# Advanced Intelligence Systems

## Assignment 03: Part 02

## Saurabh Jawahar Kakade

## [sk2354@nau.edu](mailto:sk2354@nau.edu)

Solution 01:

* Data sets used: FASHION-MNIST and KMINST
* Three train/splits are used.

Solution 02:

* Output: Epoch = 100 (on local machine)

Text

Description automatically generated

Chart

Description automatically generated

* Batch size: 32
* Learning rate: 3 sec (avg)
* Max Epochs used: 100.
* The number of epochs that minimized the validation loss, the test error percent for every data set / model / split:

(selectedEpochs below is the number of epochs that minimized the validation loss.)

FOLD 0:

* + FeatureLess Model: Test Error: Accuracy: 10.4%, Avg loss: 0.072578
  + DATA=FashionMNIST model=fullyConnected splitNum=0 selectedEpochs=9
  + Test Error: Accuracy: 10.3%, Avg loss: 0.072655
  + DATA=FashionMNIST model=convolutional splitNum=0 selectedEpochs=9
  + Test Error: Accuracy: 9.7%, Avg loss: 0.072411

FOLD 1:

* + FeatureLess Model: Test Error: Accuracy: 12.5%, Avg loss: 0.072613
  + DATA=FashionMNIST model=fullyConnected splitNum=1 selectedEpochs=9
  + Test Error: Accuracy: 13.5%, Avg loss: 0.072382
  + DATA=FashionMNIST model=convolutional splitNum=1 selectedEpochs=7
  + Test Error: Accuracy: 13.2%, Avg loss: 0.072571

FOLD 2:

* + FeatureLess Model: Test Error: Accuracy: 9.5%, Avg loss: 0.072807
  + DATA=FashionMNIST model=fullyConnected splitNum=2 selectedEpochs=14
  + Test Error: Accuracy: 10.5%, Avg loss: 0.072718
  + DATA=FashionMNIST model=convolutional splitNum=2 selectedEpochs=8
  + Test Error: Accuracy: 10.0%, Avg loss: 0.072740

FOLD 0:

* + FeatureLess Model: Test Error: Accuracy: 7.1%, Avg loss: 0.072672
  + DATA=KMNIST model=fullyConnected splitNum=0 selectedEpochs=3
  + Test Error: Accuracy: 9.0%, Avg loss: 0.072586
  + DATA=KMNIST model=convolutional splitNum=0 selectedEpochs=5
  + Test Error: Accuracy: 11.9%, Avg loss: 0.072470

FOLD 1:

* + FeatureLess Model: Test Error: Accuracy: 7.9%, Avg loss: 0.072748
  + DATA=KMNIST model=fullyConnected splitNum=1 selectedEpochs=4
  + Test Error: Accuracy: 10.7%, Avg loss: 0.072558
  + DATA=KMNIST model=convolutional splitNum=1 selectedEpochs=4
  + Test Error: Accuracy: 8.4%, Avg loss: 0.072705

FOLD 2:

* + FeatureLess Model: Test Error: Accuracy: 11.3%, Avg loss: 0.072491
  + DATA=KMNIST model=fullyConnected splitNum=2 selectedEpochs=5
  + Test Error: Accuracy: 10.7%, Avg loss: 0.072481
  + DATA=KMNIST model=convolutional splitNum=2 selectedEpochs=4
  + Test Error: Accuracy: 9.8%, Avg loss: 0.072589
* Neural network accuracy:
  + The two neural networks such as Fully Connect Network and Convolution Network shows **better** accuracy than the baseline featureless model.
  + Fully Connected Network showed more accuracy than convolution network.
  + Fully Connected Model shows better performance on an average than rest of the two networks.
  + Other data analyzed: KMNIST
    - KMNIST found to be difficult for the neural network to learn.
    - Test error accuracy by KMINST on both the neural networks (on an average) showed little poor performance as compared to Fashion-MINST data set.
    - The accuracy for KMNIST was lesser than Fashion-MNIST data set.
* Analysis of more than one train/split:
  + The getTrainTestData(data\_name, splitNum) function in the program was called for 3 folds to split everytime.
  + For every call, the data was split in train= 6667 and test=3333
  + Here is the actual function:
    - def getTrainTestData(data\_name, splitNum):

train\_data, test\_data = torch.utils.data.random\_split(full\_data\_dict[data\_name],[6667,3333])

return train\_data, test\_data

Solution 03: Python Code

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Advanced Intelligence Systems: Assignment 03 by Saurabh Kakade (sk2354@nau.edu)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import numpy as np

import os

import torch

import torchvision

import torch.nn.functional as F

from torch.utils.tensorboard import SummaryWriter

from torch import nn, optim

from torch.utils.data import Dataset, ConcatDataset, DataLoader

from torch.utils.data.dataset import random\_split

from torchvision import datasets, utils

from torchvision.transforms import ToTensor, Lambda, Resize, Compose, transforms

import matplotlib.pyplot as plt

from sklearn.model\_selection import KFold

from collections import Counter

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

full\_data\_dict = {}

MAX\_EPOCHS = 100 #10000

NUM\_SPLITS = 3

DATA\_SETS = ["FashionMNIST","KMNIST"]

loss\_function = nn.CrossEntropyLoss()

results = {}

writer\_dict = {}

subtrainResults = {}

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def FeatureLess(train\_set):

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

pred\_model = FullyConnected().to(device)

x1 = torch.rand(1, 28, 28, device=device)

logits = pred\_model(x1)

pred\_probab = nn.Softmax(dim=1)(logits)

y\_pred = pred\_probab.argmax(1)

return y\_pred

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class FullyConnected(torch.nn.Module):

def \_\_init\_\_(self, input\_size=784, h1=300, h2=100, output\_size=10):

super().\_\_init\_\_()

self.layer\_1 = torch.nn.Linear(input\_size, h1)

self.layer\_2 = torch.nn.Linear(h1, h2)

self.layer\_3 = torch.nn.Linear(h2, output\_size)

def forward(self, x):

x = torch.flatten(x, start\_dim=1)

x = F.relu(self.layer\_1(x))

x = F.relu(self.layer\_2(x))

x = self.layer\_3(x)

return x

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class LeNet(torch.nn.Module):

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

super().\_\_init\_\_()

self.conv1 = torch.nn.Conv2d(in\_channels=1, out\_channels= 6, kernel\_size=5)

self.conv2 = torch.nn.Conv2d(6, 16, 5)

self.fc1 = torch.nn.Linear(4\*4\*16, 120)

self.fc2 = torch.nn.Linear(120, 84)

self.output = torch.nn.Linear(84, 10)

def forward(self, x):

x = F.relu(self.conv1(x))

# use x.shape to check the current size

# print (x.shape)

x = F.max\_pool2d(x, 2, 2)

x = F.relu(self.conv2(x))

x = F.max\_pool2d(x, 2, 2)

x = x.view(-1, 4\*4\*16)

x = F.relu(self.fc1(x))

x = F.relu(self.fc2(x))

x = self.output(x)

return x

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def reset\_weights(m):

'''

Try resetting model weights to avoid

weight leakage.

'''

for layer in m.children():

if hasattr(layer, 'reset\_parameters'):

# print(f'Reset trainable parameters of layer = {layer}')

layer.reset\_parameters()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def newModel(model):

if model == "fullyConnected":

return FullyConnected()

if model == "convolutional":

return LeNet()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

for data\_name in DATA\_SETS: # For Extra Credit add another data set.

data\_loader = getattr(datasets, data\_name)

full\_data\_dict[data\_name] = data\_loader(

root="data",

train=False,

download=True,

transform=ToTensor()

)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def getTrainTestData(data\_name, splitNum):

train\_data, test\_data = torch.utils.data.random\_split(full\_data\_dict[data\_name],[6667,3333])

return train\_data, test\_data

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def getMostFrequentLabel(train\_set):

return FeatureLess(train\_set)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def splitData(train\_set):

transform\_data = transforms.Compose([transforms.ToTensor(), transforms.Resize((28,28))])

train\_dataset, val\_dataset = torch.utils.data.random\_split(train\_set,[5557,1110])

train\_loader = torch.utils.data.DataLoader(train\_dataset, batch\_size=32, shuffle=True)

validation\_loader = torch.utils.data.DataLoader(val\_dataset, batch\_size=32, shuffle=True)

return train\_loader, validation\_loader

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def learn(subtrain\_net, subtrain\_set, MAX\_EPOCHS, validation):

torch.nn.CrossEntropyLoss()

lenet\_model = LeNet()

fully\_connected\_model = FullyConnected()

for mm in 'LeNet', 'fully\_connected\_model':

for ss in "subtrain", 'validation':

writer\_dict[mm + '\_' + ss] = SummaryWriter("runs/" + ss + '\_' + 'with' + '\_' + mm)

d\_loader = {'subtrain': subtrain\_set, 'validation': validation}

neural\_networks = {'LeNet':lenet\_model, 'fully\_connected\_model':fully\_connected\_model}

for pattern in "LeNet", "fully\_connected\_model":

optimizer = optim.Adam(neural\_networks[pattern].parameters(),lr = 0.001)

#Make an instance of the loss function

criterion = torch.nn.CrossEntropyLoss()

#use a for loop over epochs of learning

for epoch in range(MAX\_EPOCHS):

loss\_dict = {"epoch":epoch, "pattern":subtrain\_net}

for s in "subtrain", "validation":

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if s == "subtrain":

train\_loss = 0

#writing a for loop over batches using your DataLoader (for x,y in data\_loader).

for batch\_index, (data,target) in enumerate(d\_loader[s]):

pred = neural\_networks[pattern](data) #predictions

loss = criterion(pred, target) #loss

optimizer.zero\_grad() #zero the gradient

loss.backward() #compute gradient

optimizer.step() #Update the neural network weights using gradient

train\_loss += loss.item()

#Compute loss

train\_loss /= len(d\_loader[s])

loss = train\_loss

writer\_dict[pattern + "\_" + s].add\_scalar('Train/Loss',loss,epoch)

# print("Subtrain: {}".format(pred))

# print(max(pred))

else:

train\_loss = 0

for data,target in d\_loader[s]:

pred = neural\_networks[pattern](data)

train\_loss += criterion(pred,target).item()

#Compute loss

train\_loss /= len(d\_loader[s])

loss = train\_loss

subtrainResults[epoch] = loss

writer\_dict[pattern + "\_" + s].add\_scalar('Val/Loss',loss,epoch)

loss\_dict[s] = loss

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#print these values on the screen and log these loss values to the tensorboard writer

# print("pattern=%(pattern)s epoch=%(epoch)d subtrain=%(subtrain)f validation=%(validation)f" % loss\_dict)

return subtrainResults

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def getBestEpochs(subtrain\_result):

min\_loss = min(subtrain\_result.values())

for keyValue in subtrain\_result:

if subtrain\_result[keyValue] == min\_loss:

min\_losskey = keyValue

# print(min\_losskey)

return min\_losskey

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def PredictOnSet(train\_net, test\_set, model):

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

if model == "fullyConnected":

pred\_model = FullyConnected().to(device)

x1 = torch.rand(1, 28, 28, device=device)

logits = pred\_model(x1)

pred\_probab = nn.Softmax(dim=1)(logits)

y\_pred = pred\_probab.argmax(1)

return y\_pred

else:

pred\_model = LeNet().to(device)

x1 = torch.rand(1, 1, 28, 28, device=device)

logits = pred\_model(x1)

pred\_probab = nn.Softmax(dim=1)(logits)

y\_pred = pred\_probab.argmax(1)

return y\_pred

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def PercentError(test\_predictions, test\_set):

network = FullyConnected()

network.apply(reset\_weights)

criterion = torch.nn.CrossEntropyLoss()

test\_loader = torch.utils.data.DataLoader(test\_set, batch\_size=32, shuffle=True)

size = len(test\_loader.dataset)

test\_loss, correct = 0, 0

with torch.no\_grad():

for X, y in test\_loader:

pred = network(X)

test\_loss += criterion(pred, y).item()

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

test\_loss /= size

correct /= size

print(f"Test Error: Accuracy: {(100\*correct):>0.1f}%, Avg loss: {test\_loss:>8f}")

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def PercentError1(test\_predictions, test\_set):

# a = PredictOnSet(FullyConnected(), test\_set)

# print(f"percent error {a}")

# print(f"test prediction: {test\_predictions}")

network = FullyConnected()

network.apply(reset\_weights)

optimizer = torch.optim.Adam(network.parameters(), lr=1e-4)

test\_loader = torch.utils.data.DataLoader(test\_set, batch\_size=32, shuffle=True)

# Evaluationfor this fold

correct, total = 0, 0

# for folds in range(MAX\_EPOCHS):

with torch.no\_grad():

# Iterate over the test data and generate predictions

for i, data in enumerate(test\_loader, 0):

# Get inputs

inputs, targets = data

# Generate outputs

outputs = network(inputs)

# Set total and correct

# \_, predicted = torch.max(outputs.data, 1)

total += targets.size(0)

correct += (test\_predictions == targets).sum().item()

# Print accuracy

# print('Accuracy for fold %d: %d %%' % (splitNum,100.0 \* correct / total))

# print('--------------------------------')

results[splitNum] = 100.0 \* (correct / total)

# Print fold results

print(f'K-FOLD CROSS VALIDATION RESULTS FOR {splitNum} FOLDS')

print('--------------------------------')

sum = 0.0

for key, value in results.items():

print(f'Fold {key}: {value} %')

sum += value

print(f'Average: {sum/len(results.items())} %')

return None

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def TestErrorOneSplit(data\_name, splitNum, model):

train\_set, test\_set = getTrainTestData(data\_name, splitNum)

if model == "featureless":

test\_predictions = getMostFrequentLabel(train\_set)

else:

subtrain\_set, validation\_set = splitData(train\_set)

subtrain\_net = newModel(model)

subtrain\_result = learn(subtrain\_net, subtrain\_set, MAX\_EPOCHS, validation=validation\_set)

selected\_epochs = getBestEpochs(subtrain\_result)

print("DATA=%s model=%s splitNum=%d selectedEpochs=%d" % (data\_name, model, splitNum, selected\_epochs))

train\_net = newModel(model)

learn(train\_net, train\_set, selected\_epochs,validation=validation\_set)

test\_predictions = PredictOnSet(train\_net, test\_set, model)

return PercentError(test\_predictions, test\_set)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

for data\_name in DATA\_SETS:

for splitNum in range(NUM\_SPLITS):

for model in "featureless", "fullyConnected", "convolutional":

TestErrorOneSplit(data\_name, splitNum, model)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*