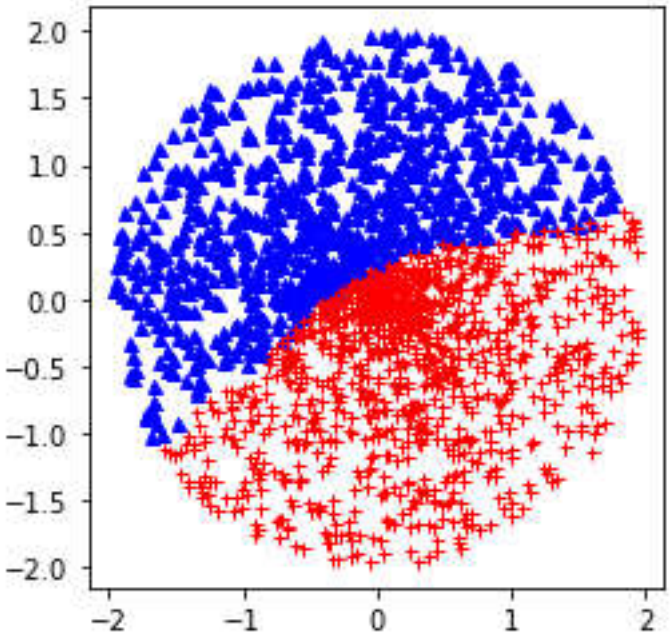
Problem 4

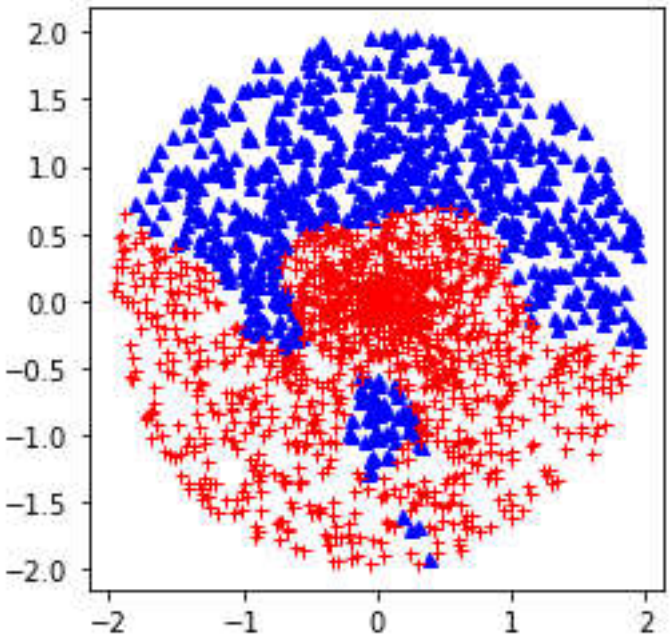
Epochs = 10

Training loss : 0.6843632
Validation loss : 0.6840534
test_accuracy: 0.623



Epochs = 100

Training loss : 0.6901484
Validation loss : 0.6899923
test_accuracy: 0.6875

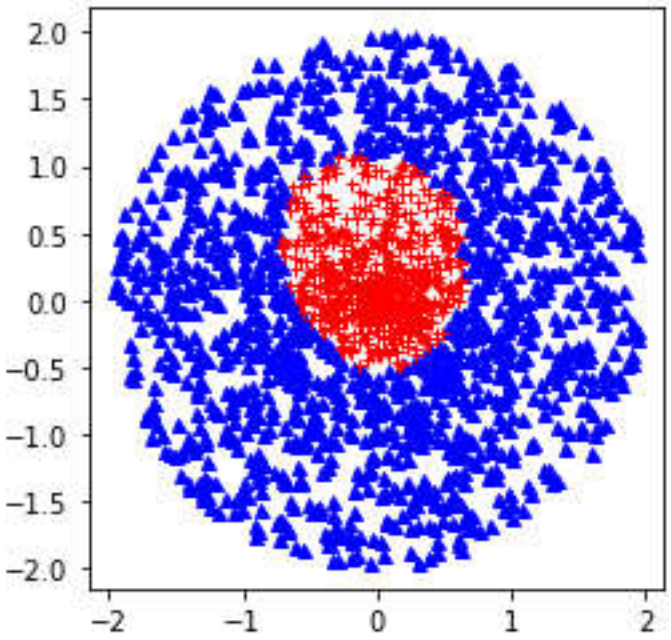


Epochs = 1000

Training loss : 0.6164542

Validation loss : 0.6161433

test_accuracy: 0.8305



Code:

```
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F
#%%
def getdata(n):

    radius = np.random.uniform(low=0,high=2,size=n).reshape(-1,1)
    angle = np.random.uniform(low=0,high=2*np.pi,size=n).reshape(-1,1)
    x1 = radius*np.cos(angle)
    x2 = radius*np.sin(angle)

    x = np.concatenate([x1,x2],axis=1)
    y = (radius<1).astype(int).reshape(-1)

    x = torch.from_numpy(x).float()
    y= torch.from_numpy(y).long()

    return x,y
#%%
np.random.seed(1)
(xtrain,ytrain) = getdata(10000)
(xval,yval) = getdata(2000)
(xtest,ytest) = getdata(2000)
#%%
input_size = 2
hidden_size = 128
num_classes = 2
```

```
#%%
class myNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(myNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)
        self.sig2 = nn.Sigmoid()

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        out = self.sig2(out)
        return out
#%%
def training_routine(myNN,dataset,n_iters):
    train_data,train_labels,val_data,val_labels = dataset
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(myNN.parameters(),lr=0.01)

    for i in range(n_iters):
        train_output = myNN(train_data)
        train_loss = criterion(train_output,train_labels)
        train_loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if i%1==0:
            print("At iteration",i)
            train_prediction = train_output.cpu().detach().argmax(dim=1)
```

```
            train_accuracy = (train_prediction.numpy()==train_labels.numpy()).mean()
            val_output = myNN(val_data)
            val_loss = criterion(val_output,val_labels)
            val_prediction = val_output.cpu().detach().argmax(dim=1)
            val_accuracy = (val_prediction.numpy()==val_labels.numpy()).mean()
            print("Training loss :",train_loss.cpu().detach().numpy())
            print("Training accuracy :",train_accuracy)
            print("Validation loss :",val_loss.cpu().detach().numpy())
            print("Validation accuracy :",val_accuracy)
#%%
def testing_routine(net_,dataset):
    data = dataset[0]
    y = dataset[1]
    yhat = net_(data).cpu().detach().argmax(dim=1)
    test_accuracy = (yhat.numpy()==y.numpy()).mean()
    print('test_accuracy:', test_accuracy)
    return yhat
#%%
def abulplot(xtest_, ytest_):
    class1 = xtest_[ytest_ == 1]
    class0 = xtest_[ytest_ == 0]
    plt.plot(class0[:,0],class0[:,1], 'b^', ms = 5)
    plt.plot(class1[:,0],class1[:,1], 'r+', ms = 5)
    plt.gca().set_aspect('equal')
    plt.show()
#%%
net = myNN(input_size, hidden_size, num_classes)
training_routine(net,(xtrain,ytrain,xval,yval),10)
yhat = testing_routine(net,(xtest, ytest))
#%%
abulplot(xtest.numpy(), yhat.numpy())
```