CSCI3230 / ESTR3108 2023-24 First Term Assignment 3 I declare that the assignment here submitted is original except for source material explicitly acknowledged, and that the same or closely related material has not been previously submitted for another course. I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the following websites. University Guideline on Academic Honesty: http://www.cuhk.edu.hk/policy/academichonesty/ Faculty of Engineering Guidelines to Academic Honesty: http://www.erg.cuhk.edu.hk/erg-intra/upload/documents/ENGG\_Discipline.pdf

Student Name:Fong Long Wai

Student ID :1155177220

Q3C.

By using batch size **=** 256, learning rate **=** 0.01, training epochs **=** 50, and seed is not set,

test accuracy of 98.8% is achieved.

It is observed that the training loss can differs with the seed, and it can converge to different value in the training.

The following is the final weight.

W1:

[-1.7558],

[ 1.6814],

[ 0.2296],

[-1.7671]

W2:

[ 0.9549, 0.9903, -0.1146, -0.4299]

**import** torch

**from** torch **import** nn

**from** torch.utils.data **import** DataLoader

**import** numpy **as** np

**from** torch.utils.data **import** Dataset

**import** pandas **as** pd

**from** tqdm **import** tqdm

**from** matplotlib **import** pyplot **as** plt

In [ ]:

batch\_size **=** 256

learning\_rate **=** 0.01

epoches **=** 50

In [ ]:

**class** Data(Dataset):

**def** \_\_init\_\_(self, csv\_file, transform**=None**):

super(Data, self)**.**\_\_init\_\_()

file **=** pd**.**read\_csv(csv\_file)

self**.**input\_data **=** file[['x']]**.**values

self**.**labels **=** file['y']**.**values**.**astype(int)

**def** \_\_getitem\_\_(self, index):

data\_item **=** self**.**input\_data[index]

data\_item **=** torch**.**tensor(data\_item)**.**float()

label **=** self**.**labels[index]

**return** data\_item, label

**def** \_\_len\_\_(self):

**return** len(self**.**input\_data)

In [ ]:

train\_data **=** Data('train\_q3.csv')

test\_data **=** Data('test\_q3.csv')

train\_dataloader **=** DataLoader(train\_data, batch\_size**=**batch\_size)

test\_dataloader **=** DataLoader(test\_data, batch\_size**=**batch\_size)

In [ ]:

**class** NeuralNetwork(nn**.**Module):

**def** \_\_init\_\_(self):

super(NeuralNetwork, self)**.**\_\_init\_\_()

self**.**linear\_relu\_stack **=** nn**.**Sequential(

nn**.**Linear(1,4),

nn**.**ReLU(),

nn**.**Sigmoid(),

)

self**.**initialize\_weights()

**def** forward(self, x):

logits **=** self**.**linear\_relu\_stack(x)

**return** logits

**def** initialize\_weights(self):

**for** m **in** self**.**modules():

**if** isinstance(m, nn**.**Linear):

nn**.**init**.**kaiming\_uniform\_(m**.**weight)

**if** m**.**bias **is** **not** **None**:

nn**.**init**.**zeros\_(m**.**bias)

In [ ]:

device **=** "cuda" **if** torch**.**cuda**.**is\_available() **else** "cpu"

model **=** NeuralNetwork()**.**to(device)

In [ ]:

loss\_fn **=** nn**.**CrossEntropyLoss()

optimizer **=** torch**.**optim**.**SGD(model**.**parameters(), lr**=**learning\_rate)

In [ ]:

model**.**train()

epoches\_time**=**0

**for** epoch **in** tqdm(range(epoches)):

epoches\_time**=**epoches\_time**+**1

train\_loss, correct **=** 0, 0

**for** X, y **in** train\_dataloader:

X, y **=** X**.**to(device), y**.**to(device)

pred **=** model(X)

loss **=** loss\_fn(pred, y**.**long())

*# Backpropagation*

optimizer**.**zero\_grad()

loss**.**backward()

optimizer**.**step()

*# record loss*

train\_loss **+=** loss**.**item()

correct **+=** (pred**.**argmax(1) **==** y)**.**type(torch**.**float)**.**sum()**.**item()

size **=** len(train\_dataloader**.**dataset)

train\_loss **/=** len(train\_dataloader)

correct **/=** size

print(f" Train accuracy: {(100**\***correct):>0.1f}%, Avg loss: {train\_loss:>8f}")

plt**.**plot(epoches\_time,train\_loss,"ro")

plt**.**xlabel("Epoch")

plt**.**ylabel("Average Loss")

plt**.**show()

2%|▏ | 1/50 [00:00<00:15, 3.20it/s]

Train accuracy: 54.7%, Avg loss: 1.340353

4%|▍ | 2/50 [00:00<00:12, 3.82it/s]

Train accuracy: 63.4%, Avg loss: 1.333190

Train accuracy: 72.1%, Avg loss: 1.326437

8%|▊ | 4/50 [00:00<00:10, 4.34it/s]

Train accuracy: 81.0%, Avg loss: 1.320077

Train accuracy: 90.1%, Avg loss: 1.314076

12%|█▏ | 6/50 [00:01<00:10, 4.02it/s]

Train accuracy: 98.1%, Avg loss: 1.308418

14%|█▍ | 7/50 [00:01<00:10, 4.14it/s]

Train accuracy: 99.7%, Avg loss: 1.303068

16%|█▌ | 8/50 [00:01<00:09, 4.20it/s]

Train accuracy: 99.6%, Avg loss: 1.297999

18%|█▊ | 9/50 [00:02<00:10, 3.94it/s]

Train accuracy: 99.6%, Avg loss: 1.293199

20%|██ | 10/50 [00:02<00:10, 3.64it/s]

Train accuracy: 99.6%, Avg loss: 1.288656

22%|██▏ | 11/50 [00:02<00:11, 3.50it/s]

Train accuracy: 99.6%, Avg loss: 1.284362

24%|██▍ | 12/50 [00:03<00:10, 3.76it/s]

Train accuracy: 99.5%, Avg loss: 1.280300

Train accuracy: 99.5%, Avg loss: 1.276446

28%|██▊ | 14/50 [00:03<00:09, 3.79it/s]

Train accuracy: 99.5%, Avg loss: 1.272794

30%|███ | 15/50 [00:03<00:09, 3.76it/s]

Train accuracy: 99.5%, Avg loss: 1.269322

32%|███▏ | 16/50 [00:04<00:09, 3.70it/s]

Train accuracy: 99.5%, Avg loss: 1.266016

34%|███▍ | 17/50 [00:04<00:08, 3.71it/s]

Train accuracy: 99.5%, Avg loss: 1.262867

36%|███▌ | 18/50 [00:04<00:08, 3.74it/s]

Train accuracy: 99.4%, Avg loss: 1.259869

Train accuracy: 99.4%, Avg loss: 1.257009

40%|████ | 20/50 [00:05<00:07, 4.26it/s]

Train accuracy: 99.4%, Avg loss: 1.254279

Train accuracy: 99.4%, Avg loss: 1.251668

44%|████▍ | 22/50 [00:05<00:06, 4.54it/s]

Train accuracy: 99.4%, Avg loss: 1.249171

46%|████▌ | 23/50 [00:05<00:05, 4.50it/s]

Train accuracy: 99.4%, Avg loss: 1.246782

48%|████▊ | 24/50 [00:06<00:06, 3.96it/s]

Train accuracy: 99.4%, Avg loss: 1.244496

Train accuracy: 99.4%, Avg loss: 1.242306

52%|█████▏ | 26/50 [00:06<00:05, 4.35it/s]

Train accuracy: 99.4%, Avg loss: 1.240206

54%|█████▍ | 27/50 [00:06<00:05, 4.44it/s]

Train accuracy: 99.4%, Avg loss: 1.238191

56%|█████▌ | 28/50 [00:06<00:05, 4.16it/s]

Train accuracy: 99.4%, Avg loss: 1.236254

58%|█████▊ | 29/50 [00:07<00:04, 4.27it/s]

Train accuracy: 99.4%, Avg loss: 1.234388

Train accuracy: 99.4%, Avg loss: 1.232591

62%|██████▏ | 31/50 [00:07<00:04, 4.52it/s]

Train accuracy: 99.4%, Avg loss: 1.230861

Train accuracy: 99.4%, Avg loss: 1.229193

68%|██████▊ | 34/50 [00:08<00:03, 4.98it/s]

Train accuracy: 99.4%, Avg loss: 1.227584

Train accuracy: 99.4%, Avg loss: 1.226030

70%|███████ | 35/50 [00:08<00:03, 4.92it/s]

Train accuracy: 99.4%, Avg loss: 1.224523

Train accuracy: 99.3%, Avg loss: 1.223063

74%|███████▍ | 37/50 [00:08<00:02, 5.00it/s]

Train accuracy: 99.3%, Avg loss: 1.221647

Train accuracy: 99.3%, Avg loss: 1.220273

78%|███████▊ | 39/50 [00:09<00:02, 5.17it/s]

Train accuracy: 99.3%, Avg loss: 1.218941

Train accuracy: 99.2%, Avg loss: 1.217646

84%|████████▍ | 42/50 [00:09<00:01, 4.84it/s]

Train accuracy: 99.2%, Avg loss: 1.216385

Train accuracy: 99.2%, Avg loss: 1.215156

86%|████████▌ | 43/50 [00:10<00:01, 4.68it/s]

Train accuracy: 99.1%, Avg loss: 1.213958

Train accuracy: 99.1%, Avg loss: 1.212790

90%|█████████ | 45/50 [00:10<00:01, 4.69it/s]

Train accuracy: 99.1%, Avg loss: 1.211650

Train accuracy: 99.0%, Avg loss: 1.210538

94%|█████████▍| 47/50 [00:10<00:00, 4.90it/s]

Train accuracy: 99.0%, Avg loss: 1.209453

96%|█████████▌| 48/50 [00:11<00:00, 4.75it/s]

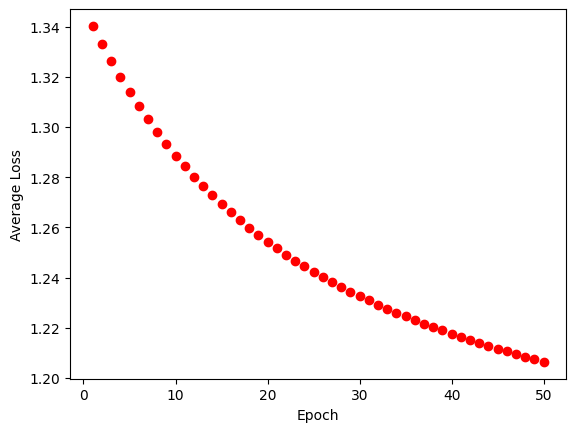
Train accuracy: 99.0%, Avg loss: 1.208394

98%|█████████▊| 49/50 [00:11<00:00, 4.69it/s]

Train accuracy: 99.0%, Avg loss: 1.207363

100%|██████████| 50/50 [00:11<00:00, 4.35it/s]

Train accuracy: 99.0%, Avg loss: 1.206356



In [ ]:

model**.**eval()

correct **=** 0

*# Turn off gradient descent*

**with** torch**.**no\_grad():

**for** X, y **in** tqdm(test\_dataloader):

X, y **=** X**.**to(device), y**.**to(device)

pred **=** model(X)

correct **+=** (pred**.**argmax(1) **==** y)**.**type(torch**.**float)**.**sum()**.**item()

size **=** len(test\_dataloader**.**dataset)

correct **=** correct **/** size

print(f" Test accuracy: {(100**\***correct):>0.1f}%")

100%|██████████| 8/8 [00:00<00:00, 159.98it/s]

Test accuracy: 98.8%

In [ ]:

**for** param **in** model**.**parameters():

print(param)

Parameter containing:

tensor([[-1.7558],

[ 1.6814],

[ 0.2296],

[-1.7671]], requires\_grad=True)

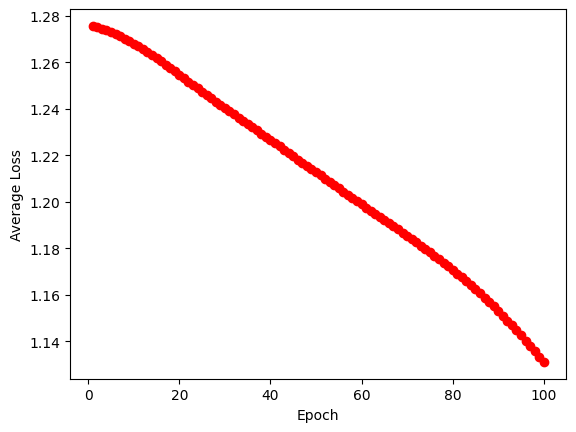
Parameter containing:

tensor([ 0.9549, 0.9903, -0.1146, -0.4299], requires\_grad=True)

Q4a.

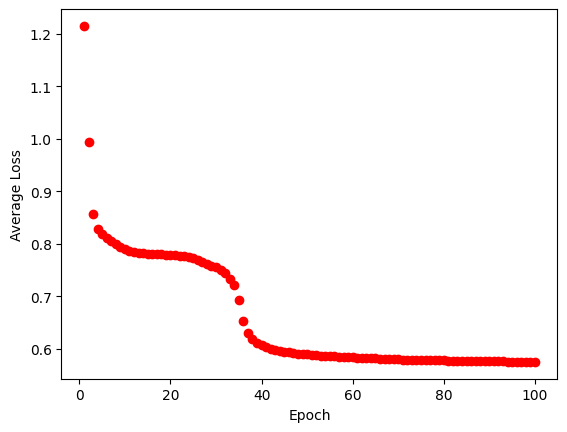
Learning rate=0.001

Accuracy=44.8



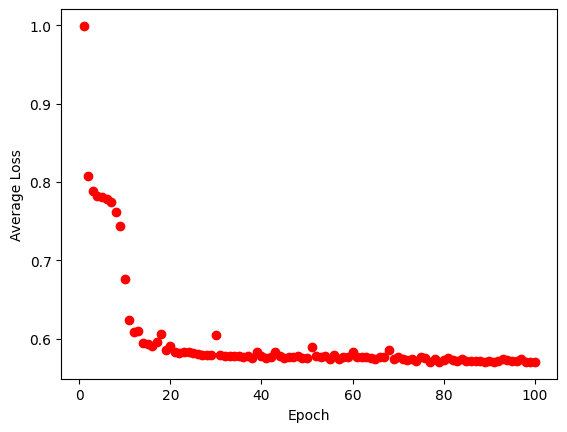
Learning rate= 0.1

Accuracy=82.9



Learning rate=0.4

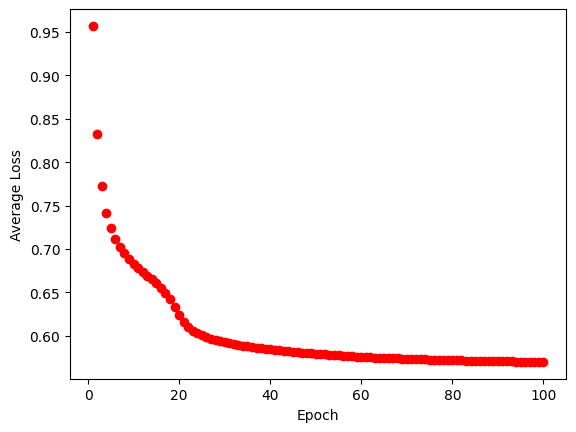
Accuracy=92.2



Q4b.

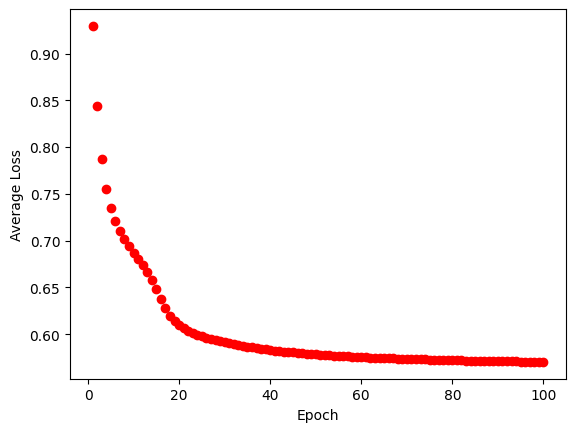
kaiming\_normal

Testing accuracy=95.0%



xavier\_normal

Testing accuracy=94.5%



xavier\_normal, but set the input tensor as m.weight\*1.1

Testing accuracy=97%

一張含有 文字, 螢幕擷取畫面, 圖表, 行 的圖片

自動產生的描述

Q4 Code

import torch

from torch import nn

from torch.utils.data import DataLoader

import numpy as np

from torch.utils.data import Dataset

import pandas as pd

from tqdm import tqdm

import torch.backends.cudnn as cudnn

import random

import numpy as np

from matplotlib import pyplot as plt

seed = 1443

cudnn.benchmark = False

cudnn.deterministic = True

random.seed(seed)

np.random.seed(seed)

torch.manual\_seed(seed)

torch.cuda.manual\_seed(seed)

# %%

batch\_size = 256

learning\_rate = 0.1

epoches = 100

# %%

class Data(Dataset):

    def \_\_init\_\_(self, csv\_file, transform=None):

        super(Data, self).\_\_init\_\_()

        file = pd.read\_csv(csv\_file)

        self.input\_data = file[['x1', 'x2']].values

        self.labels = file['y'].values.astype(int)

    def \_\_getitem\_\_(self, index):

        data\_item = self.input\_data[index]

        data\_item = torch.tensor(data\_item).float()

        label = self.labels[index]

        return data\_item, label

    def \_\_len\_\_(self):

        return len(self.input\_data)

# %%

train\_data = Data('train\_q4.csv')

test\_data = Data('test\_q4.csv')

train\_dataloader = DataLoader(train\_data, batch\_size=batch\_size)

test\_dataloader = DataLoader(test\_data, batch\_size=batch\_size)

# %%

class NeuralNetwork(nn.Module):

    def \_\_init\_\_(self):

        super(NeuralNetwork, self).\_\_init\_\_()

        self.linear\_relu\_stack = nn.Sequential(

           nn.Linear(2,4),

           nn.ReLU(),

           nn.Linear(4,4),

           nn.ReLU(),

           nn.Linear(4,3),

           nn.Softmax(),

        )

        self.initialize\_weights()

    def forward(self, x):

        logits = self.linear\_relu\_stack(x)

        return logits

    def initialize\_weights(self):

        for m in self.modules():

            if isinstance(m, nn.Linear):

                nn.init.kaiming\_normal\_(m.weight)

                if m.bias is not None:

                     nn.init.zeros\_(m.bias)

# %%

device = "cuda" if torch.cuda.is\_available() else "cpu"

model = NeuralNetwork().to(device)

# %%

loss\_fn = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model.parameters(), lr=learning\_rate)

# %%

model.train()

epoches\_time=0

for epoch in tqdm(range(epoches)):

    epoches\_time=epoches\_time+1

    train\_loss, correct = 0, 0

    for X, y in train\_dataloader:

        X, y = X.to(device), y.to(device)

        pred = model(X)

        loss = loss\_fn(pred, y.long())

        # Backpropagation

        optimizer.zero\_grad()

        loss.backward()

        optimizer.step()

        # record loss

        train\_loss += loss.item()

        correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    size = len(train\_dataloader.dataset)

    train\_loss /= len(train\_dataloader)

    correct /= size

    print(f" Train accuracy: {(100\*correct):>0.1f}%, Avg loss: {train\_loss:>8f}")

    plt.plot(epoches\_time,train\_loss,"ro")

plt.xlabel("Epoch")

plt.ylabel("Average Loss")

plt.show()

# %%

model.eval()

correct = 0

# Turn off gradient descent

with torch.no\_grad():

    for X, y in tqdm(test\_dataloader):

        X, y = X.to(device), y.to(device)

        pred = model(X)

        correct += (pred.argmax(1) == y).type(torch.float).sum().item()

size = len(test\_dataloader.dataset)

correct = correct / size

print(f" Test accuracy: {(100\*correct):>0.1f}%")