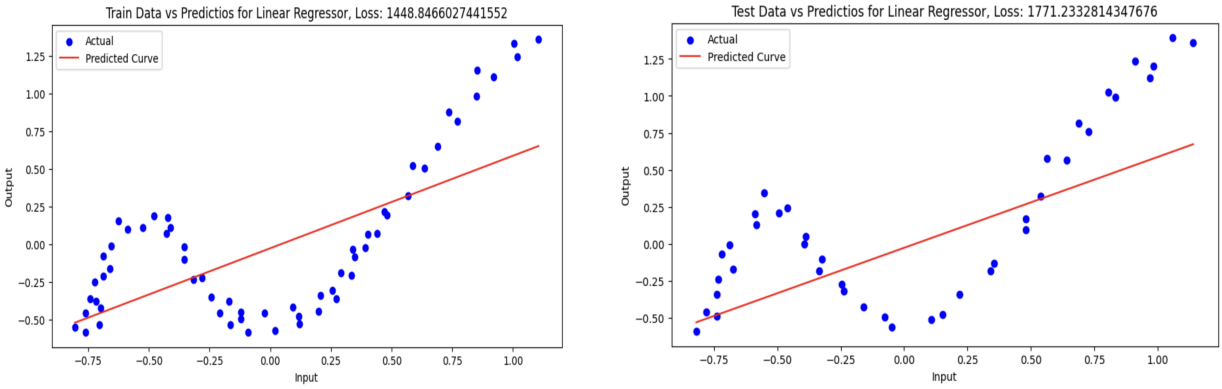
Bilkent University - GE461 Introduction to Data Science - Project 3 Supervised Learning - Report

Kaan Tek - 21901946

For the first part of the project, I wrote my own implementation of a simple ANN class that supports an ANN for linear regression (no hidden layer) and an ANN with a single hidden layer, without using any machine learning libraries.

Question a)

First, I experimented with a linear regressor (an ANN without any hidden layers) to see if it is enough to learn the parameters and obtain an acceptable result. Upon training the model and evaluating it on train and test data, I obtained the below plots:



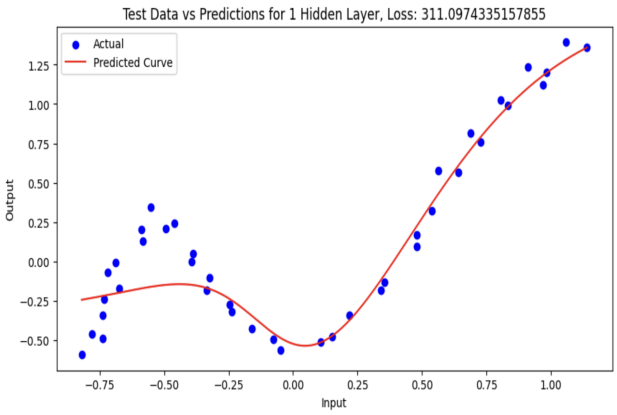
As can be seen from the above plots, the connection between the input and output is nonlinear. As a result, a linear regression model is unable to represent this nonlinear dynamic. The linear regressor fails to effectively represent the underlying data distribution, resulting in high SSE loss. Therefore, for this dataset, using hidden layers in our network is required.

In order to obtain a model that yields the best results for our dataset, I trained and evaluated models of different configurations. Specifically, I trained and evaluated a model for each combination of the below configurations:

```
# Configuration
input_size = 1
hidden_layers = [2, 4, 8, 16, 32]
learning_rates = [0.001, 0.01, 0.1]
epoch_settings = [5000, 10000, 50000, 100000]
initializations = [False, True]  # False for Xavier, True for Random
```

Therefore, I trained a total of 5 * 3 * 4 * 2 = **120** models, and noted the training loss, standard deviation for training set, test loss, and standard deviation for test set for all of these models. Below are my findings:

Hidden Units: As suggested in the part c of this assignment, I tried hidden units of size 2, 4, 8, 16, 32. The details of how these numbers affect the model's performance are discussed in more detail in part c. In order to determine the minimum number of hidden units required for our model to effectively learn, I also trained a model with only 1 hidden unit and evaluated it on the test dataset. From the plot below, it can be inferred that using only a single hidden unit is not really successful in capturing the patterns of the data. Therefore, I would use a minimum of 2 hidden units for this dataset, which can also lead to considerably good performance with certain configurations as can be seen in part c.



Learning Rates: For the learning rates, I mainly experimented with the alpha values 0.001, 0.01 and 0.1. Anything higher led to bad performance, and anything lower resulted in a long training time, without much contribution to the performance. Overall, for my configurations and the dataset, it can be said that the alpha value of 0.01 gave the best results in general.

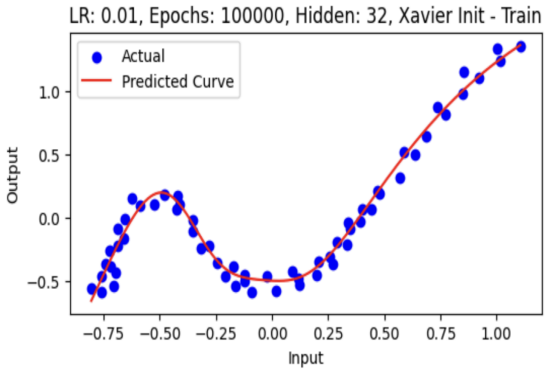
Weight Initialization: I tried two methods, randomly initializing the weights and using a normal Xavier initialization. Normal Xavier initialization is a well known method used to overcome the vanishing gradients problem, and since we are using a Sigmoid as the activation, this could have been an issue. This initialization method establishes the initial weights by selecting them from a Gaussian distribution with a mean of zero and a specific standard deviation defined by the formula $\sigma = \sqrt{\frac{2}{n_{prev} + n_{cur}}}$ in which n_{prev} indicates the number of units in the previous layer and n_{cur} indicates the number of units in the current layer. Overall, from my tests, using a Xavier initialization significantly improved the performance.

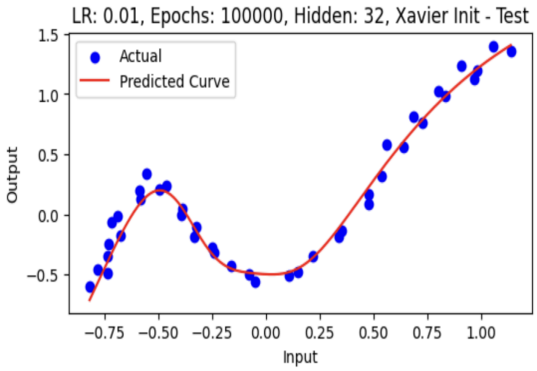
Number of epochs: I tried 5000, 10000, 50000, and 100000 epochs. Anything lower did not yield favorable results, and anything higher required too much training time. From my experiments, for most of the cases, the performance of the models kept increasing as I increased the epochs. Overall, using 100000 epochs yielded the best performance for most of the configurations. I did not use a stopping condition since I could still see an improvement in my models when using 100000 epochs, and if my machine allowed, I would have tested with 200000 epochs as well. Perhaps in that condition I might have not seen an improvement in the performance after some certain number of epochs, and in that case, I would have used the early stopping method, and terminated the training process after a certain point.

Normalization: I initially tried to train the models without any normalization applied, and obtained considerably low performance. The main reason behind this might have resulted from the vanishing gradients problem. Therefore, I applied normalization by subtracting the mean and then dividing by the max value, separately for the train and test dataset. Upon doing this, I noticed a significant increase in the performance, and continued training all the models on the normalized data.

Question b)

For this part, I trained and evaluated the performance of a model that has 32 hidden units, learning rate of 0.01, trained on 100000 epochs, and uses Xavier method for initialization. This model was one of the best performing models when I tried different configurations in the previous part. More details and the plot over the training and the test set are below:





ANN used (specify the number of hidden units): 32

Learning rate: 0.01

Range of initial weights: For each weight, a random value was selected from a normal distribution with mean 0 and standard deviation σ (σ is determined from the formula given in the previous page)

Number of epochs: 100000

When to stop: When max epoch is reached

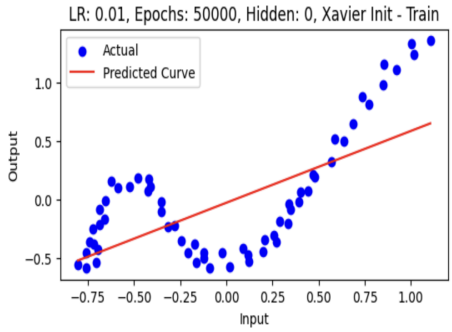
Is normalization used: Yes

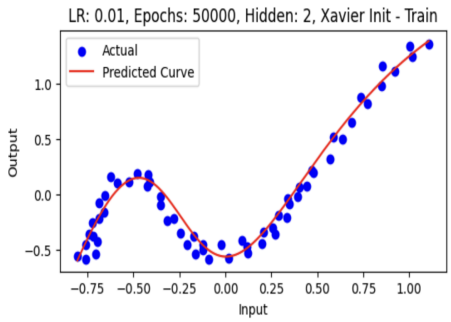
Training loss (averaged over training instances): 60.2529 (SSE)

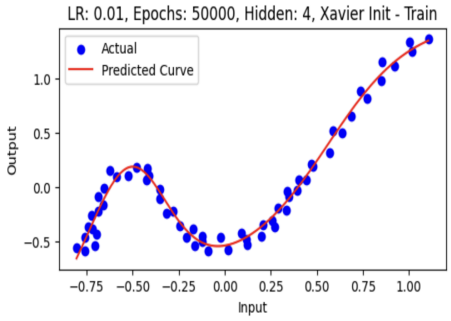
Test loss (averaged over test instances): 85.7450 (SSE)

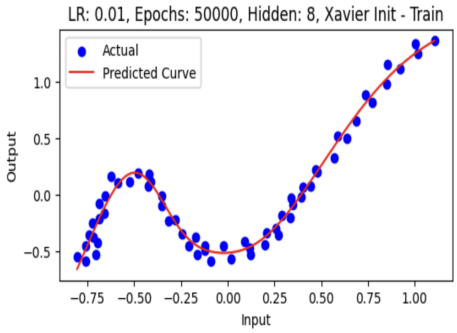
Question c)

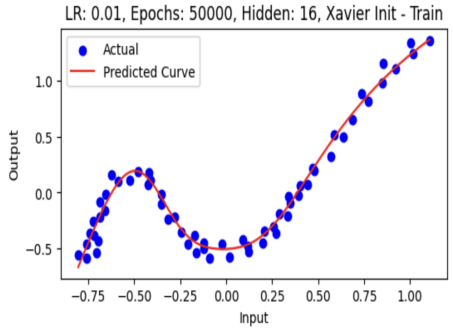
Below are the plots for the performances of the models that use 6 different number of hidden units (0, 2, 4, 8, 16, 31), alpha value of 0.01, 50000 epochs, and Xavier initialization:

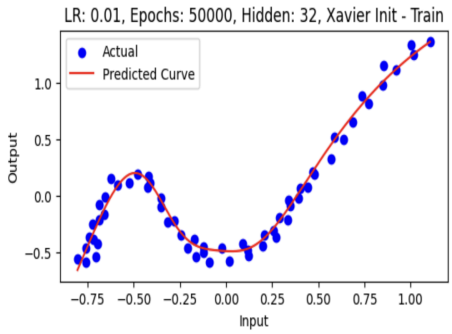












Configuration	Train SSE Loss	Train Std	Test SSE Loss	Test Std
0 hidden units	1448.8466	1351.2800	1771.2333	1604.8085
2 hidden units	95.1967	139.4146	109.7300	139.9444
4 hidden units	58.6930	100.8040	82.1962	135.3791
8 hidden units	57.8878	102.4545	82.9415	133.8103
16 hidden units	61.0024	107.0224	85.4658	128.7914
32 hidden units	62.3797	106.2808	86.3040	129.2252

The above plots and table indicate that the number of hidden units used significantly determines its ability to learn. First of all, it can be inferred that a linear regressor fails to capture the non-linearity of our data. Using 2 hidden layers seems to be enough to capture this pattern, however, one could also see from the table that starting from 4 hidden units, there is significant improvement in the performance. It can also be argued that increasing the model's complexity might not be needed for our dataset, since models with 4 and 8 hidden units yield slightly better performances than those with 16 and 32. Note that as a result of using stochastic gradient descent, slightly different results could be obtained, but still, one would not need to use a high complex network to achieve a great performance for this dataset.