LEARNING DISTRIBUTIONS WITH PARTICLE MIRROR DESCENT

BY

AILING ZHANG

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Advisor:

Assistant Professor Oluwasanmi Koyejo
Assistant Professor Niao He

Abstract

As an effective and provable primal method to estimate posterior distribution, Particle Mirror Descent is appealing for its simplicity and flexibility. In this thesis we explore the applications of Particle Mirror Descent in both supervised and unsupervised learning.

In the general classification problem with a parametric discriminative function, we seek a posterior distribution over parameters that maximizes classification accuracy. Existing algorithms usually solve the dual problem and the number of variables to optimize depends on the number of examples in the dataset. Therefore such methods suffer from the poor scalability in large datasets. We propose Constrained Particle Mirror Descent to effectively estimate posterior distribution in primal space even with expectation constraint.

By marrying Bayesian probabilistic inference and deep neural networks, deep generative networks have shown remarkable success in various kinds of generative tasks. However, such models usually make an assumption that posterior distribution can be simply characterized as a Gaussian distribution, which is not always true since real data like images and audios yield complex structures in latent space. Motivated by the recent wide application of multi-modal posterior, we introduce a variant of Variational Auto-encoder model that uses a mixture of customized kernels as posterior distribution in latent space. Our deep generative model produces visually plausible images as well as good clustering performance using latent representations.

To my parents for all their love and support.

Acknowledgments

I would like to thank my advisors, Professor Oluwasanmi Koyejo and Niao He for their expert advice as well as extraordinary guidance throughout this project. They always provide timely suggestions whenever I have a question in my research. Both my research skill and presentation skill are improved a lot during our weekly meetings. This work would not have been possible without them.

I would like to also thank my fellow master students, for their valuable feedback and comments of this thesis.

Last but not the least, I would like to thank my parents and friends, for their unfailing support and continuous encouragement along the way.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CDF	Cumulative Distribution Function.
CSA	Corporative Stochastic Approximation.
GAN	Generative Adversarial Network.
HMM	Hidden Markov Model.
KDE	Kernel Density Estimator.
KNN	K Nearest Neighbor.
LDA	Latent Dirichlet Allocation.
MAP	Maximum A Posterior Probability.
MED	Maximum Entropy Discrimination.
MCMC	Markov Chain Monte Carlo.
PMD	Particle Mirror Descent.
RKHS	Reproducing Kernel Hilbert Space.
SFGD	Stochastic Functional Gradient Descent.
SMO	Sequential Minimal Optimization.
SVM	Support Vector Machine.

Chapter 1

Introduction

1.1 Background

Given data examples $X = \{x_1, x_2, \cdots, x_N\}$, the posterior distribution of unknown parameter $p(\theta|X)$ is intractable. Computing exact inference is time-consuming and sometimes unnecessary. Two kinds of methods are developed to solve posterior distribution approximately.

Sampling based approximate inference algorithms such as Markov Chain Monte Carlo (MCMC) [3], and all manners of its variants [9, 18] can generate samples for which the distribution converges to true posterior, yet require very high computational cost. To generate a single sample, it requires evaluating likelihood at each example and a complete scan over the dataset, which may result in undesirable delay in the whole training process.

Variational inference algorithms [13] formulate probabilistic inference as optimization problems. We usually have to restrict the optimization over some manageable classes of distributions to solve it analytically. For example, mean field variational approximation inference [25] assumes the posterior can be fully factorized as a multiplication of distributions from some prior families. Variational inference algorithms are more favorably used since they can approximate the posterior density at a relatively low computational cost, and provide comparable accuracy in real applications. Nonetheless, typically true posterior is not in the variational family. In consequence, we don't have a theoretical guarantee for such algorithms. It is considerably difficult to quantitatively evaluate and control the introduced error.

Recently developed Particle Mirror Descent (PMD) [7] is a primal method that uses kernels to estimate a density and performs stochastic functional gradient descent [8, 15] iteratively to converge

to the true posterior. It's proved that PMD converges at a rate of $O(1/\sqrt{m})$, where m is the number of kernels / particles used in each iteration. The algorithm obtains competitive performance in a diverse collection of large-scale inference tasks, including Latent Dirichlet Allocation (LDA).

We extend Particle Mirror Descent to two problems: one is classification problem with expectation constraint, the other is deep generative networks. In this chapter we will briefly introduce the background and latest development in these areas.

1.1.1 Maximum Entropy Discrimination

Maximum Entropy Discrimination (MED) algorithm [12] was proposed to solve a minimum relative entropy (MRE) problem stated as follows.

Given a set of training examples $X = \{(x_1, y_1), \cdots, (x_N, y_N)\}$, a parametric discriminant function $\mathcal{L}(X|\theta)$, and $\xi = [\xi_1, \cdots, \xi_N]$ as margin variables, assuming a prior distribution $p_0(\theta)$, the discriminative minimum relative entropy (MRE) distribution $p(\theta)$ can be found by minimizing

$$\begin{aligned} \min_{p \in \mathcal{P}} \quad & KL(p||p_0) = \int p(\theta) \log \frac{p(\theta)}{p_0(\theta)} d\theta \\ \text{s.t.} \quad & \int p(\theta)[y_t \mathcal{L}(x_t, \theta) - \xi_t] d\theta \geq 0, \forall t. \end{aligned}$$

where $\mathcal{P}$ is the admissible distribution set. The decision rule for any new example x is

$$\hat{y} = \int p(\theta)[\mathcal{L}(x|\theta)] d\theta.$$

To generalize, MRE is an optimization problem with expectation constraint. We use KL divergence from $p(\theta)$ to a prior distribution $p_0(\theta)$ as objective function, under the assumption that we usually have some prior knowledge about the desired shape of target distribution. In the context of this thesis, we choose normal distribution as prior for the benefits of having closed-form gradients and integrations.

MED algorithm solves the dual problem of the minimization problem above and obtains a closed

form [12] as follows:

$$p(\theta) = \frac{1}{Z(\lambda)} p_0(\theta) e^{\sum_t \lambda_t [y_t \mathcal{L}(x_t|\theta) - \xi_t]},$$

where $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_N\}$ defines a set of non-negative Lagrange multipliers and $Z(\lambda)$ is the normalization constant. λ in the solution are set by finding the unique maximum of the jointly concave objective function

$$\begin{aligned} J(\lambda) &= -\log Z(\lambda), \\ Z(\lambda) &= \int p_0(\theta) e^{\sum_t \lambda_t [y_t \mathcal{L}(x_t;\theta) - \xi_t]} d\theta. \end{aligned} \tag{1.1}$$

1.1.2 Generative Models

Generative probabilistic models are random sources that can generate samples according to certain probability distribution $P(X)$ over a set of data points X. It has a lot of applications in applied machine learning, especially in image and audio generation. Using a generative approach, we are able to simulate the process of how the data points are generated in order as if they were generated naturally. The crux of generative models are to project the high dimensional space $\mathcal{X}$ to a low dimensional latent space first, and then produce new data points from the latent distribution. This is natural to all generative processes. Suppose you are required to write down a digit on the paper, the first decision your brain makes is to decide which character to write. And some other minor questions may be asked before you start, for example the style of calligraphy you want to use, the position of the digit on the paper and so on.

Generative models use likelihood as objective. It's straightforward that a generative model should have high probability to produce data points in the dataset while low probability to produce noises. To achieve this goal, generative models sometimes make strong assumptions about the data structure and approximations to deal with the intractability of posterior. Latent Dirichlet Allocation (LDA) [5], Hidden Markov Model (HMM) [21] are two popular models using aforementioned strategy. However, in some generative tasks, we are not only interested in assigning high probabilities to examples already known, but somehow employing the captured style features to produce brand

new examples that have never been seen before.

Generative Adversarial Networks (GAN) [10] is one of the most successful deep generative networks. GAN introduces an adversarial process into training that it simultaneously train two models. The generator G learns data distribution and try to deceive the discriminator D with its generated samples. And D tries to differentiate original dateset from the samples faked by generator G. GAN ultimately ends up with a powerful generator that can fully deceive its discriminator. GAN is well-known for its high quality / sharpness of generated images although it's extremely difficult to train.

Variational Auto-Encoders (VAE) [14], however, is much easier and flexible in training process while still generates pretty good images. Auto-encoders is a neural network that trained to reconstruct its input. It was originally designed as a tool for dimension reduction. Recent research Variational Auto-Encoder marries Bayesian probabilistic inference with deep neural networks and gains notable success in deep generative tasks. VAE jointly optimizes the reconstruction error and the KL divergence between approximate posterior and a predefined prior. They are excellent in learning and recovering a low-dimensional representation of the example.

1.2 Motivation

Using dual method to solve constraint optimization problem has been well researched for a long time. It suffers from limited scalability on large datasets. We combine the recently developed Corporative Stochastic Approximation (CSA) framework with PMD to solve it in primal space. It shares the benefit with PMD that the number of variables doesn't scale as the number of examples.

The standard Variational Auto-Encoders model makes a strong assumption that posterior distribution could be simply featured as a Gaussian distribution. In practical applications, however, true posterior could be multi-modal even in irregular shapes. Latest development in this area have realized the importance of an accurate structure assumption [11, 19, 22, 23] and our work also follows this line of research. By using mixture of kernels, PMD is capable of capturing more details

in posterior distribution.

1.3 Contribution of this Thesis

In this thesis we explore the applications of Particle Mirror Descent in both supervised and unsupervised learning.

We propose a primal method, Constrained Particle Mirror Descent (CPMD) that effectively handles classification problem with expectation constraint. It takes advantage of the the flexibility of PMD method and approximates the posterior distribution by particle filtering. Our experimental results demonstrate that CPMD algorithm runs much faster in large datasets, while still provides good classification accuracy compared to dual methods.

Secondly, we introduce a variant of VAE model where a mixture of particles / Gaussian kernels is used as posterior in latent space. It better characterizes the true posterior than the single modal Gaussian distribution used in standard VAE. Our analysis show better marginal likelihood estimation for test examples and produces visually plausible images of both digits and facial expressions.

Chapter 2

Related Work

2.1 Particle Mirror Descent

Particle Mirror Descent targets to solve the convex optimization problem

$$\min_{q(\theta)\in\mathcal{P}} L(q) := -\sum_{n=1}^{N}\Big[\int q(\theta)\log p(x_n|\theta)d\theta\Big] + KL(q(\theta)||p(\theta)), \tag{2.1}$$

where θ is latent parameter, $q(\theta)$ is the approximation of posterior, $\mathcal{P}$ is the valid density space, $p_0(\theta)$ is the prior distribution and $p(x_n|\theta)$ is the likelihood. The objective consists of the negative log likelihood of the examples in the dataset, and the KL divergence between learnt distribution and prior distribution. It is well-known that the optimal solution to (2.1) leads to the true posterior distribution.

The functional gradient of objective function with regard to $q(\theta)$ in density space is shown in Equation (2.2).

$$g(\theta) = \frac{\partial\mathcal{L}}{\partial q} = -\sum_{n=1}^{N}\log p(x_n|\theta) + \log q(\theta) - \log p_0(\theta) \tag{2.2}$$

PMD solves (2.1) by applying the inexact stochastic mirror descent algorithm and iteratively maintains and improves the weighted kernel density estimtors of the approximate solution. For a distribution $q(\theta)$, the kernel density estimator (KDE) based on m samples, $\{\theta_1,\ldots,\theta_m\}\sim q(\theta)$, can be written as $\hat{q}(\theta) = \sum_i \alpha_i K(\theta-\theta_i)$. Here, KDE is used as a general concept which could be either particle $K(\theta)=\delta(\theta)$, or Gaussian kernel $K(\theta) = e^{-\frac{\theta^T\theta}{2\sigma^2}}$.

It's forthright that two kinds of KDEs mentioned above both have their own advantages and lim-

itations. Given infinite particles, they are flexible enough to represent any distribution. It's not realistic to sample enormous particles in each iteration, since it could impose extensive computational cost. Limited number of particles forces loss of details in the approximation. As a result, we may not get very high quality samples from mixture of particles. Gaussian kernels, on the other hand, are naturally more capable of characterizing continuous distribution than particles. Thereby the samples from mixture of Gaussian kernels are more accurate and useful in providing details at the cost of more complex computations.

To keep a balance between high quality samples and speed concerns, PMD algorithm switches between two kinds of KDEs alternately. When the prior is close to true posterior, PMD samples m particles from prior. When the prior is less desirable, PMD samples m points as mean of Gaussian kernels from posterior approximation in last iteration. Subsequently it descends along the gradient direction and redistribute the weights over the KDEs. Thereafter it proceeds to use the redistributed mixture distribution as the latest posterior approximation and iterate until convergence.

2.2 Maximum Entropy Discrimination in Binary Classification

The MRE problem falls back to a variant of Support Vector Machine(SVM) when the decision rule is chosen as $\mathcal{L} = \theta^T x - b$ and prior as normal distribution $p_0(\theta) = \mathcal{N}(0, I)$ [12]. The dual problem of (1.1) can be further formulated as

$$\arg\max_{\lambda} J(\lambda) = \sum_t [\lambda_t + \log(1 - \lambda_t/c)] - \frac{1}{2} \sum_{t,t'} \lambda_t \lambda_{t'} y_t y_{t'} (X_t^T X_{t'}) \tag{2.3}$$
$$\text{s.t } 0 \leq \lambda_t \leq c, \sum_t \lambda_t y_t = 0.$$

It differs from the dual form of SVM in the additional term $\log(1 - \lambda_t/c)$ which enlarges the penalty of miss-classification. When $c \to \infty$, the problem regresses to exactly the same as dual SVM.

The optimization above can be solved by Sequential Minimal Optimization (SMO) algorithm pro-

posed by Platt [20]. It simultaneously updates at least 2 Lagrange multipliers at one time to meet the summation constraint.

MED converts the optimization to a dual problem and solves it by simultaneously optimizing a set of N dual variables, where N is equal to the number of examples in the dataset. This largely confines the scalability of the solution and requires strong assumption on the prior distribution. Solving it using a primal method, however, the time complexity will not scale up as the number of examples.

2.3 Stochastic Optimization with Expectation Constraints

Lan and Zhou [17] proposed a simple yet fast training framework, Corporative Stochastic Approximation (CSA), to solve the classic stochastic programming problem with expectation constraints over the decision variable x.

$$\min f(x) := \mathbb{E}[F(x, \xi)]$$
$$\text{s.t } g(x) := \mathbb{E}[G(x, \xi)] \leq 0$$

In each iteration of the algorithm, they first check if current solution x_t violates the condition. If so, the search direction is set as the stochastic gradient of constraint function $h_t = G^{'}(x_t, \xi_t)$ to control the violation. Otherwise, it searches along the stochastic gradient of objective function $h_t = F^{'}(x_t, \xi_t)$ in order to minimize target function value. Finally CSA performs a prox-mapping $x_{t+1} = P_{x_t,X}(\gamma_t h_t)$ in order to find the next estimation x_{k+1} in set X. Here we use $w_X : X \to R$ to denote a distance generation function. Then prox-mapping of a prox-function $V_X(z, x) = w_X(x) - w_X(z) - \langle \nabla w_X(z), x - z \rangle$ is defined as

$$P_{x,X}(\phi) := \arg\min_{z \in X} \{\langle \phi, z \rangle + V_X(x, z)\}.$$

The framework achieves theoretically optimal convergence rate of $O(1/\sqrt{N})$ where N is the number of iterations. We will show later in this thesis that, by integrating CSA framework, we are able to

Figure 2.1: Pipeline of Variational Auto-Encoders

apply particle mirror descent to solve MRE problem in primal space.

2.4 Variational Auto-Encoders

As shown in Figure 2.1, standard Variational Auto-Encoders model is mainly composed of three components, encoder $q(z|X)$, sampling layer and decoder $p(X|z)$. Encoder and decoder are both instantiated as neural networks with parameter set ϕ, θ respectively and jointly optimized using loss function

$$\min_{\phi,\theta} E_{x\sim D}\Big[- E_{z\sim q_\phi}\big[\log p_\theta(X|z)\big] + KL(q_\phi(z|X)||p(z))\Big]. \tag{2.4}$$

Sampling operation is usually not differentiable and thus interrupts the back propagation process in neural network. To fix this, VAE employs a reparameterization trick [14]. Instead of sampling from $z \sim \mathcal{N}(\mu, \sigma^2)$, VAE samples from normal distribution $\epsilon \sim \mathcal{N}(0, I)$ instead and constructs samples using $z = \mu + \epsilon\sigma$. In this way VAE moves the sampling operation as an input to neural network and thus whole pipeline is now differentiable.

Chapter 3

Particle Mirror Descent in Classification

3.1 Constrained Particle Mirror Descent

Thinking about solving MRE problem in primal space, PMD comes into play naturally since the task is to estimate a distribution. However, original PMD is not capable of handling expectation constraint. Inspired by Corporative Stochastic Approximation(CSA) [17] algorithm, we fit PMD in the CSA training framework by viewing $q(\theta)$ as a decision variable. Therefore the problem is now an optimization over a convex functional space. According to CSA framework, we first check if the constraint is met with current example. If so we minimize along the stochastic gradient of target function $F = KL(q(\theta)||p(\theta))$, otherwise we descend along the stochastic gradient of constraints $G = -\int q(\theta)[y_t\mathcal{L}(x_t, \theta) - \xi_t]d\theta)$. This new algorithm, Constrained Particle Mirror Descent (CPMD), allows us to solve the MRE problem in primal space directly.

An overview of the complete algorithm is shown in Algorithm 1. Theoretically, to guarantee a convergence rate of $O(1/T)$, we should set the number of particles sampled to increase quadratically of iteration number $m_t = O(t^2)$. Despite that as we sample and evaluate at only one example in each iteration, sampling $\{\theta_i\}_{i=1}^{m_t}$ from $q(\theta)$ will be the major speed bottleneck after a short time. As a trade-off, we initialize m_t to be a relatively large number and increase it linearly in the experiment.

Algorithm 1 CPMD: A primal method to solve optimization prolem with expectation constraint

Input: dataset $\{(x_i, y_i)\}_{i=1}^N$, prior density $p_0(\theta)$, number of iterations T, number of samples in each iteration $\{m_t\}$, margin $\{\xi_t\}$, step size $\{\gamma_t\}$

Initialize: $q_0(\theta) = p_0(\theta)$

for $t = 1, 2, \ldots, T$ **do**

$\{\theta_i\}_{i=1}^{m_t} \overset{i.i.d}{\sim} q_{t-1}(\theta)$

sample $x_t \overset{unif.}{\sim} X$

$$g_t(\theta_i) = \begin{cases} F'|_{\theta=\theta_i} = \log q_{t-1}(\theta_i) - \log p_0(\theta_i), & G(\theta_i) \leq 0 \\ G'|_{\theta=\theta_i} = \xi_t - y_t \mathcal{L}(x_t, \theta_i), & \text{otherwise} \end{cases}$$

$\alpha_i \leftarrow q_{t-1}(\theta_i) \exp^{-\gamma_t g_t(\theta_i)}, \forall i$

$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{i=1}^{m_t} \alpha_i}, \forall i$

$q_t(\theta) = \sum_{i=1}^{m_t} \alpha_i K(\theta - \theta_i)$

Output: $q_T(\theta) = \sum_{i=1}^{m_T} \alpha_i K(\theta - \theta_i)$

Dataset	**(#examples, #features)**	**Dual acc/time**	**Primal acc/time**
breast-cancer	(683,10)	0.9561 / **20.1s**	**0.971** / 41.5s
svmguide3	(1105,22)	**0.798** / 200s	0.756 / **54s**

Table 3.1: Comparison of dual and primal methods on classification tasks

3.2 Experiments

We perform experiments on two commonly used dataset for binary classification. Both breast-cancer dataset and svmguide3 are from UCI Machine Learning Repository[1]. The size of each dataset is shown in Table 3.1. We randomly split each dataset to use 70% as training and 30% for testing. Dual solution of (2.3) is implemented using SMO algorithm as stated in MED paper. And primal method is implemented using the proposed CPMD algorithm above. We sample 1000 particles in the first iteration and this number increases 30 after every epoch. In total the primal method runs over the training set 8 epochs. Starting with a Gaussian prior, we use particles as major KDE and switch to Gaussian kernel once every epoch to help capture better samples. To avoid duplicate statements, we use a unified $K(\theta - \theta_i)$ to represent both KDEs.

Classification accuracies and running time shown in Table 3.1 indicate that in a relatively small dataset, primal solution enjoys slightly better classification performance but it takes longer to train. However when the size of dataset grows, time consumed by dual solution grows dramatically while

[1] `http://archive.ics.uci.edu/ml/`

primal solution still achieve a good performance in a consistent amount of time. This actually makes sense since in PMD, the speed is determined by the number of kernels used in each iteration and total number of iterations. The size of dataset doesn't have much impact in term of speed.

Also it worths pointing out that after training svmguide3 dataset within 6184 iterations (8 epochs through training set), there are 1412 times that constraint is met so the optimization descends towards gradient of objective, and 4772 times that it descends towards the gradient of constraint function. As shown in the Algorithm 1, computing the gradient of objective function involves a log-sum-exp when using Gaussian kernel as KDE, so the fast speed also benefits from alternately optimizing over objective and constraint.

In this chapter we proposed a primal method, Constrained Particle Mirror Descent to solve classification problems with expectation constraints. Our experimental results on breast-cancer and svmguide3 dataset showed notable scalability while maintaining comparable accuracy.

Chapter 4

Particle Mirror Descent in Deep Generative Networks

4.1 Pipeline Overview

PMDVAE is a variant of VAE where the distribution in latent space is structured as a weighted sum of m KDEs $q(z|x) = \sum_{i=1}^{m} \alpha_i K(z - z_i(x))$ instead of a single Gaussian distribution. The weight vector $\{\alpha_i\}_{i=1}^{m}$ is updated by functional gradient similarly as in that of PMD.

The whole pipeline is shown in Figure 4.1. Just like standard VAE, our model consists of an encoder to project examples X to a latent space representation z, and a decoder to reconstruct X from z. Both encoder and decoder are learnt jointly to maximize log likelihood with the presence of a KL divergence in latent space as regularization term. Since we use multiple KDEs instead of one, the reshape layer converts the output to desired shape. Then sampling layer samples from $q(z)$ and feed it into decoder. The major change we made in the pipeline is that we have an additional module to update α after training each batch.

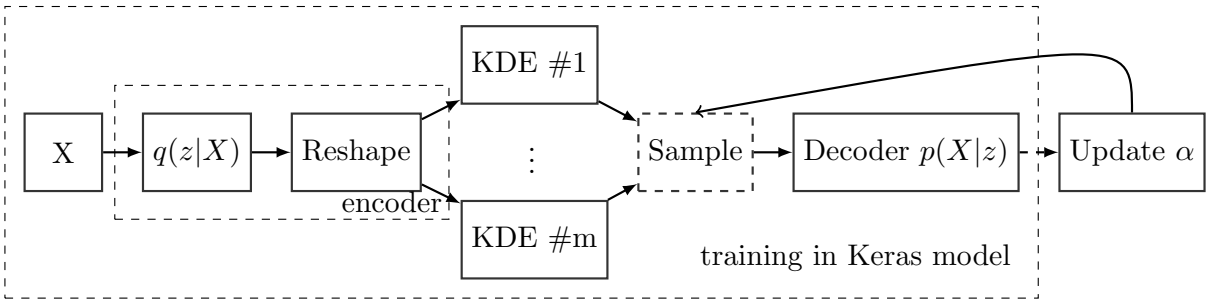


Figure 4.1: Pipeline of PMDVAE

4.2 Generative Process

For the simplicity of notation, we use b to denote batch size, k to denote the dimension of examples, d to represent the dimension of latent space, which is usually set to 2 in our experiments. The output of encoder are $\{z_i(x)\}_{i=1}^m$. We assume the likelihood function is a Gaussian distribution with reconstructed $\mu(z)$ as mean and identity matrix I as variance. The generative process in each iteration is summarized in following three steps.

1. A batch of input examples x of shape (b, k) is fed into encoder which is instantiated as a neural network with parameter set ϕ, the output is $\{z_i\}_{i=1}^m$ of shape (b, m, d) which are m particles/means of Gaussian kernels. If Gaussian kernels are used as KDE, we also have an identical structure to learn diagonal variance matrix σ_i of the same shape.

2. Construct posterior approximation from $\{z_i\}_{i=1}^m$ and sample z of shape (b, d) from $q(z|x)$.

$$z \sim q_\phi(z|x, \alpha) = \begin{cases} \sum_{i=1}^m \alpha_i \delta(z - z_i), & \text{KDE = particle} \\ \sum_{i=1}^m \alpha_i \mathcal{N}(z_i, \sigma_i), & \text{KDE = Gaussian kernel} \end{cases} \tag{4.1}$$

3. Feed latent representation z to decoder $p_\theta(x|z)$ which is also instantiated as a neural network with parameter set θ. The output is reconstructed example mean $\mu(z)$ of shape (b, k).

4.3 Approximation Inference as Optimization

Recall that in the objective function of original Variational Auto-Encoders as shown in Equation (2.4), the KL divergence part actually also works as a regularization over z distribution. Comparing VAE loss function with the PMD objective in Equation (2.1), it's obvious that these two objective functions are identical. The only difference is that in VAE it replaces z with $z|x$ in posterior approximation. So here we reformulate the objective function for our PMDVAE model as:

$$\min \mathcal{L} = \min_{\phi,\theta,\alpha} E_{x\sim D}\Big[- E_{z\sim q_\phi}\big[\log p_\theta(x|z)\big] + KL(q_\phi(z|x,\alpha)||p(z))\Big]. \tag{4.2}$$

Note that the major difference between the original loss function and the loss function above is that instead of only training normal neural network using gradient descent with respect to parameter ϕ, ψ, we also perform Stochastic Functional Gradient Descent(SFGD) with respect to q in function space. Since q is represented as a weighted sum of particles/Gaussian kernels with regard to weight vector α, the distribution update of q is reflected as the redistribution of weights in α vector.

While optimizing the objective in Equation (4.2), it is straight forward that θ and ϕ could be updated through gradient descent in neural network. Next we will briefly go through some slight differences in optimization depends on particles or Gaussian kernels we use. And finally we introduce how to perform weight redistribution for parameter α in Chapter 4.4.

4.3.1 Particle as KDE

Here we assume posterior distribution

$$q(z|x,\alpha) = q(\{z_i, \alpha_i\}_{i=1}^m) = \sum_{i=1}^m \alpha_i \delta(z = z_i) \tag{4.3}$$

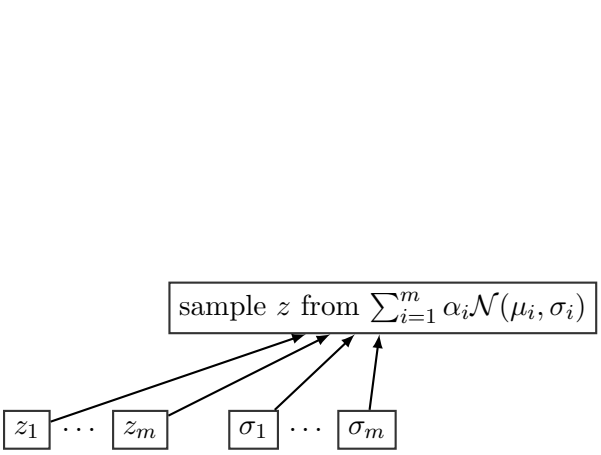
is a weighted sum of m particles, where $\{z_i\}_{i=1}^m$ are generated from encoder neural network.

The likelihood part of loss function Equation (4.2) could be simplified as:

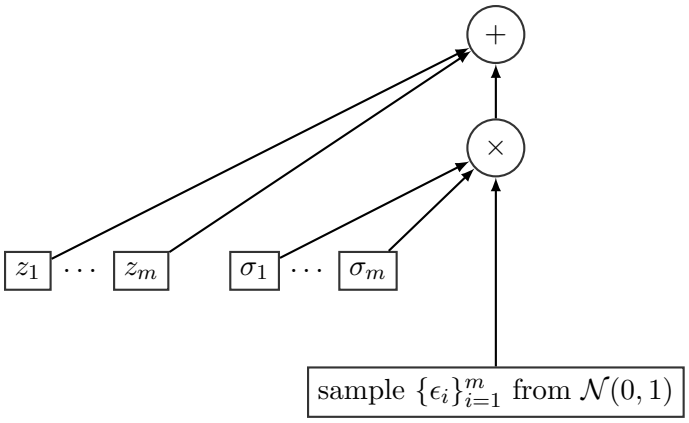
$$-E_{z\sim q} \log p_\theta(x|z) = \sum_{i=1}^m \log p_\theta(x|z_i). \tag{4.4}$$

And the regularization part could be simplified as:

$$\begin{aligned} KL(q(\{z_i, \alpha_i\}_{i=1}^m)||p(z)) &= \int_{z\sim q} \Big[\log q(z) - \log p(z)\Big] \\ &= \sum_{i=1}^m q(z_i)\Big[\log q(z_i) - \log p(z_i)\Big] \\ &= \sum_{i=1}^m (\alpha_i \log \alpha_i - \alpha_i \log p(z_i)). \end{aligned} \tag{4.5}$$



(a) Sampling from mixture of Gaussian kernels



(b) Sampling from mixture of Gaussian kernels with reparameterization trick

Figure 4.2: Reparameterization trick in PMDVAE

4.3.2 Gaussian kernel as KDE

Here we assume the posterior distribution

$$q(z|x,\alpha) = q(\{z_i, \sigma_i, \alpha_i\}_{i=1}^{m}) = \sum_{i=1}^{m} \alpha_i \mathcal{N}(z_i, \sigma_i) \tag{4.6}$$

is a mixture of Gaussian model. And in the experiments we either use a prior normal distribution $p(z) \sim \mathcal{N}(0, I)$ or a predefined mixture of Gaussian distribution. The inference below is based on $\mathcal{N}(0, I)$ and it could be easily extended to the mixture prior case. For the simplicity of notation, we use k to denote the dimension of original space and d to denote the dimension of latent space.

Similar to standard VAE, sampling from mixture of Gaussian cut the connection between likelihood and encoder parameter θ since it's a non-differentiable operation. This could also be solved using the reparameterization trick [14] which is explained in Figure 4.2. Thus the likelihood part could be written as:

$$\begin{gathered} -E_{z\sim q_\phi} \log p_\theta(x|z) = -E_{\epsilon\sim\mathcal{N}(0,1)} \log p_\theta(x|z), \\ \text{where } z \sim \sum_{i=1}^{m} \alpha_i \delta(z_i + \sigma_i * \epsilon_i) \text{ and the shape of } \epsilon \text{ is } (m, d). \end{gathered} \tag{4.7}$$

And the regularization part includes a weighted sum of KL divergence between two Gaussian

distributions, it could be simplified as:

$$KL(q(\{z_i, \alpha_i\}_{i=1}^m)||p(z)) = \sum_{i=1}^{m} \alpha_i KL(\mathcal{N}(z_i, \sigma_i)||\mathcal{N}(0, I))$$
$$= \sum_{i=1}^{m} \frac{1}{2}\alpha_i \Big[-\sum_{j=1}^{d} \log \sigma_j - d + \sum_{j=1}^{d} \sigma_j + z_i^T z_i \Big]. \tag{4.8}$$

4.4 Weight Redistribution

The functional gradient of Equation (4.2) over distribution q differs when we use different KDEs. Therefore the update rule for weight distribution is also different.

4.4.1 Particle as KDE

The update rule for particles [7] is

$$\alpha_i = \alpha_i^{1-\gamma_t} p(x_t|z_i)^{N\gamma_t}, \forall i. \tag{4.9}$$

According to our assumption, the decoder $p(X|z)$ is Gaussian thus $x_t|z_i \sim \mathcal{N}(\mu(z_i), I)$. Then the likelihood function in Equation (4.9) can be written in the following way.

$$p(x_t|z_i) = \frac{1}{\sqrt{(2\pi)^k}} e^{-\frac{1}{2}(x_t - \mu(z_i))^T (x_t - \mu(z_i))}.$$

4.4.2 Gaussian kernel as KDE

The update rule for mixture of Gaussian kernels [7] is

$$\alpha_i = q_t(z_i)^{-\gamma_t} p(z_i)^{\gamma_t} p(x_t|z_i)^{N\gamma_t}, \forall i. \tag{4.10}$$

Each part can be computed separately as follows. Σ_j is a diagonal matrix with σ_j as diagonal

elements.

$$q_t(z_i) = \sum_{j=1}^{m} \alpha_j K(z_i, z_j)$$
$$= \sum_{j=1}^{m} \alpha_j \frac{1}{\sqrt{2\pi\Sigma_j}} e^{-\frac{1}{2}(z_i - z_j)^T (z_i - z_j)}, \tag{4.11}$$

$$p(z_i) = \frac{1}{\sqrt{(2\pi)^d}} e^{-\frac{1}{2} z_i^T z_i}, \tag{4.12}$$

$$p(x_t|z_i) = \frac{1}{\sqrt{(2\pi)^k}} e^{-\frac{1}{2}(x_t - \mu(z_i))^T (x_t - \mu(z_i))}. \tag{4.13}$$

Thus we can optimize $\mathcal{L}$ using coordinate gradient descent as shown in Algorithm 2.

Algorithm 2 PMDVAE

Input: training set X, $p(z) \sim \mathcal{N}(0, I)$, step size $\{\gamma_t\}$
Initialize: $\alpha_0 = \{\alpha_0^i\}_{i=1}^m$ and ϕ_0, θ_0.
for batch t = 1:T **do**
fix $\theta_{t-1}, \phi_{t-1}, \alpha_{t-1}$
1. Pass example x_t into encoder, get latent representation $\{z_i\}_{i=1}^m$ (particle) or $\{z_i, \sigma_i\}_{i=1}^m$ (Gaussian kernel)
2. Sample z from (4.3) (particle) or (4.6) (Gaussian kernel)
3. Pass z to decoder, get the reconstructed mean $\mu(z)$
4. Back propagate in encoder and decoder jointly using loss function shown in (4.4), (4.5) (particle) or (4.7), (4.8) (Gaussian kernel) to get weight update θ_t and ϕ_t.
5. Use (4.9) (particle) or (4.10) (Gaussian kernel) to compute α_t.
6. Normalize α_t, $\alpha_i^t = \frac{\alpha_i^t}{\sum_{i=1}^m \alpha_i^t}$

4.5 Implementation Details

Keras [6] is a high-level neural network API built on top of either TensorFlow [1] or Theano [2]. Keras provides fully configurable modules of layers, cost functions, optimizers and so on. It's easy to get started and enables fast experiments on both CPU and GPUs. Also Keras provides an example of VAE model that we could use as baseline. Therefore our PMDVAE model is implemented on top of Keras framework with Theano as backend.

4.5.1 Adaption of Keras

PMDVAE is a mixture model for both particles and Gaussian kernel in latent space. Thus the output of encoder is several particles / means for Gaussian kernels. We have to perform a sampling over the mixture distribution to extract the final latent representation. In Figure 4.1 we use a dashed box to represent sampling layer because this mixture ensemble operation is actually done in the loss function implementation, not in the middle of the pipeline. In other words, this sampling simply pass along all particles without change or in the Gaussian kernel case, samples from each Gaussian kernel and pass all samples to decoder. Weighted sum of these samples are done when defining loss function of the whole neural network.

The main reason for this change is that when we update α using equation (4.9) and (4.10), $p(x_t|z_i)$ is essential in both cases. Consequently it indicates that likelihood of x_t in reconstructed mean for each latent particle/mean of Gaussian kernel should be computed. This implies the weighted sum operation should not be done before decoder.

The other reason that weighted sum should be implemented after decoder is that, when Keras builds the layers, its inputs and outputs, as well as every operation will be compiled to memory space as tensors. Theano manages its own memory space which is separate from that of the Python code uses. This isolation forces the parameter passed from Python to remain the same all along the training process. Any python variable that is passed into Keras layer definition will be fixed in the model initialization and doesn't change as the python variable changes.

However, our model specifically requires intensive updates from python code to Keras model that each time our weight vector α is updated, we should manage to update the α variable in Theano memory space in real-time as well. This operation is not common thus not supported by Keras API. To dynamically apply the α updated by functional gradient descent in python code, we have to find a place in Keras that our variable values could be somehow refreshed in a "manual" real-time manner. To work around, we re-compile the Keras model after training each batch after α is updated using the gradient evaluated at current batch. Recompiling in Keras won't change the weights in each layer, but only rebuild the loss function which is the only place that α comes into play in the whole process.

4.5.2 Speed Up

In Theano, computing derivatives of some expression f with respect to its parameter x can be done blindly and automatically by calling tensor function $T.grad$. Thus when we pass the loss function into Keras model, we actually don't have to care about how back propagation is calculated in each layer. It's all done in a black box.

On the other hand, α is updated through functional gradient descent which is not compatible with what Keras currently supports. Integrating functional gradient computation with Keras requires a lot of changes in Keras built-in layer architecture as well as optimizer design. To adhere to the current Keras framework without completely change it, we have to sacrifice the speed and implement the weight redistribution module in python instead of inside Keras model.

By profiling the python program, we figure out that the speed bottleneck in our implementation are caused by frequently use of two operations on large ndarray matrices, multi-dimensional array tiling and sum of squares in equation (4.13), which together consume more than 80% of total running time.

The first tile part can be improved by using numpy [24] broadcast function instead of tile given that it allows us to perform computation without actually creating multiple copies in memory.

To speed up sum of squares computation, We tried four kinds of implementations.

- Naive python for loop
- Numpy functions which are commonly used in scientific python programming.
- Cython [4] which is a Python extension that directly compiles to C executable.
- Numba [16] which is a just-in-time compiler designed for scientific programming.

Their performance are shown in Table 4.1.

It's obvious that Cython yields the highest speed in the comparison. Therefore, this implementation is used in our experiments.

Implementation	time
python for loop	21.1s
Cython	304ms
Numpy	178ms
Numba of python for loop	78.1ms

Table 4.1: Speed comparison of four sum of squares implementations. The experiments are done on a 3d-array of size 100*1000*784

4.6 Experiments

In the following, we will show the results of PMDVAE model on two generative tasks. For the MNIST dataset, quantitative measurements based on marginal likelihood and clustering accuracy are provided along with the generated digits. In addition, we display a 2d visualization of generated face expressions from Frey Face dataset to demonstrate PMDVAE can be applied on complex generative tasks as well.

4.6.1 MNIST Dataset

MNIST dataset of handwritten digits[1] constains a train set of 60000 examples and a test set of 10000 examples, each of the same size $(28, 28)$. We train two PMDVAE models on MNIST dataset: one with a mixture of 10 particles, the other with a mixture of 10 Gaussian kernels. For simplicity, we will refer them as particle model and Gaussian kernel mode respectively. To obtain a better sense of the quality of generated images along with how they gradually change over latent space, linearly spaced coordinates on latent 2d unit square are transformed through the inverse CDF of 2d normal distribution and then used as latent representation to feed into decoders. Their generated images look visually similar. So we only show the reconstructed digits from particle model in Figure 4.3. Please refer to Figure A.1 in appendix for the generated image of standard VAE and mixture of Gaussian kernel model.

One interesting small experiment is that when we eliminate the neural network for estimating variance and update the KL divergence accordingly in standard VAE, it actually falls back to a single particle as posterior but is still capable of generating good images of almost the same quality.

[1] `http://yann.lecun.com/exdb/mnist/`

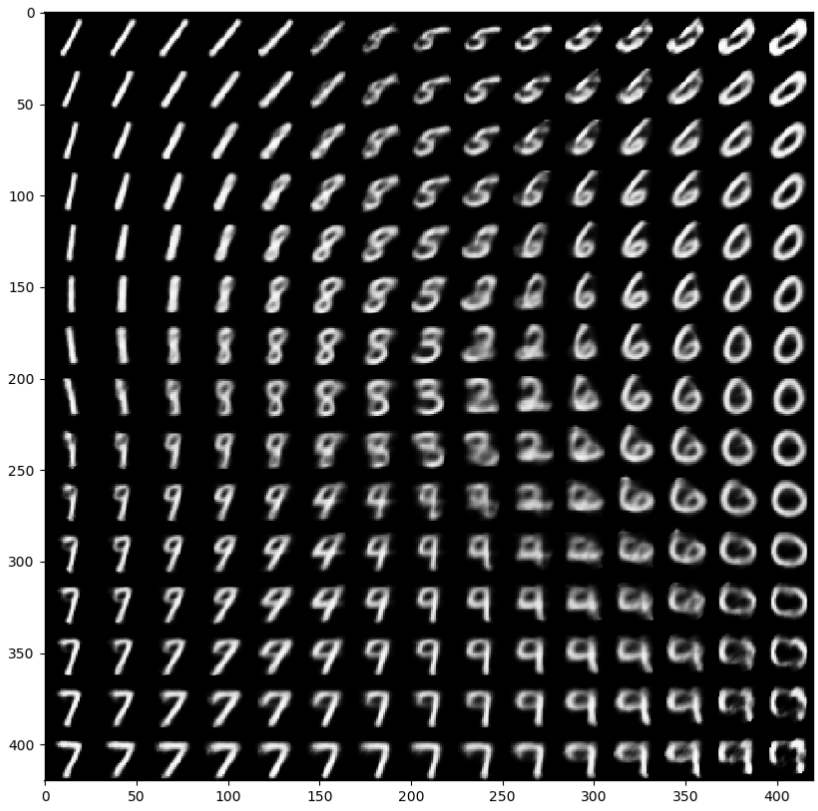
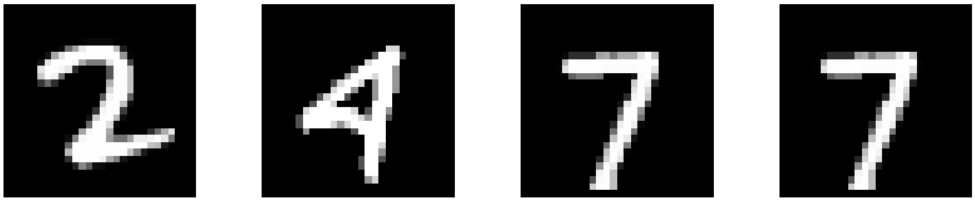


Figure 4.3: PMDVAE with mixture of 10 particles: reconstructed digits from linearly spaced coordinates in latent space.



(a) test image 467 (b) test image 2847 (c) test image 5402 (d) test image 1721

Figure 4.4: Visualization of 4 random examples in MNIST test set

Algorithm	$-\log p(x_{467})$	$-\log p(x_{2847})$	$-\log p(x_{5402})$	$-\log p(x_{1721})$
standard VAE	755.06	949.62	3941.33	1660.0
Mixture of 10 particles	745.03	866.89	2177.16	925.48
Mixture of 10 Gaussian kernels	737.57	737.08	729.42	727.60

Table 4.2: Marginal likelihood

This observation indicates that using neural networks as estimators, VAE has gained excessive learning capacity. Thus the performance bottleneck of VAE model is not learning capacity, but the posterior structure assumption as we will show shortly.

To quantitatively evaluate these generative models, we compare the marginal likelihood of random examples in the final converged models in Table 4.2. These four examples are sampled from test set and the digits are shown in Figure 4.4. Then we evaluate the marginal likelihood using MC approximation $\log p(x_i) \approx \log \frac{1}{S} \sum_S p(x_i|\hat{z}_{i,s}) p_0(\hat{z}_{i,s})/q(\hat{z}_{i,s})$ using 100 samples.

By comparing mixture models with standard VAE, we can conclude that multi-modal approximation is closer to true posterior and thus gives better estimation of marginal likelihood. Besides, it shows that mixture of 10 Gaussian kernels has more balanced marginal likelihood distribution over different examples. This could be possibly caused by the smoothness of Gaussian kernel distribution and the weighted sum operation.

Model	$K = 3$	$K = 5$	$K = 10$
Single Gaussian kernel	0.6831	0.7015	0.7185
Mixture of 10 particles	0.6879	0.7058	0.7211
Mixture of 10 Gaussian kernels	0.695	0.7086	0.7239

Table 4.3: K Nearest Neighbor clustering using latent representation

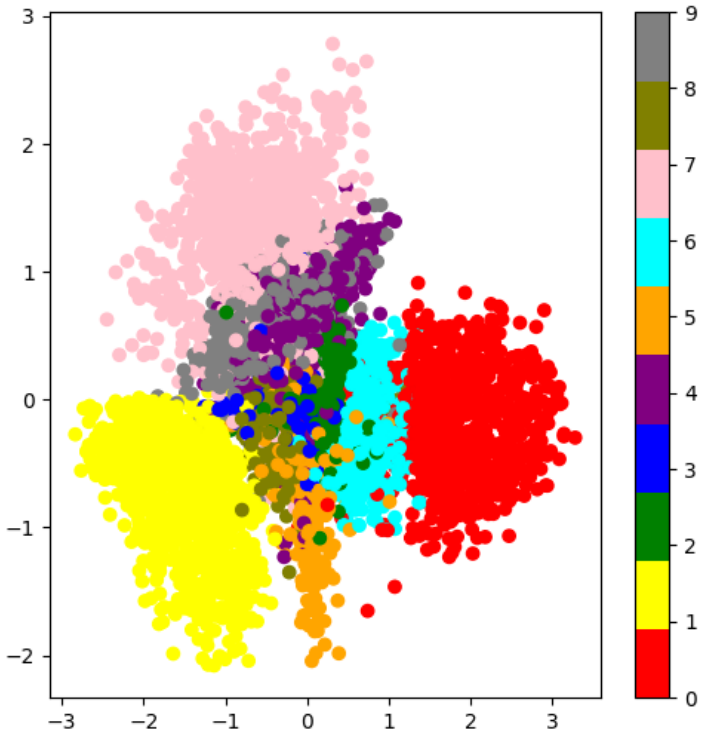


Figure 4.5: Clustering result on test set.

As a side benefit, VAE and its variants could be used as a tool for dimension reduction. Clustering using latent features provides an additional measurement to evaluate the encoder model performance. We perform K Nearest Neighbor classification on the latent representation of test set with different choices of K as shown in Table 4.3. The clustering result on test set can visualized in Figure 4.5.

Models using mixture posterior both slightly outperform standard VAE in the classification task in each choice of K. This is actually in accordance with our analysis above. Using multi-modal posterior approximation could better characterize true posterior and obtain more accurate latent representations.

4.6.2 Frey Face Dataset

Frey Face dataset [2] contains almost 2000 images of Brendan's face, taken from sequential frames of a small video. As a generative task, frey face dataset is more difficult than digit since it requires

[2] `http://www.cs.nyu.edu/~roweis/data.html`

jointly capture the position of eyes, nose, mouth and then form a visually plausible face expression. It's more likely to be multi-modal when projected to 2d latent space. Thus we train a PMDVAE model with mixture of two Gaussian kernels to approximate posterior for 200 epochs. We show reconstructed face distribution from trained decoder in Figure 4.6. PMDVAE can generate high quality images even in face expression dataset.

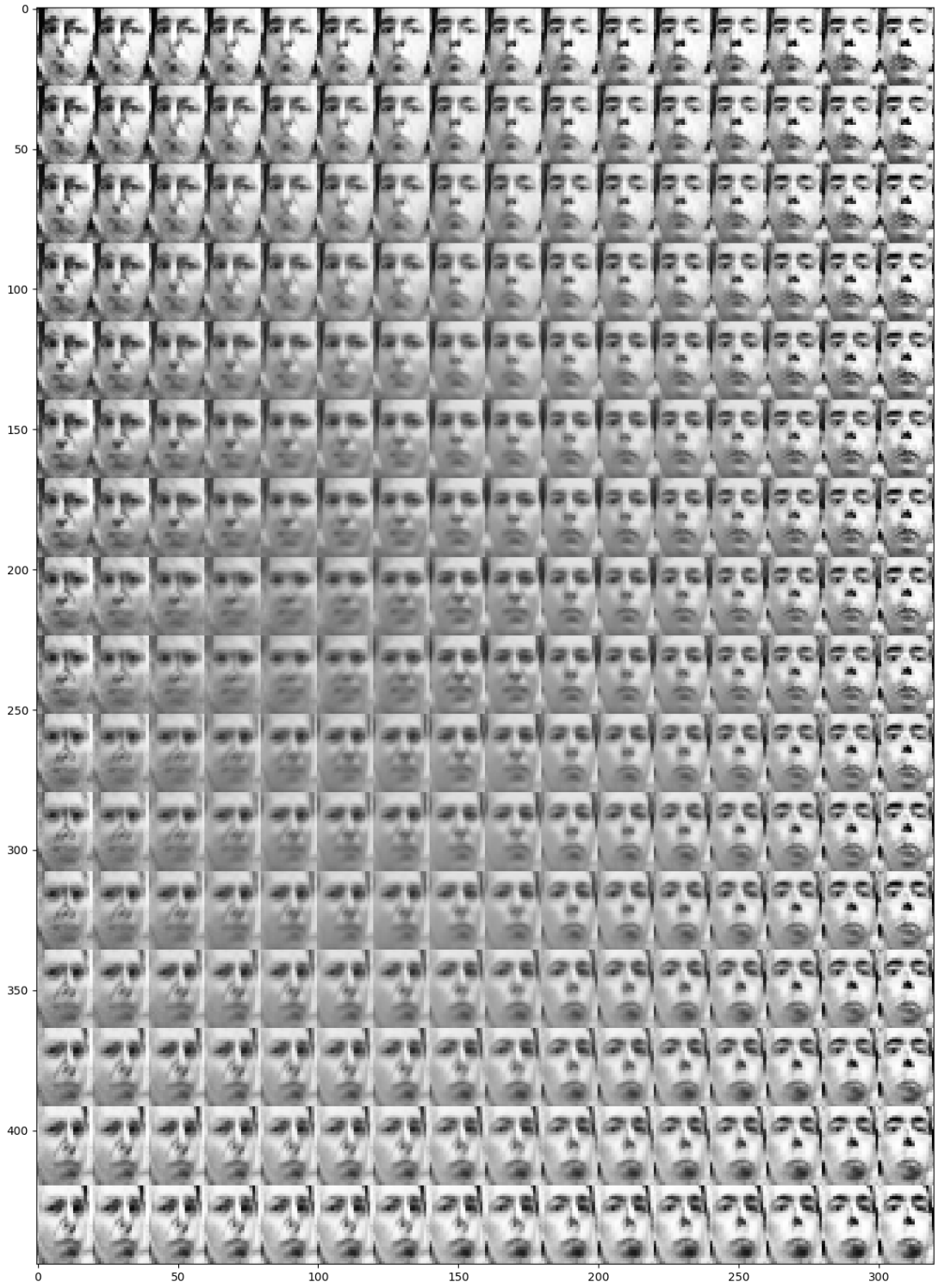


Figure 4.6: PMDVAE reconstructed images of Frey Face Dataset

Chapter 5

Conclusions

In this thesis we delved into the variants of Particle Mirror Descent in both supervised and unsupervised contexts. Leveraging the CSA framework, we proposed a simple yet scalable algorithm, CPMD to solve the MRE problem in primal space. Furthermore, we investigated the learning capacity of deep generative networks and introduced a variant of Variational Auto-Encoders that could flexibly better characterize irregular multi-modal true posterior distributions in latent space. Our experiment results clearly showed that using more flexible posterior structure could effectively improve the quality of latent representations of deep generative models.

Although we observed considerably improvements in the both models, there are still plenty of future directions. In the Constrained PMD work, the framework can be easily extended to customized prior distribution, likelihood function and various constraints. More experiments are needed to evaluate the efficiency and flexibility of the CPMD framework.

Our PMDVAE algorithm, although preserves the flexibility of modeling multi-modal posterior distributions, it still can be improved. Firstly, moving weight redistribution module out of Keras imposes huge computation cost compared to normal Keras model. We had to make complex computation on large matrices and recompile Keras model in each iteration, which can be avoided by integrating functional gradient descent in Keras. Although SFGD is getting more and more popular in research, an authoritative implementation within deep learning framework is still needed. Secondly, besides particles and Gaussian kernels used in our experiments, there are a significant amount of other good kernels could be used as kernel density estimators. Eventually the model should be free to select any KDE that best fit our prior knowledge of the task.

References

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

[2] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

[3] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1):5–43, 2003.

[4] S. Behnel, R. Bradshaw, C. Citro, L. Dalcín, D. S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science and Engineering*, 13:31–39, 2011.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2001.

[6] F. Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[7] B. Dai, N. He, H. Dai, and L. Song. Provable bayesian inference via particle mirror descent. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 985–994, 2016.

[8] B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M.-F. F. Balcan, and L. Song. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, pages 3041–3049, 2014.

[9] D. Foreman-Mackey, D. W. Hogg, D. Lang, and J. Goodman. emcee: The mcmc hammer. *Publications of the Astronomical Society of the Pacific*, 125(925):306, 2013.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[11] P. Goyal, Z. Hu, X. Liang, C. Wang, and E. Xing. Nonparametric variational auto-encoders for hierarchical representation learning. *arXiv preprint arXiv:1703.07027*, 2017.

[12] T. S. Jaakkola, M. Meila, and T. Jebara. Maximum entropy discrimination. In *nips*, volume 12, pages 470–476, 1999.

[13] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[14] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[15] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. *IEEE transactions on signal processing*, 52(8):2165–2176, 2004.

[16] S. K. Lam, A. Pitrou, and S. Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, page 7. ACM, 2015.

[17] G. Lan and Z. Zhou. Algorithms for stochastic optimization with expectation constraints. *arXiv preprint arXiv:1604.03887*, 2016.

[18] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[19] E. Nalisnick, L. Hertel, and P. Smyth. Approximate inference for deep latent gaussian mixtures. In *nips workshop*, 2016.

[20] J. C. Platt. 12 fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods*, pages 185–208, 1999.

[21] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[22] A. B. Said, A. Mohamed, T. Elfouly, K. Harras, and Z. J. Wang. Multimodal deep learning approach for joint eeg-emg data compression and classification. *arXiv preprint arXiv:1703.08970*, 2017.

[23] K. Sohn, W. Shang, and H. Lee. Improved multimodal deep learning with variation of information. In *Advances in Neural Information Processing Systems*, pages 2141–2149, 2014.

[24] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science and Engineering*, 13:22–30, 2011.

[25] E. P. Xing, M. I. Jordan, and S. Russell. 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., 2002.

Appendix A

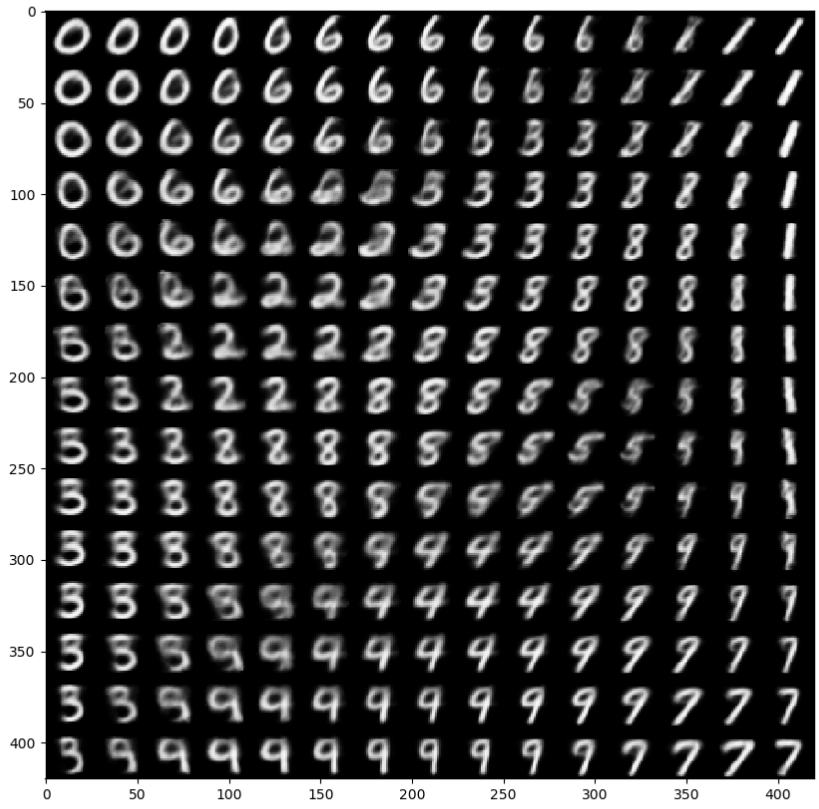


Figure A.1: PMDVAE with mixture of 10 Gaussian kernels: reconstructed digits from linearly spaced coordinates in latent space.