Mixture of Experts using Discrete VAE

Gurpreet Singh 150259
Abhisek Panda 150000
Aakarsh Gajbhiye 150000

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 $\boldsymbol{\pi} = [\,\pi_1 \dots \pi_K\,]$. 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.

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

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 $\mathcal{Q}(\mathbf{z} \,|\, \mathbf{x}, \boldsymbol{\phi})$. 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

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

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.

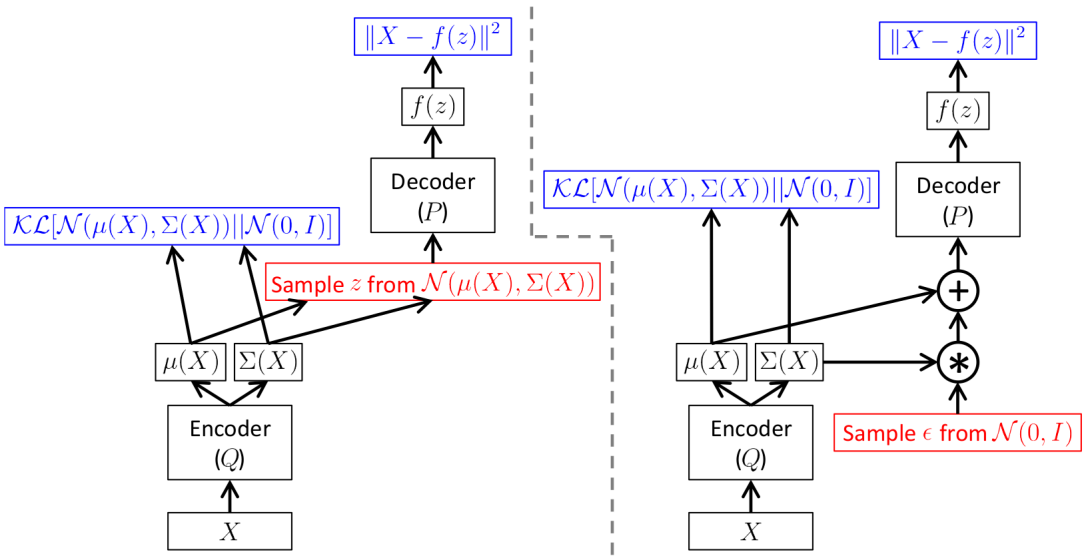

Figure 1: Without (left) and with (right) the reparametrization trick

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 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 $\mathcal{N}\left(\mathbf{z} \,|\, \boldsymbol{\mu}(\mathbf{x}; \boldsymbol{\phi}), \boldsymbol{\sigma}^2(\mathbf{x}; \phi)\mathbf{I}\right)$. 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

[1] Image taken from Doersch (2016)

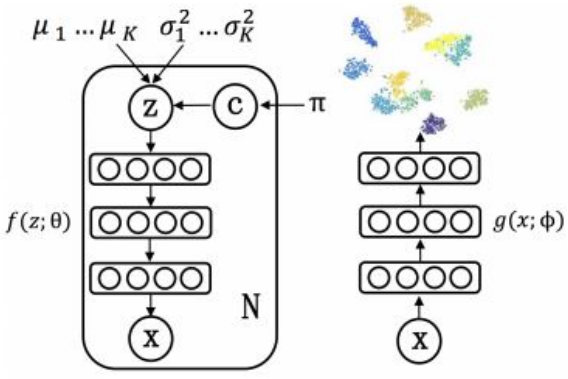

Figure 2: Plate notation for VaDE (left) and an encoder network $g(\mathbf{x};\phi)(right)$

generative story, the joint probability can be written as

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

An instance of VaDE is tuned to maximize the likelihood of the given data points. Given the generative process described above we have

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

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

Mean field is assumed on the proposal distribution, giving:

$$\mathcal{Q}(\mathbf{z},c\,\big|\,\mathbf{x}) \quad = \quad \mathcal{Q}(\mathbf{z}\,\big|\,\mathbf{x})\mathcal{Q}(c\,\big|\,\mathbf{x})$$

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

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

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 $\mathcal{Q}(c\,\big|\,\mathbf{x})$, which although looks absurd, in practice works surprisingly well. First, one can realize, the ELBO can be re-written in the

following manner,

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

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

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

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

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 $\mathbb{P}\left[\,c\,\middle|\,\mathbf{x}\,\right]$ 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 -

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

where K is the number of experts.

We usually represent $\mathbb{P}\left[c = k \,\middle|\, \mathbf{x}\right]$ 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 $\mathbb{P}\left[\mathbf{y} \,\middle|\, \mathbf{x}, c, \mathbf{\Theta}_e\right]$ 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 $\mathbb{P}\left[\mathbf{y} \,\middle|\, \mathbf{x}\right]$ 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].

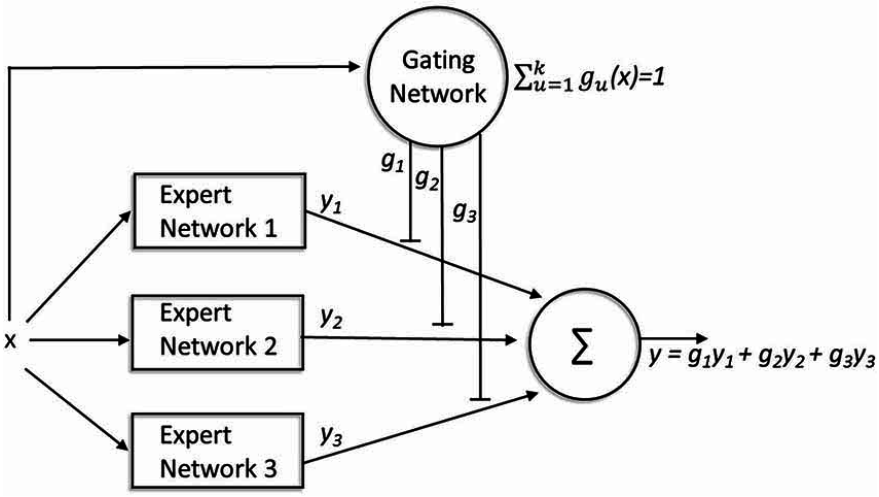

Figure 3: Graphical representaion of a standard Mixture of Experts model

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 -

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

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 $\mathbb{P}\left[\mathbf{y} \,\middle|\, \mathbf{x}, \mathbf{\Theta}\right]$ is used as the output of the architecture. This is, therefore, given by

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

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

[2] Image taken from **?**

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 work as a generative model. For using VaDE in classification/regression tasks, the original theory can be extended in a straight-forward manner by maximizing the log of joint probability for $\mathbf{X}$ & $\mathbf{Y}$.

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

6. Our Model (Some name)

7. Experiments

7.1. VAE Training

7.2. ME Results

8. Related Work

9. Conclusion

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.