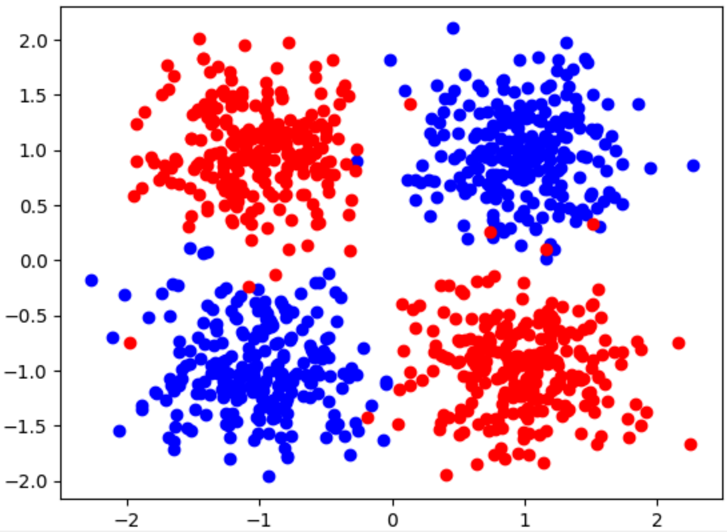
```
import torch
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt


#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 0x791b0b2f4f70>
```




```
#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)
```

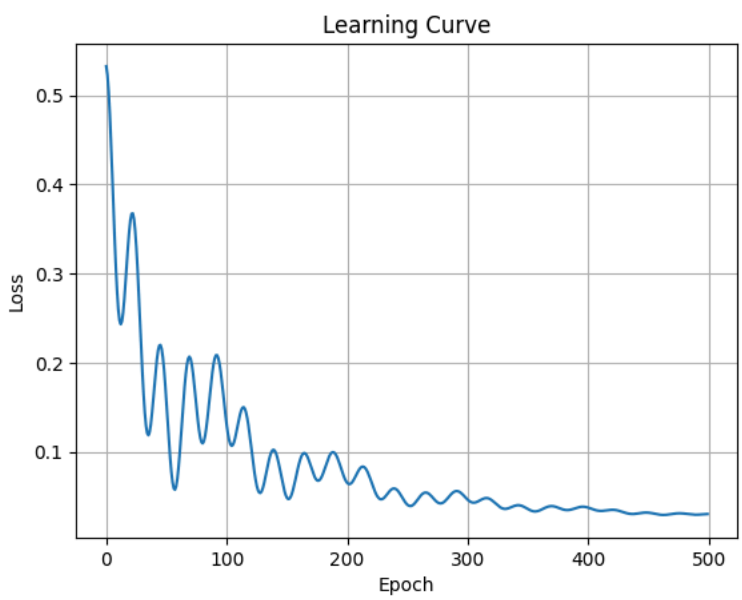
```
        self.fc2 = torch.nn.Linear(self.hidden_size, self.hidden_size*4,)
        self.fc3 = torch.nn.Linear(self.hidden_size*4, self.hidden_size*2,)
        self.fc4 = torch.nn.Linear(self.hidden_size*2, num_classes)
        self.relu = torch.nn.ReLU()
        self.sigmoid = torch.nn.Sigmoid()
      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 = 0.99

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 447: train loss: 0.03152424842119217
Epoch 448: train loss: 0.031557682901620865
Epoch 449: train loss: 0.031530290842056274
Epoch 450: train loss: 0.03144175931811333
Epoch 451: train loss: 0.031295470893383026
Epoch 452: train loss: 0.031097887083888054
Epoch 453: train loss: 0.030858393758535385
Epoch 454: train loss: 0.03058871626853943
Epoch 455: train loss: 0.03030208870768547
Epoch 456: train loss: 0.030012544244527817
Epoch 457: train loss: 0.029734168201684952
Epoch 458: train loss: 0.02948031760752201
Epoch 459: train loss: 0.029262932017445564
Epoch 460: train loss: 0.029091693460941315
Epoch 461: train loss: 0.028973760083317757
Epoch 462: train loss: 0.02891341969370842
Epoch 463: train loss: 0.028911741450428963
Epoch 464: train loss: 0.028966829180717468
Epoch 465: train loss: 0.02907375991344452
Epoch 466: train loss: 0.029224848374724388
Epoch 467: train loss: 0.029410339891910553
Epoch 468: train loss: 0.029618840664625168
Epoch 469: train loss: 0.02983803302049637
Epoch 470: train loss: 0.030055468901991844
Epoch 471: train loss: 0.03025909699499607
Epoch 472: train loss: 0.030438033863902092
Epoch 473: train loss: 0.030582968145608902
Epoch 474: train loss: 0.03068666160106659
Epoch 475: train loss: 0.030744504183530807
Epoch 476: train loss: 0.030754433944821358
Epoch 477: train loss: 0.030717190355062485
Epoch 478: train loss: 0.030636079609394073
Epoch 479: train loss: 0.030516676604747772
Epoch 480: train loss: 0.030366623774170876
Epoch 481: train loss: 0.030194994062185287
Epoch 482: train loss: 0.030011780560016632
Epoch 483: train loss: 0.0298272967338562
Epoch 484: train loss: 0.029651544988155365
Epoch 485: train loss: 0.02949369139969349
Epoch 486: train loss: 0.029361506924033165
Epoch 487: train loss: 0.02926097996532917
Epoch 488: train loss: 0.029195992276072502
Epoch 489: train loss: 0.029168222099542618
Epoch 490: train loss: 0.029176976531744003
Epoch 491: train loss: 0.029219264164566994
Epoch 492: train loss: 0.02929018810391426
Epoch 493: train loss: 0.02938300371170044
Epoch 494: train loss: 0.029489651322364807
Epoch 495: train loss: 0.029601242393255234
Epoch 496: train loss: 0.029708653688430786
Epoch 497: train loss: 0.029802991077303886
Epoch 498: train loss: 0.0298761073499918
Epoch 499: train loss: 0.029921049252152443
```

```
#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 0.030277404934167862
Validation accuracy: 100.00%
```

Kodlamaya başlayın veya yapay zeka ile kod oluşturun.