```
import torchvision
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from sklearn.model_selection import train_test_split
import numpy as np
import json
from datetime import datetime
BATCH_SIZE=50
epoch_size = 15


device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
# customize Transform variable is to take input and return a tensor object
# Also by setting normalizer, I scaled pixel values between -1 and 1
#directly taken from
https://medium.com/@aayush1/fashion-mnist-data-training-using-pytorch-7f6ad71e96f4
# cusTransform =
transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5,),(0.5,),)])
cusTransform = transforms.ToTensor()
# training set
train_data = torchvision.datasets.FashionMNIST('./data', train = True, download = False,
transform = cusTransform)
#Splitting data
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split
.html
train_set, val_set = train_test_split(train_data, test_size=0.1, random_state=42)
# test set
test_data = torchvision.datasets.FashionMNIST('./data', train = False,
transform = cusTransform)

val_generator = torch.utils.data.DataLoader(val_set, batch_size = BATCH_SIZE, shuffle =
False)


# example mlp1 classifier
class mlp1(torch.nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(mlp1, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Linear(input_size, hidden_size)
        self.fc2 = torch.nn.Linear(hidden_size, num_classes)
        self.relu = torch.nn.ReLU()
    def forward(self, x):
        x = x.view(-1, self.input_size)
        hidden = self.fc1(x)
        relu = self.relu(hidden)
        output = self.fc2(relu)
        return output
class mlp1s(torch.nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(mlp1s, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Linear(input_size, hidden_size)
        self.fc2 = torch.nn.Linear(hidden_size, num_classes)
        self.sigmoid = torch.nn.Sigmoid()
    def forward(self, x):
        x = x.view(-1, self.input_size)
        hidden = self.fc1(x)
        sigmoid1 = self.sigmoid(hidden)
        output = self.fc2(sigmoid1)
        return output
#--------------------------------------------------------------------------------
# example mlp2 classifier
class mlp2(torch.nn.Module):
    def __init__(self, input_size, hidden_size, hidden_size2, num_classes):
        super(mlp2, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Linear(input_size, hidden_size)
        self.fc2 = torch.nn.Linear(hidden_size, hidden_size2,bias=False)
        self.fc3 = torch.nn.Linear(hidden_size2, num_classes)
        self.relu = torch.nn.ReLU()
    def forward(self, x):
        x = x.view(-1, self.input_size)
```

```
        hidden = self.fc1(x)
        relu = self.relu(hidden)
        hidden2 = self.fc2(relu)
        output = self.fc3(hidden2)
        return output
# example mlp2 classifier
class mlp2s(torch.nn.Module):
    def __init__(self, input_size, hidden_size, hidden_size2, num_classes):
        super(mlp2s, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Linear(input_size, hidden_size)
        self.fc2 = torch.nn.Linear(hidden_size, hidden_size2,bias=False)
        self.fc3 = torch.nn.Linear(hidden_size2, num_classes)
        self.sigmoid = torch.nn.Sigmoid()
    def forward(self, x):
        x = x.view(-1, self.input_size)
        hidden = self.fc1(x)
        sigmoid1 = self.sigmoid(hidden)
        hidden2 = self.fc2(sigmoid1)
        output = self.fc3(hidden2)
        return output
#-----------------------------------------------------------------------------
class cnn_3(torch.nn.Module):
    #Layer Definition

#https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-netw
ork-cnn/
    def __init__(self,input_size,num_classes):
        super(cnn_3, self).__init__()
        self.input_size = input_size
        #in_channel = input_size mı 0 mı
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        self.fc1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3,
stride=1, padding='valid')
        self.relu1 = torch.nn.ReLU()
        self.fc2 = torch.nn.Conv2d(in_channels=16, out_channels=8, kernel_size=7,
stride=1, padding='valid')
        self.relu = torch.nn.ReLU()
        self.maxpool1 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride=2, padding=0)
        self.fc3 = torch.nn.Conv2d(in_channels=8, out_channels=16, kernel_size=5,
stride=1, padding='valid')
        self.maxpool2 = torch.nn.MaxPool2d(kernel_size=(2,2), stride=2,padding=0)
        self.fc4 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #It didin't work ??????
        hidden1 = self.fc1(x)
        relu1 = self.relu(hidden1)
        hidden2 = self.fc2(relu1)
        relu2 = self.relu(hidden2)
        pool1 = self.maxpool1(relu2)
        hidden3 = self.fc3(pool1)
        pool2 = self.maxpool2(hidden3)
        pool2=pool2.view(50,144)
        #Reshaping linear input
        output = self.fc4(pool2)
        return output
class cnn_3s(torch.nn.Module):
    #Layer Definition

#https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-netw
ork-cnn/
    def __init__(self,input_size,num_classes):
        super(cnn_3s, self).__init__()
        self.input_size = input_size
        #in_channel = input_size mı 0 mı
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        self.fc1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3,
stride=1, padding='valid')
        self.sigmoid1 = torch.nn.Sigmoid()
        self.fc2 = torch.nn.Conv2d(in_channels=16, out_channels=8, kernel_size=7,
stride=1, padding='valid')
        self.sigmoid2 = torch.nn.Sigmoid()
        self.maxpool1 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride=2)
        self.fc3 = torch.nn.Conv2d(in_channels=8, out_channels=16, kernel_size=5,
```

```
stride=1, padding='valid')
        self.maxpool2 = torch.nn.MaxPool2d(kernel_size=(2,2), stride=2)
        self.fc4 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #It didin't work ??????
        hidden1 = self.fc1(x)
        sigmoid1 = self.sigmoid1(hidden1)
        hidden2 = self.fc2(sigmoid1)
        sigmoid2 = self.sigmoid2(hidden2)
        pool1 = self.maxpool1(sigmoid2)
        hidden3 = self.fc3(pool1)
        pool2 = self.maxpool2(hidden3)
        pool2=pool2.view(50,144)
        output = self.fc4(pool2)
        return output
#-------------------------------------------------------------------------------------------
---------------------------
class cnn_4(torch.nn.Module):
    #Layer Definition

#https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-netw
ork-cnn/
    def __init__(self,input_size,num_classes):
        super(cnn_4, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3,
stride=1, padding='valid')
        self.relu1 = torch.nn.ReLU()
        self.fc2 = torch.nn.Conv2d(in_channels=16, out_channels=8, kernel_size=5,
stride=1, padding='valid')
        self.relu2 = torch.nn.ReLU()
        self.fc3 = torch.nn.Conv2d(in_channels=8, out_channels=8, kernel_size=3,
stride=1, padding='valid')
        self.relu3 = torch.nn.ReLU()
        self.maxpool1 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2)
        self.fc4 = torch.nn.Conv2d(in_channels=8, out_channels=16, kernel_size=5,
stride=1, padding='valid')
        self.relu4 = torch.nn.ReLU()
        self.maxpool2 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2)
        self.fc5 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #x = x.view(-1, self.input_size)
        #It didin't work ??????
        hidden1 = self.fc1(x)
        relu1 = self.relu1(hidden1)
        hidden2 = self.fc2(relu1)
        relu2 = self.relu2(hidden2)
        hidden3 = self.fc3(relu2)
        relu3 = self.relu3(hidden3)
        pool1 = self.maxpool1(relu3)
        hidden4 = self.fc4(pool1)
        relu4 = self.relu4(hidden4)
        pool2 = self.maxpool2(relu4)
        pool2=pool2.view(50,144)
        #Reshaping linear input
        output = self.fc5(pool2)
        return output
class cnn_4s(torch.nn.Module):
    #Layer Definition

#https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-netw
ork-cnn/
    def __init__(self,input_size,num_classes):
        super(cnn_4s, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3,
stride=1, padding='valid')
        self.sigmoid1 = torch.nn.Sigmoid()
        self.fc2 = torch.nn.Conv2d(in_channels=16, out_channels=8, kernel_size=5,
stride=1, padding='valid')
        self.sigmoid2 = torch.nn.Sigmoid()
        self.fc3 = torch.nn.Conv2d(in_channels=8, out_channels=8, kernel_size=3,
stride=1, padding='valid')
        self.sigmoid3 = torch.nn.Sigmoid()
```

```
        self.maxpool1 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2)
        self.fc4 = torch.nn.Conv2d(in_channels=8, out_channels=16, kernel_size=5,
stride=1, padding='valid')
        self.sigmoid4 = torch.nn.Sigmoid()
        self.maxpool2 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2)
        self.fc5 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #x = x.view(-1, self.input_size)
        #It didin't work ??????
        hidden1 = self.fc1(x)
        sigmoid1 = self.sigmoid1(hidden1)
        hidden2 = self.fc2(sigmoid1)
        sigmoid2 = self.sigmoid2(hidden2)
        hidden3 = self.fc3(sigmoid2)
        sigmoid3 = self.sigmoid3(hidden3)
        pool1 = self.maxpool1(sigmoid3)
        hidden4 = self.fc4(pool1)
        sigmoid4 = self.sigmoid4(hidden4)
        pool2 = self.maxpool2(sigmoid4)
        pool2=pool2.view(50,144)
        #Reshaping linear input
        output = self.fc5(pool2)
        return output
#-------------------------------------------------------------------------------------------
-----------------------
class cnn_5(torch.nn.Module):
    #Layer Definition

#https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-netw
ork-cnn/
    def __init__(self,input_size,num_classes):
        super(cnn_5, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3,
stride=1, padding='valid')
        self.relu1 = torch.nn.ReLU()
        self.fc2 = torch.nn.Conv2d(in_channels=16, out_channels=8, kernel_size=3,
stride=1, padding='valid')
        self.relu2 = torch.nn.ReLU()
        self.fc3 = torch.nn.Conv2d(in_channels=8, out_channels=8, kernel_size=3,
stride=1, padding='valid')
        self.relu3 = torch.nn.ReLU()
        self.fc4 = torch.nn.Conv2d(in_channels=8, out_channels=8, kernel_size=3,
stride=1, padding='valid')
        self.relu4 = torch.nn.ReLU()
        self.maxpool4 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2)
        self.fc5 = torch.nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3,
stride=1, padding='valid')
        self.relu5 = torch.nn.ReLU()
        self.fc6 = torch.nn.Conv2d(in_channels=16, out_channels=16, kernel_size=3,
stride=1, padding='valid')
        self.relu6 = torch.nn.ReLU()
        self.maxpool6 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2)
        self.fc7 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #x = x.view(-1, self.input_size)
        #It didin't work ??????
        hidden1 = self.fc1(x)
        relu1 = self.relu1(hidden1)
        hidden2 = self.fc2(relu1)
        relu2 = self.relu2(hidden2)
        hidden3 = self.fc3(relu2)
        relu3 = self.relu3(hidden3)
        hidden4 = self.fc4(relu3)
        relu4 = self.relu4(hidden4)
        pool4 = self.maxpool4(relu4)
        #pool4=pool4.view()
        hidden5 = self.fc5(pool4)
        relu5 = self.relu5(hidden5)
        hidden6 = self.fc6(relu5)
        relu6 = self.relu6(hidden6)
        pool6 = self.maxpool6(relu6)
        #Reshaping linear input
        pool6=pool6.view(50,144)
```

```
        output = self.fc7(pool6)
        return output
class cnn_5s(torch.nn.Module):
    #Layer Definition

#https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-network-cnn/
    def __init__(self,input_size,num_classes):
        super(cnn_5s, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding='valid')
        self.sigmoid1 = torch.nn.Sigmoid()
        self.fc2 = torch.nn.Conv2d(in_channels=16, out_channels=8, kernel_size=3, stride=1, padding='valid')
        self.sigmoid2 = torch.nn.Sigmoid()
        self.fc3 = torch.nn.Conv2d(in_channels=8, out_channels=8, kernel_size=3, stride=1, padding='valid')
        self.sigmoid3 = torch.nn.Sigmoid()
        self.fc4 = torch.nn.Conv2d(in_channels=8, out_channels=8, kernel_size=3, stride=1, padding='valid')
        self.sigmoid4 = torch.nn.Sigmoid()
        self.maxpool4 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2,padding=0)
        self.fc5 = torch.nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3, stride=1, padding='valid')
        self.sigmoid5 = torch.nn.Sigmoid()
        self.fc6 = torch.nn.Conv2d(in_channels=16, out_channels=16, kernel_size=3, stride=1, padding='valid')
        self.sigmoid6 = torch.nn.Sigmoid()
        self.maxpool6 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2,padding=0)
        self.fc7 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #x = x.view(-1, self.input_size)
        #It didin't work ??????
        hidden1 = self.fc1(x)
        sigmoid1 = self.sigmoid1(hidden1)
        hidden2 = self.fc2(sigmoid1)
        sigmoid2 = self.sigmoid2(hidden2)
        hidden3 = self.fc3(sigmoid2)
        sigmoid3 = self.sigmoid3(hidden3)
        hidden4 = self.fc4(sigmoid3)
        sigmoid4 = self.sigmoid4(hidden4)
        pool4 = self.maxpool4(sigmoid4)
        #pool4=pool4.view()
        hidden5 = self.fc5(pool4)
        sigmoid5 = self.sigmoid5(hidden5)
        hidden6 = self.fc6(sigmoid5)
        sigmoid6 = self.sigmoid6(hidden6)
        pool6 = self.maxpool6(sigmoid6)
        #Reshaping linear input
        pool6=pool6.view(50,144)

        output = self.fc7(pool6)
        return output
#Model Types
models=['relu_mlp_1','sigmoid_mlp_1','relu_mlp_2','sigmoid_mlp_2','relu_cnn_3','sigmoid_cnn_3','relu_cnn_4','sigmoid_cnn_4','relu_cnn_5','sigmoid_cnn_5']

relu_loss_curve=[]
relu_grad_curve=[]
sigmoid_loss_curve=[]
sigmoid_grad_curve=[]
for modelselected in models:
    now = datetime.now()
    current_time = now.strftime("%H:%M:%S")

    print(f"Training is started for model {modelselected}")
    print("At time =", current_time)
    if modelselected[0]=='r':
        relu_loss_curve=[]
        relu_grad_curve=[]
        sigmoid_loss_curve=[]
        sigmoid_grad_curve=[]
```

```
    if modelselected == 'relu_mlp_1':
        model_mlp = mlp1(784,64,10).to(device)
    elif modelselected == 'sigmoid_mlp_1':
        model_mlp = mlp1s(784,64,10).to(device)
    elif modelselected == 'relu_mlp_2':
        model_mlp = mlp2(784,16,64,10).to(device)
    elif modelselected == 'sigmoid_mlp_2':
        model_mlp = mlp2s(784,16,64,10).to(device)
    elif modelselected == 'relu_cnn_3':
        model_mlp = cnn_3(784,10).to(device)
    elif modelselected == 'sigmoid_cnn_3':
        model_mlp = cnn_3s(784,10).to(device)
    elif modelselected == 'relu_cnn_4':
        model_mlp = cnn_4(784,10).to(device)
    elif modelselected == 'sigmoid_cnn_4':
        model_mlp = cnn_4s(784,10).to(device)
    elif modelselected == 'relu_cnn_5':
        model_mlp = cnn_5(784,10).to(device)
    elif modelselected == 'sigmoid_cnn_5':
        model_mlp = cnn_5s(784,10).to(device)

    criterion = nn.CrossEntropyLoss()

    optimizer = torch.optim.SGD(model_mlp.parameters(), lr = 0.01, momentum=0.0)
    #Recorded values for each try
    for epoch in range(epoch_size):
        print(f"Epoch is {epoch+1}/{epoch_size}")
        total=0
        correct=0
        train_generator = torch.utils.data.DataLoader(train_set, batch_size = BATCH_SIZE,
shuffle = True)
        total_step = len(train_generator)
        #https://stackoverflow.com/questions/62833157/cnn-model-using-pytorch
        #Train DATA

#https://androidkt.com/calculate-total-loss-and-accuracy-at-every-epoch-and-plot-using-matp
lotlib-in-pytorch/
        for i, (images, labels) in enumerate(train_generator):
            model_mlp.train()
            # Move tensors to the configured device
            images = images.to(device)
            labels = labels.to(device)
            model_mlp.to('cpu')
            weightBefore=model_mlp.fc1.weight.data.numpy().flatten()
            model_mlp.to(device)
            # Forward pass
            outputs = model_mlp(images)
            loss = criterion(outputs, labels.to(device))

            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            running_loss=loss.item()


            if (i+1) % 10 == 0:
                model_mlp.to('cpu')
                weightAfter=model_mlp.fc1.weight.data.numpy().flatten()
                model_mlp.to(device)
                running_grad=float(np.linalg.norm(weightAfter - weightBefore)/0.01)

                #https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html

                if(modelselected[0]=='r'):#That means it is RELU
                    relu_loss_curve.append(running_loss)
                    relu_grad_curve.append(running_grad)
                elif(modelselected[0]=='s'):#That means it is sigmoid
                    sigmoid_loss_curve.append(running_loss)
                    sigmoid_grad_curve.append(running_grad)
                #print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Grad {:.4f}'
                #       .format(epoch+1, epoch_size, i+1, total_step,
running_loss,running_grad))
```

```
    if modelselected[0]=='s':
        dictionary={
            'name': modelselected[-5:] ,#mlp_1 mlp_2 cnn_1 etc.
            'relu_loss_curve': relu_loss_curve,#the training loss curve of the ANN with
ReLU
            'sigmoid_loss_curve':sigmoid_loss_curve, #the training loss curve of the ANN
with logistic sigmoid
            'relu_grad_curve': relu_grad_curve,#the curve of the magnitude of the loss
gradient of the ANN with ReLU
            'sigmoid_grad_curve': sigmoid_grad_curve, #the curve of the magnitude of the
loss gradient of the ANN with ReLU
        }
    #print(train_losses_total)
        with open("FAKEq3_"+modelselected[-5:]+".json","w") as outfile:
            json.dump(dictionary,outfile)
```