SGH

Bayesian Neural Networks for regression problems

Author: Rafał Wójcik

Warsaw 2020

Table of contents

1. Introduction

Goal of this project was to evaluate efficiency and adequacy of Bayesian Neural Networks (BNN) in regression problems – in this project in modeling flat and house prices from Warsaw, Poland.

Bayesian Neural Networks are an extension of classic Artificial Neural Networks (ANN). Main difference between BNN and ANN is the fact, that BNN's allow estimation of approximate distribution of each parameter of the Neural Network, and not only point estimate of each parameter value. It is a significant difference to ANN's, as it allows on one hand to keep track of model's uncertainty associated with its predictions, but also allows better generalization of the model [1.].

Dataset used in this project contained flats and house prices from Warsaw, obtained with web scraping from site https://www.otodom.pl/.

Potentially, ANN's stand for much more complex model's than simple regression techniques such as Ordinal Least Squares (OLS) regression, and because of that they could be preferred because their performance can be much higher on higher amounts of data. On the other hand, they do not allow keeping track of model's certainty, and the only way of measuring how sure model is about its predictions is to evaluate it with some metric on holdout data it has not seen. Additionally, ANN's have been proved to be prone to overfitting on training datasets [1.]. On the other hand, BNN's potentially allow to estimate models of the same complexity, but in the same time they allow to control certainty of the model, and regularize model to not overfit training data [1.].

To conclude, BNN's can be promising alternative to classic ANN's, and because of that evaluation of their performance was done in this project. As a baseline performance model there was simple OLS model estimated.

2. Bayesian estimation of Neural Networks

As mentioned in previous chapter, main difference between BNN's and ANN's is that in BNN's there is an estimation of distributions of parameters in the model, and not only parameters' point values. In classic ANN i-th layer's parameters can be seen as n by k dimensional matrix of numbers, where n is the number of parameters in each neuron of i-th layer, and k is the number of neurons in i-th layer. Illustration 1 presents simple scheme of activations in small ANN with 3 neurons in first layer, and 1 neuron in second layer:

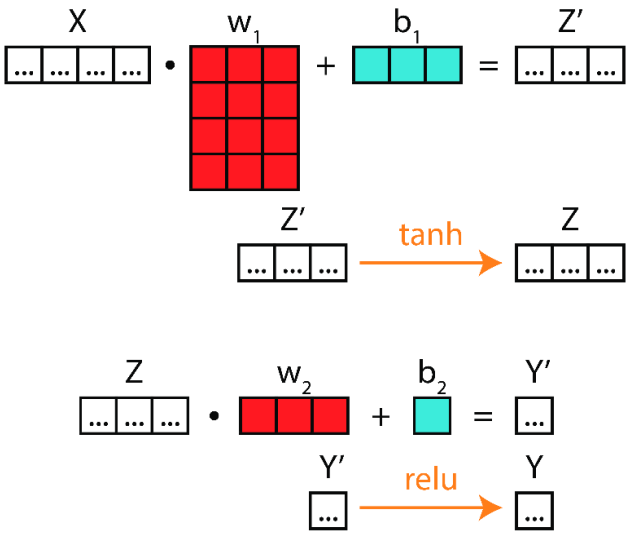

Illustration 1: Scheme of activation's in ANN. Source: https://ericmjl.github.io/bayesian-deep-learning-demystified/images/deepnet_regressor-matrices.png

In BNN's however, matrix w_1 is not a matrix of real numbers, but each weight is being sampled from distribution that is estimated throughout the models' training:

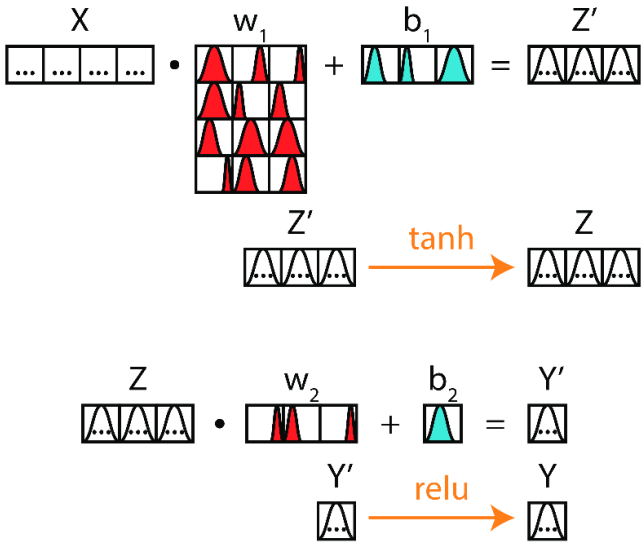

Illustration 2: Scheme of activation's in BNN. Source: https://ericmjl.github.io/bayesian-deep-learning-demystified/images/deepnet_regressor-matrices-bayesian.png

Essentially, the optimization problem in BNN can be viewed as estimating posterior distribution $P(\boldsymbol{w}|D)$ of weights **w**, given training data *D*. Such distribution cannot be estimated analytically for BNN's of reasonable size, as calculations associated with solving such problem would require to essentially estimate uncountable number of artificial neural networks [1., 3.]. Reason for that is that in BNN's, for single observed value of response variable y_i and vector of independent variables $\boldsymbol{x_i}$, the prediction from is obtained based on every possible configuration of BNN's weights from posterior distribution [1.]

Because of that, posterior distribution of parameters $P(w|D)$ is approximated in BNN's by sampling it from simpler distribution $q(w|\theta)$, which parameters can be learnt by minimizing Kullback-Leibler divergence (KL) function between parameters **w** learned from simplified posterior distribution and weights **w** from true posterior distribution:

$$\theta^* = F(D,\theta) = argmin\, KL_\theta[q(w|\theta)\,||\,p(w|D)] =$$
$$= argmin\, KL_\theta[q(w|\theta)\,||\,P(w)] - E_{q(w|\theta)}\,[logP(D|w)]\ ,\ \textit{(1)}$$

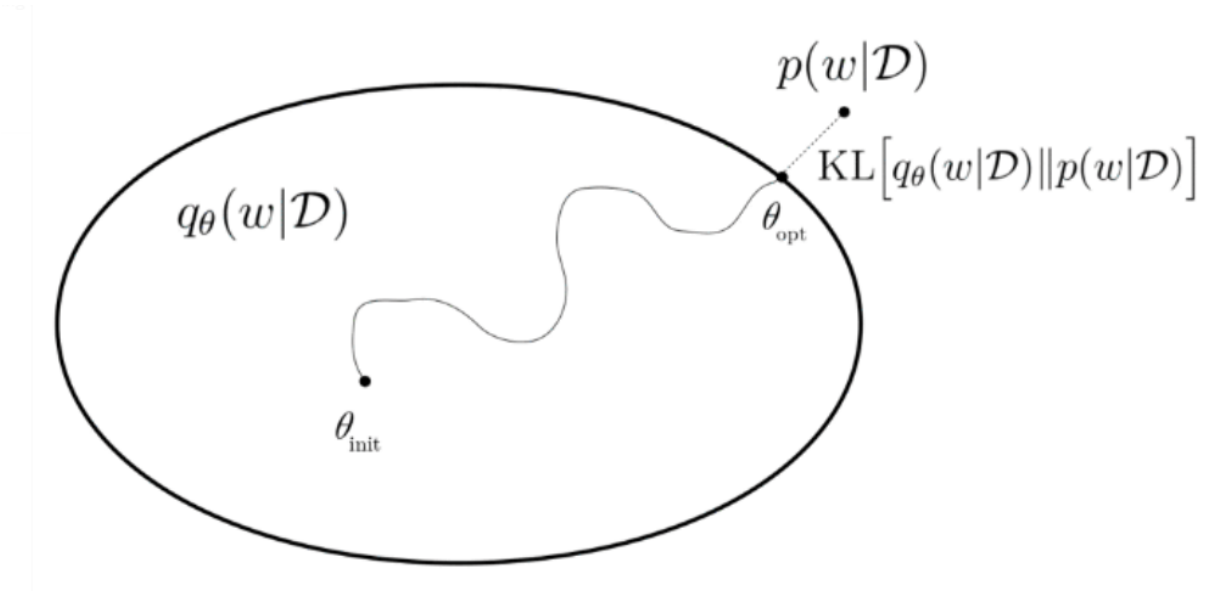

Illustration 3: Scheme of approximation of true posterior distribution with variational parameters of simpler distribution by minimization of KL divergence. Source: [3.]

Reason for sampling weights from simplified distribution q(w|θ) instead of solving equation (1) analytically to find set of thetas that would minimize KL divergence is that it would require to solve another intractable integral, as the number of thetas is too big. [1., 3.]

Because of that final optimized equation in BNN's is equal to:

$$F(D,\theta) \approx \sum_{i=0}^{n} \log q\left(w^i|\theta\right) - \log P\left(w^i\right) * P\left(D|w^i\right),\ (2)$$

Where $q(w^i|\theta)$ is simplified distribution that approximates true posterior distribution of weights **w**, w^i is i-th sample of weights drawn from $q(w^i|\theta)$, $P(w^i)$ is probability for certain set of weights of being drawn from prior distribution and it serves as regularization part of the optimized cost function [1.] . Such approach can be thought of as 'Bayesian' one, as the model is being optimized to on one hand satisfy the data-dependent part of the equation, and on the other it's prior, initial distribution. The prior part of the equation is also referred as complexity cost, as it increases if the posterior distribution estimated during training differs much from the initial and simple prior distribution (their distributions shift towards too big negative or positive values). Prior distribution according to [1.] should be sampled from mixture of two Gaussian density functions with the mean – zero, and different variances.

Final step in making equation (2) optimizable with Gradient Decent techniques, which are main tool for optimizing neural networks, is a local reparameterisation trick, that introduces for every distribution of every weight two local (named also variational) parameters - μ and σ [1., 3.] These parameters shift prior distribution from its initial values to ones that fit data *D* better, and they are the parameters optimized in each iteration of model's training. Variational parameters are changed during models' training with Gradient Descent-based algorithms,

3. Dataset Description and cleaning

Dataset used in this project have been collected with use of web scraping from https://www.otodom.pl/. Originally the dataset contained 13499 observations with 33 variables. Because of the relatively unstructured source of the data, there were many variables that contained mostly nans. Because of that all variables which contained more than 5000 missing rows were removed from the dataset at the beginning.

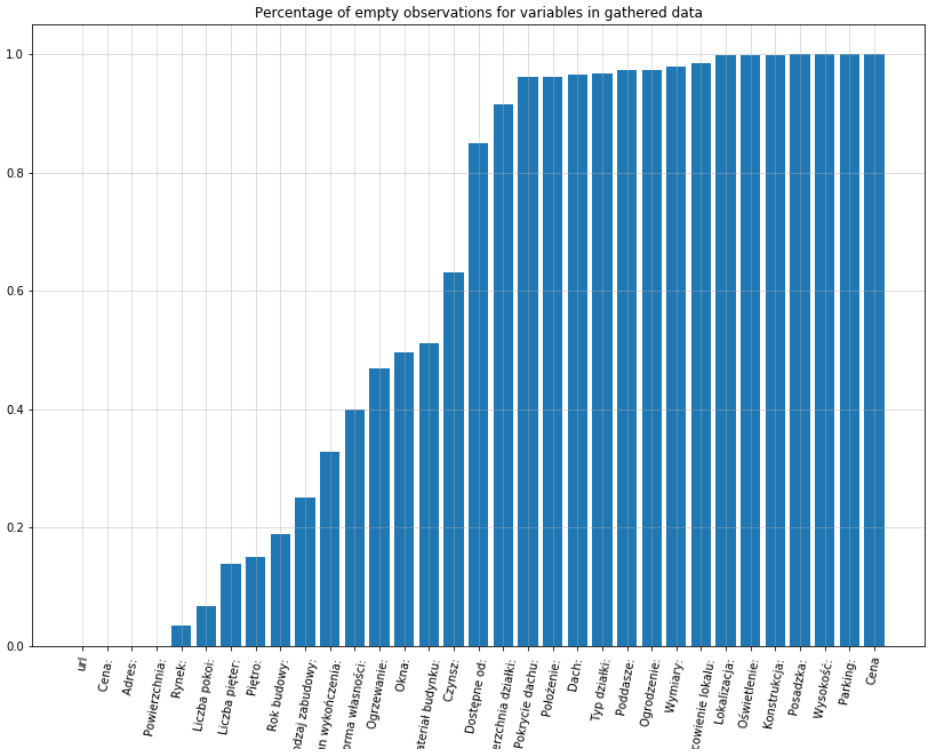

Illustration 4: Percentage of missing values in each variable of original dataset

Furthermore there were several steps taken while cleaning the data, that included i.e.:

- Cleaning all columns' names
- Extraction of city district from 'Adres' variable
- Encoding of categorical variables with one hot encoding technique
- Cleaning incorrect values in response variable and removing all rows in which price was equal to 0

Cleaned dataset contained 5878 rows with 33 variables (including response variable).

Variable name	Variable type	Description
Rok budowy	Ordinal	Year of construction
Cena	Continuous	Price in zł
Powierzchnia	Continous	Area of flat in m^2
Liczba pokoi	Ordinal	Number of rooms
Liczba pięter	Ordinal	Number of rooms
Stan wykończenia	Categorical	Finishing condition (one hot encoded into separate binary variables)
Rodzaj zabudowy	Categorical	Type of construction (one hot encoded into separate binary variables)
Dzielnica	Categorical	City's district (one hot encoded into separate binary variables)

Table 1: Variables used in analyzed dataset. All one hot encoded variables were grouped to their corresponding categorical variables, to make table more readable

Next, the dataset has been scaled with min max scaling technique, and additionally response variable have been transformed logarithmically, as its distribution was highly skewed.

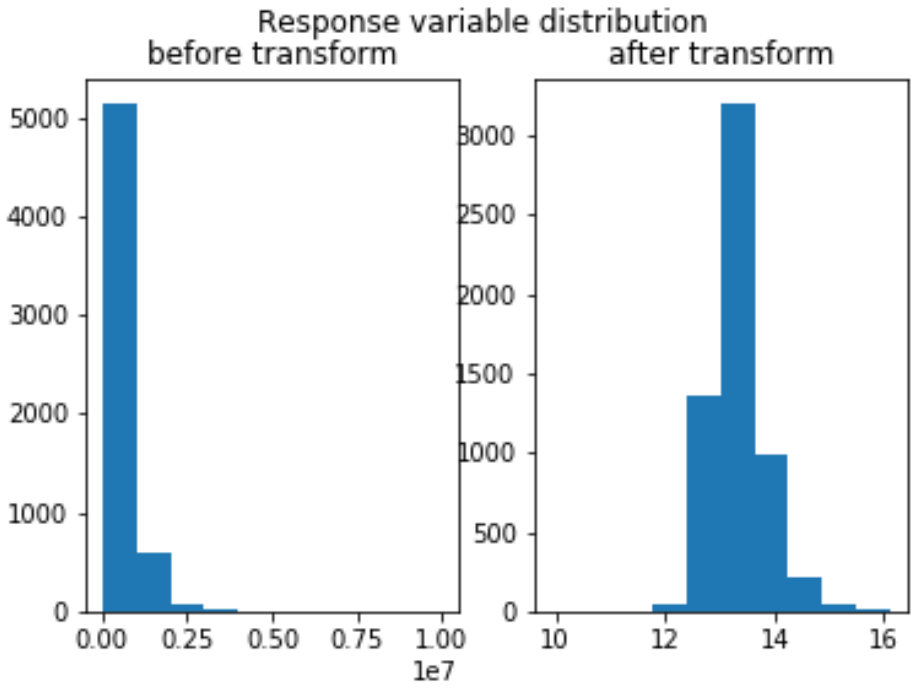

Illustration 5: Response variable distributions before and after logarithmic scaling. Source: own elaboration

Main reason for min max scaling was that prior distributions in BNN are sampled from Gaussian distributions with mean 0 and chosen variance. Because of that, it was important to scale all values to range between 0 and 1, to allow easier estimation of weights, that are always constrained by their prior distribution of mean 0.

Lastly, dataset have been split into train and test datasets with 75% in train dataset and 25% in test dataset.

4. Model definition and prior distribution choosing

Models trained in this project consisted of 2 linear layers with ReLU activation function after first layer. First layer layer of neurons consisted of 256 neurons, and second (output layer) of one neuron.

There were many different architectures of models tested during initial experiments, some with even 6 layers with 1024 neurons, but none of these models performed better than simple model described above, so only scores of model's with such structure will be discussed further.

Finally, there were two model's trained with different prior distributions. First one's weights have been initialized with equal mixture of 2 Gaussian distributions with mean equal to 0 and variances equal to 0.1 and 0.01 respectively. This means, that prior part of model's loss (approximate of likelihood) function was calculated based on negative logarithm of likelihood, that current set of BNN's posterior weights was sampled from such mixture of two distributions. Both distributions were weighed with the same value (0.5). Choice of using two Gaussian distributions weighted with certain value comes from [1.]

On the other hand, second model was trained with prior distributions with variances equal to 1 and 0.5 respectively, also weighted equally. Higher choices of prior distributions variance were mainly used to allow better fit of the model to the data – with higher prior variance, the cost of using weights that differ much from their prior distribution is lower. Higher variance on prior distribution can be also seen as weaker regularization of the network, as it allows the weights to be optimized further away from values of mean 0.

5. Training and models' scores

Both models were trained for 1000 epochs, with batch size equal to 256. Models were optimized using Adam optimizer with initial learning rate equal to 0.01, and Mean Squared Error as loss function.

In each iteration metric used for evaluating model's performance was on one hand mean loss of all batch losses in given epoch, and on the other accuracy of confidence interval associated with model's predictions.

Loss of each batch was obtained as average of 10 losses obtained by sampling model's weights from their current posterior distributions. Accuracy of confidence interval obtained with model on test dataset was calculated by sampling model's weights 25 times and averaging how many times the true value of response variable was inside of confidence interval associated with model's predictions. Confidence interval was calculated as a mean of predictions from these 25 samples, +/- their standard deviation. Such set of metrics was inspired by examples of BNN training from [4.]. In the process of choosing the best model, only loss function on training dataset was chosen as an indicator.

As a baseline model there was simple OLS model trained from sklearn library. It's scores on train and test datasets were presented in table 2 below:

	Train Dataset	Test Dataset
Mean Squared Error	20129.78	20135.87

Table 2: Baseline model's scores on train and test datasets

First model's scores:

As prior distributions of first model had lower variance, weights sampled from posterior distributions during the training could not differ much from their prior value. As a consequence, model's training broke during epoch 14. Broke in this context means, that it's loss value was so big, that in computing libraries used in this project (PyTorch), it approached infinity. Reason for that, was that posterior distributions of weights obtained during the training shifted too far away from chosen prior distributions. Because of that probability that certain sample of weights came from their prior distribution approached 0. Whilst it the prior probability approached 0, it's logarithm approached infinity, which caused the whole loss function approach infinity, and as a result the training had to stop.

Nevertheless, best model obtained from such distributions before the training broke was saved, and table 3 presents its mean squared errors on train and test datasets:

	Train Dataset	Test Dataset
Mean Squared Error	20151.27	20160.95

Table 3: First model's scores on train and test datasets

Second model's scores:

For second model, chosen set of variances for prior distribution allowed to finish whole training without loss approaching infinite values, and also allowed model to fit the data better. With such approach, overall scores were not significantly better than of the 'backup' version of first model, though there was a small improvement. It can be questionable if second model wouldn't generalize worse, but to determine this issue it would be necessary to collect more data and evaluate model further, and for such small differences this issue shouldn't be a problem. Table 4 contains second models' scores on train and test datasets:

	Train Dataset	Test Dataset
Mean Squared Error	20074.94	20095.76

Table 4: Second model's scores on train and test datasets

To visualize how mean predictions of best model shifted during the training there were 4 plots created of mean predictions (mean from 200 samples) of the model after first epoch of the training and for the best obtained model. Plots were made for each model and for 50 observations with lowest prices, and 50 observations with highest prices, in ascending order.

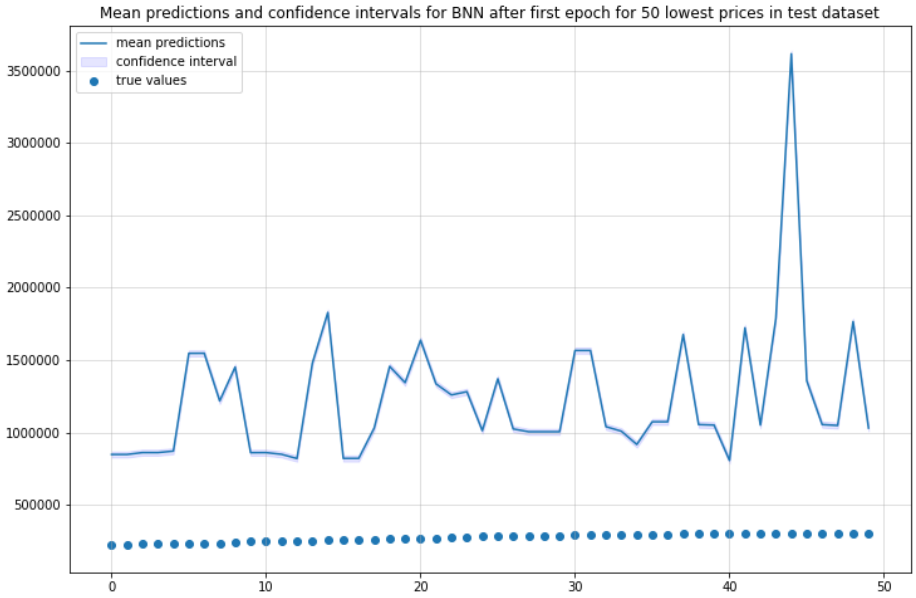

Illustration 6: Model's mean predictions on 50 lowest price observations from test dataset after first epoch

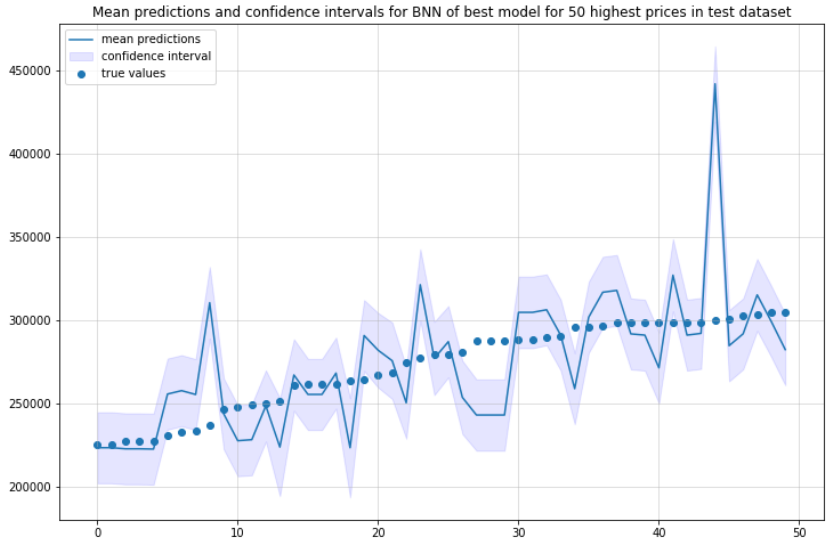

Illustration 7: Mean predictions of best model on 50 lowest price observations from test dataset

Both on illustration 6 and 7 there is a clear shift of model's mean predictions, that fits the data better. On the other hand, confidence intervals (approximated only with use of standard deviation) associated with these predictions became higher, in comparison to predictions value. Depending on modeled problem, such uncertainty could be or could not be acceptable, though in this case it was acceptable (~25000).

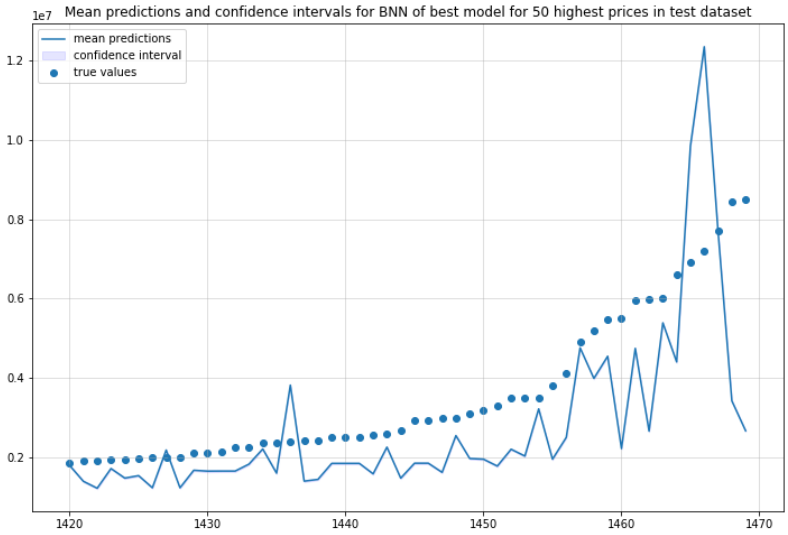

Illustration 8:Model's mean predictions on 50 highest prices observations from test dataset after first epoch

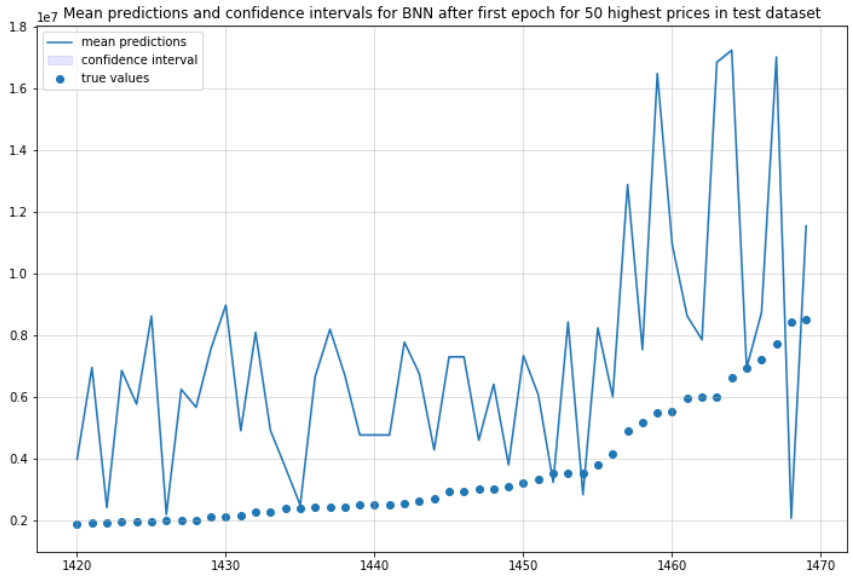

Illustration 9: Best model's scores on 50 highest price observations from test dataset

On illustrations 9 and 10 however, differences between mean predictions of model and true observed value are higher, and uncertainty associated with these predictions is relatively small. Most of these predictions are also over-estimated, so it could indicate that there are outliers in analyzed dataset that maybe should have been take into account. It is clear that with such posterior distributions of parameters, predictions of

the model are very certain for high values, and relatively less certain for lower values of response variable

6. Conclusions

Neither of trained BNN models performed significantly better than much simpler baseline OLS model. There could be several explanations for such situation. First of all, dataset after cleaning contained relatively small amount of training data. Smaller datasets usually benefit simpler models and do not require such complex structures as classic ANN's, or BNN's in this case. On the other hand, if the training dataset consisted of more observations, and more variables, it could be reasonable to use more complex model such as BNN's trained in this project. Secondly, for simplicity and from limited time reasons, categorical variables in both BNN's were encoded with one-hot encoding technique. Another technique for encoding used in ANN's and potentially in BNN's, that allows better extraction of information is embedding of categorical variables with vectors of trainable weights. It was not used in this project, mainly because of lack of time, but also because it was not implemented in used library. Reportedly ([5.]) embedding techniques prove to be very efficient for tabular data, as they enhance ability of the network, to model better any latent dependencies hidden in the data. Nevertheless, for analyzed dataset, OLS model could be preferred one, as it didn't perform much worse, and it is much easier to interpret.

7. Summary

Goal of this project was to evaluate Bayesian Neural Network on regression task, and check if it would outperform simple regression techniques such as OLS regression on analyzed dataset. Importance of BNN's, if they would perform significantly better, is that they offer on one hand complexity and effectiveness similar to classic Artificial Neural Networks, but in the same time they allow to control certainty associated with model's predictions [1.]. There were two BNN model's trained with different sets of prior distributions of weights. Second model, with higher prior variance allowed better fit of the model to the data, and also it allowed to train the model for chosen amount of epochs without loss approaching infinity. In the project, there wasn't big improvement of BNN model's performance over simple OLS regression identified.

8. Bibliography

1. https://arxiv.org/pdf/1505.05424.pdf
2. https://ericmjl.github.io/bayesian-deep-learning-demystified/
3. https://medium.com/neuralspace/probabilistic-deep-learning-bayes-by-backprop-c4a3de0d9743
4. https://github.com/piEsposito/blitz-bayesian-deep-learning
5. https://course.fast.ai/videos/?lesson=4

9. Table of graphs