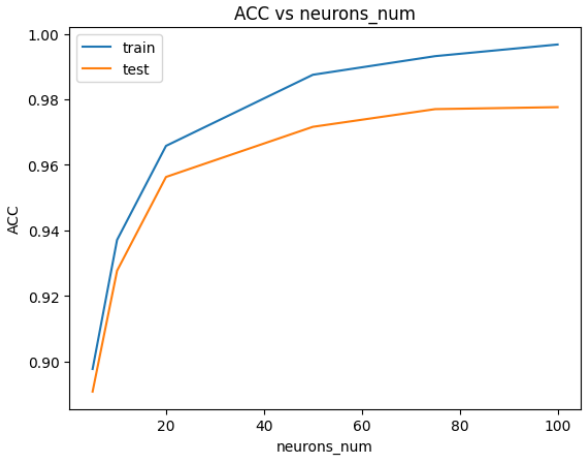
ML HW3 Report

110511277 蔡東宏

1. **Part I:**

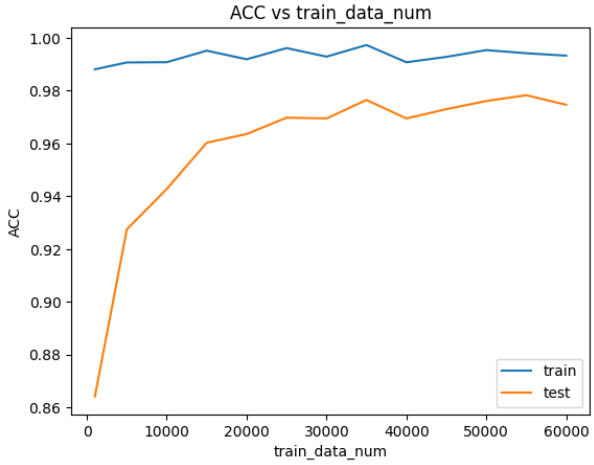
   A. Construct a DNN with one layer. Modify the number of neurons of the layer, 5, 10, 20, 50, 75, 100 with the whole data set with the whole training dataset. Plot the training and testing accuracy versus the number of neurons.




```
neurons_num: 5 train_ACC: 0.8977 test_ACC: 0.8908
neurons_num: 10 train_ACC: 0.9371 test_ACC: 0.9277
neurons_num: 20 train_ACC: 0.9657833333333333 test_ACC: 0.9563
neurons_num: 50 train_ACC: 0.9874833333333334 test_ACC: 0.9716
neurons_num: 75 train_ACC: 0.9931666666666666 test_ACC: 0.977
neurons_num: 100 train_ACC: 0.9967166666666667 test_ACC: 0.9776
```

在 layer 只有一層時，中間的 hidden neurons 數量越多時，training data 以及 testing data 的 accuracy 都有明顯的上升，且當 hidden neurons 為 100 時，training data 的 accuracy 已經達到 0.997，非常接近 100%

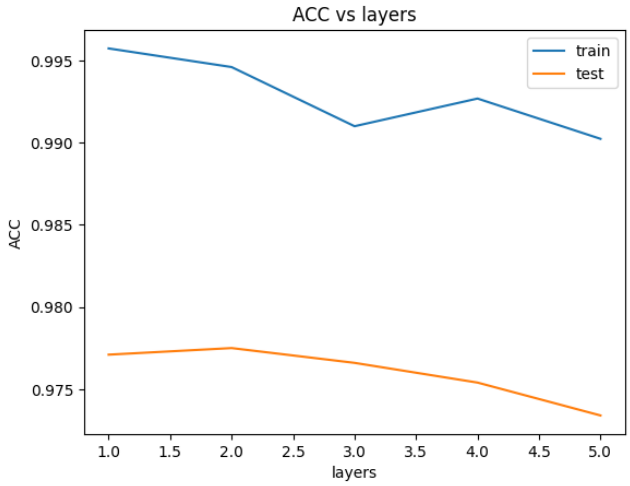
B. Construct a DNN with two layers. The number of neurons is 100 in each layer. Modify the number of training data and plot the training and testing accuracy versus the number of training data




```
train_data_num: 1000 train_ACC: 0.988 test_ACC: 0.8641
train_data_num: 5000 train_ACC: 0.9906 test_ACC: 0.9274
train_data_num: 10000 train_ACC: 0.9907 test_ACC: 0.9427
train_data_num: 15000 train_ACC: 0.9950666666666667 test_ACC: 0.9602
train_data_num: 20000 train_ACC: 0.9918 test_ACC: 0.9635
train_data_num: 25000 train_ACC: 0.99604 test_ACC: 0.9697
train_data_num: 30000 train_ACC: 0.9928 test_ACC: 0.9694
train_data_num: 35000 train_ACC: 0.9972 test_ACC: 0.9764
train_data_num: 40000 train_ACC: 0.990675 test_ACC: 0.9694
train_data_num: 45000 train_ACC: 0.9927111111111111 test_ACC: 0.9729
train_data_num: 50000 train_ACC: 0.99528 test_ACC: 0.976
train_data_num: 55000 train_ACC: 0.9940909090909091 test_ACC: 0.9782
train_data_num: 60000 train_ACC: 0.9931833333333333 test_ACC: 0.9746
```

在 layer 為兩層且每一層的 hidden node 數量為 100 時，無論 training data 的數量為多少，training data 的 accuracy 都非常接近 100%，但可以觀察到若 training data 數量越多時，testing data 的 accuracy 也會慢慢變大，在 training data 只有 1000 筆時，testing data 的 accuracy 只有 0.8641，而當 training data 來到 60000 筆時，testing data 的 accuracy 來到 0.9746。

C. Construct a DNN and modify the number of hidden layers from 1 to 5 with the whole training dataset. The number of neurons is 100 in each layer. Plot the training and testing accuracy versus the number of hidden layers.

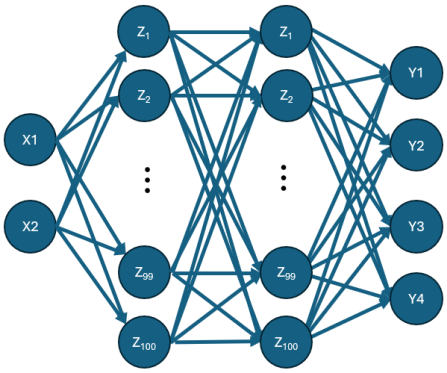



```
layers: 1 train_ACC: 0.9957333333333334 test_ACC: 0.9771
layers: 2 train_ACC: 0.9946 test_ACC: 0.9775
layers: 3 train_ACC: 0.991 test_ACC: 0.9766
layers: 4 train_ACC: 0.9926833333333334 test_ACC: 0.9754
layers: 5 train_ACC: 0.9902333333333333 test_ACC: 0.9734
```

改變 hidden layer 數量時，training data 以及 testing data 的 accuracy 幾乎沒有影響，training data 的 accuracy 都大約為 0.99，接近 100%，而 testing data 的 accuracy 都大約為 0.98。

2. Part II:

the structure of DNN:

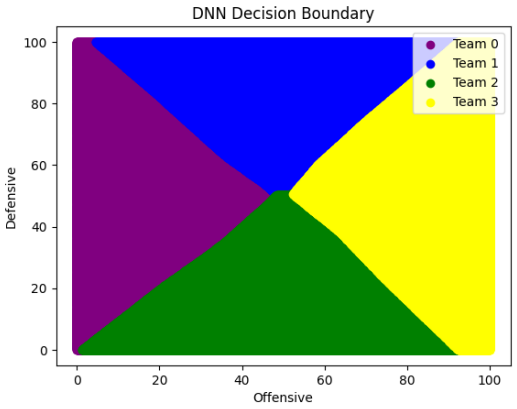


我的 DNN 架構如上圖，input 為二維的 data，為了能夠更準確地進行分類，因此我使用兩層 hidden layer，且每一層包含 100 的 hidden nodes，使用較複雜的 DNN 結構使得分類結果能夠更準確，最後有四個 output(因為要將 input 資料分為四類)。

Accuracy and Confusing Matrix:

```
Training Accuracy: 0.946923076923077
Train Confusion Matrix:
 [[285.   7.   8.   0.]
 [  7. 228.   1.  14.]
 [  5.   2. 335.   8.]
 [  0.  11.   6. 383.]]
Test Accuracy: 0.9106666666666666
Test Confusion Matrix:
 [[186.  10.   4.   0.]
 [  5. 268.   1.  26.]
 [  7.   0. 138.   5.]
 [  0.   3.   6.  91.]]
```

Decision Boundary:



Compare DNN with the generative model and discriminative model:

	Training Accuracy	Testing Accuracy	Training Confusing Matrix	Testing Confusing Matrix
DNN	0.947	0.911	[[285, 7, 8, 0] [7, 228, 1, 14] [5, 2, 335, 8] [0, 11, 6, 383]]	[[186, 10, 4, 0] [5, 268, 1, 26] [7, 0, 138, 5] [0, 3, 6, 91]]
Generative Model	0.943	0.912	[[287, 6, 7, 0] [9, 229, 0, 12] [10, 2, 329, 9] [0, 12, 7, 381]]	[[187, 10, 3, 0] [6, 271, 1, 22] [8, 0, 135, 7] [0, 3, 6, 91]]
Discriminative Model	0.942	0.91	[[285, 6, 9, 0] [9, 226, 1, 14] [9, 2, 331, 8] [0, 10, 7, 383]]	[[185, 10, 5, 0] [6, 267, 1, 26] [7, 0, 138, 5] [0, 2, 6, 92]]

觀察上述比較，我發現無論使用 DNN 或 Generative Model 或 Discriminative Model 所訓練出來的結果都差不多，training data 的 accuraccy 都大約落在 0.94 左右，testing data 的 accuracy 都大約落在 0.91，且 training data 以及 testing data 的 confusing matrix 都差不多。

Discussion:

1. **What is the difference between DNN and traditional methods (generative model, and discriminative model)**

   Generative model 是直接透過假設 model，並根據 training data 得到 model 所需要的參數，直接得到 P(X, Y)，在進一步透過推導得到 P(Y|X)，而 Discriminative model 是直接透過 training data 得到 P(Y|X)，而 DNN 是屬於 Discriminative model 的一種，差別在於 DNN 除了由輸入層以及輸出層外，還包含了一些隱藏層，這使得他能夠學習更大規模以及複雜的數據。在傳統的判別模型中，選擇不好的 basis function 會導致訓練出的模型準確率非常低。而 DNN 可以根據訓練數據自動學習較好的基函數，這使得訓練出的模型準確率非常穩定。然而，因為 DNN 架構包含的許多 hidden layers，因此也讓他的架構變得非常複雜，因此需要非常大的計算量，且訓練時間也非常久。

2. **How do you utilize the toolbox of Pytorch?**

   首先是先定義一層 hidden layer 的 neuron networks，hidden nodes 數量為 neurons_num，forward 函式先將 28*28 的 input 變成 1 維陣列並對第一層的輸出套進 ReLU function，最後經過第二層的 neuron netorks 並輸出。

```
class DNN_Single_Layer(nn.Module):
    def __init__(self, neurons_num):
        super(DNN_Single_Layer, self).__init__()
        self.fc = nn.Linear(28*28, neurons_num)
        self.output = nn.Linear(neurons_num, 10)

    def forward(self, x):
        x = x.view(-1, 28*28) # 1-D array
        x = F.relu(self.fc(x))
        x = self.output(x)
        return x
```

   接下來是定義損失函數以及優化器，用來得到合適的參數，這邊採用的方法是 adagrad，並且學習率為 0.001，AdaGrad 就是會依照梯度去調整 learning rate 的優化器，接下來會跑 epochs 個週期，每一個周期都經過 feed-forward (outputs = model(inputs)那行)並計算 loss，以及 error back-propagation(loss.backward())並進行參數的優化。

```
model = DNN_Single_Layer(neurons_num)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
for epoch in range(epochs):
    for inputs, labels in trainloader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

接下來是計算 accuracy 的部分，model.eval()將模型切換到評估模式，並在沒有梯度計算的情況下(with torch.no_grad())進行模型的評估準確率，利用 torch.max 得到哪個位置的機率最大，並將其分為那類，再比較預期的值與實際的值是否吻合，若吻合則為正確的分類。

```
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in trainloader:
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
train_ACC = correct / total
```

這裡的 torch.utils.data.Subset 是將 training data 切開，因為第二題需要比較不同 training data 數量對訓練結果的影響，因此需要透過此函式取出特定數量的 training data。

```
subset_trainset = torch.utils.data.Subset(trainset, list(range(train_data_num)))
trainloader = torch.utils.data.DataLoader(subset_trainset, batch_size=batch_size, shuffle=True)
```

第三題需要比較不同 hidden layers 對訓練結果的影響，因此第三題採用 nn.ModuleList 來存所有 hidden layers。

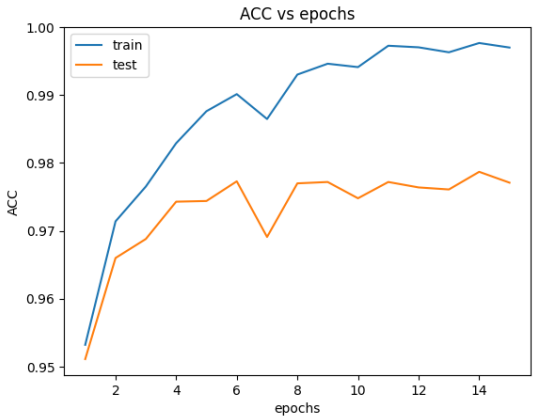
```
class DNN_multi_layer(nn.Module):
    def __init__(self, layers):
        super(DNN_multi_layer, self).__init__()
        self.fc = nn.ModuleList()
        self.fc.append(nn.Linear(28*28, 100))
        for i in range(1, layers):
            self.fc.append(nn.Linear(100, 100))
        self.output = nn.Linear(100, 10)

    def forward(self, x):
        x = x.view(-1, 28*28) # 1-D array
        for i in range(len(self.fc)):
            x = F.relu(self.fc[i](x))
        x = self.output(x)
        return x
```

Part II 的部分是要利用 DNN 取處理 hw2 的分類問題，因此在讀完檔之後先將數據利用 standaedScalar 進行標準化，並且將 x_train、y_train、x_test 以及 y_test 轉換成 tensor 並創建成 trainloader 以及 testloader。在利用 partI 的方法進行訓練以及計算 accuracy。最後再採用 hw2 的方法並套進模型算出預期值，就可以得到所要的 decision boundary。

```
EPOCHS = 10
train = pd.read_csv('HW2_training.csv')
test = pd.read_csv('HW2_testing.csv')
x_train = np.array(train.iloc[:, 1:])
y_trian = np.array(train.iloc[:, 0])
x_test = np.array(test.iloc[:, 1:])
y_test = np.array(test.iloc[:, 0])

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

x_train = torch.tensor(x_train, dtype=torch.float32)
y_train = torch.tensor(y_trian, dtype=torch.long)
x_test = torch.tensor(x_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.long)

train_dataset = TensorDataset(x_train, y_train)
test_dataset = TensorDataset(x_test, y_test)

trainloader = torch.utils.data.DataLoader(train_dataset, batch_size = 64, shuffle=True)
testloader = torch.utils.data.DataLoader(test_dataset, batch_size = 64, shuffle=False)
```

3. Compare different epochs

Epochs 是 DNN 模型中的訓練週期，若 epochs 太少，模型可能無法學到足夠的訊息，這會導致 training set 以及 testing set 的 accuracy 都相對較低，隨著 epochs 慢慢變大，模型能夠充分的學習訓練數據，這導致 training set 以及 testing set 的 accuracy 能夠達到非常高，然而，若 epochs 過多時，模型可能會出現 overfitting 的情況，這會導致 training set 的 accuracy 達到非常高，但 testing set 的 accuracy 卻降低了。觀察下圖可以發現當 epochs 越來越大時，training set 以及 testing set 的 accuracy 都慢慢變大，且還沒開始出現 overfitting。(給定一層 hidden layer 且 100 個 hidden nodes)



4. Using different optimizer(Adam vs SQD)

SGD 就是一般的 gradient descent，此時的 learning rate 為固定值，若此值太小會花費太多運算時間，若值太大會造成 overfitting，無法正確訓練 model。而 Adam 則是會依照梯度去調整 learning rate，因此使用 Adam 的 optimizer 能夠更快速收斂，因此它訓練的時間較快且準確度較高。給定一層 hidden layer 且 100 個 hidden nodes，比較 Adam 優化器以及 SGD 優化器的差別，發現在 training set 以及 testing set 中，Adam 優化器的 accuracy 皆大於 SGD 優化器。

```
ADAM : train_ACC: 0.9950666666666667 test_ACC: 0.9772
SGD : train_ACC: 0.8866166666666667 test_ACC: 0.893
```