university of groningen

faculty of science and engineering

Uncertainty Quantification Methods

Dr. Matias Valdenegro
Department of Artificial Intelligence
University of Groningen

January 25, 2024

Today's Agenda

1. Direct Uncertainty Estimation
2. Bayesian Neural Networks
3. Sampling-Based Uncertainty Estimation
4. Variational Inference for BNNs
5. Uncertainty Disentanglement
6. Comparison of UQ Methods

Outline

Maximum Likelihood Estimation

Given data, how can we fit a probability distribution to it? Maximum Likelihood is a statistical principle/method that can be used to find parameters for a distribution. It starts with the following formula:

$$L(\theta; y) = \prod_i f_X(y_i; \theta) \tag{1}$$

Where f_X is the PDF of a selected distribution (assumption) with parameters θ. Eq 1 is called a likelihood function. This function gives you the joint probability of all data points y under the assumed distribution. We can find the best parameters θ by maximizing the likelihood function under θ:

$$\theta^* = \arg\max_\theta L(\theta; y) \tag{2}$$

Maximum Likelihood Estimation

Now the form of the likelihood function is not so easy to optimize in a closed form, an alternative

$$\log L(\theta; y) = \log \prod_i f_X(y_i; \theta) = \sum_i \log f_X(y_i; \theta) \tag{3}$$

The (natural) logarithm is a increasing function, so it does not change the position (values) of the optimal parameters, but now this form is easier to optimize. Eq 3 is called the log-likelihood.

The optimal parameters are usually found with gradient-based optimization:

$$\frac{\partial L}{\partial \theta} = 0 \leftrightarrow \frac{\partial \log L}{\partial \theta} = -\infty \text{ (Approx.)} \tag{4}$$

Two-Headed Models in Regression

A very simple idea is, if we need a model to output some uncertainty measure, why not just add a second output head that outputs this confidence?

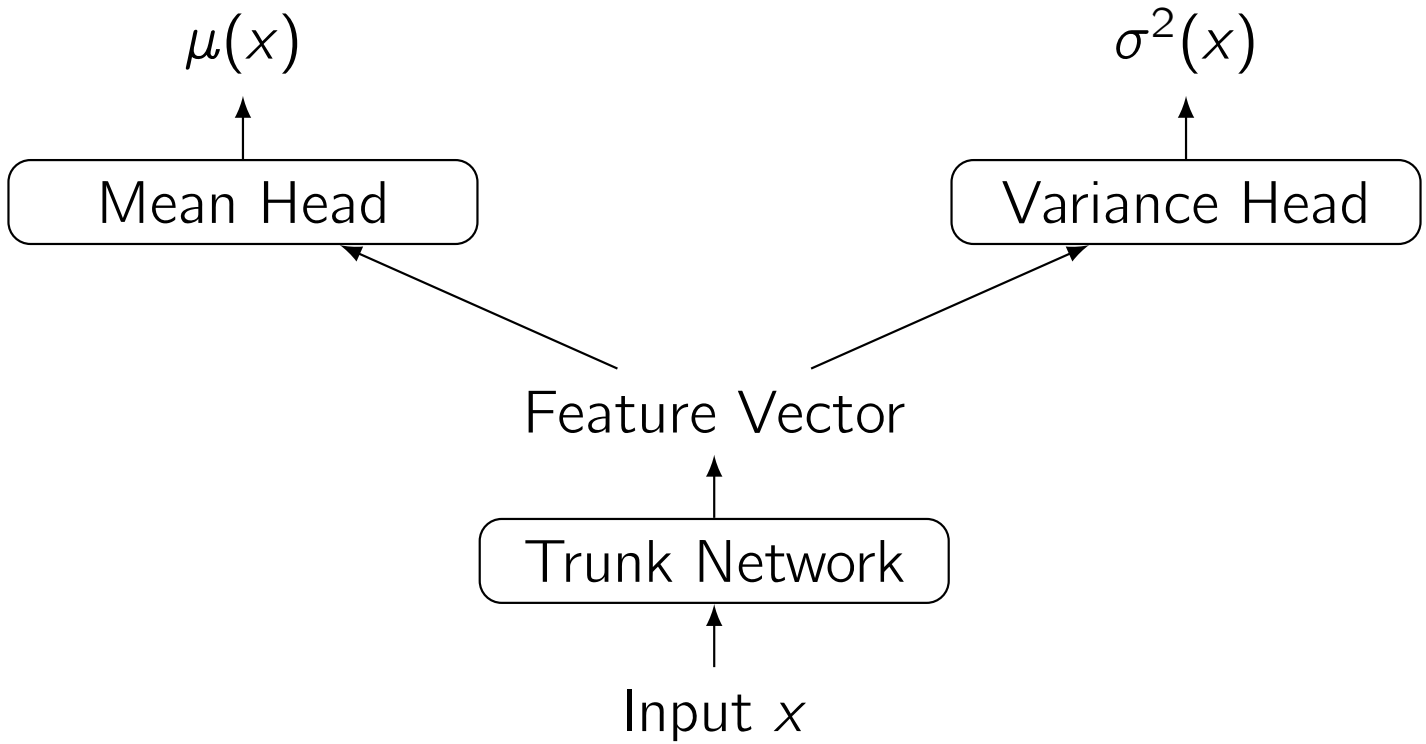


Figure: Basic idea of a network with two output heads for regression.

Two-Headed Models in Regression

The question is then, how to train such a network? We can make some assumptions, like that we want to output a Gaussian distribution, we can do this by designing a loss to maximize likelihood:

$$L(y, \mu(x), \sigma^2(x)) = \prod_i (2\pi\sigma_i^2)^{-\frac{1}{2}} e^{-\frac{1}{2}(\frac{y_i-\mu_i}{\sigma_i})^2}$$

$$\log L(y, \mu(x), \sigma^2(x)) = \sum_i \log\left((2\pi\sigma_i^2)^{-\frac{1}{2}} e^{-\frac{1}{2}(\frac{y_i-\mu_i}{\sigma_i})^2}\right)$$

$$= \sum_i -\frac{1}{2}(\log 2\pi + \log\sigma^2) - \frac{1}{2}\left(\frac{y_i-\mu_i}{\sigma}\right)^2$$

Now this is a log-likelihood that we want to maximize, but in general we want a loss function to minimize. Maximizing L is equivalent to minimizing $-L$. y here is the target label.

Two-Headed Models - Gaussian Negative Log-Likelihood

$$-\log L(y, \mu(x), \sigma^2(x)) = \sum_i 0.5(\log 2\pi + \log \sigma_i^2) + 0.5 \left(\frac{y_i - \mu_i}{\sigma_i} \right)^2$$

Now if we remove all constant terms (they do not contribute to gradients and do not change the optimal solution), we arrive at:

$$-\log L(y, \mu(x), \sigma^2(x)) = \sum_i \log \sigma_i^2 + \frac{(y_i - \mu_i)^2}{\sigma_i^2} \tag{5}$$

This loss is called the Gaussian Negative Log-Likelihood.

Gaussian Negative Log-Likelihood - Interpretation

$$-\log L(y, \mu(x), \sigma^2(x)) = \sum_i \log \sigma_i^2 + \frac{(y_i - \mu_i)^2}{\sigma_i^2}$$

Now that we derived the Gaussian NLL, what does it actually do? We first note that the mean μ_i is supervised by the label y_i, while the variance σ_i^2 does not have direct supervision, but the loss influences it.

- $\sigma_i^2 \to \infty$ The logarithm term goes to ∞ and the squared error term tends to 0.
- $\sigma_i^2 \to 0$ The logarithm term goes to $-\infty$ and the squared error term tends to ∞.

This loss is also called variance attenuation, can we see why?

Gaussian Negative Log-Likelihood - Interpretation

$$-\log L(y, \mu(x), \sigma^2(x)) = \sum_i \log \sigma_i^2 + \frac{(y_i - \mu_i)^2}{\sigma_i^2}$$

$(y_i - \mu_i)^2$ Small Variance is usually small.

$(y_i - \mu_i)^2$ Big Variance is usually large.

When the model predicts correctly, the variance is also small, but when the model predicts incorrectly (large squared error), then the only way to minimize the loss is to increase the variance.

The two loss terms involve the variance and they counter-act each other.

Gaussian NLL - Learning

Important question: What kind of uncertainty does the Gaussian NLL learn?

Gaussian NLL - Learning

Important question: What kind of uncertainty does the Gaussian NLL learn?

Answer: Aleatoric Uncertainty.... But why?

Gaussian NLL - Learning

Important question: What kind of uncertainty does the Gaussian NLL learn?

Answer: Aleatoric Uncertainty.... But why?

Because the mean head is directly supervised by the data, if there is noisy data, it will be hard to predict correctly, but the variance head will cover all noisy data points, estimating its aleatoric uncertainty (noise). For clean data, the variance will be low.

Ensembles [Lakshminarayanan et al., 2017]

- Ensembles have also powerful uncertainty estimation properties.
- Ensembling consists of training M instances of the same model, but with different randomly drawn initial weights, and then combining their predictions.
- For regression, each ensemble member has two output heads, one for the mean $\mu_i(x)$ and one for the variance $\sigma_i^2(x)$, and a special loss is used for training:

$$-\log p(y_n|\mathbf{x}_n) = \frac{\log \sigma_i^2(\mathbf{x}_n)}{2} + \frac{(\mu_i(\mathbf{x}_n) - y_n)^2}{2\sigma_i^2(\mathbf{x}_n)} + C$$

- This loss is a negative log-likelihood with heteroscedastic variance, the model predicts a variance for each data point.

Ensembles - Combination [Lakshminarayanan et al., 2017]

Where p_i is the output of the i-th member in the ensemble:

Classification

Ensemble output is average of the probabilities:

$$p_e(y \mid \mathbf{x}) = M^{-1} \sum_i p_i(y \mid \mathbf{x})$$

Regression

Ensemble output is a Gaussian mixture model:

$$p_e(y \mid \mathbf{x}) \sim \mathcal{N}(\mu_*(\mathbf{x}), \sigma_*^2(\mathbf{x}))$$

$$\mu_*(\mathbf{x}) = M^{-1} \sum_i \mu_i(\mathbf{x})$$

$$\sigma_*^2(\mathbf{x}) = M^{-1} \sum_i (\sigma_i^2(\mathbf{x}) + \mu_i^2(\mathbf{x})) - \mu_*^2(\mathbf{x})$$

Ensembles - Toy Regression [Lakshminarayanan et al., 2017]

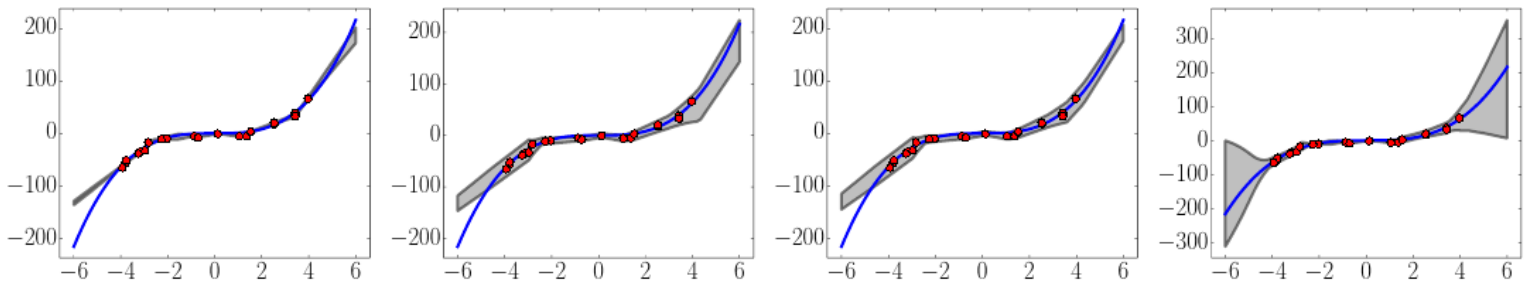
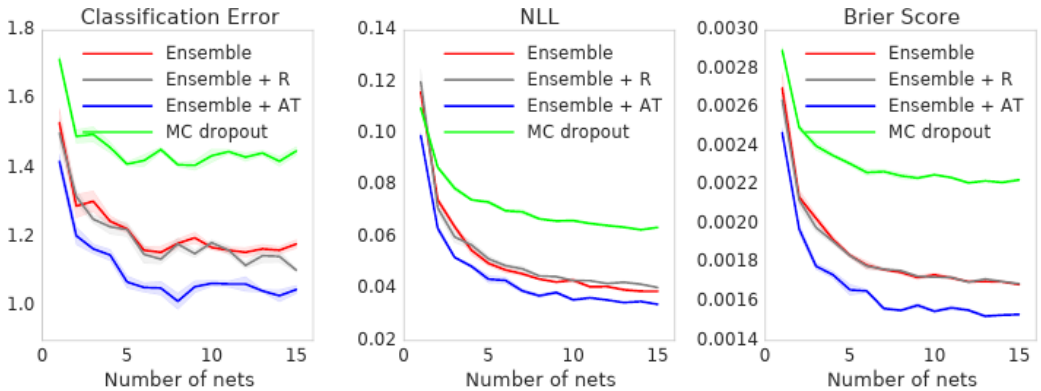
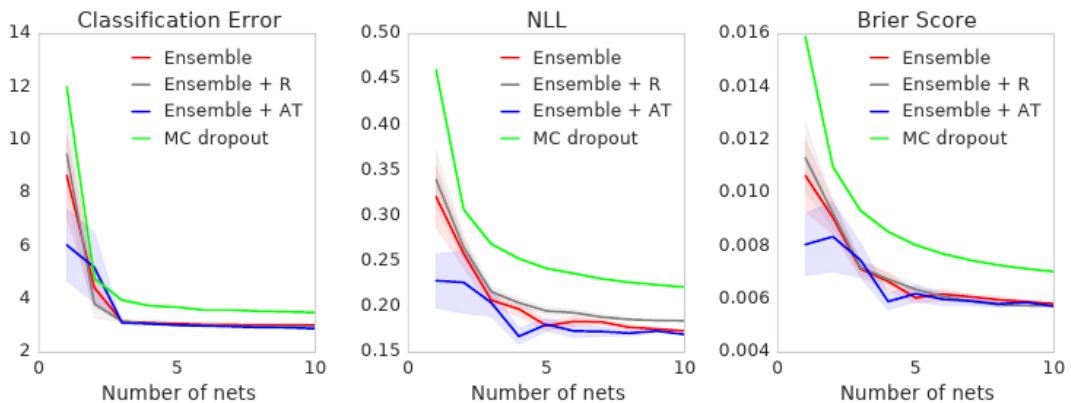


Figure 1: Results on a toy regression task: x-axis denotes x. On the y-axis, the blue line is the *ground truth* curve, the red dots are observed noisy training data points and the gray lines correspond to the predicted mean along with three standard deviations. Left most plot corresponds to empirical variance of 5 networks trained using MSE, second plot shows the effect of training using NLL using a single net, third plot shows the additional effect of adversarial training, and final plot shows the effect of using an ensemble of 5 networks respectively.

Ensembles - Classification [Lakshminarayanan et al., 2017]

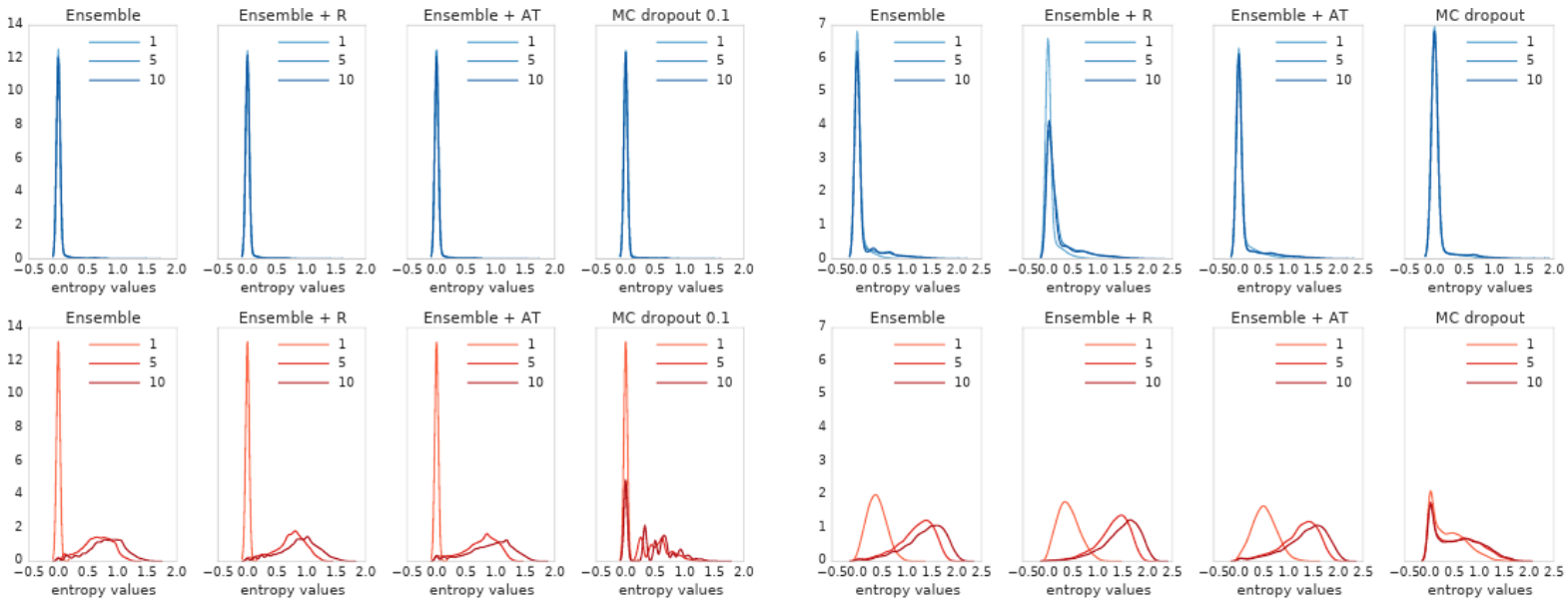


(a) MNIST dataset using 3-layer MLP



(b) SVHN using VGG-style convnet

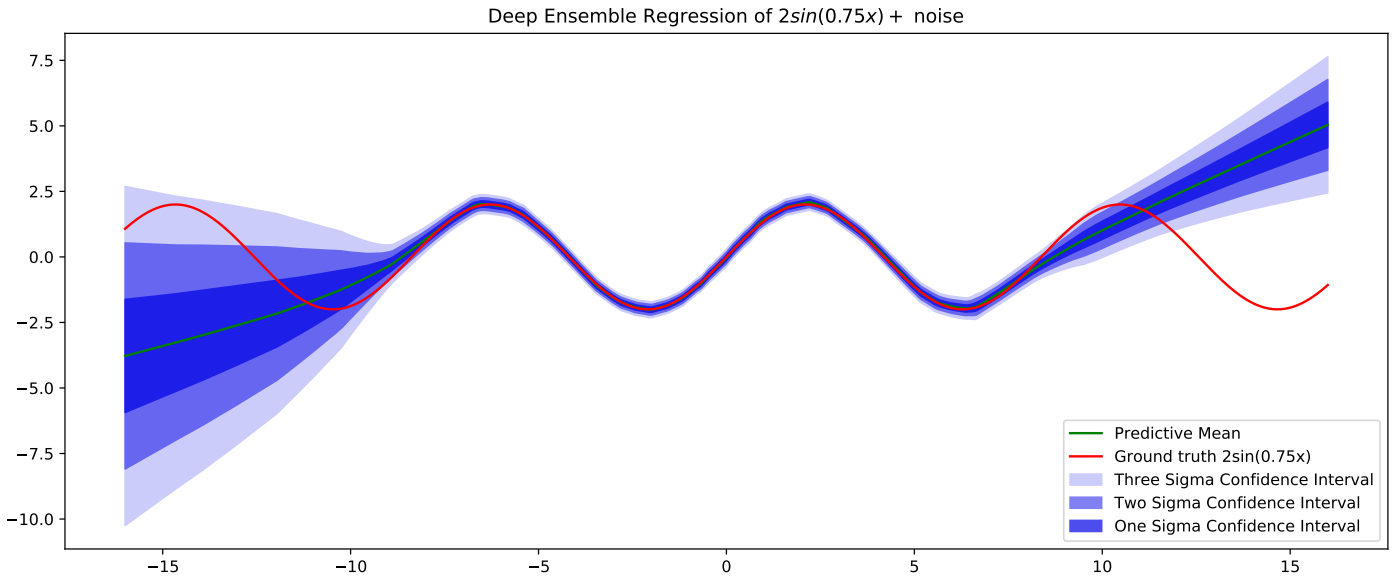
Ensembles - Unseen Examples [Lakshminarayanan et al., 2017]



(a) MNIST-NotMNIST (b) SVHN-CIFAR10

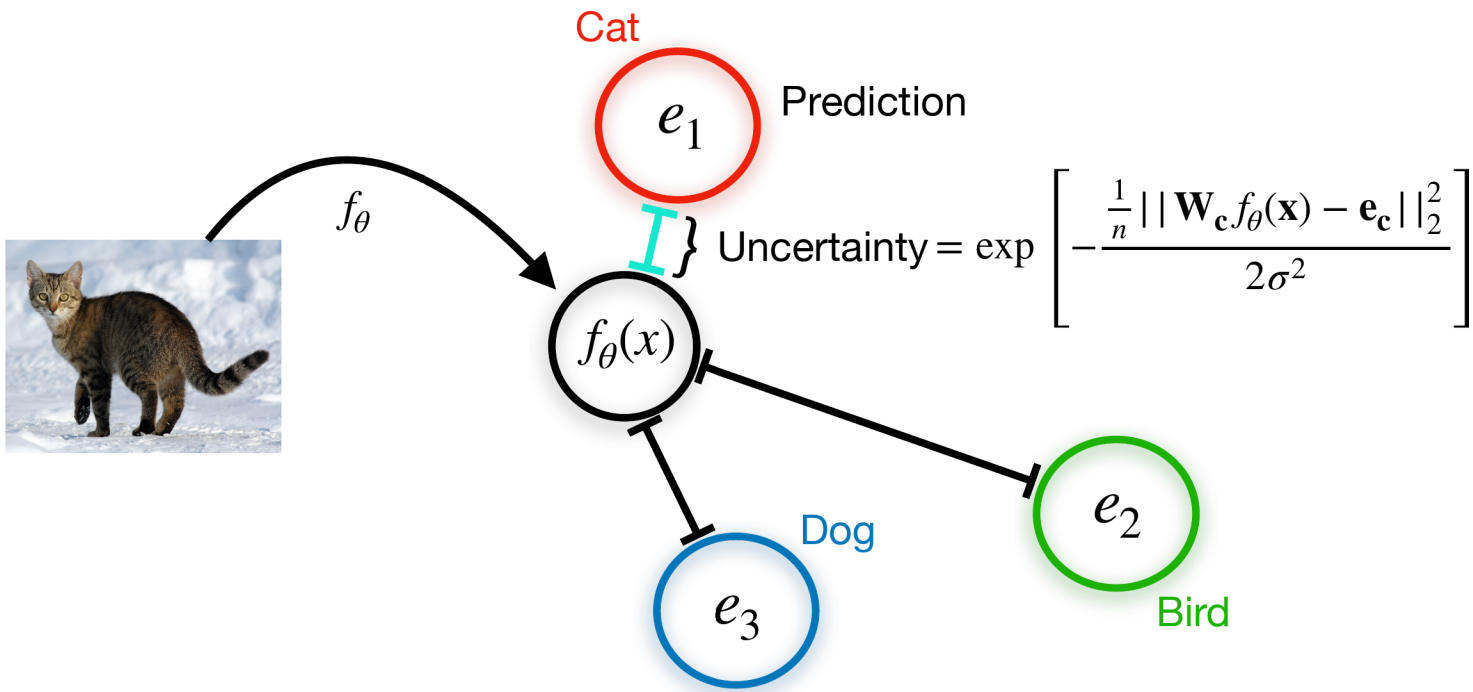
Figure 3: : Histogram of the predictive entropy on test examples from known classes (top row) and unknown classes (bottom row), as we vary ensemble size M.

Ensembles - Sinusoid Regression



DUQ - Direct Uncertainty Quantification [Van Amersfoort et al., 2020]

It is a special kind of output layer that uses a radial basis function with learnable centroids, with distance being proportional to uncertainty.



DUQ - Direct Uncertainty Quantification [Van Amersfoort et al., 2020]

- Predictions are made with:

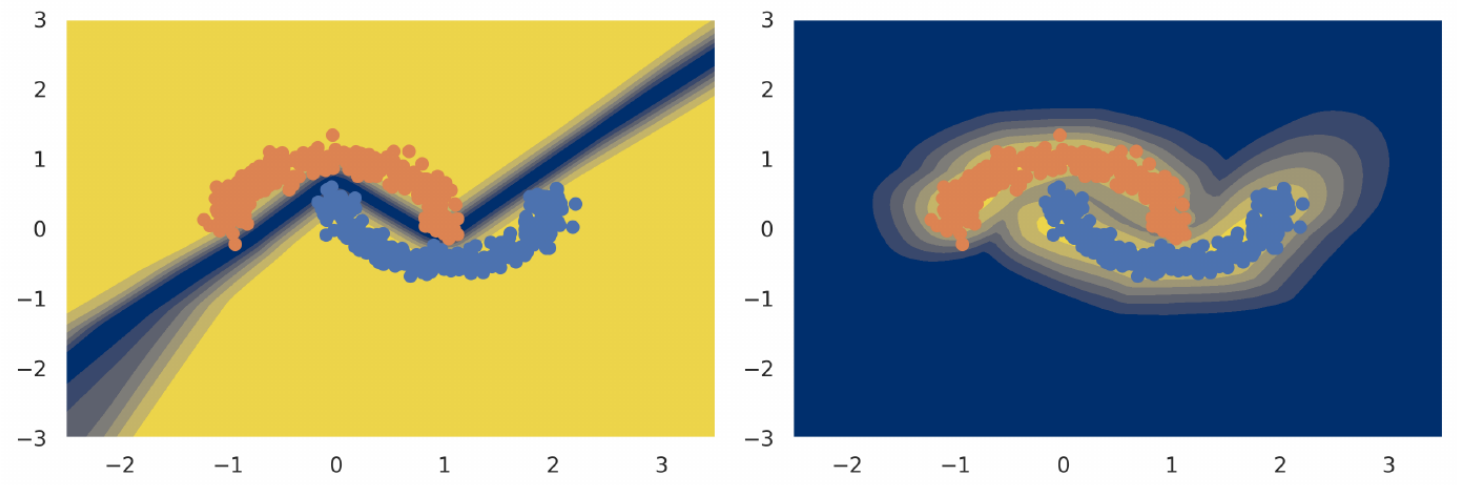
$$K_c(f_\theta(x), e_c) = \exp\left(-\frac{n^{-1}||W_c f_\theta(x) - e_c||_2^2}{2\sigma^2}\right) \tag{6}$$

- And class predictions are made with:

$$\arg\max_c K_c(f_\theta(x), e_c) \tag{7}$$

- Loss is binary cross-entropy. Per-class weight matrix W_c is learned with gradient descent, while centroids e_c are learned with a running mean over input features.
- For small number of samples per class, centroids e_c can also be learned using gradient descent.
- This method is only formulated for classification.

DUQ - Direct Uncertainty Quantification [Van Amersfoort et al., 2020]



(a) Deep Ensembles (b) Our model - DUQ

Figure 1. Uncertainty results on two moons dataset. Yellow indicates high certainty, while blue indicates uncertainty. DUQ is certain only on the data distribution, and uncertain away from it: the ideal result. Deep Ensembles is uncertain only along the decision boundary, and certain elsewhere.

Gradient Uncertainty [Oberdiek et al., 2018]

It has been proposed to use gradient of loss with respect to input as a uncertainty measure:

$$\sigma(x) = ||\nabla L(f(x), y)|| \tag{8}$$

At inference time, no labels y are available, so for the cross-entropy loss, $y = \text{one\_hot}(\hat{y})$, where $\hat{y}$ is the class prediction made by the model.

$||x||$ is a norm over the gradient vector, in order to get a scalar value. Possible choices are L1 and L2 distances, mean, standard deviation, minimum, maximum, etc.

Gradient Uncertainty [Oberdiek et al., 2018]

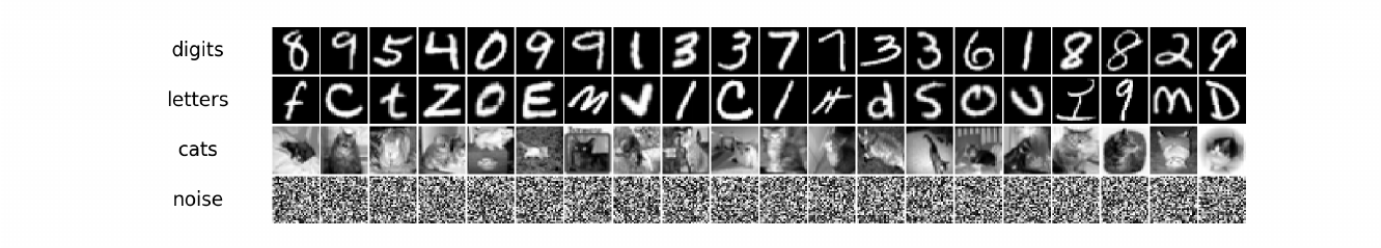


Figure 1: Different concepts used for our statistical experiments

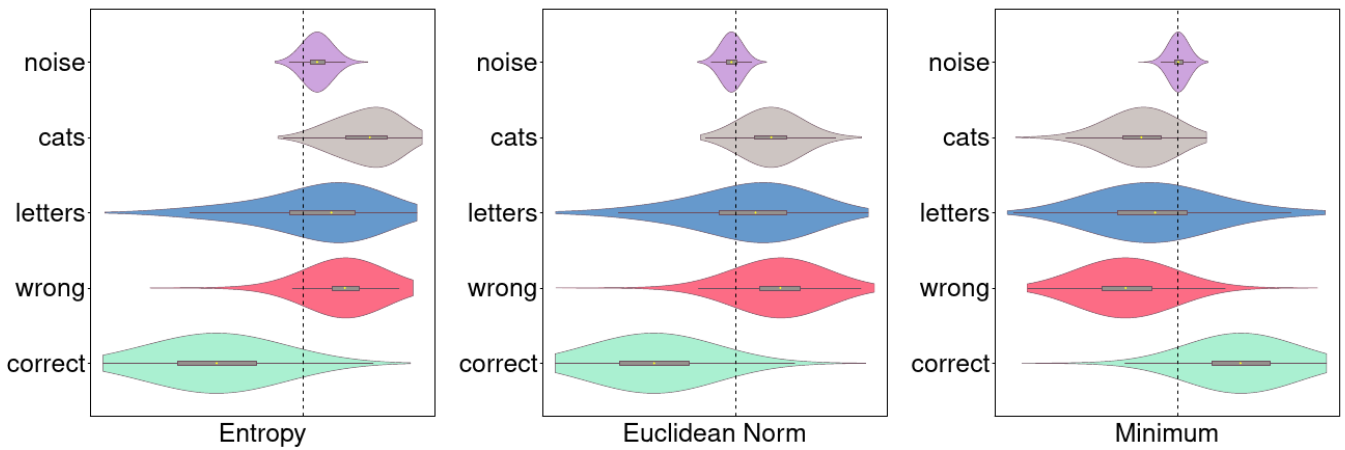


Figure 2: Empirical distribution for entropy, euclidean norm and minimum applied to correctly predicted and incorrectly predicted digits from the test data (green and red) of one CNN. Further distributions are generated from EMNIST samples with unlearned letters (blue), CIFAR10 images (gray) and uniform noise images (purple).

The model here is trained on digits, and tested on a variety of classes to show how gradient uncertainty changes.

Outline

Bayesian Thinking

What is called Bayesian Statistics or Thinking, is the application of Bayes rule to update beliefs, that is, to start from previously known information, update it with an observation, to obtain new information, which can then be repeatedly updated with new observations.

There is a way to do this in a systematic and mathematical way. We will first discuss this approach, and later connect it to uncertainty in machine learning. But the spoiler is that it is how we build Bayesian Neural Networks.

Bayesian Thinking

Given a hypothesis (H) and some evidence (E), we usually have some previous information or assumption about the hypothesis validity, and then we observe some evidence which tells us some information about the hypothesis ($E \mid H$), then we want to find out how this evidence influences our current belief about the hypothesis ($H \mid E$).

This can be formalized as:

$$\mathbb{P}(H \mid E) = \frac{\mathbb{P}(E \mid H)\mathbb{P}(H)}{\mathbb{P}(E)} \tag{9}$$

This is known as Bayes Rule.

Bayesian Thinking - Notation

These terms have standardized names:

$$\overbrace{\mathbb{P}(H \mid E)}^{\text{Posterior}} = \frac{\overbrace{\mathbb{P}(E \mid H)}^{\text{Likelihood}}\overbrace{\mathbb{P}(H)}^{\text{Prior}}}{\underbrace{\mathbb{P}(E)}_{\text{Evidence}}} \tag{10}$$

Or in a simpler way:

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}} \tag{11}$$

Bayesian Thinking - Evidence

The probability of the evidence is hard to compute conceptually, but it can be inferred using the law of total probability:

$$\mathbb{P}(E) = \mathbb{P}(E \mid H)\mathbb{P}(H) + \mathbb{P}(E \mid \neg H)\mathbb{P}(\neg H) \tag{12}$$

This works well if the hypothesis is binary, but not if the hypothesis is more complex, it cannot be enumerated, or if its continuous.

Bayesian Thinking - Concepts

Prior Your previous belief about the hypothesis, or if no information is available, a general assumption about the hypothesis.

Likelihood The probability of observing the evidence given the hypothesis, this measures the compatibility of the evidence with the hypothesis. Note that this is a function of the evidence.

Posterior Your updated belief about the hypothesis, after seeing the evidence. Note that this is a function of the hypothesis.

Bayesian Thinking - Example

Let's assume that you travel somewhere and then feel sick. You take a SARS-CoV-2 test, and it returns positive. You get worried, but what is the chance that you are actually infected?

The hypothesis is are you infected with SARS-COV-2? Possible values are yes or no (binary).

The evidence is your test, which is not perfect, let's assume that it is 80% correct. That is $\mathbb{P}(E \mid H) = 0.8$, you have 80% evidence given that you are infected (this is the commonly reported metric for accuracy in a test).

The prior is the ratio of infected people over the whole population, let's assume it to be 0.1%. Then $\mathbb{P}(H) = 0.001$.

Bayesian Thinking - Example

Then first we compute $\mathbb{P}(E)$ as:

$$\begin{aligned}\mathbb{P}(E) &= \mathbb{P}(E \mid H)\mathbb{P}(H) + \mathbb{P}(E \mid \neg H)\mathbb{P}(\neg H) \\ &= 0.8 \times 0.001 + 0.2 \times 0.9999 = 0.20078\end{aligned}$$

Then we can use Bayes rule to compute $\mathbb{P}(H \mid E)$ and obtain an updated belief:

$$\begin{aligned}\mathbb{P}(H \mid E) &= \frac{\mathbb{P}(E \mid H)\mathbb{P}(H)}{\mathbb{P}(E)} \\ &= \frac{0.8 \times 0.001}{0.20078} = 0.00398 \sim 0.4\%\end{aligned}$$

The posterior probability of being infected is only 0.4%. This is only 4 times higher than the prior probability.

Bayesian Thinking - Example

As a thought experiment, what is $\mathbb{P}(\neg H \,|\, \neg E)$?
For this we need some further information, which is $\mathbb{P}(\neg E \,|\, \neg H) = 0.8$, as test accuracy is defined for both testing positive and being healthy, and testing negative and being sick, and these two terms are equalized.

$$\begin{aligned}\mathbb{P}(\neg H \,|\, \neg E) &= \frac{\mathbb{P}(\neg E \,|\, \neg H)\mathbb{P}(\neg H)}{\mathbb{P}(\neg E)} \\ &= \frac{0.8 \times (1 - 0.001)}{1 - 0.20078} = 0.9999 \sim 99.99\%\end{aligned}$$

This is the probability of not begin infected given that the test is negative, and it is $< 99\%$, due to the low prevalence of the disease.

Bayesian View of Uncertainty

Bayesian statistics has a particular view of uncertainty.

Typically when learning a model, we wish to learn the probability distribution $P(\mathbf{y} \mid \mathbf{x})$, that is, the probability of some output given the input.

But this does not consider the model parameters $\mathbf{w}$ (also known as weights), which in the end indirectly encode uncertainty.

Then we wish to learn $P(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$ which directly considers the model parameters. As model parameters are learned from data $\mathbf{D}$, then we want to learn $P(\mathbf{w} \mid \mathbf{D})$ (this is what training a model does).

Bayesian View of Uncertainty

$$\mathbb{P}(\mathbf{w} \mid \mathbf{D}) = \frac{\mathbb{P}(\mathbf{D} \mid \mathbf{w})\mathbb{P}(\mathbf{w})}{\mathbb{P}(\mathbf{D})} \tag{13}$$

Here $\mathbf{D} = \{x_i, y_i\}$ is a labeled training set but more generally it is the concept of data.

Bayes Rule' allows us to decompose the probability of the weights given the data (which is unknown and we want to estimate it) into probability of the data given weights (computed by the model).

Bayesian View of Uncertainty - Interpretation

Prior This is $\mathbb{P}(\mathbf{w})$, the prior distribution on the weights. This is generally assumed. This has direct connection with the random weight initializers that are used in ML/NN models.

Likelihood This is $\mathbb{P}(\mathbf{D} \mid \mathbf{w})$, which is a forward pass of the model, which computes $\mathbb{P}(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$ and for this it needs weights $\mathbf{w}$. These terms are defined by your model architecture/equations.

Posterior This is $\mathbb{P}(\mathbf{w} \mid \mathbf{D})$, the posterior distribution of the weights given the data, which is the actual learning algorithm, and later can be used to make predictions.

Bayesian View of Uncertainty

- From the Bayesian POV, a model is defined by both the structure (the actual model equations) and the model's parameters.
- This means that for a Bayesian model, the model parameters encode the **model uncertainty** about each prediction. There are equations to compute the output probability distribution from the model and its inputs.
- Intuitively, this can be seen as a simple question. If we have several models trained on the same data, the model's parameters won't be the same. Some parameters might be very close, some will be equal, and some will be radically different.
- These variations on model parameters can be encoded as a probability distribution. Bayesian statistics can be used to estimate these distributions $P(\mathbf{w} \mid \mathbf{D})$

Bayesian Neural Networks

So what exactly is a Bayesian Neural Network? It should have two components:

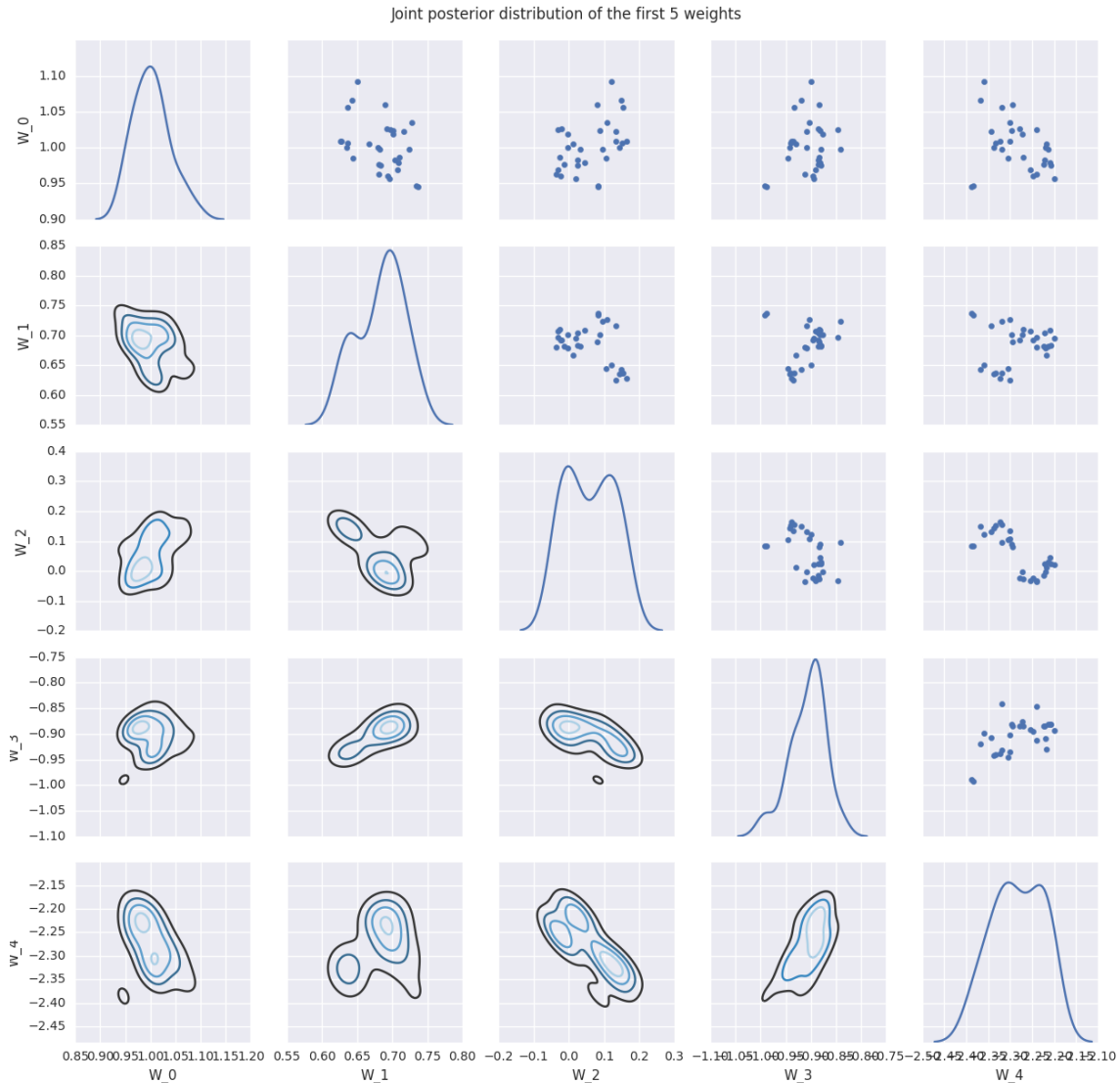
Weights / Parameters

The weights of layers in the model are probability distributions. There can be point-wise weights, but the defining characteristic is probability distribution over weights. Does not have to be in all layers.

Bayesian Learning

The Learning algorithm must consider that it learns probability distribution weights instead of point-wise weights. In a strict sense all learning algorithms are an application of Bayes rule.

Bayesian View of Uncertainty (on MNIST with SGHMC)



This is an example of learned weight distributions for an MNIST classifier (using a secret algorithm).

Note how the distributions are not Gaussian.

Bayesian Learning

Maximum Likelihood Estimation Maximize the log-likelihood given model parameters.

$$
\begin{aligned}
w^{\text{MLE}} &= \arg\max_{w} \log P(\mathbf{D} \mid \mathbf{w}) \\
&= \arg\max_{w} \sum_{i} P(y_i \mid x_i, \mathbf{w})
\end{aligned}
$$

Maximum A Posteriori MLE formulation plus a regularization term $P(\mathbf{w})$, which is a prior on model parameters.

$$
\begin{aligned}
w^{\text{MAP}} &= \arg\max_{w} \log P(\mathbf{w} \mid \mathbf{D}) \\
&= \arg\max_{w} \left[P(\mathbf{D} \mid \mathbf{w}) + P(\mathbf{w}) \right]
\end{aligned}
$$

Both of these formulations only learn what is called **point-wise** estimates of the parameters.

Bayesian Predictive Posterior

If the distribution over weights is obtained, then output distributions can be computed with the bayesian predictive posterior distribution:

$$\mathbb{P}(\mathbf{y} \mid \mathbf{x}) = \int_{\mathrm{w}} \mathbb{P}(\mathbf{y} \mid \mathbf{w}, \mathbf{x}) \mathbb{P}(\mathbf{w} \mid \mathbf{D}) \, d\mathbf{w} \tag{14}$$

This equation is obtained by marginalizing over all possible model parameters $\mathbf{w}$. This equation basically computes predictions $\mathbf{y}$ with different model parameters $\mathbf{w}$ and weights them by the probability of those parameters given the input $\mathbf{x}$.

This can also be considered as a form of Bayesian Model Averaging.

Bayesian Predictive Posterior - Interpretation

$$\mathbb{P}(\mathbf{y} \mid \mathbf{x}) = \int_{\mathbf{w}} \overbrace{\mathbb{P}(\mathbf{y} \mid \mathbf{w}, \mathbf{x})}^{\text{Forward Pass}} \quad \underbrace{\mathbb{P}(\mathbf{w} \mid \mathbf{D})}_{\text{Posterior of Weights}} \quad d\mathbf{w} \qquad (15)$$

Forward Pass This is one forward pass through your neural network, that outputs a probability distribution given inputs and weights.

Posterior of Weights This is learned using a learning algorithm, and is basically the result of learning using Bayes rule.

A simple interpretation of this equation is that you integrate the product of a forward pass (given a specific set of weights) and the probability of that set of weights, over all weight values.

Monte Carlo Approximation of Bayesian Posterior

In general cases we cannot compute the previous integral for the Bayesian Predictive Posterior distribution, but it can be approximated using Monte Carlo with M samples.

$$\mathbb{P}(\mathbf{y} \,|\, \mathbf{x}) \sim M^{-1} \sum_{i}^{M} \mathbb{P}(\mathbf{y} \,|\, \theta_i, \mathbf{x}) \mathbb{P}(\theta_i \,|\, \mathbf{D}) \quad \theta_i \sim \mathbb{P}(\mathbf{w} \,|\, \mathbf{D}) \tag{16}$$

If only forward passes $\mathbb{P}_i$ that produce samples of the posterior are available, then a more rough approximation can be:

$$\mathbb{P}(\mathbf{y} \,|\, \mathbf{x}) \sim M^{-1} \sum_{i}^{M} \mathbb{P}_i(\mathbf{y} \,|\, \mathbf{w}, \mathbf{x}) \tag{17}$$

In this case we are not marginalizing over weights but over abstract stochastic passes, which is a worse approximation but computationally tractable. Larger M means better approximation.

Intractability of BNNs

If we consider Bayes Rule, the term $\mathbb{P}(E)$ is generally difficult to compute.

In a Bayesian Neural Network, the equivalent term is $\mathbb{P}(D)$, which is the probability distribution of the whole data (not the dataset), which in general is impossible to compute.

For example if we use images of a given size, $\mathbb{P}(D)$ is the distribution of all images of that size, which we cannot estimate due to a lack of data, and a lack of methods due to its high dimensionality.

Other Bayesian NN Issues

The biggest issue with Bayesian learning methods is how to encode or represent the posterior probability distributions over the weights. Since typical usable models have millions of parameters, the distribution is very high-dimensional.

There is the issue of computational complexity. Integrating over millions of parameters and performing *multiple predictions* for each of these parameters is computationally infeasible.

In general there are no closed form representations for posterior distribution over weights, and consequently there are also no closed form computations of the Bayesian predictive posterior distribution. The only alternative is to represent the distribution with samples (histograms), and to use Monte Carlo methods to sample from the posterior distribution.

Outline

Sampling Methods for Uncertainty Quantification

- These are methods that approximate posterior probability distributions by sampling or by representing these distributions with histograms of samples.
- In general they are expensive, as they have to repeat a number of computations for each sample, but they provide very good estimates of uncertainty.
- Also generally a large number of samples (100-1000) are required to well approximate these posterior distributions, which also adds into the computation time.

Monte Carlo Dropout [Gal and Ghahramani, 2016]

- Its a interpretation of applying Dropout in a neural network, but at **inference time**.
- Yarin Gal made theoretical proofs that showed that using Dropout at inference time is approximately equivalent to sampling the predictive posterior distribution, under some assumptions.
- This has connections to Bayesian model averaging, as each forward pass using Dropout samples a different model, which may make different predictions. The mean and variance can be estimated from multiple samples as:

$$\mu(x) = N^{-1} \sum_i f_i(x) \qquad \sigma^2(x) = (N-1)^{-1} \sum_i (f_i(x) - \mu(x))^2$$

Monte Carlo Dropout [Gal and Ghahramani, 2016]

- Dropout is a well known technique for regularization of Neural Networks.
- During training, a mask $m_i \sim \text{Bernoulli}(p)$ is drawn and multiplied with the input activations, effectively making some of them zero.
- Dropout can also be enabled at inference time, where it has been proven that it works as an approximation of the predictive posterior distribution.

Monte Carlo DropConnect [Mobiny et al., 2021]

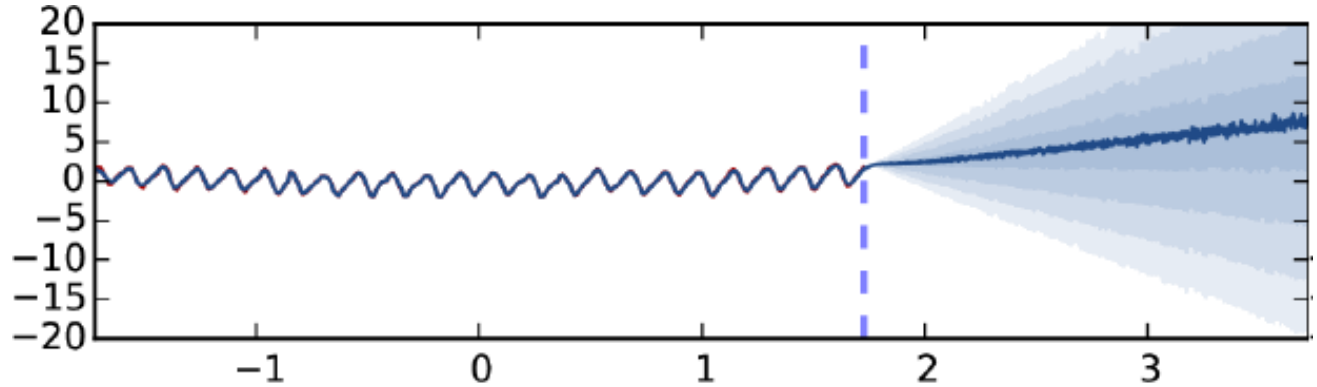
- DropConnect is a variation of Dropout, where instead of applying a mask to the activations of a layer, it is applied to the `weights` of a layer.
- It has been proven to also produce an approximation of the predictive posterior distribution. It requires the implementation of new layers that use DropConnect internally.
- In some cases it outperforms MC Dropout in both task and uncertainty performance, but not always.

What about Monte Carlo?

- Enabling Dropout or DropConnect at inference transforms the neural network into a stochastic model.
- This means each forward pass produces a different result, a sample from the predictive posterior distribution.
- The model with uncertainty can be evaluated by combining the predictions from M forward passes.
- This is the Monte Carlo version of the predictive posterior distribution:

$$p(y \mid x) \sim M^{-1} \sum_{i}^{M} p(y \mid x, \theta_i) \quad \text{where} \quad \theta_i \sim \Theta$$

Monte Carlo Dropout - Regression on Mauna Loa CO2 [Gal and Ghahramani, 2016]



(c) MC dropout with ReLU non-linearities

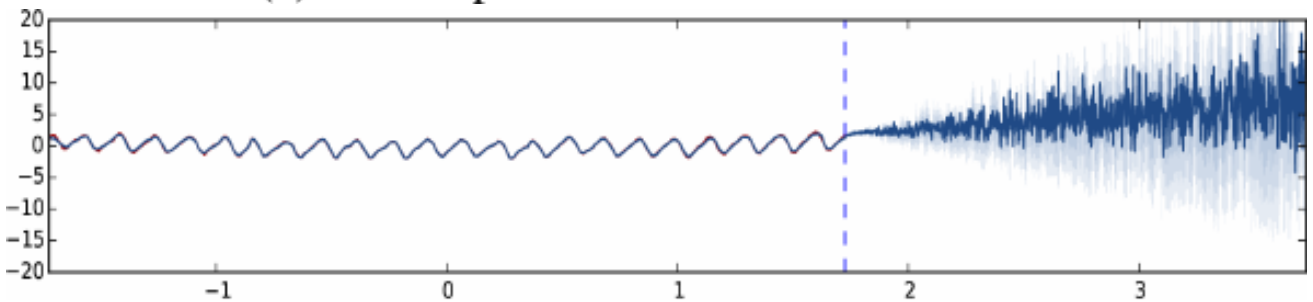
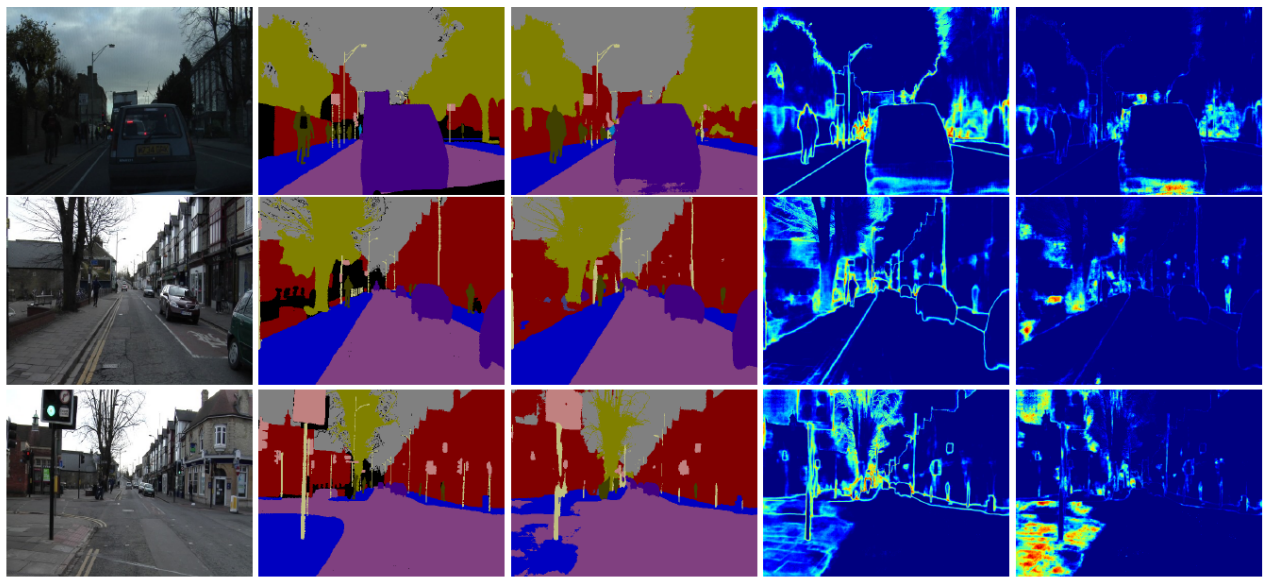


Figure 3. Predictive mean and uncertainties on the Mauna Loa CO_2 concentrations dataset for the MC dropout model with ReLU non-linearities, approximated with 10 samples.

Monte Carlo Dropout - Semantic Segmentation [Kendall and Gal, 2017]



(a) Input Image (b) Ground Truth (c) Semantic Segmentation (d) Aleatoric Uncertainty (e) Epistemic Uncertainty

Figure 1: **Illustrating the difference between aleatoric and epistemic uncertainty** for semantic segmentation on the CamVid dataset [8]. *Aleatoric* uncertainty captures noise inherent in the observations. In (d) our model exhibits increased aleatoric uncertainty on object boundaries and for objects far from the camera. *Epistemic* uncertainty accounts for our ignorance about which model generated our collected data. This is a notably different measure of uncertainty and in (e) our model exhibits increased epistemic uncertainty for semantically and visually challenging pixels. The bottom row shows a failure case of the segmentation model when the model fails to segment the footpath due to increased epistemic uncertainty, but not aleatoric uncertainty.

Monte Carlo Dropout - Depth Regression [Kendall and Gal, 2017]

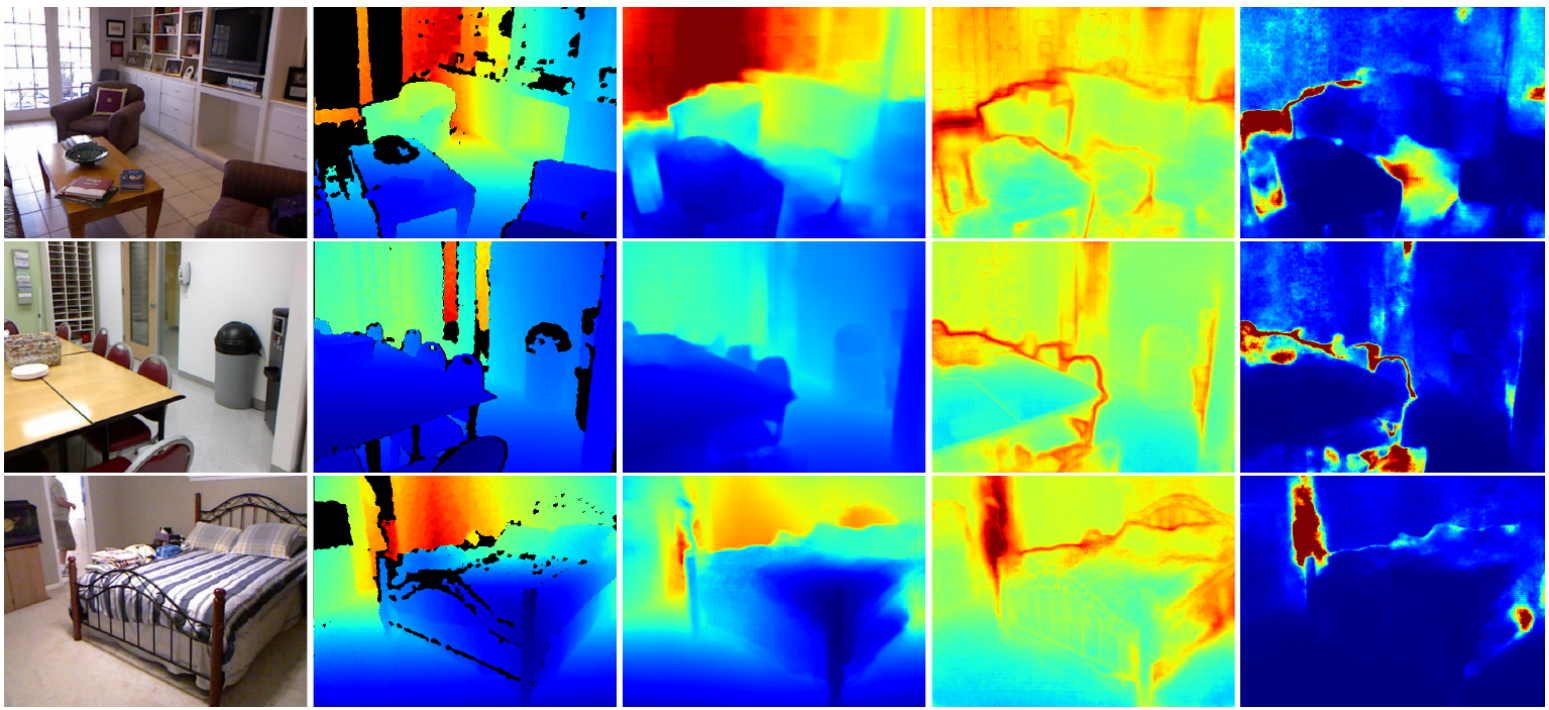
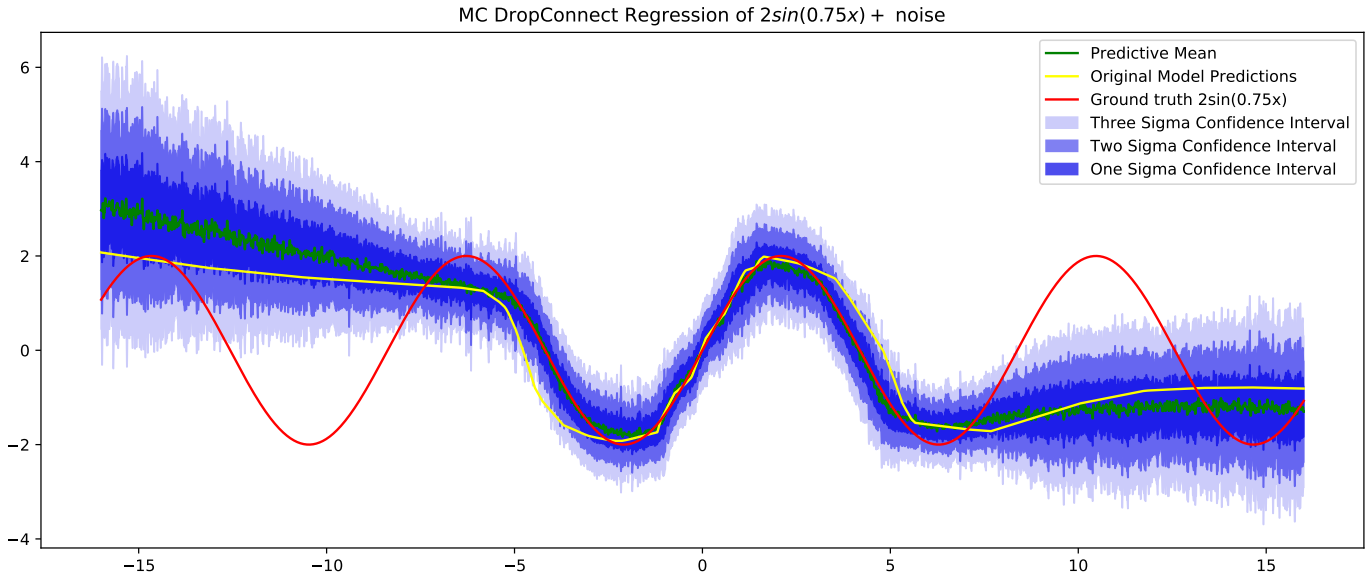


Figure 5: NYUv2 Depth results. From left: input image, ground truth, depth regression, aleatoric uncertainty, and epistemic uncertainty.

MC-DropConnect - Sinusoid Regression



Questions to Think About

- Explain the overall Bayesian Neural Network framework.
- What is the intuition for the Gaussian Negative Log-Likelihood?
- How does MC-DropConnect work? and how it is different from MC-Dropout?
- What is the difference between direct and sampling-based UQ methods?
- What are the disadvantages of the Bayesian framework applied to Neural Networks?

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:

$$\mathrm{KL}(q, p) = \int_x q(x) \log \frac{q(x)}{p(x)} \tag{18}$$

Some important properties:

- $\mathrm{KL}(q, p) \geq 0$ 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 $\mathrm{KL}(q, p) \geq 0$:

$$\log Z(\theta) = \mathrm{KL}(q, p) - J(q) \geq -J(q) \tag{19}$$

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:

$$\log Z(\theta) \geq \mathbb{E}_{q(x)}[\log \tilde{p}(x) - \tilde{q}(x)] \tag{20}$$

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 while also maximizing $\log p(x)$.

Variational Inference

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

$$L(\theta) = \mathrm{KL}(q_\theta(w \mid D), \mathbb{P}(w)) - \mathbb{E}_{q_\theta(w \mid D)}[\log \mathbb{P}(D \mid w)] \quad (21)$$

The terms in this loss function (ELBO) 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 (task) 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.

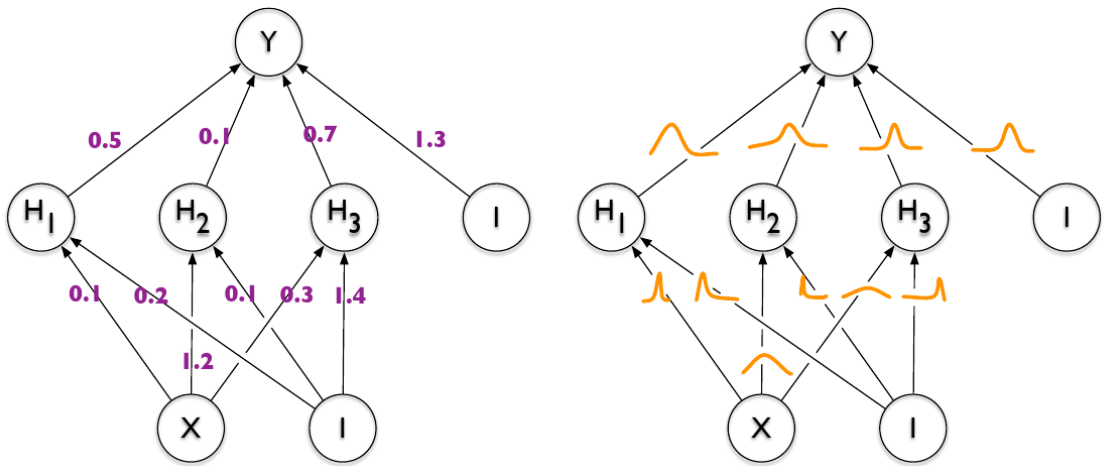


Figure 1. Left: each weight has a fixed value, as provided by classical backpropagation. Right: each weight is assigned a distribution, as provided by Bayes by Backprop.

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

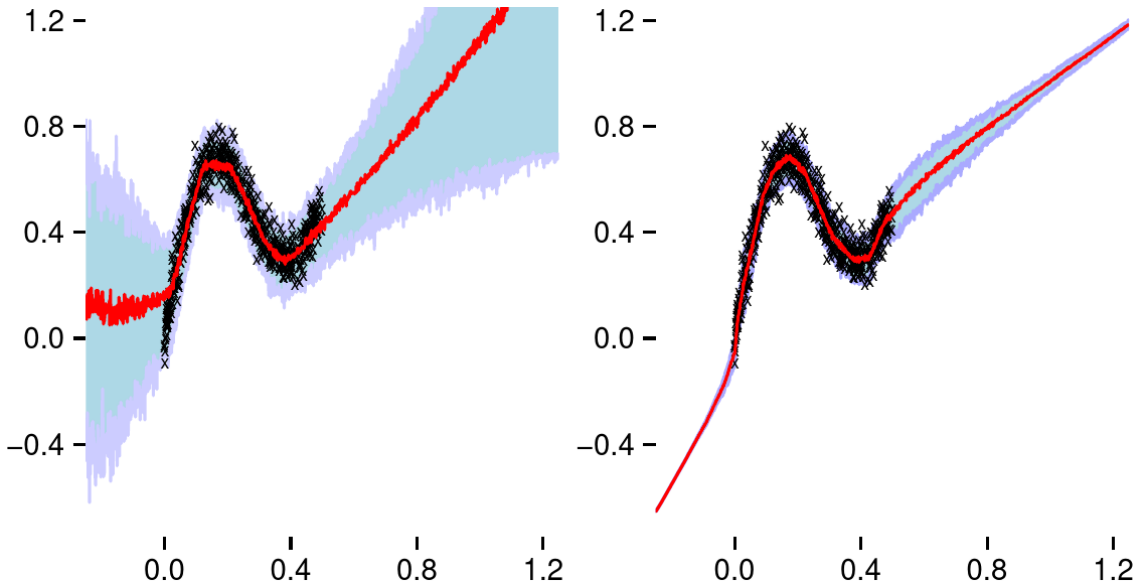
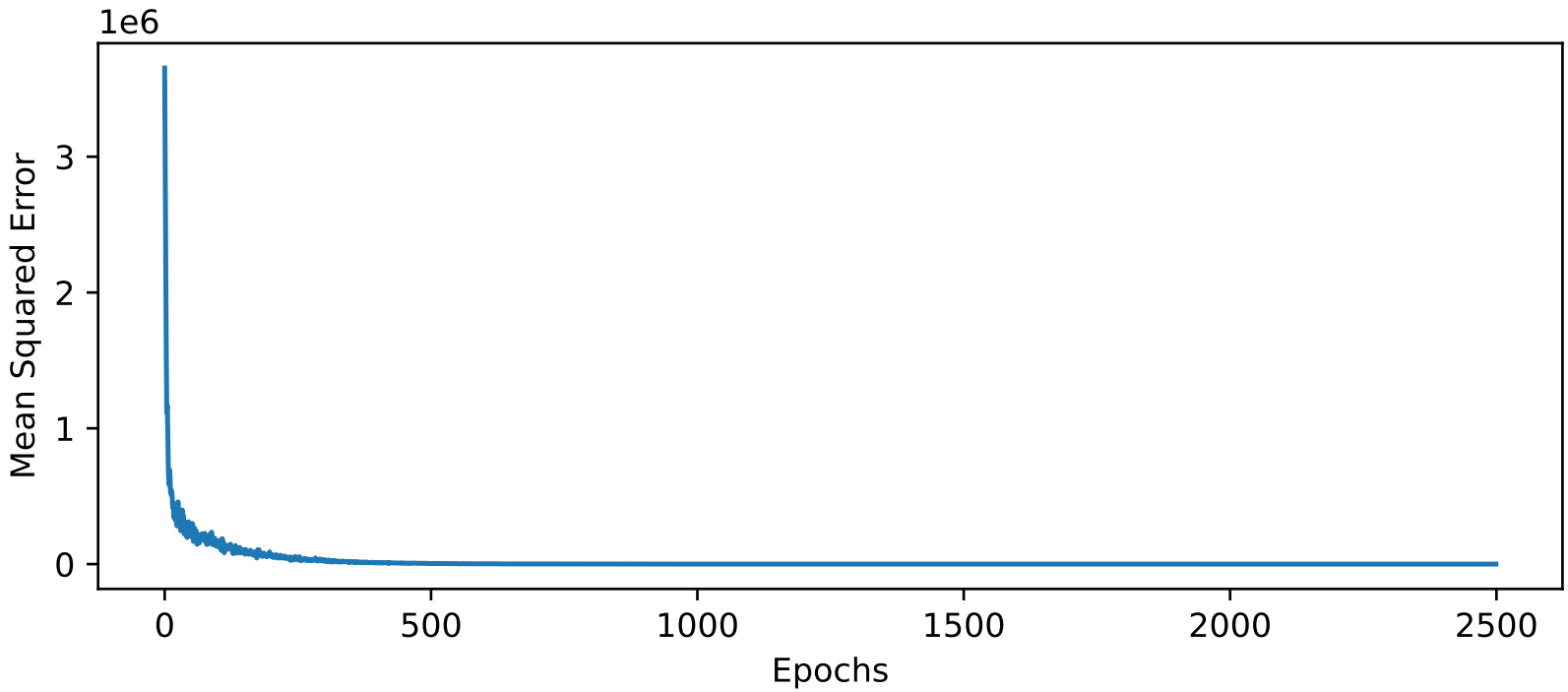


Figure 5. Regression of noisy data with interquatile ranges. Black crosses are training samples. Red lines are median predictions. Blue/purple region is interquartile range. Left: Bayes by Backprop neural network, Right: standard neural network.

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:

$$y = a(Wx + b) \qquad W \sim \mathbb{P}(w_W \mid D), b \sim \mathbb{P}(w_b \mid D) \qquad (22)$$

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:

$$w = \mu + \sigma z \text{ with } z \sim \text{Norm}(0, 1), \rightarrow w \sim \text{Norm}(\mu, \sigma) \quad (23)$$

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.:

$$\Delta W_n = \Delta\hat{W} r_n s_n^T \tag{24}$$

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]

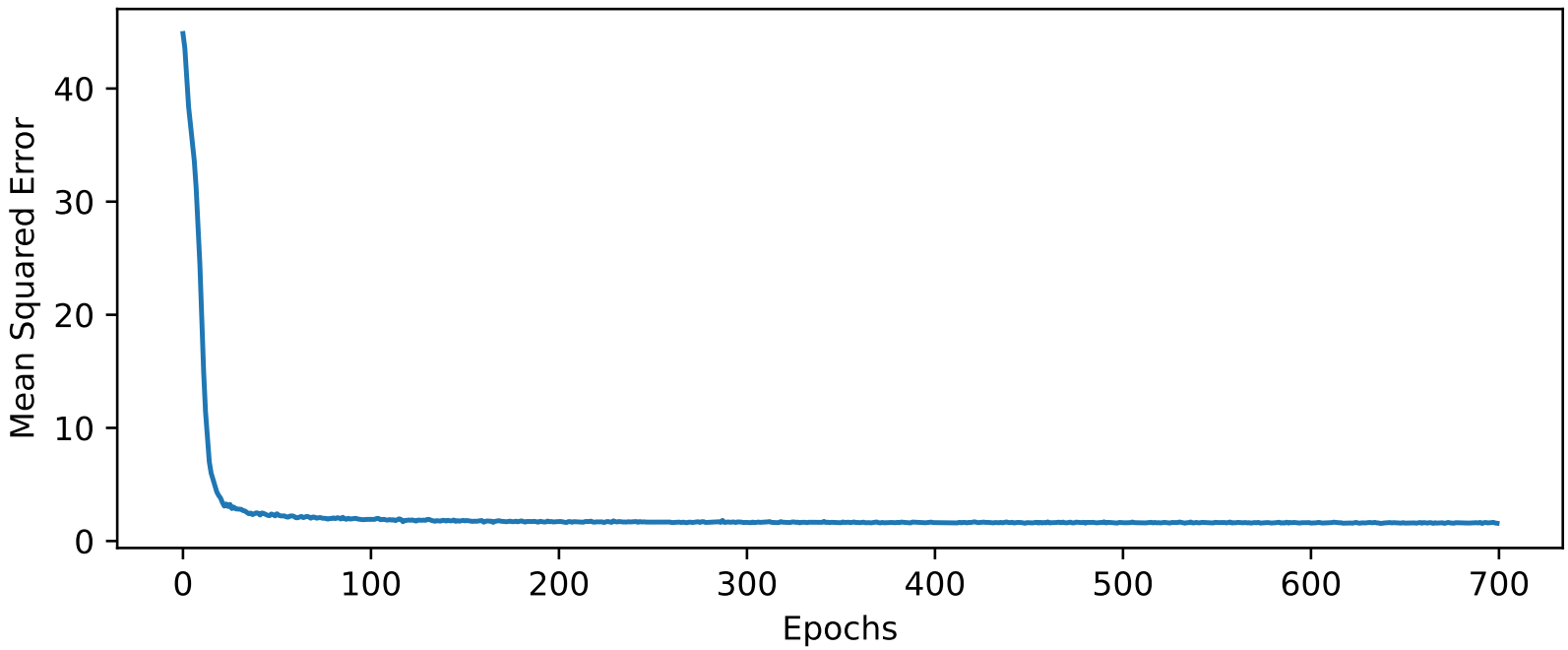
Why do this transformation of the perturbation?

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.

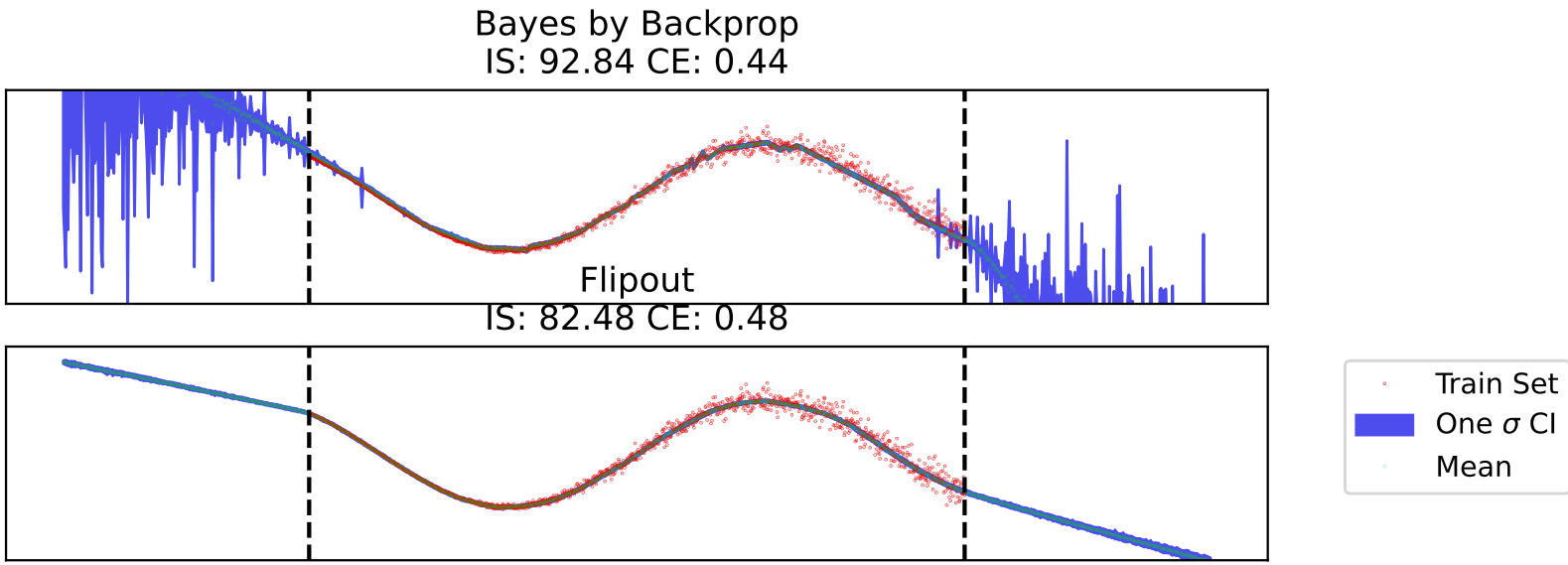
But now there is an advantage of decorrelating samples inside a batch.

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:

$$\mathbb{P}(\mathbf{y} \mid \mathbf{x}) \sim M^{-1} \sum_{i}^{M} \mathbb{P}(\mathbf{y} \mid \mathbf{w}_i', \mathbf{x}) \mathbb{P}(\mathbf{w}_i' \mid \mathbf{x}) \quad \mathbf{w}_i' \sim \mathbb{P}(\mathbf{w} \mid \mathbf{x}) \tag{25}$$

Here the Posterior distribution of the weights $\mathbb{P}(\mathbf{w} \mid \mathbf{x})$ is sampled to produce weights $\mathbf{w}'$, 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:

$$\mathbb{P}(\mathbf{y} \mid \mathbf{x}) \sim M^{-1} \sum_{i}^{M} \mathbb{P}(\mathbf{y} \mid \mathbf{w}_i', \mathbf{x}) \quad \mathbf{w}_i' \sim \mathbb{P}(\mathbf{w} \mid \mathbf{x}) \tag{26}$$

This one does not weight using the weight posterior, for cases where the weight probabilities $\mathbb{P}(\mathbf{w}_i' \mid \mathbf{x})$ 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.

$$\text{Predictive Uncertainty} = \text{Epistemic} + \text{Aleatoric} \tag{27}$$

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.

Epistemic 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.

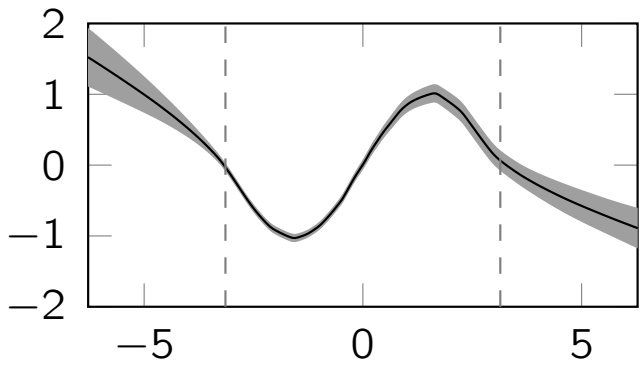
Uncertainty Disentanglement - Regression

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

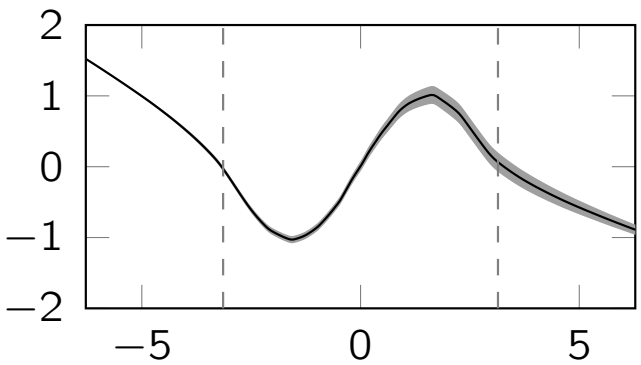
$$\begin{aligned}
\sigma_*^2(\mathbf{x}) = & \quad M^{-1}\sum_i \sigma_i^2(\mathbf{x}) & + & \quad M^{-1}\sum_i \mu_i^2(\mathbf{x}) - \mu_*^2(\mathbf{x}) \\
= & \quad \mathbb{E}_i[\sigma_i^2(\mathbf{x})] & + & \quad \mathbb{E}_i[\mu_i^2(\mathbf{x})] - \mathbb{E}_i[\mu_i(\mathbf{x})]^2 \\
= & \quad \underbrace{\mathbb{E}_i[\sigma_i^2(\mathbf{x})]}_{\text{Aleatori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

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.

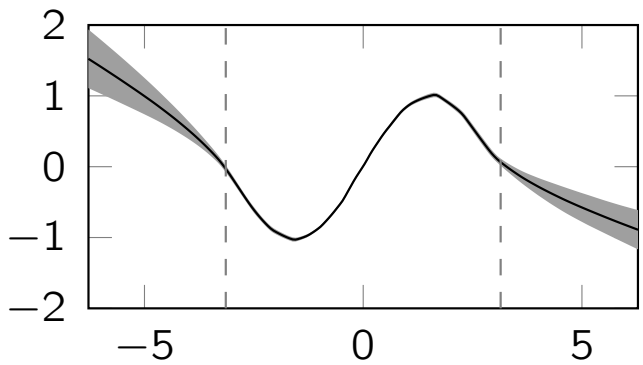
Uncertainty Disentanglement - Regression



(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 - Intuition

Considering the previous equation, what is their intuition?

Aleatoric Uncertainty

Aleatoric Uncertainty is given by the mean of per-sample/ensemble variances, which can be interpreted as combining all variances into a single variance.

Epistemic Uncertainty

Epistemic Uncertainty is given by the variance of per-sample/ensemble means. This can be interpreted as the variation or *disagreement* between models/samples, which is by definition epistemic uncertainty.

If all means are the same, the model(s) is very confident. If all means are different and varied, the model(s) disagree and this signals epistemic uncertainty.

Uncertainty Disentanglement - Classification

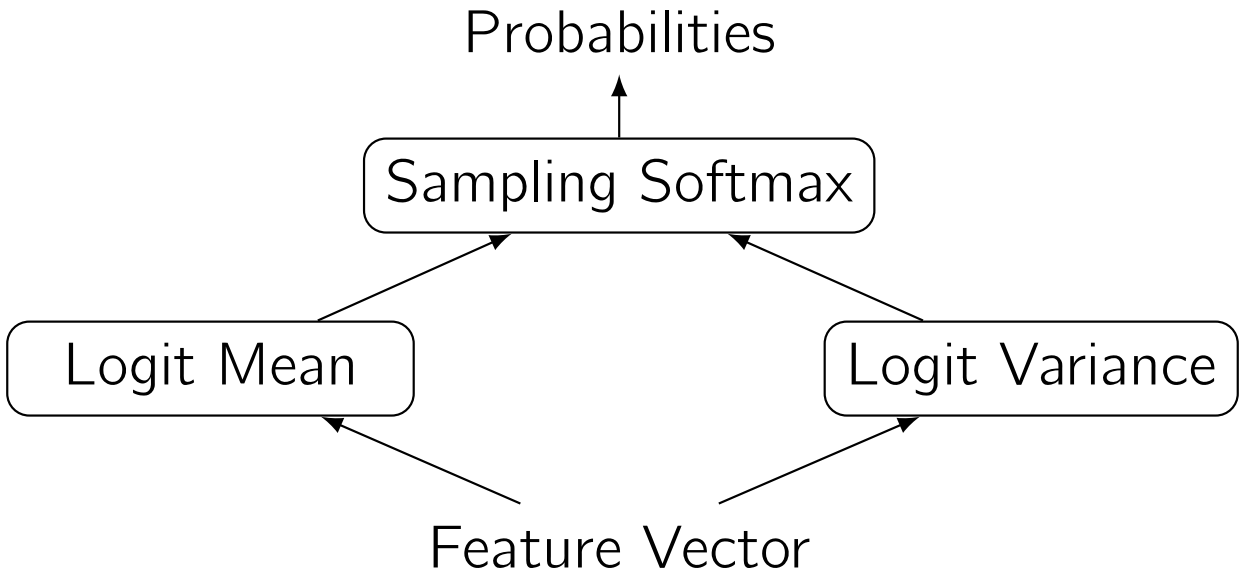
Aleatoric Uncertainty

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.

Epistemic 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.

$$\hat{\mathbf{z}}_j \sim \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x})) \tag{28}$$

$$\mathbb{P}(\mathbf{y} \mid \mathbf{x}) = N^{-1} \sum_j \text{softmax}(\hat{\mathbf{z}}_j) \qquad j \in [1, N] \tag{29}$$

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

$$\sigma^2_{\text{Ale}}(\mathbf{x}) = \mathbb{E}_i[\sigma_i^2(\mathbf{x})] \qquad \sigma^2_{\text{Epi}}(\mathbf{x}) \qquad = \text{Var}_i[\mu_i(\mathbf{x})]. \qquad (30)$$

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:

$$
\text{entropy}(p) = -\sum_i p_i \log p_i \tag{31}
$$

Uncertainty Disentanglement - Classification

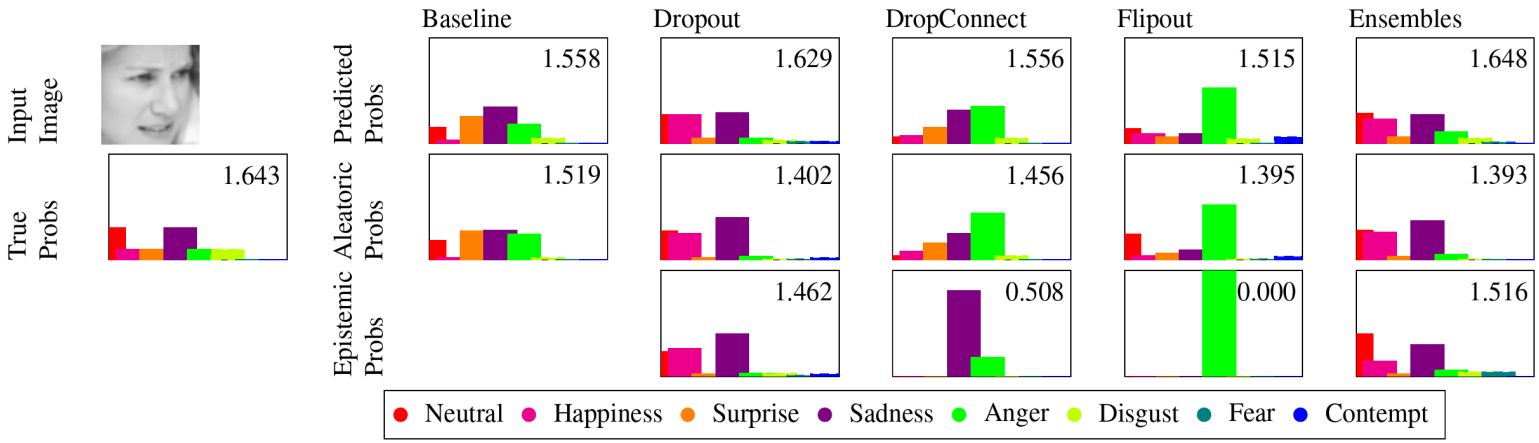


Figure 12. Facial image 813 from the FER+ test set. Here we show the aleatoric and epistemic probabilities produced by different uncertainty quantification methods. The entropy of each distribution is displayed in the top right corner. Ensembles has the highest epistemic uncertainty, while Flipout has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

$$\text{Predictive Var} = \text{Epistemic Var} + \text{Aleatoric Var} \tag{32}$$

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.

$$\text{Predictive Logits} = \text{Epistemic Logits} + \text{Aleatoric Logits} \tag{33}$$

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

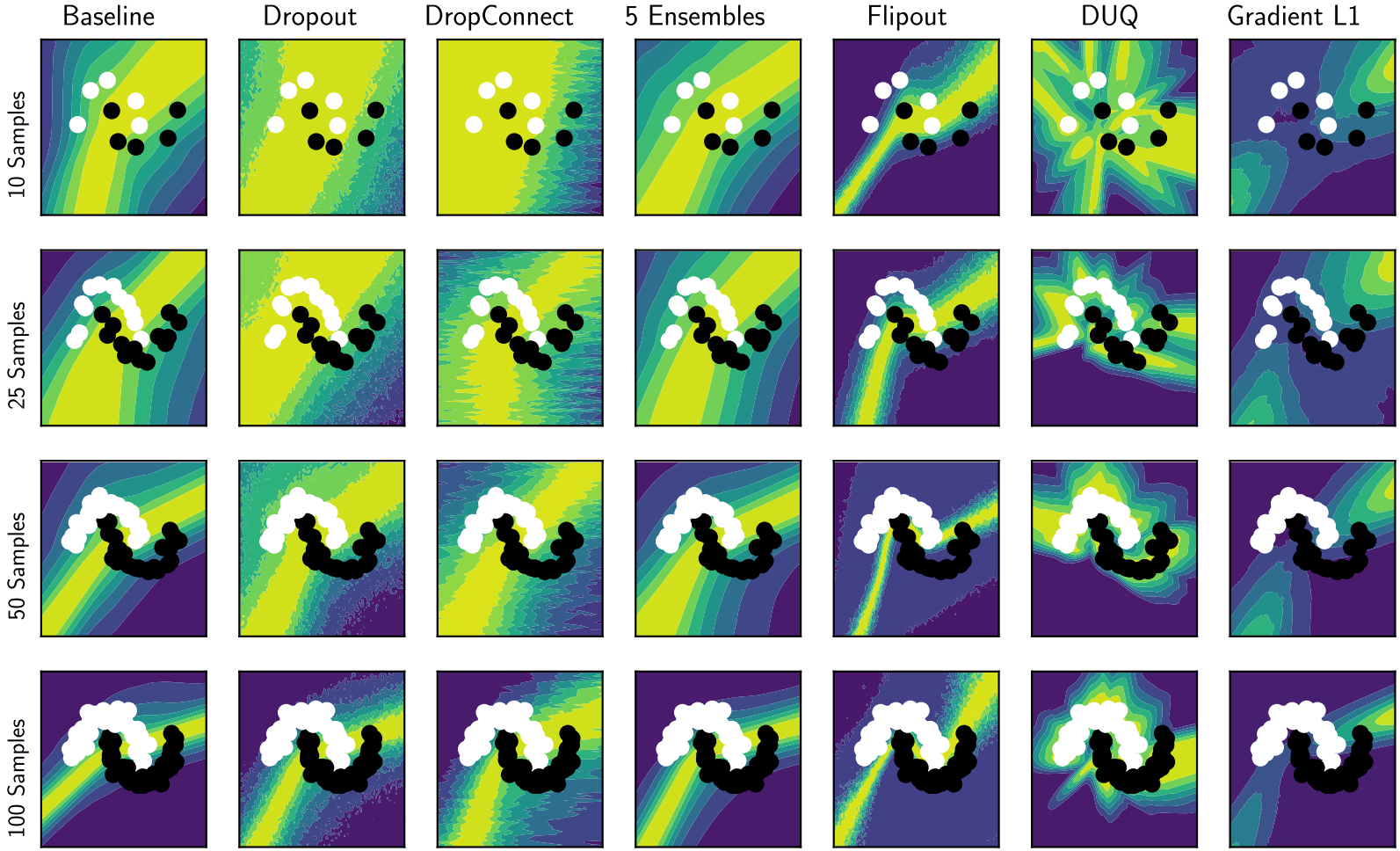
Comparison Across Train Set Size[Valdenegro-Toro, 2021]

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".

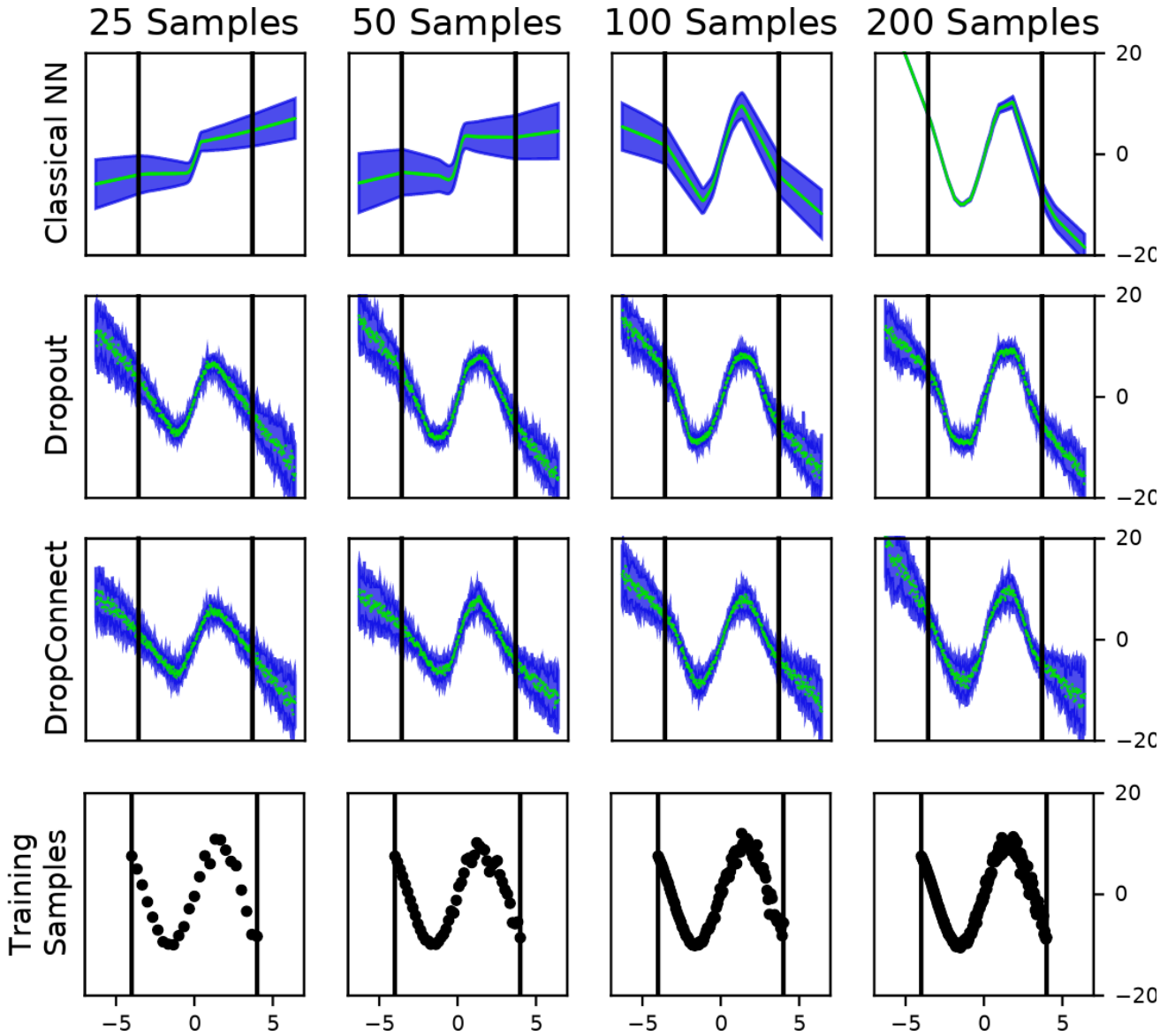
Here we compare multiple uncertainty quantification methods across different sizes of the training set, and obtain some interesting conclusions.

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

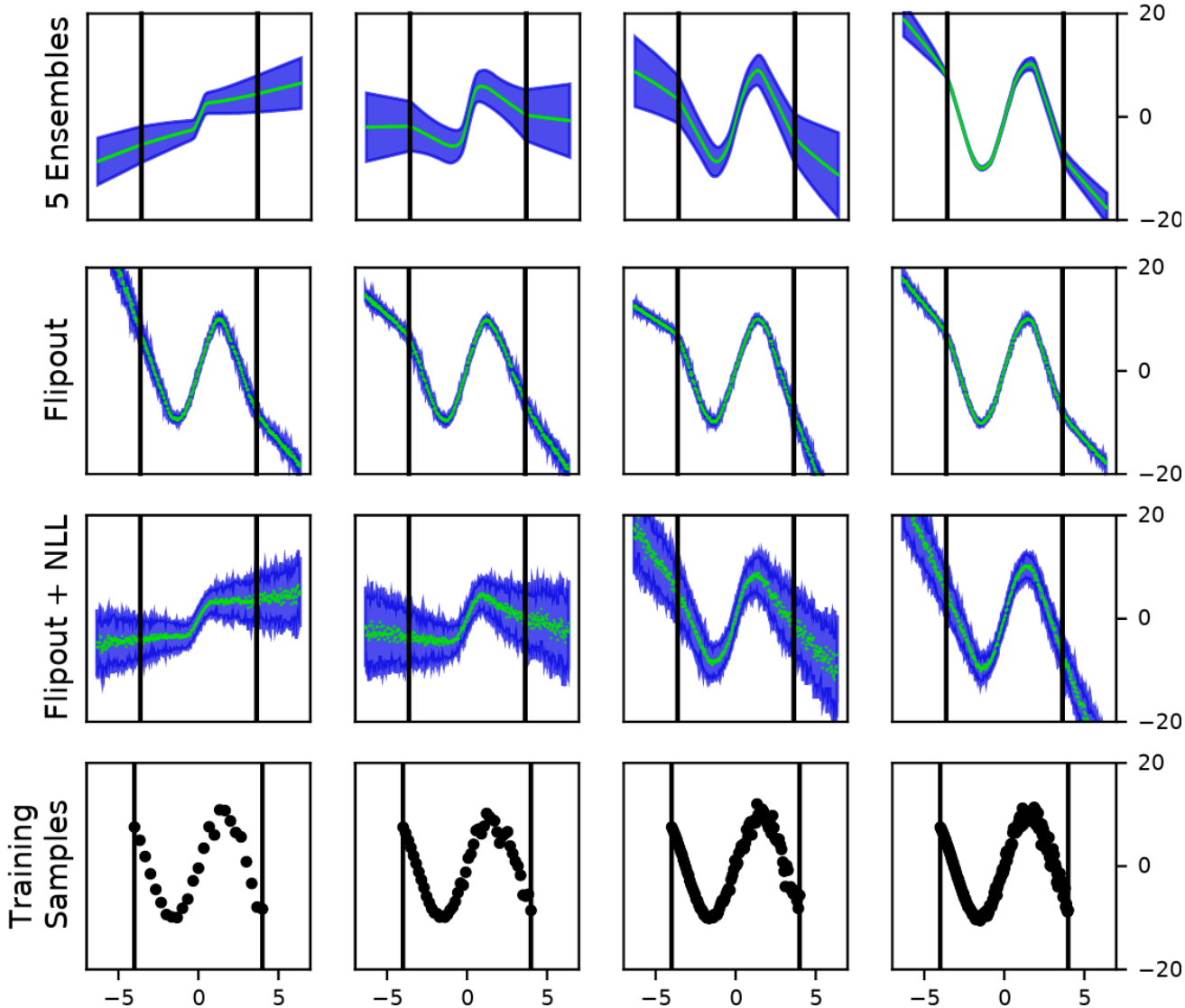
Two Moons Classification



Noisy Sinusoid Regression 1/2



Noisy Sinusoid Regression 2/2



Questions 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.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

Bibliography II

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

Aryan Mobiny, Pengyu Yuan, Supratik K Moulik, Naveen Garg, Carol C Wu, and Hien Van Nguyen. Dropconnect is effective in modeling uncertainty of bayesian deep networks. *Scientific reports*, 11(1):1–14, 2021.

Philipp Oberdiek, Matthias Rottmann, and Hanno Gottschalk. Classification uncertainty of deep neural networks based on gradient information. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 113–125. Springer, 2018.

Bibliography III

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.

Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. Uncertainty estimation using a single deep deterministic neural network. In *International conference on machine learning*, pages 9690–9700. PMLR, 2020.

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.