Uncertainty Quantification Techniques in Neural Networks Progress Report

Derrick Sun (dhsun@purdue.edu)
Vishal Gajavelly (vgajavel@purdue.edu)
Sam Liu (liu3420@purdue.edu)

April 18th, 2025

1 Introduction

Neural networks have demonstrated remarkable success in a variety of applications, such as image classification and natural language processing. However, their black-box nature and tendency to be overconfident in predictions present significant challenges, particularly in high-risk domains like healthcare and finance. Uncertainty quantification is essential for making predictions that are not only reliable but also interpretable.

Traditional deterministic deep learning methods often fail to capture predictive uncertainty, leading to overconfident and unreliable results. Our research project aims to explore and analyze existing uncertainty quantification techniques in neural networks, focusing on balancing accuracy, efficiency, and interpretability.[1]

2 Problem

We have chosen a data set[2] that provides historical daily sales data from Walmart (the world's largest company by revenue) in multiple stores in three states in the United States (California, Texas, Wisconsin) during the period 2011-2016. It includes information about sales, calendar dates, promotional events such as holidays or special events. Due to computational constraints, we have chosen to focus on only items from the "Hobbies" category, the years 2014 to 2016, and the state of California.

We aim to focus on predicting next 30 days sales for various products across multiple stores. If we predict sales too low, we risk missing opportunities for stock

[1]Our code can be found at https://github.com/DerrickhSun/CS578-Uncertainty-Project.
[2]https://www.kaggle.com/competitions/m5-forecasting-uncertainty/data

replenishment or failing to meet demand, leading to lost sales and dissatisfied customers. On the other hand, predicting too high can result in overstocking, increased inventory costs, and wasted resources. The unique aspect of this project is to quantify uncertainty, requiring a forecast distribution rather than point estimates. Uncertainty is very crucial in sales forecasting because it helps businesses understand the range of possible outcomes. This enables better decision making, risk management, and resource allocation.

In order to estimate both sales and uncertainty, we employed neural networks. Specifically, we made use of Bayesian neural networks, quantile regression, and Monte Carlo Dropout.

- **Bayesian Neural Networks (BNNs)**: Incorporate probability distributions over weights to capture uncertainty. Bayesian Neural Networks are network models that aim to reduce overfitting by seeking to find the best distribution for each weight to sample from rather than set numbers. The prediction for any data point is computed with:

  $$p(\hat{y}|\hat{x}, DATA) = \int p(\hat{y}|\hat{x}, \theta^*) p(\theta^*|DATA) d\theta^* \tag{1}$$

  Where θ^* is sampled from each weight's distribution. We then take the average of multiple samples to compute the final prediction:

  $$\hat{y} = \frac{1}{K} \sum_{k=1}^{K} F_{\theta *_k}(\hat{x}) \tag{2}$$

  We can model uncertainty in this model by computing the standard deviation of the samples for each data point. The more variance there is, the more uncertain the model is about the accuracy of the prediction.

- **Monte Carlo Dropout (MC Dropout)**: Applies dropout at inference time to approximate Bayesian inference. This method acts as a Bayesian approximation to the moments of a predictive distribution:

  $$q(y^*|x^*) = \int p(y^*|x^*, w) q(w) dw \tag{3}$$

  Where:

  - $q(y^*|x^*)$ is the predictive distribution for the new data point x^*,
  - $p(y^*|x^*, w)$ is the likelihood function of the output y^* given the input x^* and weights w,
  - $q(w)$ is the prior distribution over the weights w.

  In practice, MC Dropout approximates this integral by performing multiple forward passes through the model with different dropout masks. Each forward pass corresponds to a different set of weights, and the variance across these predictions gives an estimate of the uncertainty in the model's output.

- **Quantile Regression**: Estimates prediction intervals by learning conditional quantiles. Quantile regression is a method similar to linear regression, except that instead of estimating the mean response of the dependent variables, it attempts to estimate the median and other quantiles. This is done by weighting data points to produce the error function:

  $$w \sum_{y_i > \hat{y_i}} |y_i - \hat{y_i}| + (1 - w) \sum_{y_i < \hat{y_i}} |y_i - \hat{y_i}|$$

  Where w is the quantile level, and $w = 0.5$ for the median. The model thus uses the above loss function (known as pinball loss) to train its neural network, and uses $w = 0.5$ as its prediction. It uses $w = 0.5 \pm x$ to produce confidence intervals of $2x$.

3 Methodology

In this section, we outline the process we took to train our models and produce our results. The process can be separated into three parts, with all three models using the same preprocessing, but diverging during training.

Preprocessing

The data we received is separated into multiple tables, with the main table containing store information for each item, as well as data for 2011 to 2016 and California, Texas, and Washington. Thus, during preprocessing, one of our first steps is to remove the data from Texas and Washington, as well as items from non-hobby categories. This was the first step in order to reduce computational load. Data points that were not in 2014 through 2016 were removed later, due to being necessary in some computations.

Next, we merge the tables for store information and product information (sales and sell price) using the ID, item ID, store ID, etc., before then merging event/holiday information using date.

After these initial preprocessing steps, we then performed data cleaning, normalizing the columns and converting categorical variables into their one-hot encoding representations. Null values were filled in with their mean, as well as a column denoting whether the value was filled in or not. The sole exception was item ID, which we used value encoding due to the number of possible values. This feature would be handled in our models with embedding.

Training

As mentioned in the previous section, one of the first things we did for training is add an embedding layer to better capture the patterns in sales for each item. We found that this generally improved accuracy and reduced error, so we kept this layer for our final models. We also created several PCA graphs to examine the complexity of the data. In general, we found that the data is very complex and difficult to separate, so we all trained with the highest layer sizes possible given the constraints on processing power and memory. We then adjusted to reduce overfitting and obtain accurate models for the test dataset. We trained our models with difference loss functions; some optimized for reducing mean squared error while others were optimized for reducing calibration error.

Bayesian Neural Network

The Bayesian Neural Network was trained based on ELBO loss using negative log likelihood and KL divergence. This allows the model to fit for the best standard deviation and mean for the data while also preventing the posterior distribution from deviating too much from the prior.

One of the main issues with training was the time and space needed to run the model. The DenseVariational layer used by TensorFlow to create the hidden layers in the model is much larger than the standard Dense layer used for regular neural networks because they need to store and calculate distributions. This means that there is a very heavy time and space constraint on the model's complexity; if the model has too many layers or neurons, it is very easy to run into an OOM error, and if there is enough memory, the model could take a whole day to train. This combined with the large number of other hyperparameters (kl weight, prior and posterior distributions, activation functions, optimizer) meant that it was very difficult to optimize the hyperparameters to achieve the best result.

Monte Carlo Dropout

The MC Dropout approach leverages dropout, a technique typically used during training as a method for approximating Bayesian inference at test time. In MC-Dropout, we use dropout during inference too by performing multiple random forward passes. This generates a distribution of prediction for each input and gives us a direct way to estimate uncertainty, as both the mean and standard deviation of the predictions are computed.

The base model uses an LSTM-based sequence architecture, using an embedding layer for itemid since we were making item-wise predictions. One-hot encoding was not feasible due to the large number of unique item IDs.

During experimentation, the MC Dropout model was trained using using MSE as the loss fuction. Negative log likelihood as a loss function was also explored but MSE was with found better calibration in this setting. The mean of the predictions is used as the final prediction, and the standard deviation across predictions quantifies uncertainty.

Quantile Regression

Quantile regression was done using simple neural networks with pinball loss, which is the previously mentioned loss function:

$$w \sum_{y_i > \hat{y_i}} |y_i - \hat{y_i}| + (1 - w) \sum_{y_i < \hat{y_i}} |y_i - \hat{y_i}|$$

Where w is a percentile corresponding to the desired percentile. The neural network for quantile regression has 3 hidden layers, each mapping a thousand neurons to another thousand neurons.

These numbers were chosen as it produced the largest model the device we used for quantile regression could handle. As we will note later in the findings, insufficient model complexity appeared to be an issue we struggled with.

We have two implementations of quantile regression. The default implementation trains a single neural net to perform quantile regression for every 5th percentile with a single neural network. The secondary implementation trains 19 different models, each trained for a specific percentile. The second implementation tends to take more space, but ultimately performs better.

Unlike the other two models, quantile regression was trained on unnormalized sale values, resulting in it returning much higher MSE. However, the values can be adjusted by dividing by the square of the standard deviation of sales, which allows the MSE to be comparable.

Gathering Results

We are interested in both the performance of the model and the ability of our models to estimate uncertainty. Thus, we evaluated model for both of these.

For performance, we used mean squared error and root mean squared error. While our models were trained on varying loss functions, we chose to all use mean squared error to allow performances to be compared. In addition to mean squared error, we also plotted model predictions for sales against the actual number of sales.

For uncertainty, we used a slightly modified expected calibration error. Expected calibration error is expressed as:

$$ECE = \sum_i b_i |p_i - c_i|$$

During expected calibration error, the model places its predictions in bins depending on its confidence. Then, it performs the above computation, with i iterating over the bins, b_i being the proportion of predictions in bin i, c_i is average confidence of the predictions in bin i, and p_i is the prediction accuracy.

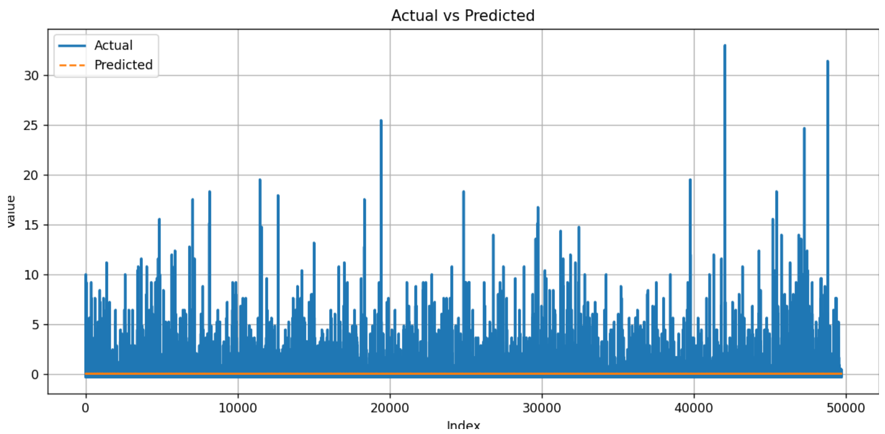
In our case, we assigned a confidence level to each of 9 bins, and had each model return an interval based on the confidence level. The confidence levels were 10%, 20%, 30%, etc, with 100% being omitted because we believed it would be trivial for a model to guess $[-\infty, \infty]$.

In addition to calibration error, we also plotted the accuracy against confidence to produce a ”calibration curve” as well.

4 Findings

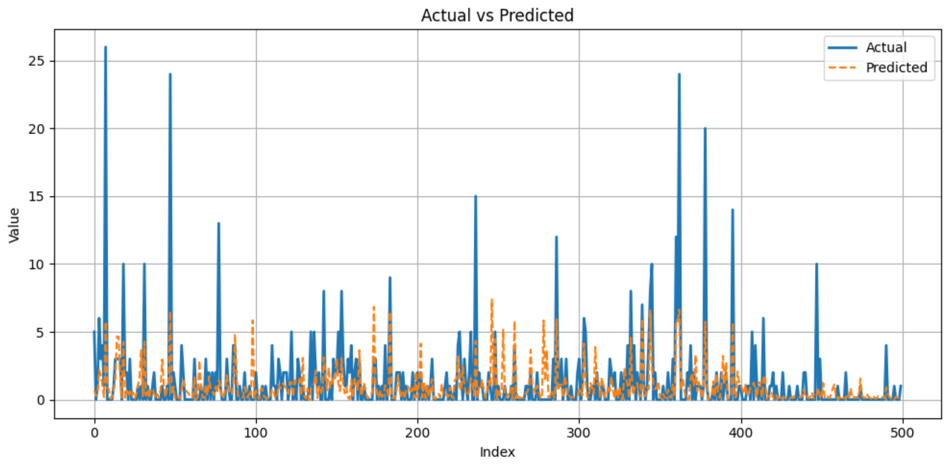
4.1 Bayesian Neural Networks

Initially, the models tended to undervalue predictions and give values below and close to 0. When evaluating the model on normalized data, it would deceptively have low RMSE, but graphs revealed the model was underfitting the data severely and produced very flat outputs.



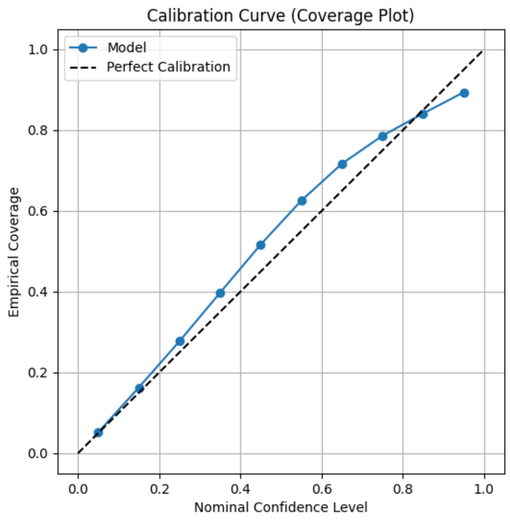
To account for this, we reduced the KL weight to stop overregularization of the posterior distribution and allow it to deviate more from the prior.

The model also needed to be adjusted to fix the prediction range. Because the values were normalized and there were many 0 values, the model consistently produced negative outputs that were slightly below the minimum of the ground truth. This led to the unnormalized RMSE to spike compared to the normalized RMSE because negative values in the predictions became more negative while the true data was adjusted back to values above 0. To account for this error, the negative log likelihood loss function was adjusted to penalize values below a certain threshold to ensure positive values after unnormalization.



As shown in the unnormalized graph above, the model is now much more balanced and now longer consistently gives predictions below 0. There is also more variability in the predictions.

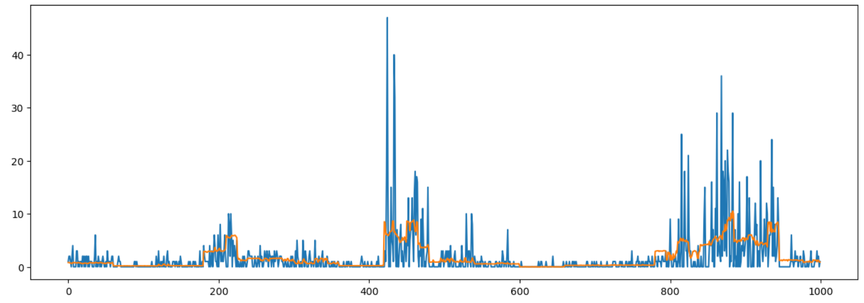
In general, BNNs modeled uncertainty very well, and its calibration score matched very closely with perfect calibration.



The predictions were also relatively accurate, with an RMSE of 0.813 for normalized values and an RMSE of 1.327 for unnormalized values.

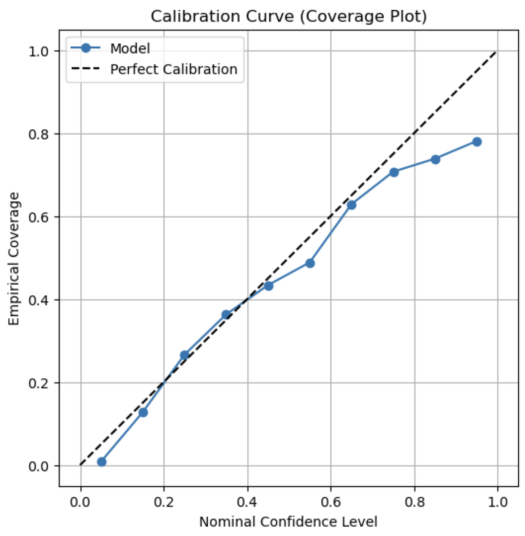
4.2 Monte Carlo Dropout

The MC Dropout model achieved a normalized test MSE of **0.6706** and a denormalized test MSE of **4.36**. The Quantile Calibration Error (QCE) was **0.9121**.



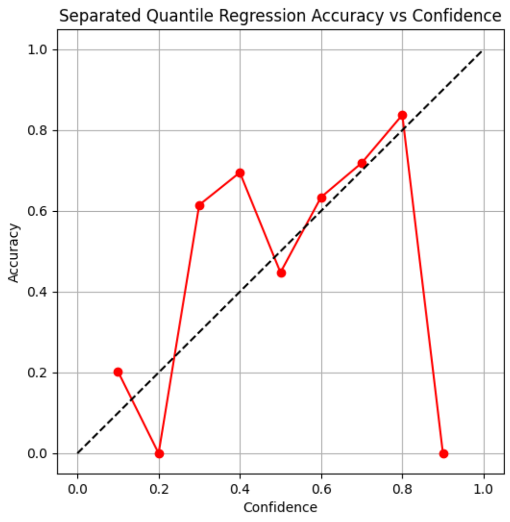
The QCE of 0.9121 remained relatively high, indicating that while global coverage was generally accurate, local miscalibration persisted in certain regions of predicted uncertainty. This is evident in the calibration curve, where deviations from the ideal diagonal suggest that the model tends to be slightly underconfident at mid-range confidence levels and overconfident at higher levels. Such behavior is a common limitation of MC Dropout, which approximates Bayesian inference by averaging stochastic forward passes.

While this technique introduces uncertainty estimation in a computationally efficient manner, it may fall short in capturing complex posterior distributions ,particularly when constrained by model capacity. This highlights a broader challenge in probabilistic forecasting: balancing predictive accuracy (e.g., low RMSE) with well-calibrated uncertainty is inherently difficult, especially in the presence of complex, noisy, or non-Gaussian data. Nonetheless, MC Dropout remains a practical and widely-used approach for uncertainty quantification in deep learning, especially when full Bayesian methods are computationally infeasible.



4.3 Quantile Regression

Initially, a singular model was trained to predict all necessary percentiles, which included the 5th percentile, 10th percentile, etc, up to the 95th percentile. This saw both poor accuracy with an unnormalized MSE of 6.6389 (adjusted MSE 1.11)[3] and poor calibration error of 0.2173.

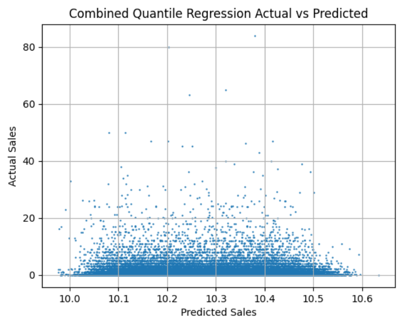


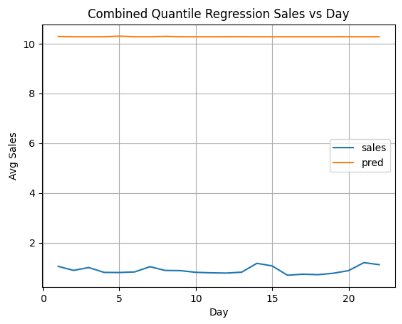
While it is clear that the model made an attempt at predicting where the per-

[3]Recall that quantile regression was trained on unnormalized sale value. The adjusted MSE should be used when comparing with the other two models.

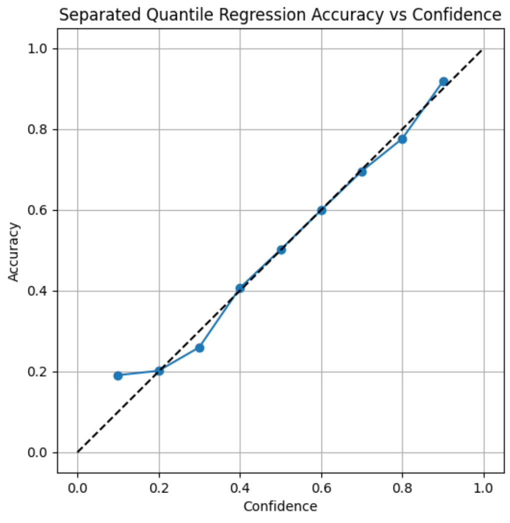
centiles should be, it also suffers major issues. A problem with the calibration error that had not been foreseen was that the model occasionally produced nonsensical confidence intervals, such as cases where an interval of greater confidence was strictly smaller than an interval of smaller confidence.

Additionally, accuracy was also very poor. As can be seen from the figures, the model almost entirely guessed at around 10 sales, despite the vast majority of sales actually being very small.

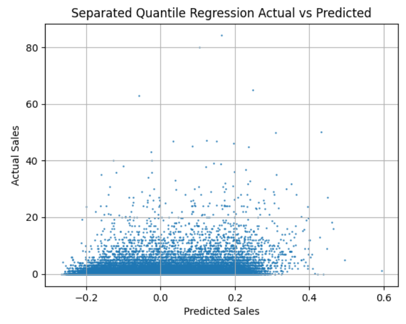


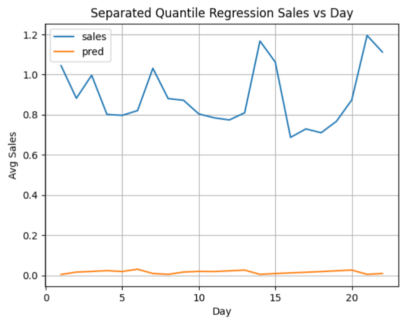


From interpreting these findings, we determined it was likely that the model did not have sufficient complexity to properly capture patterns in the data. However, we struggled to increase model complexity due to limitations on our computational power. We were able to perform a workaround, where we had separate models predicting each percentile. The results were then combined at the end, as if a singular model had done the prediction. This saw much better performance.



The calibration error of the separated model follows the ideal calibration much more closely, with a calibration error of 0.0208. However, while the actual predictions changed significantly, the actual predictions were still fairly off. Rather than predicting a value too high, the separated model often under-predicted. We can see this is likely due to the fact that the vast majority of sales are 0 (recall that sales are normalized). It also had a fairly large MSE of 6.6813 (adjusted MSE 1.1175).





4.4 Overall Conclusions

In general, just having uncertainty statistics is not enough to judge a model. In many of our simpler models, the calibration error was low, but that was mainly because the models were uncertain about many data points and also got these predictions wrong, leading to low expected calibration errors. It is more useful to have uncertainty as a supplementary statistic with an already strong model to obtain both accurate results as well as an understanding of how certain the model is that is correct.

It does seem that our models did not have enough complexity to properly predict the number of sales, even if they did decently well in terms of calibration error. The spikes shown in the test data were oftentimes not represented by our models as a result of this. Unfortunately, we were constrained by computational power and the memory of our devices.

5 Future Improvements

In the future, given more time and resources, we would look to improve our models in terms of both uncertainty and accuracy. After achieving the best possible models for each of our techniques, we would compare them on the same dataset to determine which one is better at uncertainty modeling while maintaining a certain level of accuracy.

We would also look to improve model complexity so that the peaks and patterns in the data are more expressed during predictions. Another major improvement would be to scale the models. Currently, we are still subsampling our data due to memory constraints. Our future models would be run on more powerful machines so that we can include data from all years and all store locations.