Uncertainty Quantification Methods - Part II

Variational Inference and Gaussian Processes

Dr. Matias Valdenegro

May 18th, 2022

Today's Agenda

1. Variational Inference for BNNs
2. Uncertainty Disentanglement
3. Comparison of UQ Methods

Outline

Concept

To learn a Bayesian NN, we need to learn $\mathbb{P}(w \mid D)$, the posterior probability of the weights. But for this we cannot use the standard Bayesian framework, as last week we covered its intractability.

But what if somehow we could learn *approximate distributions* for $\mathbb{P}(w \mid D)$?

Variational Inference

In VI, we wish to approximate $\mathbb{P}(w \mid D)$ with another distribution $q(w \mid D)$ that takes parameters θ (so it should be $q_\theta(w \mid D)$), such that some distance is minimized (to obtain the best approximation).

As a distance metric, the Kullback-Leibler Divergence is used, as it is a distance function for continuous probability distributions.

This process happens during training, so the KL Divergence is used as an addition to the standard training loss.

Kullback-Leibler Divergence

The KL Divergence is a distance metric for probability distributions, given by:

$$\mathrm{KL}(q, p) = \int_x q(x) \log \frac{q(x)}{p(x)} \tag{1}$$

Some important properties:

- $\mathrm{KL}(q, p) \geq q$ for all q and p.
- $\mathrm{KL}(q, p) = 0$ if and only if $q = p$.

Variational Inference

$$
\begin{aligned}
\theta^* &= \arg\min_{\theta} \mathrm{KL}(q(w \mid D), \mathbb{P}(w \mid D)) \\
&= \arg\min_{\theta} \int_w q(w \mid D) \log \frac{q(w \mid D)}{\mathbb{P}(w)\mathbb{P}(D \mid w)} \\
&= \arg\min_{\theta} \mathrm{KL}(q(w \mid D), \mathbb{P}(w)) - \mathbb{E}_{q(w \mid D)}[\log \mathbb{P}(D \mid w)]
\end{aligned}
$$

But what is the trick to allow for this to happen? We replaced $\mathbb{P}(w \mid D)$ with $\mathbb{P}(w)\mathbb{P}(D \mid w)$, this is an application of Bayes rule, but we missed the $\mathbb{P}(D)$ term, so this produces *unnormalized* probabilities.

Variational Inference - The Trick

Let's denote $\tilde{p}(x)$ our unnormalized probability, and $Z(\theta)$ the normalizing constant of bayes rule (the evidence). Then we can derive:

$$\begin{aligned} J(q) &= \int_x q(x) \log \frac{q(x)}{\tilde{p}(x)} \\ &= \int_x q(x) \log \frac{q(x)}{p(x)} - \log Z(\theta) \\ &= \mathrm{KL}(q, p) - \log Z(\theta) \end{aligned}$$

If we rearrange these terms to obtain the log-evidence, and noting that $\mathrm{KL}(q, p) \geq 0$:

$$\log Z(\theta) = \mathrm{KL}(q, p) - J(q) \geq -J(q) \tag{2}$$

This means that $-J(q)$ is a lower bound on the log-evidence $\log Z(\theta)$

Evidence Lower Bound

Another form to write the previous equation is:

$$\log Z(\theta) \geq \mathbb{E}_{q(x)}[\log \tilde{p}(x) - \tilde{q}(x)] \tag{3}$$

These forms are called Evidence Lower BOund (ELBO), particularly for $-J(q)$. Note that there are multiple versions of the ELBO that are all equivalent.

The form above shows that the difference between $\log Z(\theta)$ and $-J(q)$ is $\mathrm{KL}(q, p)$, so by maximizing the ELBO, the $\mathrm{KL}(q, p)$ is minimized by squeezing it between $-J(q)$ and $\log Z(\theta)$.

Variational Inference

After all these math details, going back to VI BNNs.

$$L(\theta) = \mathrm{KL}(q_\theta(w \mid D), \mathbb{P}(w)) - \mathbb{E}_{q_\theta(w \mid D)}[\log \mathbb{P}(D \mid w)] \quad (4)$$

The terms in this loss function can be interpreted as:

$\mathrm{KL}(q_\theta(w \mid D), \mathbb{P}(w))$ is the KL divergence between your approximate posterior of the weights $q_\theta(w \mid D)$ and the prior $\mathbb{P}(w)$.

$\mathbb{E}_{q_\theta(w \mid D)}[-\log \mathbb{P}(D \mid w)]$ is the loss of your model (negative log-likelihood) but taken as expectation over the approximate posterior of the weights.

Sampling for $\mathbb{E}_{q_\theta(w \mid D)}[-\log \mathbb{P}(D \mid w)]$

So in order to train a neural network with variational inference to approximate the posterior, we would need the following:

- Select a probability distribution for the weights.
- Use $L(\theta)$ as loss, where θ are the parameters of the approximate posterior. The loss is basically the KL between posterior and prior (for weights), plus the expectation of the prediction loss under the approximate posterior.
- To implement $\mathbb{E}_{q_\theta(w \mid D)}[]$, Monte Carlo sampling is usually used, by sampling $q_\theta(w \mid D)$, making multiple forward passes and computing the expectation as loss.
- Train using stochastic gradient descent.

Approximation Quality

One important detail when using the VI framework is that.

The posterior obtained through VI is an **approximation**.

But that is not the only issue:

The approximation quality is **unknown**, since the true posterior cannot be computed, and the ELBO only provides a lower bound which is loose.

Approximation Quality

There are many sources that contribute to the approximation quality:

- Selection of appropriate distribution for the weights.
- Depth/width of the network.
- Number of layers that are implemented using Bayesian principles (not all layers have to be Bayesian).
- Kind of layer (Recurrent, Convolutional, Fully Connected) and the selection of priors.
- Number of samples used during loss computation (directly proportional to number of forward passes).

Implementation Details

To implement the VI framework, it is required to implement custom layers with the following functionality:

1. Implement the weights/parameters of the layer as weight distributions.
2. Implement a stochastic forward pass by sampling the weight/bias distributions and make a forward pass using the standard equations.
3. Add terms to the loss to consider the KL divergence between weight distributions and the prior.

This implements new Convolutional, Dense, RNN/LSTM/GRU layers.

Bayes by Backprop [Blundell et al., 2015]

The paper reference is "Weight Uncertainty in Neural Networks" by Blundell et al. 2015.

It is an application of Variational Inference, where the weights are Gaussian distributions, plus some ways to compute gradients of the Gaussian parameters (mean and variance).

In modern implementations, we use Automatic Differentiation to compute these gradients with no issue.

Bayes by Backprop [Blundell et al., 2015]

- This method approximates the distribution of each weight with a Gaussian one, using a variational approximation.
- The weight distribution is learned automatically from the data, without additional supervision.

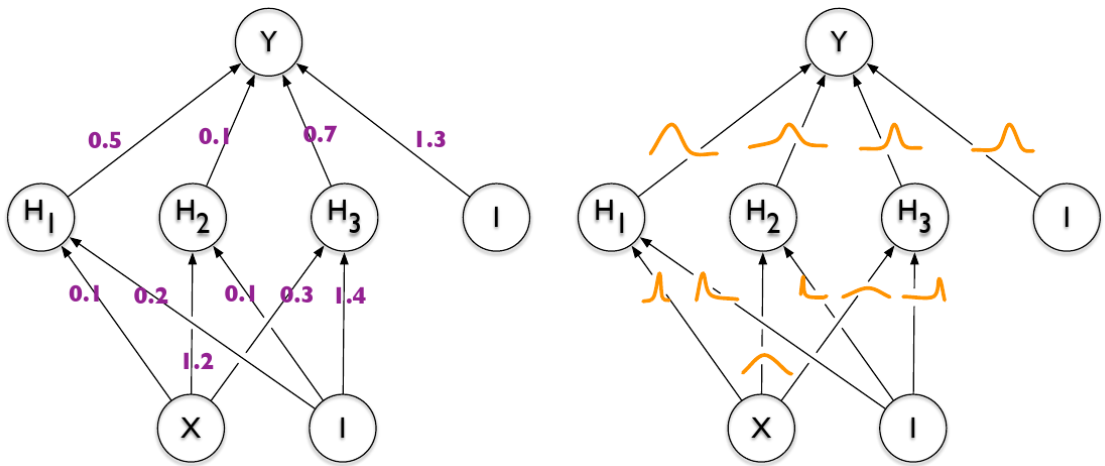


Figure 1. Left: each weight has a fixed value, as provided by classical backpropagation. Right: each weight is assigned a distribution, as provided by Bayes by Backprop.

Weight Uncertainty in Neural Networks [Blundell et al., 2015]

- Each weight in the model is no longer represented as a scalar (floating point number), but as a single Gaussian distribution with parameters $\mathcal{N}(\mu, \rho)$. This is called a variational approximation.
- The parameters of the variational distribution (μ, ρ) are updated using gradient descent.
- But since they weights are now distributions and not actual numbers, the gradient is approximated with a Monte Carlo gradient that is stochastic.
- Prediction outputs can be computed using the Bayesian predictive posterior distribution, by sampling different weight instances and making one forward pass per sample.

Weight Uncertainty in Neural Networks - Regression

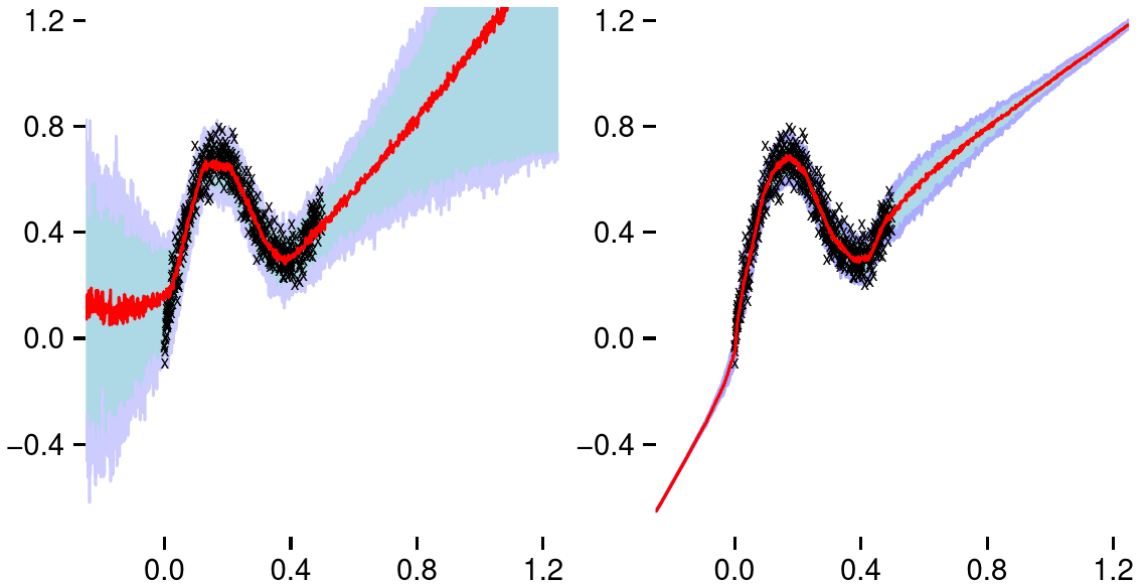
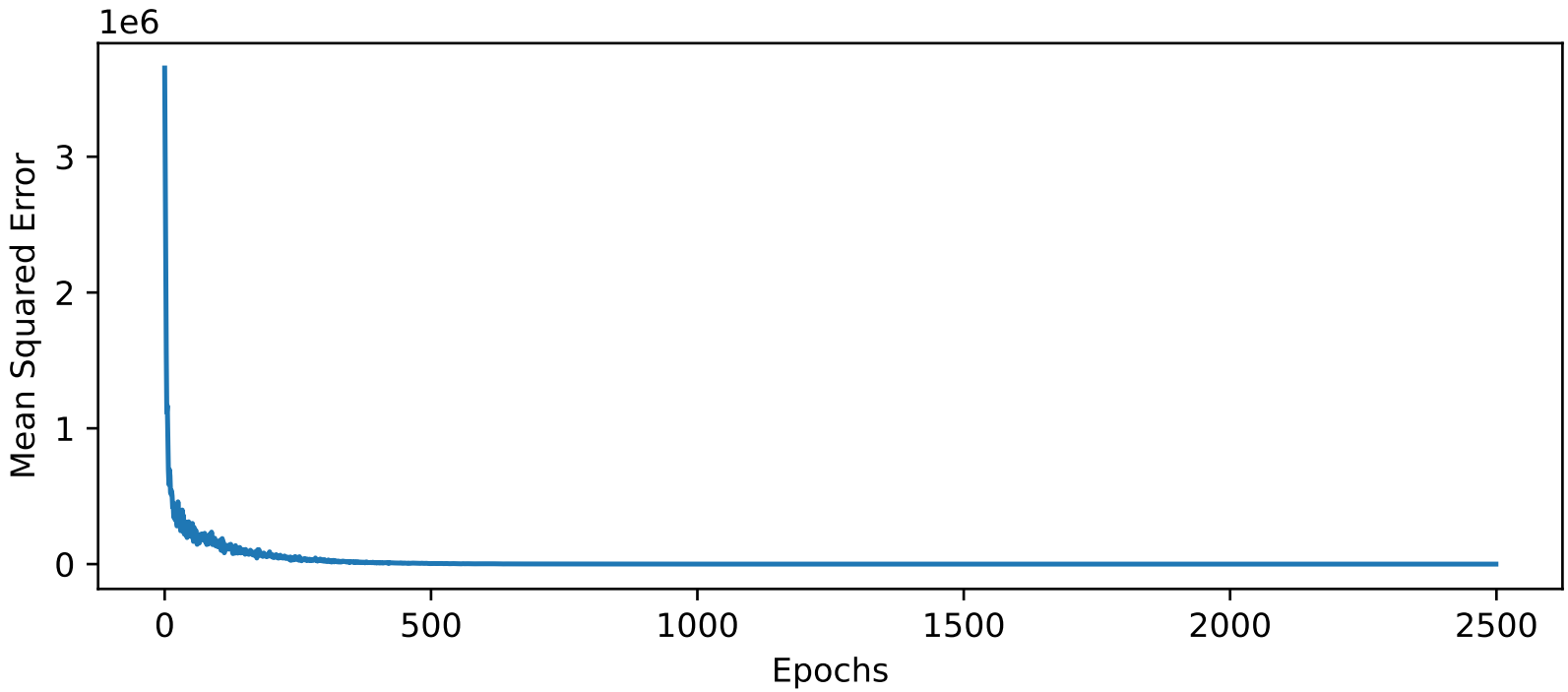


Figure 5. Regression of noisy data with interquatile ranges. Black crosses are training samples. Red lines are median predictions. Blue/purple region is interquartile range. Left: Bayes by Backprop neural network, Right: standard neural network.

Issues with Bayes by Backprop

- Training a model is unstable (loss is stochastic) due to the added noise while learning the Gaussian weight distributions.
- Training a model now requires several times more epochs, specially if a single sample is used to compute the loss. Using multiple samples increases computational requirements considerably.
- Overall BBB does not scale to large models, they simply do not converge and loss explodes (goes to infinity or NaN)

Issues with Bayes by Backprop - Regression



This is a plot of the training loss when using BBB, you can (barely) see it is noisy, but look at the range, it starts at 3 M mean squared error, this is huge. It takes more than 2000 epochs to converge.

Implementation Details

Usually the forward pass of a variational Bayesian NN is implemented like this:

$$y = a(Wx + b) \qquad W \sim \mathbb{P}(w_W \mid D), b \sim \mathbb{P}(w_b \mid D) \tag{5}$$

With slightly different equations for other layers like Conv/RNN/LSTM and a being an activation function.

In terms of batch processing, the weight matrix and bias is sampled once and applied to a whole batch of data. So each batch gets different samples of the weight/bias, but samples in the same batch get the same weight/bias sample.

This is not ideal as it reduces diversity during the training process and requires more epochs to converge.

Flipout for Variational BNNs [Wen et al., 2018]

It is problematic to train a Variational BNN due to the lack of diversity in the sampling of kernel and bias inside a batch.

It is theoretically possible to use one sample for each element in a batch, but this would make a forward pass more computationally inefficient, specially considering that all frameworks have optimized matrix multiplications with a batch of data, assuming that one of the matrices is constant and not variable.

Flipout for Variational BNNs [Wen et al., 2018]

Flipout aims to reduce the variance of the predicted distributions and training process with a mathematical trick to reduce this correlation. Flipout improves upon Bayes by Backprop. First we notice that since BBB uses a Gaussian distribution, the sampling process can be interpreted as a perturbation over the mean:

$$w = \mu + \sigma z \text{ with } z \sim \text{Norm}(0, 1), \rightarrow w \sim \text{Norm}(\mu, \sigma) \quad (6)$$

Here z has a Normal distribution, from where σz can be interpreted as an additive perturbation to the mean, which is the source of randomness.

Flipout for Variational BNNs [Wen et al., 2018]

Flipout aims to reduce the variance of the predicted distributions and training process with a mathematical trick to reduce this correlation, with the $\Delta\hat{W} = \sigma z$ now being a per-sample perturbation.:

$$\Delta W_n = \Delta\hat{W} r_n s_n^T \tag{7}$$

Where r_n and s_n are independent samples from a Rademacher distribution (binary -1 and 1 values), and ΔW_n is the perturbation added to the kernel/bias (basically the standard deviation). This ensures each sample in a batch receives a different kernel/bias sample.

Flipout for Variational BNNs [Wen et al., 2018]

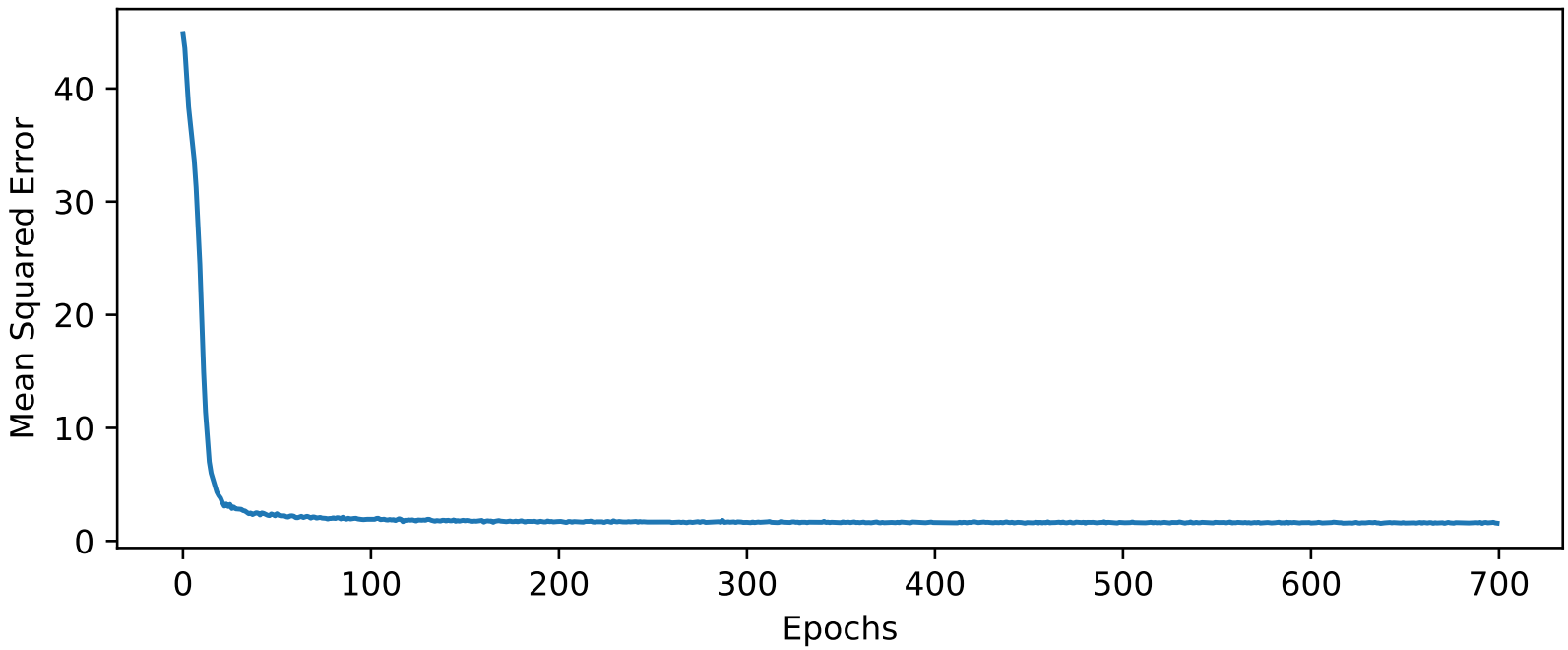
Why do this transformation of the perturbation?

First reason is to decorrelate the kernel/bias samples inside a batch.

The second is that the Flipout authors show that the distribution of the kernel/bias is the same as the original weight distribution. This is why this transformation actually works.

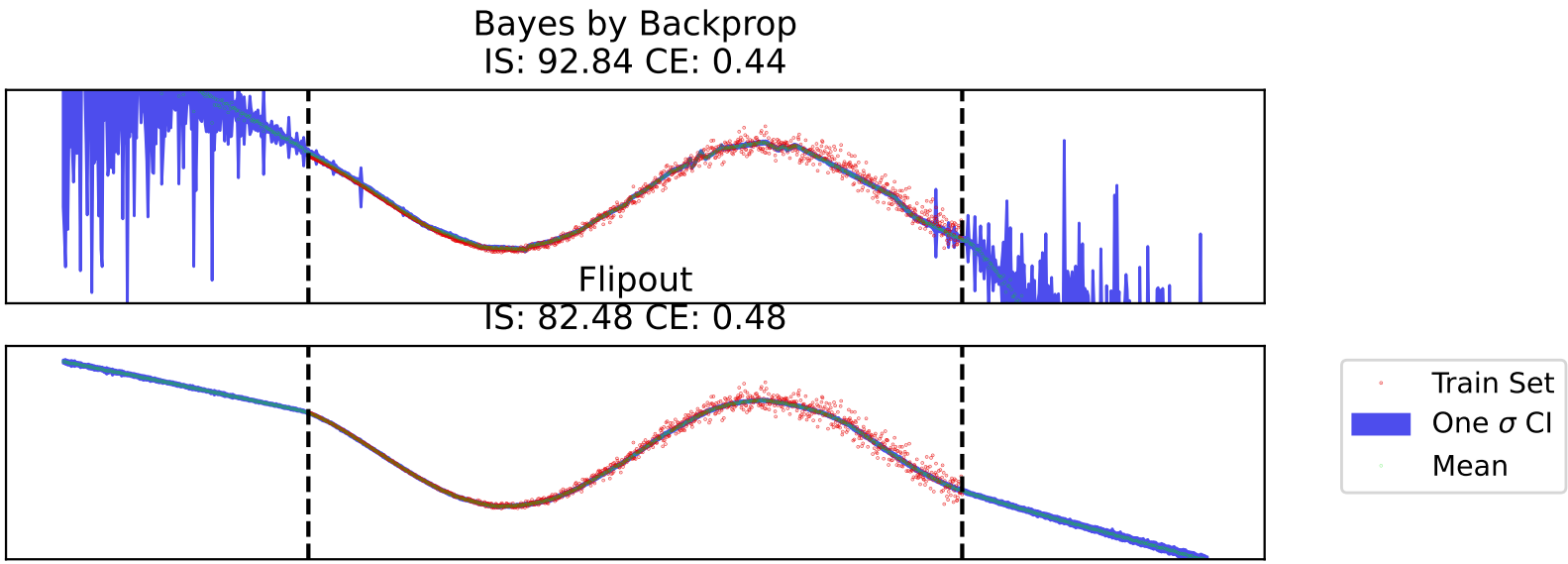
But now there is an advantage of decorrelating samples inside a batch.

Flipout for Variational BNNs - Regression



This is a plot of the training loss when using Flipout, you can see that it converges faster and it is overall less noisy. The loss also starts at a much lower value of around 50.

Flipout vs BBB - Regression



This is comparison in regression between BBB and Flipout, the difference is pretty obvious, BBB is much more noisy, while Flipout has reduced variance.

Architectural Considerations

Bayesian NNs are not required to use all layers with weight distributions (let's call this Bayesian Layers). Note that the concept of Bayesian layers also applies to other methods like MC-Dropout/DropConnect, Ensembling, etc.

Selecting more or less Bayesian layers in a model will change its behavior, and apply a trade-off between amount of computation and quality of uncertainty. Less layers will result in a faster model, but not good uncertainty, while using many Bayesian layers, model will be slower, but uncertainty will be high quality.

Question. What kind of uncertainty (aleatoric or epistemic) will change when using more or less Bayesian layers?

Making Predictions

To make predictions with a VI-approximate Bayesian neural network, we use the Monte Carlo approximation to the Predictive Posterior Distribution with M samples:

$$\mathbb{P}(\mathbf{y} \mid \mathbf{x}) \sim M^{-1} \sum_{i}^{M} \mathbb{P}(\mathbf{y} \mid \theta_i, \mathbf{x}) \mathbb{P}(\theta_i \mid \mathbf{x}) \quad \theta_i \sim \mathbb{P}(\mathbf{w} \mid \mathbf{x}) \tag{8}$$

Here the Posterior distribution of the weights $\mathbb{P}(\mathbf{w} \mid \mathbf{x})$ is sampled to produce weights, which are used to make a forward pass (the $\mathbb{P}(\mathbf{y} \mid \theta_i, \mathbf{x})$) and then take a weighted average. More commonly we use an approximation:

$$\mathbb{P}(\mathbf{y} \mid \mathbf{x}) \sim M^{-1} \sum_{i}^{M} \mathbb{P}(\mathbf{y} \mid \theta_i, \mathbf{x}) \quad \theta_i \sim \mathbb{P}(\mathbf{w} \mid \mathbf{x}) \tag{9}$$

This one does not weight using the weight posterior, for cases where the weight probabilities $\mathbb{P}(\theta_i \mid \mathbf{x})$ cannot be directly computed.

Outline

Concept

When we make a prediction with uncertainty, using any of the methods that we have covered, we usually obtain a combination of aleatoric and epistemic uncertainty.

$$\text{Predictive Uncertainty} = \text{Epistemic} + \text{Aleatoric} \tag{10}$$

But can these two uncertainties be obtained separately? And more importantly, how can you train a model to obtain both kinds of uncertainties?

Applications Benefiting from Disentanglement

- Out of Distribution Detection requires only Epistemic Uncertainty.
- If you train a model to produce measurements (from some noisy ground truth), you want the model to tell you separately about the measurement noise (aleatoric uncertainty) and the model uncertainty (epistemic).
- Active Learning requires very good Epistemic uncertainty estimates, while ignoring Aleatoric uncertainty. This is used to select which samples to label, and the model indicates which sample it should learn next.

Uncertainty Disentanglement - Regression

Aleatoric Uncertainty

For a regression setting, we can estimate Aleatoric Uncertainty using the Gaussian Negative Log-Likelihood and a Two-Head Model.

Aleatoric Uncertainty

For a regression setting, Epistemic Uncertainty is produced by using a UQ method like MC-Dropout/DropConnect, Ensembles, Bayes by Backprop, or Flipout.

Uncertainty Disentanglement - Regression

Let's start on how to combine both sources of uncertainty, since we have multiple forward passes or ensemble members, we use a Gaussian mixture model (same as an Ensemble):

$$
\begin{aligned}
p_e(y \mid \mathbf{x}) &\sim \mathcal{N}(\mu_*(\mathbf{x}), \sigma_*^2(\mathbf{x})) \\
\mu_*(\mathbf{x}) &= M^{-1} \sum_i \mu_i(\mathbf{x}) \\
\sigma_*^2(\mathbf{x}) &= M^{-1} \sum_i (\sigma_i^2(\mathbf{x}) + \mu_i^2(\mathbf{x})) - \mu_*^2(\mathbf{x})
\end{aligned}
$$

In this formulation, the index i runs along the samples/ensembles dimension, and M is the number of samples or number of ensemble members.

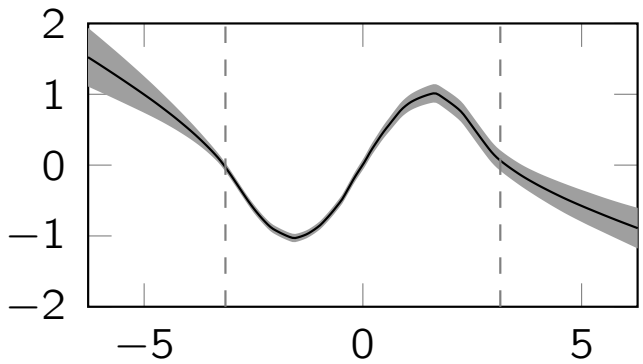
Uncertainty Disentanglement - Regression

Now we take a deeper look at the variance σ_*^2, which can be decomposed into two important terms.

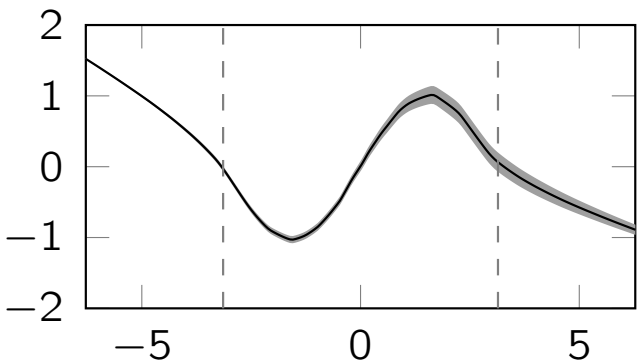
$$\begin{aligned}
\sigma_*^2(\mathbf{x}) &= M^{-1}\sum_i \sigma_i^2(\mathbf{x}) &&+ M^{-1}\sum_i \mu_i^2(\mathbf{x}) - \mu_*^2(\mathbf{x}) \\
&= \mathbb{E}_i[\sigma_i^2(\mathbf{x})] &&+ \mathbb{E}_i[\mu_i^2(\mathbf{x})] - \mathbb{E}_i[\mu_i(\mathbf{x})]^2 \\
&= \underbrace{\mathbb{E}_i[\sigma_i^2(\mathbf{x})]}_{\text{Aleatoric Uncertainty}} &&+ \underbrace{\mathrm{Var}_i[\mu_i(\mathbf{x})]}_{\text{Epistemic Uncertainty}}
\end{aligned}$$

This is easy to interpret, Aleatoric uncertainty is the mean of the per-sample/ensemble variances, and Epistemic uncertainty is the variance of the per-sample/ensemble means.

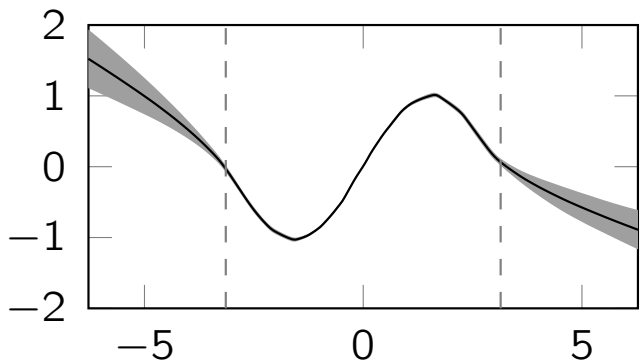
Uncertainty Disentanglement - Regression



(a) Predictive Uncertainty



(b) Aleatoric Uncertainty



(c) Epistemic Uncertainty

Figure: Sample of uncertainty disentanglement in a toy regression example, produced using an ensemble of 15 neural networks.

Uncertainty Disentanglement - Classification

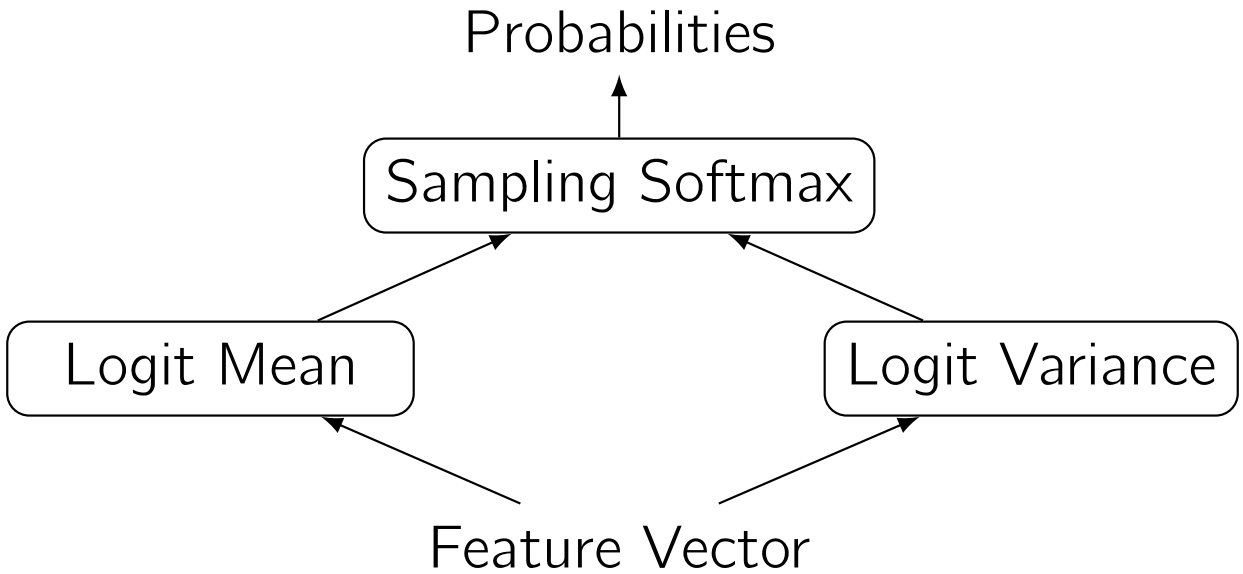
Aleatoric Uncertainty

For a classification setting, we can estimate Aleatoric Uncertainty using the standard cross-entropy but the output logits of the model should have uncertainty, and passed through the Sampling Softmax function, which takes Gaussian distributed logits and passed this distribution through the softmax function.

Aleatoric Uncertainty

For a classification setting, Epistemic Uncertainty is produced by using a UQ method like MC-Dropout/DropConnect, Ensembles, Bayes by Backprop, or Flipout.

Logits with Uncertainty for Classification



Computational graph for the sampling softmax function, showing the two fully connected layers that produce logit mean and variance, and how they relate to the final probabilities through the sampling softmax function.

Sampling Softmax Function

The big question is how to pass your Gaussian distributed logits with mean $\mu(\mathbf{x})$ and variance $\sigma^2(\mathbf{x})$, through the softmax function. The only known way is to (again) use sampling.

$$\hat{\mathbf{z}}_j \sim \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x})) \tag{11}$$

$$\mathbb{P}(\mathbf{y} \mid \mathbf{x}) = N^{-1} \sum_j \text{softmax}(\hat{\mathbf{z}}_j) \qquad j \in [1, N] \tag{12}$$

Here we generate N samples of the logit Gaussian distribution $\mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x}))$, which is denoted by $\hat{\mathbf{z}}_j$, apply the standard softmax activation to each sample, and then take the sample mean to obtain the output $\mathbb{P}(\mathbf{y} \mid \mathbf{x})$.

N is the number of samples to use, and defines the quality of the approximation to the true softmax function (which we do not know).

Uncertainty Disentanglement - Classification

Then at inference time, we again assume that the uncertainty method uses sampling through forwarding passes or ensembling a model on the i axis (with $i \in [1, M]$). Then, aleatoric σ^2_{Ale} and epistemic σ^2_{Epi} uncertainty logits can be computed

$$\sigma^2_{\text{Ale}}(\mathbf{x}) = \mathbb{E}_i[\sigma_i^2(\mathbf{x})] \qquad \sigma^2_{\text{Epi}}(\mathbf{x}) \qquad = \text{Var}_i[\mu_i(\mathbf{x})]. \qquad (13)$$

Note that these are logits, not probabilities. These steps are equivalent to the disentangling for regression, where the input to the Sampling softmax function is a kind of regression (with uncertainty) of the logits.

In this slide, $\mu(\mathbf{x})$ is the mean, and $\sigma^2(\mathbf{x})$ the variance of the logit Gaussian distribution.

Uncertainty Disentanglement - Classification

Passing each corresponding logit through a softmax function can produce probabilities, from where entropy is a possible metric to obtain a scalar uncertainty measure:

$$
\begin{aligned}
p_{\text{Ale}}(y|\mathbf{x}) &= \text{sampling\_softmax}(\mu(\mathbf{x}), \sigma^2_{\text{Ale}}(\mathbf{x})) \\
H_{\text{Ale}}(y|\mathbf{x}) &= \text{entropy}(p_{Ale}(y|\mathbf{x})) \\
p_{\text{Epi}}(y|\mathbf{x}) &= \text{sampling\_softmax}(\mu(\mathbf{x}), \sigma^2_{\text{Epi}}(\mathbf{x})) \\
H_{\text{Epi}}(y|\mathbf{x}) &= \text{entropy}(p_{Epi}(y|\mathbf{x}))
\end{aligned}
$$

With $\mu(x) = M^{-1} \sum_i \mu_i(x)$ is the predictive mean and entropy is the standard Shannon entropy:

$$
\text{entropy}(p) = -\sum_i p_i \log p_i \tag{14}
$$

Uncertainty Disentanglement - Classification

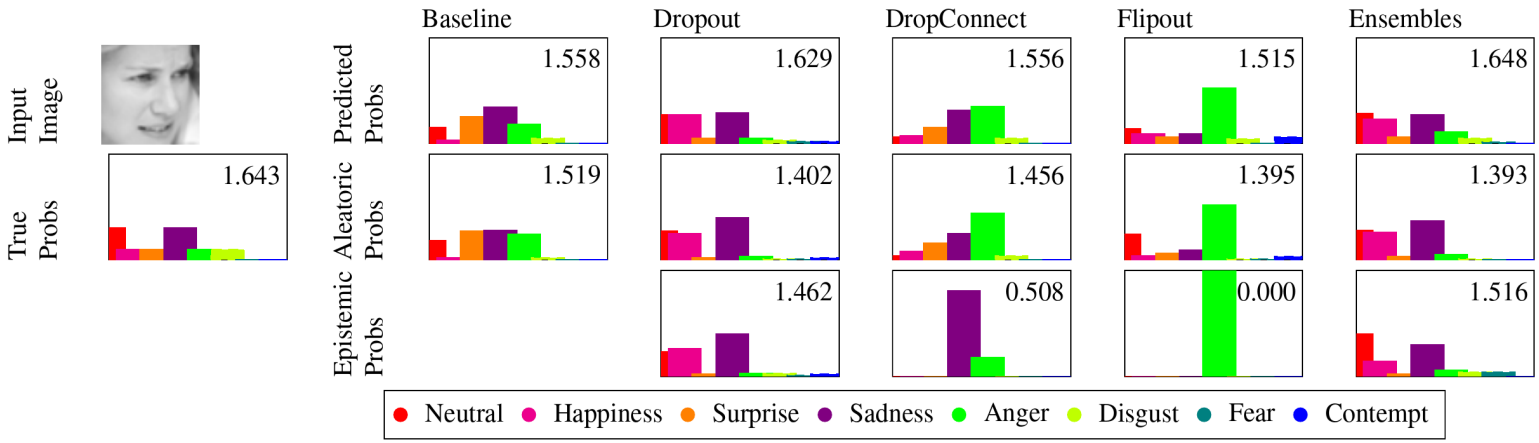


Figure 12. Facial image 813 from the FER+ test set. Here we show the aleatoric and epistemic probabilities produced by different uncertainty quantification methods. The entropy of each distribution is displayed in the top right corner. Ensembles has the highest epistemic uncertainty, while Flipout has zero and predicts an incorrect class.

These results are from my paper [Valdenegro-Toro and Saromo, 2022] to appear in this year CVPR Workshops. Here we compare different uncertainty methods according to their uncertainty disentanglement, in this case classification of facial emotions.

Uncertainty Disentanglement - Classification

Note that while in regression aleatoric and epistemic variances sum to produce predictive variance.

$$\text{Predictive Var} = \text{Epistemic Var} + \text{Aleatoric Var} \tag{15}$$

This does not apply in classification, aleatoric and epistemic probabilities do not sum to predictive probabilities. Only the (epistemic and aleatoric) logits can be summed to obtain predictive logits.

$$\text{Predictive Logits} = \text{Epistemic Logits} + \text{Aleatoric Logits} \tag{16}$$

Question. If they do not sum, what is their relationship?

Overall Concept + Architectures

From the previous content, one important detail is that by using UQ methods and specific losses like the Gaussian NLL, the neural network designer basically "adds" proper uncertainty quantification to an already made architecture.

By choosing different UQ methods, you add epistemic uncertainty.
By using a two-head with Gaussian NLL (or other NLL loss) or Sampling softmax, you add aleatoric uncertainty estimation to your model.

With both of them, you get predictive uncertainty (aleatoric + epistemic) that can be disentangled with proper formulations.

Outline

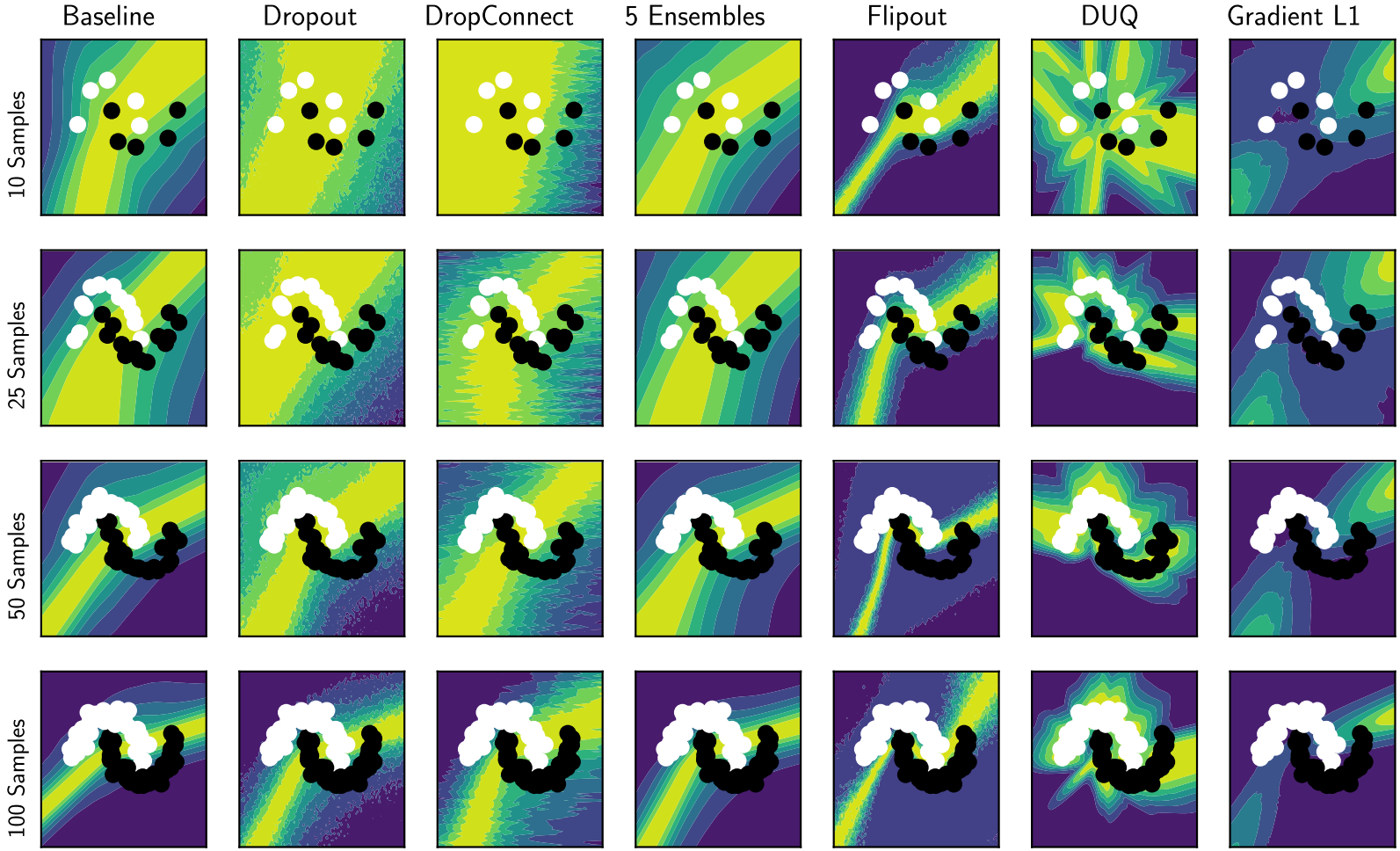
Comparison Across Train Set Size[Valdenegro-Toro, 2021]

The coming results are part of a paper I published last year at NeurIPS 2021 Workshops, titled "Exploring the Limits of Epistemic Uncertainty Quantification in Low-Shot Settings".

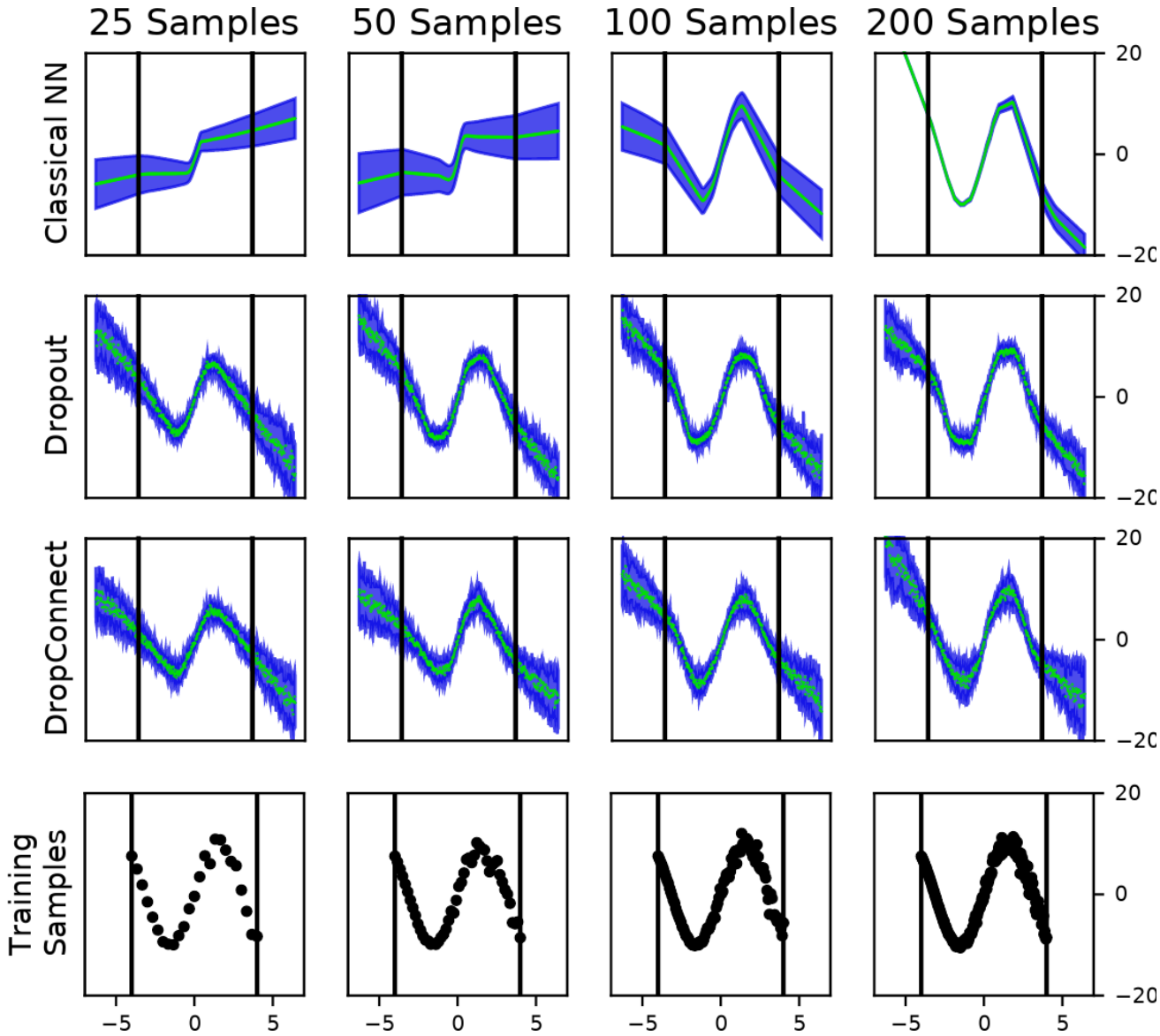
Here we compare multiple uncertainty quantification methods across different sizes of the training set, and obtain some interesting conclusions.

Question. If we change the vary the training set size, what uncertainty should change? (Aleatoric or epistemic).

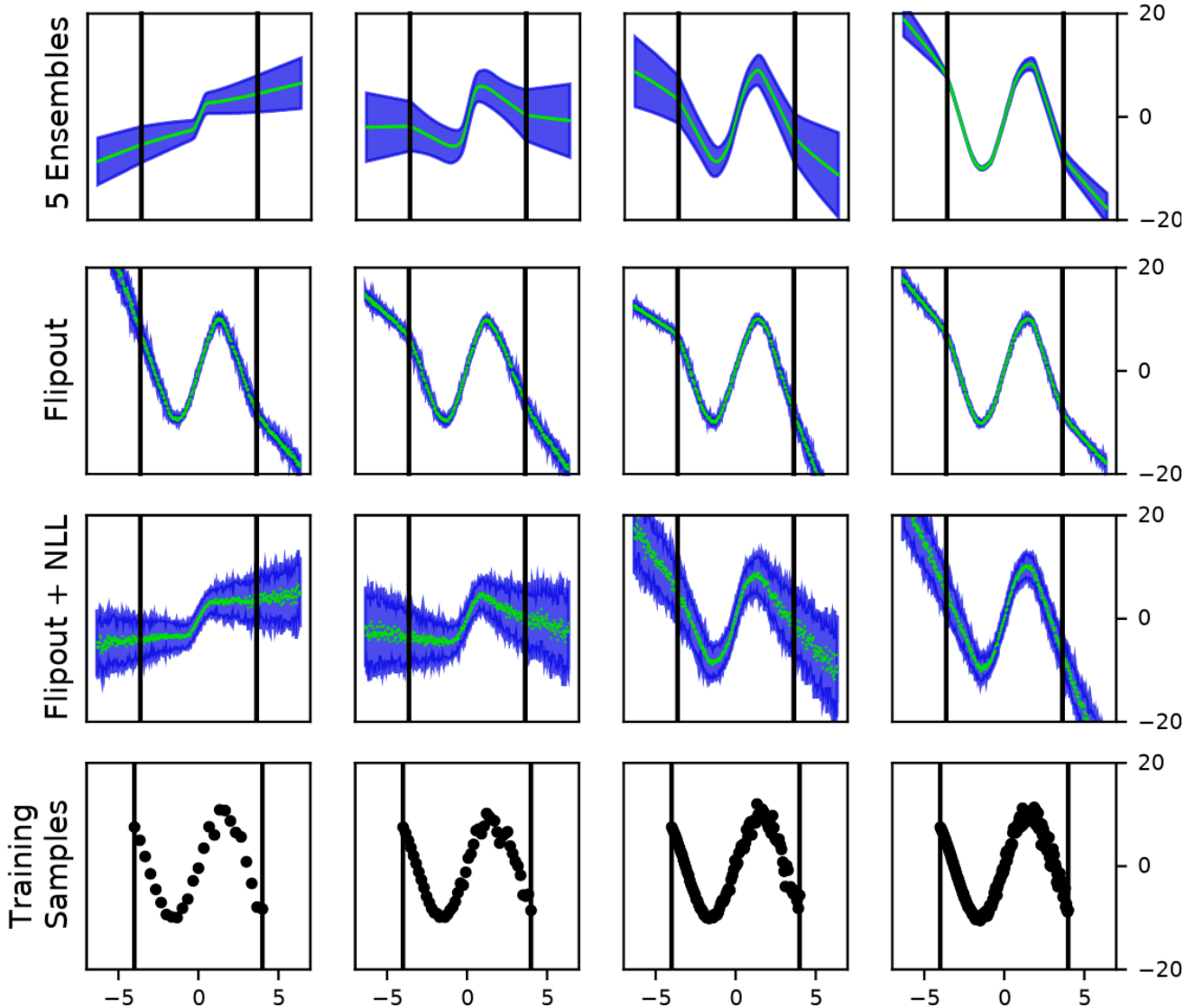
Two Moons Classification



Noisy Sinusoid Regression 1/2



Noisy Sinusoid Regression 2/2



Questions?

Questions to Think About

- What is Variational Inference in Neural Networks (in concept)?
- What are some disadvantages of Variational Inference for BNNs?
- What is the overall concept to disentangle aleatoric/epistemic uncertainty?
- How is disentangling uncertainty in classification and regression different?

Bibliography I

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR, 2015.

Matias Valdenegro-Toro. Exploring the limits of epistemic uncertainty quantification in low-shot settings. *arXiv preprint arXiv:2111.09808*, 2021.

Matias Valdenegro-Toro and Daniel Saromo. A deeper look into aleatoric and epistemic uncertainty disentanglement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2022.

Bibliography II

Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *arXiv preprint arXiv:1803.04386*, 2018.