Introduction to Bayesian Neural Networks

Yingzhen Li

(Lecture notes for the ProbAI 2022 BNN introduction lecture)

1 What is a Bayesian neural network (BNN)?

In short, a Bayesian neural network (BNN) is a neural network that uses (approximate) Bayesian inference for uncertainty estimation. For example, we can treat the NN parameters as random variables and infer them using (approximate) Bayesian posterior inference.

For a supervised learning task, a Deep learning solution is to fit a neural network $y = f_\theta(x)$ with parameters θ to a dataset $D = \{(x_n, y_n)\}_{n=1}^N$. In particular, DL training corresponds to a Maximum Likelihood Estimation (MLE) of the parameters:

$$\theta^* = \arg\max_\theta \mathbb{E}_{(x,y)\sim D}[\log p(y|x,\theta)], \tag{1}$$

with the likelihood term defined as

$$\text{for regression:} \quad p(y|x,\theta) = \mathcal{N}(y; f_\theta(x), \sigma^2 I), \tag{2}$$

$$\text{for classification:} \quad p(y|x,\theta) = \text{Categorical}(\text{logit} = f_\theta(x)). \tag{3}$$

Often regularizers such as ℓ_2 regularizers are used, in such case it returns a Maximum A Posteriori (MAP) estimate of the parameters.

In Bayesian neural networks (BNN), however, the network parameters θ are treated as random variables, and we perform (approximate) Bayesian inference on it. In detail, we define a prior distribution $p(\theta)$, which leads to a posterior with *Bayes' rule* (under the i.i.d. data setting):

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}, \quad p(D|\theta) = \prod_{n=1}^N p(y_n|x_n,\theta). \tag{4}$$

Assuming we have $p(\theta|D)$ at hand, then in prediction time, given a new test input x^*, we use the following *Bayesian predictive distribution* for predicting the corresponding output:

$$p(y^*|x^*, D) = \int p(y^*|x^*,\theta)p(\theta|D)d\theta. \tag{5}$$

Unfortunately we don't know how to directly compute $p(\theta|D)$ nor $p(y^*|x^*, D)$. This is where *approximate Bayesian inference* comes in, where many of the existing approaches solve the problem in the following 3 steps:

1. Design an approximate posterior: design a distribution family $\mathcal{Q}$ such that for each $q(\theta) \in \mathcal{Q}$, we can compute its density given any θ, and $q(\theta)$ is easy to sample from;
2. Fit the approximate posterior: find the best q distribution in $\mathcal{Q}$ so that $q(\theta) \approx p(\theta|D)$ well according to some criteria;
2. Approximate predictive inference with Monte Carlo: approximate $p(y^*|x^*, D)$ by replacing the exact posterior $p(\theta|D)$ with $q(\theta)$ and estimating the integral with Monte Carlo:

$$p(y^*|x^*, D) \approx \int p(y^*|x^*,\theta)q(\theta)d\theta \approx \frac{1}{K}\sum_{k=1}^K p(y^*|x^*,\theta_k), \quad \theta_k \sim q(\theta). \tag{6}$$

It remains to discuss how to find an approximation $q(\theta) \approx p(\theta|D)$, for which we will mainly discuss the variational inference (VI) approach below.

2 Bayes-by-backprop: Mean-field VI for BNNs

2.1 Foundations of variational inference (VI)

As discussed above, we need to find an approximation $q(\theta) \approx p(\theta|D)$. A natural way to do so is to minimise some divergence measure that tells the difference between $q(\theta)$ and $p(\theta|D)$. In particular, variational inference (VI) uses the KL divergence and finds the best approximate posterior within a distribution family Q (e.g., all Gaussians):

$$q^*(\theta) = \arg\min_{q \in Q} KL[q(\theta)||p(\theta|D)], \quad KL[q(\theta)||p(\theta|D)] = \mathbb{E}_q[\log q(\theta) - \log p(\theta|D)]. \tag{7}$$

However, we cannot directly compute this KL as we don't know how to compute $p(\theta|D)$ in the first place. Fortunately we can re-write the KL to an equivalent objective: notice that by taking the logarithm on the both sides of the Bayes' rule:

$$\log p(\theta|D) = \log \frac{p(D|\theta)p(\theta)}{p(D)} = \log p(D|\theta) + \log p(\theta) - \log p(D), \tag{8}$$

which means (notice that $\log p(D)$ is a constant w.r.t. q and θ):

$$\begin{aligned} KL[q(\theta)||p(\theta|D)] &= \mathbb{E}_q[\log q(\theta) - \log p(D|\theta) - \log p(\theta)] + \log p(D) \\ &= \log p(D) - (\mathbb{E}_q[\log p(D|\theta)] - KL[q(\theta)||p(\theta)]) \\ &:= \log p(D) - ELBO(q, D). \end{aligned} \tag{9}$$

In other words, the below optimisation problems are equivalent:

$$\begin{aligned} &\min_{q \in Q} KL[q(\theta)||p(\theta|D)] \quad \Leftrightarrow \quad \max_{q \in Q} ELBO(q, D), \\ &ELBO(q, D) = \mathbb{E}_q[\log p(D|\theta)] - KL[q(\theta)||p(\theta)]. \end{aligned} \tag{10}$$

One can think about the ELBO as a combination of two terms:

- Data fitting: $\mathbb{E}_q[\log p(D|\theta)]$ measures on average how good neural networks with parameters sampled from q fit the training data.
- Complexity regularization: $KL[q(\theta)||p(\theta)]$ describes the amount of changes of q from the prior p. In BNN literature the prior p on weights are often set to be less informative (e.g., Gaussian with zero mean and large variance), in such case the KL term can also be viewed as regularizing the complexity of q.

These interpretations also leads to a "tempered ELBO" that is often used in practice:

$$ELBO_\beta(q, D) = \mathbb{E}_q[\log p(D|\theta)] - \beta KL[q(\theta)||p(\theta)], \tag{11}$$

which strives to balance between the data fitting quality and the complexity of q.

2.2 Bayes-by-backprop: Mean-field VI for BNNs

Now let's discuss MFVI-BNN (also named Bayes-by-backprop) [Blundell et al., 2015] as a specific example. In this case we define the Q family as all possible *fully factorised distributions*. In detail, assume $\theta = \{(W^l, b^l)\}_{l=1}^L$ for an L-layer BNN. Then a fully factorised approximation looks like

$$q(\theta) = \prod_{l=1}^{L} q(W^l)q(b^l), \quad q(W^l) = \prod_{ij} q(W^l_{ij}), \quad q(b^l) = \prod_i q(b^l_i). \tag{12}$$

Assuming Gaussian distrbutions further:

$$q(W^l_{ij}) = \mathcal{N}(W^l_{ij}; M^l_{ij}, V^l_{ij}), \quad q(b^l_i) = \mathcal{N}(b^l_i; m^l_i, v^l_i). \tag{13}$$

Therefore the variational parameters to be optimised are the mean parameters $\{M_{ij}^l, m_i^l\}$ and variance parameters of the q distribution. Note that as variance is positive, one cannot directly optimize w.r.t. $\{V_{ij}^l, v_i^l\}$ without constraints. Instead in practice we often define e.g., $\log V_{ij}^l$ as a free-parameter to be optimized. Collecting all the parameters into vector and matrix forms, we have

$$\text{Variational parameters of mean-field Gaussian:} \quad \{M^l, m^l, \log V^l, \log v^l\}_{l=1}^L, \tag{14}$$

where the logarithm in above equation is applied element-wise.

We now turn to the optimization of the ELBO. If factorised and isotropic Gaussian prior is used for θ, then we can compute the KL regularizer as

$$KL[q(\theta)||p(\theta)] = \sum_{l=1}^{L} KL[q(W^l)||p(W^l)] + KL[q(b^l)||p(b^l)], \tag{15}$$

where $KL[q(W^l)||p(W^l)]$ and $KL[q(b^l)||p(b^l)]$ are KL divergences between two factorised Gaussians, which have analytic solutions.

For the data-fit term $\mathbb{E}_q[\log p(D|\theta)]$, the expectation remains intractable since $p(D|\theta)$ is defined using neural networks (i.e., non-linear transform of θ). Also for optimization we need to be able to compute the gradient of this data fit term w.r.t. the mean and variance parameters of $q(\theta)$. This problem is solved using *Monte-Carlo sampling* and the *reparameterisation trick* [Kingma and Welling, 2013]. In detail, for a Gaussian distribution $q(\theta) = \mathcal{N}(\theta; \mu, \text{diag}(\sigma^2))$, getting a sample $\theta_k \sim q$ is computed as follows:

$$\theta_k = \mu + \sigma \odot \epsilon_k, \quad \epsilon_k \sim \mathcal{N}(0, I), \tag{16}$$

with $\odot$ denoting the element-wise product. This means we can estimate the data fit term with Monte-Carlo as follows, where $\mu = \{M^l, m^l\}_{l=1}^L$ and $\sigma^2 = \{V^l, v^l\}_{l=1}^L$:

$$\mathbb{E}_q[\log p(D|\theta)] \approx \frac{1}{K} \sum_{k=1}^{K} \log p(D|\mu + \sigma \odot \epsilon_k), \quad \epsilon_k \sim \mathcal{N}(0, I). \tag{17}$$

Lastly, when the dataset D contains many datapoints, one might want to run mini-batch training, i.e., stochastic gradient descent (SGD). This is possible for VI: notice that under the i.i.d. data setting,

$$\log p(D|\theta) = \sum_{n=1}^{N} \log p(y_n|x_n, \theta) = N\mathbb{E}_{(x,y)\sim D}[\log p(y|x, \theta)]. \tag{18}$$

This means we can estimate $\mathbb{E}_{(x,y)\sim D}[\log p(y|x, \theta)]$ using a mini-batch of the data: assume that $(x_1, y_1), ..., (x_m, y_m) \sim D$:

$$\mathbb{E}_{(x,y)\sim D}[\log p(y|x, \theta)] \approx \frac{1}{M} \sum_{m=1}^{M} \log p(y_m|x_m, \theta). \tag{19}$$

Combining all the sampling estimate together, we can compute the ELBO objective as follows, using $K = 1$ Monte-Carlo sample for θ:

$$ELBO_\beta(q, D) \approx \frac{N}{M} \sum_{m=1}^{M} \log p(y_m|x_m, \mu + \sigma \odot \epsilon) - \beta KL[q(\theta)||p(\theta)], \quad \epsilon \sim \mathcal{N}(0, I). \tag{20}$$

3 Other choices of approximate posterior

Given the practical issues of MFVI in calibrated uncertainty estimation, a natural question is to ask whether other design of the q distribution would return better results. Within the Gaussian family, Gaussians with full-rank covariance matrices are more expressive than factorised Gaussians.

But at the same time, it requires much more variational parameters if we were to use full covariance Gaussians for every layer. For example, a hidden layer with both input and output dimensions as 50 would need $50 \times 50 = 2500$ parameters for parameterising the mean, but $\sum_{i=1}^{50\times 50} i = 3126250$ parameters for parameterising the (symmetric) full covariance matrix! Therefore, when selecting the q distribution family, one needs to also consider the computational & memory costs for such approximation.

In below we will discuss 2 more "economic" solutions:

1. The so-called "last-layer BNN" approach: only apply full-covariance Gaussian posterior approximation to the last layer of the network, and use MLE/MAP solutions for the previous layers.

2. Monte Carlo dropout (MC-dropout): adding dropout layers to the network, and in test time, run multiple forward passes with dropout.

3.1 Last-layer BNN with full covariance Gaussian approximation

For this "last-layer BNN" approach, the q distribution is defined differently for different layers:

$$\begin{aligned} q(W^l, b^l) &= \delta(W^l = M^l, b^l = m^l), \quad l = 1, ..., L, \\ q(\theta^L) &= \mathcal{N}(\theta^L; \mu^L, \Sigma^L), \quad \theta^L = \{W^L, b^L\}, \mu^L = \{M^L, m^L\}. \end{aligned} \tag{21}$$

In other words, we use deterministic network layers for all but the last layer, and for the last layer we use a Gaussian approximation with a full-rank covariance matrix. The corresponding ELBO objective then becomes

$$ELBO_\beta(q, D) = \mathbb{E}_q[\log p(D|\theta)] - \beta KL[q(\theta^L)||p(\theta^L)], \tag{22}$$

which in practice means we only sample the last layer's weight and compute the KL regulariser for the last layer.

In regression tasks with Gaussian likelihood, this approach can be viewed as performing Bayesian linear regression but with non-linear input feature computed by previous neural network layers. So this means, given fixed parameters $\theta^{1:L-1} = \{M^l, m^l\}_{l=1}^{L-1}$ for all the previous layers, the variational parameters $\{\mu^L, \Sigma^L\}$ can be analytically calculated, if we use full batch for training. In practice we may still prefer optimising the ELBO to find the optimal Gaussian posterior approximation for the last layer, which can leverage SGD-based optimisation methods, and this approach can be applied to other cases such as classification (thus more general).

3.2 Monte Carlo dropout

For the Monte Carlo dropout (MC-dropout) method [Gal and Ghahramani, 2016] with dropout probability π, the q distribution is a mixture of delta measures. Specifically, for a layer with parameters $\{W^l, b^l\}, W^l \in \mathbb{R}^{d_{out}\times d_{in}}, b^l \in \mathbb{R}^{d_{out}}$, the q distribution is

$$\begin{aligned} q(W^l, b^l) &= \prod_{i=1}^{d_{out}} q(W_i^l, b_i^l) \\ q(W_i^l, b_i^l) &= \pi\delta(W_i^l = 0, b_i^l = 0) + (1-\pi)\delta(W_i^l = M_i^l, b_i^l = m_i^l), \end{aligned} \tag{23}$$

with variational parameter for the layer as $\{M^l, m^l\}, M^l \in \mathbb{R}^{d_{out}\times d_{in}}, m^l \in \mathbb{R}^{d_{out}}$. This means there are two equivalent way to compute $\mathbb{E}_q[\log p(y|x, \theta)]$ with Monte Carlo: for the forward pass of a layer, below computations are equivalent:

1. drop weights: sample $W^l, b \sim q(W^l, b^l)$, then compute $y = x(W^l)^T + b^l$;

2. drop units: compute $\hat{y} = x(M^l)^T + m^l$, then apply dropout $y = \text{dropout}(\hat{y}; \pi)$.

The $KL[q||p]$ regularizer for MC-dropout is ill-defined for Gaussian prior $p(\theta)$. In practice this is replaced by an ℓ_2 regularizer, i.e., $\frac{1-\pi}{2\sigma^2_{\text{prior}}}||M^l||_2^2$ for the weight variational parameter M^l (and similarly for m^l). The intuition is the following: the q distribution used in MC-dropout is the limiting distribution of the following mixture of Gaussian distribution:

$$q(\theta_i^l) = \lim_{\eta \to 0} \pi \mathcal{N}(\theta_i^l; 0, \eta I) + (1-\pi)\mathcal{N}(\theta_i^l; \mu_i^l, \eta I), \quad \theta_i^l = \{W_i^l, b_i^l\}, \mu_i^l = \{M_i^l, m_i^l\}. \tag{24}$$

This permits a valid KL divergence $KL[q(\theta_i^l)||p(\theta_i^l)]$ if using Gaussian prior and $\eta > 0$, which includes a term that approximately equals to $\frac{1-\pi}{2\sigma^2_{\text{prior}}}||M^l||_2^2$ for the weight variational parameter M^l (and similarly for m^l). The other terms in that KL regulariser depends on η which will diverge to infinity when $\eta \to 0$, and in MC-dropout those terms are dropped. In other words, the q distribution in MC-dropout is improper in the sense that approximating a continuous variable posterior distribution with "mixture of delta measures" results in infinite KL which is poor. But in practice MC-dropout can still provide quick posterior approximates (especially in function space) and it has shown to provide useful uncertainty information in downstream tasks.

4 Case study 1: Bayesian optimisation with VI-BNNs

4.1 What is Bayesian optimisation (BO)?

Assume you are interested in optimising a function

$$x^* = \arg\max_{x \in \mathcal{X}} f_0(x), \tag{25}$$

but you don't actually know the analytic form of $f(x)$. Rather, you only have access of it as a "black-box", where you provide an input x to this "black-box", and it will return you a (noisy version of) output $y = f_0(x) + \epsilon$.

Bayesian optimisation (BO) [Snoek et al., 2012] is a class of methods that tackle this challenge.

To motivate BO, let's imagine you already have a set of datapoints $D = \{(x_n, y_n)\}_{n=1}^N$ collected by sending the queries $x_1, ..., x_N$ to the above mentioned "black-box". Then we can fit a *surrogate function* $f_\theta(x)$ to data. In large data limit ($N \to +\infty$), with flexible enough model, we expect $f_\theta(x)$ to be very close, if not identical, to the ground-truth function $f_0(x)$. In such case we can solve the optimisation problem by finding $x^* = \arg\max_{x \in \mathcal{X}} f_\theta(x)$ instead, which is tractable.

However, in practice we don't have such a big dataset to train the surrogate model. This is especially the case if the "black-box" corresponds to an expensive experiment (e.g., training a Transformer network where x represents the hyper-parameter settings). In such scenario $f_\theta(x)$ will be quite different from $f(x)$ in most of the unseen input locations. But at the end of the day, we are only interested in the maximum of $f_0(x)$ rather than the value of $f_0(x)$ at all possible input locations. This means training the surrogate model with very large datasets is unnecessary, and it is possible to use some smart algorithms that can find the optimum of f_0 without excessive queries to the "black-box".

The key idea of BO is to optimise f_0 using "helps" from the surrogate f_θ by taking the uncertainty of model fitting into account. It aims to find the optimum of f_0 with least amount of queries to the expensive "black-box". Specifically, there are 3 ingredients of an BO method:

1. Acquisition function: use the current estimated surrogate function $f_\theta(x)$ and its uncertainty estimate to compute an acquisition function $a(x)$;

2. Query the "black-box": find the next input x_* to query by maximising the acquisition function: $x_* = \arg\max_x a(x)$, and query the corresponding output value $y_* = f_0(x_*) + \epsilon_*$;

3. Surrogate model update: given new queried result (x_*, y_*) and historical data D, update $D \leftarrow D \cup \{(x_*, y_*)\}$, and use this new dataset D to update the surrogate model and its uncertainty estimate.

At the beginning since the model is uncertain, a good acquisition function will take uncertainty into account and encourage "exploration", i.e., querying inputs at different locations. As we collect more data, with proper Bayesian posterior updates and assuming the family of f_θ is flexible enough, the surrogate model f_θ will become closer and closer to the ground-truth function f_0 and the uncertainty will be reduced. So at some point, the model will become certain about its output, and a good acquisition function will also enable "exploitation" at this stage, to seek for the optimum of the surrogate function f_θ as to solve the original optimisation problem.

4.2 The Upper Confidence Bound (UCB) method

The Upper Confidence Bound (UCB) [Srinivas et al., 2010] is an acquisition function that uses both mean prediction and uncertainty. Specifically, assume the surrogate model provides both mean $m(x)$ and standard deviation $\sigma(x)$ for a given input x, then the UCB acquisition function is the following:

$$UCB(x) = m(x) + \beta\sigma(x), \tag{26}$$

and the query procedure will pick the next query input as

$$x = \arg\max UCB(x). \tag{27}$$

Initially, $\sigma(x)$ can be quite large for many regions, meaning that UCB will mainly explore. As we collect more data, $\sigma(x)$ will decrease around the regions that have been searched, and this allows the algorithm to

1. ignore some searched regions where the model confidently thinks their function value is small;
2. exploit some other searched regions where the model confidently believes the optimum might be there;
3. explore some other promising regions that has not been searched before.

Here β a hyper-parameter specified by the user at a time, to achieve a desired balance between exploration ($\sigma(x)$) and exploitation ($m(x)$). When $\beta = 0$, it means we trust the surrogate model and exploit on that. When β is large, we allow the query process to focus on regions that have large $\sigma(x)$ value (where the model is most uncertainty about). For optimal BO, this β coefficient will decrease during time; for simplicity, in the demo we only consider a fixed value of β.

5 Case study 2: Detecting adversarial examples

Neural networks has been shown to be vulnerable to adversarial attacks. In this case study, we will see whether BNNs can be more robust or not, as well as how to use uncertainty measures from BNNs to detect adversarial examples. The hypothesis that BNNs can be helpful in this setting is based on the following intuitions and conjectures:

- If BNN samples are diverse, there might exist members in the "Monte Carlo ensemble" that make correct predictions on adversarial examples;
- Adversarial examples are regarded as OOD data;
- BNNs become uncertain about their predictions on OOD data.

If the above intuitions and conjectures are correct, then it means we can use uncertainty measures computed using the BNN for adversarial example detection, even if the predictions are incorrect. Apparently in practice the validity of these conjectures rely on the quality of uncertainty estimates. Below We first discuss the uncertainty measures we will use in the case study. Then we discuss ensemble BNNs as a practical approach to further improve the uncertainty quality.

5.1 Uncertainty measures

We consider two types of uncertainty, using coin flipping as a running example:

- Epistemic uncertainty: also named model uncertainty, this is the uncertainty due to lack of knowledge, and thus can be reduced by collecting more data. For example, by flipping a coin multiple times, we become more and more certain about whether the coin is fair or bent;
- Aleatoric uncertainty: also named data uncertainty, this is the uncertainty regarding the stochasticity of individual experimental outcome, which is non-reducible. For example, even if we are 100% sure about that the coin is fair, we are still unsure about whether the next coin flip result will be head or tail.

These two type of uncertainty, when summed, returns the total uncertainty, i.e.,

$$\text{total uncertainty} \;=\; \text{epistemic uncertainty} \;+\; \text{aleatoric uncertainty}.$$

Quantitatively speaking, Shannon entropy is a commonly used measure to express uncertainty. For a categorical distribution $p = (p_1, ..., p_K)$ with $\sum_{k=1}^{K} p_k = 1$, the entropy is defined as

$$\mathbb{H}[p] = -\sum_{c=1}^{C} p_c \log p_c. \tag{28}$$

We can show that $\mathbb{H}[p]$ is maximised when $p_c = 1/C$, i.e., each category has equal probability, and in this case we are very uncertain about the sampling outcome. On the other hand, $\mathbb{H}[p] = 0$ when $p_c = 1$ for a particular $c \in \{1, ..., C\}$, meaning that the sampling outcome will be c for 100% of the time (thus certain). In other words, higher entropy means we are more uncertain, and vice versa.

For BNNs applied to classification tasks, we can also compute uncertainty based on entropy-related measures. Recall that the Bayesian predictive distribution is

$$p(y^*|x^*, D) = \int p(y^*|x^*, \theta) p(\theta|D) d\theta. \tag{29}$$

In classifcation case $p(y^*|x^*, D)$ is a categorical distribution, which means we can compute its entropy as to measure total uncertainty:

$$\mathbb{H}[y^*|x^*, D] = \mathbb{H}[p(y^*|x^*, D)] = -\sum_{c=1}^{C} p(y^* = c|x^*, D) \log p(y^* = c|x^*, D). \tag{30}$$

But also notice that for each sample $\theta \sim p(\theta|D)$, $p(y^*|x^*, \theta)$ is also a categorical distribution for which we can also compute its entropy. This enables us to compute the conditional entropy as to measure aleatoric uncertainty:

$$\mathbb{E}_{p(\theta|D)}[\mathbb{H}[y^*|x^*, \theta]] = \mathbb{E}_{p(\theta|D)}[\mathbb{H}[p(y^*|x^*, \theta)]] = \mathbb{E}_{p(\theta|D)}[-\sum_{c=1}^{C} p(y^* = c|x^*, \theta) \log p(y^* = c|x^*, \theta)]. \tag{31}$$

The intuition for conditional entropy being a measure for aleatoric uncertainty is as follows. When the posterior $p(\theta|D)$ becomes a delta mass, there is no epistemic uncertainty as we are now certain about the weight parameters. Even so, $p(y^*|x^*, \theta)$ for θ evaluated at the posterior mode can still have non-zero probability for multiple categories, which is especially the case when label noise exists (thus having non-zero aleatoric uncertainty). In this case the conditional entropy is greater than zero and can be used as a measure for aleatoric uncertainty.

As for epistemic uncertainty, from the relationship between total, epistemic and aleatoric uncertainty, we can compute it as "epistemic uncertainty = total uncertainty - aleatoric uncertainty". It terms out that this is related to the mutual information between y^* and θ:

$$\mathbb{I}[y^*; \theta|x^*, D] = \mathbb{H}[y^*|x^*, D] = \mathbb{H}[p(y^*|x^*, D)] - \mathbb{E}_{p(\theta|D)}[\mathbb{H}[y^*|x^*, \theta]]. \tag{32}$$

One can show that this mutual information can be re-written as

$$\mathbb{I}[y^*; \theta | x^*, D] = \mathbb{E}_{p(y^*|x^*,D)}[KL[p(\theta|y^*, x^*, D)||p(\theta|D)]]. \tag{33}$$

which tells us in (the model's) expectation, how much the posterior will be updated given a new observation (x^*, y^*). Notice that to reduce the epistemic uncertainty of a BNN, one can supply new datapoints at input locations that the network are uncertain about their outputs. As the reduction of epistemic uncertainty is conducted by making the posterior more concentrated, having higher mutual information typically means that the network expects to have higher epistemic uncertainty reduction if we supply (x^*, y^*) as a new observation, which also indicate that currently the network has high epistemic uncertainty about the output value at x^*.

The above discussed uncertainty measures have been used not only for OOD detection but also for sequential decision making, including acquisition function design in active learning and Bayesian optimisation. Interested readers are refer to e.g., [Houlsby et al., 2011; Hernández-Lobato et al., 2014; Gal et al., 2017] for further readings.

5.2 Ensemble BNNs

BNNs can be combined with ensemble methods [Lakshminarayanan et al., 2017] to get a better performance; indeed the winner solutions of the NeurIPS 2021 Approximate Inference in BDL competition[1] are based on ensembling BNNs. In short, one just train multiple BNNs (e.g., with MFVI) independently with different random initialisations, and then ensemble them together to produce the predictive distribution. In math, this means we fit a number of approximate posteriors $q_s(\theta), s = 1, ..., S$, and in prediction time, compute the predictive distribution by

$$p(y^*|x^*, D) \approx \frac{1}{SK} \sum_{s=1}^{S} \sum_{k=1}^{K} p(y^*|x^*, \theta_{sk}), \quad \theta_{sk} \sim q_s(\theta). \tag{34}$$

One can show that this approach also performs variational inference and the training objectives of these networks can also be combined to provide a valid lower-bound to the marginal log-likelihood $\log p(D)$. To see this, consider the following "augmented prior":

$$p(\theta, s) = p(\theta)p(s), \quad p(s) = \text{Uniform}(\{1, ..., S\}). \tag{35}$$

One can show that this "augmented model" $p(D|\theta)p(\theta)p(s)$ preserves the marginal likelihood:

$$\sum_{s=1}^{S} \int p(D|\theta)p(\theta)p(s)d\theta = \int p(D|\theta)p(\theta)d\theta = p(D). \tag{36}$$

This also allows us to define an ELBO based on this "augmented model" $p(D|\theta)p(\theta)p(s)$ and an approximate posterior $q(\theta, s)$. In particular, if we define

$$q(\theta, s) = p(s)q(\theta|s), \quad q(\theta|s) = q_s(\theta), \tag{37}$$

Then the ELBO can be written as

$$\begin{aligned} \log p(D) \geq L &= \mathbb{E}_{q(\theta,s)}[\log p(D|\theta)] - KL[q(\theta, s)||p(\theta)p(s)] \\ &= \frac{1}{S} \sum_{s=1}^{S} \left[\mathbb{E}_{q_s(\theta)}[\log p(D|\theta)] - KL[q_s(\theta)||p(\theta)]\right] = \frac{1}{S} \sum_{s=1}^{S} ELBO(q_s, D) \end{aligned} \tag{38}$$

Since the variational parameters (e.g., mean & log variance of the mean-field Gaussian) for different $q_s(\theta)$ are independent to each other, this means instead of training all S BNNs together with the above variational lower-bound L, one can simply train each of them independently using the ELBO objective $ELBO(q_s, D)$.

[1] https://izmailovpavel.github.io/neurips_bdl_competition/

References

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural network. In *International conference on machine learning*. PMLR.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*. PMLR.

Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep bayesian active learning with image data. In *International Conference on Machine Learning*, pages 1183–1192. PMLR.

Hernández-Lobato, J. M., Hoffman, M. W., and Ghahramani, Z. (2014). Predictive entropy search for efficient global optimization of black-box functions. *Advances in neural information processing systems*, 27.

Houlsby, N., Huszár, F., Ghahramani, Z., and Lengyel, M. (2011). Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the International Conference on Machine Learning, 2010*.