Adaptive Approximate Bayesian Inference with Recurrent Networks

Barada prasanna Acharya
Supervised by Eric Nalisnick
Instructed by Padhraic Smyth

Department of Computer Science
University of california, Irvine

Contents

1 Introduction

Deep learning have achieved remarkable results in areas such as natural language processing, pattern recognition and computer vision. Despite this, most deep learning models are often viewed as deterministic functions, which lack uncertainty information about their decisions, unlike Bayesian models. Deep Learning models have to be further enhanced to include metrics conveying uncertainty about their predictions, which is important when facing real world applications like diagnostics using MRI or safety critical self driving cars. In other words, "Model has to know what it doesn't know". While analysing data or making decisions, deep learning models should know certainty about its decisions, data requirements and model selection. The probabilistic view of machine learning offers uncertainty quantification for data analysis and decision making. By using bayesian deep learning we can use existing deep learning models to make decisions with uncertainty information
The main goal of this individual study is to study and analyse bayesian deep learning using variational inference and laplace approximations and investigate variational autoencoder based deep generative models. We implemented a novel method for deep generative model by using adaptive recurrent inference networks and analised its results.

2 Background

2.1 Variational Inference (VI)

Assume that x_i represents observation data and z_i are hidden variables. The posterior distribution of the hidden variables can be written as follows using the Bayes' Theorem.

$$p(z_i|x_i) = \frac{p(z_i,x_i)}{p(x_i)} = \frac{p(x_i|z_i)p(z_i)}{\int_z p(x_i, z_i)} \quad (1)$$

The denominator $p(x_i)$ is not tractable as we need to integrate all configurations of the hidden variables in order to compute the denominator.We have to approximate posterior Inference**.** The main idea behind variational methods is to find some relatively easy and tractable approximation distributions $q(z_i)$ that are as closed as possible to the true posterior distribution $p(z_i|x_i)$. These approximation distribution can have their own variational parameters: $q(z_i; v)$, and we try to find the setting of the parameters that make q close to the posterior of interest. VI turns Inference into Optimization; finding optimal v^* value by starting from v^{init} through optimization which will reduce KL divergence between $q(z; v)$ *and* $p(z_i|x_i)$.

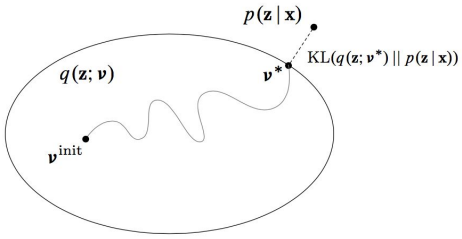

Fig 1. Optimization overview of Variational Inference.

2.2 The Evidence Lower Bound Objective (ELBO)

The main objective of optimization in neural network is to increase the marginal log- likelihood or decrease the negative log likelihood of $p(x)$. So ELBO is the lower bound for $p(x)$ which helps in maximizing intractable $log\ p(x)$ function by maximizing a tractable ELBO function.

2.2.1 First derivation: Using the Jensen's inequality

Starting from the log probability of the observations (the marginal probability of X), we can have:

$$log\ p(x) = log \int_z p(x,z)dz = log \int_z q(z) \frac{p(x,z)}{q(z)}\ dz \quad (2)$$

$$= log\ E_{q(z)}[\frac{p(x,z)}{q(z)}] \quad (3)$$

$$\geq E_{q(z)}[log\ \frac{p(x,z)}{q(z)}] \quad (4)$$

$$\geq E_{q(z)}[log\ p(x,z)] - E_{q(z)}[log\ q(z)] \quad (5)$$

$$\geq E_{q(z)}[log\ p(x,z)] + H_q(z) \quad (6)$$

$$log\ p(x) \geq E_{q(z)}[log\ p(x,z)] + H_q(z)$$

Equation (8) is the variational lower bound, also called ELBO.
The $q(z)$ in equation (4) is an arbitrary distribution which is used to approximate the true posterior distribution $p(z|x)$. Jensen's inequality $f(E[X]) \leq E[f(x)]$ is applied in equation (4) for the concave log function. $H_q[z] = -E_q[log[q(z)]]$ is the Shannon entropy.
Let us denote equation (8) as L:

$$L = E_{q(z)}[log[P(x,z)] + H_q(z)$$

Then L is a lower bound of the log probability of the observations $p(x)$.

2.2.2 Second derivation: KL divergence

To measure the closeness between the approximation distribution $q\ (z)$ and the actual distribution $p\ (z|x)$, Kullback-Leibler (KL) divergence is used. The KL divergence for variational inference is:

$$KLD[\ q(z)\ ||\ p(z|x)] = \int_z q(z)\ log\ \frac{q(z)}{p(z|x)}\ dz \quad (7)$$

$$= \int_z q(z)\ log\ \frac{q(z)p(x)}{p(x,z)}\ dz \quad (8)$$

$$= -\int_z q(z)\ log\ \frac{p(x,z)}{q(z)}\ dz + \int_z q(z)\ log\ p(x)\ dz \quad (9)$$

$$= - E_{q(z)}[log \frac{p(x,z)}{q(z)}] + log\, p(x) \quad (10)$$

$$log\, p(x) = + E_{q(z)}[log \frac{p(x,z)}{q(z)}] + KLD[\, q(z) \,||\, p(z|x)] \quad (11)$$

$$log\, p(x) = + L + KLD[\, q(z) \,||\, p(z|x)] \quad (12)$$

where L is the ELBO defined above. Equation (10) is obtained by the normalization constraint: $\int_z q(z) = 1$. Rearrange the equations we can get:

$$L = logp(x) - KL[q(z) \,||\, p(z|x)] \quad (13)$$

As KL divergence is always ≥ 0, once again we get $L \leq logp(x)$ is a lower bound of the log probability of observations. We can also define the difference between them is exactly the KL divergence of the approximation and true distribution. In others words, the lower bound L and the marginal log probability will be same iff the approximation distribution is perfectly closed to the true posterior distribution.

2.3 Stochastic Gradient Variational Bayes

The VAE's generative and variational parameters are estimated by *Stochastic Gradient Variational Bayes* (SGVB). SGVB is distinguished from classical variational Bayes by it's use of differentiable Monte Carlo (MC) expectations. To elaborate, consider SGVB's approximation of the usual ELBO:

$$\overline{L}(\theta, \Phi, x_i) = \frac{1}{S} \sum_{s=1}^{S} log\, p_\theta(x_i|\widehat{z}_{i,s}) - KL(q_\Phi(z_i|x_i) \,||\, p(z)) \quad (14)$$

$$\frac{\partial}{\partial \Phi} \sum_{s=1}^{S} log\, p_\theta(x_i|\widehat{z}_{i,s}) = \sum_{s=1}^{S} \frac{\partial}{\partial \widehat{z}_{i,s}} log\, p_\theta(x_i|\widehat{z}_{i,s}) \frac{\partial \widehat{z}_{i,s}}{\partial \Phi} \quad (15)$$

Problem with equation(16) is $\frac{\partial \widehat{z}_{i,s}}{\partial \Phi}$ is non differentiable. Therefore, an essential requirement of SGVB is that the latent variable be represented in a *differentiable, non-centered parametrization* (DNCP) what allows the gradients to be taken through the MC expectation, z must have a functional form that deterministically exposes the variational distribution's parameters and allows the randomness to come from draws from some fixed distribution. For instance, the VAE's Gaussian latent variable (with diagonal covariance matrix) is represented as $\widehat{z}_i = \mu + \sigma \odot \varepsilon \; where \; \varepsilon \sim N(0, 1)$

3 Bayesian Neural Network

A probabilistic interpretation of deep learning models can be obtained by inferring distribution over the models' weights unlike traditional neural network where we obtain point estimates for weights. This type of probabilistic models popularly known as Bayesian neural networks (BNNs, Bayesian NNs) which introduce uncertainty in the parameters estimation, robustness to overfitting, and can easily learn from small datasets. A probabilistic model $P(Y \mid X, w)$ can represent a BNN; which means given an input $x_i \varepsilon R^p$ a BNN assigns a probability to each possible output $y_i \varepsilon Y$, using the set of global parameters or weights w. A prior distribution over weights is placed by BNN which later induces a posterior distribution over a parametric set of functions. For classification, Y is a set of classes and $P(y_i \mid x_i, w)$ is a categorical distribution which uses cross-entropy or softmax loss function whereas in case of regression $Y is R$ and

$P(Y \mid X, w)$ is a Gaussian distribution and uses a mean squared error loss. Several successive layers of linear transformation given by weights w causes non-linear mapping of Inputs X onto the Y. The weights can be learnt by maximum likelihood estimation (MLE): given a set of training examples $D = (x_i, y_i)_i$, the MLE weights w^{MLE} are given by:

$$w^{MLE} = arg\ max_w\ log\ p(D|w) = arg\ max_w \sum_i log\ p(y_i|x_i, w) \tag{16}$$

gradient descent (e.g., backpropagation) can be used to achieve this. By placing a prior upon the weights w, maximum a posteriori (MAP) weights w^{MAP} can be obtained which introduce regularisation. Depending on prior distribution Gaussian prior or Laplace Prior, L2 or L1 regularization can be introduced.

$$w^{MAP} = arg\ max_w\ log\ p(w|D) = arg\ max_w\ log\ p(D|w) + log\ p(w) \tag{17}$$

Due to the intractable nature of the posterior distribution involved in bayesian equation, we can't integrate exactly over the parameters directly, so some form of approximation technique is required to approximate posterior distribution to a analytical tractable distribution. Here, the technique of Laplace approximation and variational inference, has been applied to Bayesian neural networks. For Laplace distribution, we will approximate the posterior distribution by a Gaussian, centred at a mode of the true posterior. With these two approximations, we will obtain models that are analogous to the regression and classification models.

3.1 Laplace Approximation

Laplace approximation finds a gaussian approximation to the probability density defined over a set of continuous variables so that the approximation distribution will be centered on mode of the true distribution. Assume a case of single continuous variable z, and the distribution p(z) is defined by

$$p(z) = \frac{1}{Z} f(z)\ where \quad Z = \int f(z)\ normalization\ coefficient$$

In the Laplace method, a gaussian approximation $q(z)$ is needed to be found which is centered on the mode of $p(z)$. So first, a point z_0, mode of $p(z)$ is needed to be found, where $p'(z) = 0$

$$\frac{df(z)}{dz} = 0\ at\ z = z_0$$

If a $f(z)$ is a gaussian function we can express it as a quadratic function of variables

$$f(z) = f(z_0)\ exp(-\frac{A}{2}(z - z_o)^2) \tag{18}$$

Taylor expansion of $f(z)$ centered on the mode z_0 where $A = -\frac{d^2 f(z)}{dz^2} log\ f(z)\ at\ z = z_o$

$\frac{df(z)}{dz} = 0$ so it will be absent in the taylor expansion

$$log\ f(z) \simeq log\ f(z_0) - \frac{A}{2}(z - z_o)^2 \tag{19}$$

So we can obtain a normalized equation for approximate distribution $q(z)$

$$q(z) = (\tfrac{A}{2\pi})^{1/2} \; exp(-\tfrac{A}{2}(z - z_o)^2) \tag{20}$$

For multivariate gaussian equation with M degree,

$$q(z) = \tfrac{|A|^{1/2}}{(2\pi)^{M/2}} \; exp\{-\tfrac{1}{2}(z - z_o)^T A(z - z_o)\} = N(z|z_0, A^{-1}), \; here \; A \; precision \; matrix \tag{21}$$

Steps for finding Laplace Approximation:

1. FInd parameter θ_{MAP} so that it is the mode of distribution

$$log\, p(\theta|x) = log\, p(x|\theta) + log\, p(\theta) - log\, p(x)$$

$$\frac{\partial\, log\, p(\theta|x)}{\partial\theta} = \frac{\partial\, log\, p(x|\theta)}{\partial\theta} + \frac{\partial\, log\, p(\theta)}{\partial\theta} \; as \; \frac{\partial\, log\, p(x)}{\partial\theta} = 0$$

2. Find Hessian matrix $H[\theta_{MAP}]$ for second order derivative.

3. Combine θ_{MAP} and Hessian matrix to get posterior distribution q, $q(\mu = \theta_{MAP}, \sigma^2 = (H[\theta_{MAP}])^{-1})$

3.2 Variational Approximation

In Variational approximation, Loss function used for Bayesian Neural network is

$$Loss\, Function = E_{q(w)}[-\, log\, p(Y|X, w)] + KLD[q(w) \,||\, p(w)] \tag{22}$$

Where $q(w)$ is approximate posterior parameter weight distribution, Y is actual label, x is input dataset and $p(w)$ is prior parameter weight distribution. The first term is data loss part so that we can reconstruct data with maximum likelihood and second part acts like a regularization term, which ensures the posterior distribution won't move far away from the prior distribution. Steps for the calculation of VI approximation.

1. Use Monte Carlo Approximations

$$E_{q(w)}[-\, log\, p(Y|X, w)] = -\int q(w)\, log\, p(Y|X, w)\, dw \tag{23}$$

It is hard to calculate expectation for difficult neural network function, So we should use monte-carlo method to approximation.

$$E_{q(w)}[-\, log\, p(Y|X, w)] = \tfrac{1}{S} \sum_{s=1}^{S} log\, p(Y|x, \hat{w}_s) \tag{24}$$

Where $\hat{w}_s$ will be randomly sampled from $q(w; \mu_{\phi}, \sigma_{\phi}{}^2)$ and S is the number of samples.

2. Sample via Differentiable Non-centered Parameterization.

In *Stochastic Gradient Variational Bayes ,* taking derivative of loss function with respect to sampled parameter, $\frac{d\, Loss}{\partial \hat{w}_s}$, is impossible as $\hat{w}_s$ is sampled randomly. We need to represent sampled weight parameters in terms of (DNCP) as discussed in section 2.4.

$E_{q(w)}[-\ log\ p(Y|X,w)]$ term for classification and regression will be different. Assume f is output of the neural network function, $\hat{w}_s$ is sampled weight parameter , x is input and $\hat{y}$ is the true label for the dataset.

For classification, cross entropy Error will be calculated.

$$E_{q(w)}[-\ log\ p(Y|X,w)] = \frac{1}{S}\sum_{s=1}^{S} -\hat{y}\ log\ f(\hat{w}_s,\ x) - (1-\hat{y})(1 - log\ f(\hat{w}_s,\ x)) \quad (25)$$

In case of regression, Mean square error is calculated

$$E_{q(w)}[-\ log\ p(Y|X,w)] = \frac{1}{S}\sum_{s=1}^{S} (\hat{y} - f(\hat{w}_s,\ x)^2) \quad (26)$$

3. KL Divergence Calculation

KL divergence represents how much information we lose when we approximate prior $p(w)$ distribution with posterior $q(w)$ distribution. So, this term act like a regularizer which put a bound on posterior distribution so that while optimizing the posterior shouldn't move far away from the prior.

$$KLD\ [q(w)\ ||\ p(w)] = -0.5\ (-2 * log\ \frac{\sigma_{post}}{\sigma_{prior}} - \frac{\sigma_{post}^{\ 2} + (\mu_{prior} - \mu_{post})^2}{\sigma_{prior}^{\ 2}} + 1) \quad (27)$$

3.3 Experiment

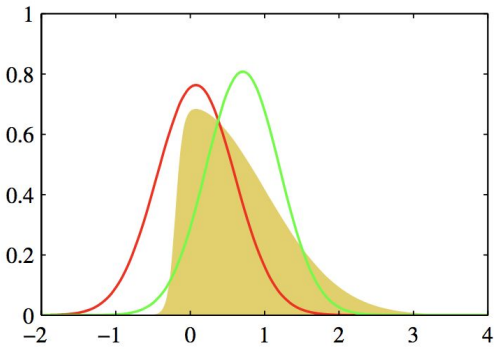

Fig 3. Yellow region shows a posterior to be approximated and the red curve represents the posterior approximated via laplace approximation and green curve represents the posterior approximated via VI approximation. (Pic taken from Bishop book for pattern recognition and machine learning)

Implemented a bayesian neural network with single hidden layer, 50 node, learning rate 0.0003 (other parameters kept at their Tensorflow defaults), batch size =128, epochs 1000, train, val and test in 70:10:20 ratio, early stopping with 25 look-ahead epochs, using VI approximation and Laplace approximation for the Boston Housing, Energy Efficiency and Yacht Hydrodynamics datasets.

Dataset	Data-points(N)	Dimension(d)	Test Error (VI)	Test Error (LA)
Boston Housing	506	13	2.275 +/- 0.132	2.764 +/- 1.189
Energy Efficiency	768	8	0.058 +/- 0.013	0.186 +/- 0.119
Yacht Hydrodynamics	308	6	1.984 +/- 0.596	1.953 +/- 0.626

Table 1 :Test Error for Variational Inference and Laplace approximation for Boston Housing, Energy Efficiency, Yacht Hydrodynamics dataset .

Test error for VI approximation is seems less than Laplace approximation for the above dataset.

4 Deep Generative Model

4.1 Variational Autoencoder

A Variational Autoencoder(VAE) model is consist of two multilayer perceptrons: one acts as as an encoder network or inference model performing the reverse mapping from x_i to z_i and the one is decoder network, also known as generative network mapping a latent variable z_i to an observed datapoint x_i . Two network together form a computational pipeline that resembles an autoencoder.

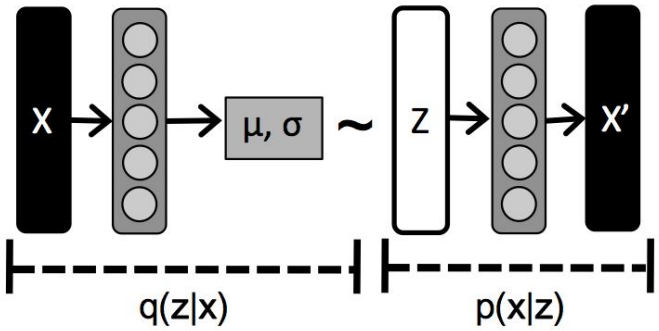

Fig 2. Variational Autoencoder.

The generative process can be written mathematically as $z_i \sim p(z), \;\; x_i \sim p_\theta(x|z_i)$
where $p(z)$ is the prior and $P_\theta(x|z_i)$ is the generative network with parameters θ. Inference network is represented as $q_\Phi(z|x_i)$, parametrized by Φ ..

$$log\, p_\theta(x_i) \geq \frac{1}{S}\sum_{s=1}^{S} log\, p_\theta(x_i|\hat{z}_{i,s}) - KL(q_\Phi(z_i|x_i) \,||\, p(z_i)) \; where\, \hat{z}_{i,s} = \mu_\Phi + \sigma_\Phi \odot \hat{\varepsilon}_s \; ; \; \hat{\varepsilon}_s \sim N(0,1)$$

Marginal log likelihood equation for VAE for S samples of z_i .

4.2 Generative Model

Generative models are combinations of neural networks which can be used to learn a given data distribution, so that we can generate fake data points from real data points which are looking alike. VAE is a major type of Generative model where by using multilayer architecture, a Deep Generative Model can be built. In VAE a latent distribution is learnt which successfully compress the input data into a

latent space and learn only important features and later reconstruct the data points. In this method, a predefined prior distribution can be given as input, so that we can know approximate distribution of the latent distribution which the hidden layer will follow. For generating our own data points, hidden layer distribution can be sampled and feed it to the output layer.

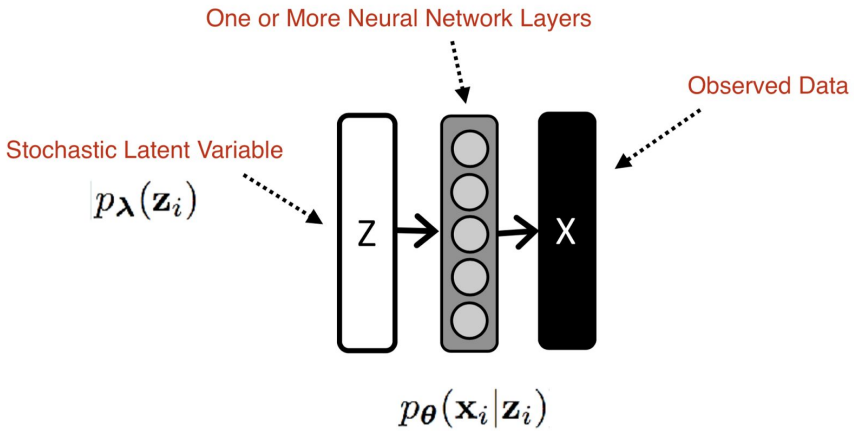

Fig 4. Deep Generative Model.

4.3 Deep Latent Gaussian Mixture Model

In traditional VAE, the prior $p(z)$ and variational posterior have been marginally gaussian, since gaussian model is very simple to be used as latent space, however, for complicated dataset, a novel modification to the VAE, concepts of mixture gaussian model is used. In gaussian mixture model based VAE, instead of drawing the latent variables from a gaussian distribution, samples can be drawn from the gaussian mixture distribution, making the hidden representation an infinite sequence of stick-breaking weights. We term this model a *Deep Latent Gaussian Mixture Model Variational Autoencoder* (DLGMM-VAE) and below detail the generative and inference processes implemented in the decoding and encoding models respectively.

Often the posteriors we encounter in the wild are multi-modal, and thus using unimodal posterior approximations can drastically misrepresent the posterior support. A straightforward extension is to use mixture approximations of the form:

$$q(\theta) = \sum_k \pi_k \, q_k(\theta) \tag{29}$$

Assume $p_\theta(x_i|\, z_i)$ as decoder/generator network and $p_\lambda(z_i)$ is prior distribution of hidden layer and $q_\Phi(z_i|\, x_i)$ is the posterior approximation then the marginal likelihood of constructed data can be

$$log\, p(x) = log \int_z p_\theta(x_i|\, z_i) p_\lambda(z_i) dz \tag{30}$$

$$= log \int_z \frac{q_\Phi(z_i|\, x_i)}{q_\Phi(z_i|\, x_i)} \, p_\theta(x_i|\, z_i) p_\lambda(z_i) dz \tag{31}$$

$$\geq \int_z q_\Phi(z_i|\, x_i) \, log \, \frac{p_\theta(x_i|\, z_i) p_\lambda(z_i)}{q_\Phi(z_i|\, x_i)} dz \tag{32}$$

$$= E_{q(z)}[log\ p_\theta(x_i|\ z_i)]\ - KLD[q_\Phi(z_i|\ x_i)\ ||\ p_\lambda(z_i)] \tag{33}$$

In the equation (33) $E_{q(z)}[log\ p_\theta(x_i|\ z_i)]$ term can be treated as data reconstruction and the $KLD[q_\Phi(z_i|\ x_i)\ ||\ p_\lambda(z_i)]$ term as regularization. However, a concern with neural network is these two terms are often analytically intractable. This problem can be solved with following steps.

Step #1: Use Monte Carlo Approximations

$$log\ p(x) = = \frac{1}{S}\sum_{s=1}^{S} log\ p_\theta(x_i|\widehat{z}_{i,s})\ -\ KL(q_\Phi(z_i|x_i)\ ||\ p_\lambda(z_i)) \tag{34}$$

where s is the number of samples. While taking derivative in SGVB for the above equation, we can use DNCP method as discussed in section 2.4.

Step #2: Sample via Differentiable Non-Centered Parameterizations

$$q_\Phi(z_i|x_i)\ =\ N(z_i;\ \mu_\Phi, {\sigma_\Phi}^2) \tag{37}$$

$$\widehat{z}_i = \mu_\Phi\ + \sigma_\Phi\ \odot \widehat{\varepsilon}_s\ where\ \widehat{\varepsilon}_s\ \sim N(0,1)$$

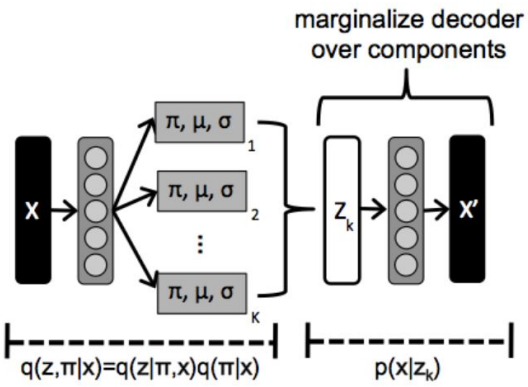

Fig 5. Deep Latent Gaussian Mixture Model.

The ELBO function for the DLGMM model can be expressed as below.

$$L_{SGVB} = \sum_k \mu_{\pi_k}\ [\frac{1}{S}\sum_s log\ p_\theta(x_i|\ \widehat{z}_{i,k,s})]\ +\ E_{q_k}\ [log\ p(z_i)]\ ]\ - KLD[q(\pi_k|x_i)\ ||\ p(\pi_k)]\ -\ \frac{1}{S}\sum_s log \sum_k \widehat{\pi}_{i,k,s} q(\widehat{z}_{i,k,s};\Phi_k) \tag{38}$$

Proof:

Let's start with the base ELBO equation(8)

$$log\ p(x) \geq\ E_{q(z)}[log\ p(x,z)]\ +\ H_q(z) \tag{39}$$

$$E_{q(z)}[\log p(x,z)] = \int_z q(z) \log (p(x|z)p(z))\, dz \tag{40}$$

if z has a prior on π , $q(z) = q_k(z|\pi)\, q(\pi)$ *and using* $q(z) = \sum_k \pi_k\, q_k(z)$

$$= \int_\pi \int_z (\sum_k \pi_k q_k(z|\pi)\, q(\pi)) \log (p(x|z)p(z)) dz\, d\pi \tag{41}$$

$$= \sum_k \int_\pi (\pi_k\, q(\pi)\, d\pi) \int_k q_k\,(z|\pi) \log (p(x|z)p(z)) dz \tag{42}$$

$$= \sum_k E[\pi_k]\, [\, E_{q_k}[\log p(x|z)] \; + \; E_{q_k}[\log p(z)]\,] \tag{43}$$

$$= \sum_k \mu_{\pi_k}\, [\, E_{q_k}[\log p(x|z)] \; + \; E_{q_k}[\log p(z)]\,] \tag{44}$$

$$= \sum_k \mu_{\pi_k}\, [\tfrac{1}{S} \sum_s \log p_\theta(x|\widehat{z_s})] \; + \; E_{q_k}[\log p(z)]\,] \; \textit{using MC sampling} \tag{45}$$

Entropy of Mixture model :

$$H_q(z) = E_{q(z)}[-\log q(z)] \tag{46}$$

$$= -\int q(z) \log q(z)\, dz \tag{47}$$

$$= -\int \sum_k \pi_k\, q_k(z)) \log(\sum_k \pi_k q_k(z))\, dz \tag{48}$$

$$= -\sum_k \pi_k \int q_k(z) \log(\sum_k \pi_k q_k(z))\, dz \tag{49}$$

$$= -\sum_k \pi_k\, E_{q_k(z)}[\log(\sum_k \pi_k q_k(z))] \tag{50}$$

$$= -\tfrac{1}{S} \sum_s \pi_k \log \sum_k \widehat{\pi}_{i,k,s} q(\widehat{z}_{i,k,s}; \Phi_k) \tag{51}$$

$$\log p(x) \geq \sum_k \mu_{\pi_k}\, [\tfrac{1}{S} \sum_s \log p_\theta(x|\widehat{z_s})] \; + \; E_{q_k}[\log p(z)]\,] \; + \; -\tfrac{1}{S} \sum_s \pi_k \log \sum_k \widehat{\pi}_{i,k,s} q(\widehat{z}_{i,k,s}; \Phi_k)$$
$$- KLD[q(\pi_k|x_i) \,||\, p(\pi_k)]$$

The last term is the KL Divergence between prior mixture weight and the posterior mixture weight which is used as regularization.

4.4 Adaptive Computation Inference Network

When using models with per-data-point latent variables of the form $z_i \sim p(z), \quad x_i \sim p(x_i|z_i)$. It may be useful to have per-data-point posteriors $q(z_i|x_i)$ of varying complexity. For example, some data points may look like a blend of two different types (for example, a digit that looks like both a one and a seven) and we may wish to place posterior mass on the region of latent space for each digit. We can define inference networks with this behavior by using an adaptive computation Recurrent Neural Network (RNN) as the inference network. Depending on the previous mixture weight, next mixture weights will be learnt by RNN in a sequential manner. We propose a network with the following form.

In the deep gaussian mixture model inference network computes the parameters of the K mixture components, and the density network is run for a sample from each. So the density network runs K^s times, where K is the number of components and s is the number of stochastic layer for each forward pass. In case of RNN network, computation time can be cut because we are using stick breaking weight and computation will stop when remaining stick goes below some threshold, which would give savings.

4.4.1 Variational Inference with Mixture Approximations

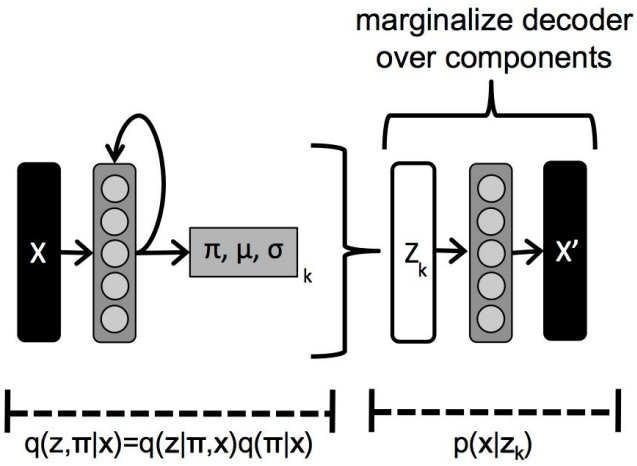

Fig 6. Deep Latent Gaussian Mixture Model with Adaptive Computation Inference Network

Often the posteriors we encounter in the wild are multi-modal, and thus using unimodal posterior approximations can drastically misrepresent the posterior support. A straightforward extension is to use mixture approximations of the form:

$$q(\theta) = \sum_k \pi_k \, q_k(\theta)$$

where π_k is a mixture weight and q_k is the kth component density. Using this approximation and the ELBO equation (8) we can derive

$$log\ p(x) \geq E_{q(\theta)}[log\ p(x,\theta)] + H_q[\theta] \tag{52}$$

$$E_{q(\theta)}[log\ p(x,\theta)] = \int_\theta q(\theta)\ log\ p(x,\theta)\ d\theta \tag{53}$$

$$= \int_{\theta} \sum_{k} \pi_k \, q_k(\theta) \, log \, p(x, \theta) \, d\theta \tag{54}$$

$$= \sum_{k} \pi_k \int_{\theta} q_k(\theta) \, log \, p(x, \theta) \, d\theta \tag{55}$$

$$= \sum_{k} \pi_k \, E_{q_k(\theta)} \, [log \, p(x, \theta)] \tag{56}$$

the entropy of the mixture model $H_q(\theta)$:

$$H_q(\theta) = E_{q(\theta)}[- log \, q(\theta)] \tag{57}$$

$$= -\sum_{k} \pi_k \int q_k(\theta) \, log(\sum_{j} \pi_j q_j(\theta)) \, d\theta \tag{58}$$

$$\geq -\sum_{k} \pi_k \, log(\sum_{j} \pi_j \int q_k(\theta) \, q_j(\theta)) \, d\theta \quad \text{using jensen's inequality} \tag{59}$$

For Gaussians, the convolution term $\int q_k(\theta) \, q_j(\theta) \, d\theta$ can be represented as

$$\int q_k(\theta) \, q_j(\theta) \, d\theta = E_{q_k}[q_j(\theta)] = N(\mu_k; \mu_j, (\sigma_j{}^2 + \sigma_k{}^2)I) \tag{60}$$

$$H_q(\theta) \geq -\sum_{k} \pi_k \, log(\sum_{j} \pi_j E_{q_k}[q_j(\theta)]) \tag{61}$$

$$\geq -\sum_{k} \pi_k \, log(\sum_{j} \pi_j \, N(\mu_k; \mu_j, (\sigma_j{}^2 + \sigma_k{}^2)I) \tag{62}$$

Using equation (56) and (62) we can get

$$log \, p(x) \geq L_{ELBO} \geq \sum_{k} \pi_k \, E_{q_k(\theta)} \, [log \, p(x, \theta)] \; +- \sum_{k} \pi_k \, log(\sum_{j} \pi_j \, N(\mu_k; \mu_j, (\sigma_j{}^2 + \sigma_k{}^2)I)$$

Intuitively, the convolution term in the entropy lower bound can be seen as encouraging the components to have low probability under one another. Or in other words, to spread out and diversify their posterior coverage.

4.4.1 Adaptive Inference with RNNs

$$h_{i,\, t=k} = f_{RNN}(h_{i,\, t=k}) \tag{63}$$

$$\mu_{i,\, t=k} = f_{\mu}(h_{i,\, t=k}) \tag{64}$$

$$\sigma_{i,\,t=k} = f_{\sigma}(h_{i,\,t=k}) \tag{65}$$

$$\gamma_{i,\,t=k} = f_{\gamma}(h_{i,\,t=k}) \tag{66}$$

The first variable $h_{i,\,t=k}$ is simply the RNN's hidden state at time step k, as generated by an LSTM unit taking the previous hidden state $h_{i,\,t=k-1}$ and the data point x_i as input. The last three variables are components of the variational mixture approximation. The first two $\mu_{i,k}$ and $\sigma_{i,k}$ are the parameters of the kth Gaussian component and are computed via independent functions of the current hidden state. The last variable is $\gamma_{i,k}$, taking on a value (0,1), and thus would have a logistic function output. We compose $\gamma_{i,k}$ the variables into the mixture components by using the Dirichlet's stick breaking construction:

$$\pi_{i,\,1} = \gamma_{i,\,1}\ ,\ \pi_{i,\,k} = \gamma_{i,\,k} \prod_{j=1}^{k-1} (1 - \gamma_{i,\,j}) \tag{67}$$

The RNN will halt computation when the remaining stick is less than some threshold, i.e. $\prod_{j=1}^{k-1} (1 - \gamma_{i,\,j}) < \varepsilon$. Following Occam's razor, we assume mixtures with few components are preferred, and thus penalize the RNN's computation according to entropy of the mixture weights:

$$\begin{aligned} H[\pi_i] &= -\sum_{k=1}^{K} \pi_{i,\,k} log\ \pi_{i,\,k} \\ &= -\sum_{k=1}^{K} \gamma_{i,\,k} \prod_{j=1}^{k-1} (1 - \gamma_{i,\,j}) [\log\ \gamma_{i,\,k} + \prod_{j=1}^{k-1} (1 - \gamma_{i,\,j})] \end{aligned} \tag{68}$$

Plugging the term $H[\pi_i]$ back into the ELBO lower bound in part #1, we have our final objective for the ith data point:

$$L_{ADPAT}(x_i) = \sum_{k} \pi_k\, E_{q_k(\theta)}\,[log\ p(x,\theta)] + - \sum_{k} \pi_k\, log(\sum_{j} \pi_j\, N(\mu_k; \mu_j, (\sigma_j{}^2 + \sigma_k{}^2) I) - H[\pi_i] \tag{69}$$

Marginal Likelihood calculation for single gaussian distribution

$$log\ p(x) = log\ E_{q(z)}[\tfrac{p(x,z)}{q(z)}] \qquad \text{from equation(5)}$$

$$\geq E_{q(z)}[log\ \tfrac{p(x,z)}{q(z)}] \qquad \text{from equation (6)}$$

$$= \tfrac{1}{S} \sum_{s=1}^{S} log\ \tfrac{p(x,\widehat{z}_s)}{q(z)} \tag{70}$$

$$= \tfrac{1}{S} \sum_{s=1}^{S} log\ \tfrac{p(x|\widehat{z}_s)p(\widehat{z}_s)}{q(z)} \tag{71}$$

$$= \frac{1}{S}\sum_{s=1}^{S} log\, p(x|\hat{z}_s) + log\, p(\hat{z}_s) - log q(z_s)$$

$$= log\ likelihood + log\_prior - log\_posterior$$

As we know marginal likelihood has lower bound on ELBO, however, to calculate exact value of marginal likelihood of test data below derivation is performed.

$$log\,(E_{q(\hat{z}_s)}[\frac{p(x,\hat{z}_s)}{q(\hat{z}_s)}]) = log\, E_{q(\hat{z}_s)}[exp\,(log\frac{p(x,\hat{z}_s)}{q(\hat{z}_s)})]$$

$$= log\, \frac{1}{S}\sum_{s} exp\{log(p(x|\hat{z}_s) + log\, p(\hat{z}_s) - log\, q(\hat{z}_s))\}$$

Marginal Likelihood calculation for mixture gaussian sample

$$k \sim Multinoulli(\pi)\,,\ \hat{z}_s \sim q_k(\hat{z}_s|x_i)$$

$$log\,(E_{q_k(\hat{z}_s)}[\frac{p(x,\hat{z}_s)}{q_k(\hat{z}_s)}]) = log\, E_{q_k(\hat{z}_s)}[exp\,(log\frac{p(x_i,\hat{z}_{s,k})}{q_k(\hat{z}_{s,k})})]$$

$$= log\, \frac{1}{S}\sum_{s} exp\{log(p(x_i|\hat{z}_{s,k}) + log\, p(\hat{z}_{s,k}) - log\, q_k(\hat{z}_{s,k}))\}$$

4.5 Experiment

Our experiment is performed with the above mentioned formula and hidden dimension of RNN is 200 and Latent variable dimension is 50 and maximum number of mixture components allowed is 5. Our optimization ends at 583 epochs with early stopping criteria with 25 look-ahead epochs. The Encoder RNN will learn mean, variance, weight attribute for corresponding mixture component.

Below ELBO of first and last epochs is mentioned for train and validation dataset. Marginal likelihood for validation and test dataset is also mentioned.

Epoch 1. Train ELBO: -222.954, Validation ELBO: -191.126
Epoch 583. Train ELBO: -37.460, Validation ELBO: -41.890
Validation Marginal Likelihood: -144.762
Test Marginal Likelihood: -143.827

Fig 7 sample generation of images for MNIST dataset using Adaptive Inference with RNN

We can infer from the result that the model is not learning that well as expected, we can see some glimpse of digit's images, however, those are not clear. One more point though ELBO function is decreasing still, marginal likelihood is high compared to other VAE models.

5 Conclusion

The main aim of this individual study was to get hands experience on different approximation technique used for inference in implementing bayesian neural network. We have studied approximate bayesian inference and laplace approximation and implemented a bayesian neural network using it. In the second half of the study, we studied about variational autoencoder based deep latent gaussian mixture model generative model. Later, we have implemented Adaptive Computation Inference Network using Latent gaussian mixture approximations and adaptive Inference with RNNs. Despite of the fact model doesn't perform that well, however the result shows promising in using RNNs in VAE based deep generative model.

6 References

1. Alex Graves. Adaptive computation time for recurrent neural networks. *ArXiv e-prints*, 2016.
2. Eric Nalisnick, Lars Hertel, and Padhraic Smyth. Approximate inference for deep latent gaussian mixtures. *NIPS Workshop on Bayesian Deep Learning*, 2016.
3. Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *International Conference on Machine Learning (ICML)*, 2014.
4. Abel Rodriguez and David B Dunson. Nonparametric bayesian models through probit stick-breaking processes. *Bayesian analysis*, 2011.
5. Tim Salimans, Diederik Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. *International Conference on Machine Learning (ICML)*, 2015.
6. Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. 2015. Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37* (ICML'15), Francis Bach and David Blei (Eds.), Vol. 37. JMLR.org 1613-1622.
7. Eric Nalisnick, Padhraic Smyth, STICK-BREAKING VARIATIONAL AUTOENCODERS , Conference paper at ICLR 2017
8. José Miguel Hernández-Lobato, Ryan P. Adams, Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks, Proceedings of the 32 nd International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.