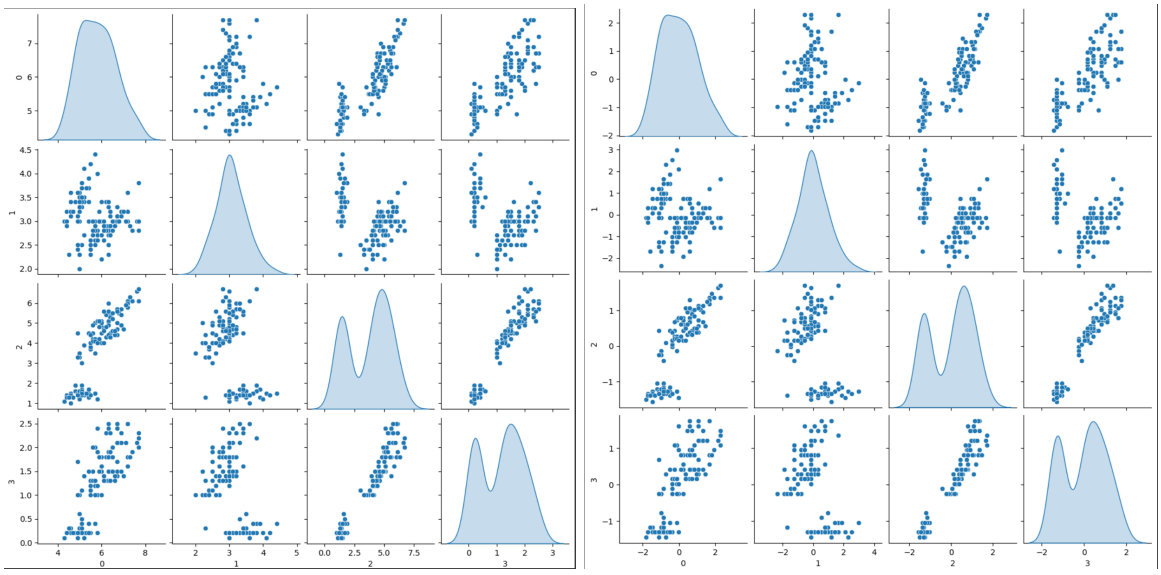
Name: Kiana Hadysadegh
Student ID: 230632224
Assignment Number 2(Classification and Neural Networks)
Module number: ECS708P

Classification:

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]

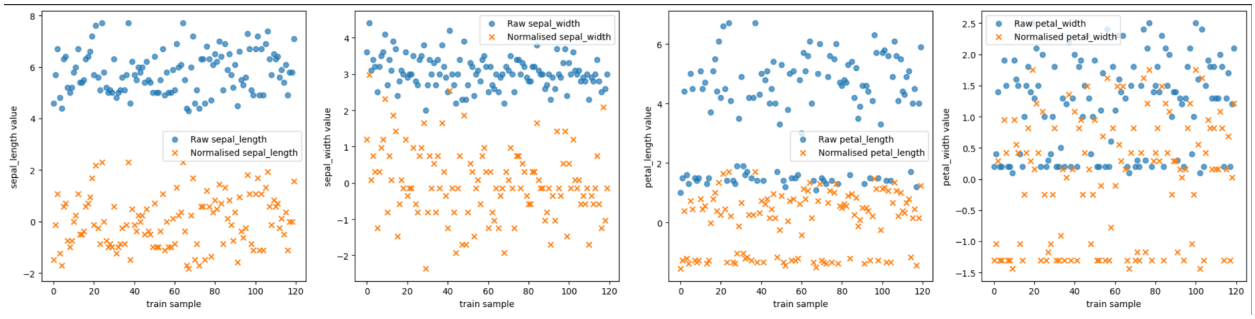
When we use a pairplot to plot the raw(right plot) and normalized data(left plot), we can observe that the shape of the distribution is exactly the same, and only the scale of the axes have changed.

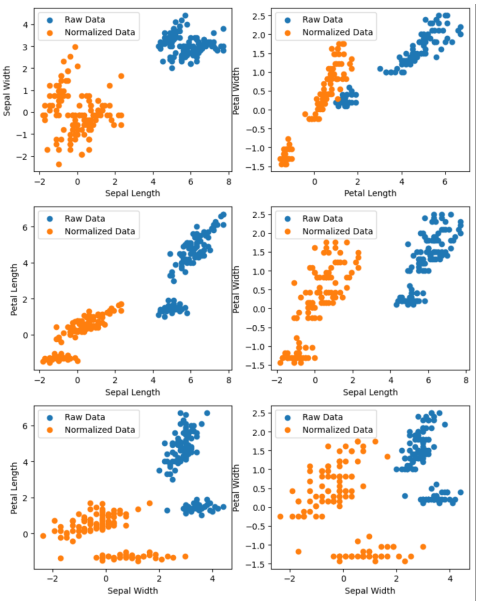


For each raw and normalized training sample, we plot its feature value:

```
features_labels = ["sepal_length", "sepal_width", "petal_length", "petal_width"]
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

for i in range(len(features_labels)):

    axes[i].scatter(range(len(x_train_original)), x_train_original[:, i], label="Raw {}".format(features_labels[i]), marker='o', alpha=0.7)
    axes[i].scatter(range(len(x_train)), x_train[:, i], label= "Normalised {}".format(features_labels[i]), marker='x', alpha=1)
    axes[i].set_ylabel("{} value".format(features_labels[i]))
    axes[i].set_xlabel("train sample")
    axes[i].legend()

plt.tight_layout()
plt.show()
```

Result:





The shape of distributions remains the same in all graphs means that normalization only changes their scale which makes features easier to compare.

In the third graph in the second figure (petal_length feature) We can see that the range of normalized data is smaller than the range of the raw data . So, normalization compressed the scale of the values of the raw training sample for the petal length feature.
It seems that normalization brings extreme values closer to the bulk of the data in the third graph in the second figure(petal_length feature).

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]

The decision boundary formula is :

$$\mathbf{w'}^{T}\mathbf{x'}=\mathbf{w}^{T}\mathbf{x}+\mathbf{w}_0$$

The decision boundary is perpendicular to the vector W (w0 used as biased = y_intercept). To draw the decision boundary, we use the weights of the model as coefficients of the straight line :

```
import numpy as np


x_range = np.linspace(test_set_1[:, 0].min(), test_set_1[:, 0].max(), 100)
y_range = (-model.weight[0,2] - model.weight[0,0] * x_range) / model.weight[0,1]
plt.scatter(test_set_1[:, 0], test_set_1[:, 1], c=setosa_test, cmap='coolwarm', marker='o')
plt.plot(x_range, y_range, color='green')

plt.xlabel('Sepal Width')
plt.ylabel('Sepal Length')

plt.show()
```

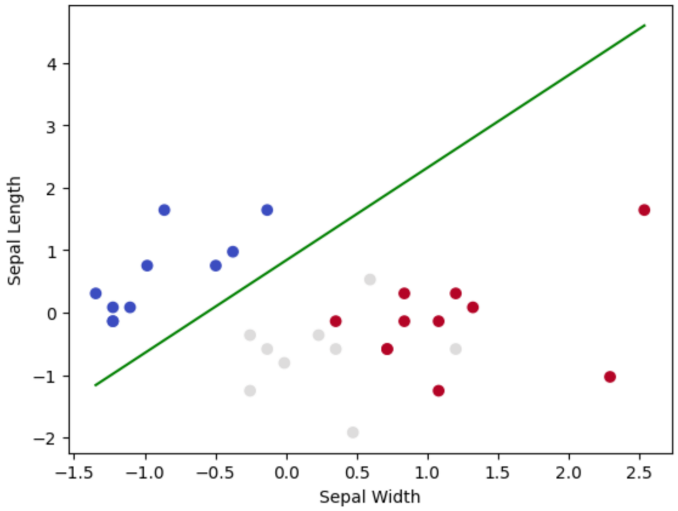
We can see that the line separates the classes correctly (like what we expected since the Setosa data is linearly separable from the two other classes (Versicolor and Virginica)). As it is obvious from the plot the line managed to separate the data in such a manner that none Setosa data is on the other side of the line and vice versa which indicates the validity of the result.

In the graph, Setosa, Versicolor, and Virginica data points are shown with blue, gray, and red colors respectively. Since we wanted to classify Setosa from non-Setosa data points:

1- we can see all of the Setosa data points are above the line (our decision boundary)

2- we can see all of the none-Setosa data points are below the line (our decision boundary)

So our accuracy is 100%. This was expected since our training cost was so small (Minimum cost: 0.047883812338113785) and we could see from the figure that setosa data is linearly separable visually from the other 2 classes.



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]

We create a data frame from predictions of the 3 classifiers using the test set as their inputs:

```
### your code here
setosa_pred = setosa_model(x_test)
versicolor_pred = versicolor_model(x_test)
virginica_pred = virginica_model(x_test)
# print(setosa_pred.tolist())

data = {
    "Setosa":setosa_pred.tolist(),
    "Versicolor":versicolor_pred.tolist(),
    "Virginica": virginica_pred.tolist()
}

df = pd.DataFrame(data)
df
```

We plot the data frame(left table) and rounded data frame(rigth table) to find the class of the given data points. For rounding the data: if our prediction value is less than 0.5 we round it to 0, otherwise we round it to 1. So for having correct predictions by this model, only the prediction value of one class should be equal to 1 which means this data point belongs to that class:

	Setosa	Versicolor	Virginica
0	[0.03971399366855621]	[0.7366571426391602]	[0.46104705333709717]
1	[0.9999624490737915]	[0.17345799505710602]	[0.10570862144231796]
2	[2.984815637319116e-06]	[0.8442276120185852]	[0.9440402984619141]
3	[0.037327125668525696]	[0.5887150764465332]	[0.7090325951576233]
4	[0.008224599994719028]	[0.7726467251777649]	[0.516614556312561]
5	[0.9996559619903564]	[0.3258649706840515]	[0.07544466108083725]
6	[0.2683156430721283]	[0.551170825958252]	[0.5270524621009827]
7	[0.0014402020024135709]	[0.3937394618988037]	[0.9675100445747375]
8	[0.00042071816278621554]	[0.9200946688652039]	[0.3662884533405304]
9	[0.07731720805168152]	[0.7360580563545227]	[0.3672623336315155]
10	[0.011683023534715176]	[0.347735136747360 23]	[0.9492672085762024]
11	[0.999363124370575]	[0.5897557735443115]	[0.021043166518211365]
12	[0.9999182224273682]	[0.31357938051223755]	[0.04447127729654 31]
13	[0.9995635151863098]	[0.5375229716300964]	[0.02498646453022957]
14	[0.9999850988388062]	[0.12801893055438995]	[0.13354848325252533]
15	[0.151181787252426 15]	[0.33680856227874756]	[0.8649206757545471]
16	[0.000735711248125881]	[0.460204720497131 35]	[0.9701134562492371]
17	[0.041696228086948395]	[0.8295100927352905]	[0.25003519654273987]
18	[0.055580079555511475]	[0.6636043787002563]	[0.5712448358535767]
19	[0.00030859129037708044]	[0.579350471496582]	[0.955568790435791]
20	[0.9996992349624634]	[0.4190889596939087]	[0.04695776849985123]
21	[0.013706497848033905]	[0.47229471802711487]	[0.8911084532737732]
22	[0.9997528195381165]	[0.2830161154270172]	[0.09812285751104355]
23	[0.000435899943113327]	[0.6037288308143616]	[0.9410666823387146]
24	[0.019186018034815 79]	[0.2039065212011337 3]	[0.9788685441017151]
25	[0.0008656629943288863]	[0.4395460486412048 3]	[0.9668485522270203]
26	[0.00010863420175155625]	[0.8537782430648804]	[0.7732750773429871]
27	[0.0011188058415427804]	[0.3438043594360351 6]	[0.9823957085609436]
28	[0.9987297654151917]	[0.5401928424835205]	[0.03752882778644562]
29	[0.9993777275085449]	[0.502560555934906]	[0.037009939551353455]

	Setosa	Versicolor	Virginica
0	[0]	[1]	[0]
1	[1]	[0]	[0]
2	[0]	[1]	[1]
3	[0]	[1]	[1]
4	[0]	[1]	[1]
5	[1]	[0]	[0]
6	[0]	[1]	[1]
7	[0]	[0]	[1]
8	[0]	[1]	[0]
9	[0]	[1]	[0]
10	[0]	[0]	[1]
11	[1]	[1]	[0]
12	[1]	[0]	[0]
13	[1]	[1]	[0]
14	[1]	[0]	[0]
15	[0]	[0]	[1]
16	[0]	[0]	[1]
17	[0]	[1]	[0]
18	[0]	[1]	[1]
19	[0]	[1]	[1]
20	[1]	[0]	[0]
21	[0]	[0]	[1]
22	[1]	[0]	[0]
23	[0]	[1]	[1]
24	[0]	[0]	[1]
25	[0]	[0]	[1]
26	[0]	[1]	[1]
27	[0]	[0]	[1]
28	[1]	[1]	[0]
29	[1]	[1]	[0]

We can observe:
If we round the predictions of the 3 classifiers to find the class of the test data (left table), we can see that sometimes both Virginica and Versicolor value is 1. This means that by using the round operation here we cannot classify Virginica and Versicolor correctly sometimes, so that is why we should use softmax on our prediction values to predict classes accurately.
The predicted values for Versicolor and Virginica are so close compared to those for Setosa with other classes. This means that Setosa data is linearly separable from two other classes since its predicted values differ greatly from other classes. Meanwhile, the predicted values of Versicolor and Virginica are so close means that they are not linearly separable from each other.

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]

Accuracy = (number of correct predictions / total number of predictions) * 100
We first find the highest prediction of softmax belongs to what class (using argmax) Then we check if the prediction is equal to the target label, and update the number of correct_predictions variable:

```
predictions = torch.cat((setosa_pred, versicolor_pred, virginica_pred), dim=1)

predictions_softmax_values = torch.exp(predictions) / torch.exp(predictions).sum(axis=1, keepdims=True)
predicted_labels = predictions_softmax_values.argmax(dim=1)
y_test = y_test.argmax(dim=1)

correct_pred = 0

for prediction, label in zip(predicted_labels, y_test):
    if prediction == label:
        correct_pred += 1

print("{} %".format((correct_pred / len(y_test)) * 100))
```

```
90.0 %
```

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]

In the XOR problem:
If x1 and x2 are equal the output is 0, Otherwise, the output is 1.
The decision boundary in the XOR problem is not linear so:

A single straight line cannot classify the data in xor problem, since it is not linearly separable and the decision boundary in Logistic regression is linear.

If we try to train a logistic regression model on the given data, we won't find the line to completely separate the data points from different classes (Since the model tries to find the parameters that minimize the cost (binary cross entropy loss) while there is no linear combination between two independent variables x_1 and x_2 that can separate data points of different classes).

We should use another model which is capable to learn non-linear decision boundaries for the XOR problem like neural networks with hidden layers and non-linear activation functions.

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]

The weights should be initialized with different values to avoid having the same outputs and updates in the backpropagation step. Otherwise, this can cause similar weights and behaviors of neurons in a layer. This makes learning different features hard for the network.
To avoid the optimization algorithm stuck in a local minimum in some conditions, it is better to initialize weights with different values. This is because if all weights are initialized to the same value, all neurons will behave in the same direction in the gradient descent step.
Using the zero value for initializing weights can slow down the network to converge. This is because neurons output and their gradients in backpropagation will be the same, so, the weights will uniformly change.

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

In the XOR problem:
If x1 and x2 are equal the output is 0, Otherwise, the output is 1. The decision boundary in the XOR problem is not linear.

A single straight line cannot classify the data in xor problem, since it is not linearly separable and the decision boundary in Logistic regression is linear. If we try to train a

logistic regression model for the given data, we won't find the line for completely separating the data points from different classes.

A neural network with one or more hidden layers can be used as a solution for this problem. In the xor problem, we need a network to learn complex (non-linear) decision boundaries and the hidden layer that does non-linear transformations to the given data: the non-linear activation function in the hidden layer is used to learn non-linear decision boundary. So, during the training process, the weights of the network are updated using the backpropagation method till the output of the network matches the answer to the xor problem. In this method, the gradients are computed and the weights of the network are updated to minimize the error (difference between target and predicted output)

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]

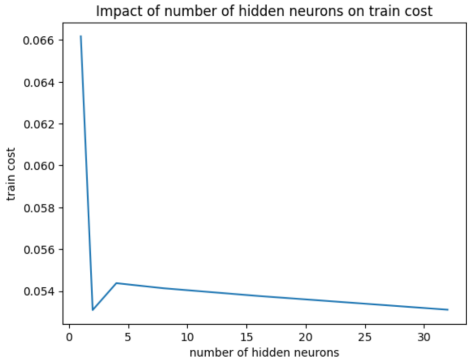
Since we want to classify the data point with one classifier between 3 class, I change the cost function to cross_entropy. Then I plot the minimum train and test cost for each number of hidden neurons to see its impact on the train and test cost:

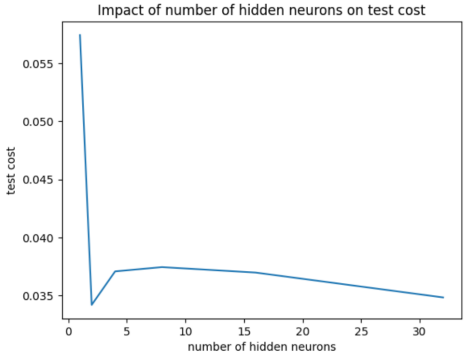
```
def train(model, x, y, test_x, test_y, optimiser, alpha, number_of_hidden_neurons):
  train_lst = list()
  test_lst = list()
  for i in range(1000):
    model.train()
    optimiser.zero_grad()
    pred = model(x)
    cost = F.cross_entropy(pred, y, reduction='mean')
    cost.backward()
    train_lst.append(cost.item())
    optimiser.step()
    model.eval()
    with torch.no_grad():
      test_pred = model(test_x)
      test_cost = F.cross_entropy(test_pred, test_y, reduction='mean')
      test_lst.append(test_cost)
  fig, axs = plt.subplots(2)
  axs[0].set_title("number of hidden neurons {}".format(number_of_hidden_neurons))
  axs[0].plot(list(range(i+1)), train_lst)
  axs[0].set_ylabel("train cost")
  axs[1].set_title("number of hidden neurons {}".format(number_of_hidden_neurons))
  axs[1].plot(list(range(i+1)), test_lst)
  axs[1].set_ylabel("test cost")
  plt.tight_layout()
  plt.show()
  print('Minimum train cost: {}'.format(min(train_lst)))
  print('Minimum test cost: {}'.format(min(test_lst)))
  return min(train_lst), min(test_lst), number_of_hidden_neurons
```

```
number_of_hidden_neurons = [1, 2, 4, 8, 16, 32]
alpha = 0.1
```

```
def calculate_precision_recall_f1_score(y_pred, y):

    prediction_max_indices = torch.argmax(y_pred, dim=1)
    prediction_max_indices_list = prediction_max_indices.tolist()


    label_max_indices = torch.argmax(y, dim=1)
    label_max_indices_list = label_max_indices.tolist()

    precision, recall, f1, support = precision_recall_fscore_support(label_max_indices_list, prediction_max_indices_list, average='macro', zero_division=1)
    print("precision {}, recall {}, f1 {}".format(precision, recall, f1))

    return precision, recall, f1
```

```python
import numpy as np
y_train = F.one_hot(y_train.reshape(-1).long(), num_classes=3)
y_test = F.one_hot(y_test.reshape(-1).long(), num_classes=3)
train_costs = []
test_costs = []

precisions = []
recalls = []
f1_scores = []


for num in number_of_hidden_neurons:

    model = nn.Sequential(nn.Linear(x_train.shape[1], num), nn.Linear(num, y_train.shape[1]))
    optimiser = optim.SGD(model.parameters(), alpha)
    train_cost, test_cost, _ = train(model, x_train.float(), y_train.float(), x_test.float(), y_test.float(), optimiser, alpha, num)
    train_costs.append(train_cost)
    test_costs.append(test_cost)
    predictions = model(x_test.float())
    prediction=model(x_test.float())
    precision, recall, f1 = calculate_precision_recall_f1_score(prediction, y_test)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)
```



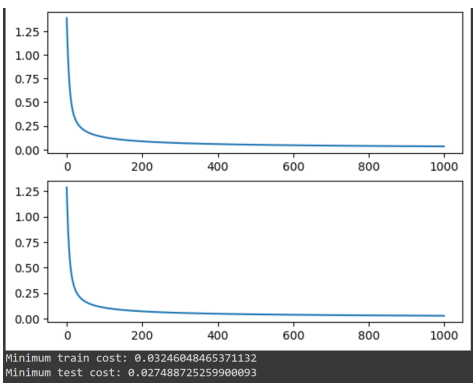


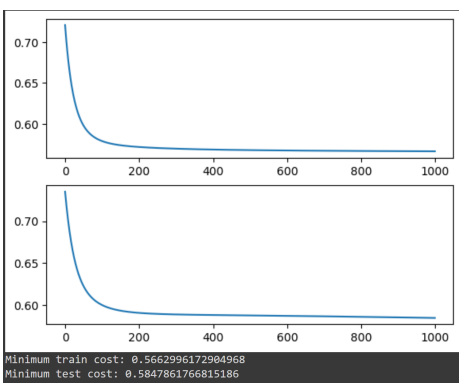
By looking at these graphs, it is obvious that training and testing costs decrease as the number of hidden neurons in the network increases.

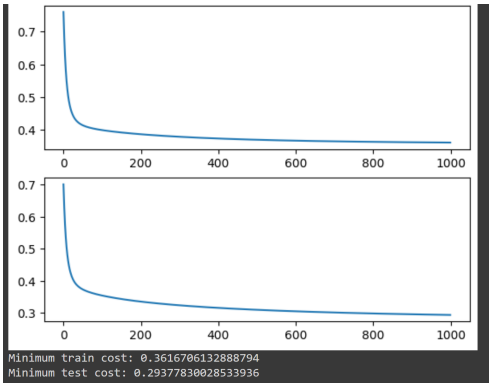
If we compare our neural network train and test costs with these costs from our three logistic regression classifiers:

we can observe that neural network classifier costs are significantly smaller than the 3 logistic regression classifiers' costs. This is because our data points in the iris dataset are not linearly separable and that is why we tried to train 3 logistic regression classifiers to classify each class from the other two classes. According to this, It is harder for a logistic regression model to learn how to classify data points compared to the neural network (specifically for Versicolor and Virginica since their logistic regression classifiers' predictions were so close to each other compared to Setosa). The reason for this is that the decision boundary in the Logistic regression model is linear but neural networks are able to learn non-linear decision boundaries. This is why the cost of training a logistic regression classifier to classify one class from other classes is higher than the cost of training a neural network for classifying all classes from each other.

3 Logistic regression classifiers minimum train and test costs:





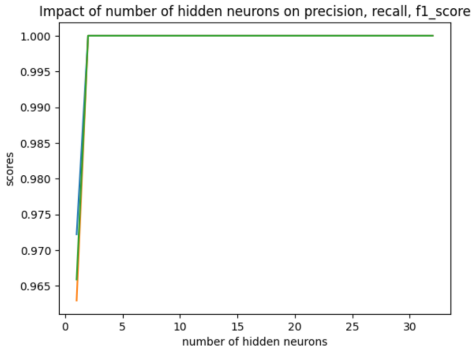


Neural Networks with different hidden neurons minimum train and test costs:

```
[272] print(test_costs)

      [tensor(0.0574), tensor(0.0342), tensor(0.0371), tensor(0.0374), tensor(0.0370), tensor(0.0348)]

[273] print(train_costs)

      [0.06617721915245056, 0.053088560700416565, 0.05437406152486801, 0.054129667580127716, 0.05376134067773819, 0.053105004131793976]
```

We can observe the better performance of the neural network compared to linear regression classifiers in terms of train and test cost by comparing them. (Except for Setosa classifier because Setosa points were linearly separable from other classes)

I also calculate precision, recall, and f1_score for each number of hidden neurons:



Precision, recall and f1_score are 1 for all hidden number of neurons except 1 hidden neuron.

Precision recall and f1_score of logistic regression classifiers (setosa, versicolor, virginica):

```
precision 1.0, recall 1.0, f1 1.0
precision 0.7142857142857143, recall 0.753968253968254, f1 0.6914285714285715
precision 0.84375, recall 0.868421052631579, f1 0.8316498316498315
(0.84375, 0.868421052631579, 0.8316498316498315)
```

Precision, recall, and f1_score for setosa classifier is 1 since it is linearly separable but these scores are less for other classifiers than our neural network classifier which proofs neural network better performance in this case.

I plot train and test cost, precision, recall, and f1_score for each number of hidden neurons: