Assignment-05: Artificial Neural Networks

Zain Abbas, CS-131, 08/08/25

Abstract

To accurately classify plants in the nursery, we designed a multi-layer neural network that accepts user input and predicts the plant type. The model was trained using Fisher's Iris dataset, which contains measurements of various Iris species. The network can be evaluated using either test data or manual user input. Structurally, the neural network is based on a linear model with two layers and incorporates activation functions to introduce non-linearity and improve learning performance.

Introduction

Linear regression models provide a foundational approach for tackling multi-class classification problems. In this project, we explore the use of a multi-layer neural network to classify different species of the Iris plant. The Iris dataset, produced by Fisher, is a benchmark in pattern recognition. It primarily includes three species: Iris Setosa, Iris Veriscolor, and Iris Virginica. Each sample contains four features: sepal length and width and petal length and width.

Our primary objective is to classify each Iris species based on these input features. Initially, we designed a two-layer neural network grounded in a linear regression model. To enhance robustness and improve model performance, we incorporated a nonlinear activation function known as ReLU (Rectified Linear Unit). Activation functions introduce non-linearity into neural networks, enabling the model to better capture complex patterns in the data. By leveraging these functions, our network can accurately predict the class of each test sample.

Methodology

I. *Preprocessing the data*

To train and test our model effectively, we split the raw data into training, validation, and test sets. Specifically, 60% of the data was used for training, 20% for validation, and the remaining 20% for testing. After splitting the data, we applied one-hot encoding to the true labels. One-hot encoding converts categorical variables into a binary format by creating a separate column for each category. A value of 1 indicates the presence of a category, while 0 indicates its absence. For example:
Iris Setosa: [1, 0, 0]
Iris Versicolor: [0, 1, 0]
Iris Virginica = [0, 0, 1]

II. *Analyzing raw data*

We visualized the raw data to examine species distribution and feature patterns. Figure 1 presents two plots: one for sepal features (left) and one for petal features (right), each displaying the three Iris species. The left plot focuses on sepal length and width. *Iris Setosa* (purple) is clearly distinguishable from the other species, characterized by a higher sepal width and shorter sepal length. In contrast, *Iris Virginica* and *Iris Versicolor* show a high degree of overlap in their sepal measurements. The right plot illustrates petal length and width. *Iris Setosa* again stands out, with significantly smaller petal dimensions compared to the other two species, which exhibit more variation and overlap.

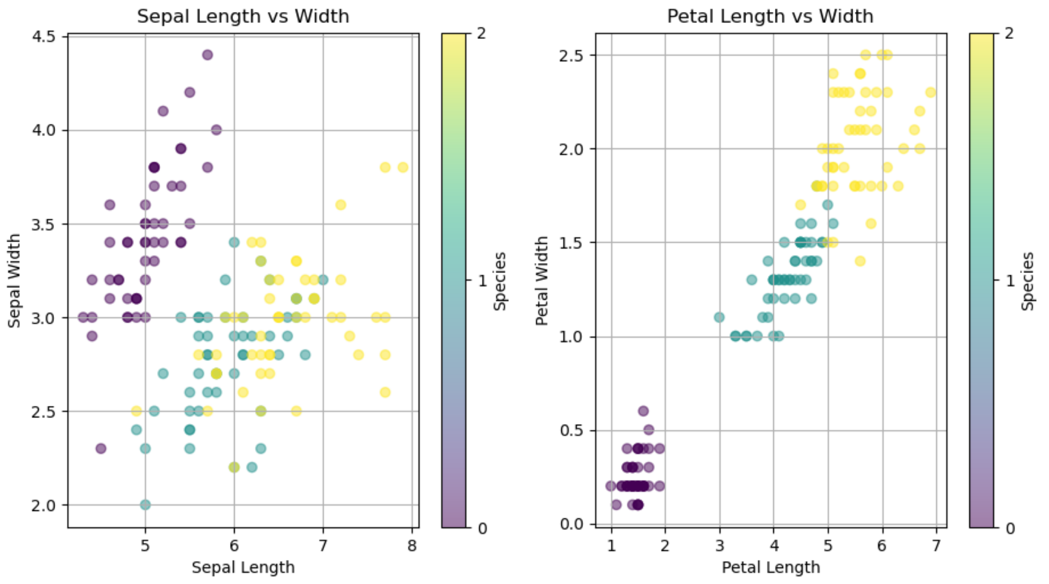


Figure 1: The above plot shows Iris species as a function of sepal and petal length and width

To better visualize the feature distributions for each species, we created a correlation matrix. Figure 2 displays this matrix, illustrating relationships among sepal and petal features across the three Iris species. The diagonal cells contain histograms that represent the aggregated distribution of each individual feature—sepal length, sepal width, petal length, and petal width. These histograms help identify typical value ranges for each feature.

The off-diagonal cells show scatter plots that reveal correlations between pairs of features. Notably, *Iris Virginica* and *Iris Versicolor* exhibit strong correlations between sepal length and sepal width. Conversely, features that show low or no correlation can be particularly useful for improving model performance, as they may provide unique, non-redundant information.

Feature Correlation Matrix for Iris Dataset

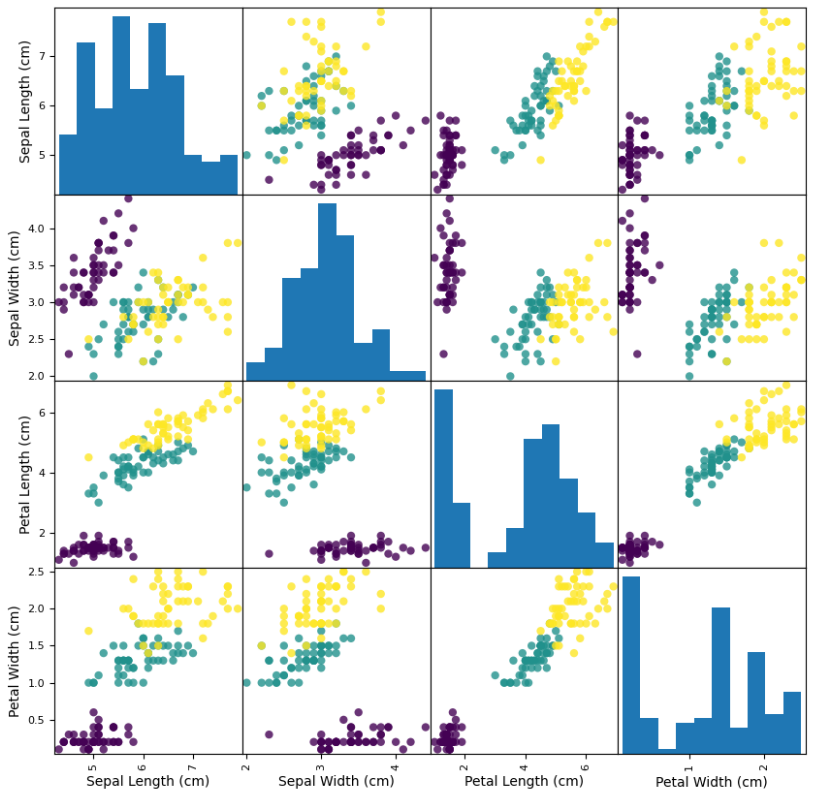


Figure 2: The above plot shows the correlation plot (off-diagonal) and histograms (diagonal).

III. *Neural Network Architecture*

The neural network architecture contains multi-layers linear regression functions to make the performance of the model robust.

• Input layer: dimension = 4 (features).

• Hidden layer: H neurons (e.g. H = 10) + activation function

• Output layer: 3 neurons + softmax.

a. Input layer:

$y^1 = W1 * x + b1$

$W1 \rightarrow 10x4$

Neuron 1:

$$y_1^1 = W_{11} . x_1 + W_{12} . x_2 + W_{13} . x_3 + W_{14} . x_4 + \text{b1}$$

Neuron 2:

$$y_2^1 = W_{21} . x_1 + W_{22} . x_2 + W_{23} . x_3 + W_{24} . x_4 + \text{b2}$$

Neuron 3:

$$y_3^1 = W_{31} . x_1 + W_{32} . x_2 + W_{33} . x_3 + W_{34} . x_4 + \text{b3}$$

$$\vdots$$

Neuron 10:

$$y_{10}^1 = W_{101} . x_1 + W_{102} . x_2 + W_{103} . x_3 + W_{104} . x_4 + \text{b10}$$

To simplify the input layer, we define each variable as matrix.

y^1 = W1 * x + b1   where W1 → 10x4   b1 → 10x1

b. *Activation function: ReLU*

It is piece-wise linear function that outputs the input directly if it is positive, otherwise it outputs zero.

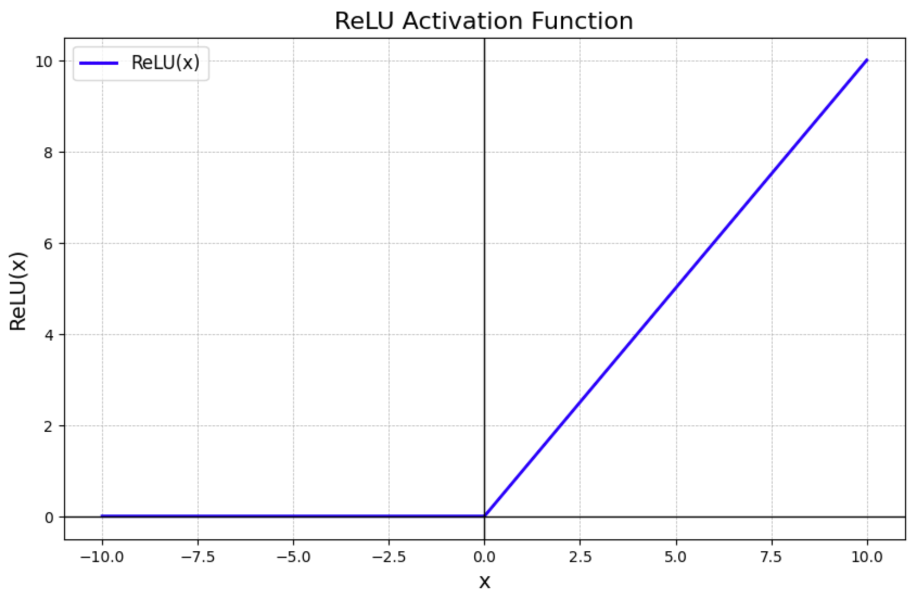


Figure 3: The above plot shows ReLU function. The function is zero for negative values.

We feed the input layer into activation function, to learn more complex data patterns and model intricate relationships between features. ReLU's ability to output zero for negative inputs introduces sparsity in the network, meaning that only a fraction of neurons activates at any given time. This can lead to more efficient and faster computation. Also, ReLU is convenient in backpropagation as its derivative is either 0 (negative input) or 1 (when input is positive)

c. *Layer 2: Output layer (3 Neurons)*

$y^2 = W_2\, h + b2$ where W2 → 3x10, b2 → 3x1

Neuron 1:

$$y_1^2 = W_{2,11} \,.\, h_1 + W_{2,12} \,.\, h_2 + W_{2,13} \,.\, h_3 + W_{2,14} \,.\, h_4 \ldots . W_{2,1,10} \,.\, h_{10} + \mathrm{b21}$$

Neuron 2:

$$y_2^2 = W_{2,12} \,.\, h_1 + W_{2,22} \,.\, h_2 + W_{2,23} \,.\, h_3 + W_{2,24} \,.\, h_4 \ldots . W_{2,210} \,.\, h_{10} + \mathrm{b22}$$

Neuron 3:

$$y_3^2 = W_{2,13} \,.\, h_1 + W_{2,23} \,.\, h_2 + W_{2,33} \,.\, h_3 + W_{2,34} \,.\, h_4 \ldots . W_{2,310} \,.\, h_{10} + \mathrm{b23}$$

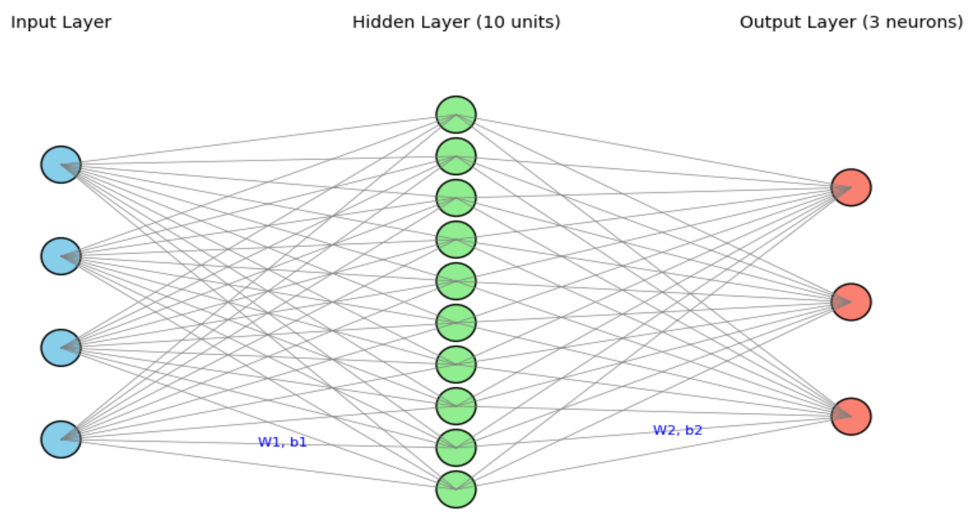


Figure 4: The above diagram shows neural network architecture. It starts with 4 inputs (4 features), 10 hidden neurons and 3 neurons for each species.

d. *__Backpropagation (Using Stochastic Gradient Descent)__*

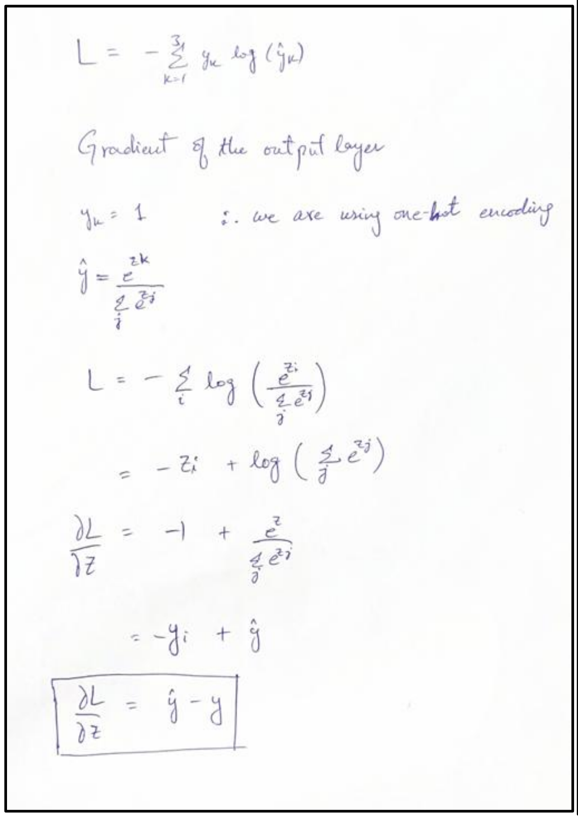
After creating the forward pass, we use backpropagation to update the weights and biases of the model. It calculates the gradient of the loss function with respect to each weight and bias in the network. It uses the chain rule to propagate errors backward from the output layer to the input layer.

The gradients tell us how to adjust the weights to reduce the error. Without backpropagation there is no systematic way to improve the model performance. In our project, we used cross entropy loss function to measure the loss of our model.

Loss function: Cross-Entropy

$$L = -\sum_{k=1}^{3} y_k \log(\hat{y}_k)$$

__Gradient of the output layer and input layer:__



$$L = -\sum_{k=1}^{3} y_k \log(\hat{y}_k)$$

Gradient of the output layer

$y_k = 1$ ∴ we are using one-hot encoding

$$\hat{y} = \frac{e^{z_k}}{\sum_j e^{z_j}}$$

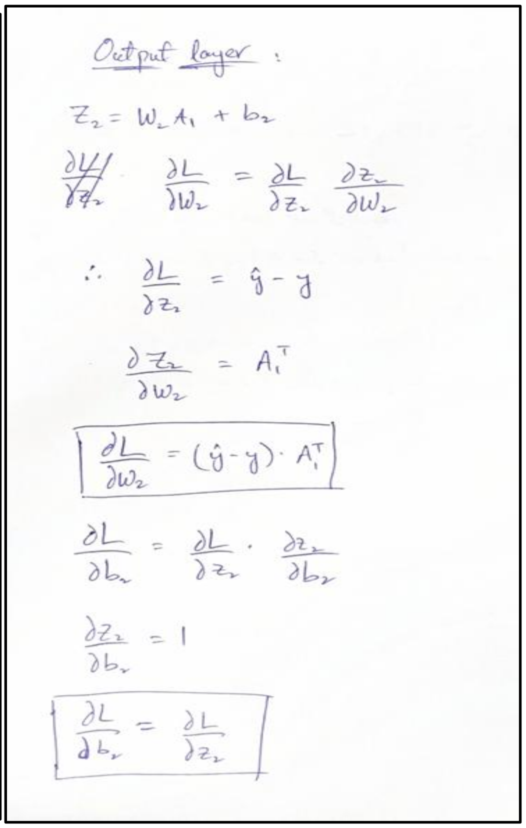
$$L = -\sum_i \log\left(\frac{e^{z_i}}{\sum_j e^{z_j}}\right)$$

$$= -z_i + \log\left(\sum_j e^{z_j}\right)$$

$$\frac{\partial L}{\partial z} = -1 + \frac{e^{z}}{\sum_j e^{z_j}}$$

$$= -y_i + \hat{y}$$

$$\boxed{\frac{\partial L}{\partial z} = \hat{y} - y}$$



Output layer :

$$Z_2 = W_2 A_1 + b_2$$

$$\cancel{\frac{\partial L}{\partial Z_2}} \quad \frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial W_2}$$

$$\therefore \frac{\partial L}{\partial Z_2} = \hat{y} - y$$

$$\frac{\partial Z_2}{\partial W_2} = A_1^T$$

$$\boxed{\frac{\partial L}{\partial W_2} = (\hat{y} - y) \cdot A_1^T}$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial b_2}$$

$$\frac{\partial Z_2}{\partial b_2} = 1$$

$$\boxed{\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial Z_2}}$$

$$\therefore Z_1 = W_1 x + b_1$$

$$\frac{\partial L}{\partial A_1} = \frac{\partial L}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial A_1}$$

$$\frac{\partial Z_2}{\partial A_1} = W_2^T$$

$$\frac{\partial L}{\partial A_1} = (\hat{y} - y) \cdot W_2^T$$

$$\boxed{\begin{aligned} \frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial Z_1} \cdot x^T \\ \frac{\partial L}{\partial b_1} &= \frac{\partial L}{\partial Z_1} \end{aligned}}$$

$$\boxed{\frac{\partial L}{\partial Z_1} = \frac{\partial L}{\partial A_1} \cdot \mathrm{Relu}(Z)'}$$

The above diagram shows the gradient descent for each parameter and model layer. We used a learning rate of 0.01 to gradually guide the learning process and reduce the risk of overfitting.

e. <u>*Implementation of the algorithm*</u>

```
1. Initialize W1 ,W2, b1, b2 with small random values.
2. Loop for each epoch:
    • Shuffle training data.
    • For each mini-batch:
        (a) Compute forward pass
        (b) Compute loss and accumulate.
        (c) Compute backprop
        (d) Update parameters: (e.g. SGD).
    • Record training loss and accuracy.
3. Evaluate on test set.
4. Plot loss and accuracy curves vs. epochs
```

Results

To evaluate the accuracy and generalization capability of our model, we monitored both the training and validation loss and accuracy across each epoch. These metrics were visualized through performance curves, which provided insight into the model's learning behavior over time.

Training and validation:

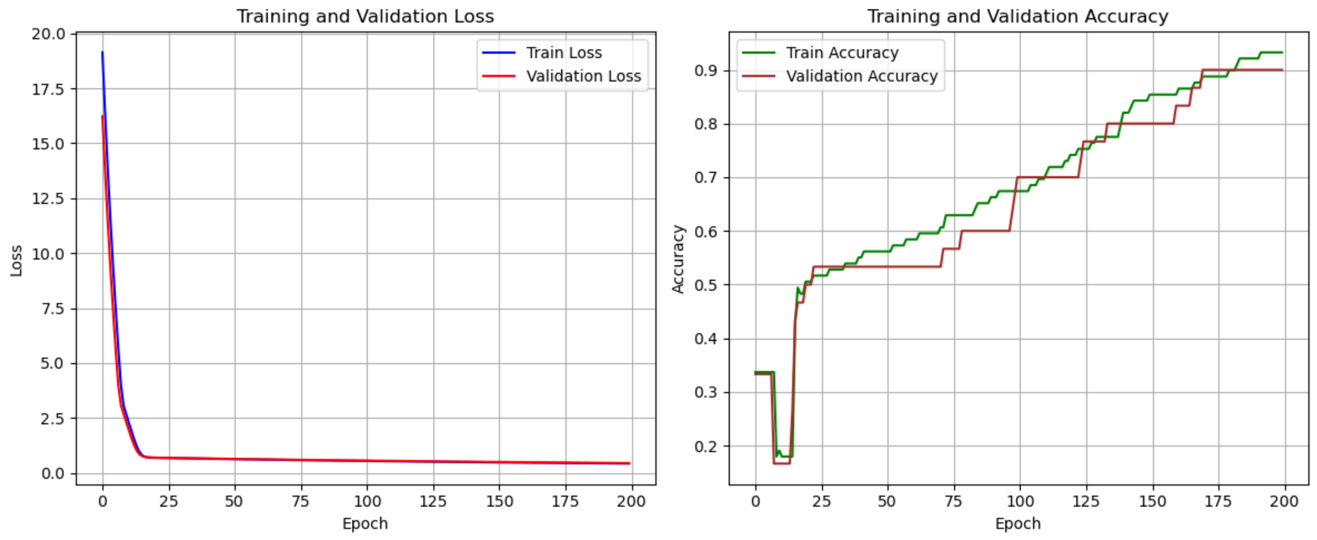
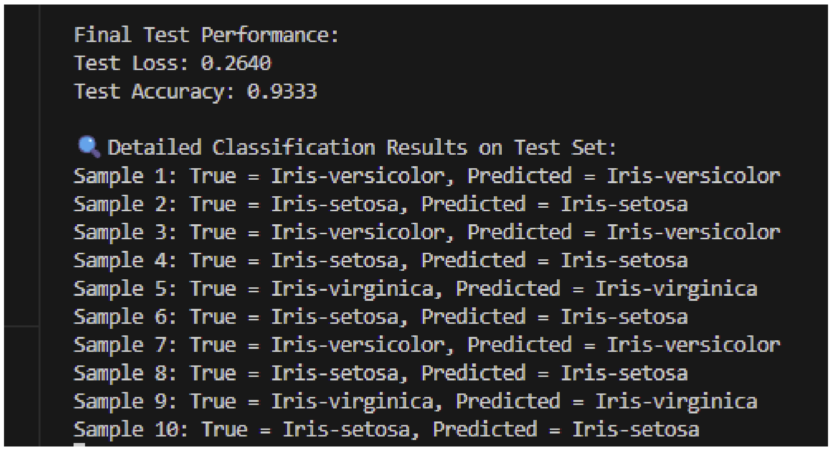


Figure 5: The above plot shows the loss and accuracy curve for training and validation data.

The plot on the left illustrates the loss curves for both the training and validation datasets. Initially, the loss was quite high due to randomly initialized weights and biases, with the training loss starting at 19 and the validation loss at 15. As training progressed, backpropagation enabled the model to update its parameters effectively. Over the course of 200 epochs, with a learning rate of 0.01, the loss was successfully minimized to approximately 0.22.

Similarly, the plot on the right presents the accuracy curves. At the outset, accuracy was low for both datasets due to the randomized parameters. However, as the model began learning and identifying relevant features, accuracy improved significantly. The model continued to gain accuracy until it plateaued at around 93%. Importantly, overfitting was avoided by carefully selecting appropriate hyperparameters throughout the training process.

Test data result



```
Final Test Performance:
Test Loss: 0.2640
Test Accuracy: 0.9333

🔍 Detailed Classification Results on Test Set:
Sample 1: True = Iris-versicolor, Predicted = Iris-versicolor
Sample 2: True = Iris-setosa, Predicted = Iris-setosa
Sample 3: True = Iris-versicolor, Predicted = Iris-versicolor
Sample 4: True = Iris-setosa, Predicted = Iris-setosa
Sample 5: True = Iris-virginica, Predicted = Iris-virginica
Sample 6: True = Iris-setosa, Predicted = Iris-setosa
Sample 7: True = Iris-versicolor, Predicted = Iris-versicolor
Sample 8: True = Iris-setosa, Predicted = Iris-setosa
Sample 9: True = Iris-virginica, Predicted = Iris-virginica
Sample 10: True = Iris-setosa, Predicted = Iris-setosa
```

Figure 6: The above figure shows test data results

Figure 6 shows the test data results for 10 samples. The model accurately predicted the test data and classified each species.

Extra experiment:

For the extra experiment, we used Sigmoid as the activation function and examined the performance of our model.

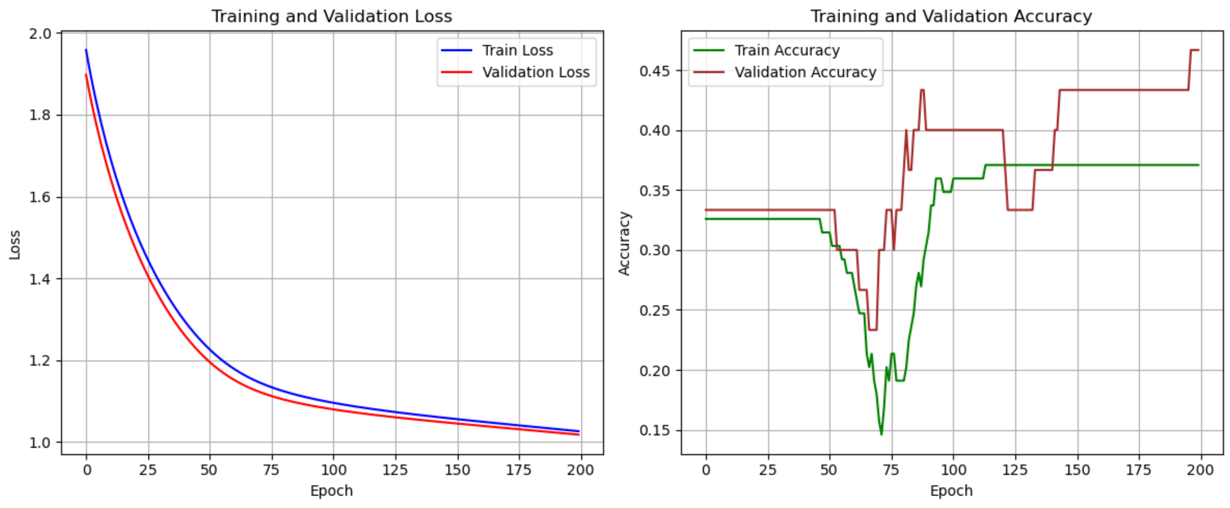
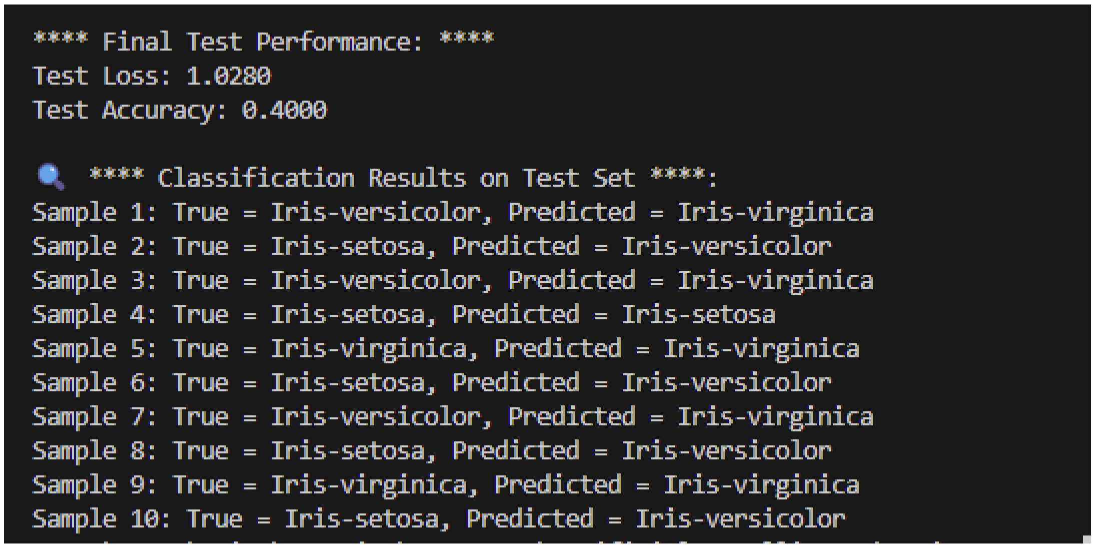


Figure 7: The plots show model accuracy and loss using sigmoid function

Figure 7 illustrates the model's accuracy and loss when using the sigmoid activation function. Initially, both training and validation losses are relatively low compared to those observed with the ReLU activation. However, the loss only decreases to around 1 even after 200 epochs. The model's accuracy begins at approximately 0.34 and rises modestly to 0.45 for the validation set and 0.36 for the training set. This limited improvement suggests that the sigmoid activation function may hinder learning efficiency. During backpropagation, gradients tend to shrink, especially when inputs are large—causing the sigmoid outputs to saturate near 0 or 1. This results in near-zero gradients, which slows the convergence of the model parameters and impedes effective training.

Figure 8 illustrates test results using sigmoid function. The model's accuracy only reaches 0.4 and loss lingers at 1.02. The poor performance on the test data shows that sigmoid is not suitable for multi-class data. Since sigmoid treats each neuron independently, assigning a probability of 0 and 1 to each class. In multi-class classification, we want the model to choose one best class. However, Sigmoid doesn't enforce this and assign high probabilities to multiple classes simultaneously.

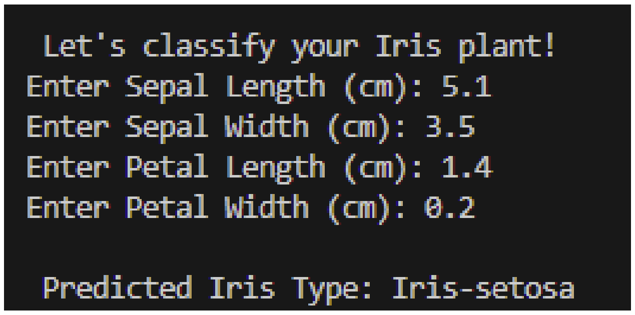


```
**** Final Test Performance: ****
Test Loss: 1.0280
Test Accuracy: 0.4000

🔍 **** Classification Results on Test Set ****:
Sample 1: True = Iris-versicolor, Predicted = Iris-virginica
Sample 2: True = Iris-setosa, Predicted = Iris-versicolor
Sample 3: True = Iris-versicolor, Predicted = Iris-virginica
Sample 4: True = Iris-setosa, Predicted = Iris-setosa
Sample 5: True = Iris-virginica, Predicted = Iris-virginica
Sample 6: True = Iris-setosa, Predicted = Iris-versicolor
Sample 7: True = Iris-versicolor, Predicted = Iris-virginica
Sample 8: True = Iris-setosa, Predicted = Iris-versicolor
Sample 9: True = Iris-virginica, Predicted = Iris-virginica
Sample 10: True = Iris-setosa, Predicted = Iris-versicolor
```

Figure 8: The above figure shows test results for sigmoid function

<u>*Manual Query of an Iris plant:*</u>



```
 Let's classify your Iris plant!
Enter Sepal Length (cm): 5.1
Enter Sepal Width (cm): 3.5
Enter Petal Length (cm): 1.4
Enter Petal Width (cm): 0.2

 Predicted Iris Type: Iris-setosa
```

Figure 9: ANN model result for manual user input

The above figure illustrates the result for a manual query. Using the user input data, the model accurately predicted the Iris type.

Conclusion

In this project, we designed a multi-layer neural network to classify Iris plant species. Without relying on external Python packages, we implemented the model architecture, forward pass, and backpropagation manually, allowing us to explore the underlying mathematics of the algorithm in depth. Our model successfully classified Iris species using both the test dataset and manually entered queries. The statistical plots above demonstrate that the model is robust, effectively avoids overfitting, and delivers promising results.