CS109B Advanced Section : Information Theory, Variational Inference for Variational AutoEncoders and Bayesian Neural Networks

Professor : Pavlos Protopapas, TF : Srivatsan Srinivasan

21 December 2018

Abstract

In this tutorial, we discuss a brief introduction to information theory that presents salient concepts such as entropy, cross entopry and KL divergence. Using these fundamentals, the tutorial then introduces variational inference formulation along with mean field approximations. Then, variational autoencoders - both the generative and conditional variants are introduced to demonstrate a salient application of amortized variational inference with a neural net parametrization. Finally, Bayesian Neural Networks are introduced and the tutorial concludes with a backpropagation friendly variational inference algorithm to perform probabilistic reasoning on feedforward neural networks.

1 Information Theory

Information theory defines precise limits on how much information can be communicated between any two components of any system. Information is usually measured in *bits*, and one bit of information allows you to choose between two equally probable alternatives. For simplicity, assume you have n forks (allows you to go right or left) on the road, leading to $m = 2^n$ final destinations. To reach any destination, you need n bits of information in order to choose one out of the m equally probable alternatives. It is key to contrast bit from a binary digit - A binary digit is the value of a binary variable, whereas a bit is an amount of information.

Consider a coin that lands heads $90\%(P(x_h) = 0.9)$ of the times. If we are to predict, we expect it to land heads up and are surprised if the coin lands tail. This *surprise* of outcome is quantified by Shannon Information

$$\text{Shannon Information}(x_h) = -\log p(x_h)^1 \tag{1}$$

1.1 Entropy

- *Entropy* of p(X) - $H(X)$ is defined as the expected *surprise* over the entire set of possible values for the random variable.

$$H(X) = -\mathbb{E}_X \log p(x) = -\sum_x p(x) \log p(x) \quad \text{or} \quad -\int_x p(x) \log p(x) dx \tag{2}$$

[1] In this section, all log are taken to the base 2 since it operates on bits.

If the entropy of a random variable X is represented by entropy H(X), it means that X could be used to represent $2^{H(X)}$ equi-probable values. For instance, a fair coin's entropy is 1 and X could represent two equi-probable values - heads and tails. Translating entropy into an equivalent number of equi-probable values serves as a key to understand the amount of uncertain information represented by a random variable. Similarly, roll of a dice has an entropy of ~ 2.58 which means this random variable could effectively represent 6 equi-probable values.

1.2 Entropy Measures of 2 Random Variables

1. ***Joint Entropy***, $H(X, Y)$ is the entropy of the random variable pair X and Y

$$H(X,Y) = -\mathbb{E}_{X,Y} \log p(X,Y) = -\sum_{x,y} p(x,y) \log p(x,y) \quad \text{or} -\int_{x,y} p(x,y) \log p(x,y) dxdy \tag{3}$$

2. ***Conditional Entropy*** of X over Y is defined as the expected(over Y) entropy of the conditional distribution $p(X|Y)$. Intuitively, it quantifies how much uncertainty one has over the joint distribution on X, Y provided Y is observed.

$$\begin{aligned} H(X|Y) &= -\mathbb{E}_Y H(X|Y) = -\sum_y p(y) \sum_x p(x|y) \log p(x|y) \\ &= -\sum_{x,y} p(x,y) \log p(x|y) \quad \text{or} \quad -\int_{x,y} p(x,y) \log p(x,y) dxdy \\ H(X|Y) &= H(X,Y) - H(Y) \end{aligned} \tag{4}$$

3. ***Mutual Information*** is the amount of information that can be obtained about one random variable by observing another - useful when we approximate any distribution with another known distribution.

$$\begin{aligned} I(X;Y) &= \mathbb{E}_{x,y} \log \frac{p(x,y)}{p(x)p(y)} \\ I(X;Y) &= I(Y;X) \quad \text{(symmetric)} \\ &= H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y) \end{aligned} \tag{5}$$

   Knowing Y, we can save an average of I(X; Y) bits in encoding X compared to not knowing Y. Distributions that characterize similar random variables tend to have a high mutual information.

1.3 KL Divergence

Kullback-Leibler (KL) Divergence is a measure of difference between two probability distributions which is defined as follows.

$$D_{KL}(p(X)||q(X)) = -\mathbb{E}_P \log \frac{q(X)}{p(X)} = -\sum_x p(x) \log \frac{q(x)}{p(x)} \quad \text{or} \quad -\int_x p(x) \log \frac{q(x)}{p(x)} dx \tag{6}$$

Throughout the tutorial, we use KL, D_{KL} inter-changably to denote KL-divergence. Note that KL divergence is not symmetric and does not satisfy triangle inequality and hence it is not true

distance metric, but rather a quasi distance measure. We can also prove that $D_{KL}(p||q) \geq 0$. Two identical distributions will have KL-divergence of 0 and very different distributions have higher values. KL-divergence can be intuitively thought of us as additional *surprise* that one incurs when one thinks that a random variable follows a distribution q when the true underlying distribution of the random variable is p.

2 Variational Inference

2.1 Definition

Variational Inference(VI) is a technique widely used to approximate hard-to-infer posterior probability estimates. Consider a general problem where we have a joint density of latent variables $\mathbf{z} = z_{1:m}$ and observations $\mathbf{x} = x_{1:n}$. The inference problem is to compute the conditional density of the latent variables given the observations $p(\mathbf{z}|\mathbf{x})$. This conditional can be used to produce point or interval estimates of the latent variables, form predictive densities of new data, and more.

$$p(\mathbf{z}, \mathbf{x}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$$

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})} \tag{7}$$

In many practical Bayesian inference problems, the denominator in Equation 7 is hard to compute (since we do not have a closed form for the evidence $p(\mathbf{x}) = \int p(\mathbf{z}, \mathbf{x}) d\mathbf{z}$)[2]. We have to often resort to sampling based inference methods such as MCMC procedures (covered extensively in AM207) or approximating the distributions via another distribution which lays the foundation for variational inference.

Variational inference transforms this problem as an optimization problem. First we posit a *family* of approximate densities $\mathcal{Q}$. We then approximate the final posterior using the optimized member of the variational family - $q^*(.)$

$$q^*(\mathbf{z}) = \arg\min_{q \sim \mathcal{Q}} \mathrm{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) \tag{8}$$

Key trick in VI is to choose a family $\mathcal{Q}$ expressive enough to approximate many complicated posteriors yet simple enough to allow for efficient optimization.

2.2 Evidence Lower Bound (ELBO)

Let us decompose the KL-divergence term to find a lower bound (ELBO) for our optimization objective in Equation 8

$$\begin{aligned} \mathrm{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) &= \mathbb{E}_{\mathbf{z}\sim q} \log q(\mathbf{z}) - \mathbb{E}_{\mathbf{z}\sim q} \log p(\mathbf{z}|\mathbf{x}) \\ &= \underbrace{\mathbb{E}_{\mathbf{z}\sim q} \log q(\mathbf{z}) - \mathbb{E}_{\mathbf{z}\sim q} \log p(\mathbf{z}, \mathbf{x})}_{\text{(a) — -1*ELBO}} + \underbrace{\log p(\mathbf{x})}_{\text{(b)}} \\ &= -\mathrm{ELBO}(q) + \log p(\mathbf{x}) \end{aligned} \tag{9}$$

[2] [1] has examples of some common hard to estimate evidence distribution examples

Observe in Equation 9 that (b) does not depend on the variational family over which we optimize and hence maximizing the ELBO term amounts to minimizing our overall optimization objective i.e. KL term. The reason the term is called the ELBO is because it lower bounds the evidence i.e. $p(\mathbf{x}) \geq \text{ELBO}(q)$ since $\text{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) \geq 0$.

We further re-write ELBO as follows[3].

$$\begin{aligned} \text{ELBO}(q) &= \mathbb{E} \log p(\mathbf{z}, \mathbf{x}) - \mathbb{E} \log q(\mathbf{z}) \\ &= \mathbb{E} \log p(\mathbf{z}) + \mathbb{E} \log p(\mathbf{x}|\mathbf{z}) - \mathbb{E} \log q(\mathbf{z}) \\ &= \mathbb{E} \log p(\mathbf{x}|\mathbf{z}) - \text{KL}(q(\mathbf{z})||p(\mathbf{z})) \end{aligned} \tag{10}$$

Note that the first term in Equation 10 encourages placing mass over latent variable configurations(drawn from our variational approximations) that best explain evidence and the second term forces us to choose a member of the variational family that is close to the true prior and this tradeoff is handled by the optimization objective.

2.3 Mean Field Variational Family and Optimization with CAVI

A *mean field variational family* is characterized by the assumption that the latent variables are mutually independent and each one is governed by a distinct factor in the variational density.

$$q(\mathbf{z}) = \prod_{i=1}^{m} q_i(z_i) \tag{11}$$

Each of the variational factor $q_i(z_i)$ can take any parametric form appropriate to the corresponding variable. Typically, this approximation does not contain the true posterior because the latent variables in its true structure could be dependent. Observe in the defintion of KL term in Equation 6 that the divergence measure penalizes a lot more when q places higher probability mass in regions where p has a lower probability mass but penalty is much lesser when it does otherwise. Hence, a good mean field approximation will ensure that the former case of mismatch does not occur drastically while keeping the variational approximation factors independent.

Coordinate ascent is a common optimization algorithm for convex objectives. Coordinate Ascent Variational Inference (CAVI) iteratively optimizes each factor of the mean-field variational density while holding the others fixed, effectively climbing the ELBO uphill to a local maximum. The proof and the exact coordinate ascent algorithm is described in detail in [1]. Finally, we also would like to note that there are a general form for models in which the coordinate updates in mean field variational inference are easy to compute and lead to closed-form updates - *exponential family conditionals* which has the following functional form

$$p(z_j|z_{-j}, \mathbf{x}) = h(z_j) \exp\{\eta(z_{-j}, \mathbf{x})^T t(z_j) - a(\eta(z_{-j}, \mathbf{x})\} \tag{12}$$

where z_{-j} indicates keeping all the other elements of $\mathbf{z}$ except z_j fixed and η, h, a, t are functions that parametrize several powerful exponential family distributions such as Normal, Gamma, Exponential, Bernouilli, Dirichlet, Categorical, Beta, Poisson, Geometric, etc. If we choose our local variational approximation $q(z_j)$ to be the same as the conditional distribution (i.e. in an exponential family), then we will see that the CAVI update step yields an optimal $q(z_j)$ in the same family.

[3]All expectations are over q

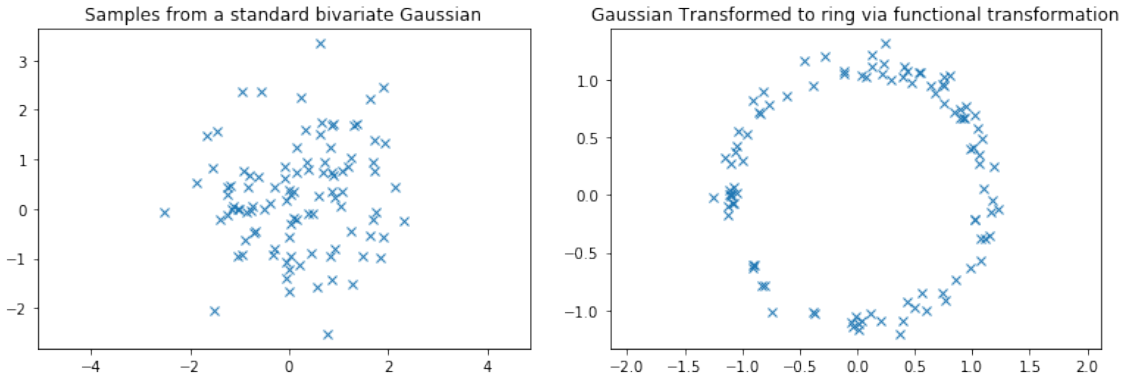

Figure 1: Given a random variable Z with one distribution (on the left - standard bivariate Gaussian), we can always create another random variable X = g(Z) with an entirely different distribution through appropriate functional transformation(on the right. $g(z) \rightarrow z/10 + z/||z||$. In VAE, we are looking to learn such transformations that allow us to generate arbitrary generative distributions sampling from simple standard MVN distribution.

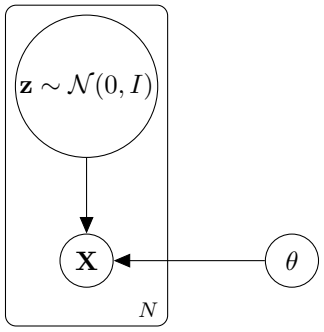

Figure 2: Graphical Model of VAE

3 Variational Autoencoders

3.1 Introduction and Loss Function

Generative models can be roughly defined as models that attempt to learn the joint distribution that generated the evidence data we have at our disposal. In many generative models, latent variables are used to model *unobserved* characteristics of our data. For instance, in a generative model that produces handwritten digits between 0 and 9, the latent variables need to implicitly learn the digit's class, the slope of the letter, curvature etc. Since it is hard to explicitly define these characteristics, we would want the model to automatically learn latent configurations that maximizes its generative performance. Let us define a function family $f(z;\theta)$ parametrized by a vector θ from its parameter space Θ such that $f : \mathcal{Z} \times \Theta \rightarrow \mathcal{X}$. In other words, for a given θ, $f(\mathbf{z};\theta)$ is a random variable in the space $\mathcal{X}$. Then, we define a normal distribution for our conditional generation model as $p(\mathbf{x}|\mathbf{z};\theta) \sim \mathcal{N}(f(\mathbf{z};\theta), \sigma^2 * I)$. To make a rigorous problem statement, we are aiming to maximize

the probability of the training set under the entire generative process as follows

$$\mathbf{x} = \int p(\mathbf{x}|\mathbf{z};\theta)p(\mathbf{z})d\mathbf{z} \tag{13}$$

We need to define $p(z)$ which effectively encodes all *unobserved* latent information. VAE tackles this problem by assuming an extremely simple $p(z)$ - standard MVNs i.e. $p(\mathbf{z}) = \mathcal{N}(0, I)$. We can see from a dummy example in Figure 1 that we can transform simple MVNs to any arbitrary distributions via appropriate functional transformations. VAEs then rely on transformation function $f(\mathbf{z};\theta)$ to produce distributions close to our evidence distribution $p(\mathbf{x})$. VAE parametrizes this function with neural networks which are known to be universal function approximators, parametrized by θ.

Performing inference in a VAE i.e. $p(\mathbf{z}|\mathbf{x})$ again is computationally intractable since equation 13 requires exponential time to compute a closed form solution since we have to integrate over all possible latent configurations (Same reasoning we had when we introduced variational inference). So we introduce variational distribution from the Gaussian Family $q(z|X;\lambda)$ parametrized by λ and transform our problem into an ELBO optimization problem whose loss function is as follows.

$$\mathcal{L}(\mathbf{x};\theta,\lambda) = D_{KL}(q(\mathbf{z}|\mathbf{x};\lambda)||p(\mathbf{z}) - \mathbb{E}_{\mathbf{z}\sim q}\log p(\mathbf{x}|\mathbf{z};\theta) \tag{14}$$

Even though we have not explicitly introduced any code space, encoding or decoding into our formulation, this form of variational loss has another natural interpretation, thus giving VAE the name of variational *autoencoder*. q serves the purpose of being an encoder network (λ) and p ends up being the decoder network (θ) that aims to reconstruct $\mathbf{x}$ based on the encoding $\mathbf{z}$.

3.2 Inference via Optimization - Sampling and Reparametrization

In standard VAE, we are going to use a Gaussian Variational Family since it is easy to calculate KL divergence of two Gaussians. Let us define $q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mu(\mathbf{x};\lambda_\mu), \Sigma(\mathbf{x};\lambda_\Sigma))$. The first term of Equation 14 can be written as

$$D_{KL}((\mathcal{N}(\mu(X), \Sigma(X)||\mathcal{N}(0, I)) = \frac{1}{2}\Big(\mathrm{Tr}(\Sigma(X)) + (\mu(X))^T(\mu(X)) - k - \log\det(\Sigma(X))\Big) \tag{15}$$

The second term of Equation 14 is a bit more tricky to compute since we need to compute expectations via taking multiple samples. A trick to make it computationally easy is grounded in empirical evidence - We take ***one*** sample of $\mathbf{z}$ and treat $p(\mathbf{x}|\mathbf{z})$ as an approximation for the expectation term. Thus, once we sample a $\mathbf{z}$ our loss function essentially becomes $\mathcal{L}_{stochastic} = D_{KL}(q(\mathbf{z}|\mathbf{x};\lambda)||p(\mathbf{z}) - \log p(\mathbf{x}|\mathbf{z};\theta)$. Now we are set to take the gradient with respect to our parameters we intend to learn $\lambda_\mu, \lambda_\Sigma$.

3.2.1 Reparametrization Trick

Yet, we have another roadblock before we can finally pass our gradients through the computational graph. Figure 3 on the left, shows why backpropagation will not be possible in the current setting where we are required to back-propagate through a layer that samples $\mathbf{z}$ from $q(\mathbf{z}|\mathbf{x})$ and sampling operation is a non-continuous operation and hence breaks the gradient computational graph. We use the "reparametrization" trick which is to move the sampling to an input layer as seen in the

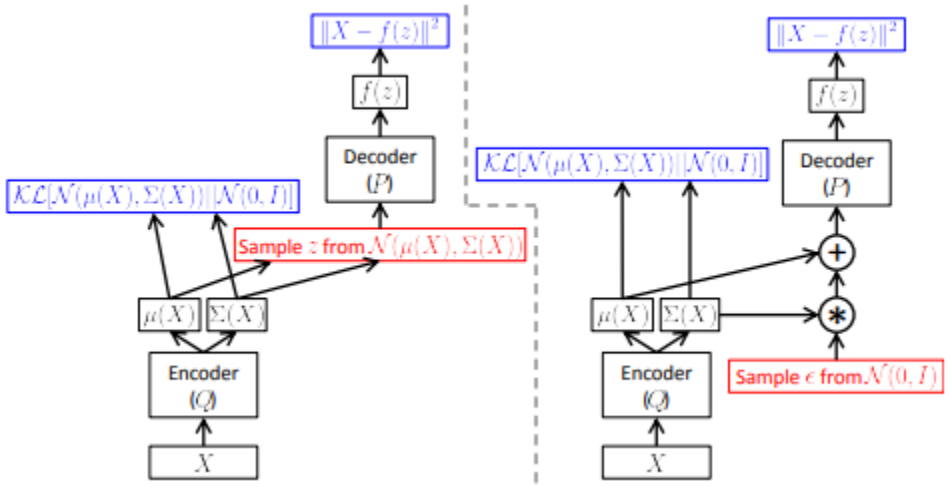

Figure 3: A training-time variational autoencoder implemented as a feedforward neural network, where $p(\mathbf{X}|\mathbf{z})$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but backpropagation can be applied only to the right network. Image Credits : [2]

right of Figure 3. Given $\mu(\mathbf{X})$ and $\Sigma(\mathbf{X})$—the mean and covariance of $Q(\mathbf{Z}|\mathbf{X})$—we can sample from $\mathcal{N}(\mu(\mathbf{X}), \Sigma(\mathbf{X}))$ by first sampling $\epsilon\mathcal{N}(0, I)$, then computing $\mathbf{z} = \mu(\mathbf{X}) + \Sigma^{\frac{1}{2}}(\mathbf{X})\epsilon$. Thus, the equation we actually take the gradient of is given by

$$\begin{aligned}\mathcal{L}(\mathbf{X}) = \frac{1}{2}\Big(\mathrm{Tr}(\Sigma(\mathbf{X})) + (\mu(\mathbf{X}))^T(\mu(\mathbf{X})) - k - \log\det(\Sigma(\mathbf{X}))\Big) - \\ \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)} \log p\Big(\mathbf{X}|\mu(\mathbf{X}) + \Sigma^{\frac{1}{2}}(\mathbf{X}) * \epsilon\Big)\end{aligned} \tag{16}$$

Note that VAE performs "amortized" variational inference i.e. the variational parameters are shared across datapoints. Mean field that we saw earlier is strictly more expressive since it has a set variational parameter for each datapoint and does not share any parameters. For computational gains and reduced overfitting of our model, we are limiting the capacity or representational power of our variational family by tying parameters across datapoints (e.g. with a neural network that shares weights and biases across data).

3.2.2 REINFORCE procedure

While "reparametrization" trick is heavily used in standard VAE, there is an essential requirement that we need an entirely differentiable, continuous distributions (like exponential families) which could be reparametrized as we saw earlier. On the other hand REINFORCE procedure works on any arbitrary distribution, discrete or continuous and is a more general approach. "Reparametrization" on the other hand is computationally fast since it is uses backpropagation and produces low variance gradient estimates whereas "REINFORCE" still produces high variance gradient estimates.

Let us define a problem setting mirroring what we had in "reparametrization".

$$\nabla_\theta \, \mathbb{E}_{x \sim p_\theta(x)}[f(x)]$$

It mainly uses the concept of "score function" estimator(Remember policy gradient approaches from Deep RL section) which implicitly uses a simple differentiation identity

$$\nabla_\theta p_\theta(x) = p_\theta(x) \nabla \log p_\theta(x) \tag{17}$$

The term $\nabla \log p_\theta(x)$ is called the score and regularly comes up in maximum likelihood estimation. It also has many properties like having zero expected value (which proves useful when using it for variational inference among other things). Continuing our derivation, we get

$$\begin{aligned}
\nabla_\theta \, \mathbb{E}_{x \sim p_\theta(x)} \, f(x) &= \nabla_\theta \int f(x) p_\theta(x) dx \\
&= \int f(x) \nabla_\theta p_\theta(x) dx \quad \text{(Leibniz Rule)} \\
&= \int f(x) p_\theta(x) \nabla \log p_\theta(x) dx \quad \text{(Equation 17)} \\
&= \mathbb{E}_{x \sim p_\theta(x)}[f(x) \nabla \log p_\theta(x)] \\
&\approx \frac{1}{N} \sum_{i=1}^{N} f(x_i) \nabla \log p_\theta(x_i) \quad \text{(Monte-Carlo Estimates)}
\end{aligned} \tag{18}$$

Now we have transformed our derivative operation into something tractable using the score function.Note that the above is an unbiased estimator of the gradients. The score function estimator assumes it is possible to cheaply sample from the distribution $p_\theta(x)$. It is key to note that REINFORCE places no restriction on the nature of the function and it doesn't even need to be differentiable for us to estimate the gradients of its expected value. The unbiased estimates also mean that the variance for these gradients are very high. To counter this, advanced approaches use techniques such as control variates, importance sampling etc. (beyond the scope of our section. For these variation reduction techniques, please refer this link)

3.3 Conditional VAE(CVAE)

Conditional VAEs are useful in settings where the model is not purely generative from a latent space, but rather takes in an input and enriches it with a latent code to produce an output. A standard application might be image completion for instance, where one is given an incomplete image and the output is expected to be the completed image. The space of plausible outputs in this case is multi-modal - there are many possibilities for the next digit or the extrapolated pixels. The math of CVAE is largely similar to VAE except that we condition the entire generative process on the input in addition. Here we use $\mathbf{X}$ for input and $\mathbf{Y}$ for output. The loss function in case of CVAE will look as follows

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}; \lambda, \theta) = D_{KL}(q(\mathbf{z}|\mathbf{Y}, \mathbf{X}; \lambda)||p(\mathbf{z}|\mathbf{X}) - \mathbb{E}_{\mathbf{z} \sim q(.|\mathbf{X}, \mathbf{Y})} \log p(\mathbf{Y}|\mathbf{z}, \mathbf{X}; \theta) \tag{19}$$

Figure 4 shows how inference is performed in the network (slightly) differently during train and test time. Finally, we see results from VAE and CVAE on MNIST data in Figures 7 and 6 in Appendix.

@PAVLOS: There is a proof that if given arbitrarily powerful learners, VAEs should have near 0 approximation errors - Slightly deep probability math. Is it necessary ?

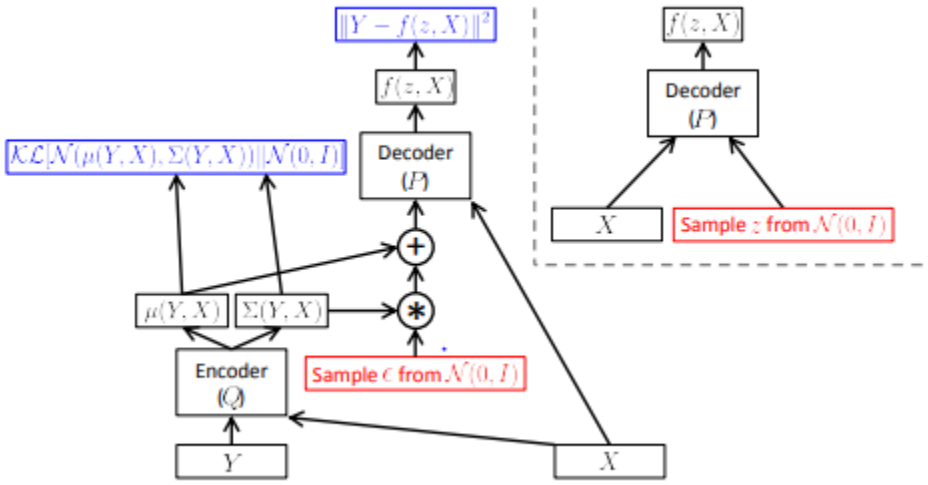

Figure 4: Left: a training-time CVAE implemented as a feedforward neural network. Right: the same model at test time, when we want to sample from P(Y|X). Credits : [2]

4 Bayesian Neural Networks

Remember L2 regularization or ridge regression from CS109a. These regularized models can also be viewed from a Bayesian perspective where we assume a prior over our weights and on observing the data, we perform posterior updates over the parameters. For instance, in L2 regularization, we assume a prior of $p(\mathbf{w}) = \mathcal{N}(0, \sigma * I)$ and with $\mathbb{L}$ as our likelihood function, our Bayesian posterior update and the MAP $\mathbf{w}^*$would be

$$\begin{aligned} p(\mathbf{w}|\mathbf{x}, \mathbf{y}) &\propto \mathbb{L}\Big(\mathbf{y}_{1:n}|\mathbf{x}_{1:n}, ; \mathbf{w}\Big) * p(\mathbf{w}) \\ \mathbf{w}^* &= \arg\max_{\mathbf{w}} p(\mathbf{w}|\mathbf{x}, \mathbf{y}) \end{aligned} \tag{20}$$

Plain feedforward neural networks are prone to overfitting. When applied to supervised or reinforcement learning problems these networks are also often incapable of correctly assessing the uncertainty in the training data and so make overly confident decisions about the correct class, prediction or action. In Bayesian Neural Nets (BNN), we introduce a prior over every weight of the neural network, effectively learning a posterior over the weights via inference. Three major advantages of this approach is - richer representations and predictions from cheap model averaging, regularization via a compression cost on the weights and learning confidence of predictions.

4.1 Problem Setup and Bayes by Backpropagation(BBB)

Throughout our BNNs, we are going to function in a discriminative setting away from the generative setting we witnessed with VAEs. Let us define our training data as $\mathcal{D} \rightarrow (\mathbf{x}_{1:n}, \mathbf{y}_{1:n})$.We view a

neural network as a probabilistic model $p(\mathbf{y}|\mathbf{x}, \mathbf{w})$ - given an input $\mathbf{x} \in^p$,a BNN assigns a probability to each possible output $y\mathcal{Y}$, using the set of parameters or weights. Transformation of inputs onto the output space $\mathcal{Y}$ is achieved by feedforward nets - a combination of linear layers with non-linear activations.

Bayesian inference for neural networks calculates the posterior distribution of the weights given the training data,$P(\mathbf{w}|\mathcal{D})$. The predictive distribution of an unknown label $\hat{y}$ of a test data item $\hat{\mathbf{x}}$, is given by $P(\hat{y}|\hat{\mathbf{x}}) = \mathbb{E}_{p(\mathbf{w}|\mathcal{D})}\, P(\hat{y}|\hat{\mathbf{x}}, \mathbf{w})$. Every possible configuration of weights gives us an estimate of the output and weighing them in expectation with respect to the posterior is equivalent to using an ensemble of an infinite number of neural networks. To get over this computational intractability of inference, we once again resort to variational inference.Variational learning finds the parameters θ of a distribution on the weights $q(\mathbf{w}|)$ that minimizes the Kullback-Leibler (KL)divergence with the true Bayesian posterior on the weights.

$$\begin{aligned}\theta^* &= \arg\min_\theta D_{KL}(q(\mathbf{w}|\theta)||p(\mathbf{w}|\mathcal{D}) \\ &= \arg\min_\theta \underbrace{D_{KL}\big[q(\mathbf{w}|\theta)||p(\mathbf{w})\big] - \mathbb{E}_{q(\mathbf{w}|\theta)} \log p(\mathcal{D}|\mathbf{w})}_{\mathcal{L}(\mathcal{D},\theta)} \quad \text{(derived similar to VI)}\end{aligned} \tag{21}$$

Since we usually operate with Gaussian variational family and Gaussian prior on weights $p(\mathbf{w}) = \mathcal{N}(0, \sigma * I)$, we usually have a closed form expression for our KL-divergence. The second term on the other hand requires reparametrization and Monte-Carlo Sampling as we saw in VAEs in order to make the entire procedure backpropagation-friendly, thus giving the procedure the name Bayes-by-Backpropagation. Thus, the overall training procedure could be summarized in the following steps. Remember $\mathbf{w} = \mu + \log(1 + \exp(\rho)) * \epsilon$

1. Sample $\epsilon \sim \mathcal{N}(0, I)$
2. Let $\mathbf{w} = \mu + \log(1 + \exp(\rho)) * \epsilon$. Here we use the "softplus" trick i.e. $\sigma = \log(1 + \exp(\rho))$ in order to have a smooth transformation over the entire space.
3. Let $\theta = (\mu, \rho)$ and $f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log p(\mathbf{w})p(\mathcal{D}|\mathbf{w})$
4. Calculate gradients

$$\begin{aligned}\nabla f_\mu &= \frac{\partial f}{\partial \mathbf{w}} + \frac{\partial f}{\partial \mu} \\ \nabla f_\rho &= \frac{\partial f}{\partial \mathbf{w}} \frac{\epsilon}{\log(1 + \exp(-\rho))} + \frac{\partial f}{\partial \rho}\end{aligned} \tag{22}$$

5. Update variational parameters via a gradient step using any optimizer.

Figure 5 shows how performing inference with a Bayesian Neural Network differs fro performing inference with a standard neural net. We see that the BBB network learns confidence intervals that are intuitive - narrow near evidence and wide out of distribution whereas the standard neural network is overconfident, particularly away from evidence points. Since Bayes by Backprop simply uses gradient updates, it is very parallelization friendly and can readily be scaled using multi-machine optimization schemes such as asynchronous SGD.

While BBB is a form of variational inference via reparametrization, there are other forms of inference

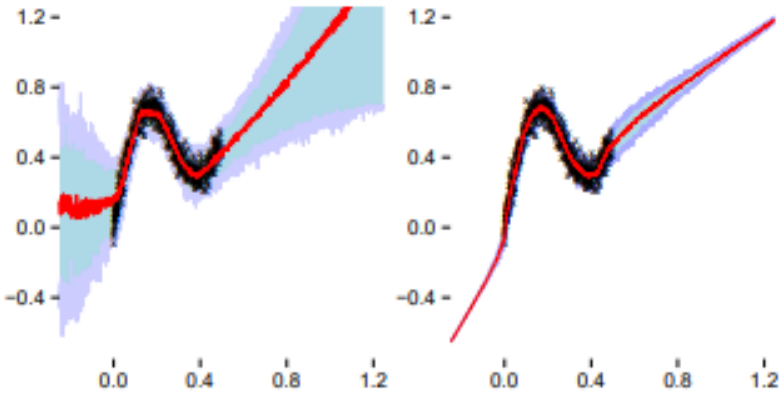

Figure 5: Left : Fit via BBB. Right:Fit via Neural Nets. Red indicates the median prediction. Blue boundaries indicate quartile ranges. Look how BBB is less confident in out of distribution regions and more confident around evidence.Credits :[3]

(not covered in this tutorial) which are common in BNNs such as Black-Box Variational Inference [4] which uses the "REINFORCE" procedure we learned earlier and other MCMC methods such as HMC and its stochastic variants. [5, 6] and [7] that relatively weighs these inference procedures.

Acknowledgements

This tutorial has been prepared for CS209b course : Advanced Data Science and is used for purely instructional purposes only. Much of the material presented in the tutorial has been adapted from a medley of sources - [2, 8, 9] (Variational Autoencoders), [1, 10] (Variational Inference), [3] (BBB) and [11, 12] (Information Theory) and we sincerely thank the authors of these papers/tutorials/websites.

References

[1] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational Inference: A Review for Statisticians," *arXiv e-prints*, p. arXiv:1601.00670, Jan. 2016.

[2] C. Doersch, "Tutorial on Variational Autoencoders," *arXiv e-prints*, p. arXiv:1606.05908, June 2016.

[3] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight Uncertainty in Neural Networks," *arXiv e-prints*, p. arXiv:1505.05424, May 2015.

[4] R. Ranganath, S. Gerrish, and D. M. Blei, "Black Box Variational Inference," *arXiv e-prints*, p. arXiv:1401.0118, Dec. 2013.

[5] "Mcmc using hamiltonian dynamics." `https://arxiv.org/pdf/1206.1901.pdf`. Accessed: 2018-12-23.

[6] T. Chen, E. B. Fox, and C. Guestrin, "Stochastic Gradient Hamiltonian Monte Carlo," *arXiv e-prints*, p. arXiv:1402.4102, Feb. 2014.

[7] "Posterior distribution analysis for bayesian inference in neural networks." `https://pdfs.semanticscholar.org/6197/dbd691037a412b67df688541df7c9ae87c0d.pdf`. Accessed: 2018-12-23.

[8] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv e-prints*, p. arXiv:1312.6114, Dec. 2013.

[9] "What is a variational autoencoder ?." `https://jaan.io/what-is-variational-autoencoder-vae-tutorial/`. Accessed: 2018-12-23.

[10] "Variational inference: Mean field approximation." `https://www.cs.cmu.edu/~epxing/Class/10708-17/notes-17/10708-scribe-lecture13.pdf`. Accessed: 2018-12-23.

[11] J. V. Stone, "Information Theory: A Tutorial Introduction," *arXiv e-prints*, p. arXiv:1802.05968, Feb. 2018.

[12] "Information theory." `https://en.wikipedia.org/wiki/Information_theory`. Accessed: 2018-12-23.

Appendix

A.1. MNIST Results on VAE, CVAE

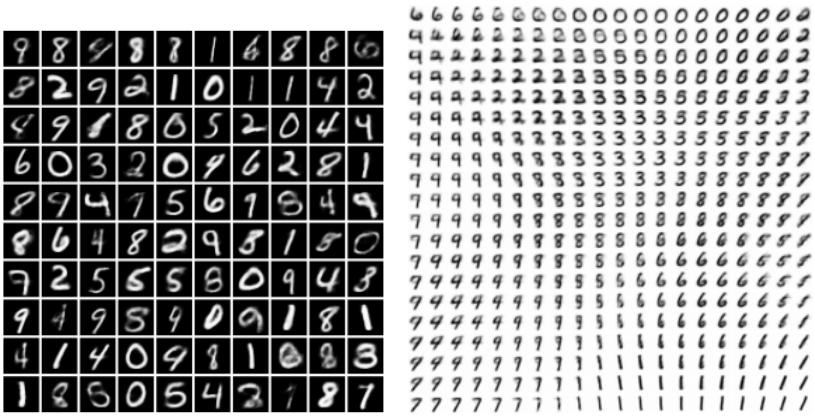

Figure 6: Left: MNIST generative results from VAE. Right : Latent code interpolation - Results generated from sampling latent codes and interpolating between those two codes. Credits : [2, 8]

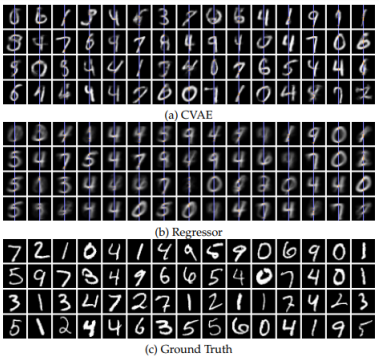

Figure 7: CVAE results comparing CVAE with a simple discriminative regressor with respect to ground truth labels. Because CVAE is generated via learning a good latent space, it performs better in image completion. The inputs(incomplete image) to CVAE are the pixels in the middle column shown in the images in blue. Credits : [2]