

```

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

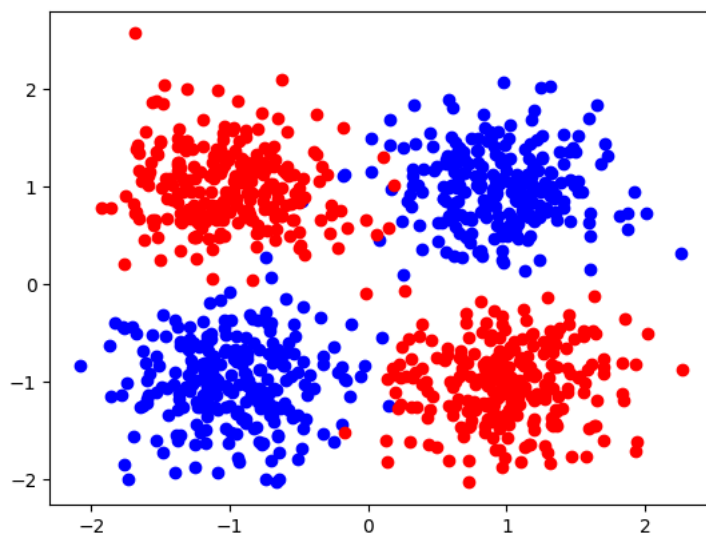
x_train, y_train = make_blobs(n_samples=1000, centers=centers, n_features=2, cluster_std=cluster_std, shuffle=True)
y_train[y_train==2] = 0 #make this an xor problem
y_train[y_train==3] = 1 #make this an xor problem
x_train = torch.FloatTensor(x_train)
y_train = torch.FloatTensor(y_train)

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 0x7dc7c5e4fa30>

```



```

#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 = 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 = 0.01
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)

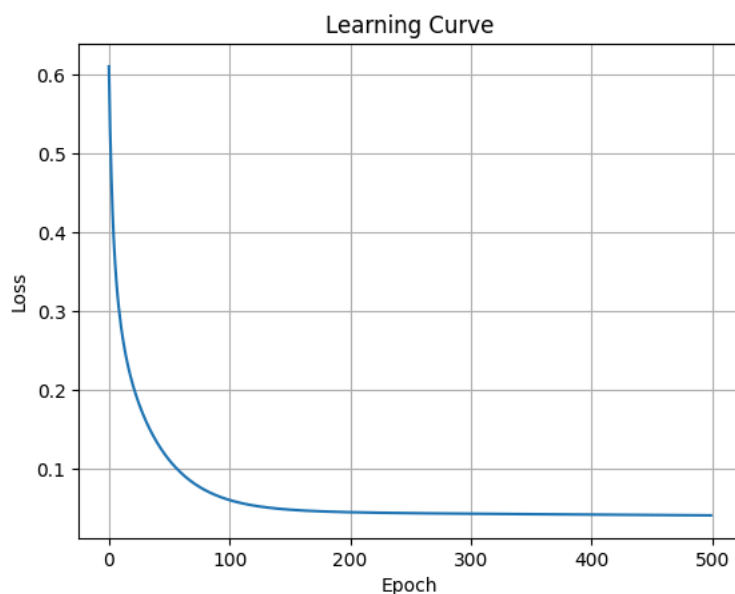
for i in range(epoch):
    optimizer.zero_grad()
    y_pred = model(x_train)    # forward
    #reshape y_pred from (n_samples,1) to (n_samples), so y_pred and y_train have the same shape
    y_pred = y_pred.reshape(y_pred.shape[0])
    loss = loss_fun(y_pred, y_train)

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



```
Epoch 468: train loss: 0.0401499119820442
Epoch 469: train loss: 0.04070548340678215
Epoch 470: train loss: 0.04069599136710167
Epoch 471: train loss: 0.04068650305271149
Epoch 472: train loss: 0.0406770296394825
Epoch 473: train loss: 0.040667567402124405
Epoch 474: train loss: 0.04065811261534691
Epoch 475: train loss: 0.040648672729730606
Epoch 476: train loss: 0.0406392365694046
Epoch 477: train loss: 0.04062981531023979
Epoch 478: train loss: 0.04062040522694588
Epoch 479: train loss: 0.04061099514365196
Epoch 480: train loss: 0.04060160368680954
Epoch 481: train loss: 0.040592219680547714
Epoch 482: train loss: 0.040582843124866486
Epoch 483: train loss: 0.040573474019765854
Epoch 484: train loss: 0.040564119815826416
Epoch 485: train loss: 0.04055476933717728
Epoch 486: train loss: 0.040545426309108734
Epoch 487: train loss: 0.040536101907491684
Epoch 488: train loss: 0.040526773780584335
Epoch 489: train loss: 0.04051745682954788
Epoch 490: train loss: 0.040508151054382324
Epoch 491: train loss: 0.04049885645508766
Epoch 492: train loss: 0.040489573031663895
Epoch 493: train loss: 0.040480297058820724
Epoch 494: train loss: 0.04047102853655815
Epoch 495: train loss: 0.04046177119016647
Epoch 496: train loss: 0.04045252129435539
Epoch 497: train loss: 0.04044327884912491
Epoch 498: train loss: 0.04043405130505562
Epoch 499: train loss: 0.04042482003569603
```

```
#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 0.028188766911625862
Validation accuracy: 100.00%