Understanding Uncertainty in Deep Learning through Bayesian Approaches

Nathaniel Cogneaux

February 21, 2024

Contents

1 Introduction to Bayesian Models

1.1 Introduction and Context

Since the early 2010s, the field of machine learning has witnessed a profound transformation, primarily driven by the resurgence and success of deep learning. Prior to this era, during the 1990s and 2000s, training large deep models on existing datasets posed significant challenges due to computational and data limitations. However, the "deep revolution" emerged in 2012, marking a pivotal moment with the outstanding success of Convolutional Neural Networks (ConvNets). These advancements simplified the analysis of data, leading to deep learning's ubiquity in various applications since 2012, ranging from image classification and speech recognition to chatbots, translation, gaming, and robotics, exemplified by achievements such as AlphaGo.

One of the key innovations of ConvNets lies in their architecture, which utilizes sparse connectivity and shared weights through convolutions with learned kernels. This approach allows for the sharing of the same parameters across different locations in the input data, facilitating local feature extraction. This innovation effectively addresses the parameter explosion problem associated with Fully Connected Networks (FCNs) when applied to images, where a 1000x1000 image would require an impractical number of parameters.

Following the success in vision tasks, the development of Recurrent Neural Networks (RNNs), including specific architectures like Long Short-Term Memory (LSTM) networks introduced in 1997 and Gated Recurrent Units (GRUs) in 2014, marked significant progress in sequence processing. RNNs have become the state-of-the-art for various sequential decision-making tasks, including speech recognition, translation, text and music generation, time series analysis, and video forecasting.

Despite these advancements, a critical challenge remains in deep learning: robustness. While deep learning models have achieved substantial gains in average performance metrics such as precision for classification tasks, their deployment in safety-critical applications necessitates performance certification. This certification involves a formal understanding of the models' generalization capabilities, optimization landscapes, and, importantly, the stability of their decision functions, especially in the face of adversarial attacks.

A crucial aspect of performance certification is the reliable estimation of confidence

or uncertainty in the models' decisions. Traditional deep learning models often exhibit overconfidence, which is inadequate for tasks requiring precise uncertainty estimation, such as healthcare and autonomous vehicle navigation. This paper aims to address these concerns by exploring Bayesian models and uncertainty estimation, starting with Bayesian linear regression and extending to non-linear feature maps.

We will then delve into Bayesian deep learning, focusing on approximate posterior inference and variational approximation techniques, as well as Monte Carlo dropout methods. The application of uncertainty estimation to enhance robustness, including failure prediction and out-of-distribution (OOD) detection, will also be discussed. This exploration seeks to provide a comprehensive understanding of how Bayesian approaches can contribute to the development of more robust and reliable deep learning models, particularly for deployment in critical applications.

1.2 Uncertainty

Understanding uncertainty in machine learning is crucial for developing robust models, especially in applications where decision-making is critical. Uncertainty can be broadly classified into two types: aleatoric and epistemic.

Aleatoric uncertainty, named after the Latin word for "dice player", *Aleator*, refers to the inherent randomness in the observations. This type of uncertainty captures the natural variability in the data, such as the stochastic nature of processes or inherent noise within the observations. For example, in medical imaging, ambiguity in the data might arise due to overlapping symptoms of different diseases, or the quality of the sensors used might introduce noise. Aleatoric uncertainty is an intrinsic property of the data that cannot be reduced by simply adding more data. Instead, addressing it may require improving the quality of the inputs or the sensors used to collect the data.

Within aleatoric uncertainty, we distinguish between homoscedastic and heteroscedastic uncertainties. Homoscedastic uncertainty remains constant across different inputs, representing a uniform level of noise throughout the data. In contrast, heteroscedastic uncertainty varies with the input, indicating that some areas of the data space may have more inherent noise than others. This variability can often be learned from the data, allowing models to adjust their confidence levels based on the input's characteristics.

On the other hand, epistemic uncertainty, derived from the Greek word *Episteme* for "knowledge", reflects uncertainty in the model itself. It is essentially our lack of knowledge about the best model to explain the observed data. This type of uncertainty is prominent in situations where the model encounters data that is significantly different from what it was trained on, highlighting the limits of the training distribution. The key feature of epistemic uncertainty is its reducibility; it can decrease as we gather more data, especially data that fills in the gaps in the model's knowledge base. Thus, epistemic uncertainty tends to diminish as the amount of data increases, theoretically approaching zero as the dataset size approaches infinity.

The distinction between aleatoric and epistemic uncertainty is fundamental for developing machine learning models that are not only accurate but also reliable in their predictions. By understanding and quantifying these uncertainties, we can better gauge the confidence of our models in their predictions, enabling safer applications in fields where precision and reliability are paramount.

1.3 Bayesian Models

In the Bayesian framework, we start by considering our observed dataset consisting of inputs $X = \{x_i\}_{i=1}^N$ and corresponding outputs $Y = \{y_i\}_{i=1}^N$, where each input x_i is a vector in $\mathbb{R}^d$ and each output y_i is a vector in $\mathbb{R}^K$. This setup can be used for tasks like classification or regression. We model the outputs Y using a function f_w parameterized by w, aiming to predict $\hat{y}_i$ from x_i.

Bayesian inference is grounded in Bayes' rule, which in this context allows us to update our belief about the model parameters w after observing data X and Y. The rule is expressed as:

$$\mathbb{P}(Y, w|X) = \mathbb{P}(Y|X, w)\mathbb{P}(w) = \mathbb{P}(w|X, Y)\mathbb{P}(Y|X)$$

So one get the following Bayes' formulae:

$$\mathbb{P}(w|X, Y) = \frac{\mathbb{P}(Y|X, w)\mathbb{P}(w)}{\mathbb{P}(Y|X)} \propto \mathbb{P}(Y|X, w)\mathbb{P}(w)$$

where $\mathbb{P}(Y|X, w)$ is the likelihood of observing Y given X and w, $\mathbb{P}(w)$ is the prior belief about w, and $\mathbb{P}(w|X, Y)$ is the posterior belief about w after observing X and Y.

In practice, what we model is the likelihood $\mathbb{P}(Y|X,w)$ and the prior $\mathbb{P}(w)$. What we compute is the posterior $\mathbb{P}(w|X,Y)$, which incorporates both the prior and the likelihood.

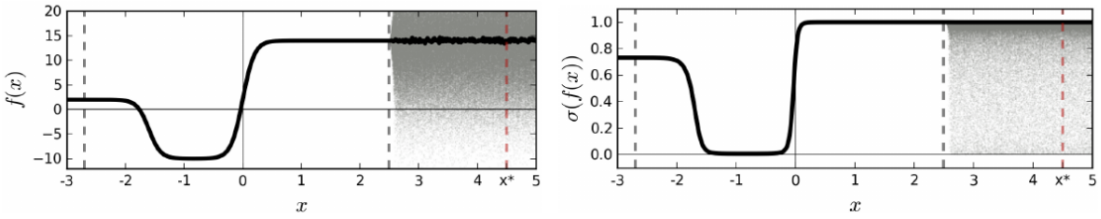
To find the optimal parameters w, we can use either Maximum Likelihood Estimation (MLE), which finds $\hat{w}$ such that $\mathbb{P}(Y|X,w)$ is maximized, often ignoring the prior, or Maximum A Posteriori (MAP) estimation, which uses the prior and finds $\hat{w}$ such that $\mathbb{P}(w|X,Y)$ is maximized.

After computing the posterior $\mathbb{P}(w|X,Y)$, we can predict new outcomes y for a new input x^* by computing the predictive distribution:
Given that $\forall y$, $\mathbb{P}(y|x^*,X,Y) = \int \mathbb{P}(y,w|x^*,X,Y)dw$, then

$$\mathbb{P}(y|x^*,X,Y) = \int \mathbb{P}(y|x^*,w)\mathbb{P}(w|X,Y)dw = \mathbb{E}_{\mathbb{P}(w|D)}[\mathbb{P}(y|x^*,w)]$$

This process inherently provides a measure of uncertainty in our predictions.

In the context of binary classification, consider training a neural network with a function $f^w(x)$ (such as the hyperbolic tangent, tanh) on a 1D dataset $X = [-3, 2]$. We then apply a Softmax function to obtain probabilities. A point estimate through Softmax may give us high confidence even for data points far from the observed data, potentially underestimating the uncertainty. On the other hand, the predictive distribution $\mathbb{P}(y|x^*,Y,X)$ can model epistemic uncertainty, which should intuitively increase for points far from the training data.



To summarize, estimating uncertainty with Bayesian models involves:

1. Defining the prior $\mathbb{P}(w)$ and the likelihood $\mathbb{P}(Y|X,w)$.
2. Computing the posterior distribution $\mathbb{P}(w|X,Y)$.
3. Computing the predictive distribution $\mathbb{P}(y|x^*,Y,X)$.

However, steps 2 and 3 are generally computationally challenging, often lacking a closed-form solution for the posterior and requiring high-dimensional integration for

the predictive distribution.

1.4 Probabilistic Linear Regression

Bayesian linear regression combines the traditional linear regression approach with a probabilistic model, using prior knowledge about parameters and observed data to refine predictions and quantify uncertainty. It models the input-output relationship through a linear framework, integrating randomness to address the inherent uncertainty in real-world data, offering a clearer, probabilistic interpretation of model predictions.

- Let us consider N training examples $(x_i, y_i)_{i=1}^N$, where each input x_i is a vector in $\mathbb{R}^p$ representing the input features, and each output y_i is a vector in $\mathbb{R}^K$ representing the target variables.

- The design matrix Φ of size $N \times (p+1)$, incorporating a bias term, is structured as follows:

$$\Phi = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Np} \end{pmatrix},$$

- The relationship between the inputs X and outputs Y is modeled linearly with added Gaussian noise $\epsilon \in \mathbb{R}^{N \times K}$, representing the error or noise in the observations:

$$Y = \Phi w + \epsilon,$$

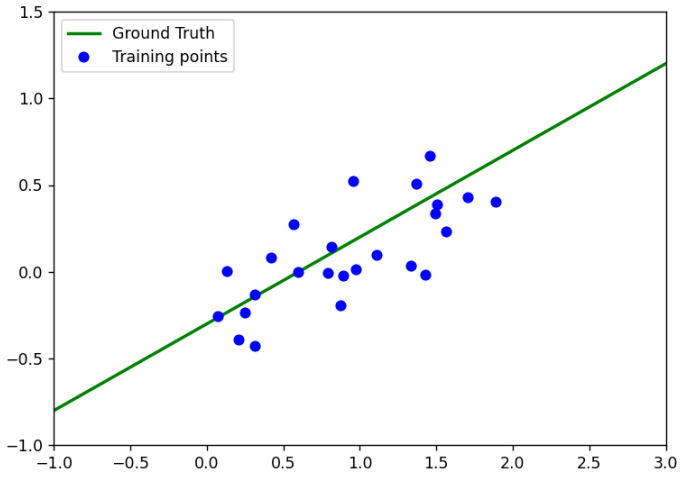
  With $Y \in \mathbb{R}^{N \times K}$, $\Phi \in \mathbb{R}^{N \times (p+1)}$, and where $w \in \mathbb{R}^{(p+1) \times K}$ represents the weight vector (including the bias), and ϵ is the noise term, with each $\epsilon_{i,k}$ following a normal distribution $\mathcal{N}(0, \sigma^2)$.

- For each training example, the model predicts the output y_i as a linear combination of the input features x_i, represented by $\Phi_i^T w$, plus noise ϵ_i:

$$y_i = \Phi_i^T w + \epsilon_i,$$

  where $\Phi_i^T = \begin{pmatrix} 1 & x_{i1} & \cdots & x_{ip} \end{pmatrix}^T \in \mathbb{R}^{1 \times (p+1)}$ is the transpose of the i-th row of Φ, converting it into a column vector suitable for the linear combination.

- The noise ϵ_i introduces uncertainty into our model, referred to as aleatoric uncertainty, which reflects the variability in the outputs Y that cannot be explained by the input features X alone.



The likelihood of observing a particular output y_i given an input x_i and the weight vector w is modeled as a Gaussian distribution centered at the predicted output $\Phi_i^T w$, with variance σ^2 representing the uncertainty of the prediction:

$$\mathbb{P}(y_i|x_i, w) \sim \mathcal{N}(\Phi_i^T w, \sigma^2) \quad (= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - \Phi_i^T w)^2}{2\sigma^2}})$$

Using this approach, we can calculate how likely our data is under a linear model, and apply Bayesian inference to update our understanding of the model's parameters, w, after seeing the data. Thus, we get updated distributions for w that better reflect our knowledge post-data. This method gives us a clearer insight into our data and model, aiding smarter decisions.
Note that, here, in our our likelihood $\mathbb{P}(y_i|x_i, w)$, σ represents aleatoric uncertainty and since it is independent of x it corresponds more precisely to homoscedastic uncertainty.

1.4.1 Using Maximum Likelihood Estimation

Now we want to estimate parameters via Maximum Likelihood Estimation (MLE), and incorporate prior knowledge.

- Training involves maximizing the likelihood of the observed data under the

assumption that examples are independent and identically distributed (i.i.d.), so:

$$\mathbb{P}(Y|X, w, \sigma) = \prod_{i=1}^{N} \mathbb{P}(y_i|x_i, w, \sigma)$$

- Then we seek parameters $(\hat{w}, \hat{\sigma})$ that maximize this likelihood, equivalent to minimizing the negative log-likelihood of the data, i.e.:

$$(\hat{w}, \hat{\sigma}) = \arg\max_{(w,\sigma)} \mathbb{P}(Y|X, w, \sigma) = \arg\min_{(w,\sigma)} -\sum_{i=1}^{N} \log[\mathbb{P}(y_i|x_i, w, \sigma)]$$

We get the log-likelihood:

$$\log L(w, \sigma) = \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - \Phi_i^T w)^2}{2\sigma^2}}\right) \tag{1}$$

Simplifying it, gives:

$$\log L(w, \sigma) = -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \Phi_i^T w)^2 \tag{2}$$

Since the first term does not depend on w, minimizing the negative log-likelihood with respect to w focuses on the second term. Thus, the MLE solution for w minimizes the following cost function:

$$\hat{w} = \arg\min_{w} C(w) = \arg\min_{w} \sum_{i=1}^{N}(y_i - \Phi_i^T w)^2$$

We recognize an ordinary least square problem (closed form):

$$\begin{aligned}
&C(w) = \|Y - \Phi w\|^2 = (Y - \Phi w)^T (Y - \Phi w) \\
&\nabla_w C = 2\Phi^T (Y - \Phi w) \\
&\nabla_w C = 0 \Leftrightarrow \Phi^T \Phi w = \Phi^T Y, \text{ thus we get the result:} \\
&\hat{w} = (\Phi^T \Phi)^{-1} \Phi^T Y
\end{aligned}$$

Then the optimal σ is found by minimizing the expression related to the variance

of errors:

$$\arg\min_{\sigma} N\log(\sigma) + \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \Phi_i^T w)^2$$

From which we get the closed form solution:

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \Phi_i^T \hat{w})^2$$

Which can be interpreted as the data standard deviation.

Remark: However, MLE admits limits. Here is an example to highlight it. Consider a practical application involving predicting the outcome of coin tosses. We treat the result of a coin toss as a Bernoulli variable, defining X with parameter p, where p is the probability of obtaining heads. The likelihood function for X is given by:

$$\mathbb{P}(X|p) = \prod_{i=1}^{N}\mathbb{P}(x_i|p) = \prod_{i=1}^{N} p^{x_i}(1-p)^{1-x_i} \tag{3}$$

The log-likelihood of observing a series of tosses is then modeled as:

$$\ln\mathbb{P}(X|p) = \sum_{i=1}^{N}[x_i \ln p + (1-x_i)\ln(1-p)] \tag{4}$$

Maximizing it with respect to p yields the MLE estimate for p:

$$p_{\text{MLE}} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{5}$$

This MLE provides an estimate for the probability of heads. However, if all observed tosses are heads ($x_i = 1$ for all i), the MLE suggests that:

$$\mathbb{P}(X|p_{\text{MLE}}) = 1 \tag{6}$$

This result misleadingly predicts heads for all future tosses, ignoring the inherent variability and unpredictability of real-world coin tosses.

To address this limitation, we can incorporate prior beliefs about the probability of heads p into a Bayesian framework, allowing for an update to the model that adjusts predictions based on both prior knowledge and observed data. For example, using

a prior knowledge that the coin is fair, we might choose a prior distribution for p, such as a Beta distribution centered around 0.5, to reflect our belief that the coin has an equal chance of landing heads or tails.

The Maximum A Posteriori (MAP) estimate then adjusts the MLE by incorporating this prior:

$$p_{\text{MAP}} = \frac{\sum_{i=1}^{N} x_i + \alpha - 1}{N + \alpha + \beta - 2} \tag{7}$$

where α and β are the parameters of the Beta prior distribution, reflecting our prior knowledge about p. This adjustment ensures that our predictions are not solely based on the observed data but are moderated by our prior beliefs, leading to more robust and realistic estimates, especially in the face of limited or biased data samples.

1.4.2 Using Maximum A Posteriori

Henceforth, going back to Bayesian Linear Regression, we now decide to incorporate the prior distribution for the weights w using Maximum A Posteriori (MAP) technique. We choose a Gaussian representation $\mathbb{P}(w|\alpha) \sim \mathcal{N}(w; 0, \alpha^{-1}I)$, indicating a belief in w being centered around zero with variance governed by α^{-1}. In fact here, we are insisting that before we even see the data, small values of w are more likely, so if we choose some data now, which suggests we should choose more extreme values of w that is going to be counteracted by this prior understanding. In other words, it would take some extreme evidence in the data for us to accept high values of w because of this prior distribution.

Then the likelihood of observing each y_i given x_i and w, is as before, modeled as a Gaussian distribution: $\mathbb{P}(y_i|x_i, w) = \mathcal{N}(\Phi_i^T w, \beta^{-1})$, where $\beta = \frac{1}{2\sigma^2}$ reflects the precision (inverse of variance) of the noise in the observations.

Thus, we can now compute the posterior, whose distribution satisfies:

$$\mathbb{P}(w|x_i, y_i) \propto \mathbb{P}(y_i|x_i, w)\mathbb{P}(w)$$

We won't go through the details of the computations but given the Gaussian nature of both the prior and the likelihood, the posterior is also Gaussian and we find as a

closed form solution of the MAP, the mean μ and covariance Σ as follows:

$$\mathbb{P}(w|X,Y) = \mathcal{N}(w|\mu,\Sigma)$$

$$\mu = \beta\Sigma\Phi^T Y, \quad \Sigma^{-1} = \alpha I + \beta\Phi^T\Phi$$

reflecting how the posterior distribution adjusts based on the data and the initial assumptions. The prior knowledge on w is introduced through a Gaussian prior, $\mathbb{P}(w|\alpha) = \mathcal{N}(w;0,\alpha^{-1}I)$ to facilitate the computation of the posterior $\mathbb{P}(w|X,Y)$.

Remark: The closed-form solution for the MAP estimates provides a direct way to compute the most probable values of w given the data. It is converging to the MLE estimates as $\alpha \to 0$ or in the absence of prior data ($N = 0$).

Remark: Here is a general result in Bayesian inference, we consider the Gaussian distributions for x and y, assuming:

- $\mathbb{P}(x) = \mathcal{N}(x|\mu_x,\Sigma_x)$
- $\mathbb{P}(y|x) = \mathcal{N}(y|Ax+b,\Sigma_y)$

Then the posterior distribution $\mathbb{P}(x|y)$ is derived as:

$$\mathbb{P}(x|y) = \mathcal{N}(x|\mu_{x|y},\Sigma_{x|y})$$

With:

$$\Sigma_{x|y}^{-1} = \Sigma_x^{-1} + A^T\Sigma_y^{-1}A$$
$$\mu_{x|y} = \Sigma_{x|y}[A^T\Sigma_y^{-1}(y-b) + \Sigma_x^{-1}\mu_x]$$

Illustrating how the model's accuracy improves as more observations become available. This approach not only estimates parameters but also quantifies uncertainty, offering a comprehensive view of the model's predictive capabilities.

So, the posterior distribution $\mathbb{P}(w|X,Y)$ is given by a Gaussian distribution:

$$\mathbb{P}(w|X,Y) = \mathcal{N}(w;\mu,\Sigma).$$

This Gaussian representation of the posterior provides a comprehensive framework

for understanding how our beliefs about the parameters change with the observation of new data. The process unfolds as follows:

- **No observation:** $\mathbb{P}(w|X,Y) = \mathbb{P}(w)$
- **First Observation:** Upon observing the first data point (x_0, y_0), we update our prior belief to reflect this new information through a Bayesian update:

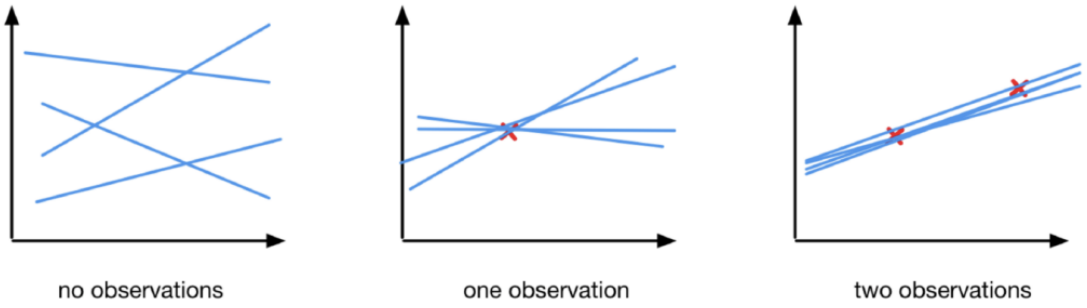
$$\mathbb{P}(w|x_0, y_0) \propto \mathbb{P}(w)\mathbb{P}(y_0|x_0, w),$$

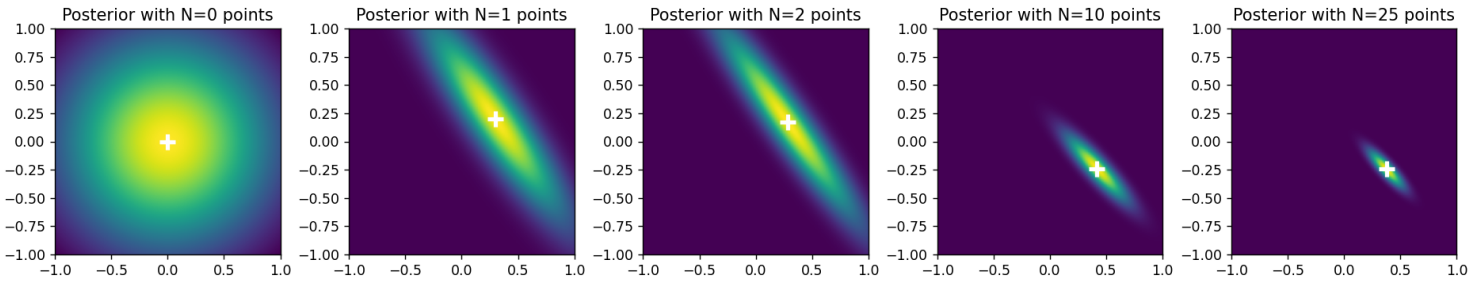
  where $\mathbb{P}(y_0|x_0, w)$ is the likelihood of observing y_0 given x_0 and parameter w, often assumed to be Gaussian as well.

- **Subsequent Observations:** Each new observation (x_i, y_i) further refines our posterior belief about w, integrating the information from all previous observations and the prior:

$$\mathbb{P}(w|(x_0, y_0), \ldots, (x_i, y_i)) \propto \mathbb{P}(w) \prod_{j=0}^{i} \mathbb{P}(y_j|x_j, w).$$

  This formula demonstrates the cumulative nature of Bayesian updating, where each new data point contributes to an increasingly accurate and nuanced understanding of w.





- Posterior with N=0 points: This is the prior distribution over the parameters before observing any data. Since no data has been observed, the distribution is centered around the prior mean, which is typically set to 0 for both parameters. It shows high uncertainty in all directions, indicated by the wide and uniform spread of the color gradient.

- Posterior with N=1 point: After observing one data point, the posterior distribution starts to update. The distribution narrows down around the parameter values that are more likely given the observed data. There is still considerable uncertainty, but less than before, and it's now centered around the values suggested by the single data point.

- Posterior with N=2 points: With two data points, the posterior continues to narrow as the model becomes more certain about the parameters. The distribution is elongated along a line, indicating that there is a linear relationship between the parameters that is consistent with the two observed data points.

- Posterior with N=10 points: As more data points are observed, the posterior distribution becomes more concentrated. This indicates increased confidence in the parameter estimates. The peak of the distribution is now sharper, and the area of high probability density is more localized, reflecting greater certainty in the model parameters.

- Posterior with N=25 points: With a larger number of data points, the posterior distribution is much more peaked and narrow. The model has a high level of certainty about the parameter values, and the high probability density region is small, indicating that the parameters are well estimated.

Key Points:

- Before any data are observed, the posterior distribution $\mathbb{P}(w|X, Y)$ is identical to the prior distribution, indicating our initial beliefs about w.

- With the observation of data ($N \geq 1$), the prior is incrementally biased by the data likelihood, resulting in a posterior that incorporates both the evidence from the data and our prior beliefs.

- The Gaussian form of the posterior $\mathbb{P}(w|X, Y) = \mathcal{N}(w; \mu, \Sigma)$ allows for straightforward sampling from the posterior, facilitating predictions and uncertainty quantification.

- This iterative Bayesian update mechanism highlights how data sequentially inform and refine our model parameters, reducing uncertainty and converging on a more precise estimate of w.

Each step signifies a Bayesian update, where prior beliefs are combined with the likelihood of the observed data to form a posterior distribution, which then serves as the new prior for the next observation. So, more data points allows reducing the posterior (epistemic) uncertainty.

Now, let's compute the predictive distribution. Given the posterior $\mathbb{P}(w|\mathcal{D}, \alpha, \beta)$, we compute the predictive distribution by marginalizing over the uncertainties in w.

- The predictive distribution for a new observation y given x^*, data $\mathcal{D} = (x_i, y_i)_{i=1}^N$, and hyperparameters α and β is computed as:

$$\mathbb{P}(y|x^*, \mathcal{D}, \alpha, \beta) = \int \mathbb{P}(y|x^*, w, \beta)\mathbb{P}(w|\mathcal{D}, \alpha, \beta)dw.$$

  (This integral captures the essence of Bayesian prediction, averaging over all possible parameter values weighted by their posterior probabilities.)

- The likelihood term: $\mathbb{P}(y|x^*, w, \beta) \sim \mathcal{N}(y; (\phi(x^*))^T w, \beta^{-1})$
- The posterior: $\mathbb{P}(w|\mathcal{D}, \alpha, \beta) \sim \mathcal{N}(w; \mu, \Sigma)$, with:
  - $\Sigma^{-1} = \alpha I + \beta \Phi^T \Phi$
  - $\mu = \beta \Sigma \Phi^T Y$
- Then, the resulting predictive distribution is Gaussian since the convolution of two Gaussians is Gaussian, with mean and variance that incorporate both the model predictions and the uncertainty about w:

$$\mathbb{P}(y|x^*, \mathcal{D}, \alpha, \beta) = \mathcal{N}(y; \mu_{pred}(x^*), \sigma^2_{pred}(x^*)).$$

  The mean $\mu_{pred}(x^*)$ represents the expected value of y for the new input x^*, and the variance $\sigma^2_{pred}(x^*)$ quantifies the prediction's uncertainty, accounting for both aleatoric and epistemic sources of uncertainty.

  - Mean of predictive distribution $\mu_{pred}(x^*) = \mu^T \phi(x^*)$
  - Variance of predictive distribution $\sigma^2_{pred}(x^*) = \frac{1}{\beta} + \phi(x^*)^T \Sigma \phi(x^*)$

Here β represents the aleatoric uncertainty and $\phi(x^*)^T\Sigma\phi(x^*)$ the uncertainty over parameters w (epistemic).

Remark: $\sigma^2_{pred}(x^*)$ actually depends on N, it should be written $\sigma^2_{pred}(x^*, N)$

- $\sigma^2_{pred}(x^*, N+1) < \sigma^2_{pred}(x^*, N)$
- As $N \to \infty$, $\phi(x^*)^T\Sigma\phi(x^*) \to 0$: the epistemic uncertainty is removed by adding data samples.

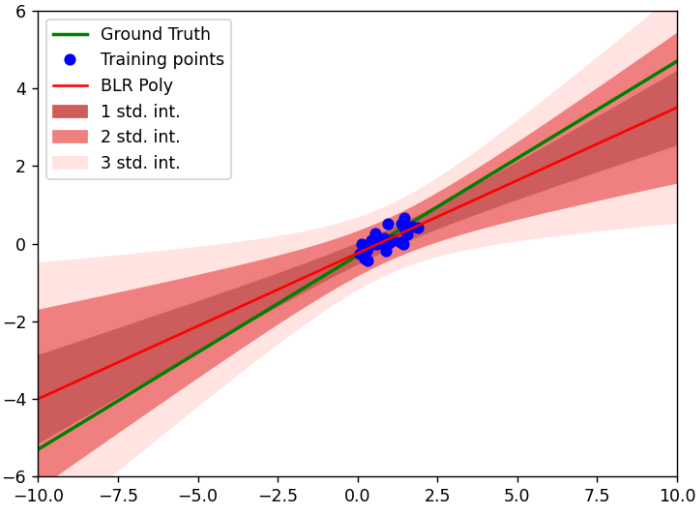
As an example, if we consider 1D inputs and outputs, where $x_i \in \mathbb{R}$, $X \in \mathbb{R}^{N\times 1}$, $w \in \mathbb{R}^2$, and $\phi \in \mathbb{R}^{N\times 2}$, the covariance matrix Σ is given by:

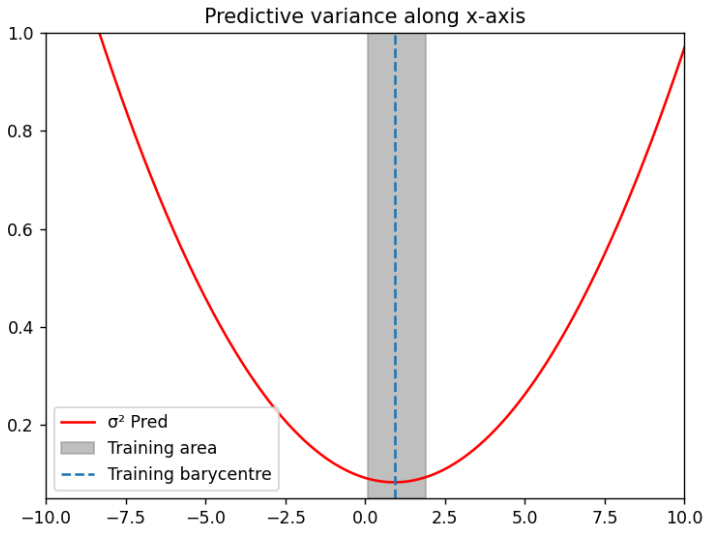
$$\Sigma^{-1} = \alpha I + \beta\Phi^T\Phi = \begin{pmatrix} \alpha + \beta N & \beta \mathbf{1}^T X \\ \beta \mathbf{1}^T X & \alpha + \beta X^T X \end{pmatrix} \tag{8}$$

This leads to an increase in $\phi(x^*)^T\Sigma\phi(x^*)$ when x^* is far from the training data, with the minimum epistemic uncertainty at:

$$x_{\min} = \frac{\sum x_i}{N + \alpha/\beta} \tag{9}$$

At the end, we get this kind of results:





Note on MAP estimate:

- The MAP estimate w_{MAP} is given by:

$$\begin{aligned} w_{\text{MAP}} &= \arg\min_w -\sum_{n=1}^{N} \log \mathbb{P}(w|x_n, y_n, \beta, \alpha) \\ &= \arg\min_w -\sum_{n=1}^{N} \log \mathbb{P}(y_n|x_n, w, \beta, \alpha) - \log \mathbb{P}(w|\alpha) \\ &= \arg\min_w \frac{\beta}{2} \sum_{i=1}^{N} \|y_n - f^w(x_n)\|^2 + \frac{\alpha}{2} w^T w \end{aligned}$$

  This implies that adding a Gaussian prior with precision on weights α acts like L_2 regularization (weight decay) with $\lambda = \alpha/\beta$. (Ridge)

- Another interesting result is that if we picked a Laplacian prior instead of a Gaussian we would have found a minimization problem of the form:

$$w_{\text{MAP}} = \arg\min_w \frac{1}{2} \sum_{i=1}^{N} \|y_n - f^w(x_n)\|^2 + \lambda \|w\|_1$$

  Which acts like a L_1 regularization (Lasso)

- The prediction distribution $\neq$ likelihood at $w = w_{\text{MAP}}$: the likelihood at $w = w_{\text{MAP}}$ is given by:

$$\mathbb{P}(y|x^*, w_{\text{MAP}}) = \mathcal{N}(y; \mu^T \phi(x^*), \sigma^2)$$

  where σ^2 is constant for all x^*
  Whereas for the prediction distribution:

$$\begin{aligned} \mathbb{P}(y|x^*, \mathcal{D}, \alpha, \beta) &= \int \mathbb{P}(y|x^*, w, \beta) \mathbb{P}(w|\mathcal{D}, \alpha, \beta)\, dw && (10) \\ &= \mathcal{N}(y; \mu^T \phi(x^*), \sigma^2_{pred}(x^*)) && (11) \\ \sigma^2_{pred}(x^*) &= \beta^{-1} + \phi(x^*)^T \Sigma \phi(x^*) && (12) \end{aligned}$$

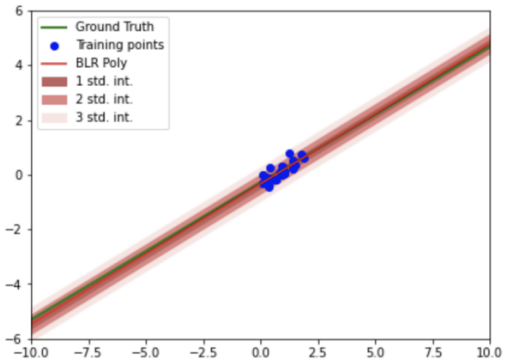


Figure 1: likelihood at $w = w_{MAP}$: $\mathbb{P}(y|x^*, w_{MAP})$

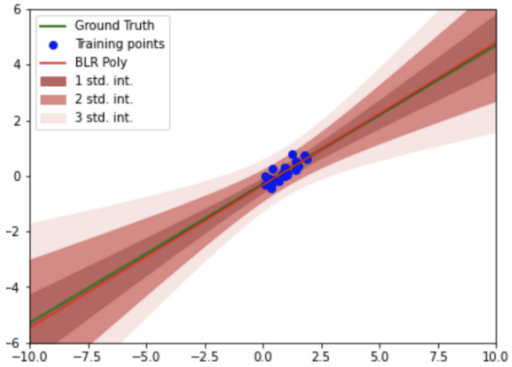


Figure 2: prediction distribution: $\mathbb{P}(y|x^*, D, \alpha, \beta)$

1.4.3 Non-linear extension:

Linear regression is limited in many datasets due to its inability to model complex relationships. A non-linear extension can be achieved by designing explicit non-linear feature maps Φ. The model can then be described as $Y = \Phi w + \epsilon$, where $\epsilon = (\epsilon_1 \ \cdots \ \epsilon_N)^T$ and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. For example, take a polynomial regression for 1D inputs, i.e., $x_i \in \mathbb{R}$, it is defined as:

$$\Phi = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^M \\ 1 & x_2 & x_2^2 & \cdots & x_2^M \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^M \end{pmatrix}$$

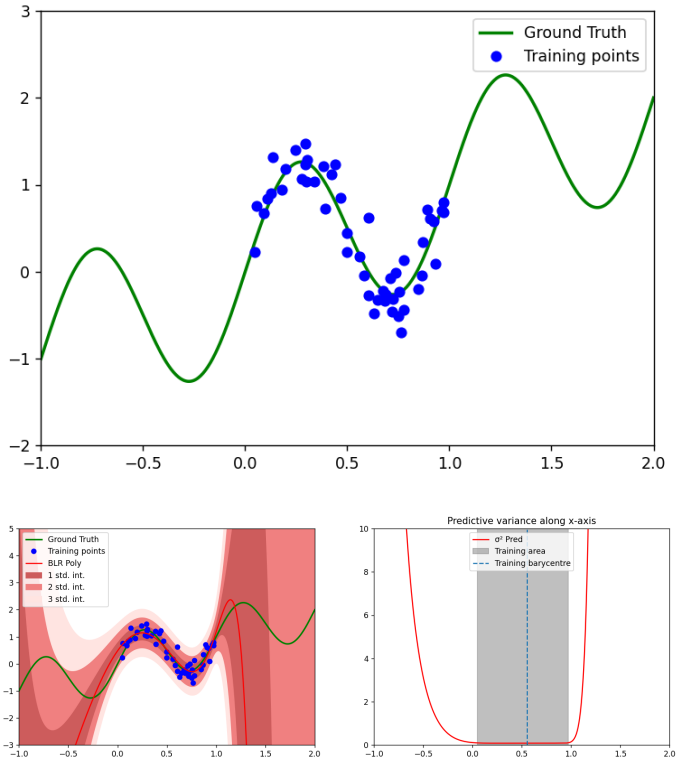
To apply Bayesian linear regression in non-linear feature space Φ, we use the same closed-form solution for posterior and predictive distribution in Φ.

With a dataset generated by a sinus:

We get the following result:

Remark: The predictive variance increase as we move away from training data. However here with polynomial features, the minimum is not the training barycenter anymore.

We can also look for Gaussian regression for 1D inputs, i.e., $x_i \in \mathbb{R}$, and the feature



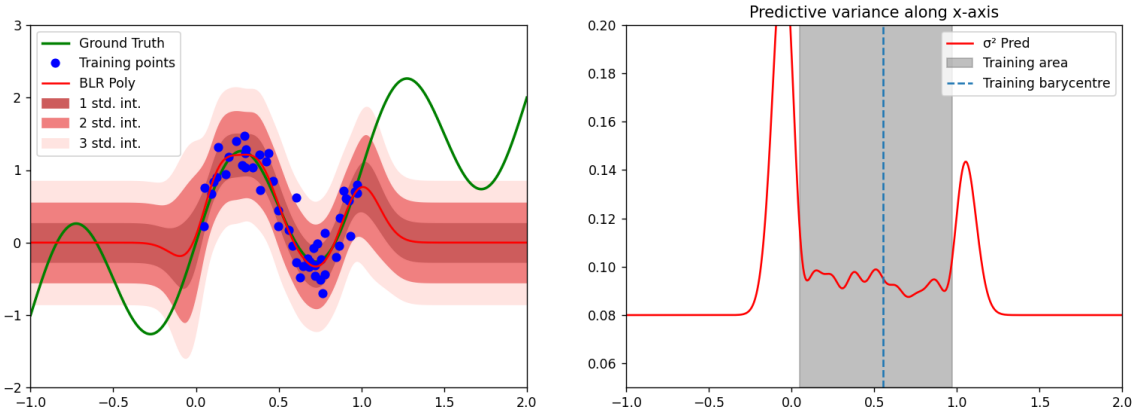
mapping $\Phi_j(x_i)$ is given by the Gaussian kernel:

$$\Phi_j(x_i) = \exp\left(-\frac{(x_i - \mu_j)^2}{2s^2}\right)$$

The design matrix Φ is then constructed as:

$$\Phi = \begin{pmatrix} \Phi_1(x_1) & \Phi_2(x_1) & \cdots & \Phi_M(x_1) \\ \Phi_1(x_2) & \Phi_2(x_2) & \cdots & \Phi_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_1(x_N) & \Phi_2(x_N) & \cdots & \Phi_M(x_N) \end{pmatrix}$$

Then we apply the standard Bayesian linear regression in the non-linear feature space Φ using the same closed-form solution for posterior and predictive distribution in Φ.



The predictive variance in the plot with Gaussian basis functions exhibits some interesting behavior:

- **Low Variance in Training Range:** Within the range of the training data, the predictive variance is relatively low. This indicates that the model is confident in its predictions where it has seen data, which is expected because the Gaussian basis functions will have significant activation and provide informative features about the target function in these regions.

- **Peaks at the Boundaries:** There are noticeable peaks in predictive variance at the boundaries of the training data range. This happens because the Gaussian basis functions are localized; they have the highest impact around their mean μ_j and their influence decreases as the input moves away from the mean. Since the model is trained on data that falls within the range of these basis functions, it becomes less confident about its predictions as the input moves away from any of the means μ_j, hence the increased variance at the boundaries.

- **Convergence to a Specific Value:** In regions far from the training distribution, the predictive variance converges to a specific value, which is equal to the prior variance $\frac{1}{\alpha}$ in the case of infinite width Gaussian basis functions. This convergence occurs because the Gaussian basis functions essentially have no activation (they are almost zero), and thus, the model's predictions revert to the prior, which is uninformed by the data. In other words, as the distance from the training data increases, the features provided by the Gaussian basis functions become increasingly irrelevant, and the model's predictions rely solely on the prior distribution over the weights.

This behavior underscores the importance of choosing the right basis functions and

their parameters. For localized basis functions like Gaussians, the placement of the means μ_j and the scale s can drastically affect the model's ability to generalize and its confidence in predictions made far from the training data. It also highlights a characteristic of Bayesian models: the predictions revert to the prior in the absence of data. This is a desirable property, as it reflects the uncertainty inherent in making predictions in regions without data.

2 Bayesian Neural Networks

2.1 Beyond Bayesian Linear Regression

Bayesian methods allow us to quantify uncertainty in model parameters through the posterior distribution and make predictions by integrating over this distribution. However, the complexity of models often precludes closed-form solutions for these distributions as opposed to the Bayesian Linear Regression.

We recall that the posterior distribution for parameters $\mathbf{w}$ given data $\mathbf{X}, \mathbf{Y}$ is given by:

$$\mathbb{P}(\mathbf{w}|\mathbf{X}, \mathbf{Y}) \propto \mathbb{P}(\mathbf{Y}|\mathbf{X}, \mathbf{w})\mathbb{P}(\mathbf{w}) \tag{13}$$

where $\mathbb{P}(\mathbf{w})$ is the prior and $\mathbb{P}(\mathbf{Y}|\mathbf{X}, \mathbf{w})$ is the likelihood.

The predictive distribution for a new data point $\mathbf{x}^*$, given data $\mathcal{D}$, is:

$$\mathbb{P}(y^*|\mathbf{x}^*, \mathcal{D}) = \int \mathbb{P}(y^*|\mathbf{x}^*, \mathbf{w})\mathbb{P}(\mathbf{w}|\mathcal{D})d\mathbf{w} \tag{14}$$

Examples of models without closed-form solutions:

- **Bayesian Logistic Regression**: A simple linear classification model without a Gaussian likelihood.
- **Neural Networks:** Even a single hidden layer precludes closed-form solutions for both regression and classification. Deep neural networks further complicate Bayesian inference due to their depth and non-linearity.

Without analytical expressions for $\mathbb{P}(\mathbf{w}|\mathcal{D})$ and $\mathbb{P}(y^*|\mathbf{x}^*, \mathcal{D})$, we resort to approximation methods:

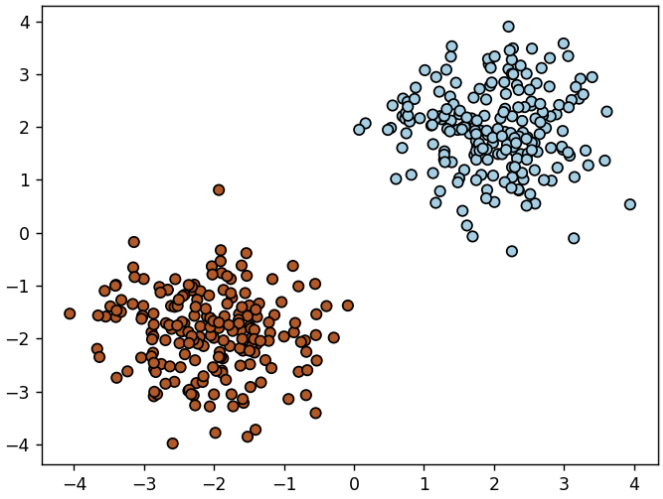
- **Gaussian Approximations:** Utilizing methods like the Laplace approximation for $\mathbb{P}(\mathbf{w}|\mathcal{D})$.
- **Monte Carlo Methods:** Sampling techniques, such as Metropolis-Hastings and Hamiltonian Monte Carlo, to evaluate the integral in the predictive distribution.

- **Variational Inference:** An optimization approach that minimizes the Kullback-Leibler (KL) divergence between $\mathbb{P}(\mathbf{w}|\mathcal{D})$ and a proposed parametric function.

These methods are crucial for performing Bayesian inference in complex models where traditional methods fail.

2.2 Bayesian Logistic Regression

In the context of Bayesian Logistic Regression (BLR), we are dealing with the classification problem from a probabilistic standpoint. The goal is to compute the posterior distribution over weights $\mathbf{w}$, which is not analytically tractable due to the non-Gaussian nature of the likelihood in logistic regression.



The relationship between the inputs $\mathbf{x}_i$ and the binary outcomes y_i is modeled as:

- $s_i = \mathbf{W}\mathbf{x}_i$ where s_i is the logit,
- The likelihood for the binary case is expressed with the sigmoid function σ:

$$\mathbb{P}(y_i = 1|\mathbf{x}_i, \mathbf{w}) = \sigma(s_i)$$

$$\mathbb{P}(y_i = -1|\mathbf{x}_i, \mathbf{w}) = 1 - \sigma(s_i)$$

- For multi-class classification, the softmax function is used to model the probabilities.

$$\mathbb{P}(y_i|x_i, w) = \hat{y}_i$$

$$\hat{y}_{i,k} = \frac{\exp(s_i)}{\sum_k \exp(s_k)}$$

In this context, the logistic regression exhibits a non-Gaussian distribution for $\mathbb{P}(\mathbf{Y}|\mathbf{X}, \mathbf{w})$. In fact, more generally this logistic likelihood does not lead to a conjugate prior, which would result in a posterior distribution that is easy to compute analytically. The preferred approach is to employ Maximum a Posteriori (MAP) estimation once again to ascertain the weights that are most likely, given the observed data.

We recall that the MAP estimate $\mathbf{w}_{\text{MAP}}$ is given by:

$$\mathbf{w}_{\text{MAP}} = \arg\max_{\mathbf{w}} \mathbb{P}(\mathbf{X}, \mathbf{Y}|\mathbf{w})\mathbb{P}(\mathbf{w}) = \arg\max_{\mathbf{w}} \prod_{n=1}^{N} \mathbb{P}(y_n|\mathbf{x}_n, \mathbf{w})\mathbb{P}(\mathbf{w}) \tag{15}$$

This can be reformulated as a minimization problem:

$$\mathbf{w}_{\text{MAP}} = \arg\min_{\mathbf{w}} -\sum_{n=1}^{N} \log(\mathbb{P}(y_n|\mathbf{x}_n, \mathbf{w})) - \log(\mathbb{P}(\mathbf{w})) \tag{16}$$

Assuming a Gaussian prior over the weights:

$$\mathbb{P}(\mathbf{w}) = \mathcal{N}(\mathbf{w}; 0, \sigma_0^2\mathbf{I}) \tag{17}$$

The objective function for MAP estimation with binary prediction is:

$$\mathbf{w}_{\text{MAP}} = \arg\min_{\mathbf{w}} \sum_{n=1}^{N} \big(-y_n \log(\sigma(\mathbf{w}^\top \mathbf{x}_n + b)) - (1 - y_n) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_n + b))\big) + \frac{1}{2\sigma_0^2}||\mathbf{w}||_2^2 \tag{18}$$

where σ is the sigmoid function, and b is the bias term. This objective includes a regularization term corresponding to the Gaussian prior, which is equivalent to weight decay in the optimization literature.

One can think of solving this optimization problem via Gradient Descent to get the MAP estimate of the weights $\mathbf{w}_{\text{MAP}}$.

Then, the predictive distribution for a new data point $\mathbf{x}^*$ is given by:

$$\mathbb{P}(y = 1|\mathbf{x}^*, \mathcal{D}) = \int \mathbb{P}(y = 1|\mathbf{x}, \mathbf{w})\mathbb{P}(\mathbf{w}|\mathcal{D})d\mathbf{w} \tag{19}$$

However, the full posterior distribution $\mathbb{P}(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ is still intractable, so we approximate the predictive distribution by evaluating $\mathbb{P}(y = 1|\mathbf{x}^*, \mathcal{D}) \approx \mathbb{P}(y = 1|\mathbf{x}^*, \mathbf{w}_{\text{MAP}})$ using the MAP estimate.

Posterior Approximation

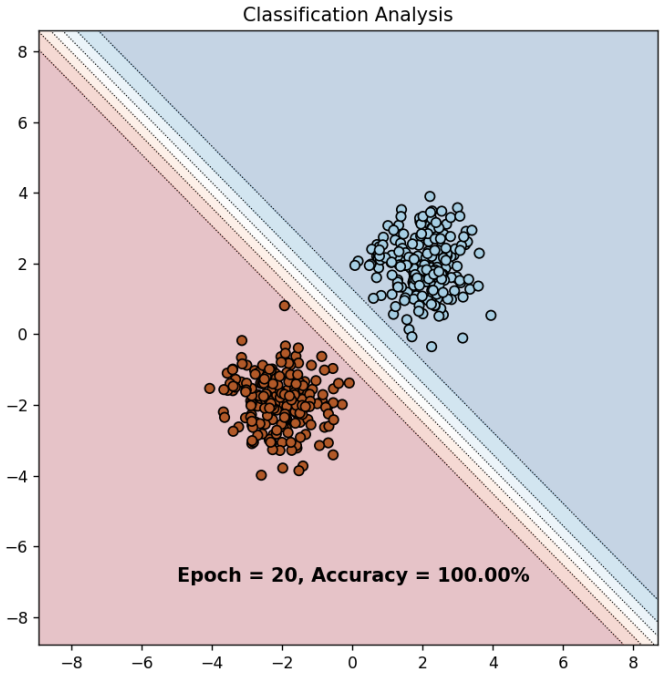
One way to approximate the posterior distribution $\mathbb{P}(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ is by a delta function centered at $\mathbf{w}_{\text{MAP}}$, which is of course a very coarse approximation:

$$\mathbb{P}(\mathbf{w}|\mathbf{X}, \mathbf{Y}) \approx \delta(\mathbf{w} - \mathbf{w}_{\text{MAP}}) \tag{20}$$

Simplifying the predictive distribution to:

$$\mathbb{P}(y = 1|\mathbf{x}^*, \mathcal{D}) \approx \mathbb{P}(y = 1|\mathbf{x}^*, \mathbf{w}_{\text{MAP}}) \tag{21}$$

The delta approximation implies that the uncertainty in our model's predictions does not increase as we move away from the training data. This is not ideal because it does not reflect the true variability of the model's predictions in regions of the input space where we have less information. This calls for the use of more sophisticated techniques, such as variational inference or sampling methods, to better approximate the true posterior distribution. This method overall gives result like this:



Another known method to deal with this intractability issue is the Laplace Approximation for $\mathbb{P}(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ which consists in approximating it by a normal distribution $q(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mu, \Sigma)$. Now we have to fit the mean and covariance accordingly.

- The mean μ of the approximating normal distribution $\mathcal{N}(\mathbf{w}; \mu, \Sigma)$ is fitted to the mode of the true posterior distribution. This is achieved by setting the gradient of the posterior to zero, which corresponds to the MAP estimate:

$$\mu = \mathbf{w}_{\text{MAP}}$$

- The inverse covariance matrix Σ^{-1} of $q(\mathbf{w})$ is fitted to the Hessian of the posterior at $\mu = \mathbf{w}_{\text{MAP}}$, providing a second-order approximation around the mode:

$$\Sigma^{-1} = \nabla\nabla_{\mathbf{w}}\mathbb{P}(\mathbf{w}|\mathbf{X}, \mathbf{Y})|_{\mathbf{w}=\mathbf{w}_{\text{MAP}}}$$

While this approach, known as the Laplace approximation, can provide a locally accurate estimate of the posterior, it has limitations:

- It approximates the posterior distribution at a single value, $\mathbf{w}_{\text{MAP}}$, and does not account for global properties of the distribution.
- It may not accurately reflect regions of high uncertainty, especially in the tails of the distribution.

Visualizing the Approximation:

So, doing a quick recap here, we have seen that for a given dataset $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\}$, the predictive distribution is given by:

$$\mathbb{P}(y^*|\mathbf{x}^*, \mathcal{D}) = \int \mathbb{P}(y^*|\mathbf{x}^*, \mathbf{w})\mathbb{P}(\mathbf{w}|\mathcal{D})d\mathbf{w} \tag{22}$$

And when approximating the posterior by a normal distribution $\mathcal{N}(\mathbf{w}; \mu, \Sigma)$:

$$\mathbb{P}(\mathbf{w}|\mathcal{D}) \approx q(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mu, \Sigma) \tag{23}$$

We get:

$$\mathbb{P}(y^*|\mathbf{x}^*, \mathcal{D}) \approx \int \mathbb{P}(y^*|\mathbf{x}^*, \mathbf{w})q(\mathbf{w})d\mathbf{w} \tag{24}$$

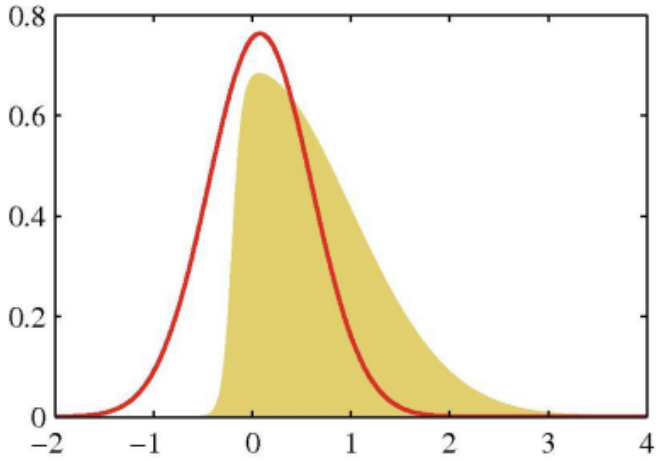


Figure 3: Comparison of the true posterior and the Laplace approximation. The normal distribution (red curve) centered at $\mathbf{w}_{\text{MAP}}$ with a covariance matrix defined by the Hessian provides a local approximation to the true posterior (yellow area).

But despite this approximation, the likelihood function is still not Gaussian, which results in an intractable posterior distribution $\mathbb{P}(y^*|\mathbf{x}^*, \mathcal{D})$. This necessitates the use of computational methods to approximate the integral and for that we can use the Monte Carlo (MC) Sampling method.

We estimate $\mathbb{P}(y^* = 1|\mathbf{x}^*, \mathcal{D})$ by drawing samples $\mathbf{w}^s$ from the approximate posterior $\mathcal{N}(\mathbf{w}; \mu, \Sigma)$. For the binary case, the predictive probability is approximated by:

$$\mathbb{P}(y^* = 1|\mathbf{x}^*, \mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S} \sigma((\mathbf{w}^s)^\top \mathbf{x}^*) \quad \mathbf{w}^s \sim q(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mu, \Sigma) \tag{25}$$

where σ denotes the sigmoid function. Note that sampling from the Gaussian $\mathcal{N}(\mathbf{w}; \mu, \Sigma)$ is straightforward and allows for an empirical approximation of the predictive distribution.

We can compare the MAP solution for Logistic Regression (LR) against Bayesian Logistic Regression (BLR) using both the Laplace approximation and Monte Carlo sampling methods to illustrate the differences in the approximations and the impact on classification boundaries:

Another option when dealing with binary classification to approximate the predictive distribution is by using the probit approximation. Indeed, one can approximate the sigmoid function $\sigma(\mathbf{w}^T x^*)$ by the probit function Φ as follows:

$$\sigma(a) \approx \Phi(\lambda a) \quad \text{with} \quad \lambda^2 = \frac{\pi}{8} \tag{26}$$

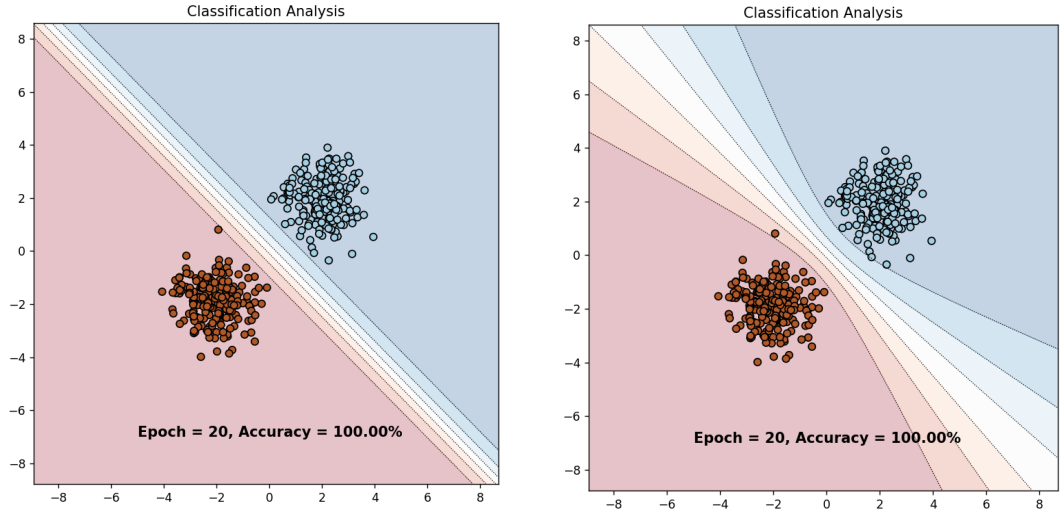


Figure 4: Comparison of classification boundaries obtained by MAP solutions for LR and BLR. The left image shows the result for LR, and the right image for BLR. Epoch and accuracy are noted to compare the performance.

Then, since the convolution of the probit function with a Gaussian gives another probit function, it leads to the approximation:

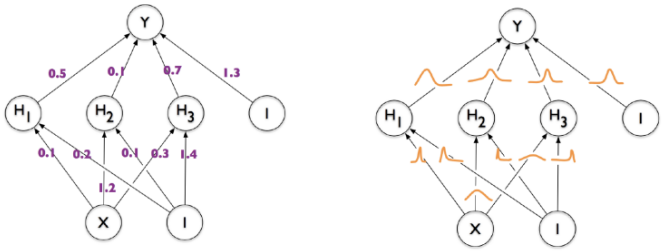
$$\mathbb{P}(y^* = 1|\mathbf{x}^*, \mathcal{D}) \approx \Phi\left(\frac{\mu_f}{\sqrt{\lambda^{-2} + \sigma_f^2}}\right) \tag{27}$$

where $\mu_f = \mu^\top \mathbf{x}^*$ and $\sigma_f^2 = \mathbf{x}^{*\top} \Sigma \mathbf{x}^*$.

2.3 Bayesian Neural Networks

Bayesian Neural Networks (BNNs) incorporate uncertainty into the modeling by placing probability distributions over the weights of the neural network and compute outputs based on their probability given the inputs and data, denoted as $\mathbb{P}(y_i|x_i, D)$. This approach contrasts with traditional neural networks, where weights are fixed values once the training is completed. In BNNs, predictions are made by considering the distribution of possible weights, which allows for more robust inference, particularly in situations where data is scarce or noisy.

The prior distribution over the weights is defined as $\mathbb{P}(\mathbf{w})$, for instance, a Gaussian distribution $\mathbb{P}(\mathbf{w}) = \mathcal{N}(\mathbf{w}|0, \alpha^{-1}I)$. This choice reflects a belief that, prior to observing data, weights are likely to be small in magnitude. In practice, the variance σ^2 is often set to α^{-1} for each layer.



The likelihood for regression problems is defined as $\mathbb{P}(y_i|x_i, \mathbf{w}) = \mathcal{N}(y_i; f^{\mathbf{W}}(x_i), \beta^{-1})$.

The goal is again to compute the posterior distribution $\mathbb{P}(\mathbf{w}|X, Y)$ by the following relation:

$$\mathbb{P}(\mathbf{w}|X, Y) \propto \mathbb{P}(\mathbf{w}) \prod_{i=1}^{N} \mathbb{P}(y_i|x_i, \mathbf{w})$$

Bayesian Neural Networks (BNNs) utilize Gaussian distributions for the prior and likelihood but result in a non-Gaussian posterior distribution:

- The Gaussian prior over weights is expressed as $\mathbb{P}(\mathbf{w}) = \mathcal{N}(\mathbf{w}|0, \alpha^{-1}I)$.
- The Gaussian likelihood for regression is $\mathbb{P}(y_i|x_i, \mathbf{w}) = \mathcal{N}(y_i; f^{\mathbf{W}}(x_i), \beta^{-1})$.
- Despite the Gaussian prior and likelihood, the posterior $\mathbb{P}(\mathbf{w}|X, Y)$ is not Gaussian due to the non-linear dependencies in the neural network.

This non-linear dependency is represented by the function $f^{\mathbf{W}}(x)$ which maps weights to predictions.

Recap of linear models for comparison:

- For a linear model, $\mathbb{P}(x) = \mathcal{N}(x|\mu_x, \Sigma_x)$ and $\mathbb{P}(y|x) = \mathcal{N}(y|Ax + b, \Sigma_y)$ where A and b are linear coefficients.
- The conditional probability $\mathbb{P}(x|y)$ for linear models is Gaussian due to the linear relationship.
- For BNNs, this linear Gaussian property does not hold, which complicates the computation of the posterior distribution.

So, the true predictive distribution in BNNs, denoted as $\mathbb{P}(y^*|x^*, D)$, does not admit closed form solutions and thus it requires approximation methods. Since the true

predictive distribution is represented by this integral:

$$\mathbb{P}(y^*|x^*, D) = \int \mathbb{P}(y^*|x^*, \mathbf{w})\mathbb{P}(\mathbf{w}|D)d\mathbf{w}$$

One can think of Monte Carlo estimation of this integral:

$$\mathbb{P}(y^*|x^*, D) \approx \frac{1}{S}\sum_{s=1}^{S} \mathbb{P}(y^*|x^*, \mathbf{w}^s) \quad \text{where} \quad \mathbf{w}^s \sim \mathbb{P}(\mathbf{w}|D)$$

Or if it is not possible to sample directly from $\mathbb{P}(\mathbf{w}|D)$, MCMC methods such as Metropolis-Hasting (MH) and Hamiltonian Monte Carlo (HMC) can be used for approximation.

However, these methods are effective for accurate posterior inference but have a drawback of not scaling well with large datasets.

2.3.1 Variational Inference

Variational Inference (VI) in Bayesian Neural Networks (BNNs) provides a practical approach to approximate this posterior distribution. Since exact inference is computationally infeasible in BNNs due to the high dimensionality of the parameter space and the complexity of the models. The process consists of several key components:

- We approximate the true posterior with what we call "the approximating variational distribution": defined as $q_\theta(\mathbf{w})$, parameterized by θ, the "variational parameters".
- The choice of the variational family for $q_\theta(\mathbf{w})$ is crucial. A common choice is to use a Gaussian distribution for each weight, where the mean and variance of these distributions are the variational parameters to be optimized. This choice is motivated by the trade-off between flexibility and computational tractability. So, the variational approximate posterior $q_\theta(\mathbf{w})$ is defined as a fully factorized Gaussian:

  $$q_\theta(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\theta) = \mathcal{N}(\mathbf{w}|\mu, \Sigma) = \prod_{i=1}^{D} \mathcal{N}(w_i|\mu_i, \sigma_i)$$

  And The variational parameters θ are $\{(\mu_j, \sigma_j)\}_{j=1}^{D}$, where μ_j and σ_j are the mean and variance for each weight w_j of the network.

- Then the goal of VI is to minimize the Kullback-Leibler divergence between the approximate posterior $q_\theta(\mathbf{w})$ and the true posterior:

$$KL(q_\theta(\mathbf{w})||\mathbb{P}(\mathbf{w}|X,Y)) = \int q_\theta(\mathbf{w}) \log \frac{q_\theta(\mathbf{w})}{\mathbb{P}(\mathbf{w}|X,Y)} d\mathbf{w}$$

  This is often operationalized through the maximization of the Evidence Lower Bound (ELBO), which serves as a lower bound to the marginal likelihood of the data. Then, the variational parameters θ are optimized to maximize the ELBO. This optimization is typically performed using gradient-based methods, where gradients can be efficiently computed using techniques such as the reparameterization trick for continuous distributions. We will go through this later.

- Once the optimal variational parameters are found, predictive inference in BNNs with VI involves integrating over the approximate posterior distribution of the weights to make predictions for new data.

$$\mathbb{P}(y^*|x^*, X, Y) \approx \int \mathbb{P}(y^*|x^*, \mathbf{w}) q_{\theta^*}(\mathbf{w}) d\mathbf{w}$$

  This is often approximated using Monte Carlo methods by sampling from the approximate posterior $q(\mathbf{w}|\theta)$.

Variational Inference in BNNs thus offers a scalable and flexible approach to Bayesian inference, enabling the modeling of uncertainty in neural networks while managing computational complexity.

So, let's now tackle the part on the Evidence Lower Bound (ELBO), which is a central concept in Variational Inference. First, we rewrite the Kullback-Leibler divergence as follows:

$$
\begin{aligned}
KL(q_\theta(\mathbf{w})||\mathbb{P}(\mathbf{w}|X,Y)) &= \int q_\theta(\mathbf{w}) \log \frac{q_\theta(\mathbf{w})}{\mathbb{P}(\mathbf{w}|X,Y)} d\mathbf{w} \\
&= -\int q_\theta(\mathbf{w}) \log \frac{\mathbb{P}(\mathbf{w}|X,Y)}{q_\theta(\mathbf{w})} d\mathbf{w} \\
&= -\int q_\theta(\mathbf{w}) \log \frac{\mathbb{P}(Y|X,\mathbf{w})\mathbb{P}(\mathbf{w})}{q_\theta(\mathbf{w})\mathbb{P}(Y|X)} d\mathbf{w} \\
&= -\int q_\theta(\mathbf{w}) \log \mathbb{P}(Y|X,\mathbf{w}) d\mathbf{w} + \int q_\theta(\mathbf{w}) \log \frac{q_\theta(\mathbf{w})}{\mathbb{P}(\mathbf{w})} d\mathbf{w} + \log \mathbb{P}(Y|X) \\
&= -\int q_\theta(\mathbf{w}) \log \mathbb{P}(Y|X,\mathbf{w}) d\mathbf{w} + KL(q_\theta(\mathbf{w})||\mathbb{P}(\mathbf{w})) + \log \mathbb{P}(Y|X)
\end{aligned}
$$

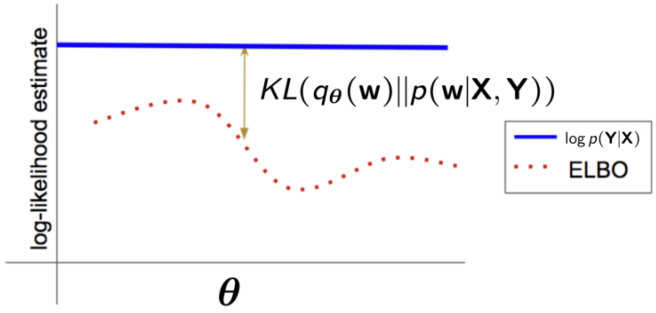
Then, we get the following result:

$$
KL(q_\theta(\mathbf{w})||\mathbb{P}(\mathbf{w}|X,Y)) = -\mathcal{L}_{VI}(X,Y,\theta) + \log \mathbb{P}(Y|X)
$$

Where the ELBO, denoted $\mathcal{L}_{VI}(X,Y,\theta) =: \mathcal{L}_{VI}(\theta)$, is defined by:

$$
\begin{aligned}
\mathcal{L}_{VI}(\theta) &= \int q_\theta(\mathbf{w}) \log \mathbb{P}(Y|X,\mathbf{w}) d\mathbf{w} - KL(q_\theta(\mathbf{w})||\mathbb{P}(\mathbf{w})) \\
\mathcal{L}_{VI}(\theta) &= \log \mathbb{P}(Y|X) - KL(q_\theta(\mathbf{w})||\mathbb{P}(\mathbf{w}|X,Y)) \leq \log \mathbb{P}(Y|X)
\end{aligned}
$$

which states that maximizing the ELBO is equivalent to minimizing the KL divergence, thereby improving the approximation of the true log likelihood.



Optimizing the Evidence Lower Bound (ELBO) involves two components:

- To maximize the ELBO, one must first minimize the KL divergence with

respect to $q_\theta(\mathbf{w})$:

$$\mathcal{L}_{VI}(\theta) = \mathbb{E}_{q_\theta(\mathbf{w})}[\log \mathbb{P}(Y|X, \mathbf{w})] - KL(q_\theta(\mathbf{w})||\mathbb{P}(\mathbf{w})) \leq \log \mathbb{P}(Y|X)$$

- This can be expanded to the summation over the data points:

$$\mathcal{L}_{VI}(\theta) = \sum_{i=1}^{N} \int q_\theta(\mathbf{w}) \log \mathbb{P}(y_i|f^{\mathbf{w}}(x_i))d\mathbf{w} - KL(q_\theta(\mathbf{w})||\mathbb{P}(\mathbf{w}))$$

- The expected log likelihood and prior KL terms are:
  - $\mathbb{E}q_\theta(\mathbf{w})[\log \mathbb{P}(Y|X, \mathbf{w})]$
    Maximized when $q_\theta(\mathbf{w})$ explains the observed data well. The expected log likelihood usually requires tractable calculations or estimation through sampling due to the absence of a closed-form solution.
  - $L(q_\theta(\mathbf{w})||\mathbb{P}(\mathbf{w}))$
    Minimized when $q_\theta(\mathbf{w})$ is close to the prior $\mathbb{P}(\mathbf{w})$. The KL divergence between the variational distribution and the prior can be integrated analytically in some cases.

So, optimizing the ELBO involves computing derivatives of the Evidence Lower Bound (ELBO) with respect to the variational parameters θ:

$$\mathcal{L}_{VI}(\theta) = \sum_{i=1}^{N} \int q_\theta(\mathbf{w}) \log \mathbb{P}(y_i|f^{\mathbf{w}}(x_i))d\mathbf{w} - KL(q_\theta(\mathbf{w})||\mathbb{P}(\mathbf{w}))$$

To make the computation of gradients manageable, a technique known as batch sampling is employed. This approach allows the ELBO, to be broken down linearly across individual training samples. This decomposition is fundamental for efficiently updating model parameters using subsets of the data at a time.

Modern solutions approximate the integral in the ELBO using Monte Carlo integration with weights sampled from the variational distribution:

$$\mathcal{L}_{VI}(\theta) = \sum_{i \in S} \log \mathbb{P}(y_i|f^{\mathbf{w}}(x_i)) - KL(q_\theta(\mathbf{w})||\mathbb{P}(\mathbf{w}))$$

The challenge in computing gradients with respect to variational parameters θ arises because sampling weights from $q_\theta(\mathbf{w})$ depends on θ.

To overcome this obstacle, the reparameterization trick is utilized, especially in cases involving Gaussian distributions. This technique involves expressing each weight as a deterministic function of a set of parameters and a source of randomness that is independent of those parameters:

$$\mathbf{w}_j = g((\mu_j, \sigma_j), \epsilon_j) = \mu_j + \sigma_j \cdot \epsilon_j,$$
$$\epsilon_j \sim \mathcal{N}(0,1), \quad \mathbf{w}_j \sim \mathcal{N}(\mu_j, \sigma_j)$$

The reparameterization of the ELBO facilitates the computation of gradients with respect to θ through backpropagation, leveraging the deterministic transformation of the random noise ϵ. The reparameterized ELBO remains as shown earlier:

$$\mathcal{L}_{VI}(\theta) = \sum_{i \in S} \log \mathbb{P}(y_i | f^{\mathbf{w}}(x_i)) - KL(q_\theta(\mathbf{w}) || \mathbb{P}(\mathbf{w}))$$

Here is an example of algorithm to do this:

Algorithm 1 Minimize divergence between $q_\theta(\mathbf{w})$ and $p(\mathbf{w}|X,Y)$

1: Given dataset X, Y,
2: Define learning rate schedule η,
3: Initialise parameters θ randomly,
4: **repeat**
5: Sample M random variables $\epsilon_i \sim p(\epsilon)$, S a random subset of $\{1, \ldots, N\}$ of size M.
6: Calculate stochastic derivative estimator w.r.t. θ:

$$\Delta\theta \leftarrow \frac{N}{M} \sum_{i \in S} \left(\frac{\partial}{\partial \theta} \log p(y_i | f^{g(\theta, \epsilon_i)}(x_i)) - \frac{\partial}{\partial \theta} KL(q_\theta(\mathbf{w}) || p(\mathbf{w})) \right)$$

7: Update θ:

$$\theta \leftarrow \theta + \eta \Delta\theta$$

8: **until** θ has converged.

Now that the optimal variational parameters are found, we want to deal with predictive inference. Recalling once more that the predictive distribution is given by:

$$\mathbb{P}(y^*|x^*, D) \approx \int \mathbb{P}(y^*|x^*, \mathbf{w}) q_\theta(\mathbf{w}) d\mathbf{w}$$

With no closed-form solution for $\mathbb{P}(y^*|x^*, D)$ even with $q_\theta(\mathbf{w})$ being Gaussian, due to non-linear dependence on $\mathbf{w}$.

So here is what we can do to tackle this issue:

1. Either use Monte Carlo (MC) sampling, which is straightforward from $q_\theta(\mathbf{w})$ since it's Gaussian.

2. Or use a Taylor expansion of y^* around $\mathbf{w}_{MAP}$ for regression (for classification we can look at the logit Taylor expansion):

$$y(x, \mathbf{w}) \approx y(x, \mathbf{w}_{MAP}) + \left.\frac{\partial y}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_{MAP}} (\mathbf{w} - \mathbf{w}_{MAP})$$

   Then, the predictive distribution approximates to a Gaussian:

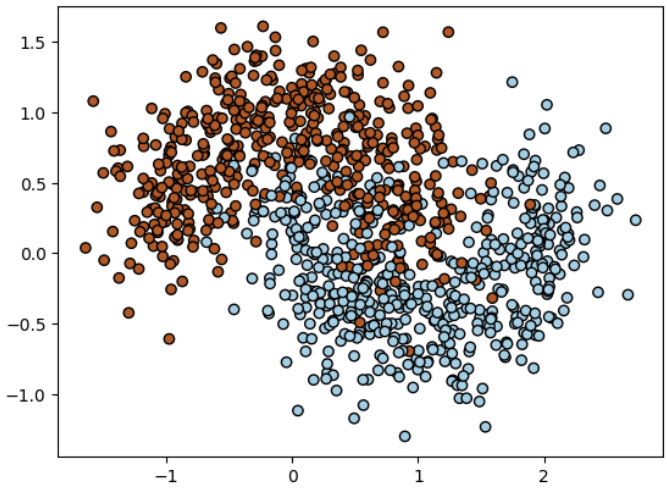
$$\mathbb{P}(y^*|x^*, \mathbf{w}) \approx \mathcal{N}(y^*|\mu_y, \beta^{-1}),$$

   where μ_y is given by the Taylor expansion around $\mathbf{w}_{MAP}$.
   So we get the closed-form solution of $\mathbb{P}(y^*|x^*, D)$:

$$\mathbb{P}(y^*|x^*, D) = \mathcal{N}(y^*|y(x, \mathbf{w}_{MAP}), \sigma^2(x))$$

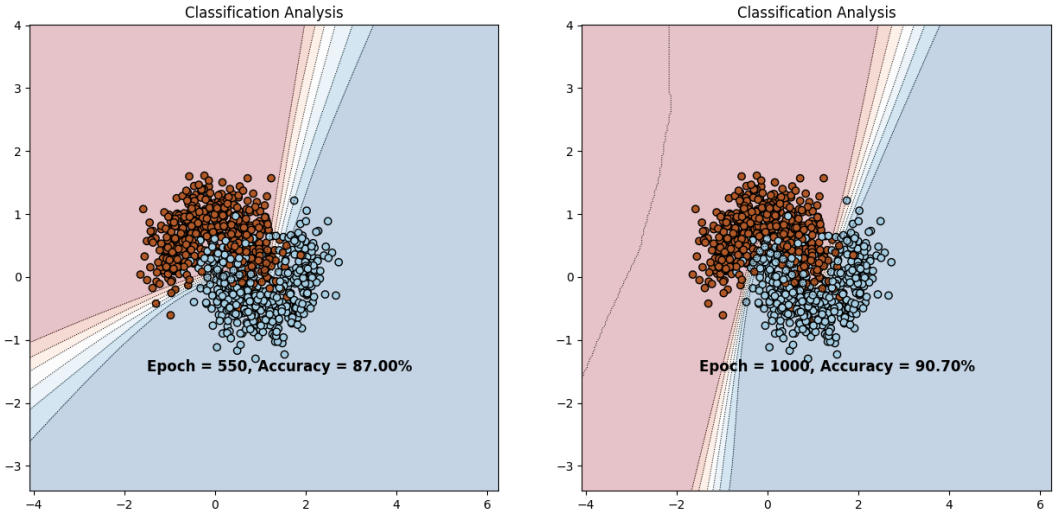
   with $\sigma^2(x) = \beta^{-1} + g^T A^{-1} g$, where g is the gradient and A is the Hessian at w_{MAP}

Here are an example of what we can get using VI on a moon dataset:

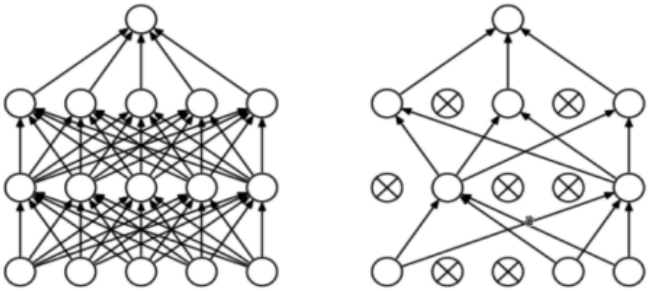


2.4 Monte Carlo Dropout

Monte Carlo Dropout is a regularization technique to estimate uncertainty in neural network predictions by applying dropout at test time as well as during training.



Dropout, which involves randomly ignoring a subset of neurons, simulates various network architectures, it is also used to limit over-fitting, by preventing co-adaptation of neurons, promoting better model generalization. In practice, we randomly set a proportion of the elements in the input vector to zero during training. This is akin to sampling from a Bernoulli distribution for each element with a certain probability p, typically $p = 0.5$, deciding whether it stays or is 'dropped out'.



This method can be viewed as averaging over many neural network architectures. Then the variance in predictions from these architectures can estimate the model uncertainty. But of course it may leads to slower convergence.

Here are the steps to follow to do Monte Carlo Dropout:

- Train the network with dropout.
- During inference, keep dropout enabled and make multiple forward passes.
- Average the predictions to obtain a final output. The prediction spread estimates uncertainty.

We highlight the main differences between Monte Carlo dropout and VI with pros and cons:

- Modeling of Uncertainty: Monte Carlo Dropout treats dropout as a Bayesian approximation, while VI models the posterior distribution of weights explicitly.
- Mathematical Foundation: VI is based on Bayesian statistics with a formal approximation of the posterior, whereas Monte Carlo Dropout is a heuristic for uncertainty estimation.
- Complexity: VI is generally more complex and computationally intensive due to the optimization of the variational posterior.
- Interpretability: VI parameters have probabilistic interpretations, unlike dropout rates in Monte Carlo Dropout.

Monte Carlo Dropout pros:

- Simplicity: It is relatively easy to implement with existing neural networks.
- Efficiency: Less computationally intensive than VI.

Monte Carlo Dropout cons:

- Less Theoretical: Lacks the formal Bayesian interpretation of VI.
- Approximation Quality: The uncertainty estimate may not always be as reliable as VI.

VI pros:

- Rigorous: Grounded in the Bayesian framework with a clear interpretation of model uncertainty.
- Flexible: Can accommodate complex posterior distributions.

VI cons:

- Complexity: Requires more complex optimization and can be computationally expensive.
- Implementation: More challenging to implement and tune compared to dropout.

Let us now study this in detail, our framework is the following:

The input to the network is a vector $x \in \mathbb{R}^D$, and the corresponding latent vector is $h \in \mathbb{R}^L$. This latent vector is obtained by applying a non-linear function σ (e.g., sigmoid or ReLU) to the product of the input vector and the weights matrix $\mathbf{W}_1$.

Mathematically, this is represented as: $h = \sigma(x\mathbf{W}_1)$.

Then, for dropout sampling, we represent the dropout mask as $\hat{\boldsymbol{\epsilon}} = \{\hat{\epsilon}_i^{(1)}\}_{i=1;D}$, where each $\hat{\epsilon_i}$ is sampled from a $Bernoulli(1 - \Pr(\epsilon))$ distribution. The probability $\Pr(\epsilon)$ here is the probability of an element being set to zero.

Applying dropout to the first layer: We modify the input by element-wise multiplying it with the dropout mask, and then compute the weighted sum with the weights matrix $\mathbf{W}_1$.

The operation can be written as: $h = \sigma((x \odot \hat{\boldsymbol{\epsilon}})\mathbf{W}_1)$.

In effect, this is equivalent to: Randomly setting to zero the rows of $\mathbf{W}_1$ with probability $\Pr(\epsilon)$, which can be represented by a diagonal matrix of the dropout mask. We will denote:

$$(x \cdot \text{diag}(\hat{\epsilon}))W_1 = \hat{x}(\text{diag}(\hat{\epsilon})W_1) = x\hat{W}_1$$

Typically for a 2 layer NN (1 hidden):

Dropout is applied as a Bernoulli distribution with probability $1 - p_i$, so $\hat{\boldsymbol{\epsilon}}_1 \sim Bernoulli(1 - p_i)$:

- The hidden layer output h_1 is computed as:

$$h_1 = \sigma(\hat{x}\mathbf{W}_1) = \sigma(x\hat{\mathbf{W}}_1), \quad \hat{\mathbf{W}}_1 = \text{diag}(\hat{\boldsymbol{\epsilon}}_1)\mathbf{W}_1$$

  where σ is the activation function.

- The predicted output $\hat{y}$ is then computed as:

$$\hat{y} = f^{\hat{\mathbf{W}}_1, \hat{\mathbf{W}}_2}(x) = \hat{h_1}\mathbf{W}_2 = h_1\hat{\mathbf{W}}_2, \quad \hat{\mathbf{W}}_2 = \text{diag}(\hat{\boldsymbol{\epsilon}}_2)\mathbf{W}_2$$

So, now that we understand dropout we come back to the main idea of the Monte Carlo Dropout sampling: approximate the predictive distribution:

$$\frac{1}{S}\sum_{s\in S}\mathbb{P}(y_i|f^{\hat{\mathbf{W}}}(x_i)) \approx \int \mathbb{P}(y^*|f^{\mathbf{W}}(\mathbf{x}^*))q(\mathbf{W})d\mathbf{w}$$

We generalize the previous example:

- For each layer l with random variable $\mathbf{W}_l$, the dropout mask $\hat{\boldsymbol{\epsilon}}_{l,i} \sim Bernoulli(1-p_l)$ is applied:

$$\mathbf{W}_l \sim q(\mathbf{W}_l) = g(\mathbf{M}_l, \hat{\boldsymbol{\epsilon}}_l) = \text{diag}(\hat{\boldsymbol{\epsilon}}_l)\mathbf{M}_l$$

  where $\mathbf{M}_l$ are deterministic parameters.

- This new variational distribution $q(\mathbf{W})$ is factored across layers:

$$q(\mathbf{W}) = \prod_{l=1}^{L} q(\mathbf{W}_l)$$

We now provide a **big result**:

- Training a NN with dropout is equivalent to training a BNN with variational posterior approximation $q_M(\mathbf{W})$ and a prior $p(\mathbf{W})$. With the variational parameters $M = (M_l)_{l\in(1,L)}$.

- Sampling several passes with dropout is equivalent to perform Monte Carlo approximation of inference with the variational posterior $q_M(\mathbf{W})$.

$$\frac{1}{S}\sum_{s\in S}\mathbb{P}(y_i|f^{\hat{\mathbf{W}}}(x_i)) \approx \int \mathbb{P}(y^*|f^{\mathbf{W}}(\mathbf{x}^*))q(\mathbf{W})d\mathbf{w} \approx \mathbb{P}(y^*|x^*, X, Y)$$

Here is a sketch of the **proof** for a 2 layer NN doing regression:

- The prediction in a NN with dropout is done by passing the input x through the network with dropout masks applied to the weights $\mathbf{W}_1$ and $\mathbf{W}_2$. For each

layer l, the modified weights $\hat{\mathbf{W}}_l$ are the product of the dropout mask $\text{diag}(\hat{\boldsymbol{\epsilon}}_l)$ and the original weights $\mathbf{M}_l$:

$$\hat{y} = \sigma(x\hat{\mathbf{W}}_1)\hat{\mathbf{W}}_2 =: f^{\hat{\mathbf{W}}}(x)$$

with $\hat{\mathbf{W}} = \{\hat{\mathbf{W}}_1, \hat{\mathbf{W}}_2\}$.

- The objective function for training with dropout, $\mathcal{L}_{\text{dropout}}$, combines the mean squared error with L2 regularization for each set of weights:

$$\mathcal{L}_{\text{dropout}}(M_1, M_2) = \frac{1}{M}\sum_{i\in S}\left\|f^{\hat{\mathbf{W}}}(x_i) - y_i\right\|^2 + \lambda_1\|M_1\|^2 + \lambda_2\|M_2\|^2$$

where M is the number of samples and S is the set of samples.

- Assuming a Gaussian likelihood for the predictions, which implies a negative log-likelihood proportional to the mean squared error, we have:

$$\mathbb{P}(y_i|f^{\hat{\mathbf{W}}}(x_i)) = \mathcal{N}(y, f^{\hat{\mathbf{W}}}(x), \tau^{-1}), \quad \text{and} \quad \left\|f^{\hat{\mathbf{W}}}(x_i) - y_i\right\|^2 = -\frac{1}{\tau}\log\mathbb{P}(y_i|f^{g(M,\hat{\boldsymbol{\epsilon}}_i)}(x))$$

- The dropout objective function can be rewritten in terms of the Gaussian likelihood:

$$\mathcal{L}_{\text{dropout}}(M_1, M_2) = -\frac{1}{M\tau}\sum_{i\in S}\log\mathbb{P}(y_i|f^{g(M,\hat{\boldsymbol{\epsilon}}_i)}(x)) + \lambda_1\|M_1\|^2 + \lambda_2\|M_2\|^2$$

- There is a strong similarity between the dropout objective function $\mathcal{L}_{\text{dropout}}$ and the regularization term in the cost function. The same algorithms can be used if the derivative of the Kullback-Leibler divergence with respect to the model parameters M is proportional to the derivative of the regularization term:

$$\frac{\partial}{\partial M}KL(q(W)||p(W)) = \frac{\partial}{\partial M}\mathcal{N}_T(\lambda_1||M_1||^2 + \lambda_2||M_2||^2)$$

- Gal and Ghahramani (2016) demonstrated that this condition can be satisfied if:

  - The prior distribution $p(W)$ is factored over layers and defined as a product of independent multivariate Gaussian distributions:

$$p(W) = \prod_l p(W_l) = \prod_l \mathcal{MN}(W_l; 0, I/l^2, I)$$

– The variational distribution $q(W)$ is defined as a product of Bernoulli-distributed diagonal matrices multiplied by deterministic matrices M:

$$q(W_l) = \text{diag}(\hat{\boldsymbol{\epsilon}}_l) M_l, \quad \hat{\boldsymbol{\epsilon}}_{l,i} \sim Bernoulli(1 - p_i), \quad q(W) = \prod_l q(W_l)$$

– This variational distribution can be approximated by a mixture of two Gaussians with a small standard deviation and one component fixed at zero:

$$q_{\theta_{i,k}}(W_{i,k}) = (1 - p_i)\mathcal{N}(W_{i,k}; m_{i,k}; \sigma^2/l) + p_i \mathcal{N}(W_{i,k}; 0; \sigma^2/l)$$

Consequently, a neural network employing dropout can be interpreted as performing variational Bayesian approximation, where the dropout technique corresponds to sampling from the approximate posterior distribution. This interpretation allows us to view the use of dropout not just as a regularization method, but also as an approximation of Bayesian inference, where the network's predictions take into account the uncertainty in its weights. This is akin to having an ensemble of neural networks and averaging their predictions.

2.4.1 Model Uncertainty

Model uncertainty in predictions is quantified using variational inference, approximating the predictive distribution as follows:

$$\begin{aligned} \mathbb{P}(y^*|x^*, X, Y) &= \int \mathbb{P}(y^*|f^{\mathbf{W}}(x^*))\mathbb{P}(\mathbf{W}|X, Y)d\mathbf{w} \\ &\approx \int \mathbb{P}(y^*|f^{\mathbf{W}}(x^*))q(\mathbf{W})d\mathbf{w} =: q_{\mathbf{W}^*}(y^*|x^*) \end{aligned}$$

Recalling:

- $\mathbb{P}(y^*|f^{\mathbf{W}}(x^*))$ is the likelihood of the target y^* given the output of the model $f^{\mathbf{W}}(x^*)$, parameterized by weights $\mathbf{W}$.
- $\mathbb{P}(\mathbf{W}|X, Y)$ is the posterior distribution of the weights given the training data.

- $q(\mathbf{W})$ is the variational approximation of the posterior distribution of the weights.
- $q_{\mathbf{W}^*}(y^*|x^*)$ represents the variational approximation of the predictive distribution.

To estimate the predictive distribution, Monte Carlo (MC) sampling is used:

$$\text{Estimate } \mathbb{P}(y^*|x^*, X, Y) \text{ by MC sampling of } \mathbb{P}(y^*|f^{\hat{\mathbf{W}}}(x^*)), \quad \hat{\mathbf{W}} \sim q(\mathbf{W})$$

Here:

- $\mathbf{W} = \{\mathbf{W}_l\}_{l=1}^{L}$ represents the set of all random variables corresponding to the weights of the model across all layers l.
- $f^{\mathbf{W}}(x^*)$ is the stochastic output of the model for input x^*, using weights $\mathbf{W}$.
- $q_{\mathbf{W}^*}(\mathbf{W})$ is the optimal variational distribution approximating the posterior distribution over the weights.

To quantify model uncertainty in the context of regression, we perform moment-matching and estimate the first two moments of the predictive distribution empirically. To do so, we use these two propositions:

Proposition 1:

- If the likelihood $\mathbb{P}(y^*|f^{\mathbf{W}}(x^*))$ is Gaussian with mean $f^{\mathbf{W}}(x^*)$ and precision τ^{-1} for some $\tau > 0$, then the expectation $\mathbb{E}_{q^*_{\mathbf{W}}(y^*|x^*)}[y^*]$ can be estimated with the unbiased estimator:

$$\widetilde{\mathbb{E}}[y^*] := \frac{1}{T}\sum_{t=1}^{T} f^{\hat{\mathbf{W}}_t}(x^*) \to \mathbb{E}_{q^*_{\mathbf{W}}(y^*|x^*)}[y^*] \text{ as } T \to \infty$$

- Here, $\hat{\mathbf{W}}_t \sim q(\mathbf{W})$ represents the weights sampled from the variational distribution at each iteration t.
- This process is equivalent to performing T stochastic forward passes through the network and averaging the results, which provides an empirical estimation of the predictive mean.

Proposition 2: Given the Gaussian likelihood of the predictions, the covariance of the predictive distribution can be estimated with an unbiased estimator:

$$\widetilde{\mathbb{E}}[(y^*)^T(y^*)] := \tau^{-1}I + \frac{1}{T}\sum_{t=1}^{T} f^{\hat{\mathbf{W}}_t}(x^*)^T f^{\hat{\mathbf{W}}_t}(x^*),$$

where $\hat{\mathbf{W}}_t \sim q(\mathbf{W})$ are the weights sampled from the variational distribution. As T approaches infinity, this estimator converges to the true covariance under the variational distribution $q^*_{\mathbf{W}}(y^*|x^*) : \mathbb{E}_{q^*_{\mathbf{W}}(y^*|x^*)}[(y^*)^T(y^*)]$.

Corollary: The variance of the predictive distribution y^* can also be estimated with an unbiased estimator:

$$\widetilde{\mathrm{Var}}[(y^*)] := \tau^{-1}I + \frac{1}{T}\sum_{t=1}^{T} f^{\hat{\mathbf{W}}_t}(x^*)^T f^{\hat{\mathbf{W}}_t}(x^*) - \left(\frac{1}{T}\sum_{t=1}^{T} f^{\hat{\mathbf{W}}_t}(x^*)\right)^T \left(\sum_{t=1}^{T} f^{\hat{\mathbf{W}}_t}(x^*)\right),$$

Again, as T grows large, this estimator converges to the true variance under the variational distribution $q^*_{\mathbf{W}}(y^*|x^*)$: $\mathrm{Var}_{q^*_{\mathbf{W}}(y^*|x^*)}[(y^*)]$

This method allows for the quantification of uncertainty in the model's predictions by considering the variability in the output due to the stochastic nature of the weights. This is crucial for understanding the confidence in the predictions made by the model.

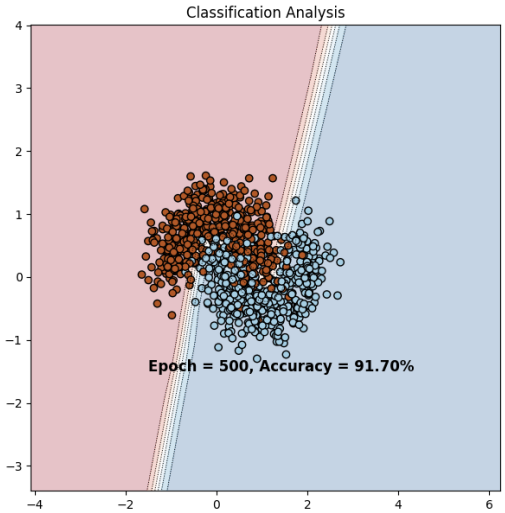


Figure 5: Deterministic NN

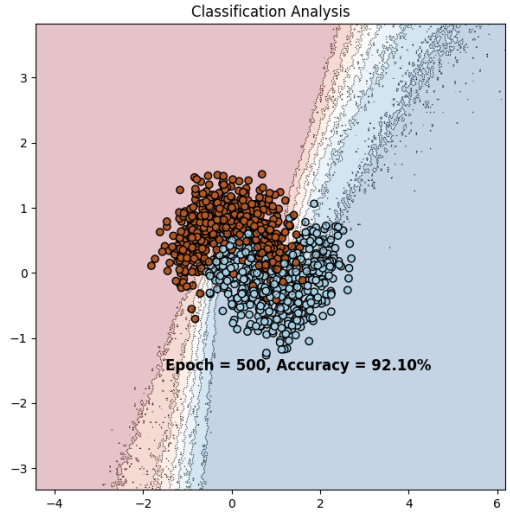


Figure 6: Bayesian NN (MC dropout)

References

[1] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[2] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[3] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

[4] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. *CoRR*, abs/1706.04599.

[5] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[6] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[7] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1613–1622.

[8] Gal, Y. (2016). *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge.

[9] Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, pages 1050–1059.

[10] Graves, A. (2011). Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems 24*, pages 2348–2356.

[11] Hernandez-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1861–1869.

[12] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

[13] Hinton, G. E. and van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, pages 5–13.

[14] Jylänki, P., Nummenmaa, A., and Vehtari, A. (2014). Expectation propagation for neural networks with sparsity-promoting priors. *Journal of Machine Learning Research*, 15:1849–1901.

[15] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Conference Track Proceedings*, Banff, AB, Canada, April 14-16.

[16] MacKay, D. J. C. (1992). A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472.

[17] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

[18] Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer-Verlag Berlin, Heidelberg.