

(a) (5 points) Data generation. Please refer to the illustration of two regions and corresponding labels as follows:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import torch
4 import sys

1 points = np.random.rand(100000, 2) * 10 - 5
2 dists_to_coord = np.sqrt(np.sum(points ** 2, axis=1))
3 labels = np.sqrt(np.sum((points - np.array([[0, 0]])) ** 2, axis=1)) <= 2.5

1 plt.scatter(points[labels == 1, 0], points[labels == 1, 1], c='red', s=0.1)
2 plt.scatter(points[labels == 0, 0], points[labels == 0, 1], c='blue', s=0.1)
3 plt.axis('square')
4 plt.show()
```

(b) (5 points) Network design, implementation, training, and testing. Please report training loss, validation accuracy at every epoch by a line graph and report the testing accuracy of the final model of the MLP classifier.

- Network design: The MLP should contain at least one hidden layer, with at most 50

perceptrons (excluding perceptrons for input and output layers).

- Training method: The MLP must be trained using Cross Entropy Loss, Adam optimizer with a learning rate of 0.001, mini-batch size of 128 samples, and in at most 20 epochs.

```

1 points = np.random.rand(100000, 2) * 10 - 5
2 labels = np.sqrt(np.sum((points - np.array([[0, 0]])) ** 2, axis=1)) <= 2.5
3 train_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float32),
4 train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=True)
5
6 points = np.random.rand(20000, 2) * 10 - 5
7 labels = np.sqrt(np.sum((points - np.array([[0, 0]])) ** 2, axis=1)) <= 2.5
8 val_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float32),
9 val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=128, shuffle=False)
10
11
12 points = np.random.rand(20000, 2) * 10 - 5
13 labels = np.sqrt(np.sum((points - np.array([[0, 0]])) ** 2, axis=1)) <= 2.5
14 test_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float32),
15 test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=128, shuffle=False)

1 model = torch.nn.Sequential(
2     torch.nn.Linear(2, 20),
3     torch.nn.LeakyReLU(),
4     torch.nn.Linear(20, 1),
5     torch.nn.Sigmoid(),
6 ).cuda()
7 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
8 criterion = torch.nn.BCELoss()
9 losses=[]
10 validation_accuracy = []
11 test_accuracy = []
12 for epoch in range(20):
13     plt.clf()
14     for points, labels in train_loader:
15         preds = model(points.cuda())
16         loss = criterion(preds, labels.cuda())
17         loss.backward()
18         optimizer.step()
19         optimizer.zero_grad()
20         sys.stdout.write(f'\rEpoch {epoch}. Training Loss: {loss.item()}')
21         sys.stdout.flush()
22     losses.append(loss.detach().cpu().numpy())
23
24     with torch.no_grad():
25         total_val_loss = 0.0
26         acc_avg = 0
27         for points, labels in val_loader:
28             preds = model(points.cuda())
29             total_val_loss += criterion(preds, labels.cuda())

```

```
29         total_val_loss += criterion(preds, labels.cuda())
30     preds = preds.squeeze(1).detach().cpu().numpy()
31     points = points.detach().numpy()
32
33     acc = 0
34     for i in range(preds.shape[0]):
35         if labels[i]==0 and preds[i]<0.5:
36             acc+=1
37         elif labels[i]==1 and preds[i]>=0.5:
38             acc+=1
39     acc = acc/128
40     acc_avg += acc
41     plt.scatter(points[preds >= 0.5, 0], points[preds >= 0.5, 1], c='red', s=0.1)
42     plt.scatter(points[preds < 0.5, 0], points[preds < 0.5, 1], c='blue', s=0.1)
43     validation_accuracy.append(acc_avg/len(val_loader))
44
45     avg_val_loss = total_val_loss.item() / len(val_loader)
46     print()
47     sys.stdout.write(f'\rEpoch {epoch}. Validation Loss: {avg_val_loss}')
48     sys.stdout.flush()
49     avg_test_total = []
50     with torch.no_grad():
51         total_test_loss = 0.0
52         acc_avg_test = 0
53         for points, labels in test_loader:
54             preds = model(points.cuda())
55             total_test_loss += criterion(preds, labels.cuda())
56             preds = preds.squeeze(1).detach().cpu().numpy()
57             #if points[preds>=0.5, 0]
58             #pred+=1
59
60         acc_test = 0
61         for i in range(preds.shape[0]):
62             if labels[i]==0 and preds[i]<0.5:
63                 acc_test+=1
64             elif labels[i]==1 and preds[i]>=0.5:
65                 acc_test+=1
66         acc_test = acc_test/128
67         acc_avg_test += acc_test
68
69         points = points.detach().numpy()
70         labels=labels.squeeze()
71         plt.scatter(points[preds >= 0.5, 0], points[preds >= 0.5, 1], c='red', s=0.1)
72         plt.scatter(points[preds < 0.5, 0], points[preds < 0.5, 1], c='blue', s=0.1)
73         avg_test_loss = total_test_loss.item() / len(test_loader)
74         avg_test_total.append(avg_test_loss)
75     test_accuracy.append(acc_avg_test/len(test_loader))
76
77     #sys.stdout.write(f'\rEpoch {epoch}. Testing Loss: {avg_test_loss}')
78     sys.stdout.flush()
79     plt.axis('square')
```

```
80     plt.show()
```



```
1 import matplotlib.pyplot as plt
2
3 #plt.figure(figsize=(12, 4))
4 #plt.subplot(1, 2, 1)
5 plt.plot(range(len(losses)), losses)
6 plt.title('Training Loss')
7 plt.xlabel('Epoch')
8 plt.ylabel('Loss')
9
10 plt.plot(range(len(losses)), losses)
11 plt.title('Training Loss')
12 plt.xlabel('Epoch')
13 plt.ylabel('Loss')
14
```

```
1
2 plt.subplot(1, 2, 2)
3 plt.plot(range(len(validation_accuracy)), validation_accuracy)
4 plt.title('Validation Accuracy')
5 plt.xlabel('Epoch')
6 plt.ylabel('Accuracy')
7
8 plt.tight_layout()
9 plt.show()
```

```
1 #the testing accuracy of the final model of the MLP classifier
2 print(test_accuracy[-1])
```

0.992734872611465

(a) (10 points) Data generation.

```
1 !wget https://user-images.githubusercontent.com/29124670/266203829-06705829-b2a2-472d-8:
2
3 import cv2
4 from google.colab.patches import cv2_imshow
5
6 # bear_img = cv2.imread("bear_binary.jpg")
7 # cv2_imshow(bear_img)
```

```
--2023-10-03 22:13:53-- https://user-images.githubusercontent.com/29124670/266203829
Resolving user-images.githubusercontent.com (user-images.githubusercontent.com)... 18
Connecting to user-images.githubusercontent.com (user-images.githubusercontent.com)|1
HTTP request sent, awaiting response... 200 OK
Length: 38071 (37K) [image/jpeg]
Saving to: 'bear_binary.jpg'
```

```
bear_binary.jpg      100%[=====>]  37.18K  --.-KB/s    in 0.001s
```

```
2023-10-03 22:13:53 (24.5 MB/s) - 'bear_binary.jpg' saved [38071/38071]
```

```
1 from PIL import Image
```

```
1 bear_img=Image.open("bear_binary.jpg")
```

```
1 def get_pixel_fromxy(bear_img,xy:tuple[float,float])>int:
2     w=bear_img.width-1
3     h=bear_img.height-1
4
5     p_x=w*(1/2)*(xy[0]+1)
6     p_y=h*(1/2)*(-xy[1]-1)
7
8     p=bear_img.getpixel((p_x,p_y))
9
10    return 1*(p>128)
```

```
1 points = np.random.rand(100000, 2) * 2 - 1
2 labels =np.array([get_pixel_fromxy(bear_img,xy=(i[0],i[1])) for i in points])
```

```
1 plt.scatter(points[labels == 1, 0], points[labels == 1, 1], c='white', s=0.1)
2 plt.scatter(points[labels == 0, 0], points[labels == 0, 1], c='black', s=0.1)
3 plt.axis('square')
4 plt.show()
```

```
+ plt.show()
```

(b) (10 points) Network design, implementation, training, and testing. Please report training loss, validation accuracy at every epoch by a line graph and report the testing accuracy of the final model of the MLP classifier.

- Network design: The MLP should contain at least one hidden layer, with at most 500 perceptrons (excluding perceptrons for input and output layers).
- Training method: The MLP must be trained using Cross Entropy Loss, Adam optimizer with a learning rate of 0.001, mini-batch size of 128 samples, and in at most 100 epochs.

```
1 points = np.random.rand(100000, 2) * 2 - 1
2 labels = np.array([get_pixel_fromxy(bear_img,xy=(i[0],i[1])) for i in points])
3 train_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float32),
4 train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=True)
5
6 points = np.random.rand(20000, 2) * 2 - 1
7 labels = np.array([get_pixel_fromxy(bear_img,xy=(i[0],i[1])) for i in points])
8 val_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float32),
9 val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=128, shuffle=False)
10
11
12 points = np.random.rand(20000, 2) * 2 - 1
13 labels = np.array([get_pixel_fromxy(bear_img,xy=(i[0],i[1])) for i in points])
14 test_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float32),
```



```
15 test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=128, shuffle=False)

1 model = torch.nn.Sequential(
2     torch.nn.Linear(2, 100),
3     torch.nn.LeakyReLU(),
4     torch.nn.Linear(100, 100),
5     torch.nn.LeakyReLU(),
6     torch.nn.Linear(100, 100),
7     torch.nn.LeakyReLU(),
8     torch.nn.Linear(100, 100),
9     torch.nn.LeakyReLU(),
10    torch.nn.Linear(100, 1),
11    torch.nn.Sigmoid(),
12 ).cuda()
13 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
14 criterion = torch.nn.BCELoss()
15 losses=[]
16 validation_accuracy = []
17 test_accuracy = []
18 for epoch in range(100):
19     plt.clf()
20     for points, labels in train_loader:
21         preds = model(points.cuda())
22         loss = criterion(preds, labels.cuda())
23         loss.backward()
24         optimizer.step()
25         optimizer.zero_grad()
26         sys.stdout.write(f'\rEpoch {epoch}. Training Loss: {loss.item()}')
27         sys.stdout.flush()
28     losses.append(loss.detach().cpu().numpy())
29
30     with torch.no_grad():
31         total_val_loss = 0.0
32         acc_avg = 0
33         for points, labels in val_loader:
34             preds = model(points.cuda())
35             total_val_loss += criterion(preds, labels.cuda())
36             preds = preds.squeeze(1).detach().cpu().numpy()
37             points = points.detach().numpy()
38
39             acc = 0
40             for i in range(preds.shape[0]):
41                 if labels[i]==0 and preds[i]<0.5:
42                     acc+=1
43                 elif labels[i]==1 and preds[i]>=0.5:
44                     acc+=1
45             acc = acc/128
46             acc_avg += acc
47             plt.scatter(points[preds >= 0.5, 0], points[preds >= 0.5, 1], c='white', s=
48             plt.scatter(points[preds < 0.5, 0], points[preds < 0.5, 1], c='black', s=0.1)
```

```
49     validation_accuracy.append(acc_avg/len(val_loader))
50
51     avg_val_loss = total_val_loss.item() / len(val_loader)
52     print()
53     sys.stdout.write(f'\rEpoch {epoch}. Validation Loss: {avg_val_loss}')
54     sys.stdout.flush()
55     avg_test_total = []
56     with torch.no_grad():
57         total_test_loss = 0.0
58         acc_avg_test = 0
59         for points, labels in test_loader:
60             preds = model(points.cuda())
61             total_test_loss += criterion(preds, labels.cuda())
62             preds = preds.squeeze(1).detach().cpu().numpy()
63             #if points[preds>=0.5, 0]
64             #pred+=1
65
66             acc_test = 0
67             for i in range(preds.shape[0]):
68                 if labels[i]==0 and preds[i]<0.5:
69                     acc_test+=1
70                 elif labels[i]==1 and preds[i]>=0.5:
71                     acc_test+=1
72             acc_test = acc_test/128
73             acc_avg_test += acc_test
74
75             points = points.detach().numpy()
76             labels=labels.squeeze()
77             plt.scatter(points[preds >= 0.5, 0], points[preds >= 0.5, 1], c='white', s=60)
78             plt.scatter(points[preds < 0.5, 0], points[preds < 0.5, 1], c='black', s=60)
79             avg_test_loss = total_test_loss.item() / len(test_loader)
80             avg_test_total.append(avg_test_loss)
81         test_accuracy.append(acc_avg_test/len(test_loader))
82
83         #sys.stdout.write(f'\rEpoch {epoch}. Testing Loss: {avg_test_loss}')
84         sys.stdout.flush()
85     plt.axis('square')
86     plt.show()
```

```
1 import matplotlib.pyplot as plt
2
3 #plt.figure(figsize=(12, 4))
4 #plt.subplot(1, 2, 1)
5 plt.plot(range(len(losses)), losses)
6 plt.title('Training Loss')
7 plt.xlabel('Epoch')
8 plt.ylabel('Loss')
9
```

```
10 plt.plot(range(len(losses)), losses)
11 plt.title('Training Loss')
12 plt.xlabel('Epoch')
13 plt.ylabel('Loss')
```

```
1 plt.subplot(1, 2, 2)
2 plt.plot(range(len(validation_accuracy)), validation_accuracy)
3 plt.title('Validation Accuracy')
4 plt.xlabel('Epoch')
5 plt.ylabel('Accuracy')
6
7 plt.tight_layout()
8 plt.show()
```

```
1 #the testing accuracy of the final model of the MLP classifier
2 print(test_accuracy[-1])
```

```
0.9899482484076433
```