Bayesian deep learning

Deep Learning Practical Work

Aymeric Delefosse & Charles Vin

2023 – 2024

Contents

Chapter 1

Bayesian Linear Regression

In this chapter, we delve into Bayesian Linear Regression, a fundamental concept in Bayesian Deep Learning. After discussing the general landscape of deep learning technologies and acknowledging the associated challenges, including problems related to interpretability and uncertainty in predictions, we turn our attention to Bayesian models. Bayesian Linear Regression serves as an ideal starting point for understanding Bayesian methods due to its relative simplicity and the availability of a closed-form solution for its posterior distribution. The variance present in these distributions captures epistemic uncertainty, setting it apart from the deterministic single-point predictions provided by traditional neural networks. Throughout this chapter, we not only introduce the concept of Bayesian Linear Regression but also contextualize its importance within the broader framework of Bayesian Deep Learning, providing a solid foundation for comprehending Bayesian approaches in machine learning.

1.1 Linear Basis function model

In this section, we introduce the fundamental concepts of Bayesian Linear Regression. We focus on laying a strong theoretical foundation, explaining the underlying mathematics, and outlining why a Bayesian approach is chosen for linear regression tasks.

1.1.1 Gaussian basis functions

1.2. Recall closed form of the posterior distribution in linear case. Then, code and visualize posterior sampling. What can you observe? Let N denote the number of training examples, K the number of dimensions of the outputs, and p the number of features (or predictors) in the input data. Consider $X \in \mathbb{R}^{N \times p}$, the input matrix, and $Y \in \mathbb{R}^{N \times K}$, the output matrix.

We typically define the posterior distribution as $p(w|X, Y)$. This distribution represents our updated beliefs about the parameters w after observing the data X and Y. According to Bayes' rule, we can express the posterior distribution as the product of two components:

1. $p(w)$, which represents our prior beliefs about the distribution of w before observing the data.
2. $p(Y|X, w)$, indicating the probability of observing the data Y given the parameters w and the data X. This quantifies how likely our data is under different hypotheses represented by w.

Therefore, the formula for the posterior distribution is given by:

$$p(w|X, Y) \propto p(Y|X, w)p(w)$$

In this formula, we start with our initial beliefs (priors) and then update these beliefs based on the new data (likelihood). In the case of our linear model, we know that $y_i = \Phi_i^T w + \epsilon$, with $\Phi \in \mathbb{R}^{N \times (p+1)}$ representing the design matrix and ϵ denoting the residual. Assuming that the error follows a centered Gaussian distribution with standard deviation $\beta^{-1} = 2\sigma^2$, meaning that $\epsilon \sim \mathcal{N}(0, \beta^{-1})$. Consequently, we can conclude that:

$$p(y_i|x_i, w) \sim \mathcal{N}(\Phi_i^T w, \beta^{-1})$$

Furthermore, we selected a centered Gaussian prior with a variance of $\alpha^{-1}I$ where α governs the prior distribution over the weights $\mathbf{w}$:

$$p(w|\alpha) \sim \mathcal{N}(0, \alpha^{-1}I)$$

In this specific case, we can demonstrate that the posterior distribution $p(w|X,Y)$ follows a Gaussian distribution as follows:

$$p(w|X,Y) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

The precision matrix $\boldsymbol{\Sigma}$, which is the inverse of the covariance matrix of the distribution parameters, is defined as:

$$\boldsymbol{\Sigma}^{-1} = \alpha I + \beta \Phi^T \Phi$$

The mean of the distribution parameters is given by:

$$\boldsymbol{\mu} = \beta \boldsymbol{\Sigma} \Phi^T Y$$

Parameters α and β serve analogous roles, with α governing the prior distribution and β regulating the likelihood.

Now, we can proceed to sample from the updated (with data) posterior distribution $p(w|X,Y)$. This process is demonstrated in Figure 1.1, with $\alpha = 2$ and $\beta = (2 \times 0.2^2)^{-1}$. It is worth noting that when $N = 0$, the posterior distribution $p(w|X,Y)$ simplifies to the prior distribution $p(w)$. As we increase the number of data points, we can observe how the model's certainty increase, demonstrated by the reduction in the variance of the parameters of the distribution. In other words, having more data points reduces aleatoric uncertainty.

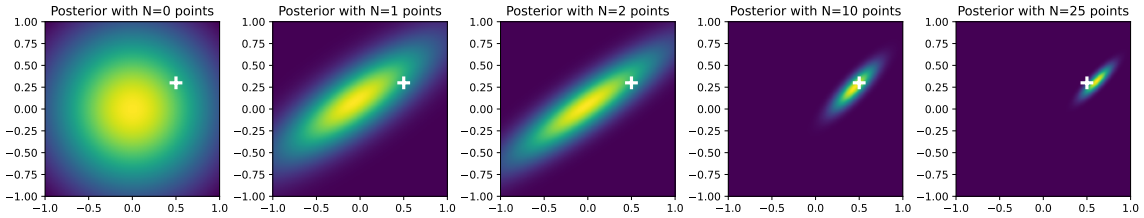


Figure 1.1: Evolution of a Bayesian posterior distribution with increasing data points: A visual representation of Bayesian inference, depicting how the posterior distribution updates as more data is incorporated. From left to right, the figures show the posterior with $N = 0$ (prior distribution), $N = 1$, $N = 2$, $N = 10$, and $N = 25$ data points, respectively. The cross mark represents the ground truth.

1.3. Recall and code closed form of the predictive distribution in linear case. Using the information from the previous question, we can now calculate the predictive distribution for new data point $x^\star$ by marginalizing over the parameter w. This is represented by the following equation where $\mathcal{D} = \{X, Y\}$ denotes the dataset:

$$p(y|x^\star, \mathcal{D}, \alpha, \beta) = \int p(y|x^\star, w, \beta) p(w|\mathcal{D}, \alpha, \beta)$$

By leveraging the property that the convolution of two Gaussian distributions results in another Gaussian distribution, we can demonstrate that the closed-form expression for the predictive distribution in the linear case is as follows:

$$p\left(y|x^\star; \mathcal{D}, \alpha, \beta\right) = \mathcal{N}\left(y; \mu^T \boldsymbol{\Phi}(x^\star), \frac{1}{\beta} + \boldsymbol{\Phi}(x^\star)^T \boldsymbol{\Sigma} \boldsymbol{\Phi}(x^\star)\right)$$

We can see that the variance in the predictive distribution σ^2_{pred} for a new observation can be divided into two parts:

1. The aleatoric uncertainty, which represents the inherent noise in the data, that we fixed around β^{-1};
2. The epistemic uncertainty related to the model parameters w, characterized by $\boldsymbol{\Phi}(x^\star)^T \boldsymbol{\Sigma} \boldsymbol{\Phi}(x^\star)$.

It's worth noting that as the number of data points N approaches infinity ($\lim_{N \to \infty} \boldsymbol{\Phi}(x^\star)^T \boldsymbol{\Sigma} \boldsymbol{\Phi}(x^\star) = 0$), our understanding of the model parameters becomes nearly perfect. In this scenario, the only remaining source of uncertainty is the aleatoric uncertainty, stemming from the noise in the data.

1.4. Based on previously defined `f_pred()`, predict on the test dataset. Then visualize results using `plot_results()` defined at the beginning of the notebook. Figure 1.2 illustrates a comparison between the predictions made by a Bayesian Linear Regression model and the actual ground truth. In the left panel, you can see the model's linear fit to the training data, shown as blue points, alongside the true ground truth represented by the green line. To visualize the model's predictive uncertainty, shaded areas are used, ranging from dark to light, which correspond to one, two, and three standard deviation intervals, respectively.

The right panel of the figure focuses on the predictive variance σ^2_{pred} along the x-axis. This variance is depicted by a curve that widens as it moves away from the center of the training data, marked by the vertical

dashed line. These visual elements together provide a comprehensive view of the model's confidence in its predictions across the entire domain.

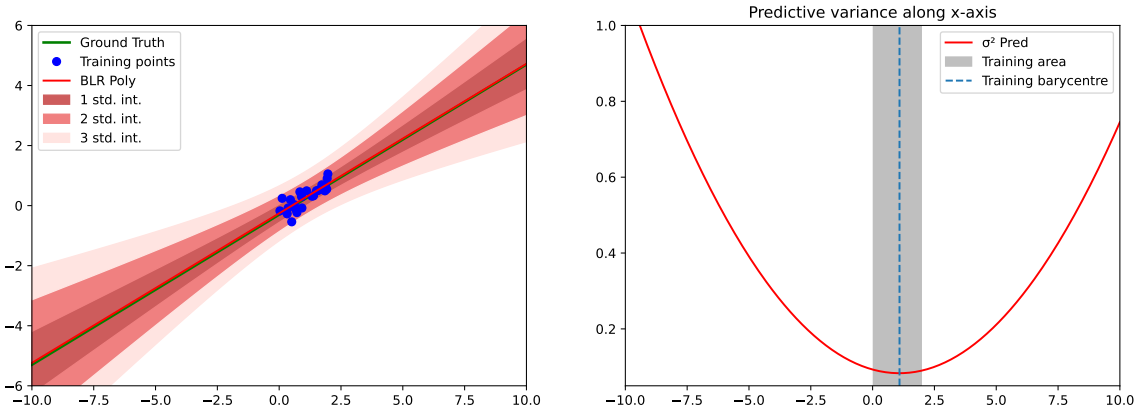


Figure 1.2: Visualization of predictive distribution of a linear dataset using Bayesian Linear Regression. The left panel illustrates the fit (red line) to the training data (blue points) against the ground truth (green line). The shaded areas represent the predictive uncertainty, with one, two, and three standard deviation intervals shown in progressively lighter shades. The right panel displays the predictive variance σ^2_{pred} across the x-axis. The vertical dashed line indicates the center of the training data.

1.5. Analyse these results. Why predictive variance increases far from training distribution? Prove it analytically in the case where $\alpha = 0$ and $\beta = 1$. In the left panel of Figure 1.2, we can observe that the confidence intervals are narrowest near the cluster of training points. This suggests higher confidence in predictions within this area. As we move away from the center of the training data (towards the extremities of the x-axis), the confidence intervals become wider, indicating increasing uncertainty in the model's predictions.

Looking at the right panel of Figure 1.2, we notice that the variance remains low in the region where the training data is located (the grey shaded area). This correlates with the tight confidence intervals shown in the left panel. As expected, the predictive variance is lowest near the training barycentre, reflecting greater model certainty (i.e. lower epistemic uncertainty) in this region due to the presence of more training data points. As we move away from the training area on either side, the predictive variance increases significantly, which is consistent with the expanding confidence intervals in the left panel. This sharp increase in variance indicates a significant decrease in the model's confidence (i.e. a significant increase in the model's epistemic uncertainty) in its predictions outside the range of the training data. This pattern is expected, as the model has less points to rely on for predictions in these regions.

Let's prove it analytically. With $\alpha = 0$ and $\beta = 1$, the computation of $\mathbf{\Sigma}^{-1}$ simplifies as follows:

$$\begin{aligned}
\mathbf{\Sigma}^{-1} &= 0 \cdot I_3 + 1 \cdot \Phi^T \Phi \\
&= \Phi^T \Phi \\
&= \begin{pmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_N \end{pmatrix} \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_N \end{pmatrix} \\
&= \begin{pmatrix} N & \sum x_i \\ \sum x_i & \sum x_i^2 \end{pmatrix}
\end{aligned}$$

To invert $\mathbf{\Sigma}$, we use the classic formula for a 2×2 matrix:

$$\mathbf{\Sigma} = \frac{1}{\det \mathbf{\Sigma}^{-1}} \begin{pmatrix} \sum x_i^2 & -\sum x_i \\ -\sum x_i & N \end{pmatrix}$$

Returning to our expression for $\sigma^2_{\text{pred}} = \Phi(x^\star)^T \mathbf{\Sigma} \Phi(x^\star)$, we have:

$$\begin{aligned}
\sigma^2_{\text{pred}} = \Phi(x^\star)^T \mathbf{\Sigma} \Phi(x^\star) &= \frac{1}{\det \mathbf{\Sigma}^{-1}} \begin{pmatrix} 1 \\ x^\star \end{pmatrix} \begin{pmatrix} \sum x_i^2 & -\sum x_i \\ -\sum x_i & N \end{pmatrix} \begin{pmatrix} 1 & x^\star \end{pmatrix} \\
&= \frac{1}{\det \mathbf{\Sigma}^{-1}} \begin{pmatrix} \sum x_i^2 - x^\star \sum x_i \\ x^\star N - \sum x_i \end{pmatrix} \begin{pmatrix} 1 & x^\star \end{pmatrix} \\
&= \frac{1}{\det \mathbf{\Sigma}^{-1}} \left(\sum_{i=1}^N x_i^2 - x^\star \sum_{i=1}^N x_i + x^{\star 2} N - x^\star \sum_{i=1}^N x_i \right) \\
&= \frac{1}{\det \mathbf{\Sigma}^{-1}} \left(\sum_{i=1}^N x_i^2 - 2x^\star \sum_{i=1}^N x_i + x^{\star 2} N \right) \\
&= \frac{1}{\det \mathbf{\Sigma}^{-1}} \sum_{i=1}^N (x_i - x^\star)^2
\end{aligned}$$

This formula reveals that the predictive variance is directly proportional to epistemic uncertainty, which, in the case of our linear regression, is the squared differences between each training data point x_i and the prediction point $x^\star$. As $x^\star$ moves away from the region where the training data is concentrated, these squared differences grow, consequently leading to an increase in the predictive variance.

Bonus: What happens when applying Bayesian Linear Regression on the following dataset? Examining the right panel in Figure 1.3, an intriguing observation is made: the variance is unexpectedly minimized at the barycenter, i.e. the "hole" where there are no training data points. Normally, one would anticipate an increase in variance in data-scarce regions. However, this phenomenon can be explained by the fact that a point within this region is positioned closely between two clusters of training points. This proximity results in lower squared differences (epistemic uncertainty) and, as previously discussed, leads to a lower predictive variance. Additionally, due to the broader distribution of our training dataset, we observe lower predictive variance at the endpoints when compared to our previous results. This overconfidence can be attributed to our strong prior belief in the linearity of the data. In fact, we can perceive it in this way: when we draw a line between two points, we have more certainty about its direction when it passes through two distant points than through two close points.

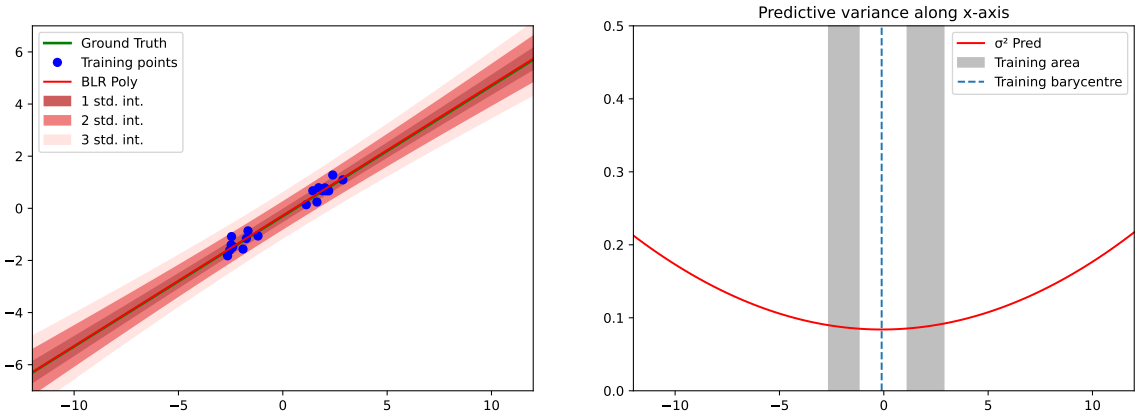


Figure 1.3: Visualization of predictive distribution of a linear dataset featuring a "hole" using Bayesian Linear Regression.

1.2 Non Linear models

In this section, we apply the theoretical concepts learned in the previous section to more intricate datasets. The goal is to gain insights into the significance of the selected basis function and its impact on the behavior of predictive variance.

1.2.1 Polynomial basis functions

2.2. Code and visualize results on sinusoidal dataset using polynomial basis functions. What can you say about the predictive variance? Figures 1.4 and 1.5 illustrate a comparison between the predictions

made by a Bayesian Polynomial Regression model and the actual ground truth. Given that the closed-form solution for the posterior and predictive distribution in Φ is similar to the linear case, we can draw the same conclusions: as we move further away from our training data, uncertainty increases. However, due to our polynomial kernel, the model exhibits higher predictive variance (i.e. higher epistemic uncertainity) when considering the values alone. Nevertheless, thanks to this basis function, our model demonstrates the ability to capture more intricate patterns, such as those resembling a sinusoidal function. It's worth noting that beyond the training data points, the model struggles to generalize, as it lacks the necessary data points to accurately capture the periodic nature of the sinusoidal ground truth.

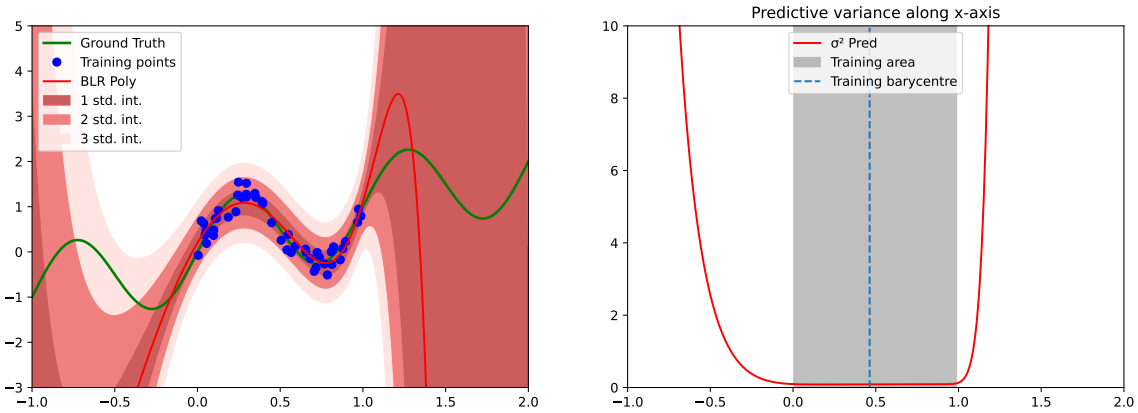


Figure 1.4: Visualization of predictive distribution of a sinusoidal dataset using Bayesian Polynomial Regression.

Furthermore, we decided to explore the effects of providing data points that are more dispersed and less abundant, as shown in Figure 1.5. Unsurprisingly, we observe that the variance around these small clusters is minimal, and as we move away from these regions, the variance increases. This phenomenon highlights the strength of these models: when given data, they become increasingly confident in specific areas. Therefore, in the context of uncertainty, we can gradually narrow down our desired outcomes by continuously adding more data points over time.

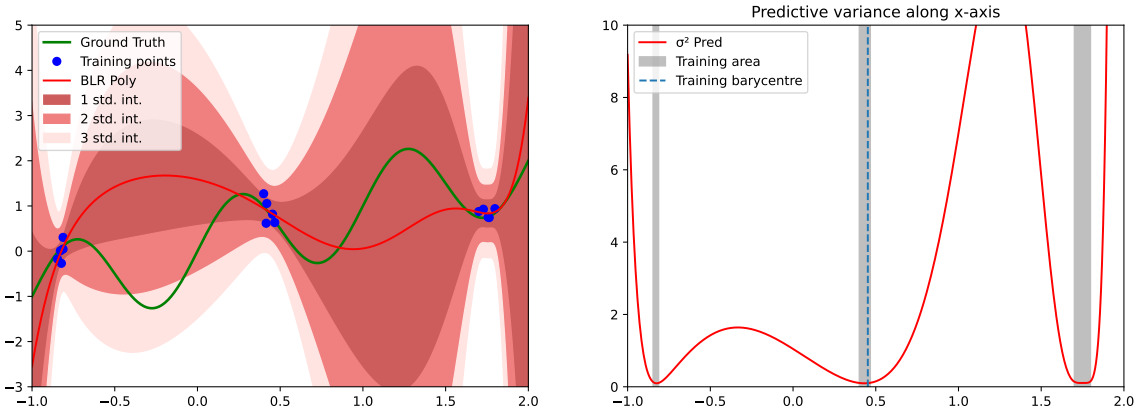


Figure 1.5: Visualization of predictive distribution of a sinusoidal dataset featuring "holes" and sparse data points using Bayesian Polynomial Regression.

1.2.2 Gaussian basis functions

2.4. Code and visualize results on sinusoidal dataset using Gaussian basis functions. What can you say this time about the predictive variance? Figures 1.6 and 1.7 offer a comparison between the predictions generated by an RBF Gaussian Regression model and the actual ground truth. In contrast to the previous two models, our analysis leads to distinct conclusions. Notably, the predictive variance is at its highest not in regions far from the training barycenter. This is because the mean encompasses the entire range of data, resulting in predictive variance being concentrated solely within this zone. Beyond this range, predictive variance diminishes, which we explain in the next question. The predictive variance demonstrates a fluctuating pattern, marked by peaks and troughs corresponding to regions where the density of training points varies. Consequently, the model closely adheres to the mean of the training data, which is evident in its impact on the behavior of the sinusoidal curves.

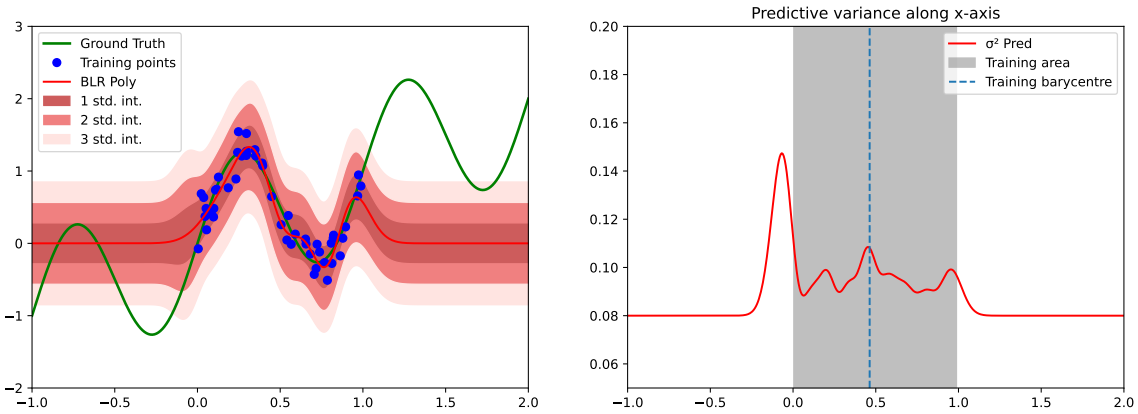


Figure 1.6: Visualization of predictive distribution of a sinusoidal dataset using RBF Gaussian Regression, where we set $\mu \in [0.0, 1.0]$ and $M = 9$.

When we visualize our model with a reduced number of data points, it reinforces what has been discussed and emphasizes the role of hyperparameters. Notably, the model struggles to capture the information provided by the data points at the extremes, as they lie outside the range of the specified mean. Nonetheless, the predictive variance exhibits a significant increase as anticipated in regions where no data points are present.

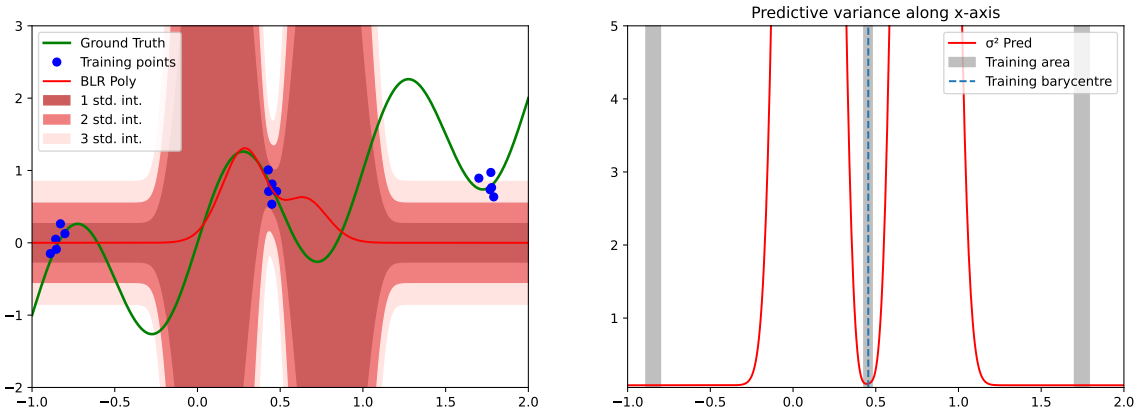


Figure 1.7: Visualization of predictive distribution of a sinusoidal dataset featuring "holes" and sparse data points using RBF Gaussian Regression, where we set $\mu \in [0.0, 1.0]$ and $M = 9$.

2.5. Explain why in regions far from training distribution, the predictive variance converges to this value when using localized basis functions such as Gaussians. Let us recall the definition of the Radial Basis Function:

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

where μ_j represents the center of the j-th Gaussian basis function and s is a parameter controlling the spread of the Gaussian.

The primary observation is that this Gaussian function reaches its maximum at its center, where $x = \mu_j$ since $(x - \mu_j)^2 = 0$. As the distance $(x - \mu_j)^2$ increases, the exponential term rapidly tends toward zero. So, if x^* is far from μ_j, we can approximate $\Phi_j(x^*) \approx 0$. As a result, when we multiply this vector by the covariance matrix and its transpose, the epistemic uncertainty converges toward zero. Therefore, $\sigma^2_{\text{pred}} = \beta^{-1} + \Phi(x^*)^T \Sigma \Phi(x^*) \approx \beta^{-1}$, signifying that the predictive variance is effectively reduced to aleatoric uncertainty. This can be observed in Figures 1.6 and 1.7, where $\sigma^2_{\text{pred}} = \beta^{-1} = 0.08$.

Consequently, μ and M are two critical hyperparameters (with s derived from them), and their selection should be based on our data. For instance, setting $\mu \in [-2, 2]$ without changing M yields significantly improved results, as demonstrated in Figures 1.8 and 1.9. We observe that these outcomes outperform the polynomial approach while maintaining similarity, thereby highlighting the robustness and popularity of the RBF kernel.

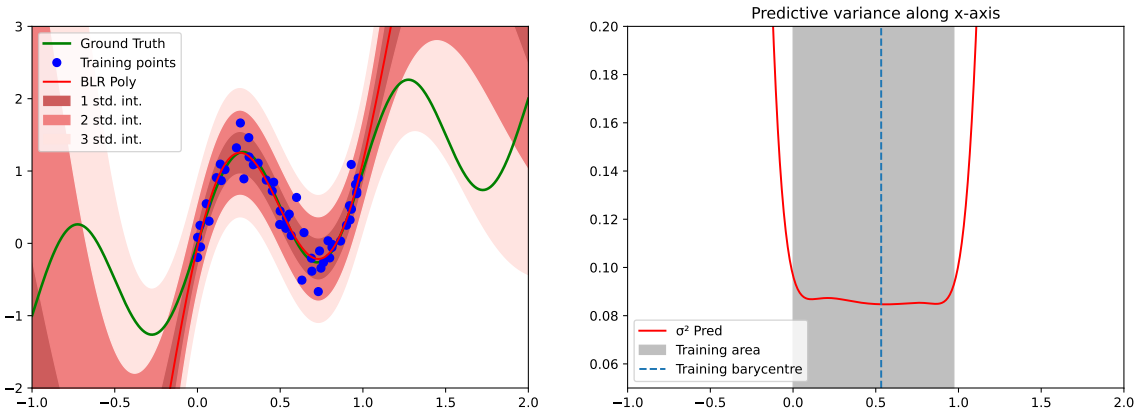


Figure 1.8: Visualization of predictive distribution of a sinusoidal dataset using RBF Gaussian Regression, where we set $\mu \in [-2.0, 2.0]$ and $M = 9$.

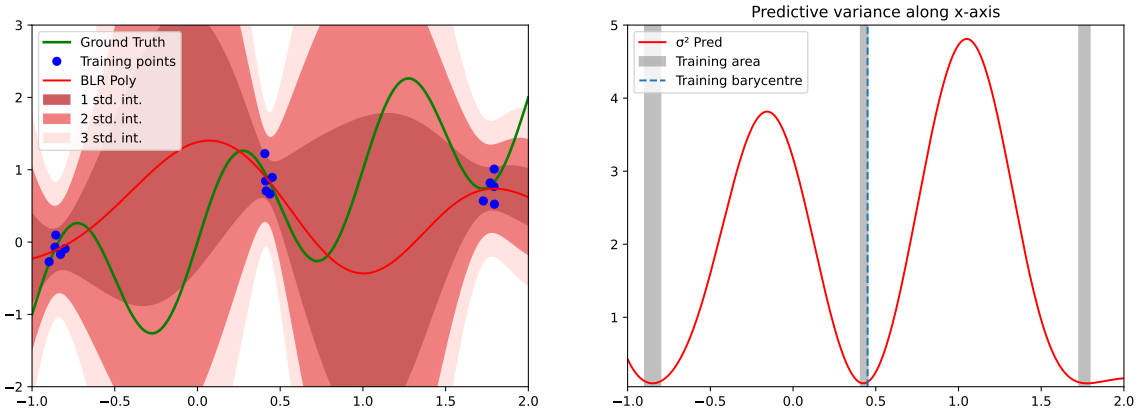


Figure 1.9: Visualization of predictive distribution of a sinusoidal dataset using RBF Gaussian Regression, where we set $\mu \in [-2.0, 2.0]$ and $M = 9$.

Chapter 2

Approximate Inference in Classification

This chapter extends the exploration of Bayesian Deep Learning. Building upon the previous chapter, it delves into the complexities of classification tasks. The focus here is on the challenges associated with these tasks, particularly the lack of a closed-form solution for the posterior distribution in Logistic Regression. The chapter conducts a comparative analysis of different approximate inference techniques, including Laplacian approximation and variational inference, applied to binary classification problems. This exploration is crucial for gaining insights into the application of Bayesian methods in more complex machine learning scenarios, representing a substantial advancement from the linear regression models discussed earlier.

2.1 Bayesian Logistic Regression

In this section, our initial focus is on Bayesian Logistic Regression. We begin by highlighting he difference between the continuous predictions found in linear regression models and the discrete class label predictions characteristic of classification models. Specifically, we introduce binary classification through logistic regression and explore several methods for approximating the posterior distribution, as the posterior distribution is no longer tractable due to the absence of conjugacy between the likelihood and the prior.

2.1.1 Maximum-A-Posteriori Estimate

1.1. Analyze the results provided by Figure 2.1. Looking at $p(y = 1|x, \mathbf{w_{MAP}})$, what can you say about points far from train distribution? Approximating $p(\mathbf{w}|\mathbf{X}, \mathbf{Y})$ with a Dirac delta function is essentially akin to approximating the predictive distribution using $\mathbf{w}_{\mathrm{MAP}}$, meaning $p(y = 1|x, \mathbf{w}_{\mathrm{MAP}}) \approx p(y = 1|x, \mathbf{Y})$. This approximation is quite straightforward.

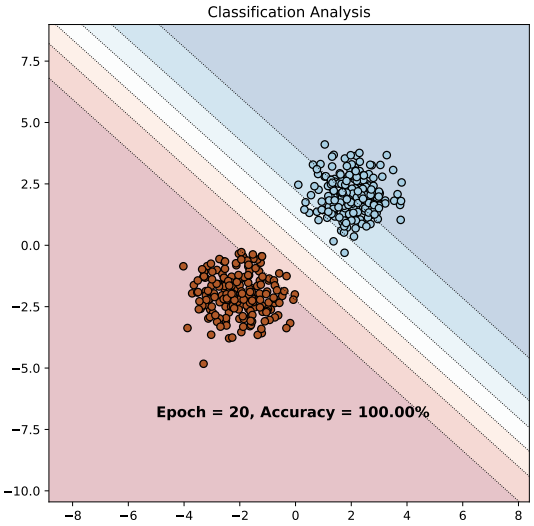


Figure 2.1: Illustration of a Bayesian Logistic Regression model applied to a binary classification task with uncertainty display, with a weight decay of 5×10^{-2}. Two distinct data point clusters — blue and red — represent separate classes, while the surrounding shaded areas reflect the model's predictive uncertainty, with lighter shades indicating lower confidence.

As depicted in Figure 2.1, the boundary decision remains linear and the model's uncertainty doesn't significantly increase far from the training data. Essentially, it provides a confidence measure for the linear boundary. This indicates that the point-wise estimate of the parameters can only confidently assign points to their respective classes but lacks the capacity to provide nuanced uncertainty measures for points that deviate far from the training data distribution. Thus, this approach is not effective in assessing the uncertainty of outliers or points not well-represented in the training set.

2.1.2 Laplace Approximation

1.2. Analyze the results provided by Figure 2.2. Compared to previous MAP estimate, how does the predictive distribution behave? Compared to the MAP estimate, Bayesian Logistic Regression using the Laplace approximation better captures uncertainty about the model parameters. While the MAP estimate gives a single point estimate of the weights ($\mathbf{w}_{\text{MAP}}$) and therefore a single decision boundary, the Laplace approximation treats the weights as a normal distribution. This distribution is centered around $\mathbf{w}_{\text{MAP}}$ and has a covariance matrix based on the Hessian of the log posterior. This approach allows for uncertainty in the decision boundary, as clearly shown in Figure 2.2.

For instance, moving in the southwest direction away from the red cluster — directly opposite the blue cluster — the model exhibits high confidence that this region predominantly consists of red points. The same holds true for the blue cluster. This directional certainty reflects the model's aleatoric uncertainty, which is the irreducible uncertainty inherent in the observations due to noise or other stochastic effects. Conversely, if we move sideways, out of the line between the clusters to areas with less or no data, the model becomes less certain. This reflects the epistemic uncertainty of our model, relating to what the model doesn't know about its own parameters.

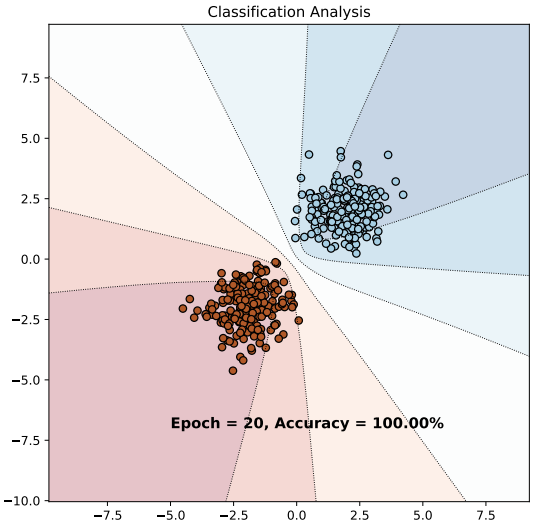
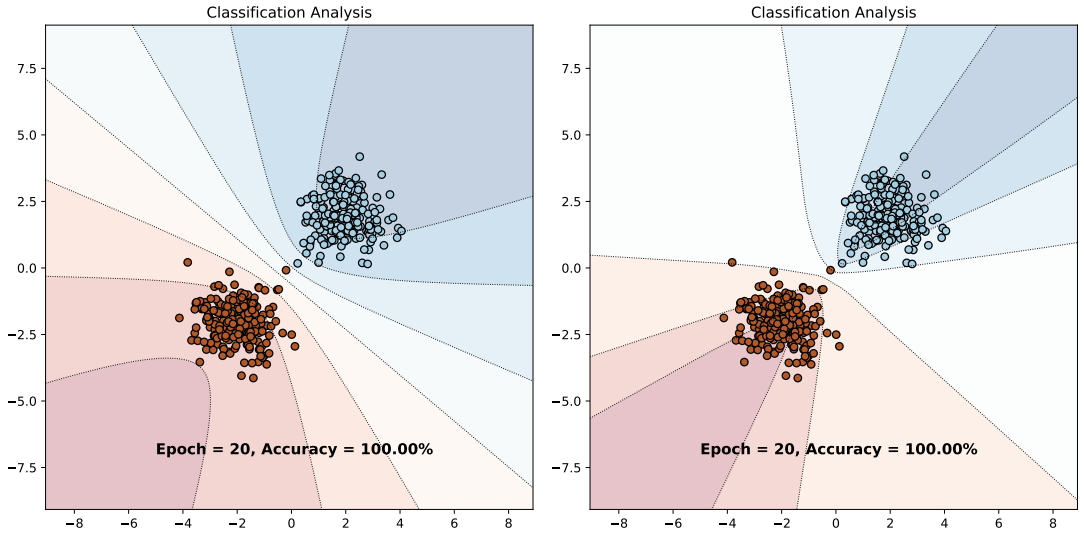


Figure 2.2: Illustration of a Bayesian Logistic Regression with Laplace approximation model applied to a binary classification task, with a weight decay of 5×10^{-2}.

1.3. Comment the effect of the regularisation hyper-parameter `WEIGHT_DECAY`. The weight decay hyper-parameter controls the complexity of the model. It adds a penalty to the loss function for large weights, effectively encouraging the model to maintain smaller weight values, which usually help prevent overfitting. In Bayesian terms, weight decay corresponds to the precision (inverse variance) of the prior distribution over the weights. A higher weight decay value means a tighter prior, which pulls the weights closer to zero, unless the data provides strong evidence to the contrary. This can affect the predictive distribution by potentially making it more conservative. As a result, the decision boundary may be less flexible and the model may exhibit higher uncertainty, especially in regions far from the training data. This behavior is verified in Figure 2.3. When the weight decay is too high, it results in increased predictive uncertainty (wider shaded areas), whereas when it's too low, it results in high confidence (narrower shaded areas), which may not be justified for unseen data.



(a) (b)

Figure 2.3: Illustration of a Bayesian Logistic Regression with Laplace approximation model applied to a binary classification task, with a weight decay of (a) 0.5 and (b) 5×10^{-5}.

2.1.3 Variational Inference

1.4. Comment the code of the `VariationalLogisticRegression` and `LinearVariational` classes. `LinearVariational` represents a single linear layer with variational inference applied. It approximates the weights and biases of the layer with distributions rather than fixed values.

- The class is initialized with the variational parameters for the weights (`w_mu`, `w_rho`) and biases (`b_mu`) of the layer, i.e. the parameters we want to learn. `prio_var` represents the variance of the prior distribution (σ_p^2), which specify our prior belief about the distribution of the weights.
- The `sampling` method uses the reparametrization trick to sample from the variational posterior distribution for the weights $\boldsymbol{w}_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$. To do so, we sample from a centered isotropic multivariate Gaussian where $\sigma^2 = \log(1+e^{\rho})$ to avoid numerical issues. Thus, $\boldsymbol{w}_i = \mu_i + \sigma_i \odot \boldsymbol{\varepsilon}_s$, where $\boldsymbol{\varepsilon}_s \sim \mathcal{N}(0, 1)$ is a Gaussian noise. The reparametrization trick allows the gradient of the loss function to backpropagate through the randomness of the sampling process.
- The `kl_divergence` method calculates the Kullback-Leibler divergence between the variational posterior and the prior distribution for the weights:

$$\mathrm{KL}\left[q_\theta(\boldsymbol{w}) \| p(\boldsymbol{w})\right] = \log\left(\frac{\sigma_p}{\sigma_i}\right) + \frac{\sigma_i^2 + \mu_i^2}{2\sigma_p^2} - \frac{1}{2}$$

  where σ_p^2 is the variance of our prior distribution $p(\boldsymbol{w})$ and (μ_i, σ_i^2) the mean and variance of the variational distribution $q_\theta(\boldsymbol{w})$.
- The `forward` method defines the forward pass by performing a linear transformation, i.e. $w^T x + b$. We sample the weights then compute the output of the layer using the sampled weights and the mean of the biases.

`VariationalLogisticRegression` represents a logistic regression model using variational inference:

- The class is initialized with one linear variational layer used to perform the linear transformation in logistic regression.
- The `forward` method defines the forward pass for the logistic regression model by returning $f(x) = \sigma(w^T x + b)$ where σ is the sigmoid function.
- The `kl_divergence` method simply calls the same method of the `LinearVariational` layer to obtain the KL divergence term for the loss computation.

1.5. Comment the code of the training loop, especially the loss computation. Analyze the results provided by Figure 2.4. Compared to previous MAP estimate, how does the predictive distribution behave? What is the main difference between the Variational approximation and the Laplace approximation? The training loop uses a standard PyTorch format. The loss function calculates the Evidence Lower Bound (ELBO), which we want to maximize. In theory, we aim to maximize the likelihood of the data directly, but this is often intractable due to the integral over the weights. Therefore, we compute the Kullback-Leibler divergence $\mathrm{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{w})\|p(\boldsymbol{w}))$ between the variational distribution $q_\theta(\boldsymbol{w})$ and the prior distribution $p(\boldsymbol{w})$. This acts as a regularization term, encouraging the variational distribution to be similar to the prior distribution. It represents the information lost when using $q_\theta(\boldsymbol{w})$ to approximate $p(\boldsymbol{w})$, which we want to minimize.

To ensure the model fits the data effectively, we compute the negative log-likelihood $NLL(\boldsymbol{\theta};\mathcal{D})$ of the data under the model parameterized by the weights sampled from $q_\theta(w)$. This is done using a binary cross-entropy loss. Subsequently, as maximizing ELBO is equivalent to minimizing:

$$\mathcal{L}_{\mathrm{VI}}(\boldsymbol{\theta};\mathcal{D}) = NLL(\boldsymbol{\theta};\mathcal{D}) + \mathrm{KL}(q_{\boldsymbol{\theta}}(\boldsymbol{w})||p(\boldsymbol{w})),$$

we employ gradient descent to update the parameters of the variational distribution to better approximate the true posterior.

The results of this variational approximation yield decision boundaries that are less rounded, falling between Maximum a Posteriori and Laplace approximation, as visualized in Figure 2.4. It provides a better encompassing of noisy data points, and all training points fall within regions of high confidence. Consequently, it yields a decision frontier that is noticeably distinct in the central area while effectively accounting for aleatoric uncertainty.

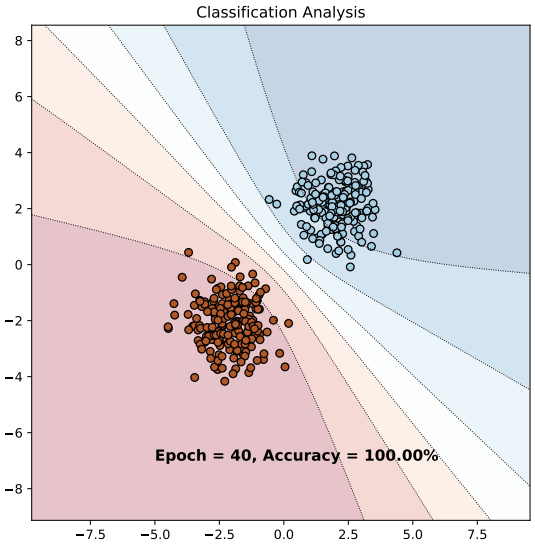


Figure 2.4: Illustration of a Variational Logistic Regression model applied to a binary classification task.

Compared to a Maximum a Poseteriori estimate, the variational approach does not just find the most probable weights (as MAP does) but instead approximates the entire posterior distribution over the weights. In the variational approach, the predictive distribution captures the model's uncertainty about its predictions.

Compared to the Laplace approximation, the variational approximation actively optimizes a parameterized distribution to closely resemble the true posterior, with resemblance quantified by the KL divergence. This optimization typically involves a more complex objective function. On the other hand, Laplace approximation passively fits a Gaussian distribution around the MAP estimate, relying on the curvature of the log-posterior at that point. It assumes that the posterior is locally Gaussian and primarily focuses on finding the MAP estimate and computing the Hessian at that location.

2.2 Bayesian Neural Networks

In this section, we illustrate the extension of Bayesian methods to neural network architecture, with a specific focus on applying variational inference to a Multi-Layer Perceptron. Here, we practically apply the concepts introduced in the previous section to showcase their effectiveness in dealing with complex, non-linear datasets.

2.2.1 Variational Inference with Bayesian Neural Networks

2.1. Analyze the results showed on Figure 2.5. By applying Bayesian principles to a neural network with two hidden layers, we create a model capable of capturing intricate patterns within the data. In this context, each neuron's weight is treated as a random variable, representing our uncertainty about its true value. Consequently, we obtain a complex and non-linear decision boundary. Notably, the shaded regions, indicating the model's predictive uncertainty, reveal that the model demonstrates high confidence near the training data points and experiences increased uncertainty as it moves further away. Interestingly, this behavior leads to the emergence of "clusters" resembling the moon-shaped patterns found in the dataset. Moreover, this approach offers explainability, as it provides outputs that are not just binary but instead resemble the level of confidence one might have when predicting the locations of new data points.

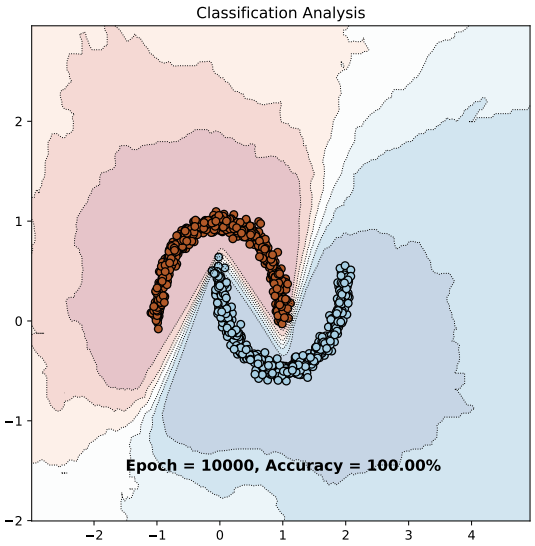


Figure 2.5: Illustration of a Bayesian Neural Network model applied to a binary classification task.

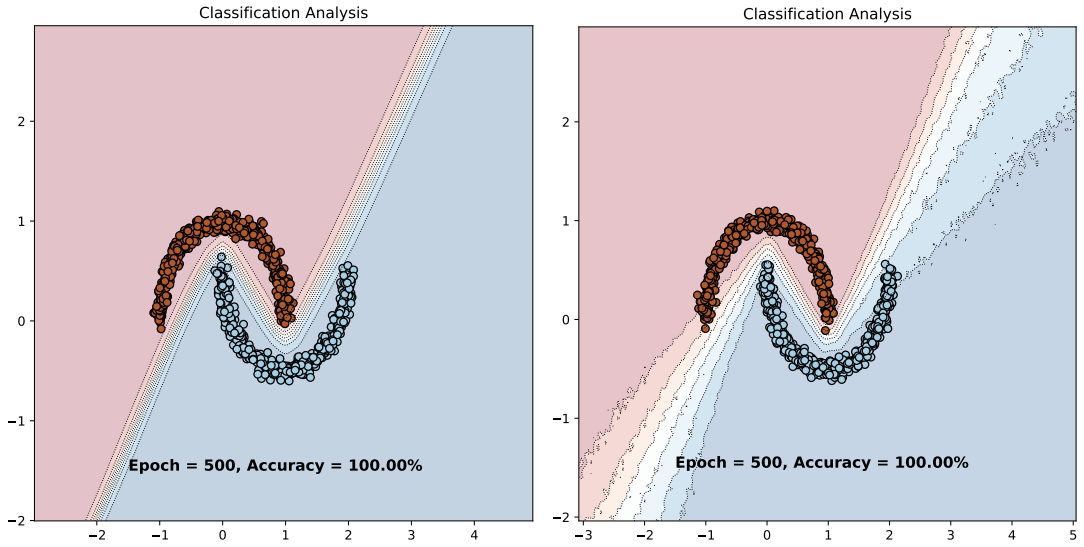
2.2.2 Monte Carlo Dropout

2.2. Again, analyze the results showed on Figure 2.6. What is the benefit of MC Dropout variational inference over Bayesian Logistic Regression with variational inference? Monte Carlo dropout is a technique that aligns with variational inference in Bayesian neural networks, where the dropout mechanism acts as a variational distribution for the network weights. Essentially, dropout introduces Bernoulli random variables, leading to a posterior predictive distribution that accounts for weight uncertainty. The formula for the predictive distribution of an output y for a new input x^* is:

$$p(y|x^*, \mathbf{X}, \mathbf{Y}) \approx \frac{1}{S} \sum_{s \in S} p(y^*|x^*, \mathbf{w}_s),$$

where $\mathbf{w}_s$ represents the network weights after dropout, and S is the number of MC samples or dropout iterations.

The results, shown in Figure 2.6b, include a decision boundary with adjacent bands indicating the model's confidence levels. These bands are formed by applying dropout during inference and averaging results from multiple stochastic passes. This approach illustrates the model's uncertainty in predictions, which contrast to the smooth gradients of deterministic neural networks (shown in Figure 2.6a). The speckled appearance of uncertainty regions in the plot reflects how confidence varies across different input regions, due to the randomness introduced by MC Dropout. Incorporating MC Dropout in a standard neural network effectively turns it into Bayesian-like model, capable of expressing uncertainty in predictions. This approach not only allows the network to learn complex decision boundaries but also provides estimates of uncertainty, something often missing in regular neural networks. This probabilistic interpretation can lead to more informed decisions, as it shows how reliable the network's predictions are across different input areas.



(a) (b)

Figure 2.6: Illustration of a Bayesian Neural Network model applied to a binary classification task using (a) dropout and (b) Monte-Carlo dropout.

Chapter 3

Uncertainty Applications

In this chapter, we address a significant concern in neural networks: their tendency to exhibit overconfidence in predictions. We tackle this issue by exploring uncertainty estimation techniques, notably highlighting the capabilities of Monte-Carlo Dropout variational inference, which was introduced in the previous chapter. These methods play a crucial role in mitigating overconfidence, particularly in scenarios where accurate uncertainty estimation is essential, such as failure prediction and the identification of out-of-distribution instances. This concluding chapter aims to offer a deeper comprehension and practical experience in understanding how Bayesian methods can assist us in effectively managing neural network overconfidence.

3.1 Monte-Carlo Dropout on MNIST

In this section, we focus on training a model using Monte Carlo Dropout (MC Dropout) on the MNIST dataset and analysing the behaviour of the outputed probabilities for certain and uncertain samples. To find out the most uncertain images, we used the variation ratio metric, which effectively measures epistemic uncertainty and is straightforward to calculate. For a given image, denoted as $\mathbf{x}$, we perform T stochastic forward passes through the model and record the predicted labels. We then determine the frequency $f_{\mathbf{x}}^{c^\star}$ of the most common label ($c^\star$) across the T passes. The variation ratio for image $\mathbf{x}$ is calculated using the formula:

$$\text{var-ratio}[\mathbf{x}] = 1 - \frac{f_{\mathbf{x}}^{c^\star}}{T}.$$

This formula provides a quantitative measure of uncertainty for an image.

1.1. What can you say about the images themselfs? How do the histograms along them helps to explain failure cases? Finally, how do probabilities distribution of random images compare to the previous top uncertain images? In this experiment, we utilized a LeNet-5 style model with Monte-Carlo dropout for variational inference, training it on the MNIST dataset for 20 epochs using standard cross-entropy. We then applied the model to calculate variation ratios for each test image, enabling us to sort images based on their uncertainty. These images are displayed in Figure 3.1, accompanied by five types of measurements derived from the model's probability outputs. To summarize how the output probabilities fluctuate across $T = 100$ stochastic forward passes, we used histograms depicting three different distributions:

1. The first column shows the distribution of the mean output probability for each class.
2. The second column displays the distribution of the predicted class across the T forward passes.
3. The last three columns present the distribution of the output probability for specific classes (the most predicted class for the third column, the ground truth class for the fourth, and a different class for the fifth).

These histograms illustrate the variability of output probabilities across the $T = 100$ iterations. A model lacking confidence in its prediction will show this through greater variation in output probabilities between draws, resulting in more dispersed histograms and changes in the predicted class. Conversely, a confident model will have much more concentrated histograms, with the predicted class remaining the same across all iterations.

Figure 3.1 displays images deemed certain by the model. To the human eye, these images clearly belong to their assigned classes. The distributions here are characterized by a single peak, indicating the same value is drawn repeatedly. The mean probabilities for the predicted class equal one, and all other classes consistently have a zero probability, suggesting the model is very confident in its predictions.

Conversely, Figure 3.2 shows images classified as uncertain by the model. These images appear more ambiguous, and even to the human eye, it can be challenging to determine their actual numbers. The distributions here are more spread out. This implies various output class neurons may be activated, either simultaneously or independently at various levels in each run. To gain a deeper understanding of this phenomenon, we examine the distribution of the predicted class along with the mean probabilities. The similarity between these two histograms suggests that the model may select a certain class as the prediction even when the average probability for that class is relatively low.

We explored two possible explanations for this:

- A small mean doesn't necessarily imply the class neuron never reaches high values. Indeed, the last column shows that "small mean" classes can occasionally reach values around 0.8, increasing their likelihood of being predicted.
- The class neuron may sometimes record low values, but it is still the highest compared to other class neurons.

To verify these hypotheses, we plotted the Maximum Class Probability (MCP) in Figure 3.3. MCP, which is the probability used for making the prediction, is defined as $MCP(\boldsymbol{x}) = \max_{k \in \mathcal{Y}} p(Y = k|\boldsymbol{w}, \boldsymbol{x})$ and serves as a basic confidence baseline. Figure 3.3a reveals that most predictions for the image were based on high probability, even though it's often misclassified and considered one of the most uncertain images in the dataset. This highlights the model's overconfidence. However, sometimes a decision is made at a lower probability, such as below 0.4. The subsequent plot in Figure 3.3b helps identify which class the model tends to predict when the MCP is either low or high. For instance, class 8 is consistently predicted with a high MCP, whereas class 7 is always associated with a low MCP.

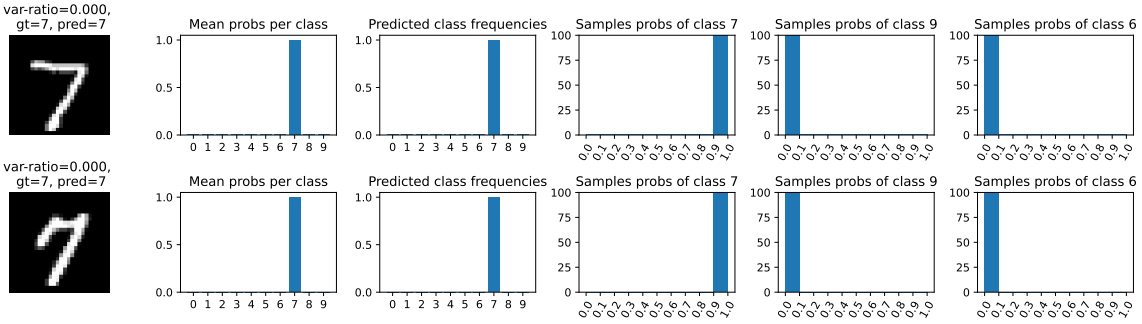


Figure 3.1: **Certain Images with Output Probability Distributions.** Images that the model classified with high certainty, featuring clear, identifiable digits. Histograms show highly concentrated distributions, with mean probabilities for the predicted class at one and other classes at zero, indicating strong model confidence in its predictions.

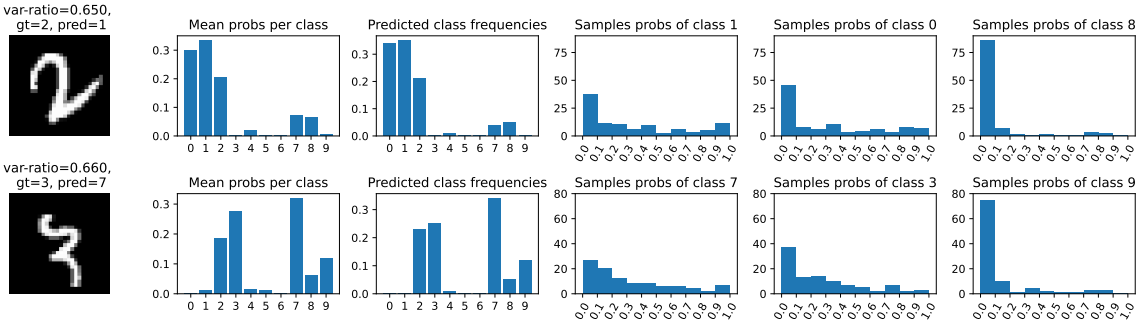


Figure 3.2: **Uncertain Images with Output Probability Distributions.** Ambiguous images that the model classified with uncertainty. Distributions are spread out, suggesting fluctuating confidence across different classes. The similarity between histograms of the predicted class and mean probabilities indicates that the model sometimes chooses a class with a relatively low average probability.

3.2 Failure prediction

In this section, we have conducted a comparison of various methods aimed at obtaining a reliable confidence measure for model predictions. Such a measure permit to distinguish correct and incorrect prediction and so identify instances where a machine learning model does not perform as expected or fails to make accurate predictions. Failure in a machine learning context can occur due to various reasons, such as poor model training, noisy or out of distribution data, inadequate feature representation, overfitting or underfitting. An intelligent decision system equipped with such metrics in operational settings can make informed choices, including adhering to the model's prediction or, conversely, involving a human operator, activating a backup system equipped with additional sensors, or triggering an alarm. This field of application is commonly referred to as failure prediction.

During the lecture and in the previous section, we found that Maximum Class Probability (MCP) is not a great metric for failure prediction. It assigns high confidence values to both correct and erroneous predictions because modern models tend to be overconfident, resulting in overlapping distributions between successes and errors. This issue persists even when using temperature scaling calibration.

Alternatively, when the model makes a misclassification, the probability associated with the true class y tends to be lower than the maximum probability, often falling to a low value. This observation leads us to consider the True Class Probability (TCP) as a suitable measure of uncertainty. However, the true class labels y are not available when estimating confidence for test inputs. This motivates the development of ConfidNet, whose primary objective is to directly regress the TCP value from the input image, allowing us to obtain a reliable measure of uncertainty without access to ground truth labels.

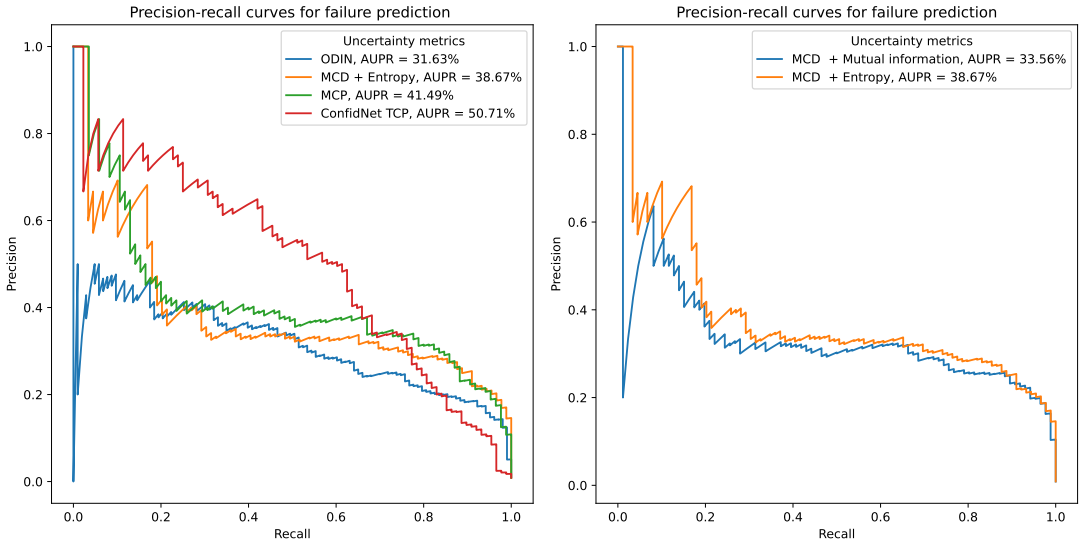
In this practical section, we implemented ConfidNet to address failure predictions and compared it to two other methods that rely solely on the model's output probabilities. The first method is MCP, and the second is the entropy of the output probabilities. For these two methods, we used the previously trained MC Dropout model to compute the output probabilities. ConfidNet was trained for 30 epochs with the previous MC Dropout model as a teacher with frozen parameters and Mean Squared Error as the loss function.

2.1. Compare the precision-recall curves of each method along with their AUPR values. Why did we use AUPR metric instead of standard AUROC? To assess and compare these methods, we require a suitable metric. Our objective is to identify classification errors, which we consider as the positive detection class, while correct predictions serve as the negative detection class. Since our models excel at making accurate predictions, we anticipate a low occurrence of classification errors, resulting in an imbalanced setting with a significant number of true negatives.

We have opted to employ the AUPR (Area Under the Precision-Recall Curve) instead of the AUROC (Area Under the Receiver Operating Characteristic Curve) due to the latter's unsuitability for imbalanced datasets. AUROC treats both classes equally and can yield misleading results, particularly under the influence of a large number of true negatives. This may overstate the model's performance, particularly in distinguishing the minority class, which in this context comprises classification errors. Conversely, AUPR is a more suitable choice, as it prioritizes precision and recall for the minority class. Precision measures the fraction of actual positives among the positive predictions, while recall measures the proportion of correctly identified actual positives. Consequently, AUPR proves to be a more reliable metric in our situation, where the positive class (classification errors) is significantly smaller than the negative class (correct predictions).

Figure 3.4a illustrates the precision-recall curves for each method, accompanied by their respective AUPR values. The results demonstrate that ConfidNet surpasses the other two methods. This superiority can be attributed to ConfidNet's training, which directly estimates the TCP value—a more dependable uncertainty measure compared to MCP and entropy, as elaborated upon in the section introduction. Notably, ConfidNet exhibits a slower decline in precision as recall increases, indicating a more balanced performance in terms of identifying true positives without a substantial rise in false positives.

For a better understanding of the lecture's confidence metrics, we aimed to compare the performance of predictive entropy and mutual information in the context of failure detection. The results are presented in Figure 3.4b. It seems that entropy outperforms mutual information as a metric for failure detection. This distinction arises from the fact that predictive entropy measures aleatoric uncertainty, while mutual information assesses epistemic uncertainty. In the specific experiment of failure detection on MNIST, it proves more advantageous to rely on and measure aleatoric uncertainty, which originates from the natural variability or randomness of the numbers in the images. This explanation underscores why entropy is a superior metric to mutual information in this particular context.



(a) (b)

Figure 3.4: **(a) Precision-Recall Curves for Each Tested Method with AUPR Values.** The graph demonstrates ConfidNet's balanced performance in identifying true positives with fewer false positives, indicating its effectiveness in making accurate predictions with a low occurrence of classification errors.
(b) Performance Comparison of Predictive Entropy and Mutual Information for Failure Detection. The graph illustrates that predictive entropy, measuring aleatoric uncertainty, outperforms mutual information, which measures epistemic uncertainty. This suggests that in scenarios with aleatoric uncertainty, like natural variability in MNIST images, entropy is a more effective metric for failure detection.

3.3 Out-of-distribution detection

In today's machine learning landscape, models are often trained on vast datasets, such as ImageNet, which can reduce epistemic uncertainty. However, in critical real-world applications like autonomous driving, the need to identify out-of-distribution (OOD) inputs remains essential due to the infinite variability of real-life data. OOD detection plays a crucial role because machine learning models typically assume that the data they encounter during real-world predictions will resemble the data they were trained on. When a model confronts data that significantly deviates from its training data, it can result in unpredictable and unreliable predictions. OOD detection aims to identify such situations, allowing for cautious handling or even disregarding of the model's predictions on such atypical data. It's important to note that while OOD data can lead to model failure, not all model failures are solely attributed to OOD data. Model failures can occur for various reasons, including overfitting, underfitting, or architectural issues, even when the data falls within the expected distribution. Therefore, while OOD detection is a critical component of ensuring a model's robustness, it represents just one facet of the broader spectrum of failure detection within machine learning systems.

In this section, we will find out what's the best methods for OOD detection. We will use the Kuzushiji-MNIST (KMNIST) dataset as out-of-distribution data for our MC dropout MNIST predictor. This dataset consists of 70,000 28x28 grayscale number images. We will evaluate the performance using precision, recall, and AUPR as metrics to compare different methods.

Specifically, in this section, we have implemented the ODIN method (Liang et al., 2017), which enhances maximum softmax probabilities with temperature scaling and inverse adversarial perturbation. These techniques are employed to increase the distinction between in-distribution and out-of-distribution data.
Temperature scaling involves applying a simple scaling of the logit by a temperature parameter $1/T$ before the softmax: $S_i(\boldsymbol{x}, T) = \frac{\exp(f_i(\boldsymbol{x})/T)}{\sum_{j=1}^{N} \exp(f_j(\boldsymbol{x}/T))}$, where S_i represents the output probabilities, $\boldsymbol{x}$ is the input image, T is the temperature parameter, and f_i is the logit of the i-th class.
Inverse adversarial perturbation is used to preprocess the input $\boldsymbol{x}$ before feeding it to the neural network. This preprocessing involves adding a small perturbation to the input image $\boldsymbol{x}$ such that $\boldsymbol{x}' = \boldsymbol{x} - \epsilon \cdot \text{sign}(-\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, y))$, where $\mathcal{L}$ is the cross-entropy loss function, y is the ground truth label, ϵ is the perturbation magnitude, and $\boldsymbol{x}'$ is the perturbed image. The perturbation magnitude ϵ is chosen to ensure the perturbed image remains correctly classified by the neural network.

3.1. Compare the precision-recall curves of each OOD method along with their AUPR values. Which method perform best and why? Figure 3.5 displays the precision-recall curves for six uncertainty metrics (MCP, ODIN, CondifNet TCP, MC Dropout (MCD) mutual information, and MCD predictive entropy), along with their corresponding AUPR values. Given the smoothness of these curves, our analysis will primarily focus on the AUPR values.

ODIN, with its two modifications to output probabilities, performs slightly better than MCP, which aligns with expectations since ODIN builds upon MCP. However, ODIN is surprisingly surpassed by the other two MCD methods. This outcome was not anticipated for mutual information and predictive entropy since they, like ODIN, depend on the raw output probabilities. These methods appear to benefit from the stochastic nature of MC Dropout. In fact, both methods achieved impressive AUPR scores of 98%, ranking as the most effective metrics for OOD detection. In our previous experiment, although ConfidNet emerged as the top method for detecting failures, it showed a marginal lag, achieving an AUPR of 96%. This can be readily understood when considering that there is no TCP available for out-of-distribution data. Consequently, ConfidNet lacks specific data to predict or train on in these scenarios. Mutual information and predictive entropy, despite their simplicity, emerge as leading metrics for OOD detection in our experiments, but at the expense of a more complex (Bayesian) network. For instance, computing the entropy score takes 18 seconds, whereas ODIN requires only 4.1 seconds.

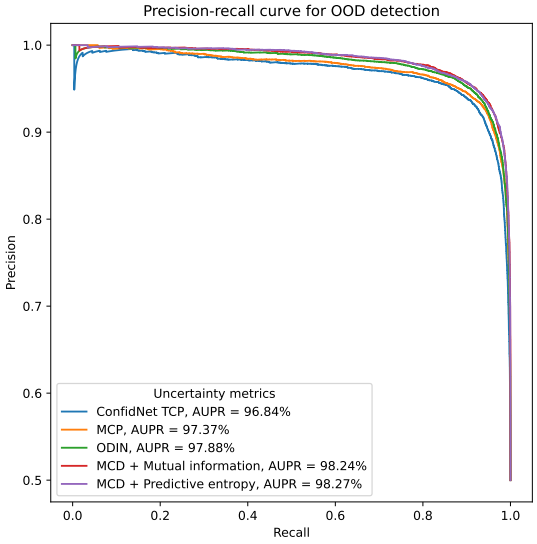
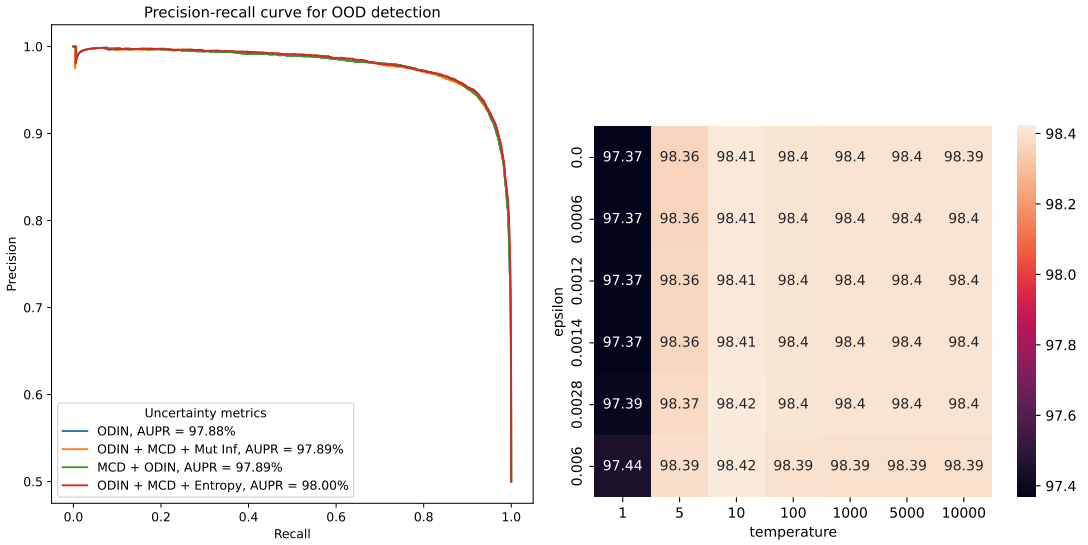


Figure 3.5: **Precision-Recall Curves and AUPR Values for Six Uncertainty Metrics.** Precision-recall curves for six uncertainty metrics (MCP, ODIN, ConfidNet TCP, MC Dropout (MCD) mutual information, and MCD predictive entropy) with their corresponding Area Under the Precision-Recall curve (AUPR) values. MCD mutual information and predictive entropy lead with AUPR scores of 98%. ODIN performs slightly better than MCP but is outperformed by MCD methods, which take advantage of Bayesian networks. However, note that the performance of MCD methods comes at a computational cost.

To further investigate ODIN's techniques, we attempted to combine them with MC Dropout (averaging over 100 stochastic forward passes), predictive entropy, and mutual information. The outcomes are presented in Figure 3.6a. This combination actually resulted in a slight performance decline for Entropy and Mutual Information, with a decrease of approximatly 0.3% in their scores. Integrating MCD with ODIN appears to have no significant impact.

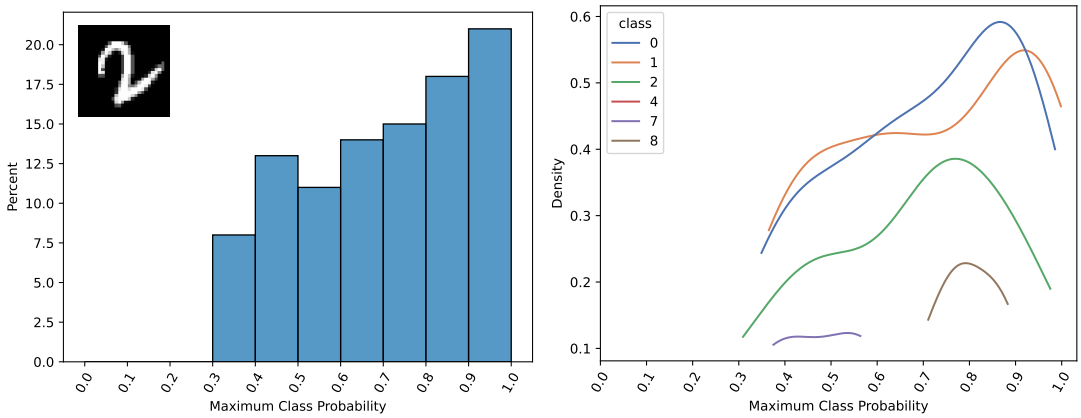


(a) (b)

Figure 3.6: **(a) AUPR Values for Combined ODIN and MC Dropout Techniques.** Analysis of the combined impact of ODIN techniques with MC Dropout (averaging over 100 stochastic forward passes), predictive entropy, and mutual information on AUPR values. The combination slightly reduces the performance of Entropy and Mutual Information by about 0.3%, with no significant impact on ODIN's performance.
(b) Grid Search Results of ODIN's AUPR Values Relative to Hyperparameters ϵ and Temperature. Grid search results showing the influence of ODIN's hyperparameters ϵ and temperature on its AUPR values. The best result is achieved with a temperature of 10, yielding a 98.42% AUPR, surpassing MCD with entropy. Temperature scaling alone enhances AUPR by 1%, while perturbation has a negligible impact, suggesting temperature scaling as the more effective parameter.

Considering ODIN's persistent underperformance, we hypothesized it might be due to suboptimal hyperparameters. The code we used suggested values for ϵ of 0.006 and 0.0006, but also recommended trying $\epsilon = 0.0014$ at the same time. Moreover, the original paper sometimes employed high temperature values, up to 10000. This led us to conduct a grid search, the results of which, showing AUPR values in relation to ϵ and temperature, are depicted in Figure 3.6b.

The overall influence of these parameters can be noticeable, with a maximum variation of 1% between the highest and lowest scores. Setting the temperature to 10 yielded the best result, achieving a 98.42% AUPR, surpassing MCD with entropy which scored 98.25%. Setting the temperature to 1 essentially disables temperature scaling, allowing us to isolate the effect of perturbation, and vice versa for $\epsilon = 0$. Hence, the first cell serves as a baseline for comparison with other cells. The first line indicates that temperature scaling alone can enhance AUPR by 1%, a modest but significant improvement given the close scores. However, examining the first column reveals no discernible impact ($< 0.08\%$) of perturbation at any ϵ value, suggesting that combining both parameters achieves similar results to using temperature scaling alone. As the provided code uses a smaller epsilon for OOD data, we also replicated the grid search with this adjustment, which led to analogous findings.



(a) (b)

Figure 3.3: **(a) Maximum Class Probability (MCP) Bar Plot.** Bar plot showcasing the Maximum Class Probability (MCP) for uncertain images. It reveals that most predictions are based on high probability, pointing to model overconfidence. However, some decisions are made at lower probabilities, suggesting inconsistency in the model's confidence level.
(b) Density plot of MCP in function of the class. KDE plot illustrating the distribution of MCP for different preditected classes, indicating the model's tendency to predict certain classes with high MCP (e.g., class 8) and others with consistently low MCP (e.g., class 7), reflecting the model's varying confidence across different classes.

Bibliography

Shiyu Liang, Yixuan Li, and R. Srikant. Principled detection of out-of-distribution examples in neural networks. *CoRR*, abs/1706.02690, 2017. URL http://arxiv.org/abs/1706.02690.