GE-461: Introduction to Data Science

Project - Supervised Learning

Muhammad Abdullah S. Mulkana – 21801075

01.05.2022

Introduction

In this project, we are looking at basic supervised learning by implementing an Artificial Neural Network. For our case, we will consider two types of ANNs: one with a single output neuron (a simple linear regressor) and the other with a single hidden layer. We will explore how the different configurations perform on the 1 dimensional dataset given.

Methodology

In this part, our aim is to find the model that can fit the data most accurately. The guidelines that we have are that we will use sum of squared error as the loss function, all hidden units will have the sigmoid function as the activation function and we have to use stochastic gradient descent. The stochastic gradient descent algorithm states that in a single epoch, we randomly select one of the training sample, do the forward pass, compute the gradients and update the weights and biases [1]. In a single epoch we go over all instances of the training set in the aforementioned way.

In order to intuitively determine if a single linear regressor can be used as a model, we can plot the training data points.

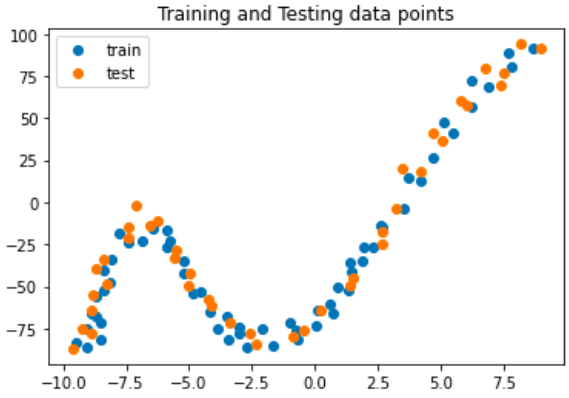

Figure 1: Training and Test data points plotted.

From Figure 1 we can see that the dataset in non-linear in nature. Therefore a single regressor with a linear activation function will not be sufficient to fit the data and therefore we need a more complex network with a hidden layer. By looking at the plot, we can see that our function is not very complex and thus a hidden layer with approximately 4-9 units may accurately fit the data. We will determine the model with the hidden layer next.

The best configuration to fit the model has a single hidden layer with 5 hidden units. Each hidden unit has a sigmoid activation function. For less hidden units, we get less of a fit to the data. Furthermore, the output neuron has a linear activation function. When applying this to the original data the configuration fits better to the positive x values but goes horizontal when x-value is negative. I tried normalizing the y values and that solved this issue and the model fits better to the data. While running the model, I noticed that the model is very sensitive to the learning rate. The best learning rate I determined is 0.03. I run the code for 4000 epochs but include an early stopping condition which stops training if the change in error is less than 2^{-6}. Weights and biases are initialized using a uniform distribution from -1 to 1.

Results

For the final configuration that we determined above, we run the model and get the following plots.

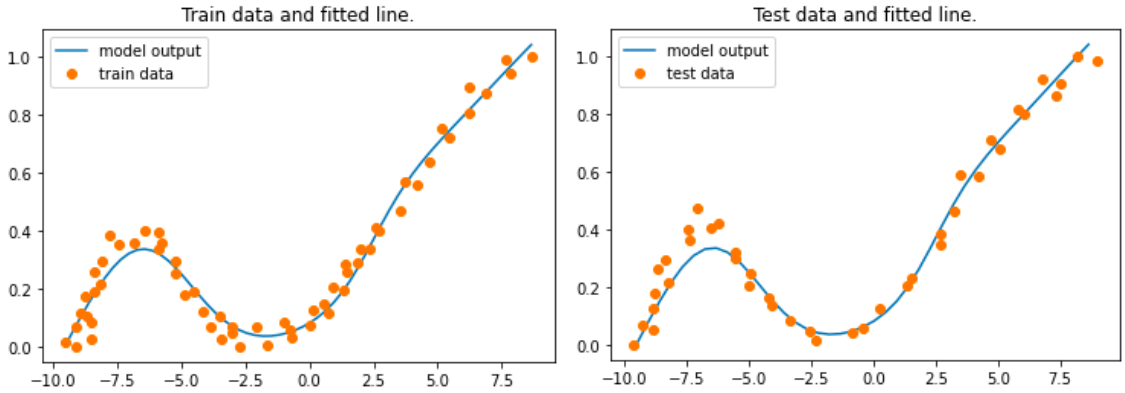

Figure 2: Plot of the normalized train and test data points and the fitted curve.

From Figure 2 we can see that the model fits both the training and test data. The fit on the training data is more accurate as compared to that of the test data but in both cases the model is able to capture the shape of the data with decent accuracy. The summary of the configuration is as follows:

ANN used: 5
Learning Rate: 0.03
Range of initial weights: [-1,1] (uniform distribution)
Number of epochs: 4000
When to stop: $loss_{t-1}$ - $loss_t < 2^{-6}$ (when the change in mse is less than 2^{-6})
Is normalization used: Yes, for y-values.
Training loss (averaged over training instances): 0.00206
Test loss (averaged over test instances): 0.00263

Note: When x-values are also normalized, the model does not fit as good to the data. Therefore we only normalize the y-values.

Next, we change the number of hidden units in order to determine how the complexity of the model effects its ability to fit our data. To do this, we first train a single regressor as before, and then an ANN with hidden units 2, 4, 8, 16, and 32.

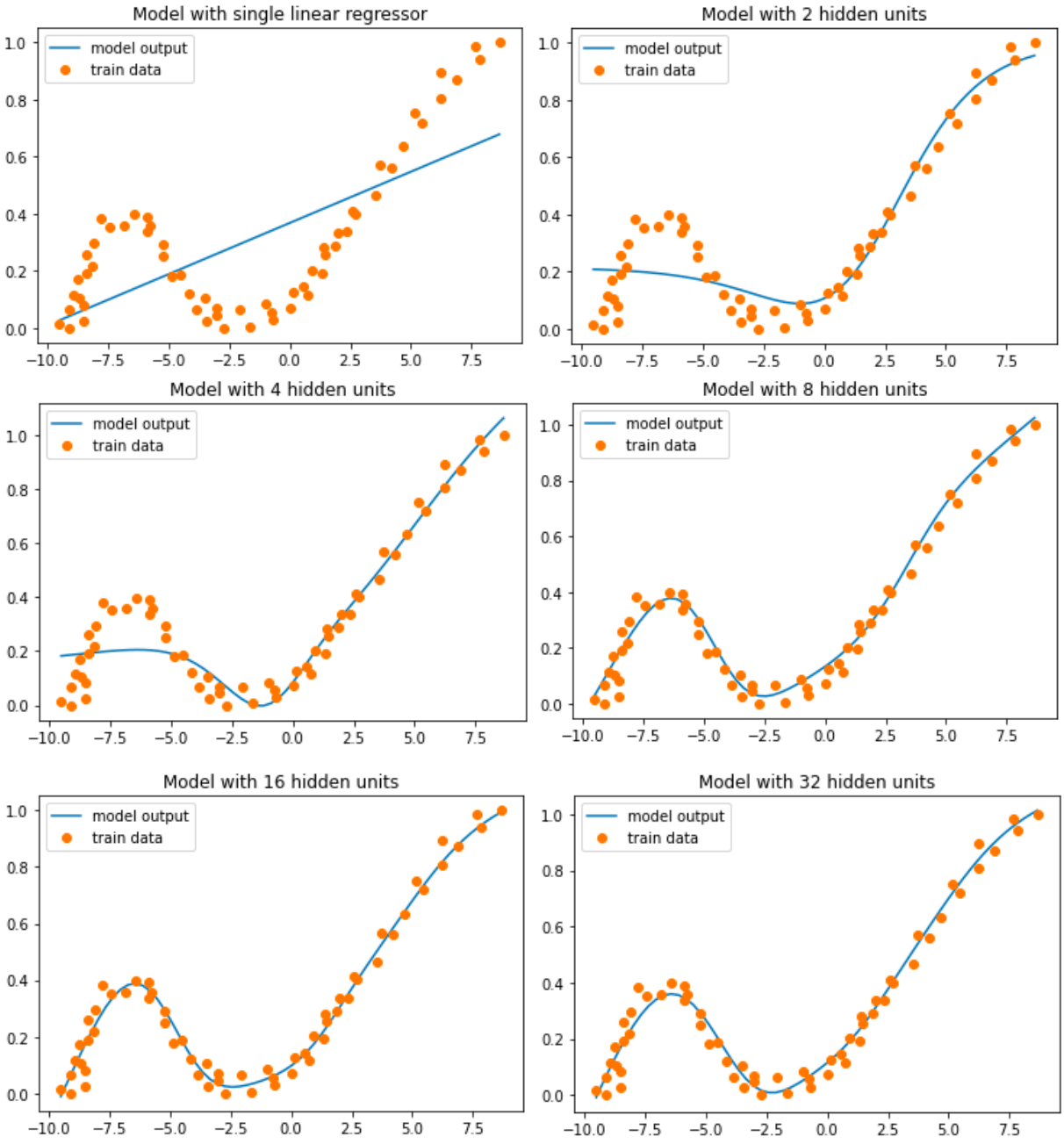

Figure 3: Plots of the training data and the fitting curve generated by the model.

Figure 3 shows us the training data points (y normalized) as orange dots and the model output for the same range of inputs. Table 1 reports the training and test loss mean and standard deviation for all model configurations whose plots are given above.

Table 1: The following table reports the mean and the standard deviation of the training and test loss.

Model	Training loss		Test loss	
	Mean	STD	Mean	STD
Single Regressor	0.03775	0.0324	0.0414	0.0353
ANN w/ 2 Hidden Units	0.00868	0.01229	0.0105	0.0173
ANN w/ 4 Hidden Units	0.00696	0.0104	0.00947	0.0155
ANN w/ 8 Hidden Units	0.00203	0.00343	0.00236	0.00342
ANN w/ 16 Hidden Units	**0.00163**	**0.00303**	**0.00215**	**0.00341**
ANN w/ 32 Hidden Units	0.00182	0.00261	0.00261	0.00404

From our results in Figure 3 we can see that with a single regressor, we always get a straight line (we are using a linear activation function, even if we were using logistic regression, i.e. sigmoid function, we would not be able to accurately fit the data). Using 2 or 4 neurons, we cannot fit the data that has high negative values. For 8, 16, and 32 hidden units, we can accurately fit the data. This is reflected in the loss values reported in Table 1 where we can see that both the training and test mean loss are decreasing with increasing complexity. Looking at the standard deviation we reach the same conclusion, i.e. as the model complexity in increased, the standard deviation of the losses decreases which means that our model is learning the data better. In the case of 32 hidden neurons we see some discrepancies where the mean of both the losses increases and the test loss standard deviation also increases.

Conclusion

In this project we experiment with different configurations of ANN models in order to determine how they would fit our data. We look at the importance of the learning rate, the weight initialization and the number of hidden units. We see how the models are very sensitive to the learning rate and the fact that the initial weight values may greatly impact our model performance. Lastly, we discuss the relationship between the complexity and the performance of the model and see that after a certain number of hidden units, we are able to satisfactorily fit our data and that making the model more complex does not give us much of an increase in performance.

References

[1] Aymen, "Stochastic Gradient Descent vs Online Gradient Descent," *StackExchange: Cross Validated*, Sept 2015. [Online]. Available: https://stats.stackexchange.com/questions/167088/stochastic-gradient-descent-vs-online-gradient-descent. [Accessed 1 May 2022].