Trainning Neural Network Process

1. Data preprocess

```
import numpy as np

X, y = # Data to be trained

# Convert into torch tensor

X_torch = torch.tensor(X).float()
y_torch = torch.tensor(y).float()
```

2. Define Neural Network

```
import torch
import torch.nn as nn

model = nn.Sequential(nn.Linerar(a, b),
                      nn.ReLU(),
                      nn.Linear(b, c),
                      nn.Sigmoid()
                      )
```

3. Define dataset and split data.

```
dataset = TensorDataset(X_torch, y_torch)
# Split dataset into separate datasets for training & testing
dataset_train, dataset_validate, dataset_test = random_split(dataset, lengths = [0.6,0.

dloader_train = DataLoader(dataset_train, batch_size = 32, shuffle = True)
dloader_validate = DataLoader(dataset_validate, batch_size = 32, shuffle = True)
```

4. Define loss function and optimizer

```
loss_fcn = nn.BCELoss()

optimizer = torch.optim.SGD(params = model.parameters(),
                            lr = 0.01
                            )
```

5. Define trainning loop function

```
def train_epoch():
    tot_loss = 0
    valid_loss = 0
    for X_train, y_train in dloader_train:
        y_pred = model(X_train)
        optimizer.zero_grad()
        loss = loss_fcn(y_pred, y_train.reshape(-1,1))
        tot_loss += loss.detach()
        loss.backward()
        optimizer.step()

    for X_valid, y_valid in dloader_validate:
        y_pred_v = model(X_valid)
        vloss = loss_fcn(y_pred_v, y_valid.reshape(-1,1))
        valid_loss += vloss.detach()

    return tot_loss/len(dataset_train), valid_loss/len(dataset_validate)
```

6. Plot data and decision boundary (Optional)

```
X_train = torch.vstack([dataset_train[i][0] for i in range(len(dataset_train))])
y_train = torch.vstack([dataset_train[i][1] for i in range(len(dataset_train))])
X_valid = torch.vstack([dataset_validate[i][0] for i in range(len(dataset_validate))])
y_valid = torch.vstack([dataset_validate[i][1] for i in range(len(dataset_validate))])
X_test = torch.vstack([dataset_test[i][0] for i in range(len(dataset_test))])
y_test = torch.vstack([dataset_test[i][1] for i in range(len(dataset_test))])

def plot_decision_boundary(ax, scatter_x, scatter_y):
    N = 1000
    X_grid = np.meshgrid(np.linspace(-7,7,N),np.linspace(-7,7,N))
    X_grid2 = np.array([X_grid[0].flatten(),X_grid[1].flatten()])
    preds = model(torch.tensor(X_grid2.transpose()).float()).reshape((N, N)).detach()
    ax.contourf(X_grid[0],X_grid[1], preds, cmap = plt.cm.cividis, alpha = 0.5)
    ax.scatter(scatter_x[:,0],scatter_x[:,1],c = scatter_y, cmap = plt.cm.cividis, edge
    ax.set_xlabel('$X_1$',fontsize = 16)
    ax.set_ylabel('$X_2$',fontsize = 16)
    ax.set_xlim(-4, 4)
    ax.set_ylim(-3.5,3.5)
    ax.xaxis.set_minor_locator(MultipleLocator(0.2))
    ax.xaxis.set_major_locator(MultipleLocator(1))
    ax.yaxis.set_minor_locator(MultipleLocator(0.2))

fig, ax = plt.subplots(1,1,figsize = (8,6),dpi = 150)
plot_decision_boundary(ax, X_train, y_train)
```

7. Trainning for arbitary epochs

```
t_loss, v_loss = [],[]
for i in range(25):
    train_loss,valid_loss = train_epoch()
    t_loss.append(train_loss)
    v_loss.append(valid_loss)
```

8. Plot trainning loss curve

```
fig, ax = plt.subplots(1,1,figsize = (8,6),dpi = 150)

ax.plot(t_loss, color='black',label='Training loss')
ax.plot(v_loss, color='#D55E00',label='Validation loss')
ax.set_xlabel('Epoch',fontsize = 16)
ax.set_ylabel('Binary cross entropy',fontsize = 16)
ax.set_title('Loss during training',fontsize = 20)
ax.tick_params(labelsize =12, which = 'both',top=True, right = True, direction='in')
ax.xaxis.set_minor_locator(MultipleLocator(1))
ax.yaxis.set_minor_locator(MultipleLocator(4))
ax.grid(color='xkcd:dark blue',alpha = 0.2)
ax.legend(loc='upper right',fontsize = 12)
```

9. Calculate accuracy

```
# Need to decide what class to predict, and make sure the prediction tensor is in the r
from sklearn.metrics import accuracy_score

train_pred = torch.Tensor([0 if x < 0.5 else 1 for x in model(X_train)]).reshape(y_trai
train_accuracy = accuracy_score(y_true = y_train,
                                y_pred = train_pred)

valid_pred = torch.Tensor([0 if x < 0.5 else 1 for x in model(X_valid)]).reshape(y_vali
valid_accuracy = accuracy_score(y_true = y_valid,
                                y_pred = valid_pred)

print('Training accuracy = {:.1f}%'.format(train_accuracy*100))
print('Validation accuracy = {:.1f}%'.format(valid_accuracy*100))
```

10. Plot ROC curve

```python
from sklearn.metrics import roc_curve, roc_auc_score

roc_score = roc_auc_score(y_true = y_valid.detach().numpy(),
                          y_score = model(X_valid).detach().numpy())

fpr, tpr, thresholds = roc_curve(y_true = y_valid.detach().numpy(),
                                 y_score = model(X_valid).detach().numpy())

fig, ax = plt.subplots(1,1,figsize = (6, 4), dpi = 150)
ax.plot(fpr, tpr, color='#D55E00')
ax.set_xlabel('False positive rate',fontsize = 20)
ax.set_ylabel('True positive rate',fontsize = 20)
ax.set_title('ROC curve, validation data',fontsize = 24)
ax.xaxis.set_minor_locator(MultipleLocator(0.04))
ax.yaxis.set_minor_locator(MultipleLocator(0.04))
ax.tick_params(which='both',direction='in',top=True,right=True,labelsize = 16)
ax.grid(color='xkcd:dark blue',alpha = 0.2)

print('ROC score = {:.3f}'.format(roc_score))
```