EE583 Pattern Recognition HW6

Kutay Uğurlu 2232841

January 8, 2022

1 Q1

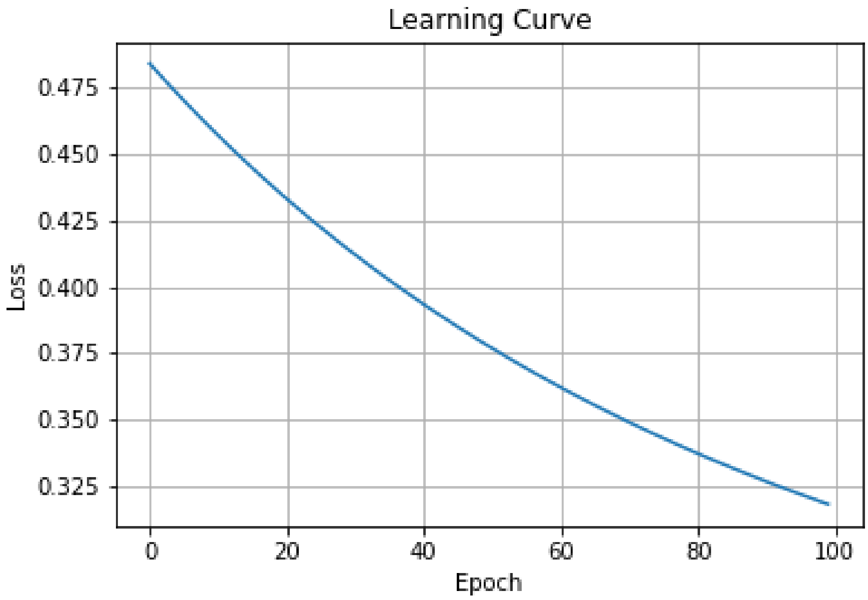

Figure 1: Loss vs Epoch

Training Loss = 0.32 Validation Loss = 0.32 Validation Accuracy = 50%

2 Q2

The number of training epochs is increased to 500.

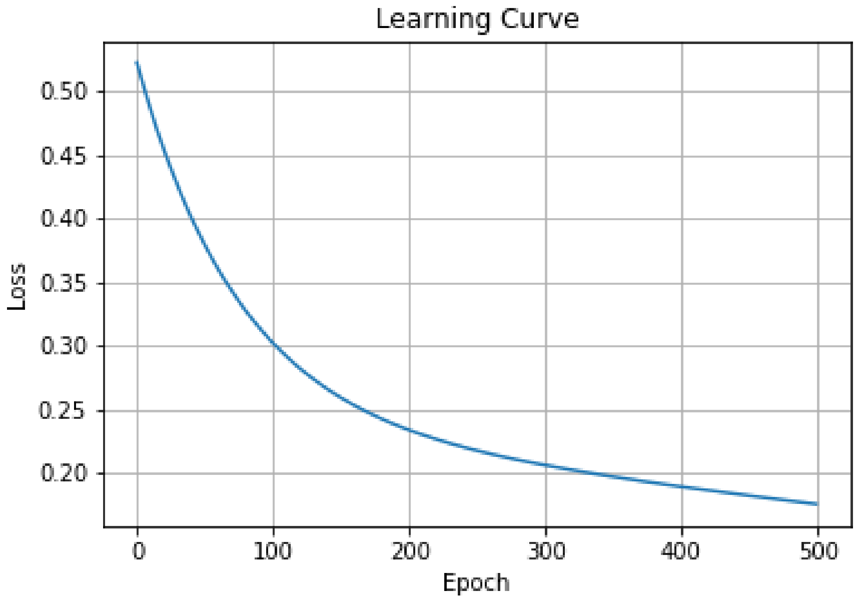

Figure 2: Loss vs Epoch

Training Loss = 0.17 Validation Loss = 0.18 Validation Accuracy = 93%

With the increased number of epochs, the network fitted the training data better and all the evaluation metrics improved, including validation accuracy.

3 Q3

Two addition fully connected layers of size $2 \times \#hiddenunitsize$ and $4 \times \#hiddenunitsize$ are added to the network.

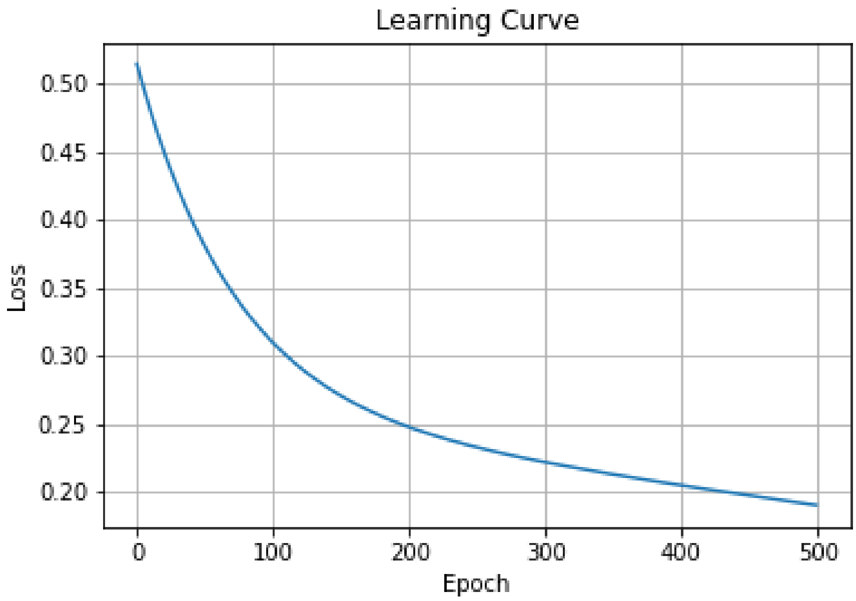

Figure 3: Loss vs Epoch

Training Loss = 0.19 Validation Loss = 0.20 Validation Accuracy = 80%

Although adding a layer to the network increases the complexity of the decision boundaries, this network had lower accuracy and higher loss than the model in Figure 2. This situation may be attributed to fixed number of training epochs. The additional layers resulted the network to have more trainable parameters, hence it requires longer time to optimize them.

4 Q4

The learning rate is increased to 0.01.

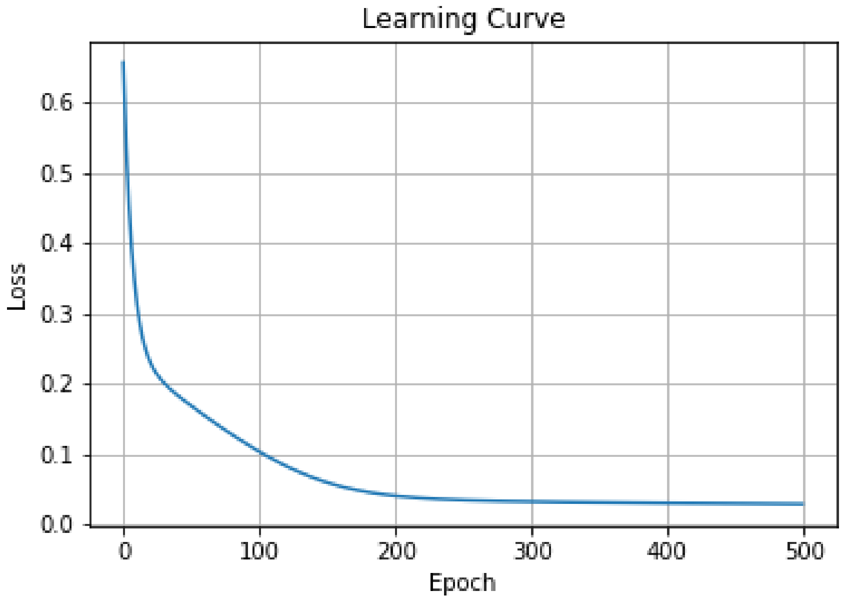

Figure 4: Loss vs Epoch

Training Loss = 0.02 Validation Loss = 0.02 Validation Accuracy = 98%

With fixed number of training epochs, the network was able to achieve lower loss and higher validation accuracy. Since the networks weights are updated with larger steps in every iteration of backpropagation. Unlike what is observed in Figure 2 and 3, the training loss stabilized at iteration 200 and it approaches at a lower number, namely 0.02.

5 Q5

A dropout layer with probability 0.5 is added to the network.

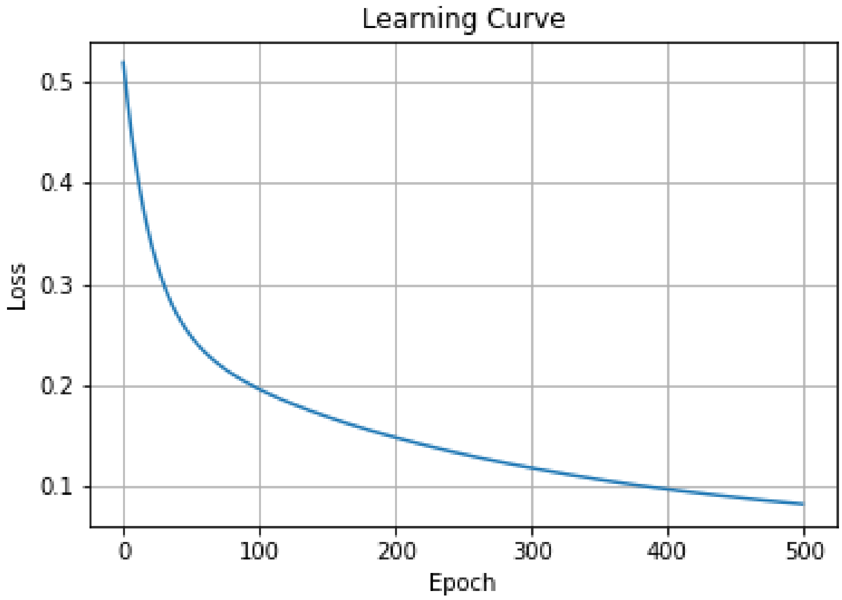

Figure 5: Loss vs Epoch

Training Loss = 0.08 Validation Loss = 0.08 Validation Accuracy = 96%

Dropout layers are utilized to prevent the network from overfitting to the training data. When compared to the Figure 4, it can be observed that the both training and validation losses are higher, and validation accuracy decreased.

6 Q6

Momentum is added to the optimizer.

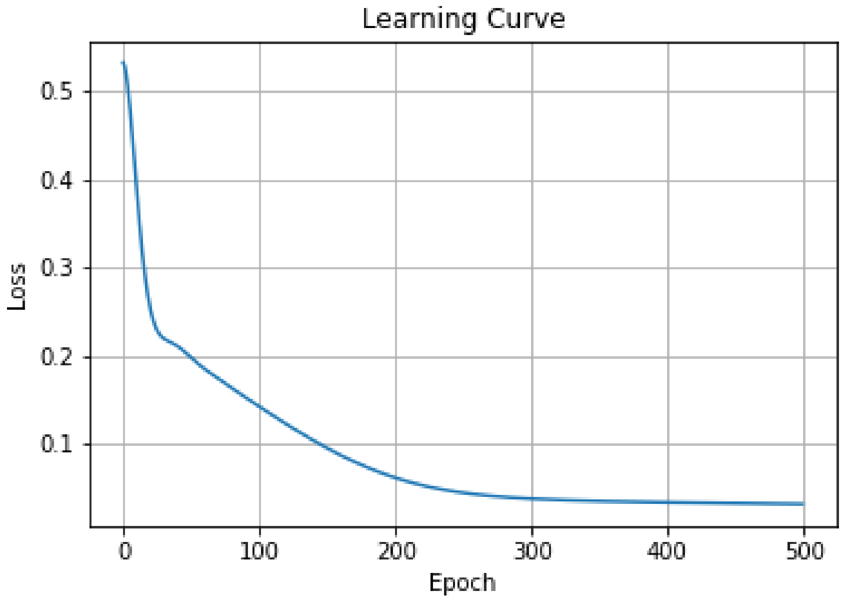

Figure 6: Loss vs Epoch

Training Loss = 0.03 Validation Loss = 0.02 Validation Accuracy = 97%

The momentum provides a weighted sum of previous update terms to the network parameters, hence it provides a more stable convergence to the optimal point. It is possible to observe that momentum resulted the network to reach a training loss of 0.1 around 150^{th} epoch, whereas it was achieved in 400^{th} epoch in Figure 5

7 Appendix

The code and results given in this section is shared @

```
[ ]: import torch
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import json
```

```
[ ]: from google.colab import drive
drive.mount('/content/drive')
```

```
[ ]: #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
```

```
[ ]: #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')
plt.savefig('/content/drive/MyDrive/583_HW6/dataset.png')
```

MODELS

```
[ ]: #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 FullyConnectedModified(torch.nn.Module):
  def __init__(self, input_size, hidden_size, num_classes):
    super(FullyConnectedModified, 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, 2*self.hidden_size)
    self.fc3 = torch.nn.Linear(2*self.hidden_size, 4*self.hidden_size)
    self.fc4 = torch.nn.Linear(4*self.hidden_size, num_classes)
    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()
  def forward(self, x):
    hidden1 = self.fc1(x)
    relu = self.relu(hidden1)
    hidden2 = self.fc2(relu)
    relu2 = self.relu(hidden2)
    hidden3 = self.fc3(relu2)
    relu3 = self.relu(hidden3)
    output = self.fc4(relu3)
    return output
class FullyConnectedQ5(torch.nn.Module):
  def __init__(self, input_size, hidden_size, num_classes):
    super(FullyConnectedQ5, 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.dropout = torch.nn.Dropout(0.5)
    self.sigmoid = torch.nn.Sigmoid()
  def forward(self, x):
    hidden = self.fc1(x)
    relu = self.relu(hidden)
    dropout = self.dropout(relu)
    output = self.fc2(relu)
    return output
```

[ ]:

```
output_path = "/content/drive/MyDrive/583_HW6"
for question in [str(i) for i in range(1,7)]:

  #CREATE MODEL
  input_size = 2
```

```python
hidden_size = 64
num_classes = 1

model = FullyConnected(input_size, hidden_size, num_classes) if question == '5' else (FullyConnectedModified(input_size, hidden_size, num_classes) if question == '3' or question == '4' else FullyConnectedModified(input_size, hidden_size, num_classes))

model.to(device)

#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.01 if question == '4' else 0.001
momentum = 0.9 if question == '6' else 0

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

model.train()
epoch = 100 if question == '1' else 500
x_train = x_train.to(device)
y_train = y_train.to(device)

loss_values = np.zeros(epoch)

# TRAIN THE MODEL
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()

training_loss = loss.item()

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

plt.savefig('/content/drive/MyDrive/583_HW6/Q'+question+'.png')
```

```
plt.close('all')

#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

val_acc = (100*correct)//(total)
print('Validation accuracy: %.2f%%' %((100*correct)//(total)))
val_acc
result_dict = {}
result_dict["loss"] = after_train.item()
result_dict["acc"] = val_acc
result_dict["trainingloss"] = training_loss
with open('/content/drive/MyDrive/583_HW6/result_dict_Q' + question + '.
↪json', 'w') as f:
    json.dump(result_dict, f)
```