Projects Statistical Analysis of Networks and Systems (SANS-MIRI) Homework 4

By
Umberto Salviati
Universitat Politècnica de Catalunya

Barcelona, January 2024

Contents

List of Figures

Chapter 1

Calibration using a FFNN

Firstly, in this homework, we need to calibrate our cheap sensor for oxygen detection using a simple neural network formed by a single hidden layer with variable width (2, 3, 4, 10 neurons or perceptrons) that employs a ReLU activation function. As a first step, I proceeded with data normalization, or more precisely, standardization, achieving a mean of 0 and a variance of 1 in the data. This step is useful to facilitate faster convergence of learning algorithms to the result.

Subsequently, I implemented the neural network as required, using the ReLU activation function. Focusing on the structure of the neural network, it is defined as follows:

$$y = f_\Theta(x) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(x;\Theta^{(1)});\Theta^{(2)}\right);\Theta^{(3)}\right)$$

Where $f^{(1)}$ represents the input layer with a width of 3, and the output layer has a width of 1 since it is a regression problem, not a classification one. Regarding the activation functions of our perceptrons, we used ReLU (leaky) and Linear (aka no activation Func).

The ReLU function is defined as $ReLU(z) = \max(0, z)$. In particular, using the term "leaky," our function is not perfectly 0 for negative values of x; instead, it features a line intersecting the origin with a minimal slope for negative x values. Subsequently, I proceeded to train the neural network. To address the minimization problem in the Feedforward Neural Network (FFNN), one can employ Stochastic Gradient Descent (SGD), defined as:

$$\Theta^{(t+1)} = \Theta^{(t)} - h_t \nabla_\Theta L(\Theta^{(t)})$$

This iterative method allows convergence to the gradient since its expected value is precisely the gradient. However, looking at the formula, it is evident that a hyperparameter h_t, known as the learning rate, is crucial. This prevents SGD from behaving too randomly without ever converging. In my case, instead of using SGD, I adopted Adam, a more common and faster algorithm for solving this type of problem.

Delving into the training of our NN, it can be observed from the code that it is divided into two parts:

1. **Forward step:** For the current values of weights Θ and input x, calculate the values of activations, logits, and the output of the network.

2. **Backward step:** Recursively calculate the gradients of the cost function with respect to Θ (and biases b).

The technique known as backpropagation is employed during training because the problem becomes significantly large, especially during the computation of SGD, requiring the calculation of the loss and its partial derivative for each parameter for each point. In particular, SGD focuses on calculating this gradient only on a restricted set of data called a batch, to alleviate the memory and computation load of the gradient. This makes the computational load of each iteration lighter, but more iterations are needed to converge to the solution.

Specifically, at each iteration, the learning rate steadily decreases. To be more precise, we can say that h_t is an infinite summation, but the summation of the squares of h_t must be finite (Robbins-Monro conditions).

To be more specific, the effectiveness of SGD in using mini-batch is due to:

$$E[\nabla_\Theta l(y_i, f_\Theta(x_i))] = \nabla_\Theta L(\Theta)$$

Where l is the loss of a single point, while L is the loss calculated on the entire training set. In particular, the computational weight of SGD is reduced by a factor of T/B, where T is the number of elements in the training set, and B is the batch size.

As can be observed from the loss plot during training, it is noticeable that the graph is quite erratic. This is likely due to Stochastic Gradient Descent (SGD), which converges towards the gradient in a chaotic manner, as depicted in Figure 1.1.

In our case, we need to train our 4 neural networks for each FFNN with different numbers of neurons. Increasing the number of neurons only serves to increase

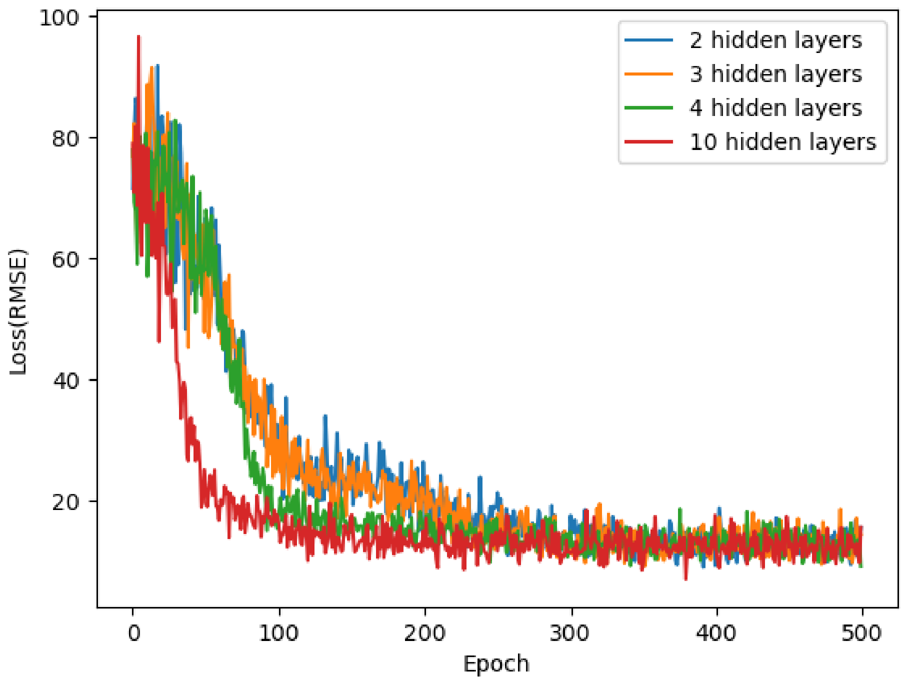

Figure 1.1: Loss plot during training.

the complexity of our model. Another method to regulate complexity is the addition of regularization, similar to what occurs in Ridge regression, where the loss is defined as:

$$L(\theta) = ||y - \Phi(x)\theta||_2^2 + \lambda||\theta||_2^2$$

This process impacts the cost of the loss by introducing the disadvantage of parameter size. In this case, the standardization of input data is crucial. By adding regularization, we modify the complexity of our model by constraining the accessible functions to our network and reducing its overall complexity.

Adjusting the complexity of a model is crucial to avoid both overfitting and underfitting. Specifically, in underfitting, we have a problem as our model has low complexity. To be more precise, the approximation error is too high, with a significant approximation error compared to the Bayes risk R^*.

$$\inf_{f \in F} R_f - R^*$$

Where R^* is the Bayes risk, i.e., the error of the best function considering all possible functions. The problem mainly lies in selecting a too-restricted subset F that may not include the function that represents our data well (unless by chance).

On the contrary, if we have a model that considers too many functions, i.e., a model that is too complex, we may encounter overfitting. With a very complex model, it's possible that among the considered functions, there is one that precisely describes our data. However, during training, we obtain a function that perfectly interpolates only our training set, failing to generalize the problem correctly. We would then have a large estimation error defined as $R_{\hat{f}} - \inf_{f \in F} R_f$, where $R_{\hat{f}}$ is the output of empirical risk minimization. Therefore, our training algorithm may return a function with a $R_{\hat{f}}$ much higher than $\inf_{f \in F} R_f$, whose function is our real objective.

Now that we have our 4 trained models, we need to understand which of them better describes our problem. In other words, we must identify the model that comes closest to reality. As highlighted earlier, we cannot use the RMSE calculated on the training set to determine which model has produced the best function, as we might encounter overfitting. Therefore, we need to use a validation set to understand which model has generated a function that best describes the data.

In general, our goal is to find a model that has neither low bias and high variance nor the opposite. Once we have obtained an estimate instance $\hat{f}$ (or, in parametric methods, $\hat{\theta}$), we can evaluate the goodness of this estimate using the Mean Square Error (MSE):

$$MSE(\hat{\theta}) = E(X,Y), TS[(Y - \hat{Y})^2]$$

Where $E(X,Y), TS$ represents the expectation with respect to the training set. Breaking down the MSE, we get:

$$E(X,Y), TS((Y - \hat{Y})^2) = E(X,Y), TS((Y - E(X,Y)(Y) + E(X,Y)(Y) - \hat{Y})^2)$$

$$= \sigma^2 + E(X,Y), TS((E(X,Y)(Y) - \hat{Y} + ETS(\hat{Y}) - ETS(\hat{Y}))^2)$$

$$= \sigma^2 + |E(X,Y)(Y) - ETS(\hat{Y})|^2 + VarTS(\hat{Y})$$

Thus, the MSE is composed of the variance of our model ($VarTS(\hat{Y})$), the square of the difference between the expected value of our response variable and the predicted response variable ($|E(X,Y)(Y) - ETS(\hat{Y})|^2$), and the intrinsic variance of the data (σ^2). This way, we can more accurately assess the goodness of our estimates and select the model that provides the best representation of the data. In other words, we need to find the point where variance and bias are similar, achieving a balance between model complexity and generalization ability. We can assess this balance using the loss on the validation set. In our case, it seems that the model with 10 neurons is the one that best describes the data. This suggests that the other, less complex models suffer from underfitting, having high bias and low variance.

After observing this result, it is important to calculate the true RMSE of our model. However, using RMSE on the validation set might lead to overfitting on this subset. Therefore, the use of a third set, the test set, is necessary, which will provide the true RMSE of our model, thus avoiding overfitting on a specific subset and ensuring a more accurate evaluation of the model's performance in general cases.

In particular the ouput that i got are:

- RMSE of the model with 2 is: 14.2456
- RMSE of the model with 3 is: 14.5190
- RMSE of the model with 4 is: 13.9696
- RMSE of the model with 10 is: 13.2396

So as we can see the best model in the validation set is the most complex model where in the test set is having as RMSE:
RMSE of the model with 10 perceprtons is: 12.4268 for test set.

We can also observe the ouput of the prediction against the real values in the graf of figure 1.2

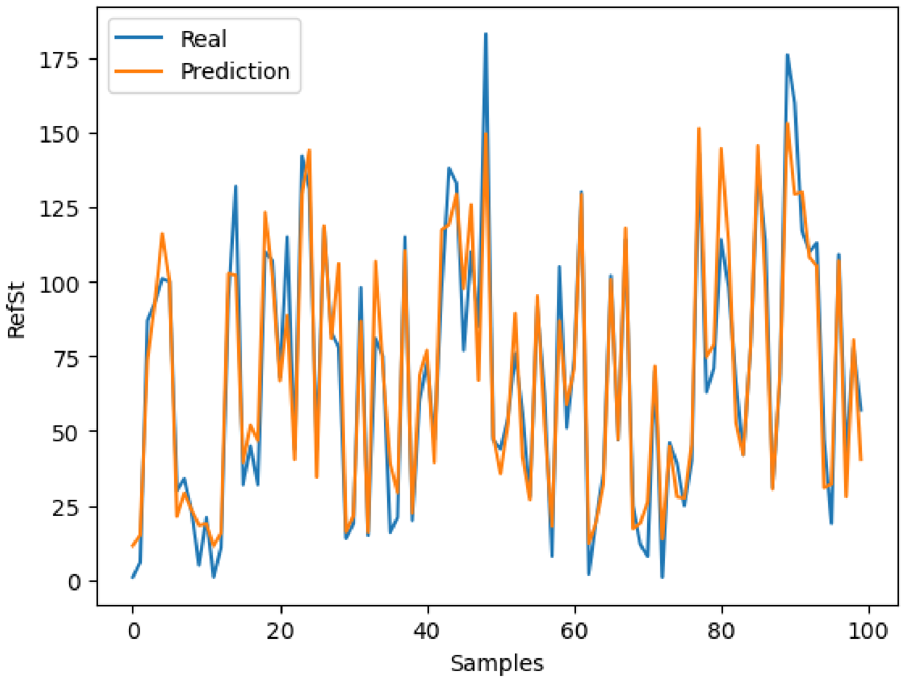

Figure 1.2: Prediction of the FFNN in the Test Set

Chapter 2

Calibration using Bayesian multiple linear regression

In this second part of the homework, we focus on calibrating the sensor using Bayesian Linear Regression, assuming that the output follows a normal distribution with mean $\theta_0 + \theta_1 x_{SO_3} + \theta_2 x_{Temp} + \theta_3 x_{RH}$ and variance σ^2. Therefore, we want to compute the posterior of our data.

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$$

In our case, the parameters are four, and the prior introduced is a normal distribution. We will proceed to calculate the evidence using the Markov Chain to finally compute our priors. In this case, we desire the complete distribution as the posterior, so we will not use $\theta_{MAP} = \text{argmax}(p(\theta|y))$. This value is called the Maximum A Posteriori (MAP).

Then, we focus on the prediction problem, which involves evaluating the posterior predictive distribution $p(y_{m+1}|y) = p(y_{m+1}|(y_1, ..., y_m))$. Subsequently, after calculating our posterior, we proceed to compute the probability of y_{m+1} as:

$$p(y_{m+1}|y) = \int p(y_{m+1}|\theta)p(\theta|y)d\theta$$

In the context of machine learning, probability effectively translates into distribution. The machine learning model is considered solved once we know $p(\theta|y, x)$ and the predictive distribution:

$$p(y_{m+1}|x_{m+1}, y, x) = \int p(y_{m+1}|x_{m+1}, \theta)p(\theta|y, x)d\theta.$$

In this context, prediction is no longer a single value as in the frequentist approach but rather an entire distribution of values. Consequently, we obtain not only an estimate of the number but also an estimate of the associated uncertainty. In summary, as input to our Bayesian estimator, we have the prior and the data, while as output, we obtain an entire distribution of our prediction.

We have thus defined the distribution of priors. However, in our case, we do not know if the probability of our output is indeed normal. In fact, if we plot the available data, we can observe that the distribution is not exactly normal. We must, therefore, apply the Markov Chain to find the correct posterior:

$$p(\theta|y,x) = \frac{1}{Z}p(y|x,\theta)p(\theta).$$

The underlying idea of Monte Carlo methods is to generate different samples $(\theta^{(0)}, \theta^{(1)}, ..., \theta^{(T)})$ from the distribution of $p(\theta|y,x)$, even without knowing the constant Z.

Thanks to the Markov Chain, we have managed to find our posterior for each attempt. In particular, the outputs are described in Figure 2.1.

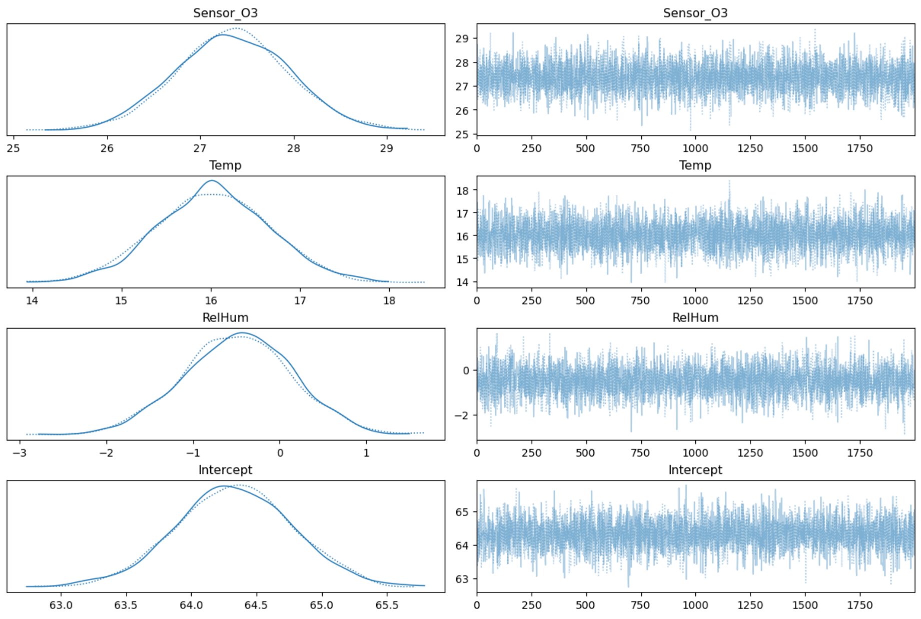

Figure 2.1: Posterior outputs of θ

Subsequently, I proceeded with Bayesian Linear Regression, defined as:

$$p(y|x,\theta) = \mathcal{N}(y; \theta_1\phi_1(x) + ... + \theta_n\phi_n(x), \sigma^2) = \mathcal{N}(y; \phi_t(x)\theta, \sigma^2)$$

I simply implemented this theory, obtaining results for the readings corresponding to the date 11/07/2017 at 11:00, as shown in this figure 2.2. Here, it can be seen that the mean of the distribution (assumed to be bell-shaped) is very similar to the actual value. In contrast, in the case of readings at 11/07/2017 at 18:00, as highlighted in the image 2.3, the prediction differs. This indicates that the second actual value is an outlier in our dataset.

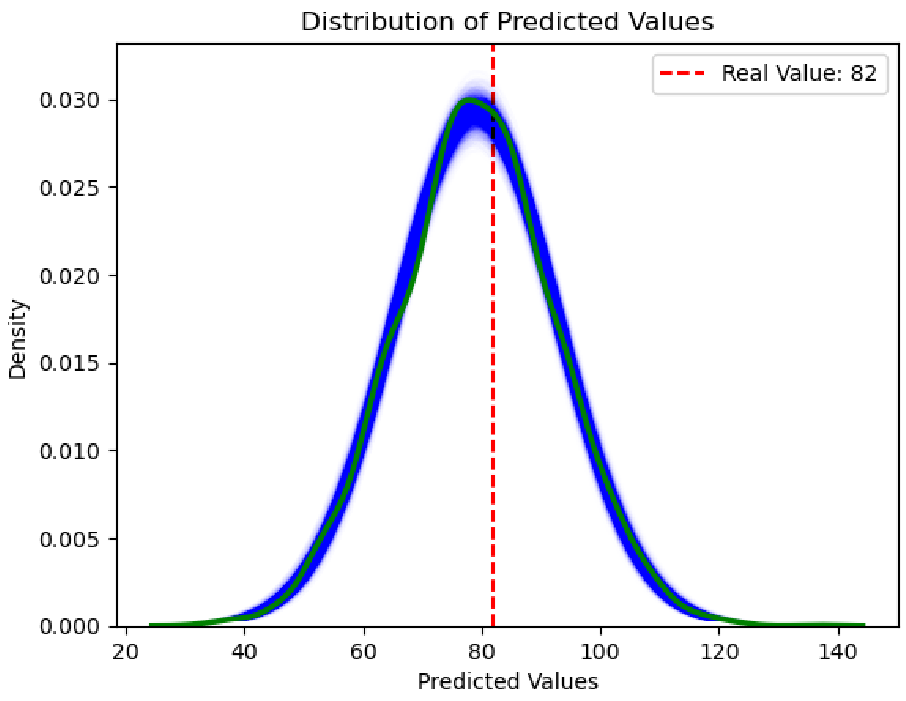

Figure 2.2: Bayesian Linear Regression results.

Now, I graphically represent the prediction in the following image, showing the expected value of the predictive distribution and the credibility interval (+/- 2σ) for July 11th (fig 2.4).

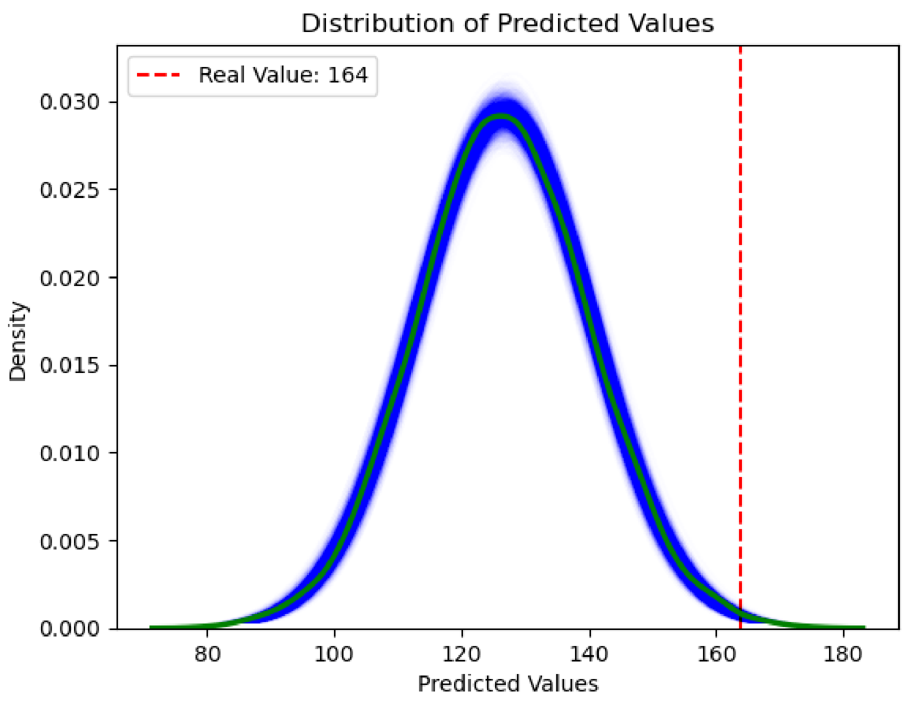

Figure 2.3: Bayesian Linear Regression results.

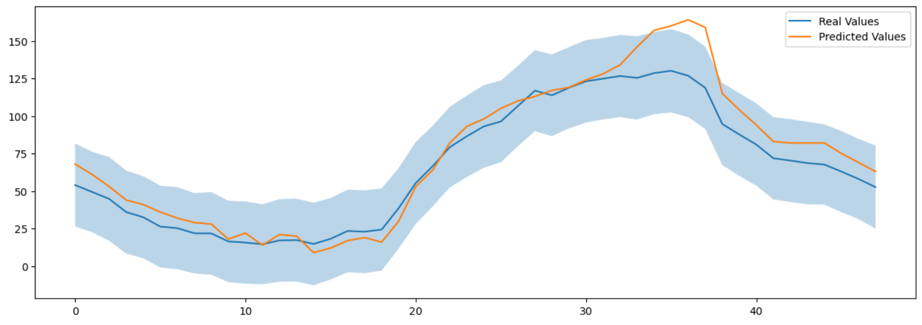

Figure 2.4: Prediction for July 11th with Credibility Interval.