Bayesian Deep Generative Models for Semi-Supervised and Active Learning

Jonathan Gordon

Supervisor: Dr José Miguel Hernández-Lobato

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning, Speech and Language Technology

Fitzwilliam's College August 2017

Abstract

Despite the recent successes in machine learning, there remain many open challenges. Arguably one of the most important and interesting open research problems is that of data efficiency. Supervised machine learning models, and especially deep neural networks, are notoriously data hungry, often requiring millions of labeled examples to achieve desired performance. However, labeled data is often expensive or difficult to obtain, hindering advances in interesting and important domains.

What avenues might we pursue to increase the data efficiency of machine learning models? One approach is semi-supervised learning. In contrast to labeled data, unlabeled data is often easy and inexpensive to obtain. Semi-supervised learning is concerned with leveraging unlabeled data to improve performance in supervised tasks. Another approach is active learning: in the presence of a labeling mechanism (oracle), how can we choose examples to be labeled in a way that maximizes the gain in performance? In this thesis we are concerned with developing models that enable us to improve data efficiency of powerful models by jointly pursuing both of these approaches.

Deep generative models parameterized by neural networks have emerged recently as powerful and flexible tools for unsupervised learning. They are especially useful for modeling high-dimensional and complex data. We propose a deep generative model with a discriminative component. By including the discriminative component in the model, after training is complete the model is used for classification rather than variational approximations. The model further includes stochastic inputs of arbitrary dimension for increased flexibility and expressiveness. We leverage the stochastic layer to learn representations of the data which naturally accommodate semi-supervised learning. We develop an efficient Gibbs sampling procedure to marginalize the stochastic inputs while inferring labels. We extend the model to include uncertainty in the weights, allowing us to explicitly capture model uncertainty, and demonstrate how this allows us to use the model for active learning as well as semi-supervised learning.

I would like to dedicate this thesis to my loving wife, parents, and sister . . .

Declaration

I, Jonathan Gordon of Fitzwilliam College, being a candidate for the M.Phil in Machine Learning, Speech and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. This dissertation contains 14,975 words including two appendices, bibliography, footnotes, tables and equations and has twenty-three figures.

Jonathan Gordon
August 2017

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. José Miguel Hernández-Lobato. His guidance, support, and technical fluency have been invaluable to the progress of this project, and I have benefited enormously from working with him.

I would also like to thank my course mates for countless helpful discussions and useful (and useless) hours passed in the lab - you helped make the thesis and the M.Phil more productive and enjoyable.

Table of contents

List of figures **xv**

List of tables **xvii**

Nomenclature **xix**

1 Introduction **1**

1.1 Data-Efficient Machine Learning 1
1.2 Deep Generative Models 2
1.3 Thesis Contributions 3

2 Background **5**

2.1 Semi-supervised Learning 5
2.2 Active Learning 6
2.2.1 Acquisition Functions 7
2.3 Generative and Latent Variable Models 8
2.4 Neural Networks and Deep Learning 9
2.4.1 Bayesian Neural Networks 10
2.5 Variational Bayesian Learning 11
2.5.1 Inference Networks and Mont-Carlo Approximation 13
2.5.2 Stochastic Backpropagation and the Reparametrisation Trick 14
2.6 Variational Learning of Deep Models 15
2.6.1 The Variational Autoencoder 15
2.6.2 BNNs with Stochastic Backpropagation 17

3 Related Work **21**

3.1 Deep Generative Models for semi-supervised Learning 21
3.1.1 Improved Semi-Supervised Learning with Auxiliary Deep Generative Models 23

3.2 Active Learning with Bayesian Neural Networks 24
3.2.1 Approximating Acquisition Functions 25
3.3 Relationship to this Work 27

4 Deep Generative Models with Discriminative Components 29
4.1 Variational Training of the Model 30
4.1.1 Labeled Data ELBO 30
4.1.2 Unlabeled Data ELBO 31
4.2 Discrete Outputs 32
4.3 Introducing Model Uncertainty 34
4.4 Prediction with the Model 35

5 Implementation, Experiments, and Results 37
5.1 Key Implementation Details 37
5.1.1 KL Warmup 37
5.1.2 Learning Rate Annealing and Weight Regularization 38
5.1.3 Variance Initialization 39
5.1.4 Batch Normalization 39
5.2 Moons Data 41
5.2.1 Experimental Setup 42
5.2.2 Accuracy and Log-Likelihood 42
5.2.3 Uncertainty Estimates 43
5.3 Batch Normalization for Bayesian Neural Networks 45
5.3.1 Effect of Batch Normalization 47
5.4 Active Learning 48
5.4.1 Examining the Acquisition Functions 48
5.4.2 Performance Comparison 49
5.5 MNIST Experimentation 51
5.5.1 Performance Comparison 51
5.5.2 Generating Samples and Encoding 53
5.5.3 MNIST Discussion 55

6 Conclusion 57
6.1 Discussion 57
6.2 Future Work 58

References 59

Appendix A Deconstruction of the Inference Network 65

Appendix B Gradients and Sampling With Mixture Densities 67

List of figures

1.1 Graphical forms of DGMs 2

2.1 Schematic representation of a MLP 10
2.2 Schematic representation of a VAE 16
2.3 VAE generative model of MNIST 17
2.4 BNN with stochastic inputs. 19

3.1 Semi-supervised VAE 22
3.2 Stacked $M1$ and $M2$ 23
3.3 ADGM and SDGM 23

4.1 Proposed graphical model 29

5.1 Generated moons data 41
5.2 Semi-supervised moons data 41
5.3 Learned Decision Functions 0.1 44
5.4 Learned Decision Functions 0.2 45
5.5 Batch Normalization for BNNs 46
5.6 Training Curves $\sigma^2 = 0.1$ 47
5.7 Training Curves $\sigma^2 = 0.2$ 48
5.8 Trained Model Acquisition Functions 48
5.9 Acquiring examples from a pool 49
5.10 Active learning experiment initialization 50
5.11 Active learning performance 50
5.12 Active learning visualization 52
5.13 Generative sampling SSLPE/SSLAPD MNIST 54
5.14 t-SNE visualization of MNIST latent spaces 55

List of tables

5.1 Moons data (0.1) performance 43

5.2 Moons data (0.2) performance 43

5.3 Accuracy results for MNIST experimentation 53

5.4 Log-likelihood results for MNIST experimentation 53

Nomenclature

Acronyms / Abbreviations

ADGM	Auxiliary Deep Generetaive Model
AEVB	Auto-Encoding Variational Bayes
AI	Artificial Intelligence
AL	Active Learning
BN	Batch Normalization
BNN	Bayesian Neural Network
DGM	Deep Generative Models
DL	Deep Learning
DNN	Deep Neural Network
ELBO	Evidence Lower Bound
EM	Expectation-Maximization
GD	Gradient Descent
KL	Kullback-Leibler
LVM	Latent Variable Models
MC	Monte-Carlo
ML	Machine Learning
MLP	Multi-Layer Perceptron

SBP	Stochastic Back-Propagation
SDGM	Skip Deep Generetaive Model
SGD	Stochastic Gradient Descent
SGVB	Stochastic Gradient Variational Bayes
SSL	Semi-Supervised Learning
SVM	Support Vector Machines
VAE	Variational Auto-Encoder
VB	Variational Bayes
VI	Variational Inference

Chapter 1

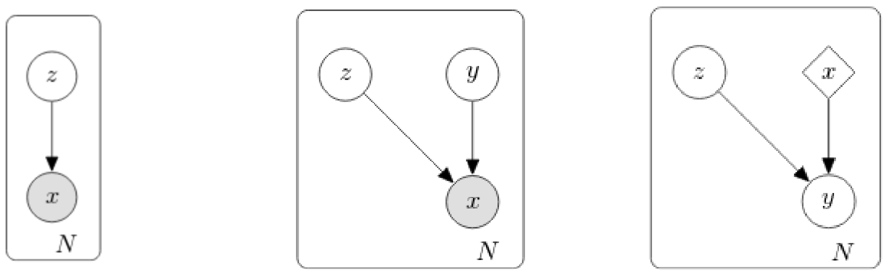
Introduction

1.1 Data-Efficient Machine Learning

Recent trends and approaches in Machine Learning have enabled large models to scale to complex and high dimensional tasks such as computer vision, speech recognition, and reinforcement learning, achieving excellent results. Deep learning has played an integral part in these recent advances. However, despite the numerous successes of machine learning, drawbacks still exist. The majority of learning systems today rely on enormous amounts of (typically labeled) data to reliably perform their designated tasks. This is especially true for deep neural networks, which often require millions of labeled examples to achieve desired performance.

The requirement for massive labeled datasets is unsatisfactory from both modeling and practical perspectives. From a modeling perspective, we desire that our models learn efficiently from data in a way that maximizes the gain from each labeled example. The need for large quantities of labeled data indicates a flaw in our models rather than a theoretical limitation, as evidenced by the ability of humans to learn rich concepts from few examples. From a practical point of view, labeling data can often be an expensive or laborious process, and the requirement for vast amounts of labeled data bottlenecks the use of machine learning in many fields. Data-efficient machine learning is concerned with improving the ability of models to learn from fewer examples or, conversely construct larger, more complex models with the same amounts of data.

The probabilistic (also referred to as the Bayesian) approach to learning provides a rich and rigorous framework for developing models and inference algorithms grounded in probability theory, as well as many promising avenues for significant advances in learning capabilities. By accounting for uncertainty and marginalizing model parameters probabilistic learning elegantly sidesteps the overfitting phenomenon and better generalizes to unseen



(a) Standard VAE (b) Conditional VAE (c) BNN - stochastic inputs

Fig. 1.1 Graphical model depictions for VAE and BNN based models.

data in the presence of smaller training sets. Further, as the output of probabilistic models is typically in the form of a predictive distribution, they naturally accommodate more complex learning tasks such as semi-supervised and active learning. This thesis is concerned with the development of deep probabilistic models that are capable of performing joint semi-supervised and active learning.

1.2 Deep Generative Models

Deep generative models (DGMs) provide a rich and flexible family for modeling complex high dimensional data via the use of latent variables. Recently, advances in procedures for training DGMs such as stochastic backpropagation [52] and the reparametrization trick [31] have made training these models feasible and efficient. DGMs are particularly powerful when neural networks are introduced to parameterize the distributions and inference (recognition) networks, leading to the Variational Autoencoder (VAE; Figure 1.1a) [31, 52].

The ability to efficiently train these types of models has lead to a plethora of interesting advances, increasing the flexibility of posterior approximations and expressiveness of the models [5, 61, 38, 51]. An important extension to standard VAEs is the conditional VAE [30], which incorporates labels into the generative model of the inputs (Figure 1.1b), and thus extends the VAE to classification tasks.

One major drawback of the use of VAEs in supervised tasks is that after training the generative model is discarded and the recognition network is typically used for classification [30, 38]. This is unsatisfactory from a modeling perspective as the recognition network is introduced as a tool for performing approximate inference of model parameters, and yet in practice the model (the object we were originally concerned with) is not used at all. A related challenge is that while in theory Bayesian treatment of the network weights is straightforward, in practice this is challenging and seldom performed.

Closely related to deep generative models are Bayesian neural networks (BNNs) [41, 46]. BNNs extend standard neural networks and express model uncertainty via uncertainty in the learned weights. Beyond introducing a probabilistic interpretation of neural networks, this leads to increased robustness and opens the door to tasks requiring uncertainty such as active learning [65] and Bayesian optimization [57]. Blundell et al. [4] extend ideas from stochastic variational inference [23, 48] to an efficient inference procedure for BNNs. Further, Depeweg et al. [10] show how neural network learning can be extended to general α-divergence minimization [20] and provide empirical evidence of the benefit in introducing stochastic inputs (Figure 1.1c). However the stochastic inputs are constrained to be of low (typically one) dimension.

We develop a deep generative model with a discriminative BNN at its core that naturally accommodates semi-supervised learning. Our intent is that after training is complete, the model is used for classification rather than a variational approximation. We introduce two recognition networks and demonstrate how they can be learned to improve training and prediction, as well as allow efficient posterior inference of arbitrary-dimensional stochastic inputs at prediction time. Finally, we show how training of the discriminative component can be extended to Bayesian learning, and demonstrate how this can be leveraged for active learning.

The rest of this thesis is structured as follows. In Chapter 2 we review the core material relevant to our work. In Chapter 3 we review recent literature in applying deep learning to semi-supervised and active learning tasks. Following this, in Chapter 4 we introduce our proposed model, and algorithms and procedures for efficient training and prediction. In Chapter 5 we detail the experiments carried out and their results. Finally, in Chapter 6 we discuss our findings, conclude, and propose avenues for future research.

1.3 Thesis Contributions

The primary contribution of this thesis is the proposal and development of a deep generative model for joint semi-supervised and active learning. Further, we show how the model can be trained using standard point estimates (SSLPE), as well as approximate Bayesian training using variational inference (SSLAPD). These contributions are covered largely in Chapter 4. We then show proof of concepts and feasibility of the model on a toy dataset in Chapter 5, as well as preliminary results from MNIST experimentation.

A secondary contribution of this thesis is an investigation into the practical issues of training complex DGMs. In practice it is often difficult to train large DGMs, and considerably more so using approximate Bayesian inference procedures. In Chapter 5 we include a discussion of a number of implementation details we found especially important for successful training, as well as the proposal of an adaptation of batch normalization [26] so as to exhibit significant improvements in variational training of Bayesian neural networks.

Finally, in Chapter 6 we discuss important avenues of future research within this framework, namely scaling to larger, more complex datasets such as MNIST. Work in this direction has begun, but only preliminary results are included in this document. Regardless, scaling up of complex Bayesian DGMs to large datasets is an important challenge in the larger field of Bayesian deep learning.

Chapter 2

Background

In this chapter we introduce and review key ideas and literature this thesis draws upon. We begin with a review of semi-supervised and active learning (Sections 2.1, 2.2). Following this we briefly discuss important concepts in latent variable models and deep learning in Sections 2.3, 2.4. Core ideas of variational inference are reviewed in Section 2.5, leading to the Variational Autoencoder (VAE) [31] (Section 2.6.1). Finally, in Section 2.6.2 we review recent advances in training BNNs, leading to Bayesian backpropagation [4] and the use of stochastic inputs [10].

2.1 Semi-supervised Learning

In the semi-supervised learning (SSL) setting we consider the problem of *regression* or *classification* when a (typically small) subset of the training data is labeled. The goal is to make use of the unlabeled data to improve performance in the supervised task.

SSL is appealing and applicable in a wide range of settings. In many cases such as language modeling, image processing and classification, or speech analysis, unlabeled data is abundant, while labeled data is expensive or difficult to obtain.

Arguably the most straightforward method for SSL is self-training, where a model bootstraps the labeled data with unlabeled examples which it has classified with high confidence [53], iterating on this procedure until some stopping condition is met. However, this method tends to reinforce prediction errors and is therefore error prone.

Other approaches to SSL include transductive SVMs [28] and graph-based methods [65]. Transductive SVMs learn a max-margin classifier [55] while encouraging the margin to be far from unlabeled data points [28]. However, this approach has difficulty in dealing with large amounts of unlabeled examples [30]. In contrast, Zhu et al. [65] model the data as a weighted graph where examples are vertices, with similarity measures weighting the edges.

Learning and labeling the unsupervised examples is then formulated in terms of a Gaussian random field, and minimum-energy states can be identified efficiently with message passing or belief propagation [64, 62].

Probabilistic modeling provides an appealing alternative to these by modeling the density and structure of the data. This is similar in principle to using the unlabeled data to learn a useful representation of the data which can be used for the supervised task [30, 49]. However, the question then becomes how to incorporate knowledge of the supervised task into the learned density/representation? Many representations of the data exist, and ideally we would like to guide the model towards a representation that is useful for the supervised task at hand. Often, this approach uses unsupervised learning as a pre-training process before the supervised training is carried out, which does not achieve the goal of incorporating knowledge regarding the task into the learned representation.

Recently, the use of neural networks has been proposed for semi-supervised learning [30, 49, 37]. Rasmus et al. [49] extend the unsupervised *ladder network* [50] to be trained with both a supervised and unsupervised objective function. This allows joint training of a network with labeled and unlabeled data, achieving state of the art performance on a number of benchmarks.

Similarly, Kingma et al. [30] develop a deep generative model that incorporates both a supervised and unsupervised lower bound. Here too, the model can be trained simultaneously on labeled and unlabeled data, incorporating information from both to learn the latent representation. These approaches are promising in that they represent a principled approach to combining the supervised and unsupervised tasks, while making use of the labeled data to guide the model towards useful representations of the data. These (and related) approaches will be discussed in more detail in Chapter 3.

2.2 Active Learning

Active learning (AL) is a setting in which we have a (typically small) labeled training set $D_{\text{train}} = \{x_i, y_i\}_{i=1}^{n_{\text{train}}}$ for a supervised task. In addition, we have a pool of unlabeled examples $D_{\text{pool}} = \{x_j\}_{j=1}^{n_{\text{pool}}}$ and an *oracle* which can be queried to label examples from D_{pool}. Our goal is to enable the model to query the oracle in an efficient manner that, ideally, is optimal in some regard [6, 56].

AL is appealing in that data labeling can often be an expensive and laborious process. For data-hungry models such as deep neural networks, a vast amount of labeled data is often required to achieve desirable performance. AL can potentially increase data-efficiency by enabling a user to initially train a model with a relatively small dataset, and then leverage the trained model to select training examples to be labeled from a pool. If properly designed, intelligent selection of labeled examples should significantly decrease the number of labeled examples necessary to achieve the desired performance [6, 15].

To query the oracle, an *acquisition function* α is typically introduced, such that:

$$x_* = \underset{x \in D_{\text{pool}}}{\operatorname{argmax}} \; \alpha(x; \theta, D_{\text{train}}), \tag{2.1}$$

where x_* is the new example to be labeled by the oracle, and θ are the model parameters. Thus, the model iteratively queries a new example x_*, observes the associated label y_* (thus moving x_* from D_{pool} to D_{train}), and updates the parameters θ based on the new observation. Crucial to the success of AL is the choice of α which dictates the querying strategy. A number of candidate functions exist in the literature and are reviewed below.

2.2.1 Acquisition Functions

A straightforward approach in the case of regression is to select x_* that maximizes the predictive variance from D_{pool}. This is an intuitive selection, and is based on the goal of decreasing overall model uncertainty. However, this strategy can select points that do not necessarily contribute to model uncertainty in other regions. Further, many tasks of interest in ML are classification, in which case this straightforward approach is not suitable [12]. A number of acquisition functions capturing different notions of uncertainty have been proposed in the literature and are discussed below.

Variation Ratios

Variation ratios is given as [12]:

$$\alpha_{\text{VarRatios}}(x; \theta) = 1 - \max_{y \in \mathscr{Y}} p_\theta(y|x), \tag{2.2}$$

where $\max_{y} p_\theta(y|x)$ represents the mode of the predictive distribution (parameterized by θ). Variation ratios therefore measures the dispersion of the distribution. It measures how confident the model is in its distribution, and by maximizing $\alpha_{\text{VarRatios}}(x; \theta)$ over the pool set we are labeling the example the model is least certain about.

Predictive Entropy

In contrast to variation ratios, predictive entropy takes an information theoretic approach [40]. It can be computed as:

$$\begin{aligned}\alpha_{\text{Entropy}}(x;\theta) &= \mathscr{H}(y|x) \\ &= -\sum_{y\in\mathscr{Y}} p_\theta(y|x)\log p_\theta(y|x).\end{aligned} \tag{2.3}$$

Predictive entropy measures the average information content in the predictive distribution. By maximizing the $\alpha_{\text{Entropy}}(x;\theta)$ we are labeling the example in D_{pool} whose predictive distribution under the model contains the least information.

Parameters-Prediction Mutual Information

Finally, rather than measure the information content of the predictive distribution, another approach is to maximize the decrease in the expected entropy of the parameter posterior distribution [40]. This is equivalent to the mutual information between the posterior distribution over the parameters $p(\theta|D_{\text{train}})$ and the predictive distribution $p_\theta(y|x)$ [25], and can be expressed as:

$$\begin{aligned}\alpha_{\text{MI}}(x;\theta) &= \mathscr{H}(\theta|D_{\text{train}}) - \mathbb{E}_{y\sim p(y|x,D_{\text{train}})}\Big[\mathscr{H}(\theta|x,y,D_{\text{train}})\Big] \\ &= \mathscr{H}(y|x,D_{\text{train}}) - \mathbb{E}_{\theta\sim p(\theta|D_{\text{train}})}\Big[\mathscr{H}(y|x,\theta)\Big]\end{aligned} \tag{2.4}$$

Mutual information as posed above (referred to in Houlsby et al. [25] as BALD) describes a slightly different measure of uncertainty then predictive entropy. Maximizing $\alpha_{\text{MI}}(x;\theta)$ selects examples from D_{pool} for which the model is uncertain on average, but can produce weight settings from $p(\theta|D_{\text{train}})$ that generate unjustifiably certain predictions [12].

Finally, it is important to note that for the general case computing the acquisition functions as described above is intractable. In Section 3.2.1 we discuss how variational distributions in BNNs can be used to approximate these functions.

2.3 Generative and Latent Variable Models

A fundamental distinction in ML is between generative and discriminative models [2, 44]. Broadly speaking, a discriminative model directly learns a model for a conditional distribution of interest, e.g., $p(y|x,\theta)$, where y is a target variable and x are the inputs. In contrast, a generative model learns the joint distribution $p(y,x)$, and is so called as it can generate samples from the data distribution [2].

Another core and powerful concept in probabilistic modeling is the incorporation of latent variables z in models to account for unobserved factors affecting the observed data x in the modeled domain. A latent variable model (LVM) is of the form:

$$\begin{aligned} p(z) &= f_z(z;\theta_z) \\ p_\theta(x|z) &= f_x(x;z,\theta_x) \end{aligned} \tag{2.5}$$

where f_z, f_x are valid distributions, and $\theta = \{\theta_z, \theta_x\}$ parameterizes the generative process. When designing a LVM, we would like z to incorporate some higher level or interesting features of x.

In general, we are interested in inference of the model parameters θ given observed data (this can be point estimation or Bayesian inference), as well posterior inference for the latent variables $p(z|x)$. We would like to do this in a general case where f_x is as flexible as possible.

Inference in these models can be challenging. One classic approach is the Expectation-Maximization (EM) [8] algorithm. However, the EM algorithm requires $p(z|x)$ to be tractable, which is true only for the very simplest of LVMs (e.g., linear-Gaussian models [54]). For complex models, more powerful and general approaches for *approximate inference* are required. One such approach is variational Bayesian (VB) learning [62], described below. In this thesis we are mainly concerned with complex, latent variable, generative models parameterized by deep neural networks and learned with variational Bayesian procedures.

2.4 Neural Networks and Deep Learning

Deep learning (DL) is a sub-field of machine learning concerned with models that automatically learn useful representations of data geared towards specific tasks [16]. DL has lead to significant advances in performance across a range of important AI tasks such as computer vision [33, 60], speech recognition [21, 9], and reinforcement learning and control [42, 43].

In this work we leverage neural networks to parameterize distributions in generative models. We use simple, deep feed-forward networks (also known as multi-layer perceptrons (MLPs)) to model probabilistic distributions, mapping from inputs x to parameters of interest.

A MLP consists of layers $l = 1, ..., L$, where each layer contains n^l neurons (as depicted in Figure 2.1 for a two hidden layer MLP with four inputs). Every neuron represents a single computational unit with some (typically nonlinear) differentiable activation function g. Thus, if a^l is the output of the layer l, every neuron outputs:

$$a^{l+1} = g(w_{l+1}^T a^l + b_l), \tag{2.6}$$

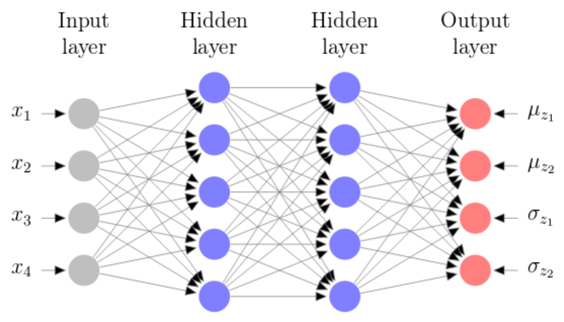


Fig. 2.1 Deep (two hidden layer) feedforward network mapping from inputs x to the parameters of a distribution for z.

where w_l, b_l are the weights and biases associated with layer l, respectively. In this work we use rectified linear units (RELUs) [45], such that the output of layer $l+1$ is:

$$a^{l+1} = \max(0, w_l^T a^l + b_l), \tag{2.7}$$

where $\{w_l, b_l\}_{l=1}^L$ are the trainable parameters of the network. Treating each layer in the network as a simple nonlinear function $f_l(a^{l-1})$ as in Eq. (2.6), the final output of the network can be written as:

$$NN(x) = f_L(...(f_1(x))). \tag{2.8}$$

An objective function (typically likelihood based) is specified and training the network consists of finding an optimal set of weights θ (e.g., in the supervised learning case):

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{N} \log p(y_i|x_i, \theta). \tag{2.9}$$

Despite the fact that this is a non-convex optimization problem [16], training is typically performed with a stochastic gradient optimization procedure [11, 29]. Deep neural networks are high capacity models that can approximate any function given enough neurons [24]. Further, given differentiable activation functions and objectives, they are trainable end-to-end with gradient based optimizers [16]. This makes them suitable for many tasks requiring structured mappings from one set of variables to another.

2.4.1 Bayesian Neural Networks

Bayesian neural networks (BNNs) are an extension of neural networks that explicitly model uncertainty via uncertainty in the model weights [41, 46]. In this setting we first define a prior

$p(W)$ on the weights (and biases), and in inference are interested in the posterior distribution over the weights:

$$p(W|\mathscr{D}) = \frac{p(\mathscr{D}|W)p(W)}{p(\mathscr{D})}, \tag{2.10}$$

where $\mathscr{D}$ is the observed data. As is standard in Bayesian settings, prediction for a new data-point x_* (e.g., in the supervised case $\mathscr{D} = \{(x_i, y_i)\}_{i=1}^N$) is given as:

$$p(y_*|x_*, \mathscr{D}) = \int_W p(y_*|x_*, W)p(W|\mathscr{D})dW. \tag{2.11}$$

Bayesian inference for neural networks is advantageous as it encourages robustness to overfitting, as standard neural networks are prone to do (especially in small data regimes). Further, explicitly modeling uncertainty in the network weights enables one to use BNNs for tasks that require probabilistic outputs such as active learning [15] or Bayesian optimization [57].

Unfortunately both posterior inference and prediction are intractable for general neural networks, and approximations must be used. Laplace's approximation and MCMC sampling methods have been proposed [41, 46], but can under-fit and are not scalable to large data/models, respectively. Below we review recently proposed methods for stochastic variational inference [17, 48, 4] which is both a scalable and powerful method for approximate inference in BNNs.

2.5 Variational Bayesian Learning

Variational Bayesian (VB) learning (or variational inference - VI) is an approach for performing approximate inference in models with intractable posterior distributions [62]. The general idea behind VI is to posit a family of tractable distributions (q), and then find the distribution within that family “closest" to the true posterior [62, 3]. One major benefit of VI is that it substitutes the problem of integration (or sampling as in the case of MCMC) with one of optimization, allowing it to scale easily to larger datasets and models [3].

In the VI methodology, we first derive a variational lower bound (ELBO) for the (log) marginal likelihood of the observed data x by introducing the approximate distribution family $q_\phi(z)$. The approximation is parameterized by the variational parameters ϕ. $\log p_\theta(x)$ can then be expressed as [62]:

$$\log p_\theta(x) = \log \int_{\mathscr{Z}} p_\theta(x,z)\frac{q_\phi(z)}{q_\phi(z)}dz \tag{2.12}$$

$$\geq \int_{\mathscr{Z}} q_\phi(z)\Big(\log p_\theta(x,z) - \log q_\phi(z)\Big) \tag{2.13}$$

$$= \mathbb{E}_{q_\phi(z)}\Big[\log p_\theta(x,z) - \log q_\phi(z)\Big] \triangleq \mathscr{L}(\theta,\phi;x), \tag{2.14}$$

here we have used Jensen's inequality in Eq. (2.13) to derive the lower bound. Another form of deriving the lower-bound is as follows. We begin by noting that:

$$\log p_\theta(x) = \log \frac{p_\theta(x,z)}{p_\theta(z|x)}, \tag{2.15}$$

due to a simple rearranging of Bayes' theorem. Introducing and taking expectation w.r.t. the approximation $q_\phi(z)$ on both sides of Eq. (2.15) yields:

$$\begin{aligned}\log p_\theta(x) &= \int q_\phi(z)\left(\log \frac{p_\theta(x,z)}{q_\phi(z)} + \log \frac{q_\phi(z)}{p_\theta(z|x)}\right)dz \\ &= \underbrace{\mathbb{E}_{q_\phi(z)}\Big[\log p_\theta(x,z) - \log q_\phi(z)\Big]}_{\text{variational lower-bound}} + \mathscr{D}_{kl}\Big(q_\phi(z)\|p_\theta(z|x)\Big),\end{aligned} \tag{2.16}$$

where $\mathscr{D}_{kl}$ is the Kullback-Leibler (KL) divergence [34]. This derivation of the ELBO (which is a lower bound due to the non-negativity of the KL-divergence) demonstrates that maximizing the ELBO w.r.t. the variational parameters ϕ is equivalent to minimizing the KL divergence between the approximation and true posterior distribution (as $\log p_\theta(x)$ is constant for a given model). Thus, the measure of "closeness" used to determine the optimal approximate distribution in q is the KL-divergence [62, 3]. The ELBO is often expressed as:

$$\mathscr{L}(\theta,\phi;x) = \mathbb{E}_{q_\phi(z)}\big[\log p_\theta(x|z)\big] - \mathscr{D}_{kl}\Big(q_\phi(z)\|p_\theta(z)\Big). \tag{2.17}$$

For certain choices of $q_\phi(z), p_\theta(z)$ the second term in the RHS of Eq. (2.17) is tractable. By maximizing the ELBO w.r.t. ϕ (an optimization problem) we identify the approximation that is closest to the true posterior. The complicating term in optimizing the ELBO is the expectation w.r.t. the approximate distribution.

Traditional approaches to VB (such as mean-field approximations [62, 63]) suffer from a few drawbacks. First, often it is necessary that expectations w.r.t. $q_\phi(z|x)$ are tractable, restricting the complexity of the approximations, tpyically to factorized and simple parametric forms. The quality of the trained model is directly related to the quality of the approximation, which we can formalize as $KL\Big(q_\phi(z)\|p_\theta(z|x)\Big)$.

Second, traditional approaches suffer in terms of both statistical and computational efficiency. VB requires introducing separate variational parameters ϕ_i for every data point x_i. This implies that the number of parameters grows (at least) linearly with N, which from a statistical point of view is undesirable. Computationally, for any new data point x_*, inference must be rerun to infer $q(z_*)$ which can be computationally demanding.

2.5.1 Inference Networks and Mont-Carlo Approximation

An elegant solution to the above problems is to introduce an inference network as the approximation $q_\phi(z|x)$. In this setting we train a powerful parameteric model (e.g., a neural network) to map from x to the posterior distribution over z. For a network with enough capacity and an appropriate training procedure, the approximation approaches the true posterior exactly [7, 5].

Further, the introduction of the inference network amortizes the inference procedure: ϕ is now of fixed dimension and shared across all x such that it is constant w.r.t. N. For a neural network parameterization of $q_\phi(z|x)$, latent inference for a new example simply requires a forward pass through the network.

The idea of inference networks was first introduced two decades ago with the proposal of the Helmholtz machine [7]. In that work, the “wake-sleep” algorithm [22] was used to iteratively optimize θ and ϕ. However, this approach was limited in that joint optimization was only possible in an iterative scheme, the two objective functions (for the recognition network and the generative process) did not correspond to maximizing the marginal likelihood of the data [31], and a number mean-field approximations were required for inference. Regardless, the Helmholtz machine is an important work that laid the foundations for the models discussed below.

Ideally, we would like to perform joint training of the model parameters θ and variational parameters ϕ. In the more general case, a naive approach to optimizing the ELBO in Eq. (2.17) is to approximate it via Monte Carlo sampling (for generality assume the KL-divergence is also intractable):

$$\mathscr{L}(\theta,\phi;x) \approx \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(x|z^l) - \Big(\log q_\phi(z^l|x) - \log p_\theta(z^l)\Big) \tag{2.18}$$

where $z^l \sim q_\phi(z|x)$. However, this estimator will be differentiable only w.r.t. θ, whereas we need to optimize ϕ as well. In light of this, we might first take the derivative of Eq. (2.17) w.r.t. ϕ, and derive an unbiased MC estimate of the derivative [47]:

$$\nabla_\phi \mathbb{E}_{q_\phi}\left[\log p_\theta(x|z)\right] \approx \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(x|z)\nabla_\phi \log q_\phi(z^l|x) \tag{2.19}$$

with $z^l \sim q_\phi(z|x)$ as before. However, Paisley et al. [47] observe that this gradient estimator exhibits high variance, and in practice the learning procedure does not converge.

2.5.2 Stochastic Backpropagation and the Reparametrisation Trick

The solution proposed by Kingma and Welling [31], Rezende et al. [52] is a simple reparameterisation of z such that the estimator in Eq. (2.18) becomes differentiable w.r.t. ϕ as well as θ. The key observation is that the problematic operation is sampling z. The solution then is to introduce an auxiliary variable ε:

$$\varepsilon \sim p(\varepsilon); \;\; \tilde{z} = g_\phi(\varepsilon, x) \tag{2.20}$$

such that $\tilde{z} \sim q_\phi(z|x)$. In this case, a general estimator can be rewritten as:

$$\mathbb{E}_{q_\phi(z|x)}\left[f_\theta(z)\right] = \mathbb{E}_{p(\varepsilon)}\left[f_\theta(g_\phi(\varepsilon, x))\right] \approx \frac{1}{L}\sum_{l=1}^{L} f_\theta(g_\phi(\varepsilon^l, x)) \tag{2.21}$$

with $\varepsilon^l \sim p(\varepsilon)$. This estimator is unbiased, and all that is required is that $g_\phi(\varepsilon, x)$ be differentiable, and the ability to sample from $p(\varepsilon)$. Substituting Eq. (2.20) into Eq. (2.18) yields the stochastic gradient variational Bayes estimator (SGVB; [31]):

$$\begin{gathered}\tilde{\mathscr{L}}_{SGVB}\left(\theta, \phi; x^{(i)}\right) \triangleq \frac{1}{L}\sum_{l=1}^{L} \log p_\theta\left(x^{(i)}, z^{(i,l)}\right) - \log q_\phi\left(z^{(i,l)}|x^{(i)}\right) \\ \text{where } z^{(i,l)} = g_\phi\left(\varepsilon^{(l)}, x^{(i)}\right) \text{ and } \varepsilon^{(l)} \sim p(\varepsilon)\end{gathered} \tag{2.22}$$

and leads to the Auto-Encoding Variational Bayes (AEVB, also known as stochastic backpropagation [52]) algorithm as detailed in Algorithm 1.

Algorithm 1 Mini-batch Auto-Encoding Variational Bayes

```
1: $\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Initial values
2: repeat
3:     $\boldsymbol{X}^m \leftarrow$ Random minibatch of size $m$ from $\boldsymbol{X}$
4:     $\boldsymbol{\varepsilon}^l \sim p(\boldsymbol{\varepsilon})$ for $l \in 1, ..., L$
5:     $\boldsymbol{g} \leftarrow \nabla_{(\boldsymbol{\theta},\boldsymbol{\phi})} \tilde{\mathscr{L}}_{SGVB} \left( \boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{X}^{(m)} \right)$
6:     $\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ parameter updates with $\boldsymbol{g}$
7: until Convergence of both $(\boldsymbol{\theta}, \boldsymbol{\phi})$
8: Return: $(\boldsymbol{\theta}, \boldsymbol{\phi})$
```

The SGVB estimator naturally lends itself to minibatch updates, allowing optimization through the use of stochastic optimizers such as Adagrad [11] or Adam [29], yielding an efficient algorithm for learning in a very general class of LVMs.

A requirement of the AEVB algorithm is that there exist a function $g_\phi(\varepsilon, x)$ such that $\tilde{z} \sim q_\phi(z|x)$. In practice, for most parametric distributions there is such a function [31]. A typical case is:

$$q_\phi(z|x) = \mathscr{N}(z; \mu_\phi(x), \sigma_\phi^2(x)), \tag{2.23}$$

where μ_ϕ, σ_ϕ^2 are functions mapping from x to the disrtibution parameters. In this case it is straightforward to show that:

$$p(\varepsilon) = \mathscr{N}(\varepsilon; 0, 1); \quad g_\phi(\varepsilon, x) = \mu_\phi(x) + \sigma_\phi^2(x) \otimes \varepsilon, \tag{2.24}$$

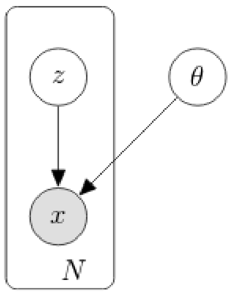
satisfy these conditions, where $\otimes$ is the elementwise product operation.

2.6 Variational Learning of Deep Models

In this section we review two recently proposed models leveraging the ideas discussed in the previous sections. The models parameterize distributions with deep neural networks, and leverage stochastic backpropagation for efficient approximate inference despite the intractability of the posterior distribution.

2.6.1 The Variational Autoencoder

The Variational Autoencoder (VAE) is an unsupervised learning model for observable data x. A latent variable z is introduced, defining the following probabilistic model:



$$p(z) = \mathcal{N}(z; 0, 1)$$
$$p(x|z) = f_\theta(x; z)$$

where f_θ is a valid distribution (typically chosen to be Normal or Bernoulli for continuous and binary data respectively). θ parameterizes DNNs mapping from z to the parameters of the distribution of x. Finally, an inference network is introduced:

$$q_\phi(z|x) = \mathcal{N}\left(z; \mu_\phi(x), \sigma^2_\phi(x)\right), \tag{2.25}$$

with μ_ϕ, σ^2_ϕ are DNNs mapping from x to the posterior distribution parameters of z. A schematic representation of VAEs if shown in Figure 2.2.[1] The use of an inference network allows efficient and flexible inference and importantly, amortizes posterior inference. The model and inference network can then be jointly trained as detailed in Section 2.5.2.

The latent space $\mathscr{Z}$ is typically of significantly lower dimension than $\mathscr{X}$, and can be thought of as a lossy encoding of the observed data.

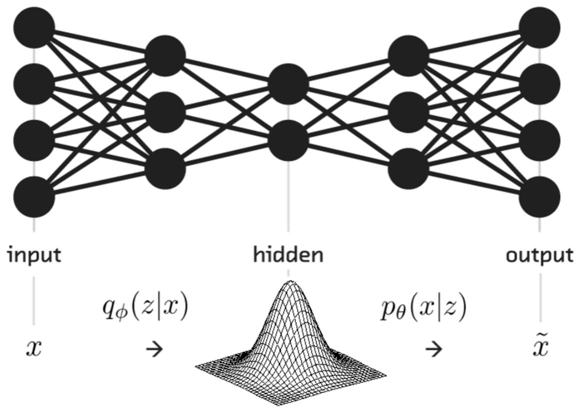
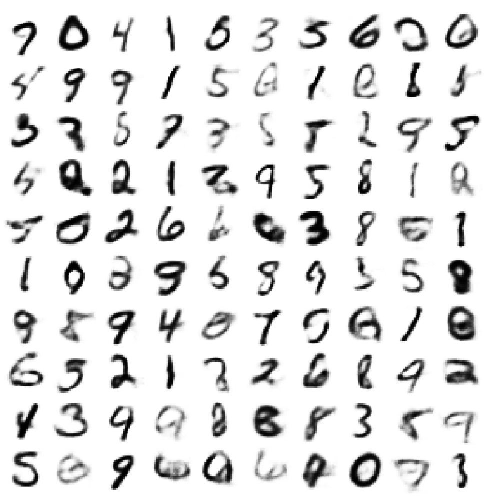


Fig. 2.2 Schematic representation of a VAE. The encoder $q_\phi(z|x)$ stochastically encodes x into the latent space, and the decoder $p_\theta(x|z)$ stochastically maps latent representations back into data space.

Since VAEs are generative models, it is straightforward to use them to generate samples from the domain using simple ancestral sampling [31, 5]. Further, the continuous latent

[1]Figure borrowed from http://blog.fastforwardlabs.com/2016/08/22/under-the-hood-of-the-variational-autoencoder-in.html.

space represents a low-dimensional manifold that can learn important characteristics of the data [31, 5, 52, 61]. These capabilities are demonstrated in Figure 2.3, which were generated using our own implementation of a VAE[2] trained on MNIST [35] (mainly for verifying the correctness of the implementation).


(a)

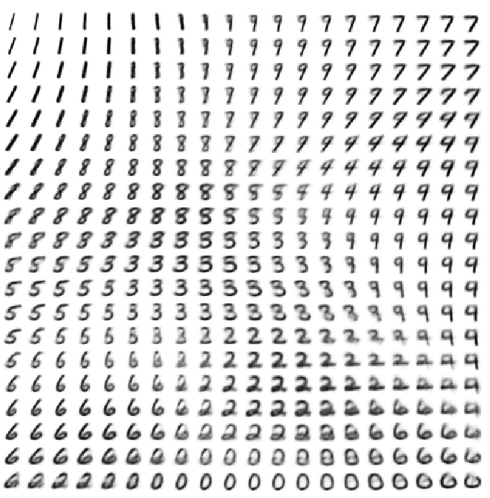
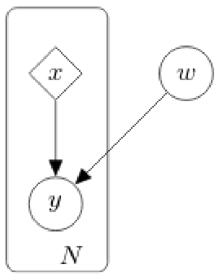

(b)

Fig. 2.3 (a) Samples from our implementation of a VAE trained on MNIST with a 10-dimensional latent space. (b) Visualizing the latent manifold learned using a two-dimensional latent space on the MNIST dataset. Samples were drawn by linearly spacing the 2d unit hypercube, and passing all values through the inverse CDF of the standard normal distribution before decoding.

2.6.2 BNNs with Stochastic Backpropagation

Closely related to VAEs is the idea of training BNNs using variational inference and reparameterization. The BNN model can be detailed as follows:



$$p(w) = \mathcal{N}(w; 0, 1)$$
$$p(y|x,w) = f_\theta(y; x, w),$$

[2]https://github.com/Gordonjo/generativeSSL/blob/master/VAEexp.py

where w are the weights of the neural network parameterizing the distribution of the labels y. Here, x is considered a deterministic observed value and therefore does not appear as part of the generative process (hence BNNs are not proper generative models). Further, note that here we have specified a standard multivariate Gaussian prior for w, but other, more complex priors can be used [4]. For a specific prior $p(w)$, it is straightforward to show that the variational lower-bound can be expressed and approximated as:

$$\begin{aligned} \mathscr{L}(w;x,y) &= \mathbb{E}_{q_\phi(w)}\Big[\log p_\theta(y|x,w) - \log q_\phi(w) + \log p(w)\Big] \\ &\approx \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(y|x,w^l) - \log q_\phi(w^l) + \log p(w^l) \end{aligned} \tag{2.26}$$

where $w^l \sim q_\phi(w)$. Since w is shared across all examples, an inference network is not necessary, and a simple Gaussian distribution can be used:

$$q_\phi(w) = \mathscr{N}\Big(w;\mu_\phi,\mathrm{Diag}(\sigma_\phi^2)\Big) \tag{2.27}$$

The reparameterization trick is then applied to w such that:

$$\begin{aligned} \mathscr{L}(w;x,y) \approx \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(y|x,g_\phi(\varepsilon^l)) - \log q_\phi(g_\phi(\varepsilon^l)) + \log p(g_\phi(\varepsilon^l)) \\ \text{with } \varepsilon^l \sim p(\varepsilon) \end{aligned} \tag{2.28}$$

The lower bound $\mathscr{L}$ is treated as a standard objective function, and can be optimized efficiently using standard gradient descent optimizers. Once trained, prediction for a new data-point x_* is performed by integrating out the weights, and can be approximated using MC integration by taking samples from $q_\phi(w)$:

$$\begin{aligned} p(y_*|x_*,\mathscr{D}) &= \int_w p(y_*|x_*,w)p(w|\mathscr{D})dw \\ &\approx \int_w p(y_*|x_*,w)q_\phi(w)dw \\ &\approx \frac{1}{L}\sum_{l=1}^{L} p(y_*|x_*,w^l), \text{ with } w^l \sim q_\phi(w). \end{aligned} \tag{2.29}$$

In Blundell et al. [4] the authors choose to approximate the KL-divergence via MC sampling, thus allowing them to introduce priors and approximations without being restricted to tractable forms of the KL-divergence.

Finally, in Depeweg et al. [10] BNNs are extended to include stochastic inputs. The graphical model representation of this can be seen in Figure 2.4.

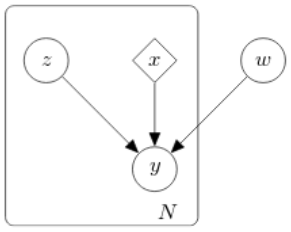


Fig. 2.4 BNN with stochastic inputs.

Depeweg et al. [10] Show that adding the stochastic inputs increases flexibility of the network, and allows it to model heteroskedasticity and multi-modal data. However, the stochastic inputs are limited to be independent of the observed inputs x (conditioned on the outputs), and are thus restricted to simple posteriors and low dimensions. Further, Depeweg et al. [10] demonstrate how one may use black-box α-divergence minimization [20] to train a BNN with stochastic inputs, generalizing the procedure proposed in Blundell et al. [4] to the entire family of α-divergences rather than just VI and the KL-divergence.

Chapter 3

Related Work

In the previous chapter we discussed general ideas in ML that that constitute the background work for this thesis. In this chapter we discuss recent literature closely related to the ideas presented in this document in more detail. In Section 3.1 we discuss advances in using DGMs for semi-supervised learning, and in Section 3.2 we discuss the use of Bayesian training in the deep learning framework for active learning.

3.1 Deep Generative Models for semi-supervised Learning

Arguably the most interesting extension of DGMs is for semi-supervised learning. From a high-level perspective, VAEs are able to learn and model underlying structure in the data. This property lends itself naturally to the idea of leveraging the learned structure from unlabeled data to improve performance on supervised tasks.

This idea was first formalized in Kingma et al. [30], where two approaches were proposed to carry this out. The first ($M1$) simply trains a standard VAE on the unlabeled data, and then encodes the labeled data into the latent space where a classifier is trained. The intuition here being that the latent representation might distill important information in the data relevant to classification. A similar pipeline is followed for new test data.

The second ($M2$) approach proposed extending the VAE model to include labels, as depicted in Figure 3.1. The generative model for this can be described as:

$$p(z) = \mathcal{N}(z; 0, I), \qquad p(y) = \mathrm{Cat}(y|\pi_y), \tag{3.1}$$

$$p(x|z,y) = f_\theta(x; y, z), \tag{3.2}$$

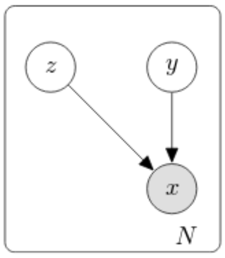


Fig. 3.1 Graphical model representation of $M2$.

where $f_\theta(x; y, z)$ is a valid likelihood function parameterized by neural networks with inputs z, y (Gaussian or Bernoulli for continuous or binary data respectively).

Separate lower bounds can be developed for the labeled and unlabeled cases with the model. For the labeled case the approximate inference network $q_\phi(z|x, y)$ is introduced. The ELBO can then be expressed as:

$$\mathscr{L}^l(x, y; \theta, \phi) = \mathbb{E}_{q_\phi(z|x,y)}\Big[\log p_\theta(x|y, z)\Big] + \log p(y) - \mathscr{D}_{kl}\Big(q_\phi(z|x, y)||p_\theta(z)\Big), \tag{3.3}$$

where we note that $\log p(y)$ is constant w.r.t. both θ, ϕ. For the unlabeled case, an additional inference network $q_\phi(y|x)$ is introduced, and the unlabeled ELBO can be expressed as:

$$\mathscr{L}^u(x; \theta, \phi) = \sum_{y \in \mathscr{Y}} q_\phi(y|x)\mathscr{L}^l(x, y; \theta, \phi) + \mathscr{H}\Big(q_\phi(y|x)\Big), \tag{3.4}$$

where $\mathscr{H}$ computes the entropy of a probability distribution. It is important to note that after training the inference network $q_\phi(y|x)$ is used for classification of unseen examples, and the generative model discarded (in terms of the supervised task at hand). Since no terms including $q_\phi(y|x)$ appear in Eq. (3.3), the authors propose an additional term to the overall cost function such that the inference network is trained with both the labeled and unlabeled data. Thus, the final objective function for training is:

$$\mathscr{L}(x, y; \theta, \phi) = \sum_{(x,y)\sim\tilde{p}_l} \mathscr{L}^l(x, y; \theta, \phi) + \sum_{(x,y)\sim\tilde{p}_u} \mathscr{L}^u(x; \theta, \phi) + \alpha\mathbb{E}_{(x,y)\sim\tilde{p}_l}\Big[\log q_\phi(y|x)\Big], \tag{3.5}$$

where $\tilde{p}_l$, $\tilde{p}_u$ represent the empirical distribution of the labeled and unlabeled data, respectively, and α is a hyper-parameter that needs to be tuned. The final additional term allows $q_\phi(y|x)$ to learn from the labeled data as well as the unlabeled data.

Finally, Kingma et al. [30] propose stacking $M1$ and $M2$ in what can be thought of as a 2-layer stochastic model, as depicted in Figure 3.2.

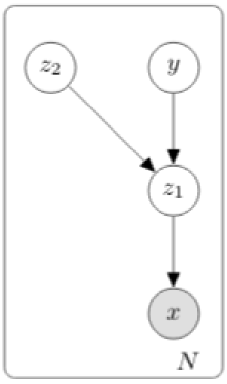
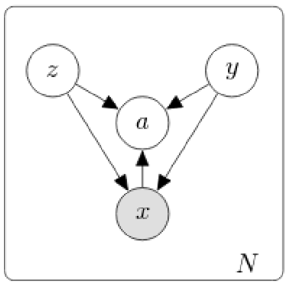


Fig. 3.2 Graphical model representation of stacked $M1$ and $M2$.

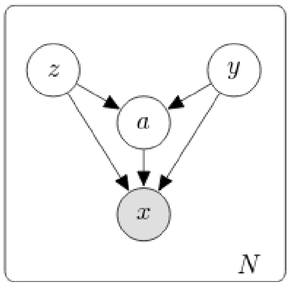
However, convergence for multi-layer deep generative models is difficult, and in Kingma et al. [30] the authors were unable to train the model jointly end to end. Instead, the authors first learned a latent representation using $M1$, then used the latent representation (z_1) to train $M2$. This version of stacking $M1$ and $M2$ without joint training achieved excellent results on a number of benchmark semi-supervised tasks, including a MNIST [35] classification task using only ten labeled instances from every class.

3.1.1 Improved Semi-Supervised Learning with Auxiliary Deep Generative Models

Building on the ideas presented in Kingma et al. [30], an auxiliary variable was proposed to be added to the models in Maaløe et al. [38], as depicted in Figure 3.3a.



(a) ADGM



(b) SDGM

Fig. 3.3 Graphical model representations of ADGM and SDGM.

Adding the auxiliary variable a leaves the generative model of x, y unchanged while significantly improving the representative power of the posterior approximation. An additional inference network is introduced such that:

$$q_\phi(a,y,z|x) = q_\phi(z|a,y,x)q_\phi(y|a,x)q_\phi(a|x). \tag{3.6}$$

It can then be shown that the labeled ELBO can be expressed as:

$$\mathscr{L}^l(\theta,\phi;x,y) = \mathbb{E}_{q_\phi(a,z|x,y)}\Big[\log p_\theta(x,y,a,z) - \log q_\phi(a,z|x,y)\Big], \tag{3.7}$$

and similarly, that the unlabeled ELBO as:

$$\mathscr{L}^u(\theta,\phi;x) = \mathbb{E}_{q_\phi(a,y,z|x)}\Big[\log p_\theta(x,y,a,z) - \log q_\phi(a,y,z|x)\Big]. \tag{3.8}$$

Similarly to Kingma et al. [30], the authors add an additional term for $q_\phi(y|a,x)$ to ensure learning from the labeled data as well. The authors show that adding the auxiliary variable improves convergence rates and lower bounds on numerous benchmarks [37, 38].

Interestingly, by reversing the direction of the dependence between x and a, a model similar to the stacked version of $M1$ and $M2$ is recovered (Figure 3.3b), with what the authors denote skip connections from the second stochastic layer and the labels to the inputs x. In this case the generative model is affected, and the authors call this the *Skip Deep Generative Model* (SDGM). This model is able to be trained end to end using SGD (according to the Maaløe et al. [38] the skip connection between z and x is crucial for training to converge). Unsurprisingly, joint training for the model improves significantly upon the performance presented in Kingma et al. [30].

3.2 Active Learning with Bayesian Neural Networks

A number of interesting approaches have recently been proposed for efficient training of large-scale BNNs. Gal and Ghahramani [13, 14] show how dropout [59] can be reinterpreted as approximating posterior Bayesian inference for neural networks, while Hernández-Lobato and Adams [19], Blundell et al. [4] show how backpropagation and variational inference can be redesigned to derive posterior distributions for the weights. Regardless of the learning procedure, one of the most important applications of Bayesian neural networks is active learning for high dimensional and complex datasets. In the following sections we discuss how BNNs can be used for scalable active learning.

3.2.1 Approximating Acquisition Functions

In Section 2.2 a number of uncertainty measures were introduced that can be used as acquisition functions for active learning in classification settings. These functions are often intractable and cannot therefore be used in a straightforward manner.

However, Gal et al. [15] show how these acquisition functions might be approximated using BNNs with dropout as an approximation for a posterior distribution over weights. In this work, we propose using Bayesian backpropagation [4] to approximate posterior inference in BNNs. Despite this difference, the methods proposed in Gal et al. [15] to approximate acquisition functions are applicable to our methodology as well. Below we discuss these approximations.

Variation Ratios

As detailed in Eq. (2.2), variation ratios is given as:

$$\alpha_{\text{VarRatios}}(x;\theta) = 1 - \max_{y\in\mathscr{Y}} p_\theta(y|x).$$

In the case of BNNs, we can approximate the posterior predictive distribution as [4, 12]:

$$\begin{aligned} p_\theta(y|x) &= \int_W p(y|x,w)p(w|D_{\text{train}})dw \\ &\approx \int_W p(y|x,w)q_\phi(w)dw \\ &\approx \frac{1}{L}\sum_{l=1}^{L} p(y|x,w^l) \text{ with } w^l \sim q_\phi(w) \\ &\triangleq \hat{p}(y|x), \end{aligned} \tag{3.9}$$

where $q_\phi(w)$ is the variational approximation to the true posterior distribution $p(w|D_{\text{train}})$. In practice in the classification case, the final layer of the network is a softmax function approximating $p(y|x,w)$. Substituting the approximation in Eq. (3.9) into Eq. (2.2), we can define the approximation to $\alpha_{\text{VarRatios}}(x;\theta)$ as:

$$\hat{\alpha}_{\text{VarRatios}}(x;\theta) \triangleq 1 - \max_{y\in\mathscr{Y}} \hat{p}(y|x). \tag{3.10}$$

Predictive Entropy

Similarly, we can use the approximation to $p_\theta(y|x)$ to approximate $\alpha_{\text{Entropy}}(x;\theta)$. Recall from Eq. (2.3) that:

$$\alpha_{\text{Entropy}}(x;\theta) = -\sum_{y\in\mathscr{Y}} p_\theta(y|x)\log p_\theta(y|x).$$

Thus, we can simply substitute the estimator in Eq. (3.9) into Eq. (2.3), such that:

$$\hat{\alpha}_{\text{Entropy}}(x;\theta) \triangleq -\sum_{y\in\mathscr{Y}} \hat{p}_\theta(y|x)\log \hat{p}_\theta(y|x). \tag{3.11}$$

Mutual Information

Finally, approximating $\alpha_{\text{MI}}(x;\theta)$ is a little more involved. Recall from Eq. (2.4) that:

$$\alpha_{\text{MI}}(x;\theta) = \mathscr{H}(y|x) - \mathbb{E}_{p(\theta|D_{\text{train}})}\Big[\mathscr{H}(y|x,\theta)\Big].$$

Eq. (2.4) consists of two terms. The first term is equivalent to the predictive entropy, and can be approximated as in Eq. (3.11). Further, as shown in Gal et al. [15] we can estimate the second term with the approximate posterior as:

$$\begin{aligned}\mathbb{E}_{p(\theta|D_{\text{train}})}\Big[\mathscr{H}(y|x,\theta)\Big] &\approx \mathbb{E}_{q_\phi(w)}\Big[\mathscr{H}(y|x,\theta)\Big]\\ &\approx -\frac{1}{L}\sum_{l=1}^{L}\sum_{y\in\mathscr{Y}}(y|x,w^l)\log p(y|x,w^l),\end{aligned} \tag{3.12}$$

where $w^l \sim q_\phi(w)$. Combining Eq. (3.12) and Eq. (3.11) we have that [15]:

$$\begin{aligned}\hat{\alpha}_{\text{MI}}(x;\theta) \triangleq &-\sum_{y\in\mathscr{Y}} \hat{p}_\theta(y|x)\log \hat{p}_\theta(y|x)\\ &+\frac{1}{L}\sum_{l=1}^{L}\sum_{y\in\mathscr{Y}}(y|x,w^l)\log p(y|x,w^l).\end{aligned} \tag{3.13}$$

Gal [12], Gal et al. [15] show that leveraging uncertainty in BNNs can be leveraged to great efficiency for active learning of high-dimensional and complex data such as images. While predictive entropy and variation ratios can be applied in deterministic models, Gal et al. [15] demonstrate the significant advantage of Bayesian models in AL, which can be easily understood as calibrated uncertainty estimates are crucial to the acquisition functions. Further, BALD only makes sense in a Bayesian setting where $p(\theta|D_{\text{train}})$ is a proper distribution and not a delta function.

3.3 Relationship to this Work

Our model is most similar to the work detailed by Kingma et al. [30]. However, since our discriminative component is a Bayesian neural network with random inputs, we use a slightly different inference network architecture.

Similarly, Bayesian deep learning has recently been shown to be highly effective in active learning regimes [19, 15]. In contrast to these works, the proposed model can perform semi-supervised and active learning simultaneously, which may lead to significant improvements. Another difference is that while Gal et al. [15] use dropout as a proxy for Bayesian inference [14] and Hernández-Lobato and Adams [19] use a technique called probabilistic backpropagation, we propose leveraging variational inference to explicitly model the weight uncertainty [4].

The proposed model builds on ideas from both DGMs and Bayesian deep networks to suggest a principled method for simultaneous semi-supervised and active learning.

Chapter 4

Deep Generative Models with Discriminative Components

We propose extending BNNs with stochastic inputs (e.g., as in Depeweg et al. [10]) by adding a dependency from latent variables z to the inputs x (as in Figure 4.1). This dependency results in a component that is similar to a VAE within the model, and enables us to naturally include an inference network for z. Further, we propose new inference procedures to allow this model to be used for semi-supervised learning, as in [30, 37].

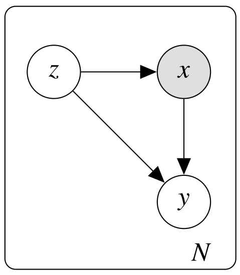


Fig. 4.1 Proposed graphical model

There are a few motivations for this model: (i) it builds on the idea of VAEs, but attempts to do so in a manner that results in an explicit probabilistic model for the labels, (ii) it extends BNNs with stochastic inputs to include inference networks for the latent variables, which should allow generalizing these to high dimensional variables, and (iii) it naturally accommodates semi-supervised and active learning with the generative model. The generative model can be described as:

$$p(z) = \mathcal{N}(z; \mathbf{0}, \boldsymbol{I}), \qquad p_\theta(x|z) = f_\theta(x; z), \tag{4.1}$$

$$p_\theta(y|z,x) = \mathrm{Cat}(y; \pi_y), \tag{4.2}$$

where $f_\theta(x;z)$ is a valid probability distribution (Normal and Bernoulli for continuous and binary data, respectively) and we parameterize the distributions of x, y with deep neural networks (e.g., in the case where $f_\theta(x;z) = \mathcal{N}(x; \mu_x, \nu_x^2)$):

$$\mu_x = NN_x(z, \theta), \qquad \log \nu_x = NN_x(z, \theta), \tag{4.3}$$

$$\pi_y = NN_y(z, x, \theta), \tag{4.4}$$

where $NN_{x,y}$ are neural networks with weights $w_{x,y}$, and $\theta = \{w_x, w_y\}$.

4.1 Variational Training of the Model

We propose a variational approach for training the model. At this point we are interested in point estimates of θ, and variational inference of z. We first develop the variational lower bound using standard techniques [62]. In the semi-supervised setting, there are two lower bounds, for the labeled and unlabeled case.

4.1.1 Labeled Data ELBO

Following recent ideas and advances in variational inference [31, 52], we introduce inference networks $q_\phi(z|x,y)$ to approximate the intractable posterior distribution. We can express a lower bound on the marginal likelihood of the observed data as:

$$\begin{aligned}
\log p_\theta(x,y) &= \log \int p_\theta(x,y,z)dz = \log \int p_\theta(x,y,z)\frac{q_\phi(z|x,y)}{q_\phi(z|x,y)}dz \\
&\geq \int q_\phi(z|x,y)\Big[\log p_\theta(x,y,z) - \log q_\phi(z|x,y)\Big]dz \\
&= \mathbb{E}_{q_\phi(z|x,y)}\Big[\log p_\theta(x,y,z) - \log q_\phi(z|x,y)\Big] \\
&= \mathbb{E}_{q_\phi(z|x,y)}\Big[\log p_\theta(x|z) + \log p_\theta(y|x,z)\Big] - \mathscr{D}_{KL}\Big(q_\phi(z|x,y)||p(z)\Big) \\
&\triangleq \mathscr{L}^l(\theta, \phi; x, y),
\end{aligned} \tag{4.5}$$

where $q_\phi(z|x,y)$ is a recognition network parameterized by ϕ:

$$q_\phi(z|x,y) = \mathcal{N}\Big(z; \mu_z(x,y), \mathrm{Diag}(\sigma_z^2(x,y))\Big). \tag{4.6}$$

where here too μ_z, σ_z^2 are parameterized by neural networks. The lower-bound contains a term for the likelihood associated with the pair of variables x and y, and a regularization term for the inference network. We can approximate expectations w.r.t. $q_\phi(z|x,y)$ by sampling from it (with the reparameterization trick [31]), and using the samples to compute likelihood terms:

$$\mathscr{L}^l(\theta,\phi;x,y) \approx \frac{1}{L}\sum_{l=1}^{L}\Big[\log p_\theta(x,y|z^l) - \log q_\phi(z^l|x) + \log p(z^l)\Big], \tag{4.7}$$

with $z^l \sim q_\phi(z|x,y)$. The KL divergence in Eq. (4.5) is analytically tractable due to the assumed form of the posterior approximation, but in the general case can be approximated via MC sampling.

4.1.2 Unlabeled Data ELBO

We follow similar methods to derive the lower bound for the unlabeled case. In this setting, we have:

$$\begin{aligned}\log p_\theta(x) &= \log \iint p_\theta(x,y,z)dydz = \log \iint p_\theta(x,y,z)\frac{q_\phi(z,y|x)}{q_\phi(z,y|x)}dydz \\ &\geq \iint q_\phi(z,y|x)\Big[\log p_\theta(x,y,z) - \log q_\phi(z,y|x)\Big]dydz,\end{aligned} \tag{4.8}$$

where we have introduced the recognition network $q_\phi(z,y|x)$. It is important to note that by decomposing $q_\phi(z,y|x) = q_\phi(z|x)q_\phi(y|x,z)$ terms for y cancel and the model reverts back to a standard VAE (see Appendix A).

We therefore suggest an alternative. Rather than decomposing the recognition network as $q_\phi(z,y|x) = q_\phi(z|x)q_\phi(y|x,z)$, we decompose it as $q_\phi(z,y|x) = q_\phi(y|x)q_\phi(z|x,y)$. Thus, instead of having two recognition networks $q_\phi(y|x,z)$ and $q_\phi(z|x)$, we will have the two recognition networks $q_\phi(z|x,y)$ and $q_\phi(y|x)$. The introduction of the network $q_\phi(y|x)$ is a little odd given the generative model $p_\theta(y|x,z)$. However, using this decomposition leads to the following lower bound for the unlabeled data:

$$
\begin{aligned}
\log p_\theta(x) &= \log \iint p_\theta(x,y,z)dydz = \log \iint p_\theta(x,y,z)\frac{q_\phi(z,y|x)}{q_\phi(z,y|x)}dydz \\
&\geq \iint q_\phi(z,y|x)\Big[\log p_\theta(x,y,z) - \log q_\phi(z,y|x)\Big]dydz \\
&= \iint q_\phi(z,y|x)\Big[\log p_\theta(y|x,z) + \log p_\theta(x|z) + \log p(z) - \log q_\phi(z,y|x)\Big]dydz \\
&= \mathbb{E}_{q_\phi(z,y|x)}\Big[\log p_\theta(x,y|z)\Big] - \mathbb{E}_{q_\phi(y|x)}\Big[\mathscr{D}_{kl}(q_\phi(z|x,y)||p(z))\Big] + \mathscr{H}\Big(q_\phi(y|x)\Big) \\
&\triangleq \mathscr{L}^u(\theta,\phi;x),
\end{aligned}
\tag{4.9}
$$

where $\mathscr{H}(\cdot)$ computes the entropy of a probability distribution. This form of the lower bound has data fit terms for both x and y in the generative model, as well as regularization terms for both recognition networks $q_\phi(z|x,y)$ and $q_\phi(y|x)$ (these too are parameterized by neural networks). Expectations w.r.t. $q_\phi(z,y|x)$ can be approximated using Monte-Carlo approximation by first sampling $y^l \sim q_\phi(y|x)$, then sampling $z^l \sim q_\phi(z|x,y^l)$.

The recognition network $q_\phi(z|x,y)$ is shared by both the unlabeled and labeled objectives. The recognition network $q_\phi(y|x)$ is unique to the unlabeled data. Following the work in [30, 38], we add a weighted term to the final objective function to ensure that $q_\phi(y|x)$ is trained on all data[1] such that:

$$
\mathscr{L}(\theta,\phi;x,y) = \sum_{(x,y)\sim\tilde{p}_l} \mathscr{L}^l(x,y;\theta,\phi) + \sum_{x\sim\tilde{p}_u} \mathscr{L}^u(x;\theta,\phi) + \alpha\mathbb{E}_{(x,y)\sim\tilde{p}_l}\left[\log q_\phi(y|x)\right], \tag{4.10}
$$

where α is a small positive constant which is initialized in a similar way as in [30], $\tilde{p}_l$ is the empirical distribution of labeled points and $\tilde{p}_u$ is the empirical distribution of unlabeled points.

4.2 Discrete Outputs

Optimizing Eq. (4.10) is straightforward when y is continuous as we can approximate expectations by taking samples from $q_\phi(z|x,y)$ and $q_\phi(y|x)$ and use stochastic backpropagation and the reparameterization trick [31, 52] to differentiate through these approximations.

[1]This is not strictly necessary in our model, but we find that it increases convergence speed and eases training.

Despite recent efforts [27], reparameterization for discrete variables is as yet not a well-understood process. Optimization of $\mathscr{L}^u$ requires taking expectations w.r.t. $q_\phi(y|x)$ which may be a continuous or discrete distribution. In the discrete case ($y \in \{1,...,K\}$), rather than Monte-Carlo approximation we propose directly computing the expectation by summing over the possible values of y:

$$\mathbb{E}_{q_\phi(y|x)}\left[f(y,z,x)\right] = \sum_{y\in\mathscr{Y}} q_\phi(y|x) f(x,y,z). \tag{4.11}$$

Substituting this for the relevant terms in Eq. (4.9) yields:

$$\mathbb{E}_{q_\phi(y|x)}\Big[\log p_\theta(x,y|z)\Big] = \sum_{y\in\mathscr{Y}} q_\phi(y|x)\mathbb{E}_{q_\phi(z|x,y)}\Big[\log p_\theta(x,y|z)\Big], \tag{4.12}$$

$$\mathbb{E}_{q_\phi(y|x)}\Big[\mathscr{D}_{kl}(q_\phi(z|x,y)||p(z))\Big] = \sum_{y\in\mathscr{Y}} q_\phi(y|x)\mathscr{D}_{kl}\Big(q_\phi(z|x,y)||p(z)\Big), \tag{4.13}$$

We note that Eq. (4.12) and Eq. (4.13) are equivalent to a finite mixture density for $q_\phi(z|x)$:

$$q_\phi(z|x) = \sum_{k=1}^{K} \gamma_k q_\phi^k(z|x), \tag{4.14}$$

where $\gamma_k = q_\phi(y=k|x)$ and $q_\phi^k = q_\phi(z|x,y=k)$. Therefore, in the case where exact enumeration is not tractable (e.g, in the case where there are many categories, such as the ImageNet dataset [33]), it may be possible to scale the model up to many categories by sampling through the mixture density. Let $h = \mathbb{E}_{q_\phi(z|x)}\left[f(x,y,z)\right]$ be the expectation of a function being maximized (e.g., Eq. 4.12), then Graves [18] shows that the gradient for the mixture weights γ_k can be approximated as:

$$\frac{\partial h}{\partial \gamma_k} \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \frac{\partial f(x,y,z^{(n)})}{\partial z_d^{(n)}}\frac{\partial z_d^{(n)}}{\partial \gamma_k}; \;\; z^{(n)} \sim q_\phi(z|x), \tag{4.15}$$

and we can sample from $q_\phi(z|x)$ by assuming a diagonal covariance matrix and using univariate inverse CDFs. This process is given in detail (including explicit gradient terms and sampling procedures) in Appendix B. This process allows us to differentiate through y, and we can differentiate through z by sampling with the reparameterization trick from individual components $q_\phi(z|x,y)$. This is left for future exploration if time permits.

Taking explicit expectations w.r.t. $q_\phi(y|x)$ rather than Monte-Carlo approximations allows us to extend the training procedure to cases where y is discrete. In the continuous case, these expectations will be approximated by Monte-Carlo sampling as per usual in AEVB algorithms [31].

4.3 Introducing Model Uncertainty

A major advantage of our proposed approach is that it enables us to express model uncertainty in the discriminative component $NN_y(z,x,\theta)$ by computing a posterior distribution on the weights W_y. To allow this we extend the model described in Eq. (4.1, 4.2) to include the weights (here we describe including W_y, but it is straightforward to include W_x as well) by considering the following prior and likelihood functions::

$$p_\theta(W_y) = \mathscr{N}(W_y; 0, \mathbf{I}), \tag{4.16}$$

$$p_\theta(y|x,z,W_y) = \text{Cat}(y; \pi_y), \tag{4.17}$$

where $\pi_y = NN_y(z,x,W_y)$ is parameterized by a neural network with weights W_y. We assume that the model weights are independent of the latent variables z. Assuming a single labeled data point, the posterior distribution for the latent variables is:

$$p_\theta(W_y, z|x,y) = \frac{p_\theta(y|z,x,W_y)p(z)p(W_y)}{p(y|x)}. \tag{4.18}$$

Unfortunately, this posterior is intractable. Following the work in Blundell et al. [4], Depeweg et al. [10], we introduce an additional approximate distribution $q_\phi(W_y) = \mathscr{N}(W_y; \mu_w, \sigma_w^2)$ which we assume is independent of the inference networks $q_\phi(y,z|x)$. Re-deriving the lower bound for the labeled case yields:

$$\begin{aligned}
\log p_\theta(x,y) &= \log \iint_{\mathscr{Z},\sqsupseteq} p_\theta(x,y,z,w)dzdw \\
&\geq \iint_{\mathscr{Z},\mathscr{W}} q_\phi(z,w|x,y)\Big(\log p_\theta(x,y,z,w) - \log q_\phi(z,w|x,y)\Big) \\
&= \mathbb{E}_{q_\phi(z,w|x,y)}\Big[\log p_\theta(x,y|z,w)\Big] - \mathscr{D}_{KL}\Big(q_\phi(z|x,y)||p(z)\Big) - \mathscr{D}_{KL}\Big(q_\phi(w)||p(w)\Big) \\
&\triangleq \mathscr{L}^l(\theta,\phi;x,y)
\end{aligned} \tag{4.19}$$

This can be approximated using the reparameterisation trick as:

$$\mathscr{L}^l(\theta,\phi;x,y) \approx \frac{1}{L}\sum_l \log p_\theta(x,y|z^{(l)},w^{(l)}) - \mathscr{D}_{KL}\Big(q_\phi(z|x,y)\|p(z)\Big) - \mathscr{D}_{KL}\Big(q_\phi(w)\|p(w)\Big) \tag{4.20}$$

where $z^{(l)} \sim q_\phi(z|x,y)$ and $w^{(l)} \sim q_\phi(w)$. Similarly, re-deriving the lower bound for the unlabeled case yields:

$$\begin{aligned}
\log p_\theta(x) &= \log \iint p_\theta(x,y,z,w)dzdydw \\
&\geq \mathbb{E}_{q_\phi(y,z,w|x)}\Big[\log p_\theta(x,y|z,w) - \log q_\phi(y,z,w|x)\Big] \\
&= \mathbb{E}_{q_\phi(y|x)}\Big[\mathscr{L}^l(\theta,\phi;x,y)\Big] + \mathscr{H}\Big[q_\phi(y|x)\Big] \\
&\triangleq \mathscr{L}^u(\theta,\phi;x)
\end{aligned} \tag{4.21}$$

This too can be approximated using the reparameterisation trick as:

$$\mathscr{L}^u(\theta,\phi;x) \approx \sum_{y\in\mathscr{Y}} q_\phi(y|x)\frac{1}{L}\sum_l \mathscr{L}^l(\theta,\phi;x,y) + \mathscr{H}\Big[q_\phi(y|x)\Big] \tag{4.22}$$

where the labeled loss is approximated using L samples as in Eq. (4.20).

We follow the work presented in Blundell et al. [4], and optimize the objective functions in Eq. (4.20, 4.22) by applying reparameterisation to the weights W, a procedure known as Bayes-By-Backprop.

Training with these objective functions us allows to explicitly capture model uncertainty via uncertainty in the weights. This in turn allows us to use the model for active learning as well as semi-supervised learning, and increases robustness to over-fitting in smaller data regimes.

4.4 Prediction with the Model

Given a trained model (point estimates for θ,ϕ), we would like to make predictions for y_* (and perhaps infer z_*) for a new test input x_*:

$$\begin{aligned}
p(y_*|x_*,\mathscr{D},\theta,\phi) &= \int_Z p(y_*|x_*,z)p(z|x_*)dz \\
&\approx \mathbb{E}_{q_\phi(z|x,y)}\left[p(y|x_*,z)\right]
\end{aligned} \tag{4.23}$$

The problematic term in Eq. (4.23) is $q_{\phi(z|x,y)}$. To approximate $p(y_\star|x_\star)$ for a new example $x_\star$, we must integrate $p_\theta(y_\star|x_\star, z, W_y)$ with respect to the posterior distribution on z and W_y. For this, W_y is sampled from $q_\phi(W_y)$, while z is sampled from the recognition network $q_\phi(z|x_\star, y_\star)$. Since this recognition network requires $y_\star$, we use a Gibbs sampling procedure, drawing the first sample of $y_\star$ from the recognition network $q_\phi(y_\star|x_\star)$. In particular,

$$
\begin{aligned}
y_\star^{(0)} &\sim q_\phi(y_\star|x_\star)\,, \\
\left.\begin{aligned}
W_y^{(\tau)} &\sim q_\phi(W_y)\,, \\
z^{(\tau)} &\sim q_\phi(z|x_\star, y_\star^{(\tau-1)})\,, \\
y_\star^{(\tau)} &\sim p_\theta(y_\star|x_\star, z^{(\tau)}, W_y^{(\tau)})\,,
\end{aligned}\right\}&\text{for } \tau = 1, \ldots, T.
\end{aligned}
$$

and the final prediction and encoding are averaged over the samples:

$$
\hat{y} = \frac{1}{T}\sum_{\tau=1}^{T} y^{(\tau)}, \qquad \hat{z} = \frac{1}{T}\sum_{\tau=1}^{T} z^{(\tau)}. \tag{4.24}
$$

Using $q_\phi(y_\star|x_\star)$ to initialize the procedure increases efficiency and negates the need for a burn in period. In our experiments $T = 10$ produced good results.

Chapter 5

Implementation, Experiments, and Results

In this section we discuss experimentation and results with the models. Before that, we review a number of key implementation details we found especially useful for achieving good results with the models. All implementations are in TensorFlow [1] and are compatible with GPU processing. Complete code to work with the models (including experiment scripts) can be found on the accompanying code repository.[1]

5.1 Key Implementation Details

All the theory and mathematics required to train, predict, and sample from the models are described in the chapters above. However, in practice training the models required a number of additional insights and machinery to be stable and successful. In this section we briefly present some of the key implementation details required for the models to work properly. The motivation for this section is to provide potential users of the model with additional insights into tuning the necessary parameters, as well as provide some practical advice for practitioners who wish to train similar models (DGMs/BNNs).

5.1.1 KL Warmup

When training the model, we observed that often the generative component $p(z)p(x|z)$ tended to severely underfit the data, producing very simple latent representations that led to low likelihoods. This in turn led to poor predictive performance of the BNN, as the generative component was not informing the discriminative component.

[1]https://github.com/Gordonjo/generativeSSL.

Further, we observed that during training the optimization procedures seemed to favor the KL divergence term; rather than achieving high conditional likelihoods $\mathbb{E}_{q_\phi(z|x,y)}\Big[p_\theta(x|z)\Big]$, the KL divergence was converging on 0. This implied that the model was 'giving up' on the data in favor of the simpler prior for the latent representation.

Following the work presented in Sønderby et al. [58], the generative model ELBO was adjusted to include a warm-up variable β^τ:

$$\mathscr{L}^l(x,y;\theta,\phi) = \mathbb{E}_{q_{\phi(z|x,y)}}\Big[\log p_\theta(x|z) + \log p_\theta(y|x,z)\Big] - \beta^\tau \mathscr{D}_{KL}\Big(q_\phi(z|x,y)||p(z)\Big) \tag{5.1}$$

where τ is the optimization epoch. The schedule for β is based on the number of warmup epochs such that β^τ is linearly incremented from zero to one in the desired number of epochs.

The warmup for the KL term encourages the model to first move towards representations that generate good reconstructions of the data before regularizing the complexity of the inference network.

Interestingly, we found that warmup was not required for all datasets. However, in certain cases it was necessary for the generative component to learn a representation that informed the discriminative component of the underlying structure and enable it to achieve desirable predictive performance.

5.1.2 Learning Rate Annealing and Weight Regularization

Optimization of the objective function was performed using an Adam optimizer [29] with parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. However, the hyper-parameter that had the strongest effect on the training procedure and needed careful tuning was the learning rate η.

One of the most influential implementation details, which we observed to greatly improve training stability and epochs to convergence, was placing η on an exponential decay schedule, such that:

$$\eta^\tau = \eta_0 * \gamma^{\tau/\omega} \tag{5.2}$$

where η_0 is an initial step-size, τ is the training epoch, $0 < \gamma < 1$ is a decay rate, and ω moderates the exponential decay. This allows the step size to be large at the beginning of training when we wish to move quickly towards an optimum, and the step size to be small towards the end of training when we wish to converge.

While the schedule for η introduces more hyper-parameters, we found these easier to reason about intuitively, and the schedule induced drastic improvements in the training procedure.

Another important factor was adding weight regularization to the deterministic models (non-Bayesian). Here, for every neural network in the model not trained with a Bayesian procedure, we added ℓ_2-norm regularization for the weights. Weight regularization encourages smaller weights (corresponding to simpler models) [16], and empirically improved the stability of training and consistency of the optimum found by the optimization procedure.

5.1.3 Variance Initialization

One problem encountered when introducing model uncertainty was that the BNN component training diverged, and training was unsuccessful. To allow convergence, we initalize the variational approximation $q_\phi(W)$ with arbitrarily small variances (on the order of $\exp(-10)$). Doing so allows training to proceed, and converge.

The success of the intialization can be understood as follows: when the weight variances are very small, $q_\phi(W)$ approaches a delta function around the means. Empirically, we observe that as training progresses, the weight variances grow, as can be expected due to the KL-divergence term in Eq. (4.20, 4.22). While the variances are small, training is equivalent to MAP estimation of the weights. Thus, initializing the variances to be very small is similar to first performing MAP estimation (i.e., "guiding" the model towards the data), and then regularizing $q_\phi(W)$, counter-acting the tendency of VI to ignore the data in favor of the simpler regularization term [2].

Another intuition regarding the importance of initializing the variances to low values can be seen by noting that the variance of the gradient estimator is proportional to the variance of the weights. Thus, if during the beginning of training the variances are large, the variance of the gradient estimator is large and training tends to diverge. By initializing the variances to be very small, we reduce the variance of the gradient estimator and encourage convergence.

5.1.4 Batch Normalization

Batch-normalization (BN; [26]) is a recently proposed technique to improve the training process of deep neural networks. A BN layer in a network renormalizes the d-dimensional input $x^{(k)}$ to the k^{th} hidden layer across a mini-batch as

$$\tilde{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\mathbb{V}(x^{(k)})}}, \tag{5.3}$$

and then applies the activation to $\tilde{x}^{(k)}$ rather than $x^{(k)}$. The basic intuition behind BN is that SGD training is complicated by the fact that changes to the weights of previous layers affect the distribution of the data coming into layer k. By performing the normalization, the inputs to layer k maintain a distribution that is approximately standard normal [26].

Further, to increase the expressiveness of the output of layer k, two additional (learned) parameters are introduced such that the normalization becomes:

$$\tilde{x}^{(k)} = \gamma_k \left(\frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\mathrm{V}(x^{(k)})}} \right) + \beta_k, \tag{5.4}$$

allowing the distribution of the inputs to vary from a standard normal distribution.

It is necessary to distinguish between the training and testing regimes in BN. In the training regime, one uses batch means and variances in the estimates of $\mathbb{E}[x^{(k)}]$ and $V[x^{(k)}]$ [26]. However, for the testing regime this may be inappropriate, as we may be interested in prediction for inputs with different statistics to the training batches (e.g., prediction for a single input, or uniformly distributed inputs across a region of interest). To deal with this, it is typical to maintain population mean and variance estimates $(\hat{\mathbb{E}}[x^{(k)}], \hat{V}[x^{(k)}])$ that are then used at test time. The population estimates are estimated with moving averages throughout the training procedure, such that at each iteration we perform the update:

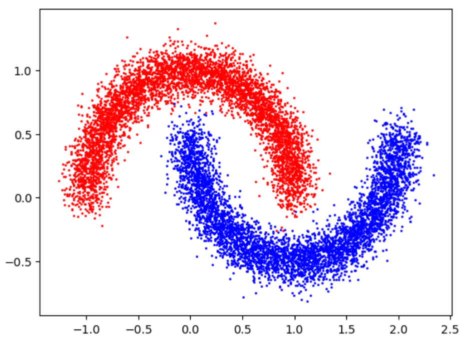
$$\begin{aligned}
\hat{\mathbb{E}}\left[x^{(k)}\right]^{(\tau)} &= \kappa \hat{\mathbb{E}}\left[x^{(k)}\right]^{(\tau-1)} + (1-\kappa)\mathbb{E}\left[x^{(k,\tau)}\right], \\
\hat{V}\left[x^{(k)}\right]^{(\tau)} &= \kappa \hat{V}\left[x^{(k)}\right]^{(\tau-1)} + (1-\kappa)V\left[x^{(k,\tau)}\right],
\end{aligned} \tag{5.5}$$

where $0 \leq \kappa \leq 1$ is a momentum rate for the moving average estimation and $x^{(k,\tau)}$ is the minibatch input to layer k at iteration τ.

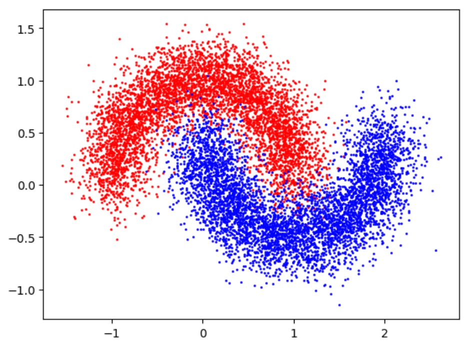
BN has been shown to significantly improve convergence rates of deep models, and Sønderby et al. [58] claim that it is especially useful for training DGMs. Empirically, we found that applying BN to the networks parameterizing the distributions $p_\theta(x|z)$, $p_\theta(y|x,z)$, $q_\phi(y|x)$, and $q_\phi(z|x,y)$ resulted in dramatic improvements in convergence, requiring as much as an order of magnitude less epochs for the model to converge. See Section 5.3 for further regarding the use of BN.

5.2 Moons Data

In this section we detail experiments conducted with the moons dataset. The data contains two classes, each generated from a deterministic function with additive Gaussian noise (Figure 5.1).



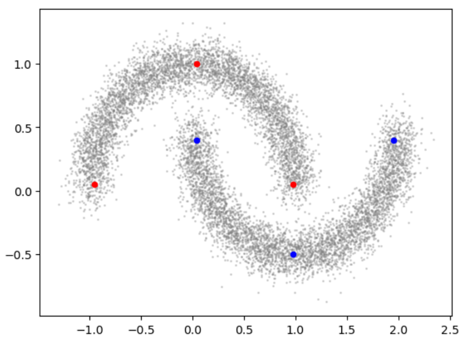
(a) $\sigma_d^2 = 0.1$



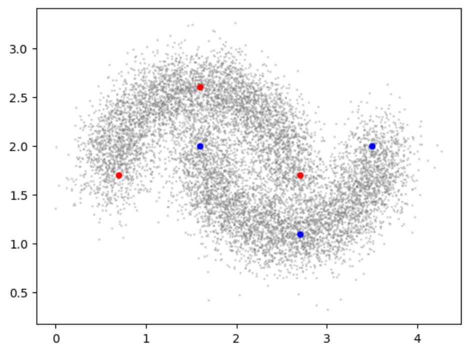
(b) $\sigma_d^2 = 0.2$

Fig. 5.1 Data generated from the moons data function with differing levels of noise.

We generate two sets (training and testing), each containing 1e4 examples. This dataset is not especially challenging for any non-linear classifier. However, we use this data to validate the performance of our model by introducing a semi-supervised scenario similarly to Maaløe et al. [38] (Figure 5.2).



(a) $\sigma_d^2 = 0.1$



(b) $\sigma_d^2 = 0.2$

Fig. 5.2 Semi-supervised scenarios for the moons data. Grey data-points are unlabeled during training.

In the semi-supervised case, the model has access to only three labeled instances from each class, and the remaining training instances are unlabeled. This case is challenging, as the model must make use of the unlabeled data to learn the structure, and use very little labeled information for classification. A standard classifier that cannot make use of the unlabeled data will not be able to achieve good performance on this task.

5.2.1 Experimental Setup

We compare the performance of a number of models in this scenario. We examine the performance of a standard DNN, M2 [30], SDGM [38], and the proposed approach for Semi-Supervised Learning using a Point Estimate for W_y (SSLPE) and the proposed approach for Semi-Supervised Learning using an Approximate Posterior Distribution for W_y (SSLAPD). Performance is compared in terms of test log-likelihood and classification accuracy.

All neural networks have two hidden layers with 128 neurons in each. We use RELU [45] activation functions on all hidden layers, and softmax or linear activations for output layers with categorical or continuous variables respectively. Adam [29] is used for optimization with a tuned learning rate and annealing schedule. Warmup periods are employed for KL-divergence terms, where the temperature is increased linearly, and the number of warmup epochs is tuned for each model.

We use an existing implementation of SDGM.[2] Note that BN is implemented for the SDGM, but not for M2 in the original implementation,[3] which may have significant impacts on performance (see Section 5.3). Further, the original repository does not contain a GPU-compatible implementation of $M2$ (only for the stacked version of $M1+M2$) which significantly slows down performance and hinders experimentation. Therefore, for $M2$ we use our own implementation (which can be found in the repository for this project).[4] We verify our implementation by comparing to the author's implementation on the Moons' data without BN, as well as on MNIST, where our implementation achieves similar performance to that detailed in Kingma et al. [30]).

5.2.2 Accuracy and Log-Likelihood

For each model, we tune the necessary hyper-parameters (learning schedule and KL-warmup), and train the model to convergence. We repeat this process five times for the moons data with additive noise of 0.1 and 0.2, and measure accuracy and log-likelihood after training is complete. The results (mean and standard deviation) are detailed in Tables 5.1, 5.2.

[2]https://github.com/larsmaaloee/auxiliary-deep-generative-models
[3]https://github.com/dpkingma/nips14-ssl
[4]https://github.com/Gordonjo/generativeSSL

Table 5.1 Model performance comparison on moons data with 0.1 additive Gaussian noise.

	DNN	M2	SDGM	SSLPE	SSLAPD
Accuracy	83.6 (3.46)	98.1 (1.21)	99.2 (0.50)	99.2 (0.30)	99.9 (0.01)
$\log p(y\|x)$	-1.207 (0.40)	-0.142 (0.06)	-0.062 (0.03)	-0.073 (0.03)	-0.007 (0.01)

Table 5.2 Model performance comparison on moons data with 0.2 additive Gaussian noise.

	DNN	M2	SDGM	SSLPE	SSLAPD
Accuracy	73.04 (0.88)	94.8 (0.37)	95.1 (0.38)	95.9 (0.35)	96.8 (0.06)
$\log p(y\|x)$	-1.303 (0.13)	-0.258 (0.01)	-0.371 (0.04)	-0.149 (0.01)	-0.084 (0.01)

In all cases SSLAPD outperforms all other models significantly. Unsurprisingly, a standard DNN is unable to achieve good performance as it is not capable of leveraging the unlabeled data.

For the the 0.1 additive noise case, SDGM and SSLPE achieve similar performance in terms of accuracy and log-likelihood. However, when noise level is increased to 0.2, SSLPE significantly outperforms SDGM in log-likelihood, though accuracy is still comparable. This implies that SSLPE produces better uncertainty estimates than SDGM, which is visually reinforced in Figure 5.3.

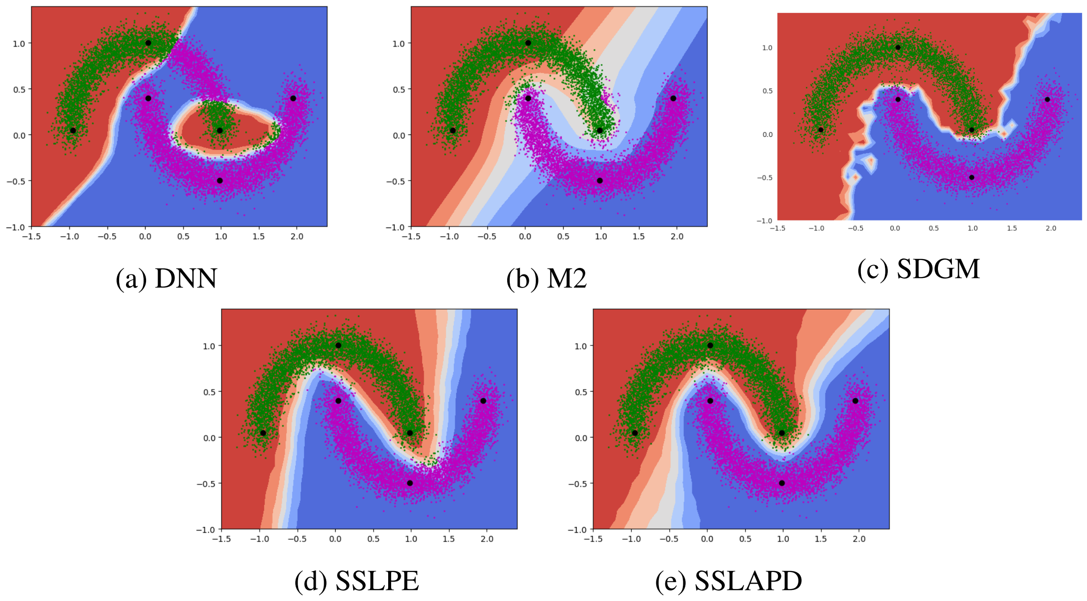
$M2$ is able to achieve comparable (though not quite as consistently high quality) performance as the other semi-supervised models. Further, similarly to the results reported in Maaløe et al. [38] we found that training the model was unstable and often diverged (even after introducing batch normalization).

In the case of 0.2 additive noise, SSLAPD significantly outperforms the other models in both accuracy and log-likelihood. Further, SSLAPD consistently converged to the same solution, while other models exhibited larger variation in performance, and did not always converge to the such a high quality solution (as indicated by the standard deviation of the performance measures).

5.2.3 Uncertainty Estimates

A major motivation for development of the proposed model was the use of the model classifier $p_\theta(y|z,x)$ for classification rather than the variational approximation $q_\phi(y|x)$ ($M2$) or $q_\phi(y|a,x)$ (SDGM). One consequence of this is that the uncertainty estimates of the model may be more accurate. This may explain the performance gap in log-likelihood between

SDGM and SSLPE/SSLAPD on the noisier data. Figures 5.3, 5.4 illustrate this point by examining the learned decision boundaries of the model.



(a) DNN (b) M2 (c) SDGM

(d) SSLPE (e) SSLAPD

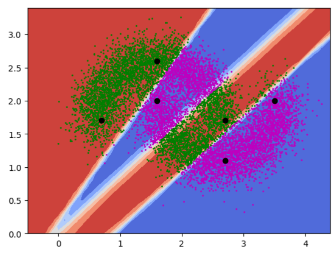
Fig. 5.3 Example decision functions learned in the semi-supervised moons scenario with 0.1 additive Gaussian noise by the different models.

$M2$ learns very wide uncertainty estimates, that while resulting in high accuracy when a 0.5 threshold is applied, may be a contributing factor to the unstable learning process, and indicate that it is not leveraging the unlabeled data as much as other models.

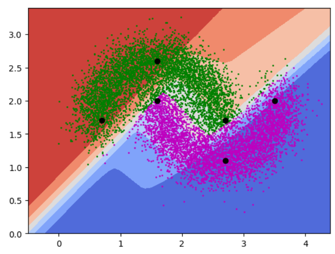
SDGM leverages the unlabeled data and learns an accurate decision boundary, but has almost no uncertainty in its predictions. This leads to poor log-likelihood values (especially in the noisier setting). This may be attributed to overfitting of the classifier $q_\phi(y|a,x)$, which could possibly arise from high dependence on the additional penalty term $\mathbb{E}_{q_\phi(a|x)}\left[\log q_\phi(y|a,x)\right]$ in the objective function. Conversely, this may be a result of using the variational approximation as a classifier, leading to poorly calibrated uncertainty estimates, a known failing of variational inference [40, 2].[5]

In contrast, SSLPE and SSLAPD exhibit well-calibrated uncertainty estimates that lead to good performance in both accuracy and log-likelihood, and allow them to deal well with noisier data. Further, as we would might expect from approximate Bayesian inference, SSLAPD exhibits uncertainty estimates that grow when predicting in regions that are further away from the training data.

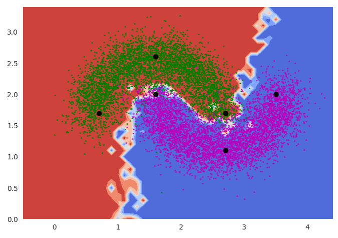
[5]Both explanations are hypotheses, and in general further investigation is required to better understand these behaviors.



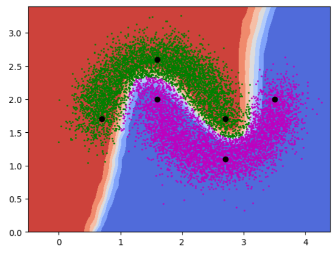
(a) DNN



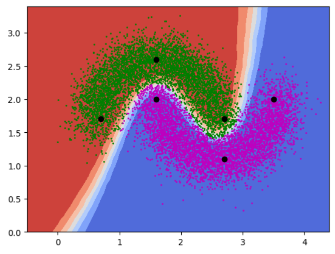
(b) M2



(c) SDGM



(d) SSLPE



(e) SSLAPD

Fig. 5.4 Example decision functions learned in the semi-supervised moons scenario with 0.2 additive Gaussian noise by the different models.

5.3 Batch Normalization for Bayesian Neural Networks

We observed that while applying BN to the networks of SSLAPD introduced significant improvements to the training procedure, the testing regime for BN produced results that were of very low quality. In this section we propose an adaptation to BN to be more appropriate for variational BNNs [4].

A straightforward implementation of BN to BNNs would be to normalize the outputs of a layer generated by a sample from the variational approximation $q_\phi(W)$. However, as the batch statistics depend on the sample of W, this procedure introduces stochasticity into the normalized batch statistics that is not consistent across iterations and thus leads to poor performance in the test regime (Figure 5.5b).

We can formalize the issue as follows. Given the output of layer $k-1$, $z^{(k-1)}$, the input to the activation function of layer k is:

$$x^{(k)} = W_k^T z^{(k-1)} + b_k; \tag{5.6}$$

with W_k, b_k being the weights and bias of the k^{th} layer. For BN, we are interested in $\mathbb{E}[x^{(k)}]$ and $V[x^{(k)}]$. $\mathbb{E}[x^{(k)}]$ can be expressed as:

$$\mathbb{E}[x^{(k)}] = \mathbb{E}\left[W_k^T z^{(k-1)} + b_k\right] \tag{5.7}$$

$$= \mathbb{E}\int_{W,b} \left(W_k^T z^{(k-1)} + b_k\right) p(W|x)dWdb \tag{5.8}$$

$$\approx \mathbb{E}\int_{W,b} \left(W_k^T z^{(k-1)} + b_k\right) q_\phi(W)dWdb \tag{5.9}$$

$$= \mathbb{E}\left[\mu_W^\phi z^{(k-1)} + \mu_b^\phi\right], \tag{5.10}$$

where μ^ϕ denotes the mean of $q_\phi(W)$. Eq. (5.9) implies that applying straightforward BN to sampled weights in a BNN is equivalent to approximate MC estimation of $\mathbb{E}[x^{(k)}]$ under the variational approximation. However, Eq. (5.10) demonstrates that the integration can be carried out analytically in this case, and implies that using the variational means may be preferable for computing the normalization statistics. A completely analogous derivation shows that the variational means should also be used for computing $V[x^{(k)}]$.

Thus, a version of BN appropriate for BNNs performs forward passes through the network with sampled weights as in Blundell et al. [4], but uses the weight means to compute the normalization statistics. We implement this version of BN, and use it for SSLAPD rather than straightforward BN.

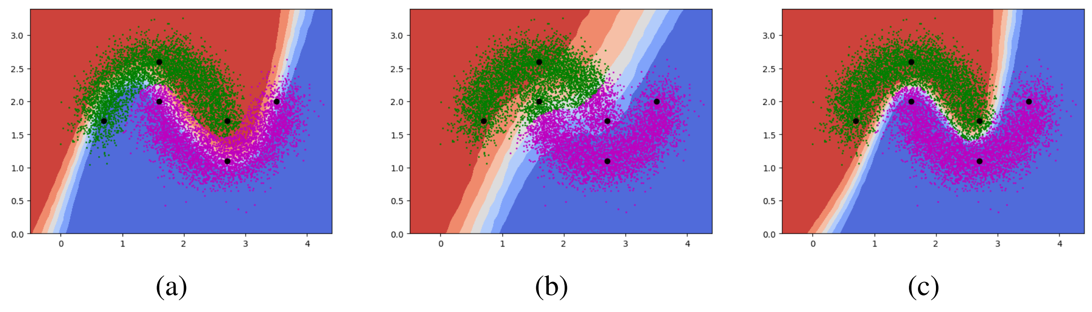


Fig. 5.5 Comparing test results for different BN regimes. (a) Training regime for test data. (b) Testing regime with straightforward BN. (c) Testing regime for adapted BN.

Figure 5.5 details the performance of different BN regimes. We trained SSLAPD for fifty epochs on the semi-supervised moons data, and applied BN in different ways. In all cases, the model converged to optimal performance during training.

In Figure 5.5a we apply the training regime to test data (i.e., use empirical batch statistics). This works well for the test data (test examples are correctly classified), which has similar statistics to the training data, but fails to predict for the entire x region, as this batch has different statistics, resulting in a warped decision boundary.

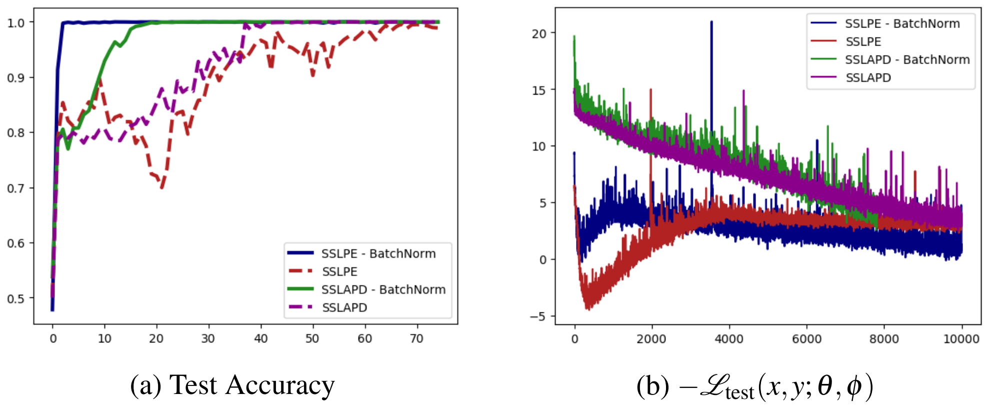
In Figure 5.5b straightforward BN has been applied. The population estimates are not good estimates of the statistics used for normalization during training, resulting in low quality predictions, despite the fact that training has converged to optimal performance.

Finally, Figure 5.5c shows the results of using the adapted BN: predictions are consistent through training and testing, and can be used to predict for inputs with statistics different from those seen throughout training.

5.3.1 Effect of Batch Normalization

In this section we highlight the effect of batch normalization on training the models. The most noticeable effect of batch normalization is in convergence of the training procedure; improvement becomes much smoother as training progresses, and convergence is dramatically faster. BN also eliminated the need for a training schedule (without BN a decay on the learning rate was necessary to achieve convergence).

We compare the performance (test accuracy and ELBO) of SSLPE and SSLAPD with and without BN on the moons data with 0.1 and 0.2 additive Gaussian noise. Learning rates (and schedules) were tuned independently for each model, and training was allowed to proceed to convergence.

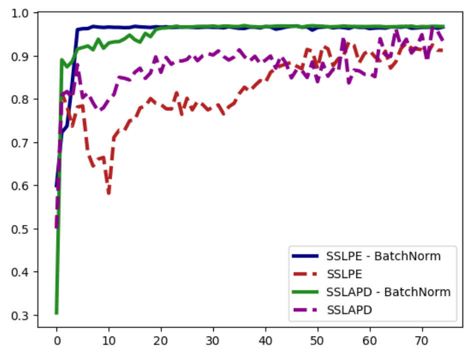


(a) Test Accuracy (b) $-\mathscr{L}_{\text{test}}(x,y;\theta,\phi)$

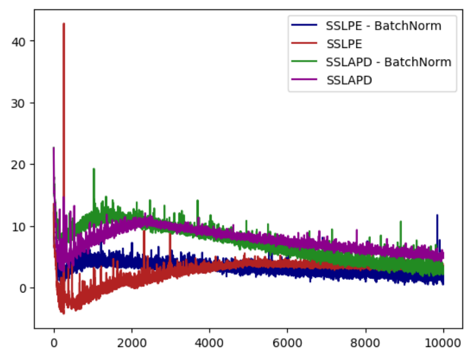
Fig. 5.6 Accuracy and ELBO curves for Moons data ($\sigma^2 = 0.1$) with and without BN.

Figure 5.6 presents (left) accuracy on a held out test set against epochs, and (right) test ELBO against training iterations. The figure demonstrates that in terms of accuracy, BN has a significant effect on convergence speed and stability. SSLPE achieves 100% accuracy after two or three epochs, and SSLAPD after approximately twenty. In contrast, without BN the models take on the order 50-75 epochs to converge, and training is much less stable.

Interestingly, the effect on the overall ELBO is much less pronounced. In both cases the models achieve slightly better ELBO values with BN, but the difference is less significant and

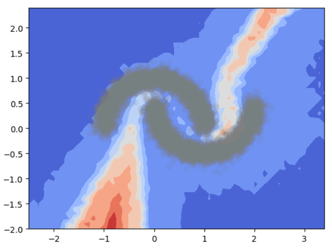


(a) Test Accuracy

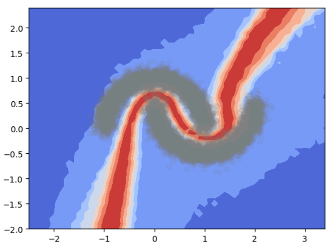


(b) $-\mathcal{L}_{\text{test}}(x, y; \theta, \phi)$

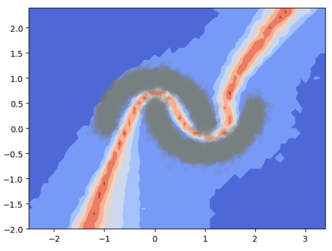
Fig. 5.7 Accuracy and ELBO curves for Moons data ($\sigma^2 = 0.2$) with and without BN.



(a) BALD



(b) Predictive Entropy



(c) Variation Ratios

Fig. 5.8 Acquisition functions computed for the x region using a trained model.

convergence speed is on the same order. Similar trends can be seen for the case of $\sigma^2 = 0.2$ in Figure 5.7. Here the effect of BN on accuracy is more pronounced, as without BN training is very unstable and the models often converge to sub-optimal solutions, or do not converge at all.

5.4 Active Learning

In this section we detail experiments carried out in the active learning setting with the moons dataset. We begin by examining the acquisition functions given a model trained in the semi-supervised setting as shown in Figure 5.2. Following this we compare performance of active learning to random selection with a random data initialization.

5.4.1 Examining the Acquisition Functions

Given a trained model (such as in Figure 5.3), we can visualize the acquisition functions across the entire X space as shown in Figure 5.8.

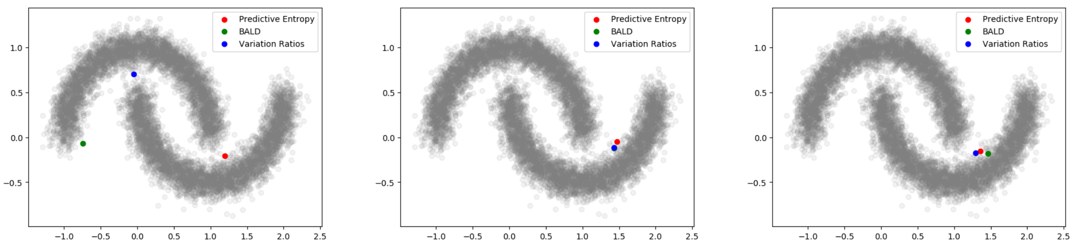


Fig. 5.9 Multiple instances of applying the acquisition functions to a pool. Highlighted examples were selected by the acquisition functions.

The acquisition functions give similar and reasonable estimates for the moons data. All functions favor regions that are close to the decision boundary, where confidence in the predictions is lowest. Predictive entropy and variation ratios give nearly identical results, as is to be expected for the binary case. BALD seems to favor regions that may be on the decision boundary, but are further away from the data, which is very encouraging.

We can also apply the acquisition functions to the unlabeled data and examine which examples each function would choose to label in a real case, as depicted in Figure 5.9. Here we use the unlabeled set as a pool of points the model can potentially query an "oracle" for labels, and apply the three acquisition functions to them. Of course, since the acquisition functions are stochastic (MC) approximations, different instances may result in different selections.

Here too it is clear that the different acquisition functions are behaving similarly, and favoring examples that are near the decision boundary. While these initial examinations are encouraging, it is necessary to experiment with a more realistic scenario for active learning.

5.4.2 Performance Comparison

In this section we examine the performance of the active learning model in a more realistic setting. We randomly select a set of four points to label (Figure 5.10) and train SSLAPD and SSLAPE. Then, we allow the model to query examples until thirty points are labeled, using predictive entropy and BALD (BALD is used only for SSLAPD, as it has no interpretation in the deterministic setting) and compare performance to a random selection scheme.

In previous sections, we saw that by intelligently selecting examples to label, we could successfully train the model with only six labeled instances. Here the task is more difficult as the initial (small) labeled set is randomly selected (though we set the random seed for this to enable fair comparison across models and acquisition functions), resulting in potentially

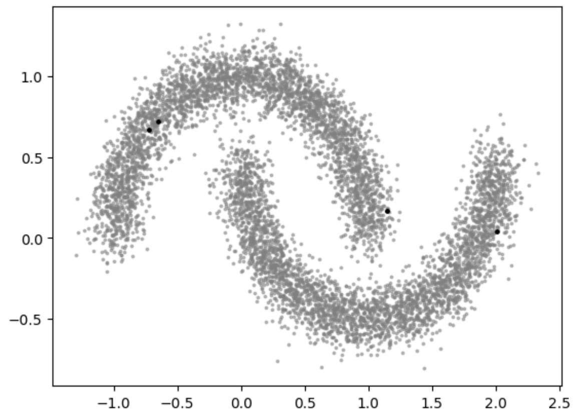


Fig. 5.10 Randomly selected initial labeled set for AL performance experiment.

poor initial models. Ideally, by using active learning the model should converge to optimal performance significantly faster than with random selection.

All models use the same architectures as in previous sections. Due to the stochasticity in the process, we average results for every model-acquisition function combination across three instances. The results of this experiment are detailed in Figure 5.11.

Figure 5.11 demonstrates that the active learning scheme significantly outperforms random selection, and is able to converge to optimal performance with significantly fewer labeled examples than without active querying. In many cases, the model was unable to achieve optimal performance even with 30 randomly selected labeled examples. In contrast, using predictive entropy to label examples enabled both SSLPE and SSLAPD to achieve optimal performance with around ten labeled examples (four of which were randomly selected for initialization).

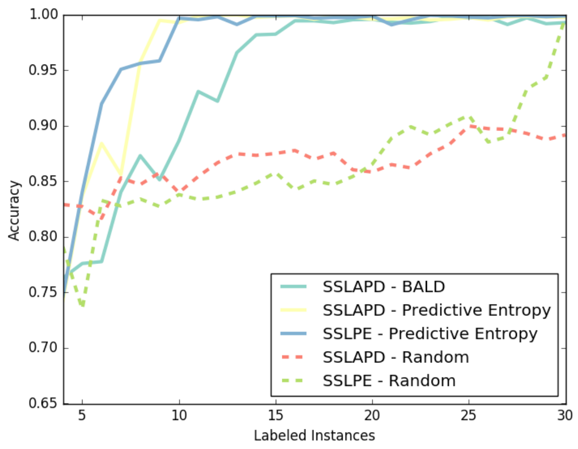


Fig. 5.11 Accuracy of SSLAPD and SSLPE against number of labeled points using different acquisition functions compared to that of random selection.

Interestingly, predictive entropy significantly outperforms BALD, converging with ten labeled examples whereas BALD requires approximately fifteen. This may be an artifact of the dataset and not carry on to more complex cases (such as MNIST), though similar results were observed in Gal et al. [15].

Finally, we can examine the querying process for a single instance. Figure 5.12 visualizes this process for one run of SSLAPD with the predictive entropy acquisition function.

It is interesting to note that the querying process seems to first traverse the lower half-moon, and then continues to the top half-moon. This may be due to the fact that the random initialization included three points from the top half-moon, and only one from the bottom. Regardless, the model seems to effectively traverse the data manifold, requiring only nine labels (in this instance) to achieve optimal performance, starting from a random initialization.

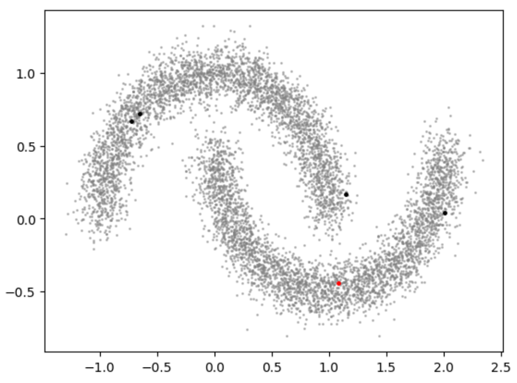
5.5 MNIST Experimentation

The previous sections have all dealt with toy data. While providing a good test bed for initial experimentation and development of the models, the moons' data is of low dimension and complexity, and is therefore not of real interest from a practical standpoint. In this section we report preliminary results from experimentation with the MNIST dataset [35].

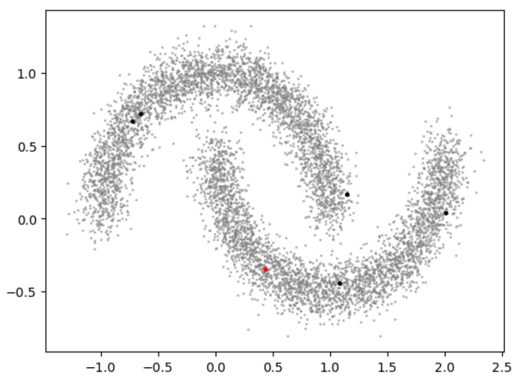
MNIST is a dataset containing 70,000 images of handwritten digits, represented as 28×28 pixel matrices, each pixel assuming a greyscale value between one and 255 (these are typically normalized to between zero and one). MNIST is a standard dataset used for development and benchmarking in the ML community.

5.5.1 Performance Comparison

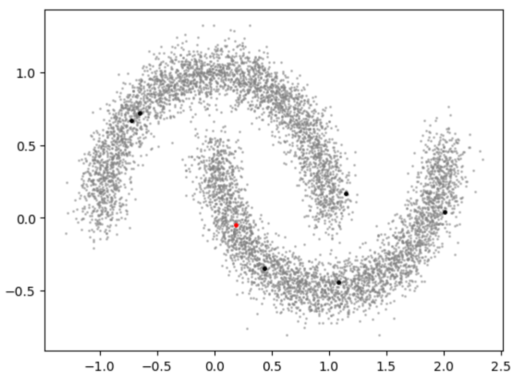
We carry out the following experiment. We use a standard training-testing split of the data set. The training set contains 60,000 examples, and 10,000 are set aside for testing. We further split the training set into a labeled and unlabeled set, with the labeled set containing $\{20, 60, 100, 300\}$ examples *per class*. We generate five versions of the un/labeled split for statistical strength and train each model (DNN, $M2$ [30], SSLPE, SSLAPD) on each of the splits. Performance is measured in accuracy and log-likelihood on the held-out test set. All neural networks contain two hidden layers of 500 neurons each, with RELU activation functions. We use a 100-dimensional latent space, and tune learning rates and warmup periods separately for each model. Tables 5.3, 5.4 report the means and standard deviations of the performance metrics.



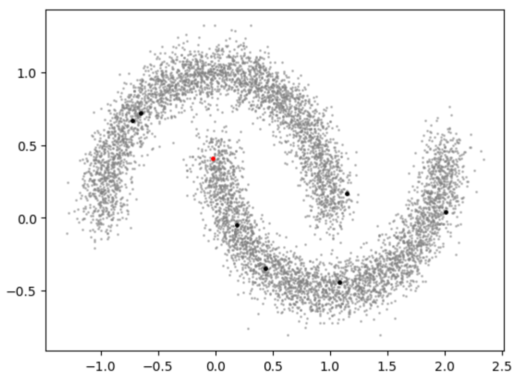
(a) Labeled: 4
Accuracy: 74.27



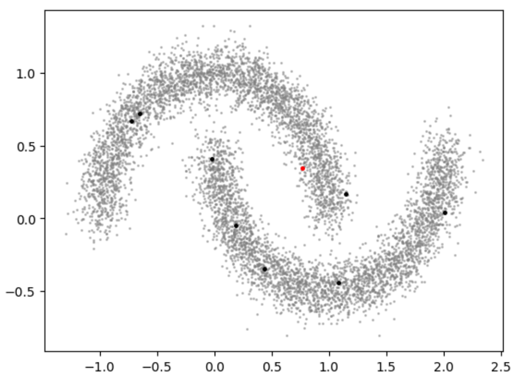
(b) Labeled: 5
Accuracy: 83.67



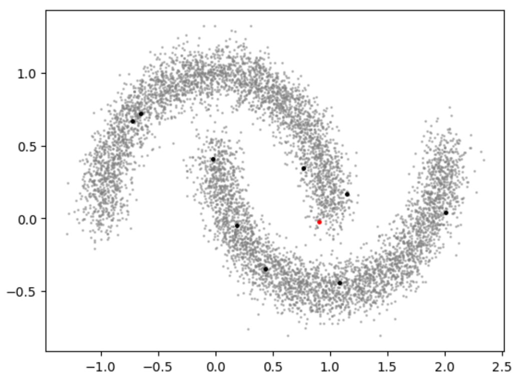
(c) Labeled: 6
Accuracy: 88.40



(d) Labeled: 7
Accuracy: 89.61



(e) Labeled: 8
Accuracy: 95.73



(f) Labeled: 9
Accuracy: 99.47

Fig. 5.12 Visualizing the active querying process. In each image, grey points are unlabeled (pool), black examples are the labeled set, and red examples are selected to be labeled in the next iteration.

Table 5.3 Accuracy results for MNIST experimentation

Labels (per class)	DNN	$M2$	SSLPE	SSLAPD
20	79.6 (0.26)	92.4 (0.71)	92.0 (0.47)	92.3 (0.73)
60	87.2 (0.21)	95.7 (0.25)	95.6 (0.34)	95.0 (0.17)
100	89.9 (0.23)	96.4 (0.32)	96.4 (0.23)	95.9 (0.18)
300	94.1 (0.12)	97.5 (0.18)	97.3 (0.20)	97.0 (0.17)

Table 5.4 Log-likelihood results for MNIST experimentation

Labels (per class)	DNN	$M2$	SSLPE	SSLAPD
20	-.990 (<0.01)	-.415 (0.08)	-.524 (0.05)	-.305 (0.04)
60	-.708 (0.02)	-.219 (<0.01)	-.332 (0.02)	-.190 (<0.01)
100	-.539 (0.02)	-.164 (<0.01)	-.198 (0.03)	-.152 (<0.01)
300	-.357 (<0.01)	-.112 (<0.01)	-.126 (<0.01)	-.110 (<0.01)

Interestingly, we observe slightly better accuracy results for $M2$ than those reported in Kingma et al. [30] with our own implementation, so we report these rather than citing the values from the original paper. As can be expected, the DNN performs very poorly compared to the remaining models as it can not make use of the unlabeled data. All models significantly outperform it in accuracy and log-likelihood.

In terms of accuracy, $M2$ outperforms (or is equal to) our models in all cases, though not to a significant degree. Further, SSLPE achieves slightly better accuracy than SSLAPD except for the case with only twenty labeled examples per class. In contrast, SSLAPD significantly outperforms both $M2$ and SSLPE in terms of log-likelihood. The significance of this decreases as the labeled set grows larger, and for 300 labels per class the performance difference between $M2$ and SSLAPD is negligible.

5.5.2 Generating Samples and Encoding

We examine the generative capabilities of the model. Note that one drawback of our proposed model compared to those proposed by Kingma et al. [30], Maaløe et al. [38] is that our model cannot perform conditional generation[6] as the target variable is a leaf node in the graphical model. Regardless, the model can be used to generate domain samples after training is

[6]By conditional generation we mean generating domain samples conditioned on a specific class.

complete. Figure 5.13 visualizes 100 digits generated by SSLPE and SSLAPD after training with varying labeled set sizes.

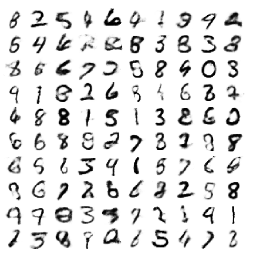
(a) SSLPE
$N_L = 20$

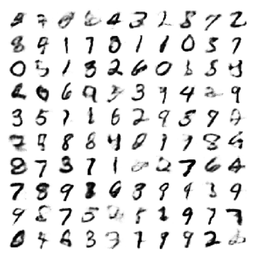
(b) SSLPE
$N_L = 60$

(c) SSLPE
$N_L = 100$

(d) SSLPE
$N_L = 300$

(e) SSLAPD
$N_L = 20$


(f) SSLAPD
$N_L = 60$


(g) SSLAPD
$N_L = 100$

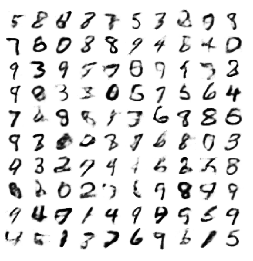
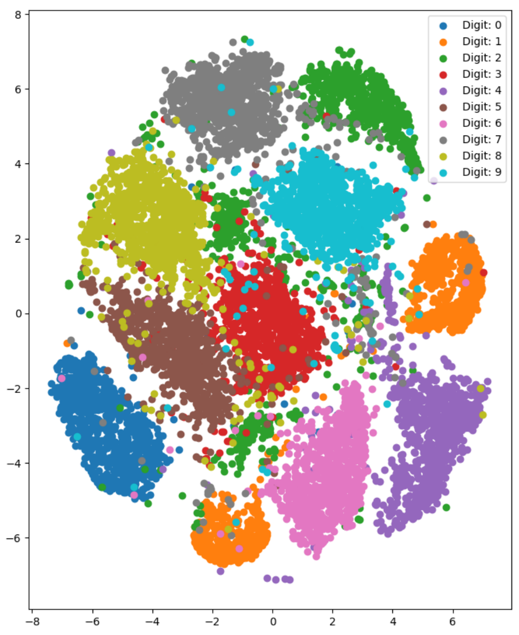

(h) SSLAPD
$N_L = 300$

Fig. 5.13 Sample from trained SSLPE/SSLAPD models on MNIST using different labeled set sizes. Number of labels (N_L) is per-class.

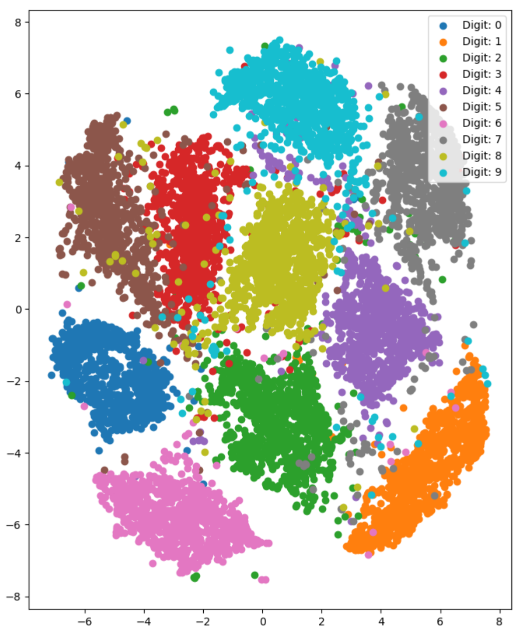
As the generative model of x is independent of the targets y in our model, no significant difference is expected between samples generated with different label set sizes. Indeed, from visual inspection of Figure 5.13 it appears that the size of the labeled set does not have a significant effect on the generated images.

Finally, we can visualize the manifold of the data in the learned latent space. To this end, we encode the test data into the latent representation z after training with SSLPE and SSLAPD. Since the encoder $q_\phi(z|x,y)$ necessitates a y value, we integrate this out with the same Gibbs process as for prediction, and average over sampled values of z rather than y.

We train SSLPE and SSLAPD with 1,000 labeled examples (and the rest of the training set unlabeled), and encode the 10,000 test examples. We use a 50-dimensional latent space for both models. To visualize the latent manifold, we reduce the latent representation of the test data to two-dimensions with t-SNE [39]. Figure 5.14 shows the latent manifold learned by SSLPE and SSLAPD. Both representations do a good job of mapping the different digits to distinct regions of the latent space, and generate well separated clusters, implying that the models have learned a latent representation of the data that is meaningful and useful for classification of the digits.



(a) SSLPE



(b) SSLAPD

Fig. 5.14 Visualizing the latent spaces learned by the models with t-SNE.

5.5.3 MNIST Discussion

The performance of SSLAPD in log-likelihood is encouraging, and implies that the decision function learned may be smoother than those of $M2$ and SSLPE, especially when the labeled set is smaller. This may be particularly useful in the future when we examine the active learning capabilities of the models on MNIST, starting with small labeled sets.

We note that experimentation was also attempted with ten labels per class. In Kingma et al. [30] results are reported for this setting, and in fact we found that $M2$ can achieve significantly better accuracy than reported there. However, as of the writing of this thesis neither SSLPE or SSLAPD are able to achieve the desired level of performance in that setting. Investigation as to the cause of this is on-going, but results are as yet preliminary.

Finally, it is also important to note that here we compare the performance of our models to $M2$. However, these are not state-of-the-art in DGM based semi-supervised learning. There is strong evidence supporting the notion that by adding stochastic layers performance in this setting can be greatly improved [30, 37, 38]. These enhancements can be applied to our models as well, and achieving performance comparable to SDGMs in the semi-supervised setting is a possible avenue of future research.

Chapter 6

Conclusion

6.1 Discussion

In this thesis we have presented a deep generative model with a discriminative component. We proposed variational methods for training the model with point estimates and Bayesian inference. Further, we developed methods to train the model in the semi-supervised setting, with both labeled and unlabeled data.

By incorporating a discriminative component within the model we were able to side-step one of the drawbacks of existing (semi)-supervised VAE methodologies, and use the model rather than the variational approximations for classification. Besides being more satisfying from a modeling perspective, this enabled us to express model uncertainty, which can be leveraged for tasks such as Bayesian optimization and active learning that require calibrated measures of uncertainty.

Finally, we conducted experiments on a toy dataset and MNIST to demonstrate the feasibility of the model in the semi-supervised and active learning setting. Our experiments show encouraging results in that the model can be trained efficiently (both deterministically and with Bayesian inference), and can be used to achieve performance comparable to state of the art models.

Our work further demonstrates the potential of deep generative models. Combining the flexibility of probabilistic modeling and the capacity of deep learning opens the door to the development of many interesting models. The model presented here is one example of the potential of this framework, but more complex and powerful models can be easily designed and implemented with the tools and principles used here.

6.2 Future Work

There are a number of interesting avenues for future work based on the results and methodologies presented in this thesis.

Complex Data

The results presented are encouraging, but are (largely) restricted to low-dimensional and simple data. In contrast, the main interest in machine learning is in its applicability to large scale and complex datasets. Therefore, it is crucial to examine the scalability of the model to more “interesting” datasets. This is especially true for the Bayesian setting where scalability is often an issue. Experimentation with the MNIST dataset has begun (both in the semi-supervised and active learning setting), and is the immediate extension of the work presented in this thesis and required to demonstrate the usefulness of the model. More rigorous experimentation is necessary with MNIST in the semi-supervised setting. It is also interesting to consider additional (more complex) datasets such as CIFAR [32] and Celebrity faces [36].

Larger Models

Initial experimentation with the model on MNIST indicates that performance is comparable to that of the $M2$ model presented in Kingma et al. [30]. However, a number of papers have since demonstrated that constructing larger models (by e.g., concatenating with a VAE [30] or adding additional stochastic layers [37, 38]) can significantly improve performance. An interesting avenue of future research is to experiment with similar larger models and improve performance of this framework to state of the art in both semi-supervised and active learning settings.

Active Learning Experimentation

One of the significant advantages of the proposed model is the possibility of joint semi-supervised and active learning. To this point, active learning experimentation has only been conducted with the toy dataset (with encouraging results). However, this dataset is too simple, and it is important to carry out similar experiments with a more complex dataset such as MNIST in the active learning regime.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

[2] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[3] Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, (just-accepted).

[4] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*.

[5] Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*.

[6] Cohn, D. A., Ghahramani, Z., and Jordan, M. I. (1995). Active learning with statistical models. In *Advances in neural information processing systems*, pages 705–712.

[7] Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The helmholtz machine. *Neural computation*, 7(5):889–904.

[8] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

[9] Deng, L., Hinton, G., and Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8599–8603. IEEE.

[10] Depeweg, S., Hernández-Lobato, J. M., Doshi-Velez, F., and Udluft, S. (2016). Learning and policy search in stochastic dynamical systems with Bayesian neural networks. *arXiv preprint arXiv:1605.07127*.

[11] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[12] Gal, Y. (2016). *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge.

[13] Gal, Y. and Ghahramani, Z. (2015). Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*.

[14] Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

[15] Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep bayesian active learning with image data. *arXiv preprint arXiv:1703.02910*.

[16] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT Press.

[17] Graves, A. (2011). Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356.

[18] Graves, A. (2016). Stochastic backpropagation through mixture density distributions. *arXiv preprint arXiv:1607.05690*.

[19] Hernández-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869.

[20] Hernández-Lobato, J. M., Li, Y., Rowland, M., Hernández-Lobato, D., Bui, T., and Turner, R. E. (2016). Black-box α-divergence minimization. In *Proceedings of The 33rd International Conference on Machine Learning (ICML)*.

[21] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

[22] Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158.

[23] Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. W. (2013). Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347.

[24] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

[25] Houlsby, N., Huszár, F., Ghahramani, Z., and Lengyel, M. (2011). Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*.

[26] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

[27] Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

[28] Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *ICML*, volume 99, pages 200–209.

[29] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[30] Kingma, D. P., Mohamed, S., Rezende, D. J., and Welling, M. (2014). Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589.

[31] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*.

[32] Krizhevsky, A. and Hinton, G. (2010). Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40.

[33] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[34] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

[35] LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

[36] Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3730–3738.

[37] Maaløe, L., Sønderby, C. K., Sønderby, S. K., and Winther, O. (2015). Improving semi-supervised learning with auxiliary deep generative models. In *NIPS Workshop on Advances in Approximate Bayesian Inference*.

[38] Maaløe, L., Sønderby, C. K., Sønderby, S. K., and Winther, O. (2016). Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*.

[39] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

[40] MacKay, D. J. (1992). Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604.

[41] MacKay, D. J. (1995). Probable networks and plausible predictions—a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505.

[42] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[43] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[44] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

[45] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[46] Neal, R. M. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

[47] Paisley, J., Blei, D., and Jordan, M. (2012). Variational bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430*.

[48] Ranganath, R., Gerrish, S., and Blei, D. M. (2014). Black box variational inference. In *AISTATS*, pages 814–822.

[49] Rasmus, A., Berglund, M., Honkala, M., Valpola, H., and Raiko, T. (2015a). Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554.

[50] Rasmus, A., Valpola, H., and Raiko, T. (2015b). Lateral connections in denoising autoencoders support supervised learning. *arXiv preprint arXiv:1504.08215*.

[51] Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*.

[52] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

[53] Rosenberg, C., Hebert, M., and Schneiderman, H. (2005). Semi-supervised self-training of object detection models.

[54] Roweis, S. and Ghahramani, Z. (1999). A unifying review of linear gaussian models. *Neural computation*, 11(2):305–345.

[55] Schölkopf, B. and Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

[56] Settles, B. (2010). Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11.

[57] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M. M. A., Prabhat, M., and Adams, R. P. (2015). Scalable bayesian optimization using deep neural networks. In *ICML*, pages 2171–2180.

[58] Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., and Winther, O. (2016). How to train deep variational autoencoders and probabilistic ladder networks. *arXiv preprint arXiv:1602.02282*.

[59] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

[60] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*.

[61] Tomczak, J. M. and Welling, M. (2016). Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*.

[62] Wainwright, M. J., Jordan, M. I., et al. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305.

[63] Xing, E. P., Jordan, M. I., and Russell, S. (2002). A generalized mean field algorithm for variational inference in exponential families. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 583–591. Morgan Kaufmann Publishers Inc.

[64] Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2003). Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8:236–239.

[65] Zhu, X., Ghahramani, Z., and Lafferty, J. D. (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919.

Appendix A

Deconstruction of the Inference Network

If we decompose $q_\phi(z,y|x) = q_\phi(z|x)q_\phi(y|x,z)$, and allow $q_\phi(y|x,z) = p_\theta(y|x,z)$ we have:

$$
\begin{aligned}
\log p(x) &= \log \iint p_\theta(x,y,z)dydz = \log \iint p_\theta(x,y,z)\frac{q_\phi(z,y|x)}{q_\phi(z,y|x)}dydz \\
&\geq \iint q_\phi(z,y|x)\left[\log p_\theta(x,y,z) - \log q_\phi(z,y|x)\right]dydz \\
&= \iint q_\phi(z,y|x)\left[\log p_\theta(x,z) + \log p_\theta(y|x,z) - \log p_\theta(y|x,z) - \log q_\phi(z|x)\right]dydz \\
&= \int q_\phi(z|x)\left[\log p_\theta(x,z) - \log q_\phi(z|x)\right]\int p_\theta(y|x,z)dydz \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \mathscr{D}_{KL}(q_\phi(z|x)||p(z))
\end{aligned}
\tag{A.1}
$$

By doing this the likelihood term for y has canceled in the expectation. Further, no other terms within the expectation depend on y, such that its expectation cancels, and we are left with the standard VAE term for the unlabeled observations, disconnecting the labeled and unlabeled data.

This reduces to simply training a standard VAE using the unlabeled data, and then perhaps using the resulting recognition network $q_\phi(z|x)$ to improve the training of the recognition network $q_\phi(z|x,y)$.

Appendix B

Gradients and Sampling With Mixture Densities

In this section the details for the mixture density optimization procedure are given. We adopt some of the notation of [18]. We can express our distribution over z as a product over dimensions:

$$p(z|\phi) = \prod_{d=1}^{D} p_d(z|z_{<d}, \phi) \tag{B.1}$$

where $z_{<d} = \{z_1, ..., z_{d-1}\}$. Then, given an inverse CDF F_d^{-1}, we can sample from $p(z|\phi)$ as follows:

$$\hat{z}_1 = F_1^{-1}(u_1);\ \ \hat{z}_d = F_d^{-1}(u_d|\hat{z}_{<d}) \tag{B.2}$$

where $u_i \sim U(0,1)$. It is straightforward then to show that:

$$\begin{aligned}
\frac{\partial \hat{z}_d}{\partial \phi} &= -\frac{1}{p_d(\hat{z}_d|\hat{z}_{<d}, \phi)} \int_{-\infty}^{\hat{z}_d} \frac{\partial p_d(t|\hat{z}_{<d}, \phi)}{\partial \phi} dt \\
&= -\frac{F_d(\hat{z}_d|\hat{z}_{<d})}{p_d(\hat{z}_d|\hat{z}_{<d}, \phi)} \int_{-\infty}^{\infty} p_d(t \leq \hat{z}_d|\hat{z}_{<d}, \phi) \frac{\partial \log p_d(t|\hat{z}_{<d}|\phi)}{\partial \phi} dt \\
&\approx -\frac{1}{N} \frac{F_d(\hat{z}_d|\hat{z}_{<d})}{p_d(\hat{z}_d|\hat{z}_{<d}, \phi)} \sum_{n=1}^{N} \frac{\partial \log p_d(t^{(n)}|\hat{z}_{<d}, \phi)}{\partial \phi};\quad t^{(n)} \sim p_d(t \leq \hat{z}_d|\hat{z}_{<d}, \phi)
\end{aligned} \tag{B.3}$$

where we can sample from $p_d(t \leq \hat{z}_d|\hat{z}_{<d}, \phi)$ with simple rejection sampling from $p_d(t|\hat{z}_{<d}, \phi)$. We can express the mixture density thus:

$$p_d(z_d|z_{<d}) = \sum_{k=1}^{K} p(k|z_{<d}) p_d(z|\phi) \tag{B.4}$$

where we are assuming here that z has a diagonal covariance matrix such that $p_d(z_d|z_{<d}) = p_d(z_d)$. Denoting $p(k|z_{<d}) = \pi_d^k$, it is defined recursively as:

$$\pi_1^k = \gamma_k; \quad \pi_d^k = \frac{\pi_{d-1}^k p_{d-1}^k(z_{d-1}|\phi)}{p_{d-1}(z_{d-1}|\phi)} \tag{B.5}$$

where p_d^k denotes the distribution for dimension d under mixture component k. Taking the derivative of $p_d(z_d|z_{<d})$ w.r.t. γ_j and substituting into Eq. B.3 yields:

$$\frac{\partial \hat{z}_d}{\gamma_j} = -\frac{F_d^k(\hat{z}_d)}{p_d(\hat{z}|\phi)} \sum_{k=1}^{K} \frac{\partial \log \pi_d^k}{\partial \gamma_j} \pi_d^k \tag{B.6}$$

where it can be shown that:

$$\begin{aligned} \frac{\partial \log \pi_d^k}{\partial \gamma_j} &= \frac{\partial \log \pi_{d-1}^k}{\partial \gamma_j} - \sum_l \pi_d^l \frac{\partial \log \pi_{d-1}^l}{\partial \gamma_j} \\ &+ \left(\frac{\partial \log p_{d-1}^k(\hat{z}_{d-1})}{\partial \hat{z}_{d-1}} - \sum_l \pi_d^l \frac{\partial \log p_{d-1}^l(\hat{z}_{d-1})}{\partial \hat{z}_{d-1}} \right) \frac{\partial \hat{z}_{d-1}}{\partial \gamma_j} \end{aligned} \tag{B.7}$$

We can obtain the derivatives in Eq. B.6, B.7 by a joint recursion starting from the initial conditions:

$$\frac{\partial \pi_1^k}{\partial \gamma_j} = \frac{\delta_{jk}}{\gamma_j}; \quad \frac{\partial \hat{z}_1}{\partial \gamma_j} = -\frac{F_1^j(\hat{z}_1)}{p_1(\hat{z}, \phi} \tag{B.8}$$

We can use these results to perform backpropagation through the mixture densities as detailed in Eq. 4.14. Specifically, we can sample from $q_\phi(z|x)$ as detailed in Eq. B.2, and optimize the mixture priors (and thus $q_\phi(y|x)$) by substituting Eq. B.6 into Eq. 4.15.