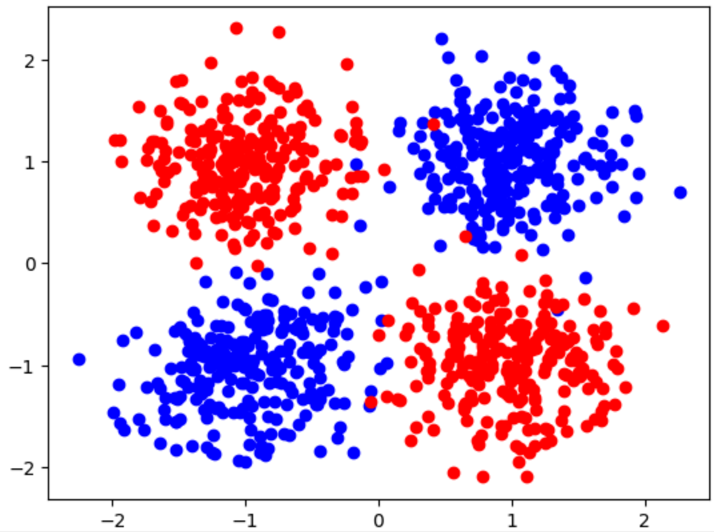
```
import torch
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt


#DEFINE YOUR DEVICE
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

print(device) #if cpu, go Runtime-> Change runtime type-> Hardware accelerator GPU -> Save -> Redo previous steps
```

```
cuda:0
```

```
#CREATE A RANDOM DATASET
centers = [[1, 1], [1, -1], [-1, -1], [-1, 1]] #center of each class
cluster_std=0.4 #standard deviation of random gaussian samples

x_train, y_train = make_blobs(n_samples=1000, centers=centers, n_features=2, cluster_std=cluster_std, shuffle=True)
y_train[y_train==2] = 0 #make this an xor problem
y_train[y_train==3] = 1 #make this an xor problem
x_train = torch.FloatTensor(x_train)
y_train = torch.FloatTensor(y_train)

x_val, y_val = make_blobs(n_samples=100, centers=centers, n_features=2, cluster_std=cluster_std, shuffle=True)
y_val[y_val==2] = 0 #make this an xor problem
y_val[y_val==3] = 1 #make this an xor problem
x_val = torch.FloatTensor(x_val)
y_val = torch.FloatTensor(y_val)


#CHECK THE BLOBS ON XY PLOT
plt.scatter(x_train[y_train==0,0],x_train[y_train==0,1],marker='o',color='blue')
plt.scatter(x_train[y_train==1,0],x_train[y_train==1,1],marker='o',color='red')
```

```
<matplotlib.collections.PathCollection at 0x7b368e243eb0>
```




```
#DEFINE NEURAL NETWORK MODEL
class FullyConnected(torch.nn.Module):
  def __init__(self, input_size, hidden_size, num_classes):
    super(FullyConnected, self).__init__()
    self.input_size = input_size
    self.hidden_size  = hidden_size
    self.fc1 = torch.nn.Linear(self.input_size, self.hidden_size)
    self.fc2 = torch.nn.Linear(self.hidden_size, num_classes)
    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()
  def forward(self, x):
    hidden = self.fc1(x)
    relu = self.relu(hidden)
    output = self.fc2(relu)
    return output


class FullyConnected2(torch.nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
      super(FullyConnected2, self).__init__()
      self.input_size = input_size
      self.hidden_size  = hidden_size
      self.fc1 = torch.nn.Linear(self.input_size, self.hidden_size)
```

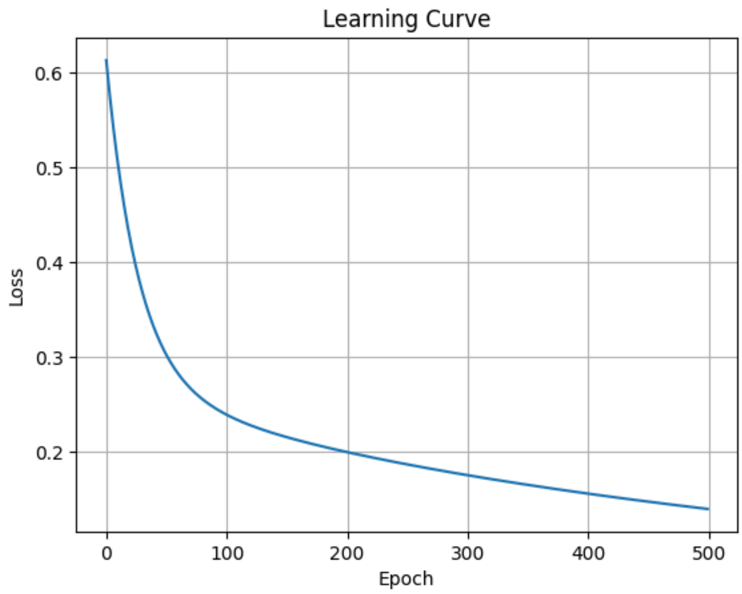
```
        self.fc2 = torch.nn.Linear(self.hidden_size, self.hidden_size*2,)
        self.fc3 = torch.nn.Linear(self.hidden_size*2, self.hidden_size*4,)
        self.fc4 = torch.nn.Linear(self.hidden_size*4, num_classes)
        self.relu = torch.nn.ReLU()
        self.sigmoid = torch.nn.Sigmoid()
    def forward(self, x):
        hidden = self.fc1(x)
        relu = self.relu(hidden)
        hidden2 = self.fc2(relu)
        relu2 = self.relu(hidden2)
        hidden3 = self.fc3(relu2)
        relu3 = self.relu(hidden3)
        output = self.fc4(relu3)
        return output


# Fully Connected Network for Question 5
class FullyConnected3(torch.nn.Module):
  def __init__(self, input_size, hidden_size, num_classes):
    super(FullyConnected3, self).__init__()
    self.input_size = input_size
    self.hidden_size  = hidden_size
    self.fc1 = torch.nn.Linear(self.input_size, self.hidden_size)
    self.fc2 = torch.nn.Linear(self.hidden_size, num_classes)
    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()
    self.dropout = torch.nn.Dropout(0.5)
  def forward(self, x):
    hidden = self.fc1(x)
    relu = self.relu(hidden)
    dropped = self.dropout(relu)
    output = self.fc2(dropped)
    return output


#CREATE MODEL
input_size = 2
hidden_size = 64
num_classes = 1

model = FullyConnected(input_size, hidden_size, num_classes)
model.to(device)
```

```
FullyConnected(
  (fc1): Linear(in_features=2, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=1, bias=True)
  (relu): ReLU()
  (sigmoid): Sigmoid()
)
```

```
#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.001
momentum = 0

loss_fun = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate, momentum = momentum)


#TRAIN THE MODEL
model.train()
epoch = 500
x_train = x_train.to(device)
y_train = y_train.to(device)

loss_values = np.zeros(epoch)

for i in range(epoch):
    optimizer.zero_grad()
    y_pred = model(x_train)    # forward
    #reshape y_pred from (n_samples,1) to (n_samples), so y_pred and y_train have the same shape
    y_pred = y_pred.reshape(y_pred.shape[0])
    loss = loss_fun(y_pred, y_train)

    loss_values[i] = loss.item()
    print('Epoch {}: train loss: {}'.format(i, loss.item()))
    loss.backward() #backward
    optimizer.step()
```

```
Epoch 448: train loss: 0.14753358066082
Epoch 449: train loss: 0.14737091958522797
Epoch 450: train loss: 0.14720851182937622
Epoch 451: train loss: 0.14704638719558716
Epoch 452: train loss: 0.14688456058502197
Epoch 453: train loss: 0.14672298729419708
Epoch 454: train loss: 0.1465616524219513
Epoch 455: train loss: 0.1464006006717682
Epoch 456: train loss: 0.14623980224132538
Epoch 457: train loss: 0.14607925713062286
Epoch 458: train loss: 0.14591901004314423
Epoch 459: train loss: 0.14575901627540588
Epoch 460: train loss: 0.14559926092624664
Epoch 461: train loss: 0.14543980360031128
Epoch 462: train loss: 0.1452805995941162
Epoch 463: train loss: 0.14512164890766144
Epoch 464: train loss: 0.14496295154094696
Epoch 465: train loss: 0.14480453729629517
Epoch 466: train loss: 0.14464637637138367
Epoch 467: train loss: 0.14448848366737366
Epoch 468: train loss: 0.14433082938194275
Epoch 469: train loss: 0.14417344331741333
Epoch 470: train loss: 0.1440163254737854
Epoch 471: train loss: 0.14385944604873657
Epoch 472: train loss: 0.14370281994342804
Epoch 473: train loss: 0.14354649186134338
Epoch 474: train loss: 0.14339038729667664
Epoch 475: train loss: 0.14323453605175018
Epoch 476: train loss: 0.14307892322540283
Epoch 477: train loss: 0.14292359352111816
Epoch 478: train loss: 0.1427685022354126
Epoch 479: train loss: 0.14261364936828613
Epoch 480: train loss: 0.14245906472206116
Epoch 481: train loss: 0.14230471849441528
Epoch 482: train loss: 0.1421506404876709
Epoch 483: train loss: 0.14199680089950562
Epoch 484: train loss: 0.14184322953224182
Epoch 485: train loss: 0.14168986678123474
Epoch 486: train loss: 0.14153678715229034
Epoch 487: train loss: 0.14138393104076385
Epoch 488: train loss: 0.14123134315013885
Epoch 489: train loss: 0.14107897877693176
Epoch 490: train loss: 0.14092688262462616
Epoch 491: train loss: 0.14077499508857727
Epoch 492: train loss: 0.14062339067459106
Epoch 493: train loss: 0.14047200977802277
Epoch 494: train loss: 0.14032086730003357
Epoch 495: train loss: 0.14016996324062347
Epoch 496: train loss: 0.14001931250095367
Epoch 497: train loss: 0.13986888527870178
Epoch 498: train loss: 0.139718696475029
Epoch 499: train loss: 0.1395687758922577
```

```
#PLOT THE LEARNING CURVE
plt.plot(loss_values)
plt.title('Learning Curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid('on')
```



```
#TEST THE MODEL
model.eval()

x_val = x_val.to(device)
y_val = y_val.to(device)

y_pred = model(x_val)
#reshape y_pred from (n_samples,1) to (n_samples), so y_pred and y_val have the same shape
y_pred = y_pred.reshape(y_pred.shape[0])
after_train = loss_fun(y_pred, y_val)
print('Validation loss after Training' , after_train.item())

correct=0
total=0
for i in range(y_pred.shape[0]):
  if y_val[i]==torch.round(y_pred[i]):
    correct += 1
  total +=1

print('Validation accuracy: %.2f%%' %((100*correct)//(total)))
```

```
Validation loss after Training 0.12944304943084717
Validation accuracy: 90.00%
```