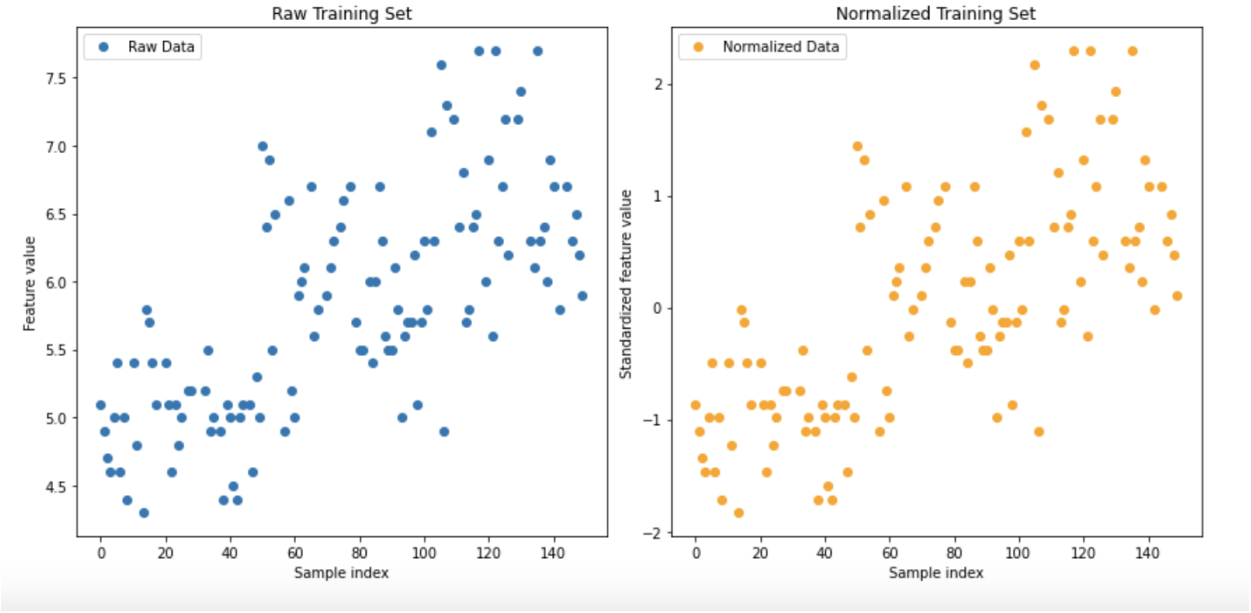
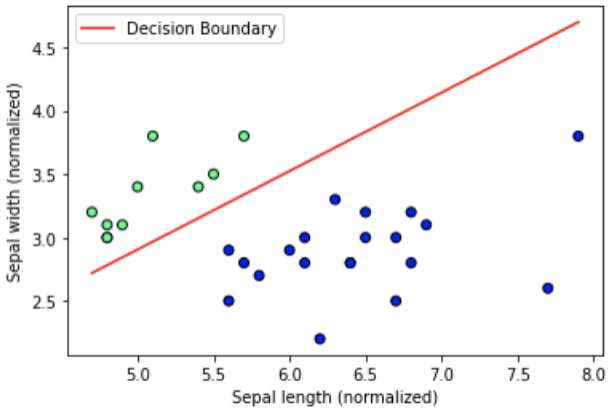
Q1. We again notice that the attributes are on different scales. Use the normalisation method from last lab, to standardize the scales of each attribute on both sets. Plot the normalized and raw training sets; what do you observe? [2 marks]



Observation:

- **Raw Data**: The attributes are on different scales ranging from 4.5 to 7.5.
- **Normalized Data**: The attributes have a mean of approximately 0 and a standard deviation of approximately 1. The attributes are on the same scale and centred around 0, making them easier for the network to process.

Q5. Draw the decision boundary on the test set using the learned parameters. Is this decision boundary separating the classes? Does this match our expectations? [2 marks]



Yes, the decision boundary seems to be separating the classes perfectly and it does meet our expectations.

Q6. Using the 3 classifiers, predict the classes of the samples in the test set and show the predictions in a table. Do you observe anything interesting? [4 marks]

Actual Class	Predicted Class
1	1
0	0
2	2
1	1
1	1
0	0
1	1
2	2
1	1
1	1
2	2
0	0
0	0
0	0
0	0
1	1
2	2
1	1
1	1
2	2
0	0
2	2
0	0
2	2
2	2
2	2
2	2
2	2
0	0
0	0

Looking at the table, I can clearly see how well the model has accurately predicted the class samples using the 3 classifiers. The predicted class samples have matched the actual class samples in every row of the data frame.

Q7. Calculate the accuracy of the classifier on the test set, by comparing the predicted values against the ground truth. Use a softmax for the classifier outputs. [1 mark]

```
# Applying softmax to the combined predictions
softmax_preds = F.softmax(combined_preds, dim=1)

# Determining the predicted classes using the softmax output
predicted_classes_softmax = torch.argmax(softmax_preds, dim=1)

# Calculating the accuracy
correct_predictions = (predicted_classes_softmax == y_test.argmax(dim=1)).sum().item()
total_predictions = y_test.size(0)
accuracy = correct_predictions / total_predictions

print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 1.0000
```

By using a softmax for the combined classifier outputs, the accuracy was calculated to be 100%.

Q8. Looking at the datapoints below, can we draw a decision boundary using Logistic Regression? Why? What are the specific issues or logistic regression with regards to XOR? [2 marks]

No, we cannot draw a decision boundary using Logistic Regression. This is because Logistic Regression is a linear model, which means it can only create a linear decision boundary. The XOR problem, however, is non-linear. In the data, when x1 and x2 are either both 0 or both 1, y is 0, but when one is 1 and the other is 0, y is 1. Therefore, this pattern cannot be separated by a straight line.

Intro to Neural Networks

Q1. Why is it important to use a random set of initial weights rather than initializing all weights as zero in a Neural Network? [2 marks]

Random initialization of weights is crucial for effective learning in neural networks, as it prevents the symmetry problem and ensures a non-zero gradient for learning different features.

Q2. How does a NN solve the XOR problem? [1 marks]

A Neural Network solves the XOR problem by using a hidden layer with non-linear activation functions. This hidden layer enables the network to create and combine multiple linear decision boundaries in a non-linear fashion. This approach allows the network to differentiate XOR data points where outputs are 1 for unequal inputs and 0 for equal inputs, overcoming the challenge of the XOR problem's non-linear separability.

Q3. Explain the performance of the different networks on the training and test sets. How does it compare to the logistic regression example? Make sure that the data you are refering to is clearly presented and appropriately labeled in the report. [8 marks]

After training the MLP model with 1, 2, 4, 8, 16, and 32 hidden neurons, I evaluated their performances on both training and test sets. The results were as follows:

```
Hidden Neurons: 1, Train Accuracy: 0.68, Test Accuracy: 0.63
Hidden Neurons: 2, Train Accuracy: 0.98, Test Accuracy: 1.00
Hidden Neurons: 4, Train Accuracy: 0.98, Test Accuracy: 1.00
Hidden Neurons: 8, Train Accuracy: 0.98, Test Accuracy: 1.00
Hidden Neurons: 16, Train Accuracy: 0.98, Test Accuracy: 1.00
Hidden Neurons: 32, Train Accuracy: 0.98, Test Accuracy: 1.00
```

1 Hidden Neuron: The model achieved 68% accuracy on the training set and 63% on the test set. This performance indicates that the model was likely underfitting, lacking the necessary complexity to capture the patterns in the data.

2+ Hidden Neurons: A significant improvement was observed in all models with more than one hidden neuron, reaching 98% accuracy on the training set and a perfect 100% on the test set. This suggests that a minimal level of complexity (at least two neurons in the hidden layer) is essential for the MLP to perform well on this dataset.

Beyond 2 Neurons: Increasing the number of hidden neurons beyond 2 did not yield any improvement in accuracy.

Comparison with Logistic Regression:

When comparing to logistic regression, which also achieved a 100% accuracy, the results were interesting. Despite logistic regression being a linear model, it performed on par with MLPs having two or more neurons on this dataset. This indicates that while MLPs have the advantage of non-linear activation functions allowing them to capture more complex patterns, the Iris dataset might be sufficiently linear or simple enough that both logistic regression and minimalistic MLPs (with just two neurons) can effectively model it with high accuracy. The similar performance of both models suggests that for datasets like Iris, where linear separability is likely, the choice between logistic regression and more complex MLPs might be more influenced by considerations of model simplicity and interpretability rather than just accuracy.