GE 461 - Project 3 - Supervised Learning

Zübeyir Bodur - 21702382
Computer Engineering
Bilkent University
Ankara, Turkey
zubeyir.bodur@ug.bilkent.edu.tr

***Abstract*—There is no abstract for this project**

***Index Terms*—artificial neural networks, supervised learning**

I. Part 1

A. Question 1: How does the number of convolutional layers influence the learning power of deep neural networks?

In lower layers, the network extracts very low level details, such as corners, edges and blobs, as if it is an interest point detector such as SIFT, or SURF.

However, these low level features are not enough. As the number of layers increase, the network realizes mid level features, like parts of the image/ object being detected.

If we have enough layers, the network will be able to extract high level features, such as an object template, or a class template.

B. Question 2: Naïve implementation of very deep convolutional neural networks may not learn from the data. What would be the main reason(s) of the aforementioned issue? How can you handle such problems?

As our neural network gets deeper, in our gradient descent algorithm, the gradient starts to vanish towards the end. Therefore, even though we think we are extracting more detailed features specific to our training data, if our gradient vanishes, we will end up having no information in the output layer.

In addition, we also have an exploding gradient problem, as our convolutional neural network becomes deeper. That is, similar to the previous problem, the gradient may explode if the largest eigenvalue in the weights matrix is larger than 1.

To handle this issue, we can use Rectified Linear Unit (ReLU) as our activation function, instead of sigmoid. In fact, with the discovery of using different activation functions such as ReLU, the accuracy of deep neural networks increased significantly. In addition, we can also use Residual Neural Networks to solve this issue.

C. Question 3: Do we observe issues similar to the ones indicated in part 1(b) for basic (vanilla) version of recurrent neural networks? Please explain/discuss, provide an example, and indicate possible solutions.

Yes, we observe such problems in Vanilla Recurrent Neural Networks (Vanilla RNN). Since we have only one hidden vector that maps each input to an output for a time frame, with less complexity, our gradient will vanish if the largest eigenvalue is less than one, and it will explode if the largest eigenvalue is more than one, which is a quite sensitive outcome. Therefore, in Vanilla RNN's, we will be loosing information about the previous signals, and may not make a good estimation if those previous signals were actually important.

One way to solve this is using Artificial Neural Networks that use Long-Short Term Memory (LSTM) idea from psychology. This way, we will be dealing with vanishing and exploding gradient problems at the same time, as we will be keeping the important information from the ”far” past, and all necessary information from the ”close” past.

II. Part 2

The data set for this project is plotted in Fig. 1.

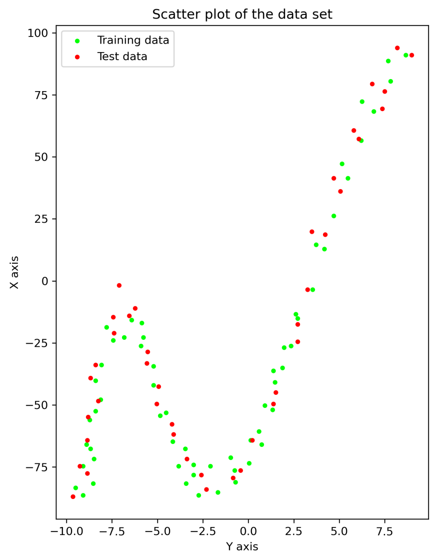

Fig. 1. The data set

A. Step 1

For this project, M-point moving averager was used on the losses of each epochs recorded to determine whether the losses have converged. If the M-point moving average of losses were larger than the last loss, I stopped the training. The number of epochs can be chosen as high as possible, say 100,000, however this dynamic stopping criteria will be powerful enough so that the training will stop if the accuracy does not improve anymore.

For normalization, min-max normalization was used, so that the range of inputs were mapped from $[min, max]$ to $[0, 1]$, where min is the smallest input, and max is the largest input. Normalization was not used on weights nor the output. By doing normalization, the speed of the training increased significantly, and the same performance was achieved using much less number of units; if we were to aim the same performance as in Table I, we would need at least 512 units. However, 16-32 units are enough if we use normalization.

In addition, for the rest of the project, the initial weights were chosen from a Gaussian distribution from a range $[0, 1]$.

For the learning rate, depending on the number of units, a value around $5 \cdot 10^{-4}$ is a good value for learning rate. If we were to set it too high, network does not learn. If we set it too low, the gradient vanishes, hence the network can not learn.

B. Step 2

A configuration that learns the data set provided is given in Table I:

TABLE I
A CONFIGURATION THAT LEARNS

Hyperparameters	**Values**
Number of Units/Layer	24
Range of Initial Weights	[0, 1]
Epochs	$100,000$
Learning Rate	$8 \cdot 10^{-4}$
M	2
Is Normalization Used	YES
Training Loss/Training Instance	50.78
Test Loss/Test Instance	83.35

The plots for predicted values of the training and the test set are given in Fig. 2 and 3.

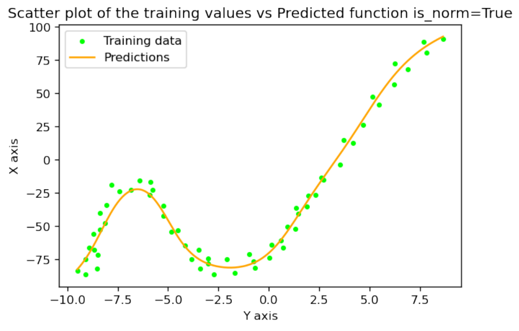

Fig. 2. Predictions for the training set

C. Step 3

The following hyperparameters in Table II are tuned to observe the relation between the complexity and the accuracy of each model.

The following parameters in Table III were fixed during the experiment, because number of units and dynamic stopping conditions were enough to meet the needs of this experiment:

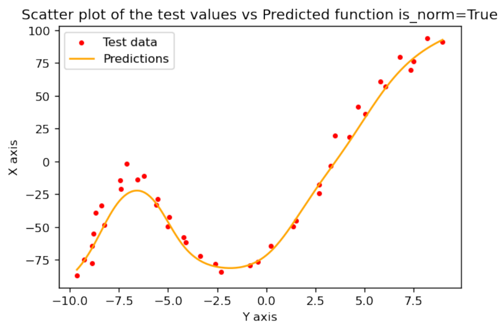

Fig. 3. Predictions for the test set

TABLE II
HYPERPARAMETERS USED IN PART C OF THE ASSIGNMENT

Number of Units
0 - Linear Regressor
2
4
8
16
32

Hence, the results were obtained are plotted in Fig. 4 and 5:

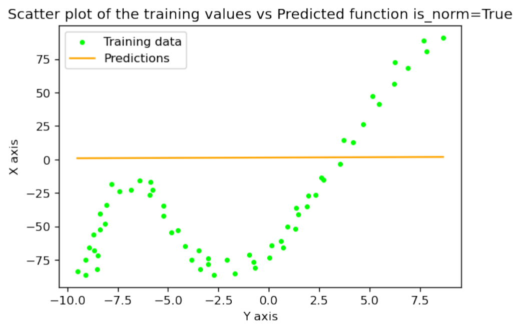

Fig. 4. Results of the training for part c when a linear regressor ANN is used.

In the assignment, it was asked to evaluate the loss averaged over training and test instances, and its standard deviation. In this paper, those terms are referred as "Mean Training/Test Error" and "Standard Deviation of Training/Test Error". It should be also noted that, the "errors" are squared before their mean and variance is computed, since we are using sum of

TABLE III
PARAMETERS FIXED IN PART C OF THE ASSIGNMENT

Parameters	**Values**
Range of Initial Weights	$[0, 1]$
Epochs	$100,000$
Learning Rate	$5 \cdot 10^{-4}$
M	3
Is Normalization Used	YES

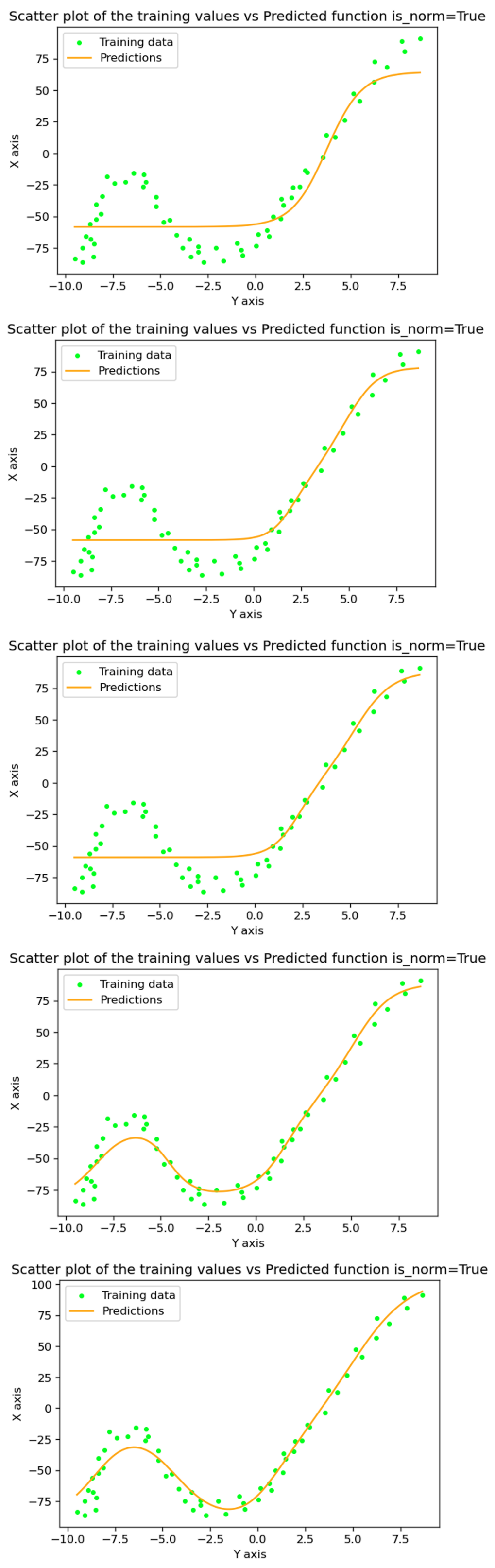

Fig. 5. Results of the training for part c when ANN's with single hidden layer is used. The plots correspond to 2, 4, 8, 16, 32 hidden units, from top to bottom.

squared errors as our loss function.

In addition to these plots in Fig. 4 and 5, each of those trainings have a list of errors, for each X points in those trainings. From these errors, we can derive a mean and standard deviation values of those errors, which will emphasize the contribution of complexity in ANN's. These values are shown in Table IV and V below:

TABLE IV
MEAN ERRORS FOR TRAINING AND TEST PREDICTIONS.

Setting Used	Mean Training Error	Mean Test Error
Linear Regressor	3486.42	3237.26
2 Hidden Units	381.83	527.67
4 Hidden Units	339.55	458.34
8 Hidden Units	337.37	462.00
16 Hidden Units	92.43	143.85
32 Hidden Units	91.26	124.20

TABLE V
STANDARD DEVIATIONS OF ERRORS FOR TRAINING AND TEST PREDICTIONS.

Setting Used	Std. Dev. of Training Errors	Std. Dev. of Test Errors
Linear Regressor	2513.68	2572.52
2 Hidden Units	444.21	695.02
4 Hidden Units	455.99	707.46
8 Hidden Units	470.49	726.46
16 Hidden Units	142.11	233.22
32 Hidden Units	144.09	195.11

D. *Discussion*

It can be observed that, for a polynomial regression problem, using hidden layers is required to make any meaningful predictions. In addition, the number of units, also contribute to the accuracy of the model. If the number of units are above a threshold, the network learns a portion of the data each time. However, after achieving necessary threshold, increasing number of units do not contribute to the learning that much as before.

In other words, after learning the minimum number of units, other parameters should be tuned, and the model's complexity should be minimum so that the accuracy of the model is maximum.