Advanced Intelligence Systems

Assignment 03: Part 02

Saurabh Jawahar Kakade
sk2354@nau.edu

Solution 01:

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

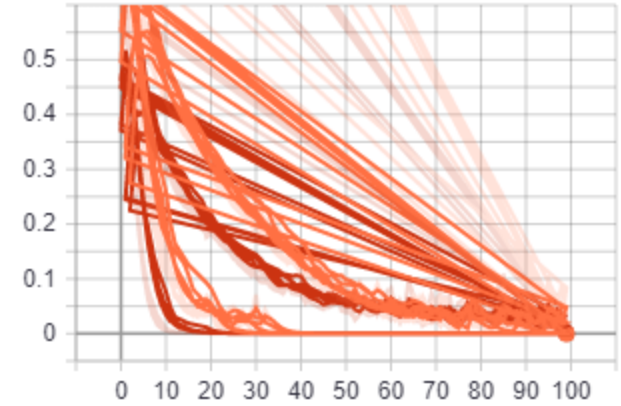
Solution 02:

- Output: Epoch = 100 (on local machine)

```
E:\Spring 2021\Subjects\AI\Assignments\Assignment 03\PART 2>python AI_A03_PART2_sk2354.py && tensorboard --logdir=runs
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
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
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
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
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
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
TensorFlow installation not found - running with reduced feature set.
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.2.1 at http://localhost:6006/ (Press CTRL+C to quit)
```

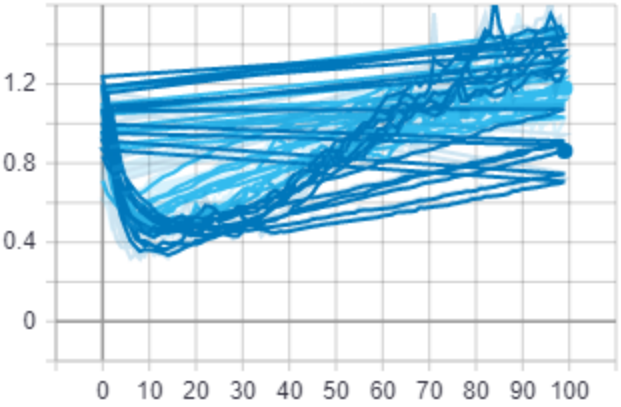
Train

Loss
tag: Train/Loss



Val

Loss
tag: Val/Loss



- 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)

#**********************************************************************
```