quantum_classifier_1

February 24, 2021

1 Installations and Imports

[1]:
```
!pip install pennylane --upgrade

from IPython.display import clear_output
clear_output(wait=False)
```

[32]:
```
import pennylane as qml
from pennylane import numpy as np
from pennylane.templates import RandomLayers


#import tensorflow as tf
#from tensorflow import keras

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_theme()
```

2 Importing data files

2.0.1 npy data files created by the quanvolutional notebook has been imported here.

[3]:
```
x_train_final = np.load("x_train_final.npy")
y_train = np.load("y_train.npy")

x_test_final = np.load("x_test_final.npy")
y_test = np.load("y_test.npy")
```

[4]:
```
x_train_final.shape, y_train.shape, x_test_final.shape, y_test.shape
```

[4]:
```
((250, 256), (250,), (65, 256), (65,))
```

train and test data has been concatenated for further size reduction as 256 features are hard to be encoded on a quantum circuit.

[5]:
```
X = np.concatenate((x_train_final, x_test_final))
Y = np.concatenate((y_train, y_test))
print(X.shape)
```

```
(315, 256)
```

from 256 features, 11 feautes has been selected that will be used further.

[6]:
```
from sklearn.svm import LinearSVC
from sklearn.feature_selection import SelectFromModel
lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, Y)
model = SelectFromModel(lsvc, prefit=True)
X = model.transform(X)
print(X.shape)
```

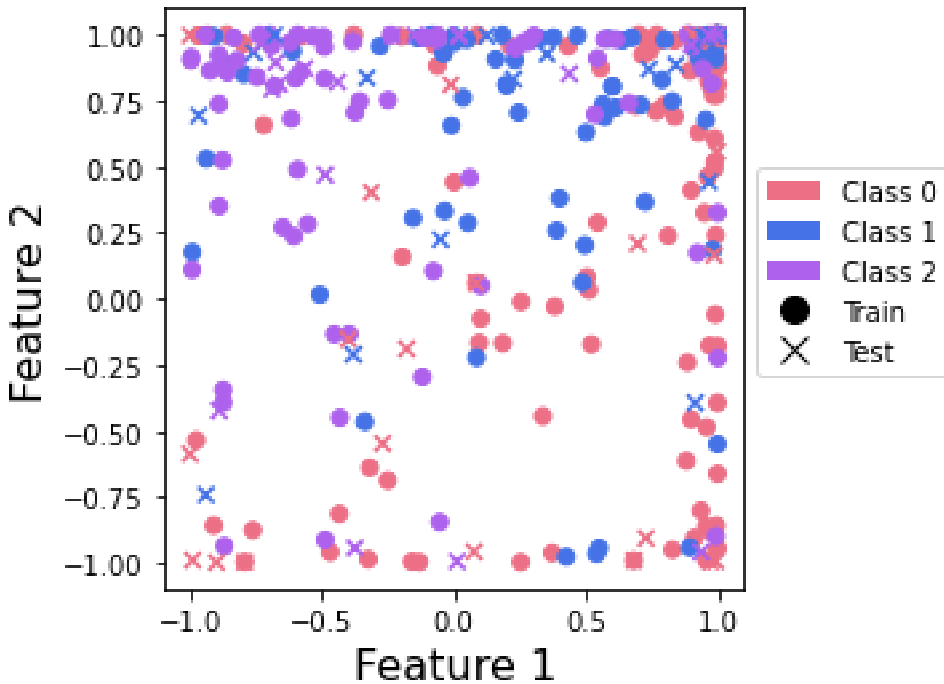
```
(315, 11)
```

Train and test data has been seperated again.

[7]:
```
x_train_final = (X[:250])
x_test_final = (X[250:])
```

[8]:
```
from matplotlib.lines import Line2D
from matplotlib.patches import Patch

colours = ["#ec6f86", "#4573e7", "#ad61ed"]


def plot_points(x_train, y_train, x_test, y_test):
    c_train = []
    c_test = []

    for y in y_train:
        c_train.append(colours[y])

    for y in y_test:
        c_test.append(colours[y])

    plt.scatter(x_train[:, 0], x_train[:, 1], c=c_train)
    plt.scatter(x_test[:, 0], x_test[:, 1], c=c_test, marker="x")

    plt.xlabel("Feature 1", fontsize=16)
    plt.ylabel("Feature 2", fontsize=16)

    ax = plt.gca()
    ax.set_aspect(1)

    c_transparent = "#00000000"
```

```
    custom_lines = [
        Patch(facecolor=colours[0], edgecolor=c_transparent, label="Class 0"),
        Patch(facecolor=colours[1], edgecolor=c_transparent, label="Class 1"),
        Patch(facecolor=colours[2], edgecolor=c_transparent, label="Class 2"),
        Line2D([0], [0], marker="o", color=c_transparent, label="Train",
               markerfacecolor="black", markersize=10),
        Line2D([0], [0], marker="x", color=c_transparent, label="Test",
               markerfacecolor="black", markersize=10),
    ]

    ax.legend(handles=custom_lines, bbox_to_anchor=(1.0, 0.75))


plot_points(x_train_final, y_train, x_test_final, y_test)
```



3 Classifier Model description

3.0.1 In the dataset we have three classses - Covid19, Viral Penumonia and Normal person.

3.0.2 Our approach is to divide the classfier in two different models. First model will classify between the classes 'Normal Person' and 'Covid19/Viral Pneumonia'. Second model will classify between the classes 'Covid' and 'Viral Penumonia'

3.0.3 First model here is denoted by Model-1 and second one by Model-2.

4 Required functions for classifier model-1

[9]:
```
def get_angles(x):

    beta0 = 2 * np.arcsin(np.sqrt(x[1] ** 2) / np.sqrt(x[0] ** 2 + x[1] ** 2 +␣
↪1e-12))
    beta1 = -2 * np.arcsin(np.sqrt(x[1] ** 2) / np.sqrt(x[2] ** 2 + x[1] ** 2 +␣
↪1e-12))
    beta2 = 2 * np.arcsin(np.sqrt(x[2] ** 2 + x[1] ** 2)/np.sqrt(x[0] ** 2 +␣
↪x[1] ** 2 + x[2] ** 2 ))

    return np.array([beta2, -beta1 / 2, beta1 / 2, -beta0 / 2, beta0 / 2])
```

[10]:
```
def statepreparation_1(a):

        qml.Hadamard(wires=0)
        qml.Hadamard(wires=1)

        qml.RZ(a[0], wires=0)
        qml.RZ(a[1], wires=1)
        qml.CNOT(wires=[0,1])
        qml.RZ(a[0]*a[1], wires=1)
        qml.CNOT(wires=[0,1])

        qml.Hadamard(wires=0)
        qml.Hadamard(wires=1)

        qml.RZ(a[1], wires=0)
        qml.RZ(a[2], wires=1)
        qml.CNOT(wires=[0,1])
        qml.RZ(a[1]*a[2], wires=1)
        qml.CNOT(wires=[0,1])
```

```
        qml.RY(a[0], wires=0)

        qml.CNOT(wires=[0, 1])
        qml.RY(a[1], wires=1)
        qml.CNOT(wires=[0, 1])
        qml.RY(a[2], wires=1)

        for i in range(3,10):
            qml.PauliX(wires=0)
            qml.CNOT(wires=[0, 1])
            qml.RY(a[i], wires=1)
            qml.CNOT(wires=[0, 1])
            qml.RY(a[i+1], wires=1)
            qml.PauliX(wires=0)
```

[11]:
```
def layer_1(W):
    qml.Rot(W[0, 0], W[0, 1], W[0, 2], wires=0)
    qml.Rot(W[1, 0], W[1, 1], W[1, 2], wires=1)
    # qml.Rot(W[2, 0], W[2, 1], W[2, 2], wires=2)

    qml.CNOT(wires=[0, 1])
    # qml.CNOT(wires=[0, 2])
    # qml.CNOT(wires=[1, 2])
```

[12]:
```
dev_1 = qml.device("default.qubit", wires=2)
@qml.qnode(dev_1)
def circuit_1(weights, x=None):
    # Feature mapping
    # angle = get_angles(x)
    statepreparation_1(x)
    # variational classifier
    for w in weights:
        layer_1(w)

    return qml.expval(qml.PauliZ(1))
```

[13]:
```
def classifier_training_1(params, x=None, y=None):
    weights = params[0]
    bias = params[1]

    out_probs = circuit_1(weights, x=x) + bias
    return (out_probs-y)**2
```

[14]:
```
def classifier_prediction_1(params, x=None):
    weights = params[0]
    bias = params[1]
```

```
    out_probs = circuit_1(weights, x=x) + bias


    if(out_probs>0):
        return 1
    else:
        return -1
```

```
[15]: def circuit_output_test(params, x=None):
          weights = params[0]
          bias = params[1]

          out_probs = circuit(weights, x=x) + bias

          return out_probs
```

```
[16]: def cost_1(params, X, Y):

          y_pred = np.array([classifier_training_1(params, x=X[i], y=Y[i]) for i in␣
      ↪range(len(Y))])

          cost = np.sum(y_pred) / len(Y)
          return cost
```

```
[27]: def accuracy_1(params, x_train, y_train, iter):

          y_pred_train = np.array([classifier_prediction_1(params, x=x) for x in␣
      ↪x_train])
          acc_train = np.sum(y_pred_train==y_train) / len(y_train)

          print("Iter=> {}   train_cost=> {}   train_acc=> {} ".format(iter+1,␣
      ↪cost_1(params, x_train, y_train), acc_train))

          return acc_train
```

5 Required functions for classifier model-2

```
[18]: def statepreparation_2(a):

              qml.Hadamard(wires=0)
              qml.Hadamard(wires=1)

              qml.RZ(a[0], wires=0)
              qml.RZ(a[1], wires=1)
              qml.CNOT(wires=[0,1])
```

```
        qml.RZ(a[0]*a[1], wires=1)
        qml.CNOT(wires=[0,1])

        qml.Hadamard(wires=0)
        qml.Hadamard(wires=1)

        qml.RZ(a[1], wires=0)
        qml.RZ(a[2], wires=1)
        qml.CNOT(wires=[0,1])
        qml.RZ(a[1]*a[2], wires=1)
        qml.CNOT(wires=[0,1])



        qml.RY(a[0], wires=0)

        qml.CNOT(wires=[0, 1])
        qml.RY(a[1], wires=1)
        qml.CNOT(wires=[0, 1])
        qml.RY(a[2], wires=1)

        for i in range(3,10):
            qml.PauliX(wires=0)
            qml.CNOT(wires=[0, 1])
            qml.RY(a[i], wires=1)
            qml.CNOT(wires=[0, 1])
            qml.RY(a[i+1], wires=1)
            qml.PauliX(wires=0)
```

[19]:
```
def layer_2(W):
    qml.Rot(W[0, 0], W[0, 1], W[0, 2], wires=0)
    qml.Rot(W[1, 0], W[1, 1], W[1, 2], wires=1)
    # qml.Rot(W[2, 0], W[2, 1], W[2, 2], wires=2)

    qml.CNOT(wires=[0, 1])
    # qml.CNOT(wires=[0, 2])
    # qml.CNOT(wires=[1, 2])
```

[20]:
```
dev_2 = qml.device("default.qubit", wires=2)
@qml.qnode(dev_2)
def circuit_2(weights, x=None):
    # Feature mapping
    # angle = get_angles(x)
    statepreparation_2(x)

    # variational classifier
    for w in weights:
```

```
        layer_2(w)

    return qml.expval(qml.PauliZ(1))
```

[21]:
```
def classifier_training_2(params, x=None, y=None):
    weights = params[0]
    bias = params[1]

    out_probs = circuit_2(weights, x=x) + bias
    return (out_probs-y)**2
```

[22]:
```
def classifier_prediction_2(params, x=None):
    weights = params[0]
    bias = params[1]

    out_probs = circuit_2(weights, x=x) + bias


    if(out_probs>0):
        return 1
    else:
        return -1
```

[23]:
```
def cost_2(params, X, Y):

    y_pred = np.array([classifier_training_2(params, x=X[i], y=Y[i]) for i in␣
 ↪range(len(Y))])

    cost = np.sum(y_pred) / len(Y)
    return cost
```

[28]:
```
def accuracy_2(params, x_train, y_train, iter):

    y_pred_train = np.array([classifier_prediction_2(params, x=x) for x in␣
 ↪x_train])
    acc_train = np.sum(y_pred_train==y_train) / len(y_train)

    print("Iter=> {}   train_cost=> {}   train_acc=> {} ".format(iter+1,␣
 ↪cost_2(params, x_train, y_train), acc_train))

    return acc_train
```

6 Model-1

6.1 Data for model-1

0->covid, 1->Normal, 2->Penumonia

Here the labels are modified accordinglly for Model-1

[25]:
```
x_train_1 = np.copy(x_train_final)
y_train_1 = []

for i in range(len(y_train)):
    if(y_train[i]==0 or y_train[i]==2):  ##Covid or pneumonia
        y_train_1.append(-1)
    elif(y_train[i]==1):  ##normal person
        y_train_1.append(1)

y_train_1 = np.array(y_train_1)
```

6.2 Binary Classifier Model-1

[30]:
```
params_1 = (0.01 * np.random.randn(2, 2, 3), 0.0)
params_1
```

[30]:
```
(tensor([[[ 0.00788197,  0.00862789, -0.01362923],
          [ 0.03096347, -0.00686296,  0.00633331]],

         [[-0.01199092, -0.01398299,  0.00185472],
          [ 0.00721716,  0.00810696,  0.00811893]]], requires_grad=True), 0.0)
```

[31]:
```
iters = 20
optimizer_1 = qml.AdagradOptimizer(stepsize=0.5)

cost_list_1 = []
train_acc_list_1 = []

for iter in range(iters):
    params_1 = optimizer_1.step(lambda v: cost_1(v, x_train_1, y_train_1),␣
 ↪params_1)

    cost_list_1.append(cost_1(params_1, x_train_1, y_train_1))
    train_acc_list_1.append(accuracy_1(params_1, x_train_1, y_train_1, iter))
```

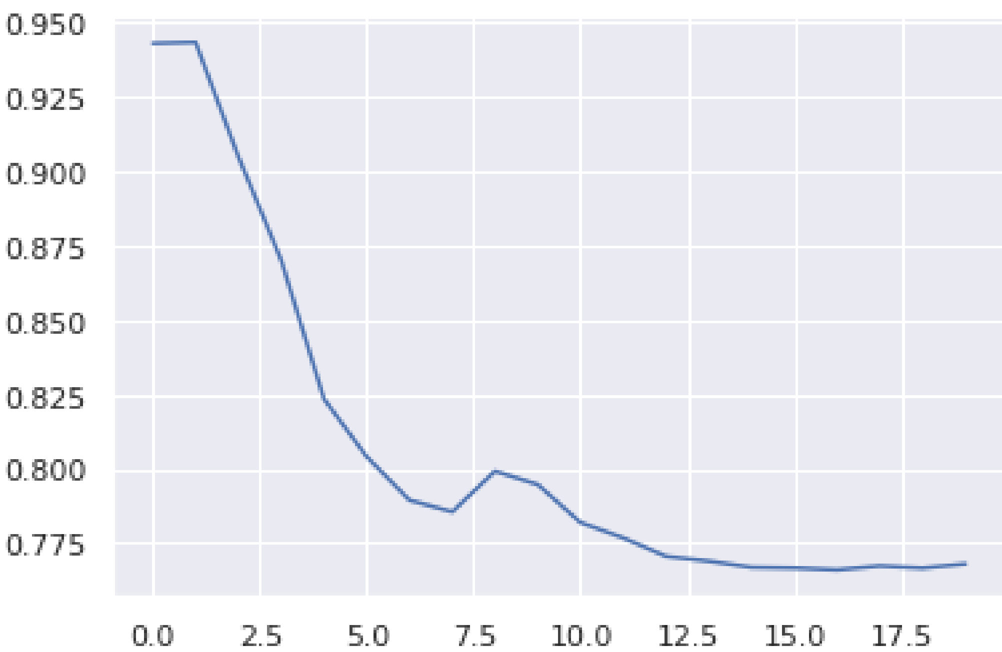
```
Iter=> 1   train_cost=> 0.9430680970436425   train_acc=> 0.636
Iter=> 2   train_cost=> 0.9434245396114554   train_acc=> 0.588
Iter=> 3   train_cost=> 0.9049820852738587   train_acc=> 0.688
Iter=> 4   train_cost=> 0.8701317914427136   train_acc=> 0.664
Iter=> 5   train_cost=> 0.8235834174948404   train_acc=> 0.72
```

```
Iter=> 6   train_cost=> 0.8041318123394667   train_acc=> 0.712
Iter=> 7   train_cost=> 0.7895082700859484   train_acc=> 0.712
Iter=> 8   train_cost=> 0.7856300343953608   train_acc=> 0.716
Iter=> 9   train_cost=> 0.7992519245856506   train_acc=> 0.696
Iter=> 10   train_cost=> 0.794766268933636   train_acc=> 0.692
Iter=> 11   train_cost=> 0.7820467651355429   train_acc=> 0.7
Iter=> 12   train_cost=> 0.7768421984495323   train_acc=> 0.692
Iter=> 13   train_cost=> 0.7706342494969206   train_acc=> 0.704
Iter=> 14   train_cost=> 0.769017766266119   train_acc=> 0.692
Iter=> 15   train_cost=> 0.7669294220914374   train_acc=> 0.708
Iter=> 16   train_cost=> 0.7667130993171604   train_acc=> 0.7
Iter=> 17   train_cost=> 0.7660947881445909   train_acc=> 0.704
Iter=> 18   train_cost=> 0.767394522447745   train_acc=> 0.708
Iter=> 19   train_cost=> 0.7666808888269254   train_acc=> 0.7
Iter=> 20   train_cost=> 0.7680832338923542   train_acc=> 0.704
```

6.3 Cost Plot of Model-1

[33]:
```
plt.plot(cost_list_1)
```

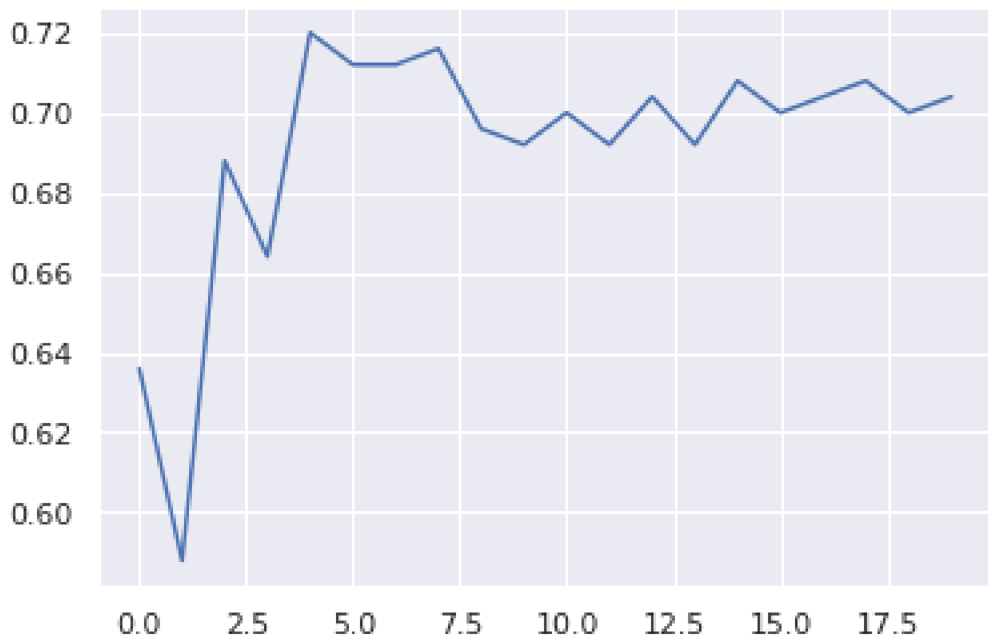
[33]: [<matplotlib.lines.Line2D at 0x7f90de0cbed0>]



6.4 Training Accuracy Plot of Model-1

[34]:
```
plt.plot(train_acc_list_1)
```

[34]: [<matplotlib.lines.Line2D at 0x7f90deb12a90>]



7 Model-2

7.1 Data for model-2

0->covid, 1->Normal, 2->Penumonia

Here the labels are modified accordingly for Model-2

[35]:
```
x_train_2 = []
y_train_2 = []

for i in range(len(y_train)):
    if(y_train[i]==0):   ## covid
        x_train_2.append(x_train_final[i])
        y_train_2.append(1)
    elif(y_train[i]==2):  ## pneumonia
        x_train_2.append(x_train_final[i])
        y_train_2.append(-1)
```

```
x_train_2 = np.array(x_train_2)
y_train_2 = np.array(y_train_2)
```

7.2 Binary Classifier Model-2

[36]:
```
params_2 = (0.01 * np.random.randn(2, 2, 3), 0.0)
params_2
```

[36]:
```
(tensor([[[-0.01642739, -0.00303444,  0.00879709],
          [ 0.00396784,  0.03160051, -0.00861578]],

         [[ 0.00302629,  0.00091985,  0.00371899],
          [ 0.01023066, -0.00811128, -0.00915326]]], requires_grad=True), 0.0)
```

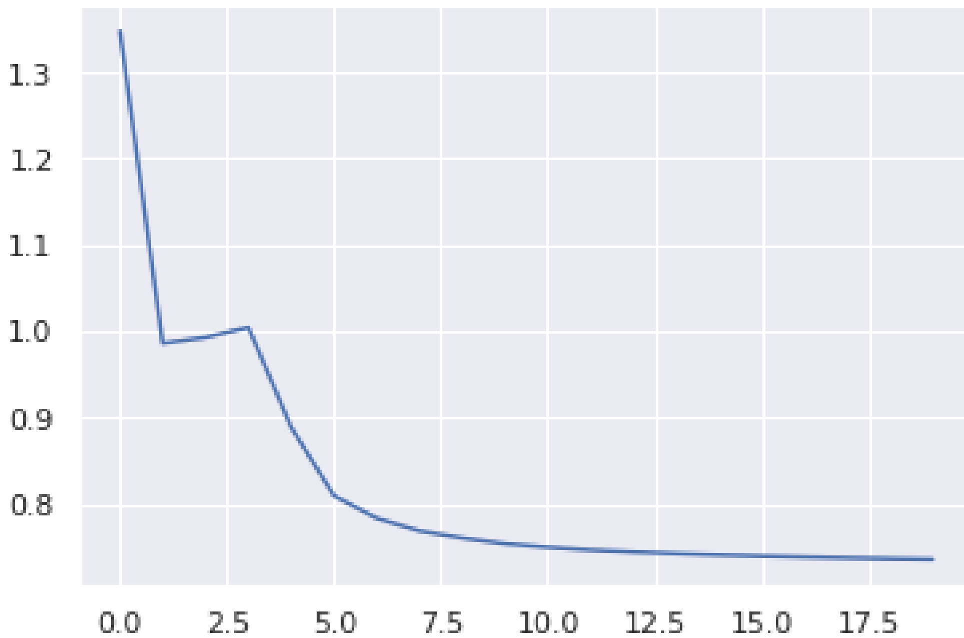
[37]:
```
iters = 20
optimizer_2 = qml.AdagradOptimizer(stepsize=0.5)

cost_list_2 = []
train_acc_list_2 = []

for iter in range(iters):
    params_2 = optimizer_2.step(lambda v: cost_2(v, x_train_2, y_train_2),␣
 ↪params_2)

    cost_list_2.append(cost_2(params_2, x_train_2, y_train_2))
    train_acc_list_2.append(accuracy_2(params_2, x_train_2, y_train_2, iter))
```

```
Iter=> 1   train_cost=> 1.347974932146223   train_acc=> 0.4444444444444444
Iter=> 2   train_cost=> 0.9856259090057776   train_acc=> 0.6388888888888888
Iter=> 3   train_cost=> 0.9925207251175971   train_acc=> 0.6222222222222222
Iter=> 4   train_cost=> 1.004129402328754   train_acc=> 0.6166666666666667
Iter=> 5   train_cost=> 0.8884518646533924   train_acc=> 0.6611111111111111
Iter=> 6   train_cost=> 0.8091313055032515   train_acc=> 0.7
Iter=> 7   train_cost=> 0.7829457458090165   train_acc=> 0.7111111111111111
Iter=> 8   train_cost=> 0.7681887981986943   train_acc=> 0.7166666666666667
Iter=> 9   train_cost=> 0.7599551749217474   train_acc=> 0.7222222222222222
Iter=> 10   train_cost=> 0.753346260706608   train_acc=> 0.7277777777777777
Iter=> 11   train_cost=> 0.7492179270476366   train_acc=> 0.7388888888888889
Iter=> 12   train_cost=> 0.7460577481431032   train_acc=> 0.7388888888888889
Iter=> 13   train_cost=> 0.7437081027338572   train_acc=> 0.7444444444444445
Iter=> 14   train_cost=> 0.7418335293994769   train_acc=> 0.7444444444444445
Iter=> 15   train_cost=> 0.7403042064384163   train_acc=> 0.75
Iter=> 16   train_cost=> 0.7390145390398503   train_acc=> 0.75
Iter=> 17   train_cost=> 0.7379022344378474   train_acc=> 0.7555555555555555
Iter=> 18   train_cost=> 0.7369217587371343   train_acc=> 0.7555555555555555
Iter=> 19   train_cost=> 0.7360424737515406   train_acc=> 0.75
Iter=> 20   train_cost=> 0.7352414799044594   train_acc=> 0.75
```

7.3 Cost Plot of Model-2

[38]:
```
plt.plot(cost_list_2)
```

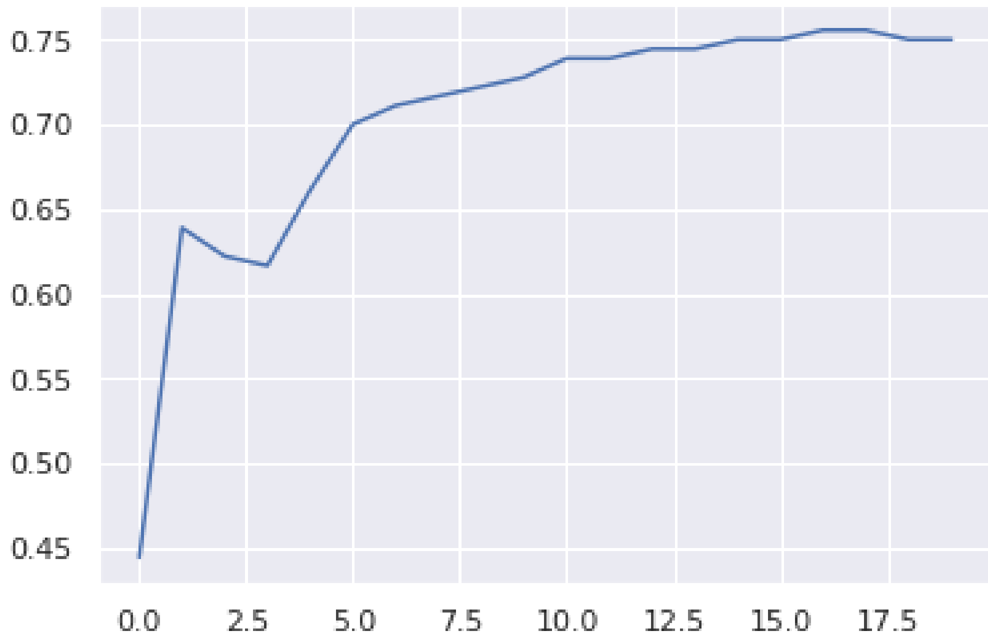
[38]: [<matplotlib.lines.Line2D at 0x7f90defe4150>]



7.4 Training Accuracy Plot of Model-2

[39]:
```
plt.plot(train_acc_list_2)
```

[39]: [<matplotlib.lines.Line2D at 0x7f90ddfd8d50>]



8 Prediction

8.0.1 While Predicting, we first give input to the Model-1. If it predicts as Normal person, then it is the final prediction assigned to the input. If not, then we give the same input to Model-2 and it finally predicts whether the chest x-ray is Covid10 patient or Viral Pneumonia patient.

[40]:
```
y_pred = []

for i in range(len(x_test_final)):
    tmp = classifier_prediction_1(params_1, x=x_test_final[i])
    if (tmp == 1):  ## Normal person
        y_pred.append(1)
    else:   ## Covid or Pneumonia
        tmp = classifier_prediction_2(params_2, x=x_test_final[i])
        if(tmp == 1):  ## covid
            y_pred.append(0)
        elif(tmp == -1): ## Pneumonia
            y_pred.append(2)

y_pred = np.array(y_pred)
```

[41]:
```
print("Final_Test_Accuracy => ", np.sum(y_pred==y_test)/len(y_test))
```

```
Final_Test_Accuracy =>  0.49230769230769234
```

8.0.2 We are hopeful to in increment in this accuracy when it will be trained on real computer with larger dataset.