Exploring Probability Concepts in Bayesian Neural Networks

Final Project Report
Kevin Thomas

1 Introduction

Neural networks are a class of machine learning models inspired by the structure and function of the human brain. They consist of layers of interconnected nodes (neurons) that process data and learn patterns through optimization. Traditional neural networks are deterministic, meaning they provide a single prediction for a given input by learning fixed weights during training. Bayesian Neural Networks (BNNs) extend traditional neural networks by incorporating probabilistic reasoning. Instead of treating the network weights as fixed values, BNNs treat them as random variables with associated probability distributions. This enables BNNs to quantify uncertainty in their predictions, making them particularly valuable in applications like medical diagnosis. A good background can be found in this paper[4] and this YouTube Video[1].

2 How Bayesian Neural Networks Work

Using Bayesian inference, BNNs use observed data to update prior beliefs into posterior distributions. The output of a BNN is a distribution of predictions, reflecting both model uncertainty (e.g., due to limited data) and inherent variability in the data. While computationally more intensive, BNNs are more uncertainty-aware in applications that require it.

2.1 How a Typical Network Works

BNNs usually have an input, hidden, and output layers. The number of neurons in the input and output layers depend on the problem being solved. The number of neurons in the hidden layer depends on the complexity of the problem[4]. The neurons are interconnected by weights. A example of a typical neural net is shown in figure 1.

2.1.1 Training Phase

- Weights between layers (e.g., Input to Hidden, Hidden to Output) are initialized as probability distributions (e.g., Gaussian, with mean μ and

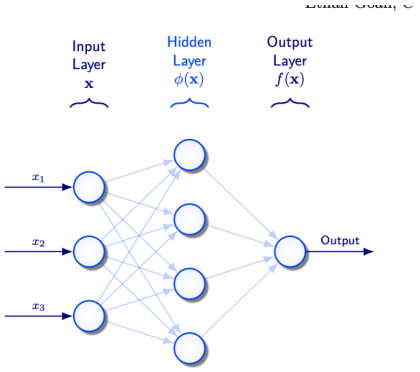

Figure 1: Typical Neural Network

variance σ^2).

- For each forward pass, weights are sampled from their distributions. The sampled weights are used to compute the weighted sum and activations at each layer.
- Loss is calculated based on the expected value of the output
- Backpropagation and gradient descent is used to update the weight distribution parameters (μ and σ^2)to better fit the data and encode appropriate uncertainty. This is repeated will the loss is low or some epochs are reached.

2.1.2 Testing Phase

- The trained weight distributions are used. For each forward pass, weights are sampled from the learned posterior distributions.
- The input is propagated through the network, computing activations using the sampled weights. This produces an output distribution for the final prediction.
- The final prediction is provided as a distribution, capturing both the predicted value and uncertainty

3 Probability Concepts in BNN - 391 Calculation

Several concepts in probability are used in BNNs. They are used either in the functioning of the network or are exhibited in the output behavior. The following subsections discuss the various probability concepts we studied in class that can be seen or used in a BNN.

3.1 Bayes Theorem - L4

Bayes' theorem in the context of a Bayesian Neural Network (BNN) describes how to update the belief about the weights $\mathbf{w}$ after observing data $\mathcal{D}$. It is expressed as:

$$P(\mathbf{w} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \mathbf{w})P(\mathbf{w})}{P(\mathcal{D})}$$

where:
- $P(\mathbf{w})$ is the prior distribution over the weights, e.g., $w_{ij} \sim \mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$
- $P(\mathcal{D} \mid \mathbf{w})$ is the likelihood, the probability of the observed data given the weights
- $P(\mathbf{w} \mid \mathcal{D})$ is the posterior distribution, reflecting the updated belief about the weights after observing the data
- $P(\mathcal{D})$ is the evidence, a normalization constant ensuring the posterior is a valid probability distribution.

3.2 Expectation and Variance - L9

The weights of the network are treated as random variables with associated probability distributions. The expectation and variance of these weights represent the model's belief about the most likely values for the weights and the uncertainty in those beliefs, respectively.

- Initially, the weights are assigned a prior distribution, reflecting the network's belief about their values before seeing any data or randomly. Each weight w_{ij} is modeled as a Gaussian distribution:

  $$w_{ij} \sim \mathcal{N}(\mu_{ij}, \sigma_{ij}^2),$$

  where:
  - μ_{ij} is the mean (expected value) of the weight.
  - σ_{ij}^2 is the variance (uncertainty) of the weight, and σ_{ij} is its standard deviation.

- After observing data, the BNN updates its belief about the weights using Bayes' theorem, resulting in a posterior distribution.

  While training the network the error (or loss $\mathcal{L}$) is calculated as a function of the expectation $\mathbb{E}_{q(\mathbf{w})}[\log P(\mathcal{D} \mid \mathbf{w})]$.

  This $\mathcal{L}$ shifts the expectation of the weights toward values that better explain the data. The variance of the weights decreases in regions where the data strongly supports specific values, reflecting increased confidence, and remains larger in regions with less informative data, capturing uncertainty.

  The mean is updated using gradient descent

  $$\mu_{ij} \leftarrow \mu_{ij} - \eta \frac{\partial \mathcal{L}}{\partial \mu_{ij}} \quad \text{where } \eta \text{ is the learning rate}$$

The standard deviation σ_{ij} is typically updated indirectly by optimizing $\log \sigma_{ij}$ for numerical stability

$$\log \sigma_{ij} \leftarrow \log \sigma_{ij} - \eta \frac{\partial \mathcal{L}}{\partial \log \sigma_{ij}}$$

3.3 Chebyshev Bound - L21

Chebyshev's inequality is a probabilistic bound that applies to any random variable with finite mean and variance. This inequality is useful for understanding the spread of a distribution and provides a conservative bound on the probability of extreme deviations.
The Chebyshev bound

$$P(|X - \mu| \geq c) \leq \frac{\sigma^2}{c^2}$$

where μ and σ^2 are mean and variance respectively[3]

3.4 Law of Large Numbers - L22

The Law of Large Numbers (LLN) states that as the size of a sample increases, the sample mean converges to the expected value of the underlying random variable[3]. The LLN principle can be validated here in the context of a BNN.

$$\lim_{n \to \infty} P\left(\left|\frac{S_n}{n} - \mu\right| < \varepsilon\right) = 1$$

3.5 Central Limit Theorem - L23

Suppose $X_1, X_2, \ldots$ are independent identically distributed random variables with finite expectation $\mathbb{E}[X_i] = \mu$ and finite variance $\text{Var}[X_i] = \sigma^2 \neq 0$. As $n \to \infty$, the standardized sample mean

$$\frac{\sqrt{n}}{\sigma}\left(\frac{X_1 + \cdots + X_n}{n} - \mu\right) \xrightarrow{d} Z \in N_{\mathbb{R}}(0, 1)$$

converges in distribution to a standard normal distribution. We should be able to see this normal distribution of the posterior predicted values from the BNN.

4 About the Simulation - Pyro

The basis for this code is from the Pyro tutorial[2]. The code from this tutorial was modifed to study the various concepts of this project.

4.1 About the problem

The example problem of learning a nonlinear curve from noisy observations using a probabilistic framework is used. The task involves generating synthetic data points that follow a predefined curve (e.g., a sinusoidal or polynomial pattern) with added noise to simulate real-world uncertainties. By modeling the relationship between the inputs and noisy outputs, the goal is to infer the true underlying curve while quantifying uncertainty in the predictions. One example of a curve that is learnt by the network is shown in figure 2.

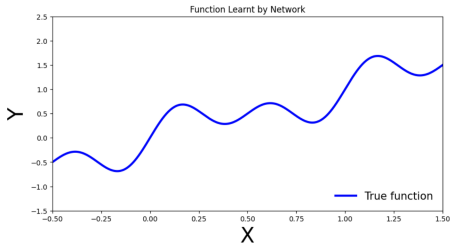

Figure 2: Function the Network Learns

4.2 Network Used

The neural network used has 3 layers as show in figure 3.

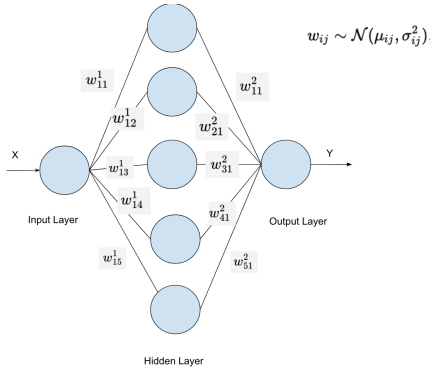

Figure 3: Network Used in Experiment

The input and output layers have a single neuron because the problem is to learn the points on a curve - we are given the X value (one neuron) and we are asked to predict the Y value (one neuron). The hidden layer can have any number of neurons depending on the complexity of the problem. In this network, 5 neurons have be used in the hidden layer.

4.3 Noise

The code generates noise as a perturbation to the input data, reflecting real-world uncertainties. Different types of noise distributions—normal, uniform, and exponential—can be selected to simulate various scenarios of randomness.

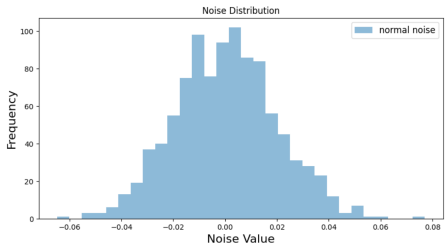

Figure 4: Noise added to input data

Figure 4 shows a visualization of the chosen noise distribution which directly influences the model's observations and predictions. The noise distribution is crucial in Bayesian modeling, as it impacts the likelihood and ultimately the posterior inference.

The noise determines how much influence the actual data has in updating the prior into the posterior. If the noise is large, the model is less confident about the observed data, leading to a broader posterior distribution. Conversely, if the noise is small, the posterior is narrower and more sharply aligned with the likelihood, reflecting higher confidence in the data.

Bayesian model predicts a distribution of possible y-values (posterior predictive distribution) for each input rather than a single deterministic output. Outputs at chosen x values is shown in figure 5

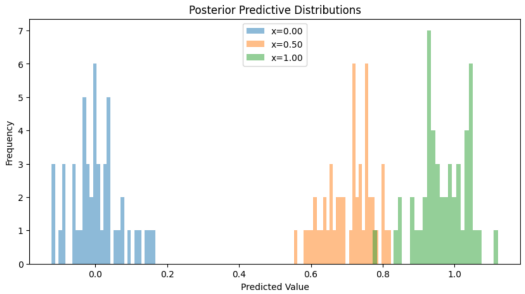

Figure 5: Posterior Distribution of Prediction

5 Probability Concepts

Some of the probability concepts enumerated in section 3 are illustrated using simulation.

5.1 Prior and Posterior Probabilities - Demonstration of Bayes

The prior and posterior distributions in this Bayesian framework represent the belief about the input-output relationship before and after observing data. Initially, a uniform prior assigns equal probability to all input values due to lack of prior knowledge or a neutral assumption. x_{obs} are the X values.

$$prior = \frac{1}{\text{len}\left(x_{\text{obs}}\right)}$$

The likelihood, derived from the observed noisy data combines with the prior to compute the posterior distribution using Bayes' theorem.

$$likelihood = e^{\left(-\frac{\left(y_{\text{obs}} - y_{\text{true.mean ()}}\right)^2}{2 \cdot variance}\right)}$$

The uniform prior (shown in blue line) and the posterior probabilities are illustrated in the figure 6. The variation in the probabilities are only in the region where noise was introduced or data was seen. The output demonstrates conditional probability by visualizing the posterior predictive distributions specific input points.

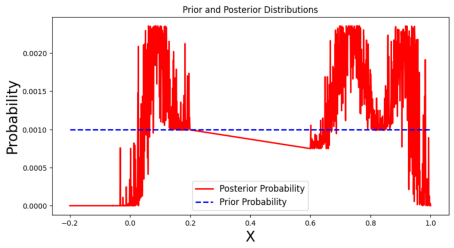

Figure 6: Illustration of Prior and Posterior Probabilities

5.2 Expectation and Variance in BNN

In figure 7, the black dots show the data that was used to train the network. The prediction of the network after training is shown in figure 8. The dark green line shows the predictive mean of the values. The green band shows the variance. Notice that the variance of the prediction on the right side of the image is large meaning the network is not confident in the prediction. This is because there was not enough data in this region during training for the posterior values to

be calculated properly.

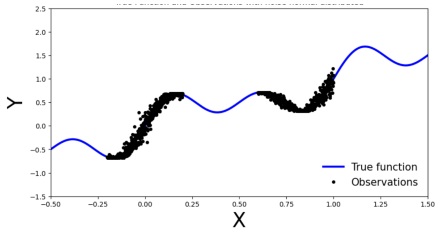

Figure 7: Training Data

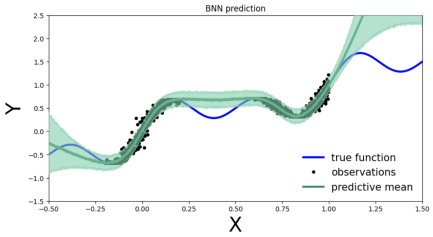

Figure 8: Test Data Prediction

If the training adds a few data points (priors) in this region (figure 9), the variation is reduced a lot (figure 10) and the network predicts much better. If there are more training data added from every curve in the original space, the expectation and variance will be predicted much better.

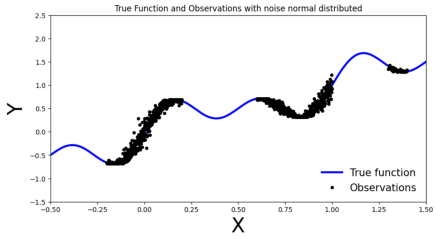

Figure 9: Added More Training Data

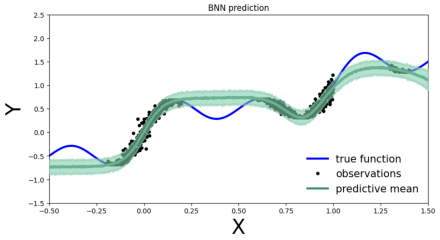

Figure 10: Better Prediction

5.3 Chebyshev Bound

In the context BNN, the code validates Chebyshev's inequality by comparing its theoretical bound with the observed probability derived from the posterior predictive distribution at various input points. There are 20 uniformly spaced x-points chosen. For each selected x-point, the code calculates the fraction of predictions within 2 standard deviations of the predictive mean and compares this with Chebyshev's bound. When there is not much data, and the variance of the predicted output at the tail end is high (figure 8) we also see a spike in the Chebyshev bound (figure 11).

5.4 Demonstration of Law of Large Numbers

To validate the LLN principle, the code computes the cumulative mean of posterior predictions at a chosen input point. It iteratively averages the predictions from the posterior predictive distribution as the sample size increases. The cumulative mean is plotted (figure 12) to show how it stabilizes and converges

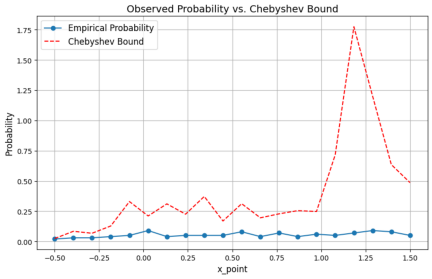

Figure 11: Demonstrataion of Chebyshev Bound

to a final value, illustrating the LLN concept. This show's how model's uncertainty resolves into a reliable mean prediction as more posterior samples are incorporated.

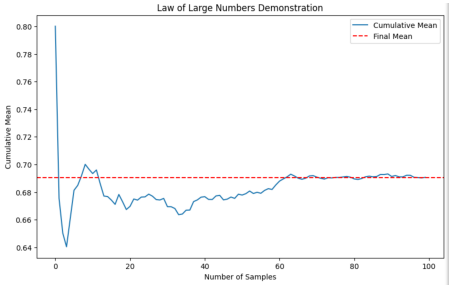

Figure 12: Demonstration of LLN

5.5 Demonstration of Central Limit Theorem

To validate CLT, 1000 samples from the posterior distribution were drawn and each sample had a sample size of 50. The mean was calculated and stored and from the figure 13 we can see that it approximates a normal distribution.

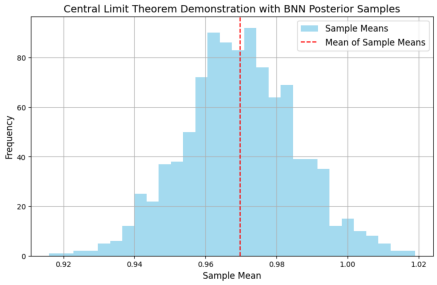

Figure 13: Demonstration of CLT

6 Conclusion

This project helped me learn about how probability concepts are used in training and predicting in BNNs. Concepts like Bayes theorem, Expectation and Variance, Chebyshev Inequality, Law of Large numbers and Central Limit Theorem were demonstrated in the network output.
To work on this project I learnt some basics of Neural Networks and also about the simulation package called Pyro (built on python) which was used to help simulate these concepts.

7 Link to the Code

The base code for this project is from the tutorial [2]. Here is the link to the notebook containing my code which adds onto the tutorial base to demonstrate various concepts.

References

[1] Bayesian neural network — deep learning. https://www.youtube.com/watch?v=OVne8jDKGUI.

[2] Tutorial 1: Bayesian Neural Networks with Pyro &x2014; UvA DL Notebooks v1.2 documentation — uvadlc-notebooks.readthedocs.io. https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/DL2/Bayesian_Neural_Networks/dl2_bnn_tut1_students_with_answers.html.

[3] David F. Anderson, Timo Seppäläinen, and Benedek Valkó. *Introduction to Probability*. Cambridge Mathematical Textbooks. Cambridge University Press, 2017.

[4] Ethan Goan and Clinton Fookes. *Bayesian Neural Networks: An Introduction and Survey*, page 45–87. Springer International Publishing, 2020.