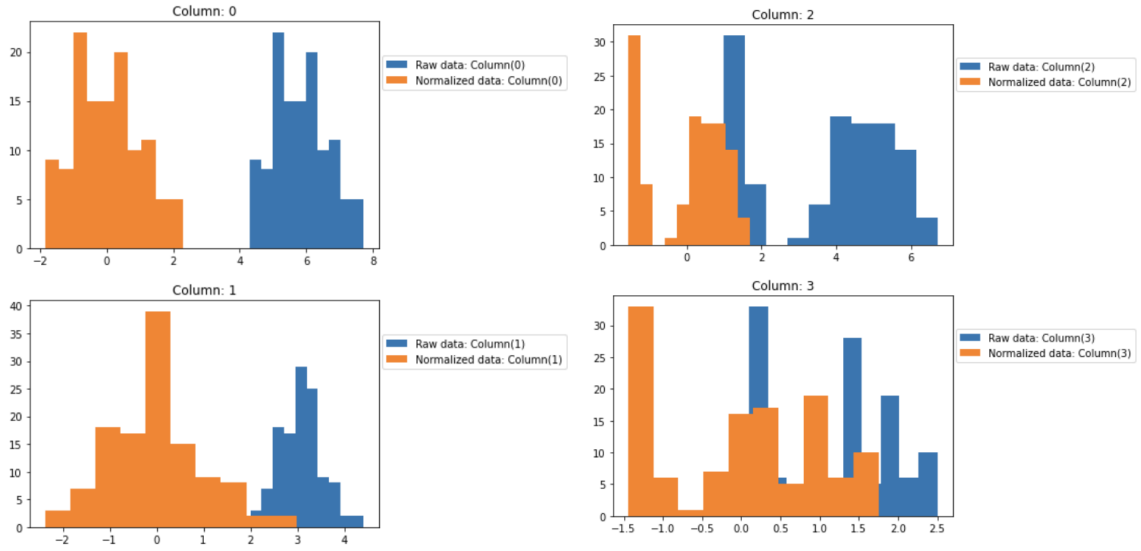
ML Assignment 1 Part 2a

Classification and Neural Networks

Gargeya Sharma (220278025) MSc Artificial Intelligence

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]

A1.



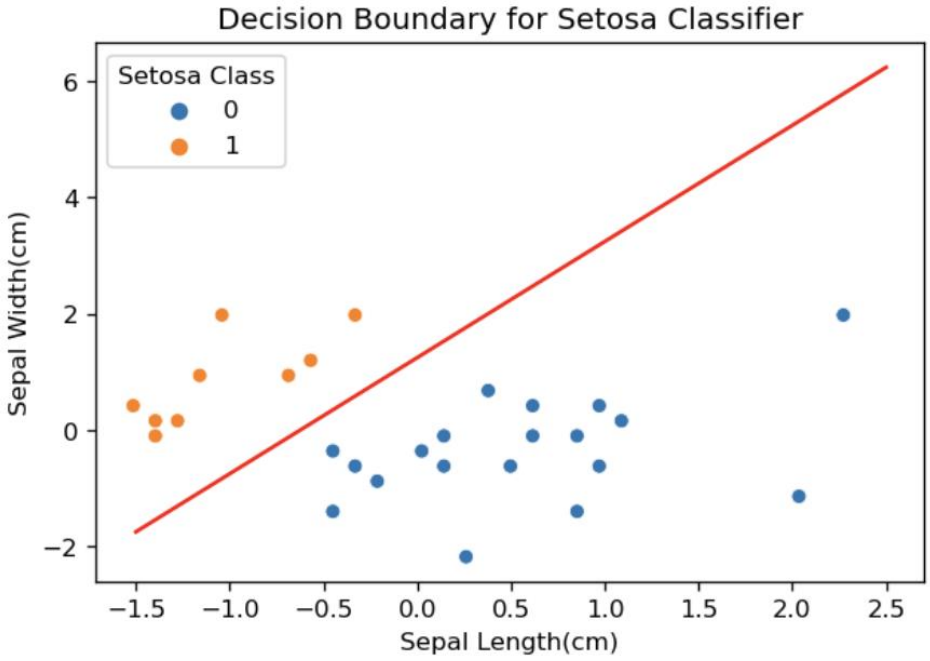
I observed that each feature in the dataset (Columns from 0-3) is having a different range of values. This kind of behavior is very common when we observe multiple types of values in real world ('features' in machine learning lingo) but for the machine; numbers are just numbers and if we want to provide unbiased values to our model to let learn while putting equal importance on each feature, we need to scale them on a single standard. Hence, usually we normalized the features. You can see that the histogram for the raw and normalized values for each feature is nearly the same and we got them on the same scale.

NOTE: some values have changed their bins but that's okay, my histogram representation only has 10 bins, but we deal with individual values while training a model. And every value is still relatively at the same distance with each other in a single feature due to collective normalization

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]

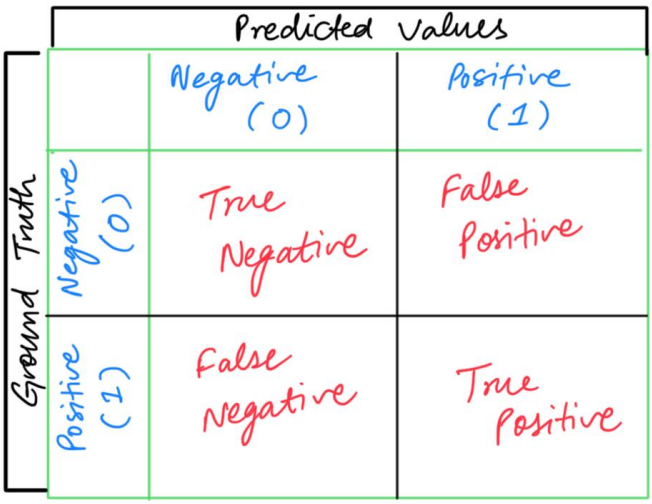
A5. As shown below, yes, this decision boundary created using our learned parameters is indeed separating the classes. I have used 'pandas' and 'seaborn' to plot this figure and shown both the classes with two different colors. The linear equation with the learned parameters is converted into the form of 'y=mx+c' so that it can be easily plotted on a 2-D plane. After this plotting, a threshold of 0.5 (middle value of the softmax activation function) is chosen to segregate the predicted values

(floating points) into Boolean values of ‘0’ and ‘1’. This clearly shows that the classifier is meeting 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]

A6. The predictions are shown as a NumPy matrix which is a table. This special kind of matrixes to show the classification prediction performance are called **confusion matrix**. Confusion table which helps us understand precisely which prediction class was predicted right and wrong, how many times was it the case and if some mismatch then with which other class. Correct predictions are termed as True Negatives and True Positives (Their predicted values match with their ground labels) and incorrect predictions are termed as False positives and False negatives (Prediction shows positive result while in reality label shows negative value and vice versa respectively). Accuracy of the model is calculated as: *(No. of True Positives + No. of True negatives) / Total Number of Instances*



Here are the confusion matrixes for all three of our classifiers:

Setosa Classifier:

```
array([[20,  0],
       [ 0, 10]], dtype=int64)
```

Versicolor Classifier:

```
array([[13,  8],
       [ 1,  8]], dtype=int64)
```

Virginica Classifier:

```
array([[13,  8],
       [ 1,  8]], dtype=int64)
```

The most interesting thing in these confusion matrixes is that they all have different distribution of values in different sections of the matrix. Setosa classifier is doing a great job at classifying the data with 100% accuracy which signifies that the data was easily linearly separable for the model to adapt to. But contrarily with other classifier, versicolor and Virginia have certain degree of error in their predictions, mostly false positives. This is very interesting as it gives us a better understanding of the distribution of these classes, seems like versicolor and virginica have some of the (sepal length and sepal width) values mixed with each other. That's why the model is not able to linearly classify them completely and produced lower prediction accuracy.

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]

A7. In my attempt to calculate the accuracy of the classifier with softmax activation function, I modified the 'y_test' values from a matrix of 3 columns to a vector. The main difference between them is that instead of writing a label as [0., 0., 1.], I am choosing 2 as an integer for easier comparison with softmax outputs.

Softmax function will provide us with prediction values in terms of probability for each class to represent that instance, for instance: [0.1, 0.05, 0.85]. This means model predicts that class at index 2 column vise is likely to be the answer for the input instance. Its easier for us to get these values from [0.1, 0.05, 0.85] to single integer value '2' using 'torch.argmax()' function.

While, both of ground labels and predicted values are converted in the same format using 'torch.argmax()' function, we can easily pass these vectors as the input of accuracy function we imported from sklearn.metrics.accuracy_score and that gives us the calculated accuracy score just like mentioned above: *(No. of True Positives + No. of True negatives) / Total Number of Instances.*

```
y_test = torch.argmax(y_test, dim=1)
print(y_test)
y_pred = torch.argmax(iris_model(x_test), dim=1)
y_pred
```

```
tensor([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2, 0, 2,
        2, 2, 2, 2, 0, 0])
tensor([1, 0, 2, 2, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 2, 2, 1, 1, 2, 0, 2, 0, 2,
        2, 2, 1, 2, 0, 0])
```

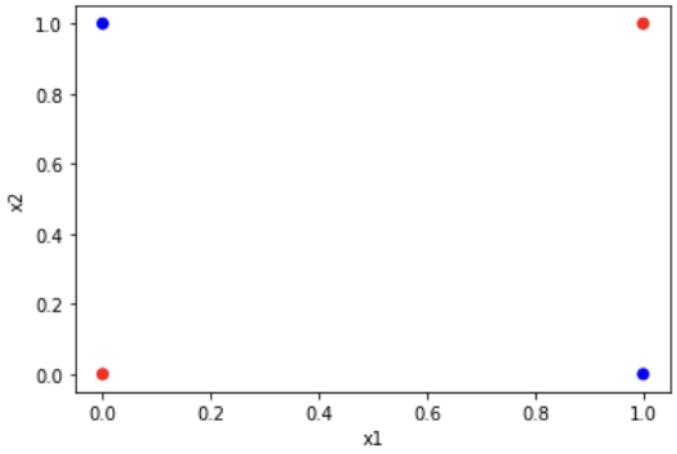
Applying the same conversion to pass inputs into 'accuracy_score' gives us the resultant accuracy values for both training set and testing set.

```
from sklearn.metrics import accuracy_score as acc
print(f"The Training accuracy of the model: {acc(torch.argmax(y_train, dim=1).numpy(),\
                                                torch.argmax(iris_model(x_train), dim=1).numpy())*100}%")
print(f"The Test accuracy of the model: {acc(y_test.numpy(), y_pred.numpy())*100}%")
```

```
The Training accuracy of the model: 82.5%
The Test accuracy of the model: 90.0%
```

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

A8. No, we can not draw a decision boundary for the XOR problem data points because they are not linearly separable but a single line.



Although we might thing that sigmoid function is a non-linear activation function and hence the model is a non-linear model, that is not the case. Logistic Regression is considered a linear model because the output from the model always depends on the sum of inputs and parameters rather than their product (which might introduce non-linear behavior).

Summing n-numbers of linear functions will always produce a linear function at the end. Sigmoid is helping us segregating the output values to available binary classes in the data. Logistic Regression model is an additive structure so there is **no** interaction between multiple parameter-input pairs (w1x1 *w2x2). XOR is a function that shows contrastive and diagonal behavior which produces the data points required more than a single line to classify them. A quadratic function might help us with the classification but as mentioned above unless we perform feature engineering and change the input values to quadratic nature, logistic regression is not capable to learn the quadratic nature of the dataset.

ML Assignment Part 2b

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]

A1. There seems to be two main reasons why I think Its important to use a random set of initial weights rather than zeros.

1. All optimization algorithms have an initial point which plays a major role in converging to an optimal value, it can be a local optimal or global (depends on the use case and the type of objective function). If our initial weights are zeros for each neuron, every time we run our model the bias term if not zero will be the only factor capable enough to change the weights during the backpropagation because rest everything will be 0 after getting multiplied by 0. Hence, training process will start inefficiently and redundantly (no randomness introduced).
2. Also, the direction of optimization for each neuron that will be introduced at every cycle of forward and backward propagation will be fixed for every one of them as mentioned above, there is no introduction to stochasticity every time, we train the model. So, eventually in a non-deterministic environment, finding different local minima (exploring different parts of the loss plane) and chances of finding a global minimum is heavily reduced.

In summary, when we initialize weights with reasonably defined random set (various ways researched by the community), every neuron in the network has a different initial position to contribute towards minimizing the loss function. Their non redundant individual efforts are what makes neural networks so powerful and robust. Having different parameters (all with high performance rate) every time you train your model is actually better than getting single results. It shows exploration in determining optimal values from the loss plane.

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

A2. Neural Network is a combination of non-linear layers on top of each other (if it's just a combination of linear layers than a single neuron without any activation function is just as same as the neural network). The input that we pass to a neural network is transformed at each layer with defined non-linearity (sigmoid in our case) to then further contribute to better understanding of the data and its underlying patterns after being merged together for the output layer. Hidden layers play a major and crucial role in dissecting the non-linear patterns in the data and allow us to minimize the loss with fine tuning happening at each forward-back propagation cycle.

XOR data is not linearly separable so modelling a logistic regression model would be a bad decision to perform the task. With neural network, we let our model understand those non-linear patters in the data and formulize a model with heuristically chosen learning rate and number of epochs to minimize the training loss. This end product can be said to understand the underlying patter of XOR to produce an approximately same result as the ground labels.

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 referring to is clearly presented and appropriately labelled in the report. [8 marks]

A3. Data: The dataset that I am using to measure the performance of different networks is of the class 'versicolor' in Iris Dataset. As we have to compare these different networks with a logistic regression, I am choosing versicolor class to formulate a binary classification problem as we did in the previous ("Classification") lab. The point to choose versicolor and not other classes from the iris dataset is that in the previous lab the confusion matrix from logistic regression on versicolor was the worst and through this lab I wish to improve the performance on that task and present the overwhelming strength of neural networks.

X_train and **X_test** values are randomly-proportionally **splitted** features values in the dataset for a fixed seed of 42. **Y_train** and **Y_test** values are the corresponding label values for each example in X_train and X_test respectively. The conversion from 3 classes output to 2 classes output is performed and stored in respective '*versicolor_labels*' and '*versicolor_testy*' variables so that versicolor examples are labelled as **'1'** while all the other classes are labelled as **'0'.**

```
print(versicolor_labels.reshape(-1))
print(versicolor_testy.reshape(-1))
```

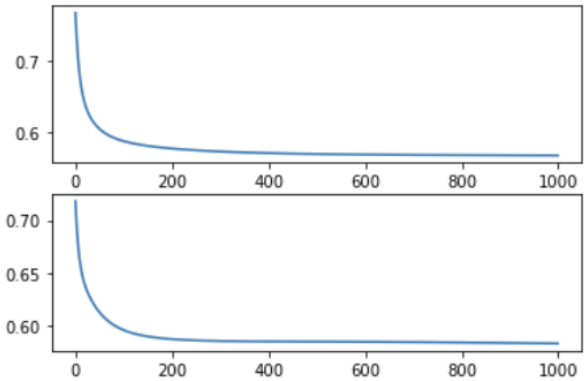
```
tensor([0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0.,
        1., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0.,
        1., 0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 0., 0., 1.,
        1., 0., 0., 0., 0., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
        0., 1., 0., 0., 0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 1., 0., 1., 1.,
        1., 0., 1., 1., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0.,
        1., 0., 1., 1., 0., 0., 0., 1., 0., 0., 1., 0.])
tensor([1., 0., 0., 1., 1., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0., 1., 0., 1.,
        1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

NOTE: Just for presenting the output compactly in the report, I reshaped them into a single dimension otherwise they are a 2-D matrix with shape: (number of examples, 1)

Comparison:

Logistic Regression:

```
train(versicolor_model, x_train, versicolor_labels, x_test, versicolor_testy, optimiser, alpha)
```




```
Minimum train cost: 0.5661290287971497
Minimum test cost: 0.583259642124176
```

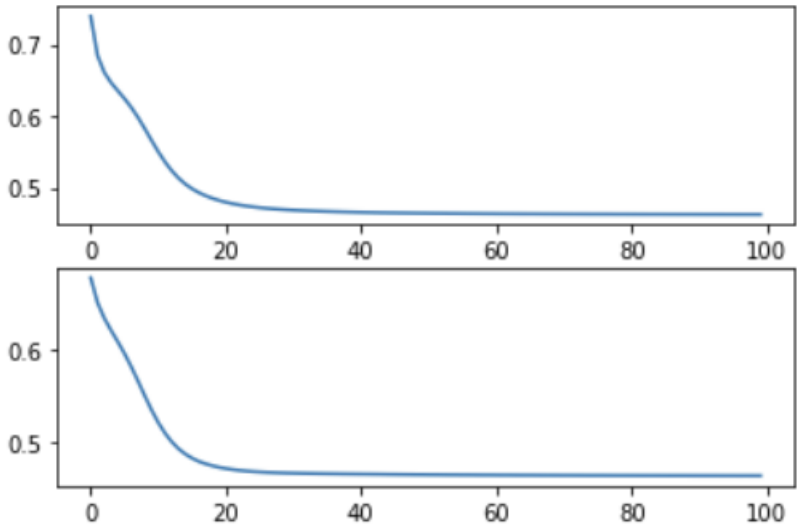
```
versicolor_y_pred = (versicolor_model(x_test) > db).int()
cm(versicolor_testy, versicolor_y_pred)
```

```
array([[13,  8],
       [ 1,  8]], dtype=int64)
```

Logistic Regression gave us a lot of false positive values as can be seen in the confusion matrix for the versicolor logistic regression model. The accuracy of this model is also pretty low due to such high false positives, accuracy = (TP + TN)/ Sum of all ➔ 21/30 ➔70%

Now, we will be building a neural network implementation while experimenting with different number of neurons in the hidden layers and realize what is happening and if yes then how the prediction is getting better with versicolor binary classification problem compared to single neuron logistic regression model mentioned above.

```
----------------- 1 Hidden Neurons--------------------
```




```
Minimum train cost: 0.4628077745437622
Minimum test cost: 0.4646872580051422
```

```
[[10 11]
 [ 0  9]]
The Training accuracy of the model: 67.50%
The Test accuracy of the model: 63.33%
```

Conventions used in the report from below here:

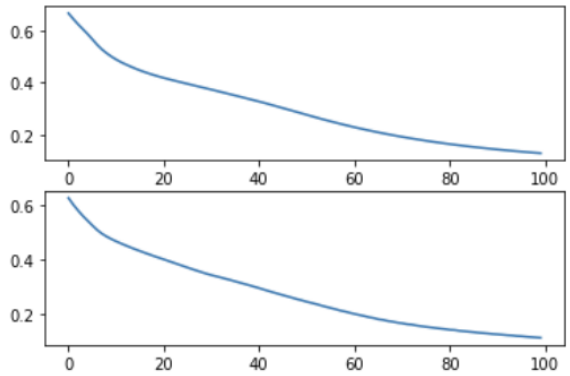
LR ➔ Logistic Regression model with single neuron

n-model ➔ 'n' number of neurons in the hidden layer

Observation: Looking at the loss values and confusion matrices of LR and 1-model we can tell that the later improved on the loss function but maybe modelled such a function which was not a good

representation of the overall dataset. Hence, the accuracy reduced from 70% ➔ 63.33% on the test set.

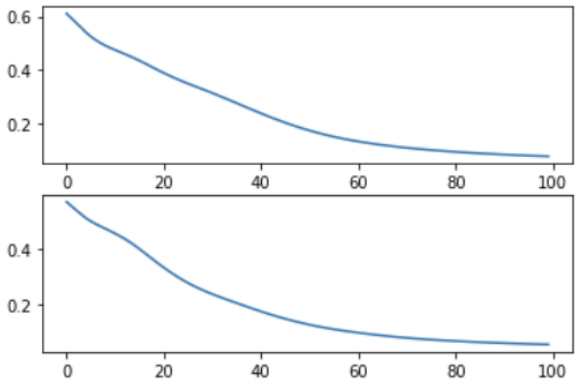
```
----------------- 2 Hidden Neurons--------------------
```




```
Minimum train cost: 0.12838751077651978
Minimum test cost: 0.11372236907482147
--------------------------------------------------------------------------

[[21  0]
 [ 0  9]]
The Training accuracy of the model: 97.50%
The Test accuracy of the model: 100.00%
```

Observation: There is a tremendous improvement by 2-neurons on the performance of versicolor classification. Just by adding 2 neurons in the hidden layer, we managed to grasp the underlying non-linearity in the data and modelled it optimally. We can see no errors on the test set and that is truly what we expect of our models. You can see there is a certain training error but that's okay because we don't want our model to just focus on training dataset and maybe overfit it, that way it could lose accuracy on the test set which is highly unwanted. Conclusively, I am very satisfied with the model with just 2 hidden neurons but for experimentation purposes, we will have a look other model with increasing number of hidden neurons as well.

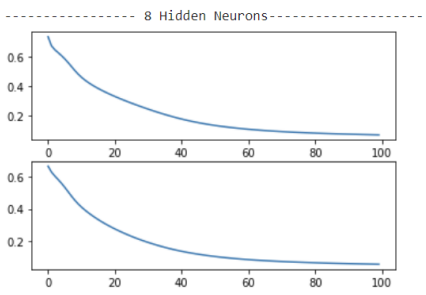
```
----------------- 4 Hidden Neurons--------------------
```



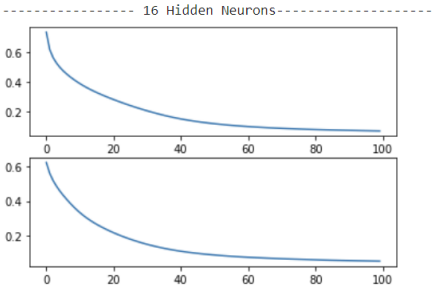

```
Minimum train cost: 0.07867659628391266
Minimum test cost: 0.060316912829875946
--------------------------------------------------------------------------

[[21  0]
 [ 1  8]]
The Training accuracy of the model: 95.83%
The Test accuracy of the model: 96.67%
```

Observation: The performance of 4-neurons is a good way to clarify the confusion that higher number of hidden neurons does not necessarily mean that you model will be better. Increasing neurons also increases the capability of the network to learn available noise (anomalies) in the data and might result in degraded performance as can be portrayed above. The approximation function that this neural network learned is not a good-and-general representation of the versicolor data.

NOTE: Nothing in the observation can be said with 100% confidence because neural networks are black boxes and there is a lot of active research going on in improving their explainability. Still, because these are some simple models on a small dataset, we can easily point out what is most likely to be going wrong with the models.




```
Minimum train cost: 0.07212495058774948
Minimum test cost: 0.053867537528276443
```

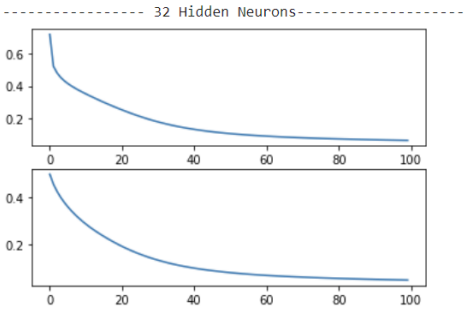



```
Minimum train cost: 0.06999407708644867
Minimum test cost: 0.05282251164317131
```

```
[[21  0]
 [ 0  9]]
The Training accuracy of the model: 95.83%
The Test accuracy of the model: 100.00%
```

```
[[21  0]
 [ 0  9]]
The Training accuracy of the model: 96.67%
The Test accuracy of the model: 100.00%
```

With 8-neurons, 16-neurons and 32-neurons, the model is getting large enough to capture all sorts of meaningful and useless(noise) information from the data (You can see from the decreasing value of loss). Yes, the test accuracy is 100% for all the cases because it has understood the region it needs to separate for versicolor examples but the function and its complexity that is being used to perform the classification is trying to overfit the data as we increase the value of 'n' neurons. Having too much dimensionality in the network over a relatively simple dataset reduces the ability to generalize well on the data.




```
Minimum train cost: 0.06710771471261978
Minimum test cost: 0.04884183406829834
```

```
[[21  0]
 [ 0  9]]
The Training accuracy of the model: 98.33%
The Test accuracy of the model: 100.00%
```

Conclusively, you can see that until 16-neurons, 2-neurons model is still better in both training and testing performance. Also, the reduced complexity in 2-neurons model is proven better to capture the general underlying pattern in the dataset than of 16-neurons. According to me, 32-neurons is just an overkill for this task as probably its' complexity is also enforcing it to overfit the training data.

Another major factor that I would like to point out from this analysis is that higher the number of neurons in the model, higher the computational time of the network. It all depends on what we expect from the data (our problem formulation) and what are our available resources.