Introduction to Machine Learning and Artificial Neural Networks CS 554: HOMEWORK 3 REPORT

Amin Deldari Alamdari
S033174

I. INTRODUCTION

The goal of this homework is to fit a model to given custom data set that generalizes the problem and predict the new inputs' outcome. In this section, single-layer perceptron, multi-layer perceptron, mean square error, **tanh** activation and back-propagation are explained in detail.

A. Single-Layer Perceptron

A single-layer perceptron which is shown in Figure 1, is a type of artificial neural network that is used for binary classification. It consists of a single layer of neurons, each of which is connected to an input and an output. The neurons are connected to each other in a way that allows them to learn from the data they receive. It can have multiple input and output units, but it has not hidden units between them.

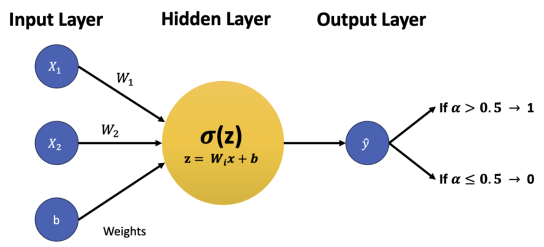

Fig. 1: Single Layer Perceptron.

Single-layer perceptron has one layer of computational node and it produces the linear combination of the wights, bias and the given inputs as follows:

$$\hat{y} = f\left(\sum_{i=1}^{n} \omega_i + b\right) \quad (1)$$

where $\hat{y}$ is the output of the nodes, f is the selected activation function which maps the linear output to a non-linear output, ω_i is the ith weight of the network, x_i is the ith feature of the given input and b is the bias of the layer.

B. Multi-Layer Perceptron

A multi-layer perceptron which is shown in Figure 2, is a type of artificial neural network that is used for more complex tasks, such as multi-class classification. It consists of multiple layers of neurons, each of which is connected to an input and an output. The neurons are connected to each other in a way that allows them to learn from the data they receive. It is very similar to single-layer perceptron, but instead of one computational layer of nodes it has multiple layers that are called hidden layers. It computes the linear combinations between inputs to hidden layers and from hidden layers to output layer.

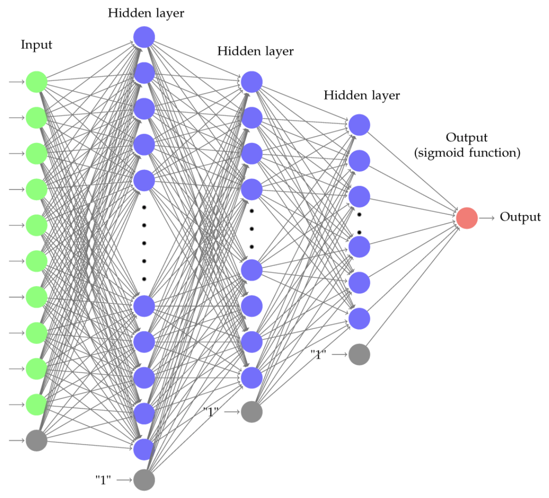

Fig. 2: Multi-Layer Perceptron.

C. Activation Function

Activation functions are employed in artificial neural networks to decide the output of a neuron when given an input or a set of inputs. These functions are mathematical equations that determine the output of a neural network based on the weighted sum of the inputs and the bias. The activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input. There are several activation functions is available, but for this work sigmoid function 1.3 has been used as follows:

D. Loss Function

Loss functions are used to evaluate machine learning models by measuring the difference between the predicted output and the actual output. They help assess model performance and identify areas for improvement. Various loss functions exist for different tasks. In this work, we use the **Mean Squared Error (MSE)** loss function. MSE is commonly used in regression problems. It calculates the average of the squared differences between the predicted values and the actual target values. The formula for MSE is given by:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2}$$

where N is the number of instances in the dataset, $\hat{y}_i$ is the ith predicted value, and y_i is the ith actual value.

E. Backpropagation for Regression with MSE

Backpropagation is a key algorithm used to train neural networks by minimizing the error between predicted and actual outputs. It works by propagating the error calculated from the output layer backward through the network, adjusting the weights to reduce the loss. In this homework, we use the Mean Squared Error (MSE) loss function in combination with the `tanh` activation function for hidden layers. This process enables the network to iteratively learn optimal parameters by updating weights in the direction that reduces the error. Backpropagation efficiently computes gradients using the chain rule, making it suitable for deep or layered architectures. The update rule for any weight w during training is:

$$w_{\text{new}} = w_{\text{old}} - \alpha\frac{\partial E}{\partial w} \tag{3}$$

where w_{new} is the updated weight, w_{old} is the current weight, α is the learning rate, and $\frac{\partial E}{\partial w}$ is the gradient of the loss function with respect to the weight.

In this assignment, the weights are updated for both the hidden and output layers using gradients derived from the MSE loss and the derivative of the $\tanh$ activation function.

II. Derivation of Update Rules for Tanh Activation

We use a multi-layer perceptron with one hidden layer and the `tanh` activation function. The output layer is linear, and we use mean squared error (MSE) as the loss function. Let:

- *Input:* $x \in \mathbb{R}^{N\times 1}$
- *Hidden layer:* $z = xW_0 + b_0$, $a = \tanh(z)$
- *Output:* $\hat{y} = aW_1 + b_1$
- *Loss:* Mean Squared Error (MSE): $L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$

Forward Pass

$$z = xW_0 + b_0$$
$$a = \tanh(z)$$
$$\hat{y} = aW_1 + b_1$$

Loss:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Backward Pass

1) Compute output error:
$$\delta_1 = \hat{y} - y$$
2) Gradients for output layer:
$$\frac{\partial L}{\partial W_1} = \frac{1}{N}a^T\delta_1, \quad \frac{\partial L}{\partial b_1} = \frac{1}{N}\sum \delta_1$$
3) Backpropagate to hidden layer using $\tanh'(z) = 1 - \tanh^2(z)$:
$$\delta_0 = (\delta_1 W_1^\top)\cdot(1 - \tanh^2(z))$$
4) Gradients for input layer:
$$\frac{\partial L}{\partial W_0} = \frac{1}{N}x^T\delta_0, \quad \frac{\partial L}{\partial b_0} = \frac{1}{N}\sum \delta_0$$
5) Update rules:
$$W_1 \leftarrow W_1 - \alpha\frac{\partial L}{\partial W_1}$$
$$b_1 \leftarrow b_1 - \alpha\frac{\partial L}{\partial b_1}$$
$$W_0 \leftarrow W_0 - \alpha\frac{\partial L}{\partial W_0}$$
$$b_0 \leftarrow b_0 - \alpha\frac{\partial L}{\partial b_0}$$

III. Results

This section presents the experimental results for the single-layer perceptron (SLP) and multi-layer perceptrons (MLPs) with 2, 4, and 8 hidden units, trained on a small regression dataset using batch gradient descent and the Mean Squared Error (MSE) loss.

A. Model Output After Training

Figures 3 to 6 show the function approximations learned by each model. The red curve represents the network's predicted output after convergence, and the blue points represent the training data. As the number of hidden units increases, the networks are able to fit more complex patterns.

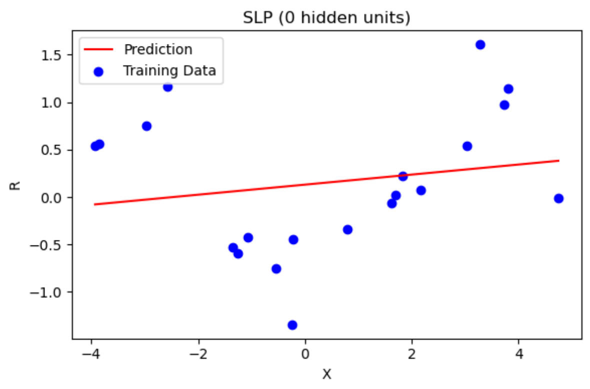

Fig. 3: SLP output after convergence (0 hidden units)

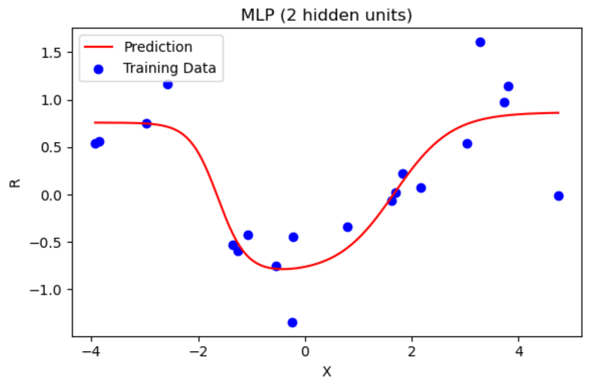

Fig. 4: MLP output after convergence (2 hidden units)

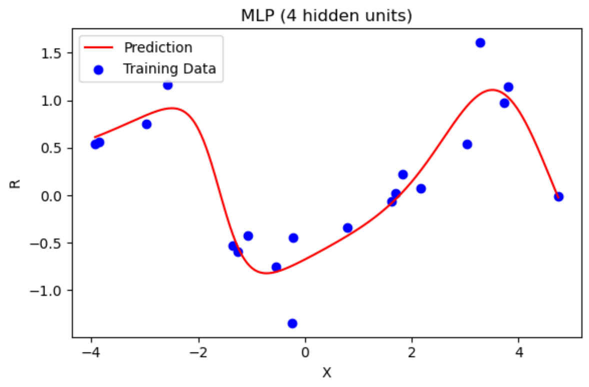

Fig. 5: MLP output after convergence (4 hidden units)

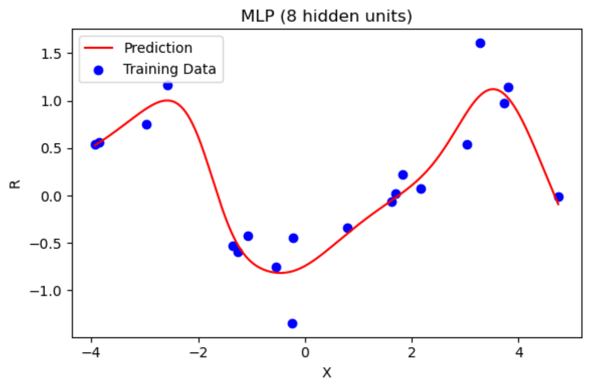

Fig. 6: MLP output after convergence (8 hidden units)

B. Training and Test MSE Over Epochs

Figures 7 to 10 display the training and test loss (MSE) across training epochs. These plots demonstrate how each model converges during training. The MLPs with more hidden units generally reduce training loss faster but may exhibit higher test error due to overfitting.

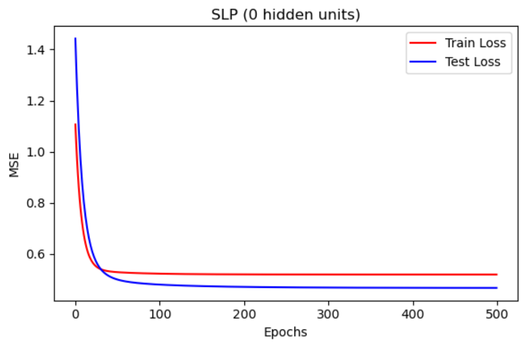

Fig. 7: MSE vs Epoch for SLP (0 hidden units)

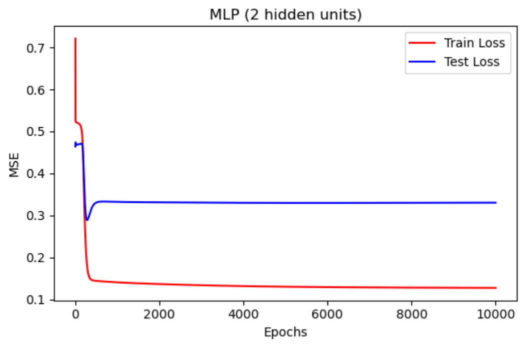

Fig. 8: MSE vs Epoch for MLP (2 hidden units)

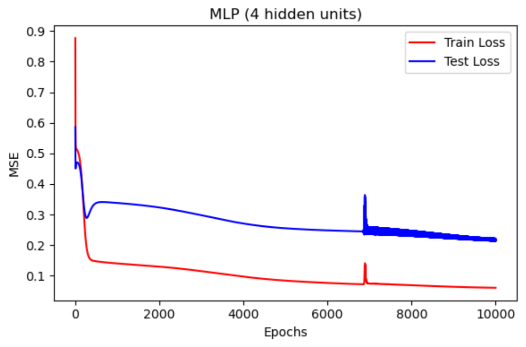

Fig. 9: MSE vs Epoch for MLP (4 hidden units)

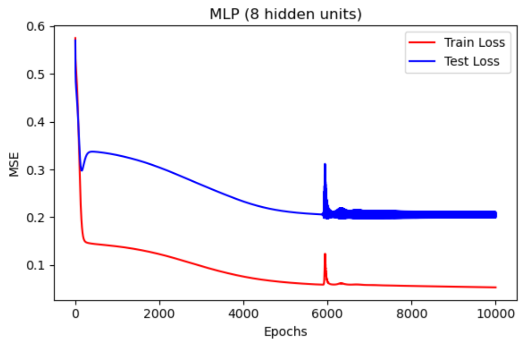

Fig. 10: MSE vs Epoch for MLP (8 hidden units)

C. Model Complexity vs Error

Figure 11 compares the final training and test losses across different network complexities. As expected, adding hidden units improves training accuracy but may lead to overfitting beyond a certain point.

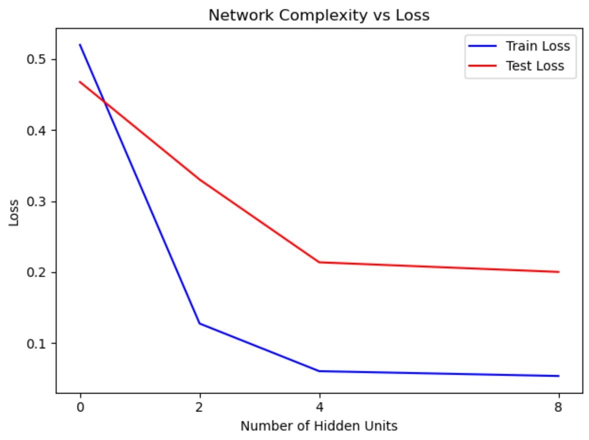

Fig. 11: Final MSE vs Network Complexity (number of hidden units)

IV. Conclusion

In this assignment, we implemented and evaluated both single-layer and multi-layer perceptrons for a regression task using a small dataset. We trained four different neural network architectures: a single-layer perceptron and multi-layer perceptrons with 2, 4, and 8 hidden units. Training was done using batch gradient descent with the Mean Squared Error (MSE) loss function and `tanh` activation in hidden layers.

Our results show that increasing the number of hidden units improves the network's ability to model complex patterns in the training data. However, this also increases the risk of overfitting, as seen by the gap between training and test loss in deeper models. The single-layer perceptron provides a baseline with limited capacity, while the multi-layer models demonstrate the power and flexibility of deeper architectures in fitting nonlinear relationships.