Q6 Batch Norm and SELU (4.5 Points)

Generate training and test datasets for a binary classification problem using Fashion-MNIST with class 1 being a combination of sneaker and pullover and class 0 being the combination of sandal and shirt categories.

- Train the model using Logistic regression (No Hidden Layers). Report train and test loss.
- Train a Neural Network with one hidden layer (100 neurons). Use Adam optimizer and Relu activation for hidden layer. First overfit a small sample to check errors and get idea of learning rate. Then train on complete dataset. Add regularization (dropout or weight decay)if needed.
- Now add another hidden layer (50 Neurons). Adjust the learning rate if you have to. Add regularization (dropout or weight decay) if needed.
- Now try adding Batch Normalization and compare the train and test loss : Is it converging faster than before? Does it produce a better model? How does it affect training speed? **Do not use dropout with batch normalization.**
- Try replacing Batch Normalization with SELU, and make the necessary adjustments to ensure the network self-normalizes (i.e., standardize the input features, use LeCun normal initialization, make sure the DNN contains only a sequence of dense layers). Compare the results with Batch Normalization. **For SELU if you are using dropout then use alpha dropout.** Alpha dropout make sure that network is self normalized.

Importing Libraries

```
if 'google.colab' in str(get_ipython()):
  print('Running on Colab')
else:
  print('Not running on Colab')
```

```
Running on Colab
```

```
if 'google.colab' in str(get_ipython()):
  !pip install wandb --upgrade
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whee
Collecting wandb
  Downloading wandb-0.13.4-py2.py3-none-any.whl (1.9 MB)
     |████████████████████████████████| 1.9 MB 19.2 MB/s
Collecting sentry-sdk>=1.0.0
  Downloading sentry_sdk-1.10.1-py2.py3-none-any.whl (166 kB)
     |████████████████████████████████| 166 kB 74.2 MB/s
Requirement already satisfied: protobuf!=4.0.*,!=4.21.0,<5,>=3.12.0 in /usr/local
Requirement already satisfied: six>=1.13.0 in /usr/local/lib/python3.7/dist-packa
Collecting pathtools
```

```
  Downloading pathtools-0.1.2.tar.gz (11 kB)
Requirement already satisfied: Click!=8.0.0,>=7.0 in /usr/local/lib/python3.7/dis
Collecting GitPython>=1.0.0
  Downloading GitPython-3.1.29-py3-none-any.whl (182 kB)
     |████████████████████████████████| 182 kB 60.8 MB/s
Collecting shortuuid>=0.5.0
  Downloading shortuuid-1.0.9-py3-none-any.whl (9.4 kB)
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-packages (
Collecting setproctitle
  Downloading setproctitle-1.3.2-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_6
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: requests<3,>=2.0.0 in /usr/local/lib/python3.7/dis
Collecting docker-pycreds>=0.4.0
  Downloading docker_pycreds-0.4.0-py2.py3-none-any.whl (9.0 kB)
Requirement already satisfied: promise<3,>=2.0 in /usr/local/lib/python3.7/dist-p
Collecting gitdb<5,>=4.0.1
  Downloading gitdb-4.0.9-py3-none-any.whl (63 kB)
     |████████████████████████████████| 63 kB 2.3 MB/s
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/pytho
Collecting smmap<6,>=3.0.1
  Downloading smmap-5.0.0-py3-none-any.whl (24 kB)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dis
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/lo
Collecting sentry-sdk>=1.0.0
  Downloading sentry_sdk-1.10.0-py2.py3-none-any.whl (166 kB)
     |████████████████████████████████| 166 kB 65.7 MB/s
  Downloading sentry_sdk-1.9.10-py2.py3-none-any.whl (162 kB)
     |████████████████████████████████| 162 kB 28.5 MB/s
  Downloading sentry_sdk-1.9.9-py2.py3-none-any.whl (162 kB)
     |████████████████████████████████| 162 kB 81.1 MB/s
  Downloading sentry_sdk-1.9.8-py2.py3-none-any.whl (158 kB)
     |████████████████████████████████| 158 kB 79.9 MB/s
  Downloading sentry_sdk-1.9.7-py2.py3-none-any.whl (157 kB)
     |████████████████████████████████| 157 kB 77.2 MB/s
  Downloading sentry_sdk-1.9.6-py2.py3-none-any.whl (157 kB)
     |████████████████████████████████| 157 kB 70.7 MB/s
  Downloading sentry_sdk-1.9.5-py2.py3-none-any.whl (157 kB)
     |████████████████████████████████| 157 kB 63.5 MB/s
  Downloading sentry_sdk-1.9.4-py2.py3-none-any.whl (157 kB)
     |████████████████████████████████| 157 kB 73.7 MB/s
  Downloading sentry_sdk-1.9.3-py2.py3-none-any.whl (157 kB)
     |████████████████████████████████| 157 kB 64.2 MB/s
  Downloading sentry_sdk-1.9.2-py2.py3-none-any.whl (157 kB)
     |████████████████████████████████| 157 kB 64.8 MB/s
```

```python
# mount google drive
if 'google.colab' in str(get_ipython()):
    from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
# Importing the necessary libraries
import torch
```

```
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import torch.nn.functional as F

from torch.optim.lr_scheduler import ReduceLROnPlateau, ExponentialLR, CyclicLR, OneCycleL

import numpy as np
import random

from datetime import datetime
from pathlib import Path
import sys
from types import SimpleNamespace

import wandb
```

Initializing Wandb

```
wandb.login()
```

```
ERROR:wandb.jupyter:Failed to detect the name of this notebook, you can set it manual
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
True
```

```
wandb.init(name = "Hw5_FASHION_MNIST.ipynb", project = 'Deep_Learning_Class_UTD')
```

```
wandb: Currently logged in as: pranavshekhar2. Use `wandb login --relogin` to force r
```

Tracking run with wandb version 0.13.4
Run data is saved locally in `/content/wandb/run-20221028_214025-23f1zng3`
Syncing run **Hw5_FASHION_MNIST.ipynb** to Weights & Biases (docs)

Q6 Batch Norm and SELU (4.5 Points)

Generate training and test datasets for a binary classification problem using Fashion-MNIST with class 1 being a combination of sneaker and pullover and class 0 being the combination of sandal and shirt categories.

- Train the model using Logistic regression (No Hidden Layers). Report train and test loss.
- Train a Neural Network with one hidden layer (100 neurons). Use Adam optimizer and Relu activation for hidden layer. First overfit a small sample to check errors and get idea of learning rate. Then train on complete dataset. Add regularization (dropout or weight decay)if needed.

- Now add another hidden layer (50 Neurons). Adjust the learning rate if you have to. Add regularization (dropout or weight decay) if needed.
- Now try adding Batch Normalization and compare the train and test loss : Is it converging faster than before? Does it produce a better model? How does it affect training speed? **Do not use dropout with batch normalization.**
- Try replacing Batch Normalization with SELU, and make the necessary adjustments to ensure the network self-normalizes (i.e., standardize the input features, use LeCun normal initialization, make sure the DNN contains only a sequence of dense layers). Compare the results with Batch Normalization. **For SELU if you are using dropout then use alpha dropout**. Alpha dropout make sure that network is self normalized

```
# This is the path where we will downlaod and save data
if 'google.colab' in str(get_ipython()):
    data_folder = Path('/content/drive/MyDrive/Deep_Learning_UTD/Dataset')
    model_folder = Path('/content/drive/MyDrive/Deep_Learning_UTD/Model')
else:
    data_folder = Path('/home/harpreet/Insync/google_drive_shaannoor/data/datasets')
```

Train and Test Dataset - FASHION_MNIST

```
# Transform to convert images to pytorch tensors
trans1 = transforms.ToTensor()

# Transform to normalize the data
# The mean and std are based on train subset which we will create below
trans2 = transforms.Normalize((0.2857,), (0.3528))
trans = transforms.Compose([trans1, trans2])

# Download the training_validation data (we will create two subsets - trainset and valset
train_val_set = torchvision.datasets.FashionMNIST(root = data_folder,
                                            train = True,
                                            transform = trans,
                                            download = True)

# Download the testing data
testset = torchvision.datasets.FashionMNIST(root = data_folder,
                                           train = False,
                                           transform = trans,
                                           download = True)
```

```
def split_dataset(base_dataset, fraction, seed):
    split_a_size = int(fraction * len(base_dataset))
    split_b_size = len(base_dataset) - split_a_size
    return torch.utils.data.random_split(base_dataset, [split_a_size, split_b_size], gener
    )
```

```
trainset, validset = split_dataset(train_val_set,0.8,10)
```

```
#DataLoaders

# Initializing the batch size
batch_size = 256

# Creating data loader for train set
train_loader = torch.utils.data.DataLoader(dataset= trainset,
                                           batch_size = batch_size,
                                           shuffle = True)

valid_loader = torch.utils.data.DataLoader(dataset = validset,
                                           batch_size = batch_size,
                                           shuffle = False)

# Creating data loader for test set
test_loader = torch.utils.data.DataLoader(dataset = testset,
                                          batch_size = batch_size,
                                          shuffle = False)
```

```
#Checking the mapping of the labels

train_val_set.data
train_val_set.class_to_idx
```

```
{'T-shirt/top': 0,
 'Trouser': 1,
 'Pullover': 2,
 'Dress': 3,
 'Coat': 4,
 'Sandal': 5,
 'Shirt': 6,
 'Sneaker': 7,
 'Bag': 8,
 'Ankle boot': 9}
```

```
#Creating a subset of data consisting of class 1 and class 0

from torch.utils.data.dataset import Subset

#Sneaker and Pullover - Class1
train_idx_1 = np.where((train_val_set.targets==7) | (train_val_set.targets==2))[0]
train_subset_1 = Subset(train_val_set,train_idx_1)


#Sandal and Shirt - Class 0
train_idx_2 = np.where((train_val_set.targets==5) | (train_val_set.targets==6))[0]
train_subset_2 = Subset(train_val_set,train_idx_2)
```

```
#Sneaker and Pullover - Class 1 - Creating labels for Class 1 Group

train_subset_1.dataset.data[train_subset_1.indices].shape
```

```
labels_subset_1 = [1]*(train_subset_1.dataset.data[train_subset_1.indices].shape[0]) #Labe
```

```
train_val_set.targets[train_subset_1.indices]
```

```
tensor([2, 7, 2,  ..., 2, 7, 2])
```

```
#Sandal and Shirt - Class 0 - Creating labels for Class 0 Group

train_subset_2.dataset.data[train_subset_2.indices].shape
labels_subset_2 = [0]*(train_subset_2.dataset.data[train_subset_2.indices].shape[0])
```

```
train_val_set.targets[train_subset_2.indices]
```

```
tensor([5, 5, 5,  ..., 6, 5, 5])
```

```
train_data1 = []
for i in range(len(train_subset_1)):
   train_data1.append([train_subset_1[i], labels_subset_1[i]])

train_data2 = []
for j in range(len(train_subset_2)):
   train_data2.append([train_subset_2[j], labels_subset_2[j]])

#Combined Dataset
train_data1.extend(train_data2)
```

Class for Model

```
class FashionMNIST_NN(nn.Module):
  def __init__(self, input_dim, output_dim, h_sizes, dprob, non_linearity, batch_norm):
    super().__init__()
    self.input_dim = input_dim
    self.h_sizes = h_sizes # list of hidden sizes
    self.non_linearity = non_linearity
    self.batch_norm = batch_norm
    self.dprob = dprob # list of dropout probabilities
    self.output_dim = output_dim

    # Initialize hidden layers
    model_layers = [nn.Flatten()]

    # hidden layers
    for i, hidden_size in enumerate(self.h_sizes):
      model_layers.append(nn.Linear(input_dim, hidden_size))
      model_layers.append(self.non_linearity)
      model_layers.append(nn.Dropout(p=dprob[i]))

      if self.batch_norm:
        model_layers.append(nn.BatchNorm1d(hidden_size, momentum=0.9))
```

```
        input_dim = hidden_size

      # output layer
      model_layers.append(nn.Linear(self.h_sizes[-1], self.output_dim))

      self.module_list = nn.ModuleList(model_layers)


    def forward(self, x):
      for layer in self.module_list:
        x = layer(x)

      # we are not using softmax function in the forward passs
      # nn.crossentropy loss (which we will use to define our loss) combines  nn.LogSoftmax(
      return x


  def train(train_loader, loss_function, model, optimizer, grad_clipping, max_norm, log_batc

    # Training Loop

    # initilalize variables as global
    # these counts will be updated every epoch
    global batch_ct_train

    # Initialize train_loss at the he start of the epoch
    running_train_loss = 0
    running_train_correct = 0

    # put the model in training mode

    model.train()
    # Iterate on batches from the dataset using train_loader
    for input_, targets in train_loader:

      # move inputs and outputs to GPUs
      input_ = input_.to(device)
      targets = targets.to(device)


      # Step 1: Forward Pass: Compute model's predictions
      output = model(input_)

      # Step 2: Compute loss
      loss = loss_function(output, targets)

      # Correct prediction
      y_pred = torch.argmax(output, dim = 1)
      correct = torch.sum(y_pred == targets)

      batch_ct_train += 1

      # Step 3: Backward pass -Compute the gradients
      optimizer.zero_grad()
```

```
      loss.backward()

      # Gradient Clipping
      if grad_clipping:
        nn.utils.clip_grad_norm_(model.parameters(), max_norm=max_norm, norm_type=2)

      # Step 4: Update the parameters
      optimizer.step()

      # Add train loss of a batch
      running_train_loss += loss.item()

      # Add Corect counts of a batch
      running_train_correct += correct

      # log batch loss and accuracy
      if log_batch:
        if ((batch_ct_train + 1) % log_interval) == 0:
          wandb.log({f"Train Batch Loss  :": loss})
          wandb.log({f"Train Batch Acc :": correct/len(targets)})


    # Calculate mean train loss for the whole dataset for a particular epoch
    train_loss = running_train_loss/len(train_loader)

    # Calculate accuracy for the whole dataset for a particular epoch
    train_acc = running_train_correct/len(train_loader.dataset)


    return train_loss, train_acc

  def validate(valid_loader, loss_function, model, log_batch, log_interval):

    # initilalize variables as global
    # these counts will be updated every epoch
    global batch_ct_valid

    # Validation/Test loop
    # Initialize valid_loss at the he strat of the epoch
    running_val_loss = 0
    running_val_correct = 0

    # put the model in evaluation mode
    model.eval()

    with torch.no_grad():
      for input_,targets in valid_loader:

        # move inputs and outputs to GPUs
        input_ = input_.to(device)
        targets = targets.to(device)

        # Step 1: Forward Pass: Compute model's predictions
        output = model(input_)
```

```
      # Step 2: Compute loss
      loss = loss_function(output, targets)

      # Correct Predictions
      y_pred = torch.argmax(output, dim = 1)
      correct = torch.sum(y_pred == targets)

      batch_ct_valid += 1

      # Add val loss of a batch
      running_val_loss += loss.item()

      # Add correct count for each batch
      running_val_correct += correct

      # log batch loss and accuracy
      if log_batch:
        if ((batch_ct_valid + 1) % log_interval) == 0:
          wandb.log({f"Valid Batch Loss  :": loss})
          wandb.log({f"Valid Batch Accuracy :": correct/len(targets)})

    # Calculate mean val loss for the whole dataset for a particular epoch
    val_loss = running_val_loss/len(valid_loader)

    # Calculate accuracy for the whole dataset for a particular epoch
    val_acc = running_val_correct/len(valid_loader.dataset)

    # scheduler step
    # scheduler.step(valid_loss)
    # scheduler.step()

  return val_loss, val_acc

def train_loop(train_loader, valid_loader, model, optimizer, loss_function, epochs, device
               file_model):

  """
  Function for training the model and plotting the graph for train & validation loss vs ep
  Input: iterator for train dataset, initial weights and bias, epochs, learning rate, batc
  Output: final weights, bias and train loss and validation loss for each epoch.
  """

  # Create lists to store train and val loss at each epoch
  train_loss_history = []
  valid_loss_history = []
  train_acc_history = []
  valid_acc_history = []

  # initialize variables for early stopping

  delta = 0
  best_score = None
  valid_loss_min = np.Inf
```

```
    counter_early_stop=0
    early_stop=False

    # Iterate for the given number of epochs
    # Step 5: Repeat steps 1 - 4

    for epoch in range(epochs):

      t0 = datetime.now()

      # Get train loss and accuracy for one epoch
      train_loss, train_acc = train(train_loader, loss_function, model, optimizer,
                                    wandb.config.grad_clipping, wandb.config.max_norm,
                                    wandb.config.log_batch, wandb.config.log_interval)
      valid_loss, valid_acc   = validate(valid_loader, loss_function, model, wandb.config.lo

      dt = datetime.now() - t0

      # Save history of the Losses and accuracy
      train_loss_history.append(train_loss)
      train_acc_history.append(train_acc)

      valid_loss_history.append(valid_loss)
      valid_acc_history.append(valid_acc)

      # Log the train and valid loss to wandb
      wandb.log({f"Train Loss :": train_loss, "epoch": epoch})
      wandb.log({f"Train Acc :": train_acc, "epoch": epoch})

      wandb.log({f"Valid Loss :": valid_loss, "epoch": epoch})
      wandb.log({f"Valid Acc :": valid_acc, "epoch": epoch})

      if early_stopping:
        score = -valid_loss
        if best_score is None:
          best_score=score
          print(f'Validation loss has decreased ({valid_loss_min:.6f} --> {valid_loss:.6f}).
          torch.save(model.state_dict(), file_model)
          valid_loss_min = valid_loss

        elif score < best_score + delta:
          counter_early_stop += 1
          print(f'Early stoping counter: {counter_early_stop} out of {patience}')
          if counter_early_stop > patience:
            early_stop = True


        else:
          best_score = score
          print(f'Validation loss has decreased ({valid_loss_min:.6f} --> {valid_loss:.6f}).
          torch.save(model.state_dict(), file_model)
          counter_early_stop=0
          valid_loss_min = valid_loss

        if early_stop:
```

```
                print('Early Stopping')
                break

        else:

            score = -valid_loss
            if best_score is None:
                best_score=score
                print(f'Validation loss has decreased ({valid_loss_min:.6f} --> {valid_loss:.6f}).
                torch.save(model.state_dict(), file_model)
                valid_loss_min = valid_loss

            elif score < best_score + delta:
                print(f'Validation loss has not decreased ({valid_loss_min:.6f} --> {valid_loss:.6

            else:
                best_score = score
                print(f'Validation loss has decreased ({valid_loss_min:.6f} --> {valid_loss:.6f}).
                torch.save(model.state_dict(), file_model)
                valid_loss_min = valid_loss

        # Print the train loss and accuracy for given number of epochs, batch size and number
        print(f'Epoch : {epoch+1} / {epochs}')
        print(f'Time to complete {epoch+1} is {dt}')
        # print(f'Learning rate: {scheduler._last_lr[0]}')
        print(f'Train Loss: {train_loss : .4f} | Train Accuracy: {train_acc * 100 : .4f}%')
        print(f'Valid Loss: {valid_loss : .4f} | Valid Accuracy: {valid_acc * 100 : .4f}%')
        print()
        torch.cuda.empty_cache()

    return train_loss_history, train_acc_history, valid_loss_history, valid_acc_history

def get_acc_pred(data_loader, model, device):

    """
    Function to get predictions and accuracy for a given data using estimated model
    Input: Data iterator, Final estimated weoights, bias
    Output: Prections and Accuracy for given dataset
    """

    # Array to store predicted labels
    predictions = torch.Tensor() # empty tensor
    predictions = predictions.to(device) # move predictions to GPU

    # Array to store actual labels
    y = torch.Tensor() # empty tensor
    y = y.to(device)

    # put the model in evaluation mode
    model.eval()

    # Iterate over batches from data iterator
    with torch.no_grad():
        for input_, targets in data_loader:
```

```
        # move inputs and outputs to GPUs

        input_ = input_.to(device)
        targets = targets.to(device)

        # Calculated the predicted labels
        output = model(input_)

        # Choose the label with maximum probability
        prediction = torch.argmax(output, dim = 1)

        # Add the predicted labels to the array
        predictions = torch.cat((predictions, prediction))

        # Add the actual labels to the array
        y = torch.cat((y, targets))

    # Check for complete dataset if actual and predicted labels are same or not
    # Calculate accuracy
    acc = (predictions == y).float().mean()

    # Return tuple containing predictions and accuracy
    return predictions, acc
```

Logistic Regression

```
class LogisticRegression(torch.nn.Module):
     def __init__(self, input_dim, output_dim):
         super(LogisticRegression, self).__init__()
         self.linear = torch.nn.Linear(input_dim, output_dim)
     def forward(self, x):
         outputs = torch.sigmoid(self.linear(x))
         return outputs
```

```
epochs = 200
input_dim = 3*32*32 # Two inputs x1 and x2
output_dim = 1 # Single binary output
learning_rate = 0.01
```

```
model = LogisticRegression(input_dim,output_dim)
```

```
criterion = torch.nn.BCELoss()
```

```
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

One Hidden Layer

```
hyperparameters1 = SimpleNamespace(
    epochs = 20,
    input_dim = 28*28,
    output_dim = 10,
    h_sizes = [100]  , #  1 Hidden Layer of 100 Neurons
    dprob = [0],
    non_linearity = nn.ReLU(),
    batch_norm = False,
    batch_size=25,
    learning_rate=0.07,
    dataset="FASHION_MNIST",
    architecture="MLP",
    log_interval = 1,
    log_batch = True,
    file_model = model_folder/'exp1_overfit_mnist.pt',
    grad_clipping = False, # DO NOT CHANGE hyperparameters below this
    early_stopping = False,
    max_norm = 1,
    momentum = 0,
    patience = 3,
    # scheduler_factor = 0.5,
    # scheduler_patience = 0,
    weight_decay = 0.00
    )


wandb.config = hyperparameters1
wandb.config
```

```
namespace(architecture='MLP', batch_norm=False, batch_size=25,
dataset='FASHION_MNIST', dprob=[0], early_stopping=False, epochs=20,
file_model=PosixPath('/content/drive/MyDrive/Deep_Learning_UTD/Model/exp1_overfit_mn
 grad_clipping=False, h_sizes=[100], input_dim=784, learning_rate=0.07,
log_batch=True, log_interval=1, max_norm=1, momentum=0, non_linearity=ReLU(),
output_dim=10, patience=3, weight_decay=0.0)
```

```
# Fix seed value
SEED = 2344
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True

# Data Loader
train_loader = torch.utils.data.DataLoader(trainset, batch_size=wandb.config.batch_size, s
valid_loader = torch.utils.data.DataLoader(validset, batch_size=wandb.config.batch_size, s
# test_loader = torch.utils.data.DataLoader(testset, batch_size=wandb.config.batch_size,

# device
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

```
wandb.config.device = device

model = FashionMNIST_NN(wandb.config.input_dim, wandb.config.output_dim, wandb.config.h_si
                          wandb.config.dprob, wandb.config.non_linearity, wandb.config.bat

# Initialize weights from normal distribution with mean 0 and standard deviation 0.01
def init_weights(layer):
  if type(layer) == nn.Linear:
    torch.nn.init.kaiming_normal_(layer.weight, mean = 0, std = 0.01)
    # torch.nn.init.normal_(layer.weight, mean = 0, std = 0.001)
    torch.nn.init.zeros_(layer.bias)

model.to(wandb.config.device)
# model.apply(init_weights)

# loss_function
loss_function = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(),
                            lr = wandb.config.learning_rate,
                            weight_decay = wandb.config.weight_decay)

# scheduler = ReduceLROnPlateau(optimizer, mode='min', factor= wandb.config.scheduler_fact
#                               patience=wandb.config.scheduler_patience, verbose=True)

#scheduler = StepLR(optimizer, gamma=0.4,step_size=1, verbose=True)


# See live graphs in the notebook.
#%%wandb
batch_ct_train, batch_ct_valid = 0, 0
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_loop(
```

```
Validation loss has decreased (inf --> 2.400696). Saving Model...
Epoch : 1 / 20
Time to complete 1 is 0:00:13.682159
Train Loss:  2.6776 | Train Accuracy:  23.7604%
Valid Loss:  2.4007 | Valid Accuracy:  11.1750%

Validation loss has decreased (2.400696 --> 2.399744). Saving model...
Epoch : 2 / 20
Time to complete 2 is 0:00:13.766537
Train Loss:  2.8962 | Train Accuracy:  12.6042%
Valid Loss:  2.3997 | Valid Accuracy:  11.3500%

Validation loss has decreased (2.399744 --> 2.296860). Saving model...
```

```
Epoch : 3 / 20
Time to complete 3 is 0:00:13.968270
Train Loss:  2.6764 | Train Accuracy:  12.2667%
Valid Loss:  2.2969 | Valid Accuracy:  10.8417%

Validation loss has decreased (2.296860 --> 2.292094). Saving model...
Epoch : 4 / 20
Time to complete 4 is 0:00:13.827049
Train Loss:  2.3503 | Train Accuracy:  10.7208%
Valid Loss:  2.2921 | Valid Accuracy:  11.1250%

Validation loss has not decreased (2.292094 --> 2.376915). Not Saving Model...
Epoch : 5 / 20
Time to complete 5 is 0:00:16.397947
Train Loss:  2.3709 | Train Accuracy:  10.6208%
Valid Loss:  2.3769 | Valid Accuracy:  11.3500%

Validation loss has not decreased (2.292094 --> 2.317251). Not Saving Model...
Epoch : 6 / 20
Time to complete 6 is 0:00:14.738176
Train Loss:  2.3280 | Train Accuracy:  11.2979%
Valid Loss:  2.3173 | Valid Accuracy:  10.5917%

Validation loss has decreased (2.292094 --> 2.282408). Saving model...
Epoch : 7 / 20
Time to complete 7 is 0:00:14.257567
Train Loss:  2.3039 | Train Accuracy:  11.6854%
Valid Loss:  2.2824 | Valid Accuracy:  11.3500%

Validation loss has not decreased (2.282408 --> 2.301172). Not Saving Model...
Epoch : 8 / 20
Time to complete 8 is 0:00:14.579827
Train Loss:  2.2998 | Train Accuracy:  10.9000%
Valid Loss:  2.3012 | Valid Accuracy:  10.6000%

Validation loss has not decreased (2.282408 --> 2.311997). Not Saving Model...
Epoch : 9 / 20
Time to complete 9 is 0:00:14.404734
Train Loss:  2.2988 | Train Accuracy:  10.3146%
Valid Loss:  2.3120 | Valid Accuracy:  10.7417%

Validation loss has not decreased (2.282408 --> 2.321152). Not Saving Model...
Epoch : 10 / 20
Time to complete 10 is 0:00:15.625823
Train Loss:  2.2987 | Train Accuracy:  10.6187%
```

```
hyperparameters2 = SimpleNamespace(
    epochs = 20,
    input_dim = 28*28,
    output_dim = 10,
    h_sizes = [100 ,50]  , #  2 Hidden Layer of 100,50 Neurons
    dprob = [0]*2,
    non_linearity = nn.ReLU(),
    batch_norm = False,
    batch_size=25,
    learning_rate=0.07,
    dataset="FASHION_MNIST",
    architecture="MLP",
```

```
        log_interval = 1,
        log_batch = True,
        file_model = model_folder/'exp1_overfit_mnist.pt',
        grad_clipping = False, # DO NOT CHANGE hyperparameters below this
        early_stopping = False,
        max_norm = 1,
        momentum = 0,
        patience = 3,
        # scheduler_factor = 0.5,
        # scheduler_patience = 0,
        weight_decay = 0.00
        )

    wandb.config = hyperparameters2
    wandb.config


    # Fix seed value
    SEED = 2344
    random.seed(SEED)
    np.random.seed(SEED)
    torch.manual_seed(SEED)
    torch.cuda.manual_seed(SEED)
    torch.backends.cudnn.deterministic = True

    # Data Loader
    train_loader = torch.utils.data.DataLoader(trainset, batch_size=wandb.config.batch_size, s
    valid_loader = torch.utils.data.DataLoader(validset, batch_size=wandb.config.batch_size, s
    # test_loader = torch.utils.data.DataLoader(testset, batch_size=wandb.config.batch_size,

    # device
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

    wandb.config.device = device

    model2 = FashionMNIST_NN(wandb.config.input_dim, wandb.config.output_dim, wandb.config.h_s
                             wandb.config.dprob, wandb.config.non_linearity, wandb.config.bat

    # Initialize weights from normal distribution with mean 0 and standard deviation 0.01
    def init_weights(layer):
      if type(layer) == nn.Linear:
        torch.nn.init.kaiming_normal_(layer.weight, mean = 0, std = 0.01)
        # torch.nn.init.normal_(layer.weight, mean = 0, std = 0.001)
        torch.nn.init.zeros_(layer.bias)

    model2.to(wandb.config.device)
    # model.apply(init_weights)

    # loss_function
    loss_function = nn.CrossEntropyLoss()

    optimizer = torch.optim.Adam(model2.parameters(),
                                 lr = wandb.config.learning_rate,
                                 weight_decay = wandb.config.weight_decay)
```

```
# scheduler = ReduceLROnPlateau(optimizer, mode='min', factor= wandb.config.scheduler_fact
#                               patience=wandb.config.scheduler_patience, verbose=True)

#scheduler = StepLR(optimizer, gamma=0.4,step_size=1, verbose=True)


# See live graphs in the notebook.
#%%wandb
batch_ct_train, batch_ct_valid = 0, 0
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_loop(
```


✓ 4m 49s completed at 4:52 PM