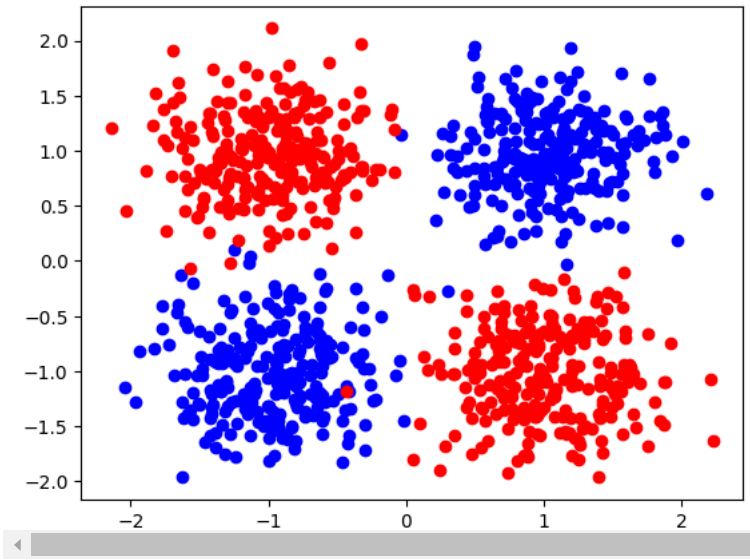
```
import torch
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt


#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 0x7c7bdddcfdf0>
```




```
#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)
```

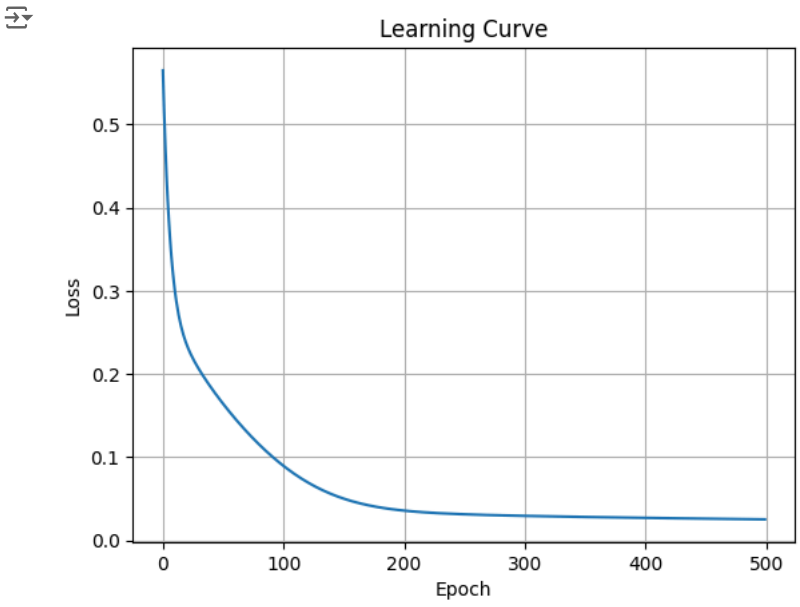
```
        self.fc2 = torch.nn.Linear(self.hidden_size, self.hidden_size*2,)
        self.fc3 = torch.nn.Linear(self.hidden_size*2, self.hidden_size*4,)
        self.fc4 = torch.nn.Linear(self.hidden_size*4, num_classes)
        self.relu = torch.nn.ReLU()
        self.sigmoid = torch.nn.Sigmoid()
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


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

model = FullyConnected2(input_size, hidden_size, num_classes)
model.to(device)
```

```
FullyConnected2(
  (fc1): Linear(in_features=2, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=256, bias=True)
  (fc4): Linear(in_features=256, out_features=1, bias=True)
  (relu): ReLU()
  (sigmoid): Sigmoid()
)
```

```
#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.01
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 211: train loss: 0.03440934792160988
Epoch 212: train loss: 0.034297846257686615
Epoch 213: train loss: 0.034188926219940186
Epoch 214: train loss: 0.03408253565430641
Epoch 215: train loss: 0.03397851064801216
Epoch 216: train loss: 0.03387684002518654
Epoch 217: train loss: 0.033777497708797455
Epoch 218: train loss: 0.03368048742413521
Epoch 219: train loss: 0.033585671335458755
Epoch 220: train loss: 0.033492956310510635
Epoch 221: train loss: 0.033402301371097565
Epoch 222: train loss: 0.03331361338496208
Epoch 223: train loss: 0.0332268625497818
Epoch 224: train loss: 0.03314197435975075
Epoch 225: train loss: 0.03305898606777191
Epoch 226: train loss: 0.032977793365716934
Epoch 227: train loss: 0.032898325473070145
Epoch 228: train loss: 0.03282048925757408
Epoch 229: train loss: 0.03274424746632576
Epoch 230: train loss: 0.032669562846422195
Epoch 231: train loss: 0.03259643539786339
Epoch 232: train loss: 0.03252479061484337
Epoch 233: train loss: 0.032454583793878555
Epoch 234: train loss: 0.032385747879743576
Epoch 235: train loss: 0.03231820836663246
Epoch 236: train loss: 0.03225194662809372
Epoch 237: train loss: 0.03218693658709526
Epoch 238: train loss: 0.03212317079305649
Epoch 239: train loss: 0.0320606455206871
Epoch 240: train loss: 0.03199929744005203
Epoch 241: train loss: 0.03193903714418411
Epoch 242: train loss: 0.03187986835837364
Epoch 243: train loss: 0.03182176873087883
```

```
#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 0.02426435984671116
Validation accuracy: 99.00%
```