GE 461 – Introduction to Data Science - Project 3 Report

Oğuz Altan – 21600966 - EEE

Electrical and Electronics Engineering Department, Bilkent University, 06800 Ankara, Turkey

1. Part I

a) As the number of convolution layers increase, convolutional neural network (CNN) can learn more abstract patters using the convolutions of convolutions, and so on. However, increasing the number of these layers also increases parameters to deal with, therefore to adjust these parameters better, network needs more training data. Therefore, because of this issue, increasing the convolutional layer increases the learning power up to a point. After this point, namely a threshold size of convolutional layers, adding more convolutional layer does not increase the learning power.

b) Naïve implementation of very deep convolutional neural networks may not learn from the data. One of the reasons that after each convolution operation of filter and the image, the pixels around the boundaries of the image get lost due to the shrinking of the image. Therefore, spatiality reduction occurs and we lose information. As image getting smaller and smaller after each convolution operation, at the end of this very deep CNN structure, we may end up with a poor information of the image. To prevent this, we can use zero padding on the image and keep the special information and the dimension of the image does not get smaller, it stays same. Another reason for the very deep CNN does not learn is that, after having done the forward pass, when doing backpropagation, gradient vanish may occur. As we calculate gradients from the end of the network to start point, we multiply activation values and the result of the multiplications get smaller and smaller, consequently becomes nearly zero at some point in the backpropagation process, namely gradient signals die. Thus, network may not learn due to this problem in backpropagation. For instance, as we look at CIFAR-10 plain nets, it can be seen that as the number of convolutional layer increases, the error rate also increases. This shows the incapability of very deep CNN. ResNet has solved this problem of having very deep CNN, by using short cuts and residual connections. By using this new network structure, after this implementation, we can see that as the network gets more and more deep, it also becomes more and more successful and error rate gets smaller. In ResNet, gradients flow over aforementioned short cuts without vanishing.

c) For the very deep vanilla recurrent neural networks, we observe similar issues, as just described for CNN. If a very long sequence is applied, RNN cannot work nicely, again gradients cause problem. When applying back propagation through time (BPTT), in each time step, gradient have possibility of vanishing (signal dying) or exploding. Because of these problems, vanilla RNN cannot learn long time dependencies. The solution for vanishing gradients is to use Long Short Term Memory (LSTM) structure. LSTM have a selection mechanism, unlike for vanilla RNN, and this selection mechanism allow LSTM to be able to select what data to forget, what do filter, choose the windows size and dictionary and so on. This selection mechanism works using gates in LSTM. Comparing vanilla RNN and LSTM, in the BPTT, gradient flows much longer, more time steps, and dies very late, in LSTM, compared to vanilla RNN. For the gradient explosion issue in vanilla RNN, we can use gradient clipping i.e. if, in the BPTT, gradient becomes higher than a preset threshold value, the gradient is equalized to this threshold to prevent gradient explosion.

2. Part II - Implementing ANN for Regression

a and b) Finding the Configuration, Training and Plots

First, we visualize the training and test data sets to understand what they look like as scatter matrices

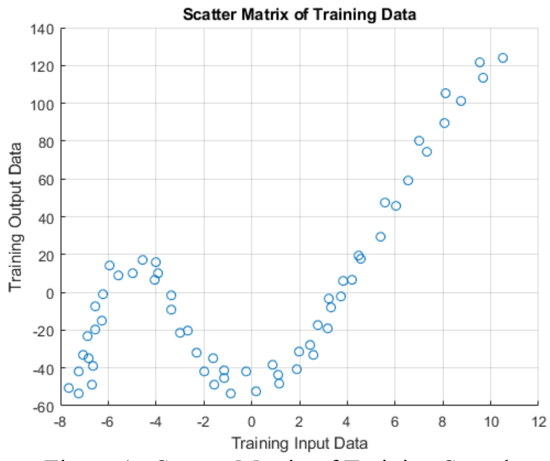

Figure 1: Scatter Matrix of Training Samples

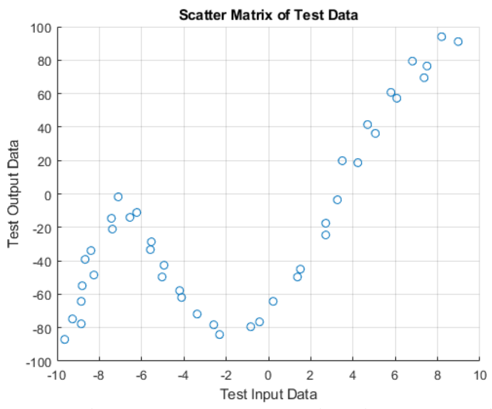

Figure 2: Scatter Matrix of Test Samples

Observing these two scatter plots, we can see that the distribution of the samples and data points are not in a linear fashion, rather the shape of distribution looks like a polynomial nonlinear curve. Therefore, using an ANN with a single hidden layer is better than a linear regressor ANN, for regression of our dataset. In our implementations, we use sum of the squared errors as loss function, sigmoid as activation function to define the hidden units and stochastic learning algorithm. The output plots of linear regressor ANN, as well as the loss function curve over iterations, are given below:

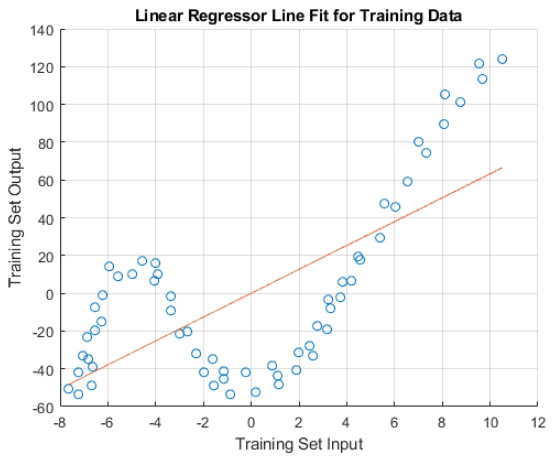

Figure 3: Linear Regressor Line Fit for Training Data

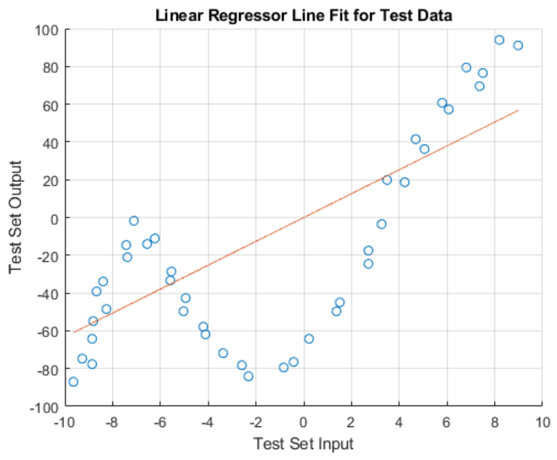

Figure 4: Linear Regressor Line Fit for Test Data

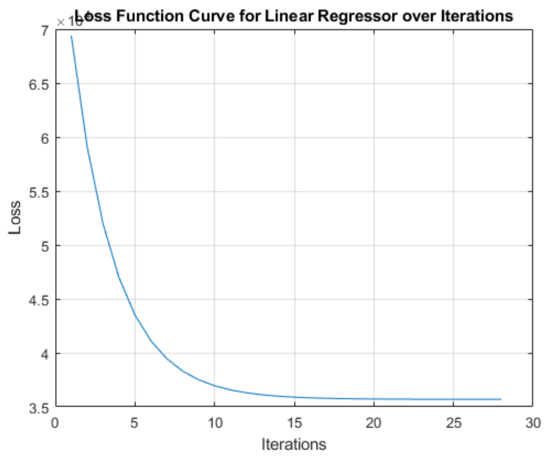

Figure 5: Loss Function for Linear Regressor

It can be seen that linear regressor is not very successful in fitting dataset and thus, prediction. Although the ANN that is preferred for our data set is obviously ANN with a single hidden layer, the configuration of the linear regressor ANN is given for the sake of completeness:

ANN: Linear Regressor
Learning Rate: 0.0001
Range of Initial Weights: Standard Normal Distribution multiplied with Standard Deviation = 0.01
Number of Epochs: Maximum number of epochs is 1000
When to Stop: Algorithm stops if the difference between two loss values for two consecutive iterations is smaller than threshold value stop_diff = 1
Is normalization used: Yes, mean normalization
Training Loss (averaged over training instances): 532.85
Test Loss (averaged over test instances): 594.89

In this ANN with no hidden layer as linear regressor, this configuration gives more or less successful outputs. Initializing weights is done by choosing values randomly from Standard Normal Distribution multiplied with a standard deviation chosen as 0.01. Stopping is used to prevent unnecessary computation, as well as overfitting. The algorithm stops if the difference between two loss values for two consecutive iterations is smaller than a threshold called stop_diff having the value 0.001. Normalization is used, this process increases the success of the network and affect the learning process for this application in a good way. To normalize, the mean of data is subtracted from the original data.

The ANN that is more successful on our dataset is the one with a single hidden layer. The output plots of this ANN, as well as the loss function curve over iterations, are given below:

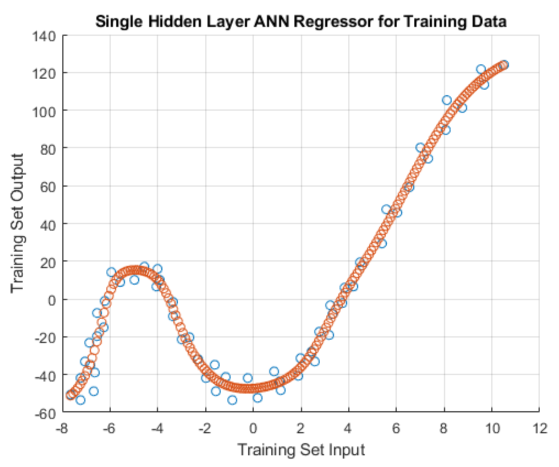

Figure 6: Predictions of Single Hidden Layer ANN Regressor for Training Data

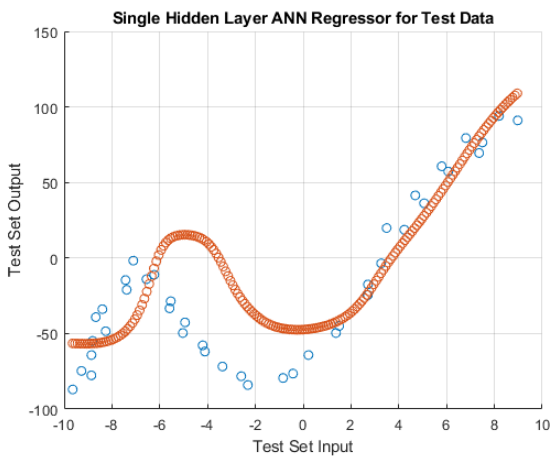

Figure 7: Predictions of Single Hidden Layer ANN Regressor for Test Data

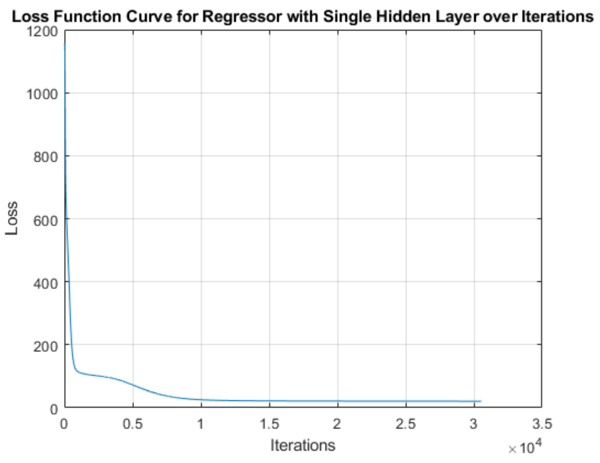

Figure 8: Loss Function Curve for ANN with Single Hidden Layer

In the estimation output plots, namely Figure 6 and 7, the red curve represents the predicted points, output of the regressor ANN with single hidden layer, put on top of original training and test data set. It can be seen that this network is very successful and better than a linear regressor ANN is. It can do the regression of the data more successful.

The configuration of this ANN is given below:

ANN: ANN with a single hidden layer, with 100 hidden layer neurons
Learning Rate: 0.0001
Range of Initial Weights: Standard Normal Distribution multiplied with Standard Deviation = 0.001
Number of Epochs: Maximum number of epochs is 10000
When to Stop: Algorithm stops if the difference between two loss values for two consecutive iterations is smaller than threshold value stop_diff = 1
Is normalization used: Yes, mean normalization
Training Loss (averaged over training instances): 11.34
Test Loss (averaged over test instances): 19.87

In this ANN with a single hidden layer as regressor, this configuration results in very successful estimations. Initializing weights is done by choosing values randomly from Standard Normal Distribution multiplied with a standard deviation chosen as 0.01. Stopping is used to prevent unnecessary computation, as well as overfitting. The algorithm stops if the difference between two loss values for two consecutive iterations is smaller than a threshold called stop_diff having the value 0.0001. Normalization is used, this process increases the success of the network and affect the learning process for this application in a good way. To normalize, the mean of data is subtracted from the original data. Losses are averaged over number of corresponding data set (training or test) instances.

c) In this part, we analyze the complexity of the ANN by implementing and after, observing different number of hidden layer neurons in ANN, which are 2, 4, 8, 16 and 32 hidden layer neurons.

For each ANN, we provide the regression and estimation plots for the training data set. We draw curves for each of the hidden units and put all of these curves on the same plot. This plot is given on the next page.

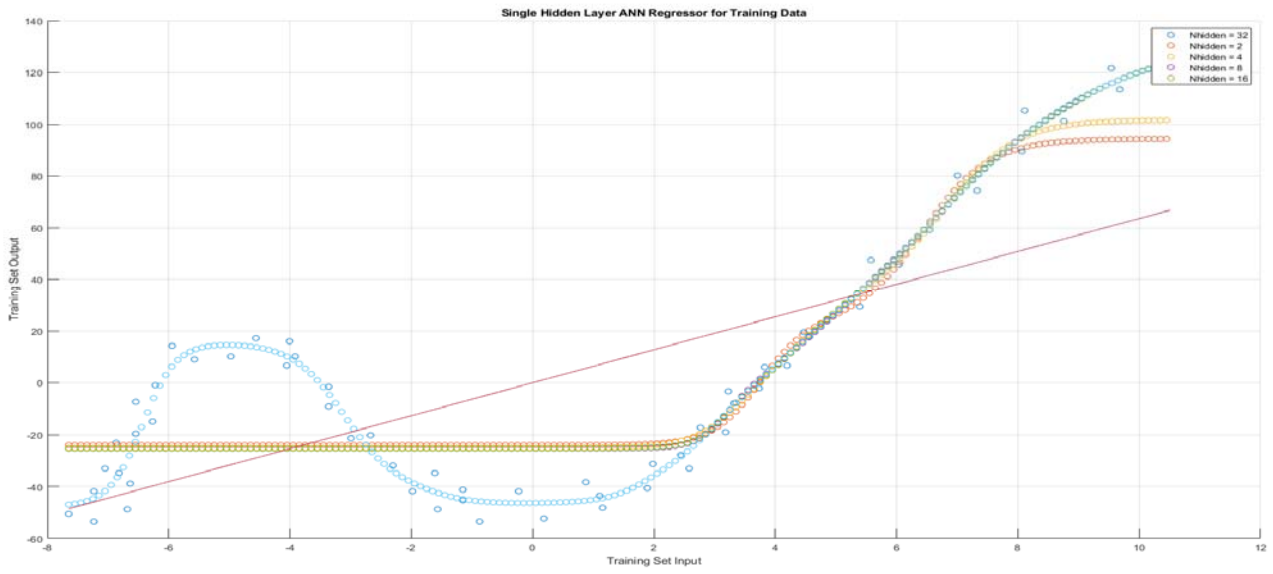

Figure 9: Regression of Single Hidden Layer ANN Regressor for Different Hidden Layer Neuron Size

	Linear Regressor
Train Loss	532.85
Test Loss	594.89

	Hidden Neurons = 2
Train Loss	211.01
Test Loss	246.97

	Hidden Neurons = 4
Train Loss	170.42
Test Loss	181.17

	Hidden Neurons = 8
Train Loss	164.31
Test Loss	187.31

	Hidden Neurons = 16
Train Loss	160.48
Test Loss	195.42

	Hidden Neurons = 32
Train Loss	20.80
Test Loss	22.59

	Standard Deviations
Train Loss Std	170.95
Test Loss Std	190.51

Analyzing the plot, we can see that regression gets better and better for higher number of hidden layer neurons. For instance, for the hidden layer with 32 neurons, the regression is nearly perfect, curve is nearly fit. The reason is, as the number of hidden layer neurons increase, features that are more abstract and patterns that hidden layer can learn. Therefore, for our case of regression of points that seems to be distributed with the shape of polynomial functions, hidden layer helps to do regression better by understanding and track the pattern of the data distribution. Looking at the train and test losses, as well as corresponding standard deviations, we can see that loss decreases as number of hidden layer neurons increase, namely complexity of the ANN increase. In addition, standard deviation for train loss is lower than the one for test loss that is expected because using training data for regression gives more successful results compared to using test data, as the network learns regression using the training data. After fine-tuning of network parameters, training, and testing process, we obtain the plot above, Figure 9, which shows the regression curves for different network architectures. Apparently, using 32 hidden layer neurons gives best results by having best regression. However, we should not assume that more hidden layer neurons always correspond to better results. In fact, increasing the hidden layer neurons means need for more work power and computational cost, meaning that network needs more training data to have successful results. For basic ANNs such as ours in this project, the training and testing data are 1D and therefore we do not deal with very complex ANN. Yet again, increasing the number of hidden layer neurons can increase the performance and success up to a limit point, after that, computational cost becomes more and more prominent.