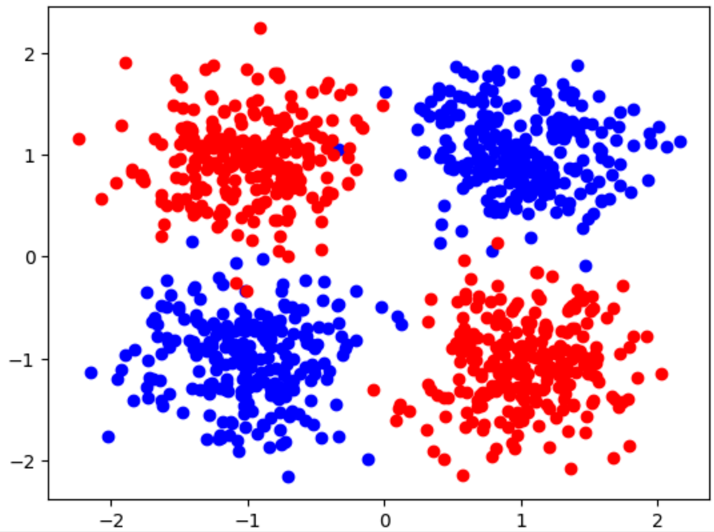
```
import torch
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt


#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 0x7e3f7602f4c0>
```




```
#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)
```

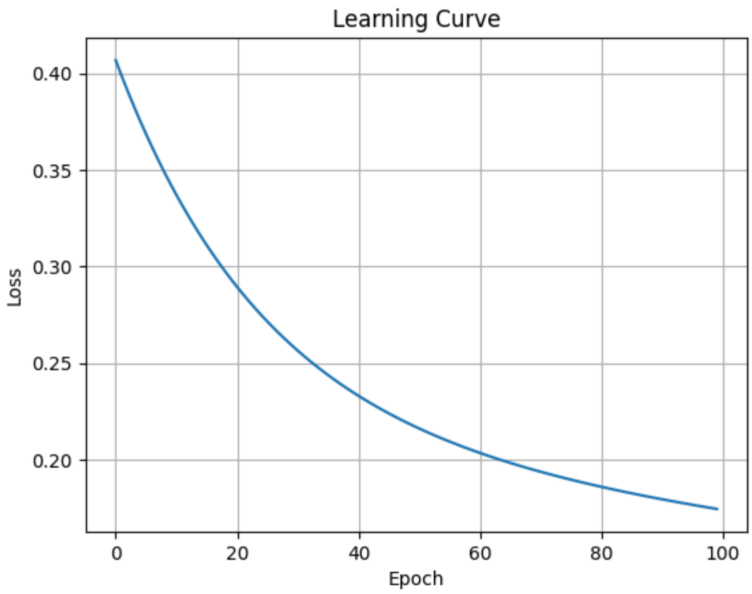
```
        self.fc2 = torch.nn.Linear(self.hidden_size, self.hidden_size*2,)
        self.fc3 = torch.nn.Linear(self.hidden_size*2, self.hidden_size*4,)
        self.fc4 = torch.nn.Linear(self.hidden_size*4, num_classes)
        self.relu = torch.nn.ReLU()
        self.sigmoid = torch.nn.Sigmoid()
      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 = 100
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 48: train loss: 0.21907207369804382
Epoch 49: train loss: 0.21755805611610413
Epoch 50: train loss: 0.21608808636665344
Epoch 51: train loss: 0.2146604061126709
Epoch 52: train loss: 0.21327340602874756
Epoch 53: train loss: 0.2119254469871521
Epoch 54: train loss: 0.21061500906944275
Epoch 55: train loss: 0.2093406617641449
Epoch 56: train loss: 0.20810095965862274
Epoch 57: train loss: 0.20689459145069122
Epoch 58: train loss: 0.2057202309370041
Epoch 59: train loss: 0.20457664132118225
Epoch 60: train loss: 0.2034626454114914
Epoch 61: train loss: 0.2023770809173584
Epoch 62: train loss: 0.2013188600540161
Epoch 63: train loss: 0.20028690993785858
Epoch 64: train loss: 0.199280247092247
Epoch 65: train loss: 0.19829793274402618
Epoch 66: train loss: 0.1973390281200409
Epoch 67: train loss: 0.1964026242494583
Epoch 68: train loss: 0.1954878866672516
Epoch 69: train loss: 0.19459401071071625
Epoch 70: train loss: 0.19372016191482544
Epoch 71: train loss: 0.19286559522151947
Epoch 72: train loss: 0.19202961027622223
Epoch 73: train loss: 0.1912115216255188
Epoch 74: train loss: 0.19041065871715546
Epoch 75: train loss: 0.18962635099887848
Epoch 76: train loss: 0.1888580471277237
Epoch 77: train loss: 0.18810507655143738
Epoch 78: train loss: 0.18736694753170013
Epoch 79: train loss: 0.1866431087255478
Epoch 80: train loss: 0.18593299388885498
Epoch 81: train loss: 0.1852361261844635
Epoch 82: train loss: 0.18455202877521515
Epoch 83: train loss: 0.18388023972511292
Epoch 84: train loss: 0.18322034180164337
Epoch 85: train loss: 0.1825718730688095
Epoch 86: train loss: 0.18193446099758148
Epoch 87: train loss: 0.18130765855312347
Epoch 88: train loss: 0.18069110810756683
Epoch 89: train loss: 0.1800844818353653
Epoch 90: train loss: 0.1794874370098114
Epoch 91: train loss: 0.1788996309041977
Epoch 92: train loss: 0.1783207505941391
Epoch 93: train loss: 0.17775046825408936
Epoch 94: train loss: 0.17718851566314697
Epoch 95: train loss: 0.17663460969924927
Epoch 96: train loss: 0.17608848214149475
Epoch 97: train loss: 0.17554986476898193
Epoch 98: train loss: 0.17501850426197052
Epoch 99: train loss: 0.17449414730072021
```

```
#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 0.16911335289478302
Validation accuracy: 83.00%
```