Mixture of Experts using Discrete VAE

Gurpreet Singh 150259
Abhisek Panda 150026
Aakarsh Gajbhiye 150067

Abstract

A lot of data in the real world exists in arbitrarily shaped clusters. Applying regression/classification models to such data can be pointless if the complexities of the hypothesis class of these models is low. Mixture of experts try to solve this problem by using a different expert/learner for each cluster, therefore creating a prediction using not one, but a mixture of learnt experts. We propose a superior gating function for these mixture of experts using Variational Autoencoders designed to perform as gating functions.

1. Introduction

Humans can perceive cluster patterns in data even with arbitrarily sized and shaped clusters. The same cannot be said for most machine learning models. In fact, most Mixture of Experts (ME) models are built assuming the underlying clusters of the data to be simple gaussians. This problem can, however, be alleviated by using latent variable models.

Latent variable models essentially decode the inherent properties of the data. In the latent space, since the complexities of the data are unravelled, the clusters are simple enough to be modelled properly as gaussians. Such clustering still allows arbitrary cluster shapes in the original space.

We use the same idea, and construct a generative model based on Variational Autoencoders (VAE) to find apt latent variables. Using this, we cluster the data, and form a simple gating function based on a Feed Forward Neural Network. The model is explained in detail in Section 3. We observe interesting results, with our model outperforming naive gating functions by a huge margin. These experiments are recorded in section 4.

2. Background

2.1. (Generative) Latent Variable Models

Latent variables are variables that are not directly observed but are rather inferred from other variables that are observed wik. Suppose we have a task of creating digits (such as in MNIST), it would help to know which digit (0-9) we wish to create, what should be the width of the digit, at what angle should the digit be tilted. Such decisions can be informally called as latent variables, as these decisions, in a generative story, help us to create the digits, and in an inference story, help us understand the

inherent properties of the digits.

Such a model where we actually generate data similar to the provided training data are known as generative models. A formal description of a generative model with latent variables is described below.

Say, we have a vector of latent variables $\mathbf{z}$ in a lower-dimensional space $\mathcal{Z}$, which we can easily sample according to some probability density function (pdf) $\mathbb{P}[\,\mathbf{z}\,]$ defined over $\mathcal{Z}$. Moreover, we also have a family of deterministic functions $f(\mathbf{z};\boldsymbol{\theta})$, parameterized by $\boldsymbol{\theta}$ in some space $\boldsymbol{\Theta}$, such that $f : \mathcal{Z} \times \boldsymbol{\Theta} \to \mathcal{X}$, where $\mathcal{X}$ denotes the original data space. Our objective, therefore, is to find the correct set of parameters $\boldsymbol{\theta}$ so that the construction of the generated data is closest to the real data. More formally, we want to find a function f, such that the quantity

$$\mathbb{P}[\,\mathbf{x}\,] = \int_{\mathbf{z}} \mathbb{P}\left[\,\mathbf{x}\,\middle|\,\mathbf{z},\boldsymbol{\theta}\,\right] \cdot \mathbb{P}[\,\mathbf{z}\,]\ \mathrm{d}\mathbf{z}$$

is maximized. For example, the probability $\mathbb{P}\left[\,\mathbf{x}\,\middle|\,\mathbf{z},\boldsymbol{\theta}\,\right]$ can be formulated as $\mathcal{N}\left(\mathbf{x}\,\middle|\,f(\mathbf{z};\boldsymbol{\theta}),\sigma^2\mathbf{I}\right)$ in case $\mathcal{X} \in \mathbb{R}$. The intuition behind this framework—called *maximum likelihood* is that if the model is likely to produce training set samples, then it is also likely to produce similar samples, and unlikely to produce dissimilar ones.

Each generative model is defined by a generative story which allows us to visually interpret the interdependence of variables, more particularly dependence of observed variables on the defined latent variables.

In most cases, the posterior of the latent variables given a data point ($\mathbb{P}\left[\,\mathbf{z}\,\middle|\,\mathbf{z}\,\right]$) is in intractable form. In such cases, these models are trained using methods such as Monte Carlo Markov Chain(MCMC) or Variational Inference. Popular choices of generative models include variants of the vanilla Variational Autoencoder (Kingma and Welling, 2013). The reasons for the same are that the VAE architecture allows us to perform amortized inference (details in the next section). We take a deeper look at latent variable models in the following sections.

2.2. Gaussian Mixture Models

A simple Latent Variable Model is the Gaussian Mixture Model (GMM). GMMs are useful for modeling data that come from one of several groups, with each group being modeled by a Gaussian distribution. It is used for data clustering and density estimation. The model assumes data generated for a mixture of K gaussians with mixing proportion $\boldsymbol{\pi} = [\,\pi_1 \ldots \pi_K\,]$. Intuitively, each $\pi_k \in (0,1)$ represents the fraction of data contributed by the k^{th} Gaussian.

In this model, we define the latent variable to be the cluster assignment of each data point, *i.e.* z represents the index of the cluster the corresponding data point belongs to with a categorical/discrete prior. Therefore, the mixing proportion can now be formally defined as the weights of the categorical distribution representing the prior of the latent variable z. Hence, we can write

$$\begin{aligned} \mathbb{P}\left[\,z = k\,\middle|\,\boldsymbol{\pi}\,\right] &= \pi_k \\ \implies \mathbb{P}\left[\,z\,\middle|\,\boldsymbol{\pi}\,\right] &= \prod_{k=1}^{K} (\pi_k)^{\mathcal{I}\,(z=k)} \end{aligned}$$

Since we now have a cluster assignment, the next step is to define the probability distribution we use for sampling each data point. As pointed out earlier, this probability distribution is given by a gaussian.

$$\mathbb{P}\left[\,\mathbf{x}\,\middle|\,z = k\,\right] = \mathcal{N}\left(\mathbf{x}\,\middle|\,\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k^{-1}\right)$$

Since doing straight-forward MLE is not possible for this model (due to the presence of latent-variables), we estimate the parameters $\{\boldsymbol{\mu}_k, \Sigma_k\}_{k=1}^K$ using Expectation Maximization(EM) Algorithm(**?**), which is a popular strategy for simple latent-variable models.

The problem with GMM is the assumed Gaussian structures for each cluster. As pointed out earlier, this limits the power of GMM therefore not allowing arbitrary clustered data to be modeled using such models. We therefore need to explore other latent variable models with higher reprentational/expressive power. One such model is the Variational Deep Embeddings (VaDE) Model (**?**), based on the Variational Autoencoder architecture. We discuss this model in the next section.

2.3. Variational Autoencoders

Variational Autoencoders were initially proposed by Kingma and Welling (2013). The key idea behind VAEs is that any distribution in D dimensions can be generated by taking a set of D variables that are normally distributed and mapping them through a sufficiently complicated function (Doersch, 2016). Hence, provided powerful function approximators, we can simply learn a function which maps independent, normally distributed variables to the latent variables needed for our model, and these latent variables are then used to model the observations $\mathbf{X}$, or rather the data distribution $\mathcal{D}$ not known to us, from which the observations are assumed to be sampled.

The task, therfore, is to find a strong non-linear mapping to approximate the latent variables needed to model the data distribution $\mathcal{D}$. This mapping, in VAEs, is modelled using a neural network, and the mapping from these latent variables to the observations is through a distribution (generally exponential family) depending on the observed space $\mathcal{X}$. For example, if $\mathcal{X} \in \mathbb{R}$, then the estimated probability distribution for $\mathbf{x} \sim \mathcal{D}$ can be given as

$$\mathbb{P}\left[\mathbf{x} \,|\, \mathbf{z}\right] \quad = \quad \mathcal{N}\left(\mathbf{x} \,|\, f(\mathbf{z}; \boldsymbol{\theta}), \sigma^2 \mathbf{I}\right)$$

for some non-linear mapping $f : \mathbb{R} \to \mathbb{R}$ (in this case) modelled using a neural network.

The objective now is to compute the posterior over $\mathbf{z}$ given the observation $\mathbf{x}$. This would allow us to model the predictive posterior using samples of $\mathbf{z}$ from this posterior. Since the posterior $\mathbb{P}\left[\mathbf{z} \,|\, \mathcal{D}\right]$ is intractable, we approximate it using a proposal distribution $\mathcal{Q}\left(\mathbf{z} \,|\, \mathbf{x}, \boldsymbol{\phi}\right)$. We use the standard Black Box Variational Inference strategy to estimate the posterior, by maximizing the ELBO bound using descent methods, which is given as

$$\begin{aligned}
\mathcal{L}(\mathbf{x}; \mathcal{Q}) \quad &= \quad \log\left(\mathbb{P}\left[\mathbf{x}\right]\right) - \mathrm{KL}\left[\mathcal{Q}\left(\mathbf{z}; \boldsymbol{\phi}\right) \;||\; \mathbb{P}\left[\mathbf{z} \,|\, \mathbf{x}\right]\right] \\
&= \quad \mathop{\mathbb{E}}_{\mathbf{z} \sim \mathcal{Q}(\mathbf{z} \,|\, \mathbf{x}, \boldsymbol{\pi})} \left[\log\left(\frac{\mathbb{P}\left[\mathbf{x}, \mathbf{z}\right]}{\mathcal{Q}\left(\mathbf{z} \,|\, \mathbf{x}, \boldsymbol{\phi}\right)}\right)\right]
\end{aligned}$$

Typically, the prior over the latent variables $\mathbf{z}$ is assumed to be a multivariate normal distribution with mean $\mathbf{0}$ and variance matrix $\mathbf{I}$.

This structure is similar to that of a standard autoencoder, where the network which maps the inputs to the parameters of the proposal is the encoder and the network mapping the sampled $\mathbf{z}$'s to the parameters of the conditional likelihood distribution is the decoder.

One major problem with Variational Inference is that it models the parameters of each latent variable independently, and therefore we need an iterative procedure to model the proposal with respect to each variable. That is, if we want to add a new data point, we need to remodel the proposal in order to compute the pdf of the new latent variable. VAE tackles this problem by assuming that the latent

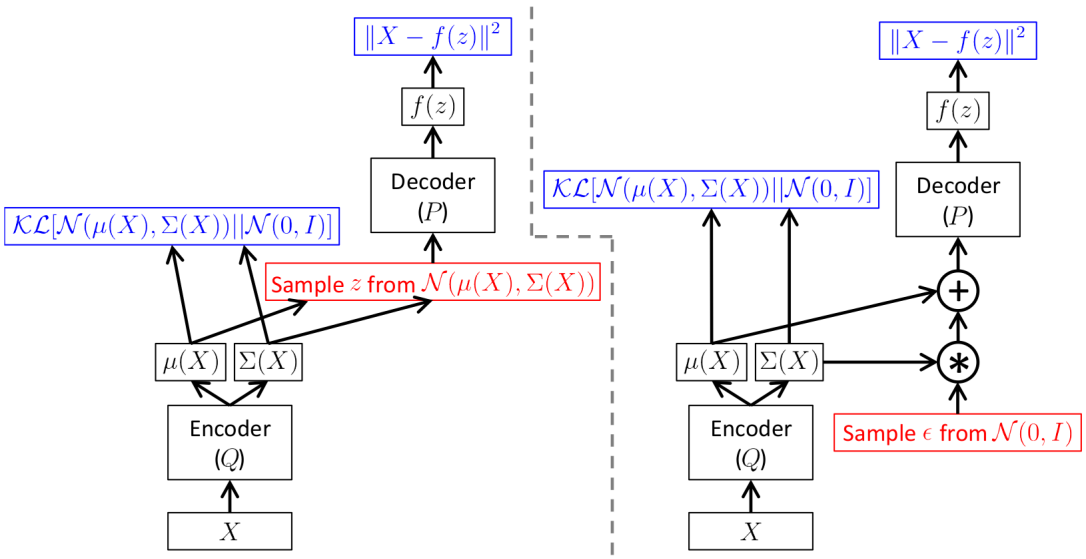

Figure 1: Without (left) and with (right) the reparametrization trick

variable is dependent on the input, and the mapping from the input to the parameters of the proposal distribution is modeled using a neural network. This, essentially, is the idea of amortized inference.

Since looking at the ELBO, one can say we wish to minimize the KL Divergence between the proposal and the prior for $\mathbf{z}$ which is a standard Normal distribution, we can assume the proposal to be of the form $\mathcal{N}\left(\mathbf{z} \,\middle|\, \boldsymbol{\mu}(\mathbf{x}; \boldsymbol{\phi}), \boldsymbol{\sigma}^2(\mathbf{x}; \phi)\mathbf{I}\right)$. The negative ELBO is then considered as our loss, and the parameters (or weights) of the Neural Networks are inferred.

However, since we are assuming the variables $\mathbf{z}$ to be sampled from the proposal, we cannot apply back propogation across the decoder and the encoder networks. This is handled using the reparameterization trick, where we write $\mathbf{z} = \boldsymbol{\epsilon} \cdot \boldsymbol{\sigma}^2(\mathbf{x}; \boldsymbol{\phi}) + \boldsymbol{\mu}(\mathbf{x}; \boldsymbol{\phi})$. This allows us to input the reparametrized latent variables $\boldsymbol{\epsilon}$, and therefore facilitate backpropogation through the stochastic nodes. The working of a standard VAE is visuallized in figure 1 [1].

3. Variational Deep Embeddings

Variational Deep Embedding (VaDE) was proposed by **?**. It is a simplistic model than the previous models discussed, and greatly simplifies the generic generative story by assuming a uniform prior on the cluster means, variances, and misture proportions. Also, it assumes a uniform prior on the model parameters $\boldsymbol{\theta}$.

Depending on whether the observations $\mathbf{x}$ is binary or real-valued we compute $\mu_x = f(\mathbf{z}, \boldsymbol{\theta})$ and choose a sample $\mathbf{x} \sim Ber(\mu_x)$ or compute $[\mu_x; \log(\sigma)_x^2] = f(\mathbf{z}, \boldsymbol{\theta})$ and choose a sample $\mathbf{x} \sim \mathcal{N}(\mu_x, \sigma_x^2 \mathbf{I})$. From the generative story, the joint probability can be written as

$$\mathbb{P}[\mathbf{x}, \mathbf{z}, c] \quad = \quad \mathbb{P}\left[\mathbf{x} \,\middle|\, \mathbf{z}, \boldsymbol{\theta}\right] \mathbb{P}\left[\mathbf{z} \,\middle|\, c\right] \mathbb{P}[c]$$

An instance of VaDE is tuned to maximize the likelihood of the given data points. Given the generative

[1] Image taken from Doersch (2016)

Algorithm 1: Generative Story

The generative story for a mixture madel based on latent variables can be written as

$$\begin{aligned} c &\sim \text{Cat}\left(\boldsymbol{\pi}\right) \\ \mathbf{z} \,|\, c = k &\sim \mathcal{N}\left(\mathbf{z} \,|\, \boldsymbol{\mu}_k, \Sigma_k\right) \\ \mathbf{x} \,|\, \mathbf{z} &\sim p\left(\mathbf{x} \,|\, f\left(\mathbf{z}; \boldsymbol{\phi}\right), f\left(\mathbf{z}; \boldsymbol{\phi}\right)\right) \end{aligned}$$

In our case, we model the function f using a neural network. The probability distribution p can be any distribution which is apt for the data space. For example, if the data space is real, *i.e.* $\mathbf{x} \in \mathbb{R}$, then the probability distribution can be a Gaussian, with the function f feeding the mean and the variance of the gaussian.

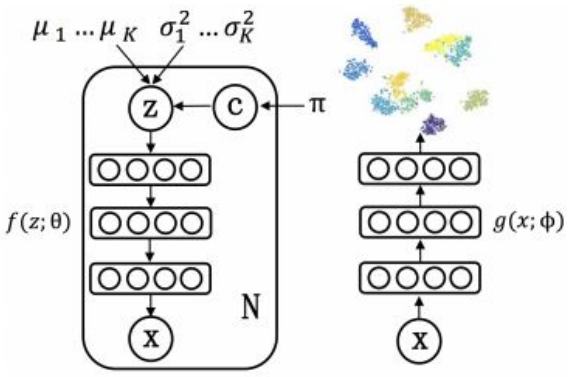

Figure 2: Plate notation for VaDE (left) and an encoder network $g(\mathbf{x}; \phi)(right)$

process described above we have

$$\begin{aligned} \log \mathbb{P}\left[\,\mathbf{x}\,\right] &= \log\left(\int_{\mathbf{z}} \sum_{c} \mathbb{P}\left[\,\mathbf{x}, \mathbf{z}, c\,\right] \,\mathrm{d}\mathbf{z}\right) \\ &\geq \mathop{\mathbb{E}}_{\mathcal{Q}(\mathbf{z}, c \,|\, \mathbf{x})}\left[\log \frac{\mathbb{P}\left[\,\mathbf{x}, \mathbf{z}, c\,\right]}{\mathcal{Q}\left(\mathbf{z}, c \,|\, \mathbf{x}\right)}\right] = \mathcal{L}_{\text{ELBO}} \end{aligned}$$

where $\mathcal{Q}\left(\mathbf{z}, c \,|\, \mathbf{x}\right)$ is the proposal distribution.

Mean field is assumed on the proposal distribution, giving:

$$\mathcal{Q}\left(\mathbf{z}, c \,|\, \mathbf{x}\right) = \mathcal{Q}\left(\mathbf{z} \,|\, \mathbf{x}\right) \mathcal{Q}\left(c \,|\, \mathbf{x}\right)$$

Similar to VAE, the distribution $\mathcal{Q}\left(\mathbf{z} \,|\, \mathbf{x}\right)$ is modelled using a neural network g as follows

$$\begin{aligned} \tilde{\boldsymbol{\mu}}, \log \tilde{\boldsymbol{\sigma}}^2 &= g(\mathbf{x}, \boldsymbol{\phi}) \\ \mathcal{Q}\left(\mathbf{z} \,|\, \mathbf{x}\right) &= \mathcal{N}\left(\mathbf{z} \,|\, \tilde{\mu}, \tilde{\boldsymbol{\sigma}}^2 \mathbf{I}\right). \end{aligned}$$

Further, a decoder model is added giving the following:

$$\mathbb{P}\left[\,\mathbf{x}\,\middle|\,\mathbf{z}\,\right] \quad = \quad \mathbb{P}\left[\,\mathbf{x}\,\middle|\,f(\mathbf{z},\boldsymbol{\theta})\,\right]$$

The authors use an interesting approach to "approximating" the proposal distribution $\mathcal{Q}\left(c\,\middle|\,\mathbf{x}\right)$, which although looks absurd, in practice works surprisingly well. First, one can realize, the ELBO can be re-written in the following manner,

$$\begin{aligned}
\mathcal{L}_{\text{ELBO}} &= \mathop{\mathbb{E}}_{\mathcal{Q}(\mathbf{z},c\,|\,\mathbf{x})}\left[\log\frac{\mathbb{P}\left[\,\mathbf{x},\mathbf{z},c\,\right]}{\mathcal{Q}\left(\mathbf{z},c\,\middle|\,\mathbf{x}\right)}\right] \\
&= \int_{\mathbf{z}}\sum_{c}\mathcal{Q}\left(c\,\middle|\,\mathbf{x}\right)\mathcal{Q}\left(\mathbf{z}\,\middle|\,\mathbf{x}\right)\left\{\log\frac{\mathbb{P}\left[\,\mathbf{x}\,\middle|\,\mathbf{z}\,\right]\mathbb{P}\left[\,\mathbf{z}\,\right]}{\mathcal{Q}\left(\mathbf{z}\,\middle|\,\mathbf{x}\right)}+\log\frac{\mathbb{P}\left[\,c\,\middle|\,\mathbf{z}\,\right]}{\mathcal{Q}\left(c\,\middle|\,\mathbf{x}\right)}\right\} \\
&= \int_{\mathbf{z}}\mathcal{Q}\left(\mathbf{z}\,\middle|\,\mathbf{x}\right)\log\frac{\mathbb{P}\left[\,\mathbf{x}\,\middle|\,\mathbf{z}\,\right]\mathbb{P}\left[\,\mathbf{z}\,\right]}{\mathcal{Q}\left(\mathbf{z}\,\middle|\,\mathbf{x}\right)}\,\mathrm{d}\mathbf{z}-\int_{\mathbf{z}}\mathcal{Q}\left(\mathbf{z}\,\middle|\,\mathbf{x}\right)\mathrm{KL}\left(\mathcal{Q}\left(c\,\middle|\,\mathbf{x}\right)\,||\,\mathbb{P}\left[\,c\,\middle|\,\mathbf{z}\,\right]\right)\,\mathrm{d}\mathbf{z}
\end{aligned}$$

In order to maximize the above ELBO, it seems we need to minimize the average KL Divergence over the latent variables, when sampled using the posterior. *Jiang et al.* approximated this simply using only one sample of $\mathbf{z}$ and therefore, we can write

$$\mathcal{Q}\left(c\,\middle|\,\mathbf{x}\right) \quad = \quad \mathbb{P}\left[\,c\,\middle|\,\mathbf{z}=\hat{\mathbf{z}}\,\right] \quad = \quad \frac{\mathbb{P}\left[\,c\,\right]\mathbb{P}\left[\,\mathbf{z}=\hat{\mathbf{z}}\,\middle|\,c\,\right]}{\sum_{c'=1}^{K}\mathbb{P}\left[\,c'\,\right]\mathbb{P}\left[\,\mathbf{z}=\hat{\mathbf{z}}\,\middle|\,c'\,\right]}$$

where $\hat{\mathbf{z}}$ is a sample from the posterior proposal $\mathcal{Q}\left(\mathbf{z}\,\middle|\,\mathbf{x}\right)$.

Given the simplicity of the model, we performed some experiments on it. We implemented the complete model on Tensorflow from scratch, and performed experiments on a couple of datasets. This is discussed in detail in the following section.

4. Mixture of Experts (ME)

Just like GMM, a mixture of experts model splits the data into soft clusters, and for each cluster, we have an expert which predicts the target labels for each mixture. Therefore, we need to define a *gating function*, which defines the mixing proportions for each data point. Suppose we denote the mixture assignment using a latent variable c, then we have a probability density/mass function $\mathbb{P}\left[\,\mathbf{y}\,\middle|\,\mathbf{x},c\,\right]$. Mostly this probability is defined using a pdf function belonging to the class of exponential distributions. For example, if the space of the target label $\mathcal{Y}=\{0,1\}$, then we can define $\mathbb{P}\left[\,\mathbf{y}\,\middle|\,\mathbf{x},c\,\right]$ to be a Bernoulli distribution.

We still need to define a mixing proportion. Since our ultimate goal is to model $\mathbb{P}\left[\,\mathbf{y}\,\middle|\,\mathbf{x}\,\right]$, we write

$$\begin{aligned}
\mathbb{P}\left[\,\mathbf{y}\,\middle|\,\mathbf{x}\,\right] &= \sum_{k=1}^{K}\mathbb{P}\left[\,\mathbf{y},c=k\,\middle|\,\mathbf{x}\,\right] \\
&= \sum_{k=1}^{K}\mathbb{P}\left[\,c\,\middle|\,\mathbf{x}\,\right]\cdot\mathbb{P}\left[\,\mathbf{y}\,\middle|\,c=k,\mathbf{x}\,\right]
\end{aligned}$$

Looking at the above equation, we can say setting $\mathbb{P}\left[\, c \,|\, \mathbf{x} \,\right]$ to be the mixing proportion as a natural choice. The most common choice for modelling this probability mass function is using a softmax distribution as follows -

$$\mathbb{P}\left[\, c = k \,|\, \mathbf{x} \,\right] \quad = \quad \frac{\exp\left(b_k + \langle \mathbf{w}_k,\ \mathbf{x} \rangle\right)}{\sum_{k'=1}^{K} \exp\left(b_{k'} + \langle \mathbf{w}_{k'},\ \mathbf{x} \rangle\right)}$$

where K is the number of experts.

We usually represent $\mathbb{P}\left[\, c = k \,|\, \mathbf{x} \,\right]$ using the function $g_k(\mathbf{x}, \mathbf{\Theta}_g)$. To sum up ME models, $g_k(\mathbf{x}, \mathbf{\Theta}_g)$ is essentially the gate's rating for the k^{th} expert given $\mathbf{x}$ and $\mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x}, c, \mathbf{\Theta}_e \,\right]$ is the probability of the k^{th} expert generating $\mathbf{y}$ given $\mathbf{x}$. For brevity, we denote this as $p_i(\mathbf{y})$.

In general, the ME training algorithm maximizes the log-likelihood $\mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x} \,\right]$ to learn the parameters of the expert and the gate. During the training ME, the gate and the experts gets decoupled, therefore the model attains a modular structure. This property is exploited in order to extend ME models to hierarchical mixture of experts (HME). A standard ME model is graphically represented in figure 3 [2].

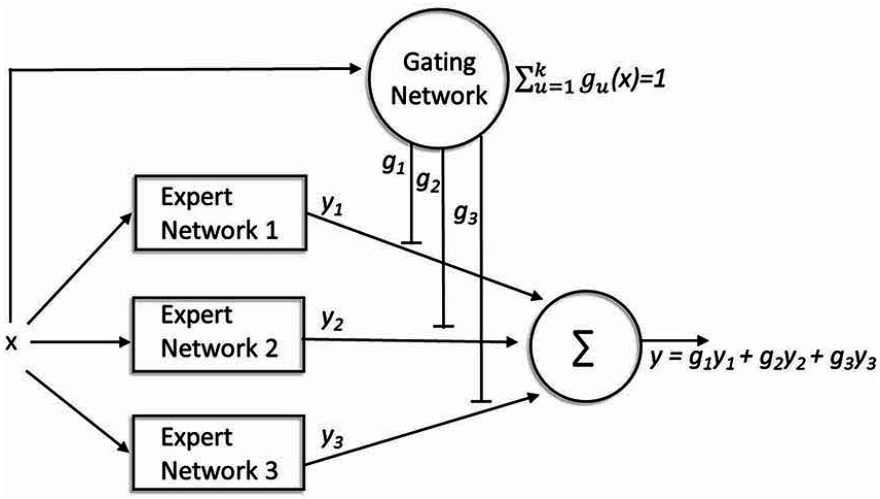

Figure 3: Graphical representaion of a standard Mixture of Experts model

4.1. ME Regression Model

Let $\mathbf{\Theta}_e = \{\mathrm{W}_k, \Gamma_k\}_{k=1}^{K}$ denote the parameters of the experts. As mentioned earlier, in the original ME regression model, the experts follow the gaussian model -

$$\mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x}, c = k \,\right] \quad = \quad \mathcal{N}\left(\mathbf{y} \,|\, \hat{\mathbf{y}}(\mathbf{x}, \mathrm{W}_k), \Gamma_k\right)$$

where $\mathbf{y} \in \mathbb{R}^{\mathrm{M}}$. The term $\hat{\mathbf{y}}(\mathbf{x}, \mathrm{W})$ is defined as $\hat{\mathbf{y}}(\mathbf{x}, \mathrm{W}) = \mathrm{W}\mathbf{x}$. We can also add a bias term to $\mathrm{W}\mathbf{x}$.

At test time, in order to make a prediction, the expectation of $\mathbf{y}$ with respect to the pdf $\mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x}, \mathbf{\Theta} \,\right]$ is used as the output of the architecture. This is, therefore, given by

$$\hat{\mathbf{y}} = \sum_{k=1}^{K} g_k(\mathbf{x}, \mathbf{\Theta}_g) \cdot \hat{\mathbf{y}}(\mathbf{x}, \mathrm{W})$$

[2] Image taken from **?**

4.2. ME Classification Model

In this case, the probability of $\mathbf{y}$ is given by a categorical/discrete distribution. In the one hot representation, the desired output $\mathbf{y}$ is of length L and $y_l = 1$ if $\mathbf{x}$ belongs to class l and 0 otherwise. Also, expert k has parameters $\{\mathbf{w}_{imk}\}_{k=1}^{K}$, corresponding to the parameters of each class. The probability given by each expert is therefore written as

$$\begin{aligned}\mathbb{P}\left[\,\mathbf{y}\,\middle|\,\mathbf{x}, c=k\,\right] &= \prod_{l=1}^{L}\hat{\mathbf{y}}_l(\mathbf{x}, b_k, \mathbf{w}_k)^{\mathcal{I}(y_l=1)} \\ \hat{\mathbf{y}}_l &= \frac{\exp\left(\langle \mathbf{w}_{ik},\ \mathbf{x}\rangle\right)}{\sum_{l'=1}^{L}\exp\left(\langle \mathbf{w}_k,\ \mathbf{x}\rangle\right)}\end{aligned}$$

The prediction is done in the same way as was shown for the ME Regression Model.

Although the paradigm of Mixture of Experts is very generic, not many methods exist for the same. The fundamental problems are the difficulties of inference, along with non-compatibility with automatic differentiation libraries for the existing inference methods. We try to remedy this problem by proposing a novel Mixture of Experts model based on Variational Autoencoders. In the next section, we show a way to extend the VaDE model for use as a gating function for ME models, and point out some difficulties in doing the same. Following that, we propose a novel gating function based on Discrete Latent Variables in Variational Autoencoders, and later show that it outperforms other naive gating functions.

5. VaDE with Mixture of Experts

The VaDE model originally described in **?** is used to learn cluster assignments and also works as a generative model. For using VaDE in classification/regression tasks, the original theory can be extended in a straight-forward manner by maximizing an appropriate loss function. This loss function needs to include the predicted label ($\mathbf{y}$) along with reconstruction of $\mathbf{x}$. Although general learning models would maximise the log-likelihood *i.e.* $\mathbb{P}\left[\,\mathbf{y}\,\middle|\,\mathbf{x}\,\right]$, we are aiming to build a generative model and therefore our maximization objective will be $\mathbb{P}\left[\,\mathbf{x}, \mathbf{y}\,\right]$.

$$\begin{aligned}\log \mathbb{P}\left[\,\mathbf{x}, \mathbf{y}\,\right] &= \log\left(\int_{\mathbf{z}}\sum_{c}\mathbb{P}\left[\,\mathbf{x}, \mathbf{y}, \mathbf{z}, c\,\right]\right) \\ &\geq \mathop{\mathbb{E}}_{\mathcal{Q}(\mathbf{z},c\,|\,\mathbf{x},\mathbf{y})}\left[\log\frac{\mathbb{P}\left[\,\mathbf{x}, \mathbf{y}, \mathbf{z}, c\,\right]}{\mathcal{Q}\left(\mathbf{z}, c\,\middle|\,\mathbf{x}, \mathbf{y}\right)}\right] \\ &= \mathop{\mathbb{E}}_{\mathcal{Q}(\mathbf{z},c\,|\,\mathbf{x},\mathbf{y})}\left[\log\mathbb{P}\left[\,\mathbf{y}\,\middle|\,\mathbf{x}, \mathbf{z}, c\,\right]\right] + \mathop{\mathbb{E}}_{\mathcal{Q}(\mathbf{z},c\,|\,\mathbf{x},\mathbf{y})}\left[\log\frac{\mathbb{P}\left[\,\mathbf{x}, \mathbf{z}, c\,\right]}{\mathcal{Q}\left(\mathbf{z}, c\,\middle|\,\mathbf{x}, \mathbf{y}\right)}\right] \\ &= \mathcal{L}_{\text{ELBO}}\end{aligned}$$

Note that the second part of this formulation closely resembles the original $\mathcal{L}_{\text{ELBO}}$, except for an additional $\mathbf{y}$ term in the proposal distribution. The proposal distribution $\mathcal{Q}$ can be modelled as follows:

$$\mathcal{Q}\left(\mathbf{z}, c\,\middle|\,\mathbf{x}, \mathbf{y}\right) = \mathcal{Q}\left(c\,\middle|\,\mathbf{x}, \mathbf{y}\right)\mathcal{Q}\left(\mathbf{z}\,\middle|\,\mathbf{x}, \mathbf{y}, c\right)$$

Again, assuming Mean Field approximation, we can write $\mathcal{Q}\left(\mathbf{z}\,\middle|\,\mathbf{x}, \mathbf{y}, c\right) = \mathcal{Q}\left(\mathbf{z}\,\middle|\,\mathbf{x}, \mathbf{y}\right)$. Further, the latent representation $\mathbf{z}$ can be assumed to be independent of $\mathbf{y}$ conditioned on $\mathbf{x}$. So, we can further simplify $\mathcal{Q}\left(\mathbf{z}\,\middle|\,\mathbf{x}, \mathbf{y}\right) = \mathcal{Q}\left(\mathbf{z}\,\middle|\,\mathbf{x}\right)$

The estimate for $\mathcal{Q}\left(c \,|\, \mathbf{x}, \mathbf{y}\right)$ remains exactly the same as that of $\mathcal{Q}\left(c \,|\, \mathbf{x}\right)$ in the original VaDE model, and is approximated as:

$$\mathcal{Q}\left(c \,|\, \mathbf{x}, \mathbf{y}\right) \quad = \quad \mathbb{P}\left[\, c \,|\, \mathbf{z} = \hat{\mathbf{z}} \,\right] \quad = \quad \frac{\mathbb{P}\left[\, c \,\right] \mathbb{P}\left[\, \mathbf{z} = \hat{\mathbf{z}} \,|\, c \,\right]}{\sum_{c'=1}^{K} \mathbb{P}\left[\, c' \,\right] \mathbb{P}\left[\, \mathbf{z} = \hat{\mathbf{z}} \,|\, c' \,\right]}$$

Therefore, the second part remains exactly same as that of the original VaDE model. The first part of $\mathcal{L}_{\text{ELBO}}$ is the familiar supervised learning term. Since $\mathbf{y}$ can be assumed to depend only on $\mathbf{x}$ and c, we have:

$$\mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x}, \mathbf{z}, c \,\right] = \mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x}, c \,\right]$$

Depending on the task at hand, $\mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x}, c \,\right]$ can be modelled in various ways. For a regression task, we can use a linear model $\{\mathbf{W}_c, \mathbf{b}_c\}_{c=1}^{K}$, so that maximizing $\mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x}, c \,\right]$ is equivalent to minimizing $\left(\hat{\mathbf{y}} - (\mathbf{W}_c \mathbf{x} + \mathbf{b}_c)\right)^2$

5.1. Test Time Approximations

During test time, we have to estimate $\mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x} \,\right]$. We can wirte this as follows:

$$\begin{aligned}
\mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x} \,\right] &= \sum_{c} \mathbb{P}\left[\, c \,|\, \mathbf{x} \,\right] \mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x}, c \,\right] \\
&= \sum_{c} \left(\int_{\mathbf{z}} \mathbb{P}\left[\, c \,|\, \mathbf{x}, \mathbf{z} \,\right] \mathbb{P}\left[\, \mathbf{z} \,|\, \mathbf{x} \,\right] \right) \mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x}, c \,\right] \\
&= \sum_{c} \left(\mathop{\mathbb{E}}_{\mathbb{P}\left[\, \mathbf{z} \,|\, \mathbf{x} \,\right]} \left[\mathbb{P}\left[\, c \,|\, \mathbf{z} \,\right] \right] \right) \mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x}, c \,\right]
\end{aligned}$$

We again assume that c is independent of $\mathbf{x}$ conditioned on $\mathbf{z}$. Further, we can use monte carlo estimates to compute the expectation, and $\mathbb{P}\left[\, c \,|\, \mathbf{z} \,\right]$ is estimated using the same approximation as that used during inference.

5.2. Shortcomings of VaDE model

A major shortcoming of the VaDE model is using a single sample to approximate $\mathcal{Q}\left(c \,|\, \mathbf{x}, \mathbf{y}\right)$. The requirement for zeroing out the KL divergence is that $\mathcal{Q}\left(c \,|\, \mathbf{x}, \mathbf{y}\right)$ is exactly equal to $\mathbb{P}\left[\, c \,|\, \mathbf{z} \,\right]$ at all points, which is not true in general. However, using a single $\hat{\mathbf{z}}$ introduces deviations from actual value which leads to slow inference.

Also, during test time, we need to approximate $\mathbb{E}_{\mathbb{P}[\mathbf{z} \,|\, \mathbf{x}]} \mathbb{P}\left[\, c \,|\, \mathbf{z} \,\right]$ which uses the same approximation along with monte carlo estimates which are known to have high variance. Hence, this may lead to inaccuracies during test time.

6. Discrete Mixture Variational Autoencoders

As pointed out in the previous section, a gating function based on VaDE will suffer from two prime problems, slow inference and slow and inaccurate predictions. We try to design a gating function keeping these two problems in mind.

We follow the same generative story as outlined in algorithm 1, and use the same mean-field assumption as VaDE, *i.e.* $\mathcal{Q}\left(\mathbf{z}, c \,|\, \mathbf{x}\right) = \mathcal{Q}\left(\mathbf{z} \,|\, \mathbf{x}\right) q(c \,|\, \mathbf{x})$. Therefore, we can write the ELBO in the same way oulined in VaDE.

In order to tackle the problems in VaDE, we propose to add a separate encoder network for inferring the latent variable c, *i.e.* the cluster assignment. The sole problem in the formulation is the discontinuous nature of the latent variable c. Since the variable is a categorical one, it is not possible to find a deterministic reparametrization formulation, we cannot use backpropogation over this stochastic node. However, there exist methods which facilitate black box variational inference with binary/categorical stochastic nodes.

The most popular strategy to facilitate BBVI, and therefore backpropogation over discrete nodes is Gumbel-Softmax. Briefly, the gumbel-softmax trick relaxes the discrete variables to continuous ones, and allow backpropogation through an appropriate reparametrization function. The details of this trick are mentioned in the following section.

6.1. Gumbel Softmax Trick

As mentioned, we need to find a smooth relaxation for discrete latent variables, along with an apt reparametrization variable to allow backpropogation, and therefore inference using black box variational inference for a model.

Firstly, we need to look at how we can sample from a categorical (consider single dimension) variable. The typical way of doing so would be to sample a uniform random variable (from 0 to 1), U and return the categorical variable as follows

$$\mathrm{C} \quad = \quad \underset{k \in [\, K \,]}{\arg\min} \; \mathrm{U} - \sum_{k'=1}^{k} \pi_{k'}$$

Note. The parameters $\{\pi_k\}_{k=1}^K$ denote the probabilities of each class for the categorical distribution, and therefore $\sum_{k=1}^K \pi_k = 1$. Below, we alternately use weights $\{\alpha_k\}_{k=1}^K$ instead of probabilities, and therefore, we will have $\pi_k = \frac{\alpha_k}{\sum_{k'=1}^K \alpha_{k'}}$

An alternate way, however, to sample from the categorical distribution is by using the Gumbel-Max trick (upon which the Gumbel-Softmax is based). The Gumbel-Max trick says that in order to sample from a categorical distribution with weights $\{\alpha_k\}_{k=1}^K$, we sample K uniform random variables $\{\mathrm{U}_k\}_{k=1}^K$ and reparametrize as follows

$$\mathrm{C} \quad = \quad \underset{k \in [\, K \,]}{\arg\max} \; \log\left(\alpha_k\right) - \log\left(-\log\left(\mathrm{U}_k\right)\right)$$

The variable $-\log\left(-\log\left(U\right)\right)$ is known to be from the Gumbel disitribution. This alternate sampling strategy allows us to write the relaxed version of the random variable C (denoted by $\mathbf{Z} = [\,\mathrm{Z}_1 \ldots \mathrm{Z}_K\,]$). which is then given as

$$\mathrm{Z}_k \quad = \quad \frac{\exp\left(\left(\log\left(\alpha_k\right) + \mathrm{G}_k\right)/\tau\right)}{\sum_{k'=1}^K \exp\left(\left(\log\left(\alpha_{k'}\right) + \mathrm{G}_{k'}\right)/\tau\right)}$$

where G_k is a sample from the gumbel distribution, and τ is a parameter, known as the temperature. It is obvious to see that as $\tau \to 0$, the smooth relaxation $Z_k \to \mathcal{I}\ (C = k)$.

Following other details from the Concrete paper, we are able to train VAEs with (relaxed) discrete latent variables. This, now. allows us to model the posterior for the latent variable c using an encoder network.

As we have an estimate of the posterior of all the latent variables, we do not need an approximation to compute $\mathcal{Q}\left(c \,|\, \mathbf{x}\right)$ as was the case in VaDE. The remaining model remains the same, with K means and variances as the prior parameters, and the same KL computation as in VaDE. One difference, however, we have from VaDE is that we have a uniform prior on the category probabilities, *i.e.* , in the prior for c, we assume each cluster to have equal probability. This can, however, be easily removed as it does not change the computations at any point. We call this model Discrete Mixture Variational Autoencoder (DMVAE).

Modelling the posterior of the latent variable C allows faster inference possible. We document the difference in inference time and results between DMVAE and VaDE in section 8. Our hypothesis is that this occurs due to the added noise in approximating $\mathcal{Q}\left(c \,|\, \mathbf{x}\right)$ using a single sample. If we take a look at the expanded ELBO term in the VaDE paper (for normal distribution over $\mathbf{x} \,|\, \mathbf{z}$), it shows a similar problem Black Box Variational Inference faces without reparametrization. Due to the shortage of time, we could not emperically confirm this hypothesis but the faster inference for DMVAE endorses this hypothesis.

This formulation of DMVAE easily extends to Mixture of Experts models with the DMVAE model as a variational gating function. We look at the details of the same in the next section.

7. Mixture of Experts with DMVAE

Since we do not alter the general formulation of the ME models, we can still write the basic equations. That is, for some latent variable C denoting the cluster/mixture assignment, we have

$$\mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x} \,\right] \quad = \quad \sum_{k=1}^{K} \mathbb{P}\left[\, c = k \,|\, \mathbf{x} \,\right] \mathbb{P}\left[\, \mathbf{y} \,|\, \mathbf{x}, c = k \,\right]$$

As discussed in earlier sections, we can model $\mathbf{y} \,|\, \mathbf{x}, c$ using an appropriate distribution (commonly exponential family). The posterior probabily for cluster assignment. in case of VaDE, caused problems during test time, as discussed. In case of DMVAE, we avoid this problem by approximating the posterior using a neural network (encoder network). This, therefore, saves time during prediction by avoiding sampling of latent variable Z.

Moreover, we still use the mean field assumption, same as VaDE. However, for the case of ME with DMVAE, we estimate the posterior $\mathbb{P}\left[\, \mathbf{z}, c \,|\, \mathbf{x}, \mathbf{y} \,\right]$ using $\mathcal{Q}\left(\mathbf{z}, c \,|\, \mathbf{x}, \mathbf{y}\right)$ having the following assumptions

$$\begin{aligned} \mathcal{Q}\left(\mathbf{z}, c \,|\, \mathbf{x}, \mathbf{y}\right) &= \mathcal{Q}\left(\mathbf{z}, c \,|\, \mathbf{x}\right) \\ &= \mathcal{Q}\left(\mathbf{z} \,|\, \mathbf{x}\right) \mathcal{Q}\left(c \,|\, \mathbf{x}\right) \end{aligned}$$

We can now write the loss function. Similar to VaDE based ME, we need to maximize the probability $\mathbb{P}\left[\, \mathbf{x}, \mathbf{y} \,\right]$. However, since we now can compute the exact (estimate of) posterior $\mathcal{Q}\left(c \,|\, \mathbf{x}\right)$, we can alter the lower bound to be tighter than the ELBO term used for ME with VaDE. This also has an additional benifit

highlighted later in this section. This altered lower bound can be written as

$$\begin{aligned}
\log \mathbb{P}\left[\,\mathbf{y}, \mathbf{x}\,\right] &= \log \mathbb{P}\left[\,\mathbf{y} \,\middle|\, \mathbf{x}\,\right] + \log \mathbb{P}\left[\,\mathbf{x}\,\right] \\
&\geq \log \mathbb{P}\left[\,\mathbf{y} \,\middle|\, \mathbf{x}\,\right] + \mathop{\mathbb{E}}_{\mathbf{z},c \,|\, \mathbf{x}}\left[\log\left(\frac{\mathbb{P}\left[\,\mathbf{z}, c, \mathbf{x}\,\right]}{\mathcal{Q}\left(\mathbf{z}, c \,\middle|\, \mathbf{x}\right)}\right)\right] \\
&= \log \mathbb{P}\left[\,\mathbf{y} \,\middle|\, \mathbf{x}\,\right] + \mathop{\mathbb{E}}_{\mathbf{z},c \,|\, \mathbf{x}}\left[\log \mathbb{P}\left[\,\mathbf{x} \,\middle|\, \mathbf{z}, c\,\right] + \log\left(\frac{\mathbb{P}\left[\,\mathbf{z} \,\middle|\, c\,\right]}{\mathcal{Q}\left(\mathbf{z} \,\middle|\, \mathbf{x}\right)}\right) + \log\left(\frac{\mathbb{P}\left[\,c\,\right]}{\mathcal{Q}\left(c \,\middle|\, \mathbf{x}\right)}\right)\right] \\
\mathcal{L}(\mathbf{x}, \mathbf{y}) &= \log \mathbb{P}\left[\,\mathbf{y} \,\middle|\, \mathbf{x}\,\right] + \mathop{\mathbb{E}}_{\mathbf{z} \,|\, \mathbf{x}}\left[\log \mathbb{P}\left[\,\mathbf{x} \,\middle|\, \mathbf{z}\,\right]\right] - \mathop{\mathbb{E}}_{c \,|\, \mathbf{x}}\left[\mathrm{KL}\left(\mathcal{Q}\left(\mathbf{z} \,\middle|\, \mathbf{x}\right) \,||\, \mathbb{P}\left[\,\mathbf{z} \,\middle|\, c\,\right]\right)\right] \\
&\quad - \mathrm{KL}\left(\mathcal{Q}\left(c \,\middle|\, \mathbf{x}\right) \,||\, \mathbb{P}\left[\,c\,\right]\right)
\end{aligned}$$

Each term can be intuitively understood, with the first term maximizing the likelihood, the second term maximizing the reconstruction power of the VAE, the third term ensuring low divergence between the prior cluster mean and the posterior encoded mean, and the last term ensuring low divergence between the mixing proportions in the prior and the posterior for the latent variable C.

The third term has the prior conditioned on C for each KL Divergence term. This conditioning is not deterministic on the mixing proportions (encoder network for C), and therefore we replace C (for this term) with its smooth relaxation using the Gumbel-Softmax Trick, say ζ. Since ζ is continuous and we cannot compute the expectation term analytically, we need to use Monte Carlo tricks, same as what is done for Z in a standard VAE (and will be done for the second term here as well). We can, therefore, write the relaxed object $\widetilde{\mathcal{L}}$ as follows

$$\begin{aligned}
\widetilde{\mathcal{L}}(\mathbf{x}, \mathbf{y}) &= \log \mathbb{P}\left[\,\mathbf{y} \,\middle|\, \mathbf{x}\,\right] + \mathop{\mathbb{E}}_{\mathbf{z} \,|\, \mathbf{x}}\left[\log \mathbb{P}\left[\,\mathbf{x} \,\middle|\, \mathbf{z}\,\right]\right] - \mathop{\mathbb{E}}_{\mathbf{z},\boldsymbol{\zeta} \,|\, \mathbf{x}}\left[\mathrm{KL}\left(\mathcal{Q}\left(\mathbf{z} \,\middle|\, \mathbf{x}\right) \,||\, \mathbb{P}\left[\,\mathbf{z} \,\middle|\, \boldsymbol{\zeta}\,\right]\right)\right] \\
&\quad - \mathrm{KL}\left(\mathcal{Q}\left(c \,\middle|\, \mathbf{x}\right) \,||\, \mathbb{P}\left[\,c\,\right]\right)
\end{aligned}$$

Although $\widetilde{\mathcal{L}}$ isn't necessarily a lower bound for the objective $\log \mathbb{P}\left[\,\mathbf{x}, \mathbf{y}\,\right]$, however, empirically it seems to work (section 8).

Note. We must ensure the relaxed prior probability $\mathbb{P}\left[\,\mathbf{z} \,\middle|\, \boldsymbol{\zeta}\,\right]$ limits to $\mathbb{P}\left[\,\mathbf{z} \,\middle|\, c\,\right]$ as $\tau \to 0$

Advantages of Using DMVAE as a gating function

[Automatic Differentiation] Use of Black Box Variational Inference with the reparametrization and gumbel trick allows us to use automatic differentiation libraries (such as Tensorflow) to compute the gradients and therefore infer the parameters of the model using gradient based techniques. This is not possible with models using the EM algorithm, as the updates need to be manually coded. **[** Handling of Arbitrary Shaped Clusters] As pointed out earlier, clustering in latent space with non-linear transformations (decoder) allows us to even cluster data split in arbitrary shaped clusters. DMVAE, therefore accurately handles clustering of data at least as good as some other non-trivial models (such as VaDE). This also helps in learning mixture of experts models where the data is complicated or high-dimensional. **[** Fast Inference] Since VAEs, and henceforth DMVAE, use amortized inference and reparametrization, the inference is usually fast. Coupled with automatic differentiation, this affors a "nice" model for ME and/or clustering tasks.

8. Experiments

We conducted a variety of experiments to test the robustness of our algorithm. The datasets we use for these tasks are MNIST **?** and spiral dataset. Firstly, we show some experiments on VaDE and DMVAE showing reconstruction and generation powers of the models. Later, we show the effectiveness of DMVAE over naive gating functions for ME.

8.1. VAE Training

The VaDE and DMVAE models in addition to learning cluster assignments also allow us to generate new data points. This section details our findings of how well these models learnt to reconstruct the original image along with their power to produce new novel images.

8.1.1. MNIST Dataset

The MNIST dataset contains 60,000 training images and 10,000 testing images, each image being greyscaled and of size 28×28. The images represent digits from 0 to 9, and are labelled accordingly.

Figures 4 and 5 show the reconstruction power of DMVAE and VaDE models, respectively, trained using MNIST dataset. It can be clearly seen that both models have learnt to reproduce images well.

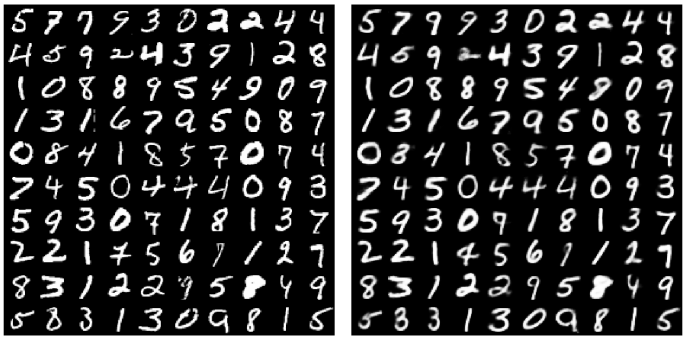

Figure 4: Using DMVAE: Original(left) vs Reconstructed(right)

Figure 5: Using VaDE: Original(left) vs Reconstructed(right)

Figures 6 and 7 show the comparison of performances in generating new images between DMVAE and VaDE. Each row corresponds to images generated as belonging to the same cluster by the model. It can be clearly seen that baring a few exceptions, the models produce extremely real looking images of digits, and hence can be used as powerful generative models.

Moreover, in case of VaDE, we can see higher ambiguity between two clusters (both produce digit 1), which was not observed in our model.

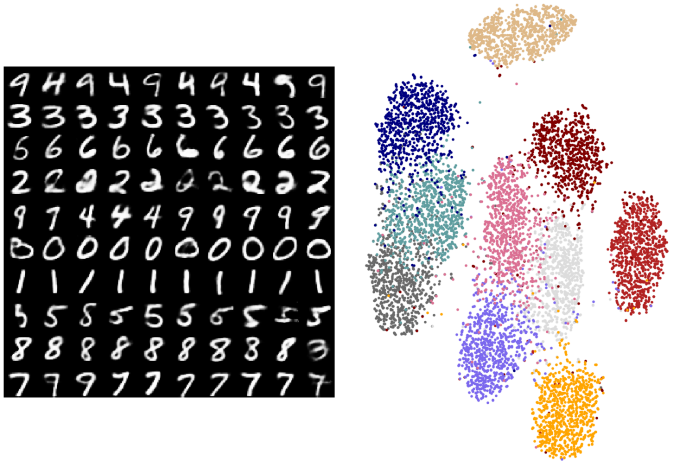

Figure 6: Using DMVAE: Generated Images(left) and tSNE plots(right)

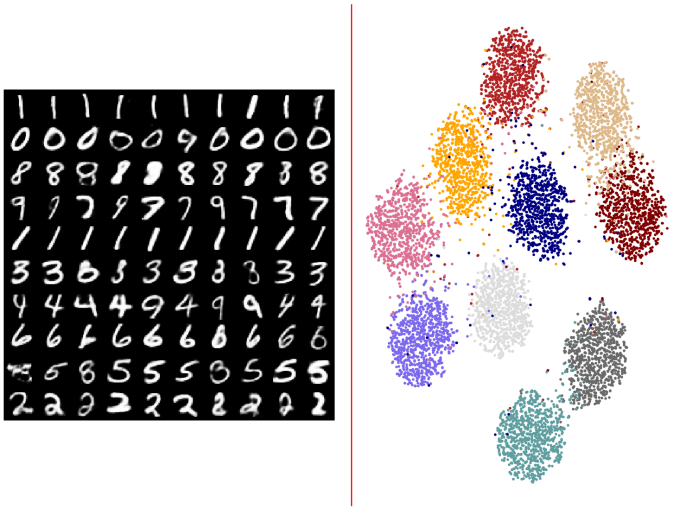

Figure 7: Using VaDE: Generated Images(left) and tSNE plots(right)

8.1.2. Spiral Dataset

Spiral dataset is an artificial dataset used by us to test the performance of our models. The spiral dataset has complicated decision boundary shapes which are not easily learnt by most models. Our dataset consists of 5 classes and we again compare the reconstruction and generative powers of both DMVAE and VaDE models for this dataset.

Figures 8 and 9 show the reconstructed datapoints by DMVAE and VaDE models respectively. Both models are able to reconstruct the points quite well, however, one interesting aspect to note is that the reconstructed points in DMVAE model suffer from very low variance and are tightly bound to their clusters, which is significantly higher in points reconstructed using VaDE model. Although this difference is quite minute, it verifies our claim that inference in DMVAE is better than that in VaDE.

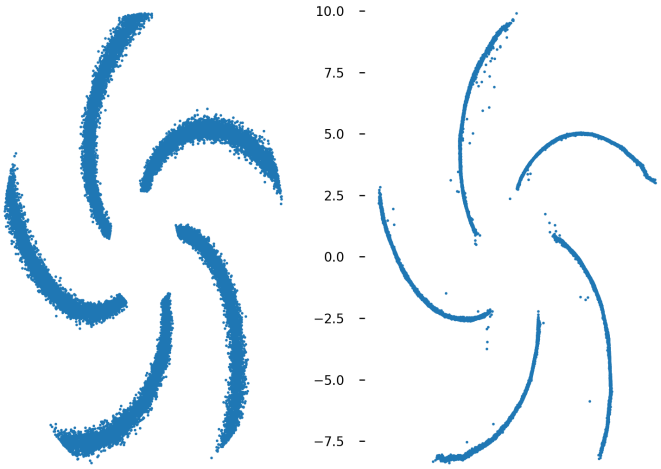

Figure 8: Using DMVAE: Original(left) vs Reconstructed(right)

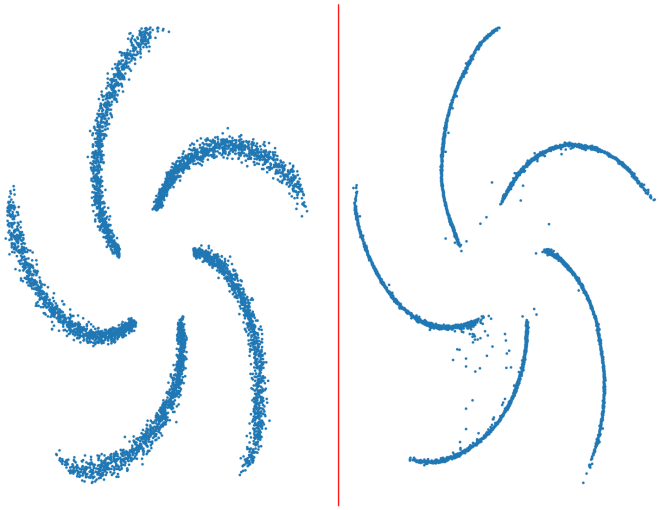

Figure 9: Using VaDE: Original(left) vs Reconstructed(right)

Figures 10 and 11 show the points generated by DMVAE and VaDE models using spiral dataset. It is evident that the DMVAE model clearly outperforms VaDE model in this case, since VaDE is unable to distinguish between the green and blue clusters(see 11). On the other hand, DMVAE generates the points almost similar to real dataset and learns very well defined clusters. This proves that for general use cases, DMVAE is superior to VaDE.

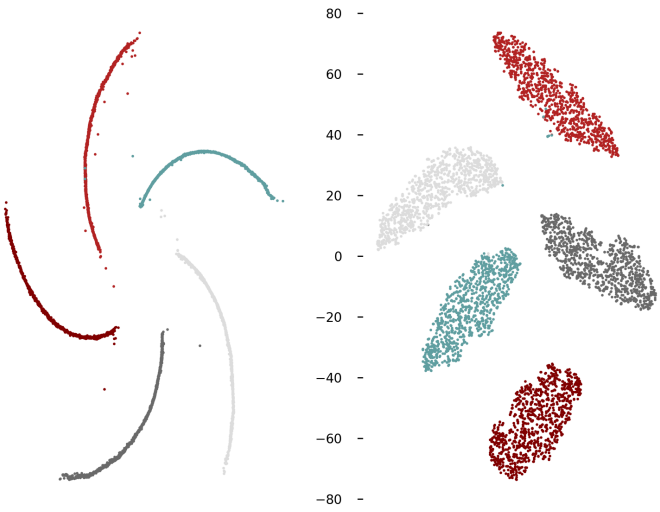

Figure 10: Using DMVAE: Generated Points(left) and tSNE plots(right)

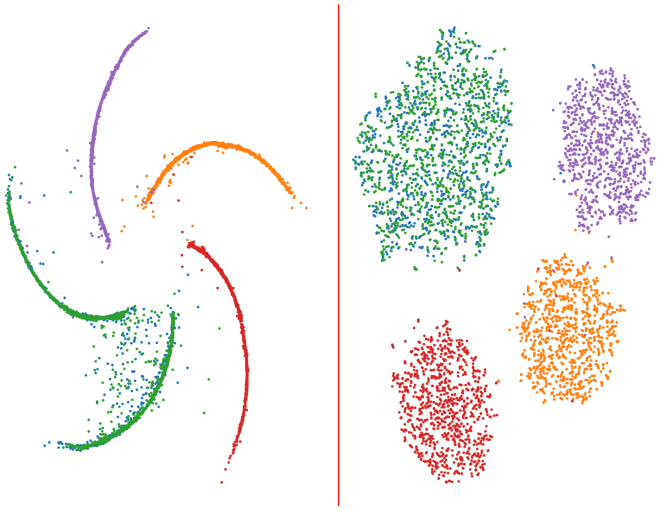

Figure 11: Using VaDE: Generated Points(left) and tSNE plots(right)

The biggest difference between VaDE and DMVAE is the training time and convergence rate. As a general trend, we found VaDE to converge in around 500-700 epochs, whereas DMVAE showed comparable results in only 200-300 epochs. In addition to this, the average training time for one epoch in case of VaDE was 13s and in case of DMVAE was 3.5s. We also observed, empirically, that VaDE performs poorly if there is no pretraining (details in **?**), but our model was more robust to initialization.

For these reasons, we consider the DMVAE model to be better than VaDE for clustering in arbitrary shapes.

8.2. Mixture of Experts Results

To measure the improvement provided by using Mixture of Experts with DMVAE as the gating function, we use a regression task. We call this model DVMoE (Discrete Variational Mixture of Experts). Each datapoint in MNIST dataset was annotated with a K-dimensional label, which was obtained by applying an affine transformation $f : \mathbf{X} \rightarrow \mathbf{wX} + \mathbf{b}$ to the original 28×28 image. Here, $\mathbf{w}$ and $\mathbf{b}$ were chosen randomly, but were same for all points, thus ensuring that the final labels inherited the latent properties of input space. The same was done for Spiral dataset as well.

The standard softmax gating function for MoE models requires a double EM loop, which suffers from some serious drawbacks like limited complexity of models, limited size of datasets, etc. Further, since all updates need to be analytically computed, the resulting models are not end-to-end differentiable and hence backpropagation may not be used.

As an alternative, we tried using a simple Gaussian Mixture Model as the gating function(**?**) which employs a single EM loop, where all the updates were manually calculated. However it proved to be extremely unstable and difficult to train since the probabilities quickly became 0 for higher dimensional spaces. Therefore, we had to discard using this algorithm. Nevertheless, we provide the code to implement the algorithm and are positive that it will perform well in lower dimensional spaces.

For comparisions, we used MoE with DMVAE and a normal neural network as the gating function. The later model is called DeepMoE. DeepMoE learns a mapping between input $\mathbf{x}$ and cluster probability distribution by using a neural network. In all experiments, each expert was fixed to be an affine transformation as described above. The output dimension for the regressors was fixed to 1 in all runs.

Table 1 shows the regression losses we obtained using DMVAE and DeepMoE(which uses a softmax gating function). We report the loss for different dimensions of the latent variable $\mathbf{z}$. It can be clearly seen that our model performs much better than DeepMoE in most scenarios. This can be attributed to the observation that DMVAE uses latent variables which help in better clustering and encode information which can be used to generate new datapoints. Hence, these latent variables provide a much better representation of the original datapoint, and using these latent variables to compute the gating function essentially improves the performance.

	Spiral		**MNIST**	
Number of Dimensions	**Deep MoE**	**DVMoE**	**Deep MoE**	**DVMoE**
1	10.0424	**0.0312**	**28.4584**	34.0551
2	20.4247	**0.0434**	**209.028**	**208.142**
5	180.117	**96.1049**	880.126	**680.714**

Table 1: Regression test results on Spiral and MNIST datasets; Deep MoE vs DVMoE

We also implemented a MoE model using VaDE as the gating function for this regression task. However, owing to limited time at hand and long inference time for VaDE, we only report the regression loss when dimensionality of latent variable was fixed to 5. The obtained regression loss for MNIST dataset was 1109.63.

9. Conclusion

We have created a simple VAE based model for clustering in arbitrary shapes and sizes. The idea is based on using a neural network to model the cluster probabilities, but adding a stochastic layer of latent variables to ensure better clustering. This concept is used by many other techniques for clustering, such as IWMM, VaDE, etc. however, our model shows faster inference that some of these methods with comparable performance and greater ease of implementation.

We further extend our model to act as a gating function for mixture of experts model, and showcase the results on synthetic dataset, showing superior performance than naive implementations for the gating function. As far as we know, there is no such application of Variational Autoencoders (with discrete stochastic layer), and we are therefore presenting a novel method for applying mixture of experts.

We can further extend the model with great ease to hierarchical clustering models by adding more stochastic layers of discrete latent variables without changing the inference method at all. However, we do not experiment with this, given the limited time for the project.

All the code for the project is available here: https://github.com/fat-fighter/vae-mixture-of-experts.git. We have also added a README including instructions on running the code. All the code is implemented on Tensorflow (Python).

References

Wikipedia - latent variables. https://en.wikipedia.org/wiki/Latent_variable.

C. Doersch. Tutorial on Variational Autoencoders. *ArXiv e-prints*, June 2016.

D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013.