration can cause a significant computational burden compared to a single-label model depending on the model architecture. The integrals over A and Z can be approximated via sampling, and typically one sample for each is sufficient provided that a batch of data is processed simultaneously.

Four sub-objectives need consideration, the labeled J_l and unlabeled J_u as well as J_a and J_{api}. The classifiers that predict the labels are used only when the labels are not known, so J_a ensures that the known labels are classified correctly. The objective J_{api} enforces a priori independence and is a top-down strengthening of the DGM. It generates data using randomly chosen labels and then ensures that the inferred labels of the generated data are correct.

This overview of DGMs is *very* brief, and we refer readers to [44, 45, 58, 59] for more detail. We created the a priori independence objective J_{api} to help with multi-label DGMs, but in this chapter we simply utilize the objective; its formulation and reasonings are provided elsewhere.

Table 7.2: Summary of SDGM's open-book testing performance. The tabulated sizes are 24300 training data points plus varying portions of the 24300 unlabeled test data. Notice that accuracy of class Y predictions increases as more open-book test data is included. The fully open-book test uses 48600 points when training and should be compared with Table 7.3.

Model	Training Size	Acc. (%)
SDGM	24300	89.00
	25515	94.40
	26730	95.23
	30375	97.33
	36450	98.27
	42525	98.77
	48600	**99.27**

7.3 Open-Book Testing with a Single Label

In semi-supervised learning with a single target label Y and data X, the training and testing data are split into several subsets. The training data is broken into X_u and X_l, where X_l contains significantly fewer points than X_u. For our experiments on the NORB data set, X_u contains 24300 images and X_l has merely 1000. The true labels Y_l are provided for the points X_l, but the true labels Y_u corresponding to the full unlabeled set are not known, hence X_u must be treated as unlabeled data. The testing data X_t is paired with true labels Y_t, and for NORB, this is another 24300 points.

Traditionally, neither X_t nor Y_t are used during the training process, but we propose open-book testing to utilize some or all of X_t in addition to X_u as unlabeled points. None of the true labels Y_t are provided when training. This form of testing is possible only because DGMs are semi-supervised and can leverage the additional unlabeled data.

Table 7.2 shows classification results of training SDGM with varying amounts of X_t mixed into the training process. When the training size is 24300, the model is tested in a fully closed-book fashion, and when the training size is 48600, testing is fully open-book. Notice that classification performance increases as the proportion of open-book data increases. All of the experiments are stopped after 100 epochs, but with further training it might be possible to obtain up to 90.6% accuracy closed-book on the NORB data set using SDGM [59] up from our result of 89.00%. The fully open-book result of 99.27% is the baseline to which we compare our multi-labeled results in the following sections.

7.4 Multi-Label Models

We propose five multi-label DGMs, one of which is a black-box extension of SDGM with the remaining four modeling the generative process that produces photos of real objects. The extension of SDGM is ML-SDGM, its graphical models are shown in Figure 7.2, and its factorized joint and conditional distributions are provided

in Equation 7.2 and Equation 7.3. It uses common continuous latent variables A and Z and has radial branches for each of the discrete variables L, E, and Y. There is no interaction between the three discrete labels except through the common continuous variables and the data X. Extending SDGM in this fashion is straight-forward, but no clear interpretation exists for the structure of this model.

The Stacked model, shown in Figure 7.1(a) while ignoring the dotted connections, has a very deliberate and designed interpretation. In the generative direction, the topmost layer pairs the discrete class Y with a continuous variable Z_Y. Fixing Y to correspond to the class of cars means that varying Z_Y will choose different kinds of cars. The immediate output of the topmost layer is Z_E, a continuous latent variable that expresses class information but does not yet have any information regarding elevation or lighting. It, when coupled with camera elevation E, produces the output Z_L. The continuous latent variable Z_L expresses both class and elevation information, and when paired with lighting L, finally can generate actual images X. Observed images X express information about class, elevation, and lighting simultaneously. The generative joint distribution is $p_S(X, \boldsymbol{Z})$ for the Stacked model.

$$
\begin{aligned}
&p_S(X, \boldsymbol{Z}) = \\
&\quad p(Y)\ p(Z_Y)\ p(A_Y|Z_Y, Y) \\
&\quad p(E)\ p(Z_E|Z_Y, Y, A_Y)\ p(A_E|Z_E, E) \\
&\quad p(L)\ p(Z_L|Z_E, E, A_E)\ p(A_L|Z_L, L) \\
&\quad p(X|Z_L, L, A_L)
\end{aligned} \tag{7.4}
$$

While present in the generative model, the auxiliary variables A_L, A_E, and A_Y do not have a clear role.

The inference network of the Stacked model reverses the generative process and also utilizes auxiliary variables A_L, A_E, and A_Y to help classification performance and store data statistics. Beginning at the bottom of the stack with data X, first A_L and lighting L are inferred. Once lighting is known, the resulting Z_L takes a representation that can ignore lighting information but retains class and elevation information. Moving up the stack causes A_E and elevation E to be inferred, with the resulting Z_E no longer needing to carry lighting or elevation information. The topmost layer infers A_Y and class Y and results in the final Z_Y, an abstract representation of the object in an image X. The conditional inference distribution is $q_S(\boldsymbol{Z}|X)$ for

Table 7.3: Summary of open-book testing performances for different DGMs. All are semi-supervised with only 1000 labeled examples in the training data. SDGM uses just the class label Y, but ML-SDGM and the Stacked models use class Y, elevation E, and lighting L. Reported accuracies are for class label Y predictions. The proposed multi-label models perform worse than the single-label model unless a priori independence is enforced via J_{api} and all shortcuts are connected.

Model	Epochs	Time (hours)	Acc. (%)
SDGM	100	0.96	99.27
SDGM	600	5.37	99.30
ML-SDGM	100	2.71	97.57
Stacked	100	4.30	81.06
StackedQ	100	4.69	80.30
StackedP	100	7.82	94.80
StackedQP	100	7.89	98.17
StackedQP with J_{api}	100	8.44	**99.67**

the Stacked model.

$$
\begin{aligned}
&q_S(\boldsymbol{Z}|X) = \\
&\quad q(A_L|X)\ q(L|X,A_L)\ q(Z_L|X,A_L,L) \\
&\quad q(A_E|Z_L)\ q(E|Z_L,A_E)\ q(Z_E|Z_L,A_E,E) \\
&\quad q(A_Y|Z_E)\ q(Y|Z_E,A_Y)\ q(Z_Y|Z_E,A_Y,Y)
\end{aligned} \tag{7.5}
$$

The Stacked model has a natural interpretation, but it is *extremely* difficult to train. Between p_S and q_S, fifteen separate neural networks are necessary to parameterize all of the distributions. A very lucky run yielded approximately 88% closed-book testing accuracy, and its latent spaces Z_Y, Z_E, and Z_L are visualized in Figure 7.3. Notice that each successive layer in the generative direction adds one source of information, all of which is expressed in the final output X. The Stacked model more typically reaches about 80% classification accuracy as shown in Table 7.3.

To improve classification performance of the Stacked model, we propose three variants: StackedQ, StackedP, and StackedQP. The StackedQ model uses the standard generative network p_S but has additional short-circuit connections between the data X and every latent variable in the inference network, resulting in q_{SQ}.

$$
\begin{aligned}
&q_{SQ}(\boldsymbol{Z}|X) = \\
&\quad q(A_L|X)\ q(L|X,A_L)\ q(Z_L|X,A_L,L) \\
&\quad q(A_E|Z_L,X)\ q(E|Z_L,A_E,X)\ q(Z_E|Z_L,A_E,E,X) \\
&\quad q(A_Y|Z_E,X)\ q(Y|Z_E,A_Y,X)\ q(Z_Y|Z_E,A_Y,Y,X)
\end{aligned} \tag{7.6}
$$

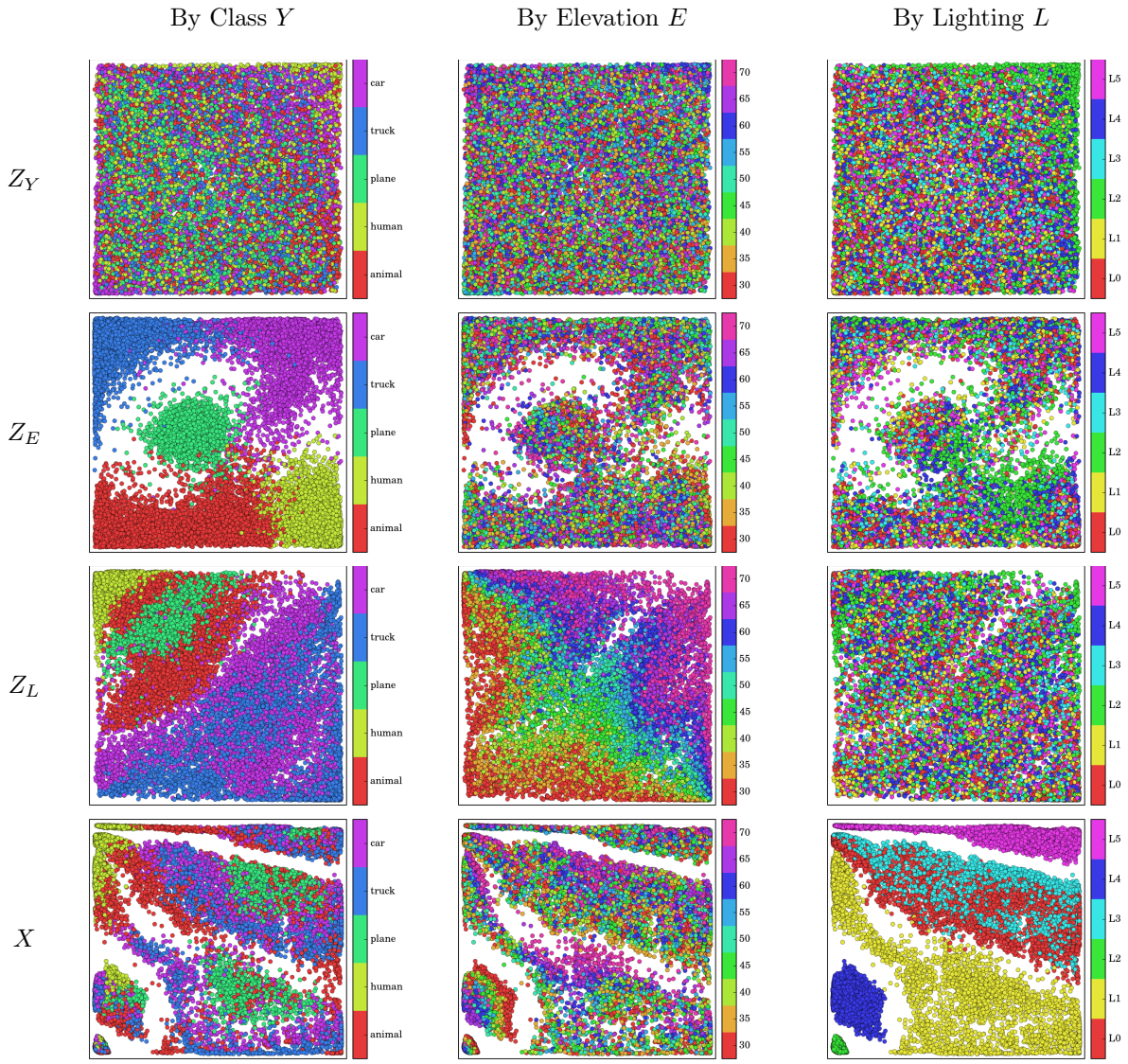


Figure 7.3: Visualizations of the raw NORB images X and their inferred positions in the Stacked model's latent spaces Z_L, Z_E, and Z_Y. Notice that the raw data exhibits clear clustering when colored by true lighting L, slightly less structure when colored by true class label Y, and some banding when colored by true elevation E. The lowest latent variable Z_L removes structure corresponding to lighting and retains patterns caused by elevation and class. The next variable Z_E removes elevation and retains only class. The topmost variable Z_Y contains no structure, as expected because in the generative model Z_Y, Y, E, and L are free parameters that can be chosen independently. Refer to Figure 7.1(a) and ignore the dotted connections for the probabilistic graphs P and Q that define the Stacked model.

The StackedP model uses the standard inference network q_S but contains short-circuits between every latent variable and X in the generative network, resulting in p_{SP}, a joint distribution that is identical to p_S except for the factor relating to X.

$$
\begin{aligned}
p_{SP}(X, \boldsymbol{Z}) = & \\
& p(Y)\ p(Z_Y)\ p(A_Y|Z_Y, Y) \\
& p(E)\ p(Z_E|Z_Y, Y, A_Y)\ p(A_E|Z_E, E) \\
& p(L)\ p(Z_L|Z_E, E, A_E)\ p(A_L|Z_L, L) \\
& p(X|Z_L, L, A_L, Z_E, E, A_E, Z_Y, Y, A_Y)
\end{aligned} \tag{7.7}
$$

The StackedQP model has short circuits in both the inference and the generative networks. The short circuits are represented as dotted connections in Figure 7.1(a). We find that the short circuits can indeed improve classification accuracy. However, the cost is that the models' interpretations become less clear, especially in the case of StackedQP where the spaces of the internal variables A_L, Z_L, A_E, and Z_E seem to collapse and are never used, presumably because information can flow to the necessary locations without their involvement. With all of the short circuits in place, the StackedQP model is very similar to ML-SDGM, both of which are difficult to interpret.

7.5 Experimental Results

The classification target of our experiments is always the class label Y, but we are interested in seeing the effect of the additional elevation E and lighting L labels. All of our experiments are trained open-book on the NORB data set, meaning that the test data's images X_t are seen during training but none of the true labels Y_t, E_t, or L_t are known. The results of all experiments are collected in Table 7.3.

As discusses previously, the Stacked model performs poorly at about 80% despite the nice interpretation of the model. Adding short circuits to the inference network in StackedQ's model does not seem to improve the result at all, implying that the Stacked generative model is difficult to learn. The short circuits on the generative network in StackedP's result shows a dramatic improvement in accuracy up to nearly 95%, further providing evidence that the generative network is the bottleneck. Adding short circuits to both the generative and inference networks yields the best results, with StackedQP achieving just over 98%.

SDGM's single-label classification accuracy of 99.27% open-book is the baseline of comparison for all of the other models, and none of the other models reaches this level of performance. The multi-label extension

ML-SDGM achieves 97.5%, nearly 2% lower than SDGM. The best StackedQP is slightly better at 98.1% but is still a full 1% below SDGM. Clearly, adding more labels causes difficulties rather than providing benefits.

The only multi-label model we tested that outperforms the single-label SDGM is StackedQP with the additional objective J_{api} added. The objective tries to enforce a priori independence and in the process ensures that generated images have the correct labels inferred. The coordination between the generative network and the inference network seems to be of critical importance for mixing together multiple labels.

Table 7.3 shows the computation time of training each model. All experiments are run on an i7 4970k CPU and a GTX 980 GPU. All of the continuous latent variables have dimension of 100, and all neural networks use rectified linear units with one hidden layer of 500 neurons. No batch normalization was used in the networks. StackedQP with J_{api} takes nearly nine times longer to train than the single-label SDGM.

7.6 Discussion

The fact that SDGM on the NORB data set reaches as high as 99% open-book is surprising. The gradation of its accuracies with varying amounts of open-book data also is surprising. We suspect that this relates to the way that the NORB data set is split between training and testing. There are a total of 50 objects, 10 from each of the five classes (animal, car, plane, truck, human), and each object is imaged under camera position and lighting variations. The training set contains all possible variations of 25 of the 50 objects, and the testing set contains all possible variations of the other 25 objects. This causes the testing data to be statistically different from the training data. For example, there are no dinosaurs in the 25 objects that constitute the training data, but there are two dinosaurs in the 25 objects of the testing data.

Several curiosities arise when considering the "completeness" of the NORB training and testing data sets as well as the way the two sets are separated, and further research is necessary to resolve those issues. First, the multi-label models we designed in this chapter need to be tested on other data sets in order to see if the same burden caused by additional labels exists elsewhere. And second, the multi-labeled experiments need to be performed using semi-supervised frameworks other than DGMs such as ladder networks and GAN-based methods.

7.7 Conclusion

Asking a classifier to predict three separate labels is more difficult than asking it to classify a single label. SDGM, the single-label classifier, reaches 99.3% accuracy on open-book tests of the NORB data set, but our multi-label extension ML-SDGM and our proposed StackedQP model perform worse at 97.5% and 98.1%, respectively. The presence of additional labels improves accuracy only when the a priori independence

objective is added to StackedQP. The open-book test results of StackedQP with J_{api} are nearly perfect at 99.7%.

Future research can branch in several directions. One direction is to test the multi-label models we propose on other data sets. Another direction is to test other semi-supervised frameworks such as ladder networks and GAN-based methods in the same fashion as this chapter's experiments. A third direction is to attempt to improve the Stacked model without adding short circuits so that the model's interpretation remains intact. A fourth direction is to design semi-supervised classifiers that continuously train and test on incoming data in an online fashion while using open-book testing.

Concluding Remarks

The multi-label model StackedQP with a priori independence performs nearly flawlessly on open-book testing, producing 99.7% accuracy on class labels in the testing data. This is remarkable because of the way that the data set's training and testing objects are separated; the model was never told that dinosaurs are animals and not cars, but it learned the correct classification on its own. Furthermore, the model was never given information about what is the top of a dinosaur and what is its side, but by seeing enough data, the model correctly reasoned about the relationships. Without a priori independence (API), multi-label models are burdened by the need to predict elevation information information in addition to class label, but the top-down strengthening of API utilizes extra information to great effect in the StackedQP model.

The relationships among the Stacked variants demand further research. The original Stacked model's latent spaces are exactly what one would expect, as seen in Figure 7.3, but its classification accuracy is very poor. On the other extreme, the StackedQP model performs much better, but its latent spaces are completely collapsed (not shown in this dissertation) with the exception of the top-most layer, indicating that what it learns is not very different from SDGM or ML-SDGM that have only one free latent variable. Comparing StackedQ to StackedP indicates that a significant factor contributing to classification accuracy seems to be generative expressiveness. All of these models need to be explored further with the addition of API.

This dissertation's models progressively tackle the five motivating requirements, summarized in Table 8.1. The StackedQP model with API incorporates every required element and reaches a score of 4.9 out of 5. Compared to SDGM, StackedQP is more steerable because it coordinates multiple labels, and compared to ML-SDGM, StackedQP has better classification accuracy, thus making StackedQP with API the best model considered in this work. It is not without faults, however. The two primary issues are computation time for training and the fact that the short-circuit connections weaken the interpretability of the model. Suggestions to improve both of these, as well as several other directions for proceeding, are provided in the following section on future research ideas.

8.1 Ideas for Future Research

Compiling this document and its results has inspired new ideas and pointed out some thoughts that need clarification or further development. Some of the ideas are small items that close gaps in this dissertation, others span the connections between different parts of this work, and a few extend beyond but use this

Table 8.1: Summary of all models' scores in relation to the five motivating requirements.

	Data Driven	Semi-supervised	Steerable	Generalization	Adaptability	Total
Hybrid Model	1	1	0.5	0	0	2.5
GPLVM	1.1	1	0.5	0.3	0	2.9
M1	1	0	0	0.3	0	1.3
M2	1	1	0.5	0.6	0	3.1
SDGM	1	1	0.5	0.6	1	4.1
SDGM with API	1	1	0.5	0.9	1	4.4
ML-SDGM	1	1	1	0.6	0.5	4.1
Stacked Model	1	1	1	1	0	4.0
StackedQP with API	1	1	1	0.9	1	4.9

research as a base of comparison. The following collections are grouped approximately by scope or difficulty level.

8.1.1 Small

The multi-label models were tested on the NORB data set, however ML-SDGM is generic and the assumptions in the Stacked model are reasonably applicable to vision/camera data. It should be relatively simple but also fruitful to examine the current collection of multi-label models on other data sets.

Even within the NORB data sets examined in this work, more analysis can be obtained regarding the characteristics of a priori independence. Specifically, its generalization qualities need further study. Exploring this would involve adding API to ML-SDGM, the Stacked model, StackedQ, and StackedP all while performing varying degrees of open-book testing like in Table 7.2.

A priori independence may be useful in a completely unsupervised fashion. Consider Seq Z, Split Z, and Full Z from Chapter 4. In those models, especially Full Z, a top-down round trip could potentially steer the learned representations to have more consistency as well as more separation between adjacent latent variables.

The computation times of the multi-label models are incredibly high due to the enumerations necessary for the expectations over discrete variables. These enumerations can be approximated with a stochastic reparameterization for discrete labels [41] which should significantly speed up the training time. Ideally the speed benefit would offset any potential degradation due to approximation.

It may be possible to restore StackedQP's interpretability by slowly disconnecting the short circuits. Over the course of training, the shorts circuits' weights could be driven to zero so that the model shifts its understanding toward the Stacked model's structure.

8.1.2 Medium

The NORB data set's unique split between training and testing objects means that the both sets are relatively "complete" in the sense that they contain all possible lightings, elevations, and azimuths of every object. It would be interesting to see results using just a single lighting for a new object while still including all elevations and azimuths. Practically, this would correspond to a more realistic situation that could result from a video stream from a camera navigating around an object but without control over the surroundings of the object. This fixed-lighting scenario could easily be incorporated in open-book testing and would be a real test of the models' generative abilities to imagine the other lighting conditions.

The latent spaces that GPLVM learns (Figure 3.5) are visually more pleasing than those in M1 (Figure 4.3) even though both are unsupervised. In GPLVM, all points interact with all other points in the latent space thus capturing some global features of the data. In M1, the only interaction happens between neighboring areas when a latent point's variance is not yet solidified. Perhaps M1 could be improved by including a global interaction.

The covariance function in GPLVM (Equation 3.2) captures the distances between points, but this causes edge and corner effects when the output layer uses a finite-range nonlinearity like sigmoid. A periodic covariance function could possibly remove edge effects by forcing sides of the space to interact with opposite sides as if they were connected. This could perhaps make the dimension reductions more unique with variation only in rotation. The current edge limitations cause both rotation and multi-modal variations.

8.1.3 Large

The quality of the Stacked model and StackedQP's latent spaces could be improved by allowing each layer to have hierarchical inputs and outputs. Currently, they require Z_Y, Y, E and L to be specified in order to generate data. It might be beneficial to create an architecture that permits just the bottom layer with Z_L and L or two layers with Z_E, E, and L to bypass the entire stack. This would cause every layer to maintain more information about the entire data space and might prevent the collapsing of latent spaces seen in StackedQP. In addition to developing the architecture, more experimentation is needed to verify if this is desirable at all.

Because GPLVM and M1 are functionally similar, it may be possible to replace M1 as the core of DGMs. Potentially, the training time could be reduced significantly because GPLVM does not need to learn an

explicit generative network. The generative network seems to provide a good link between supervised and unsupervised sets of data, but it remains to be determined whether a generative network is necessary at all when the primary goal is inference.

Open-book testing leads in the direction of online or lifelong learning. However, the dream of lifelong learning is severely hampered by fixed-architecture models. Currently, the universe of our models is roughly fixed in scope. There are predetermined classes, such as animal, car, plane, etc., that the algorithms learn to recognize, and they can reason about dinosaurs even though we never provided explicit knowledge of dinosaurs because they are seen as a subclass of animals. How would we incorporate new classes, such as furniture? Similarly, what if there are new concepts in the form of additional tags? For example, what if a model could detect animals and furniture, but we introduce the concept of legs? Could it envision a three-legged animal? A three legged-chair?

8.2 Takeaways

The main contributions of this dissertation are the tensorized framework for Deep Generative Models, the a priori independence criteria, several multi-label models, and the concept of open-book testing. The tensorized framework trivializes exploration of DGM architectures and can easily be incorporated into any existing neural network tool chain. A priori independence can be added to any semi-supervised DGM, and its top-down approach utilizes existing inference networks to solidify generation capabilities. Naive extensions of DGMs to incorporate multiple labels result in performance degradation that can be overcome with careful modeling and a priori independence. Open-book testing is a paradigm shift that challenges whether training should end or whether training and testing are separate phases. The tools and models developed in this dissertation help to solve big data challenges of data volume, velocity, and complexity.

References

[1] Chainer implementation of auxiliary deep generative models (adgm) and skip deep generative model (sdgm). https://github.com/musyoku/adgm. Accessed: 2018-03-25.

[2] Pytorch's convolutional layer documentation. http://pytorch.org/docs/master/nn.html#torch.nn.Conv2d. Accessed: 2018-03-26.

[3] Tensorflow's convolutional layer documentation. https://www.tensorflow.org/api_docs/python/tf/nn/conv2d. Accessed: 2018-03-26.

[4] Theano's convolutional layer documentation. http://deeplearning.net/software/theano/library/tensor/nnet/conv.html. Accessed: 2018-03-26.

[5] Variational inference for monte carlo objectives. https://github.com/y0ast/VIMCO. Accessed: 2018-03-25.

[6] Vq-vae (neural discrete representation learning) tensorflow. https://github.com/hiwonjoon/tf-vqvae. Accessed: 2018-03-25.

[7] What is big data? https://www-01.ibm.com/software/in/data/bigdata/. Accessed: 2018-03-20.

[8] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

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

[10] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[11] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[12] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[13] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.

[14] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[15] Alex Beutel, Jilin Chen, Zhe Zhao, and Ed H Chi. Data decisions and theoretical implications when adversarially learning fair representations. *arXiv preprint arXiv:1707.00075*, 2017.

[16] Ali Borji, Saeed Izadi, and Laurent Itti. ilab-20m: A large-scale controlled object dataset to investigate deep learning. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2221–2230. IEEE, 2016.

[17] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015. `https://github.com/yburda/iwae/blob/master/iwae.py`.

[18] Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

[19] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[20] Olivier Chapelle and Alexander Zien. Semi-supervised classification by low density separation. In *Proceedings of the tenth international workshop on artificial intelligence and statistics*, volume 1, pages 57–64, 2005.

[21] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[22] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, pages 3642–3649. IEEE, 2012.

[23] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237. Barcelona, Spain, 2011.

[24] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Juergen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *arXiv preprint arXiv:1003.0358*, 2010.

[25] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.

[26] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[27] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[28] Luke de Oliveira, Michela Paganini, and Benjamin Nachman. Learning particle physics by example: location-aware generative adversarial networks for physics synthesis. *Computing and Software for Big Science*, 1(1):4, 2017.

[29] Alexey Dosovitskiy, Jost Tobias Springenberg, Maxim Tatarchenko, and Thomas Brox. Learning to generate chairs, tables and cars with convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(4):692–705, 2017.

[30] Alexandre X Falcão, Jorge Stolfi, and Roberto de Alencar Lotufo. The image foresting transform: Theory, algorithms, and applications. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(1):19–29, 2004.

[31] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[32] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[33] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[34] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[35] Jonathan Gordon and José Miguel Hernández-Lobato. Bayesian semisupervised learning with deep generative models. *arXiv preprint arXiv:1706.09751*, 2017. `https://github.com/Gordonjo/generativeSSL/blob/master/models/sdgm.py`.

[36] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005.

[37] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[38] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425, 2002.

[39] Xun Huang, Yixuan Li, Omid Poursaeed, John Hopcroft, and Serge Belongie. Stacked generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, page 4, 2017.

[40] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[41] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[42] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *ICML*, volume 99, pages 200–209, 1999.

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

[44] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014. `https://github.com/dpkingma/nips14-ssl/blob/master/learn_yz_x_ss.py`.

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

[46] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[47] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[48] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *arXiv preprint arXiv:1603.00788*, 2016.

[49] Nicholas D Lane and Petko Georgiev. Can deep learning revolutionize mobile sensing? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 117–122. ACM, 2015.

[50] Julia A Lasserre, Christopher M Bishop, and Thomas P Minka. Principled hybrids of generative and discriminative models. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 87–94. IEEE, 2006.

[51] Neil D Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. *Advances in neural information processing systems*, 16(3):329–336, 2004.

[52] Neil D Lawrence and Joaquin Quiñonero-Candela. Local distance preservation in the gp-lvm through back constraints. In *Proceedings of the 23rd international conference on Machine learning*, pages 513–520. ACM, 2006.

[53] Quoc V Le, Rajat Monga, Matthieu Devin, Kai Chen, Greg S Corrado, Jeff Dean, and Andrew Y Ng. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[54] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[55] Yann LeCun, Fu Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–104. IEEE, 2004.

[56] Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. The variational fair autoencoder. *arXiv preprint arXiv:1511.00830*, 2015.

[57] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[58] Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Improving semi-supervised learning with auxiliary deep generative models. In *NIPS Workshop on Advances in Approximate Bayesian Inference*, 2015.

[59] Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016. `https://github.com/larsmaaloee/auxiliary-deep-generative-models/blob/master/models/sdgmssl.py`.

[60] Davide Maltoni and Vincenzo Lomonaco. Semi-supervised tuning from temporal coherence. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 2509–2514. IEEE, 2016.

[61] Michael F Mathieu, Junbo Jake Zhao, Junbo Zhao, Aditya Ramesh, Pablo Sprechmann, and Yann LeCun. Disentangling factors of variation in deep representation using adversarial training. In *Advances in Neural Information Processing Systems*, pages 5040–5048, 2016.

[62] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *arXiv preprint arXiv:1701.04722*, 2017.

[63] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *arXiv preprint arXiv:1704.03976*, 2017.

[64] Andriy Mnih and Danilo Rezende. Variational inference for monte carlo objectives. In *International Conference on Machine Learning*, pages 2188–2196, 2016.

[65] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[66] Eric Nalisnick and Padhraic Smyth. Deep generative models with stick-breaking priors. *arXiv preprint arXiv:1605.06197*, 2016. `https://github.com/enalisnick/stick-breaking_dgms`.

[67] Hariharan Narayanan and Sanjoy Mitter. Sample complexity of testing the manifold hypothesis. In *Advances in Neural Information Processing Systems*, pages 1786–1794, 2010.

[68] Santiago Pascual, Antonio Bonafonte, and Joan Serra. Segan: Speech enhancement generative adversarial network. *arXiv preprint arXiv:1703.09452*, 2017.

[69] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In *European conference on computer vision*, pages 143–156. Springer, 2010.

[70] Nikolaos Pitelis, Chris Russell, and Lourdes Agapito. Semi-supervised learning using an unsupervised atlas. In *Machine Learning and Knowledge Discovery in Databases*, pages 565–580. Springer, 2014.

[71] Anushree Priyadarshini et al. A map reduce based support vector machine for big data classification. *International Journal of Database Theory and Application*, 8(5):77–98, 2015.

[72] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in neural information processing systems*, pages 2352–2360, 2016.

[73] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015.

[74] Rastin Rastgoufard and AbdulRahman Alsamman. GPLVM and lava floor distance for label-deficient semi-supervised learning: Case study. In *Information Fusion (FUSION), 2016 19th International Conference on*, pages 84–90. ISIF, 2016.

[75] Rastin Rastgoufard and AbdulRahman Alsamman. Hybrid autoencoder and classifier for label-deficient semi-supervised learning: Case study. In *Information Fusion (FUSION), 2016 19th International Conference on*, pages 79–83. ISIF, 2016.

[76] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[77] Salah Rifai, Yann N Dauphin, Pascal Vincent, Yoshua Bengio, and Xavier Muller. The manifold tangent classifier. In *Advances in Neural Information Processing Systems*, pages 2294–2302, 2011.

[78] Salah Rifai, Grégoire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. Higher order contractive auto-encoder. In *Machine Learning and Knowledge Discovery in Databases*, pages 645–660. Springer, 2011.

[79] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[80] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[81] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55, 2016.

[82] Kevin Schawinski, Ce Zhang, Hantian Zhang, Lucas Fowler, and Gokula Krishnan Santhanam. Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit. *Monthly Notices of the Royal Astronomical Society: Letters*, 467(1):L110–L114, 2017.

[83] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pages 3483–3491, 2015.

[84] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 3738–3746, 2016.

[85] Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015.

[86] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[87] Masahiro Suzuki, Kotaro Nakayama, and Yutaka Matsuo. Joint multimodal learning with deep generative models. *arXiv preprint arXiv:1611.01891*, 2016.

[88] Masahiro Suzuki, Kotaro Nakayama, and Yutaka Matsuo. Improving bi-directional generation between different modalities with variational autoencoders. *arXiv preprint arXiv:1801.08702*, 2018.

[89] Attila Szabó, Qiyang Hu, Tiziano Portenier, Matthias Zwicker, and Paolo Favaro. Challenges in disentangling independent factors of variation. *arXiv preprint arXiv:1711.02245*, 2017.

[90] Jakub M Tomczak and Max Welling. VAE with a VampPrior. *arXiv*, 2017. `https://github.com/jmtomczak/vae_vampprior`.

[91] Dustin Tran, Matthew D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, and David M. Blei. Deep probabilistic programming. In *International Conference on Learning Representations*, 2017.

[92] Raquel Urtasun and Trevor Darrell. Discriminative gaussian process latent variable model for classification. In *Proceedings of the 24th international conference on Machine learning*, pages 927–934. ACM, 2007.

[93] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science and Engg.*, 13(2):22–30, March 2011.

[94] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[95] Thomas Wiecki. Bayesian deep learning part ii: Bridging pymc3 and lasagne to build a hierarchical neural network. `http://twiecki.github.io/blog/2016/07/05/bayesian-deep-learning/`, 07 2016.

[96] Jason Wolfe, Aria Haghighi, and Dan Klein. Fully distributed em for very large datasets. In *Proceedings of the 25th international conference on Machine learning*, pages 1184–1191. ACM, 2008.

[97] Xinchen Yan, Jimei Yang, Kihyuk Sohn, and Honglak Lee. Attribute2image: Conditional image generation from visual attributes. In *European Conference on Computer Vision*, pages 776–791. Springer, 2016. `https://github.com/xcyan/eccv16_attr2img`.

[98] Yongxin Yang and Timothy Hospedales. Deep multi-task representation learning: A tensor factorisation approach. *arXiv preprint arXiv:1605.06391*, 2016.

[99] Yongxin Yang and Timothy M Hospedales. A unified perspective on multi-domain and multi-task learning. *arXiv preprint arXiv:1412.7489*, 2014.

[100] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858, 2017.

[101] Matthew D Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[102] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *International Conference on Machine Learning*, pages 325–333, 2013.

[103] Junbo Zhang, Guangjian Tian, Yadong Mu, and Wei Fan. Supervised deep learning with auxiliary networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 353–361. ACM, 2014.

[104] Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE transactions on Knowledge and Data Engineering*, 18(10):1338–1351, 2006.

[105] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Towards deeper understanding of variational autoencoding models. *arXiv preprint arXiv:1702.08658*, 2017. `https://github.com/ermongroup/Generalized-PixelVAE`.

[106] Tiancheng Zhao, Ran Zhao, and Maxine Eskenazi. Learning discourse-level diversity for neural dialog models using conditional variational autoencoders. *arXiv preprint arXiv:1703.10960*, 2017. `https://github.com/snakeztc/NeuralDialog-CVAE`.

[107] Yingbo Zhou, Devansh Arpit, Ifeoma Nwogu, and Venu Govindaraju. Joint training of deep auto-encoders. *arXiv preprint arXiv:1405.1380*, 2014.

Vita

Rastin Rastgoufard is native to New Orleans. He is receiving his Doctorate of Philosophy from the University of New Orleans in 2018, completed his Master's Degree at the University of New Orleans in 2012, and obtained his Bachelor's Degree from Tulane University in 2008. He spent many years researching with his advisor AbdulRahman Alsamman while simultaneously working in the Power and Energy Research Lab and the start-up Unocity with Parviz Rastgoufard, Ittiphong Leevongwat, and Ebrahim Amiri. He enjoys programming, art, music, and swing dancing. You can find him cutting a rug around town.