EE449

Homework 1 - Training Artificial Neural Network

Due: 23:55, 24/04/2022

Berkay İPEK

2304814

Sec1

1. Basic Concepts

1.1 ANNs are actually parametric functions which can be used to approximate other functions. What function does an ANNs classifier trained with cross-entropy loss approximates? How is the loss defined to approximate that function? Bonus: Why?

In our experiment, this function should take input size of 784 and output size will be 10. Therefore, this function should be form of 784 -> 10. On the other hand, cross-entropy loss will be defined as follows (Reference: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html):

$$= -\sum_{c=1}^{M}(y_{o,c} * \log(p_{o,c}))$$

In this equation, M is the number of classes (10), p is the predicted probably observation, o is of a class c (any clothes), y is a binary indicator that shows if label c is correct classification (0 or 1).

Bonus: Why?

Let's say predicted answer is totally wrong, i.e., we give a t-shirt image then NN said that probability of being t-shirt is 0.0. Then p is going to be zero. Therefore, Loss value become infinity.

Let's say predicted answer is totally correct, i.e., we give a t-shirt image then NN said that probability of being t-shirt is 1.0. Then p is going to be 1. Therefore, Loss value become zero.

As value increases, its contribution to loss function will become larger. (Logarithmic increase) Therefore, as distance between correct and accurate probabilities increases, loss value will be bigger.

1.2 Gradient Computation

From our lecture slides, we can say that (from equation 6.3.2)

$W_{k+1} = W_k - \gamma * (\nabla L (W = @W_K))$

Then,

$\nabla L (W = @W_K) = (W_k - W_{k+1}) / \gamma$

1.3 Some Training Parameters and Basic Parameter Calculations

1.3.1 What are batch and epoch in the context of MLP training?

Batch size is a term used in machine learning and refers to the number of training examples utilized in one iteration. (Directly taken from radiopaedia.org)

An epoch is a term used in machine learning and indicates the number of passes of the entire training dataset the machine learning algorithm has completed. (Directly taken from radiopaedia.org)

1.3.2 Given that the dataset has N samples, what is the number of batches per epoch if the batch size is B?

= N/B

1.3.3 Given that the dataset has N samples, what is the number of SGD iterations if you want to train your ANN for E epochs with the batch size of B?

= (number of batches per epoch) * (epoch)

= (N/B) * E

1.4 Computing Number of Parameters of ANN Classifiers

1.4.1 Consider an MLP classifier of K hidden units where the size of each hidden unit is Hk for k=1, . . ., K. Derive a formula to compute the number of parameters that the MLP has if the input and output dimensions are Din and Dout, respectively.

Consecutive layers will have parameter number of product of their sizes. Then, we can say that

Number of parameters = (Input Layer weights – 1st Hidden Layer) + (1st Hidden Layer - 2nd Hidden Layer) + (2nd Hidden Layer - 3rd Hidden Layer) + … + (Kth Hidden Layer- Output Layer Weights)

Input Layer Weights - 1st Hidden Layer = (D_{in} * H_1) + H_1 (H_1 comes from bias)

1st Hidden Layer - 2nd Hidden Layer = (H_1 * H_2) + H_2 (H_2 comes from bias)

2nd Hidden Layer - 3rd Hidden Layer = (H_2 * H_3) + H_3 (H_3 comes from bias)

… = ….

K-1th Hidden Layer - Kth Hidden Layer = (H_{k-1} * H_k) + H_k (H_k comes from bias)

Kth Hidden Layer- Output Layer Weights = (H_k * D_{out}) + D_{out} (D_{out} comes from bias)

+____

$$Number\ of\ Parameters = D_{out} + (D_{in} * H_1) + \sum_{k=1}^{k=K-1} (H_k * H_{k+1}) + \sum_{k=1}^{k=K} (H_k)$$

(Reference is https://www.quora.com/How-do-you-calculate-the-number-of-parameters-of-an-MLP-neural-network)

1.4.2 Consider a CNN classifier of K convolutional layers where the spatial size of each layer is Hk×Wk and the number of convolutional filters (kernels) of each layer is Ck for k=1, . . ., K. Derive a formula to compute the number of parameters that the CNN has if the input dimension is Hin×Win×Cin.

Similar to previous question, we can say that

Number of parameters = (Input Layer weights – 1st Hidden Layer) + (1st Hidden Layer - 2nd Hidden Layer) + (2nd Hidden Layer - 3rd Hidden Layer) + … + (Kth Hidden Layer- Output Layer Weights)

Input Layer Weights - 1st Hidden Layer = (C_{in} * C_1 * H_{in} * W_{in}) + C_1 (C_1 comes from bias)

1st Hidden Layer - 2nd Hidden Layer = (C_1 * C_2 * H_1 * W_1) + C_2 (C_2 comes from bias)

2nd Hidden Layer - 3rd Hidden Layer = (C_2 * C_3 * H_2 * W_2) + C_3 (C_3 comes from bias)

… = ….

K-1th Hidden Layer - Kth Hidden Layer = (C_{K-1} * C_K * H_{K-1} * W_{K-1}) + C_K (C_K comes frombias)

+____

$$Number\ of\ Parameters = (C_{in} * C_1 * H_{in} * W_{in}) + \sum_{k=1}^{k=K-1} (C_k * C_{k+1} * H_k * W_k) + \sum_{k=1}^{k=K} (C_k)$$

2. Experimenting ANN Architectures

2.1 Experimental Work

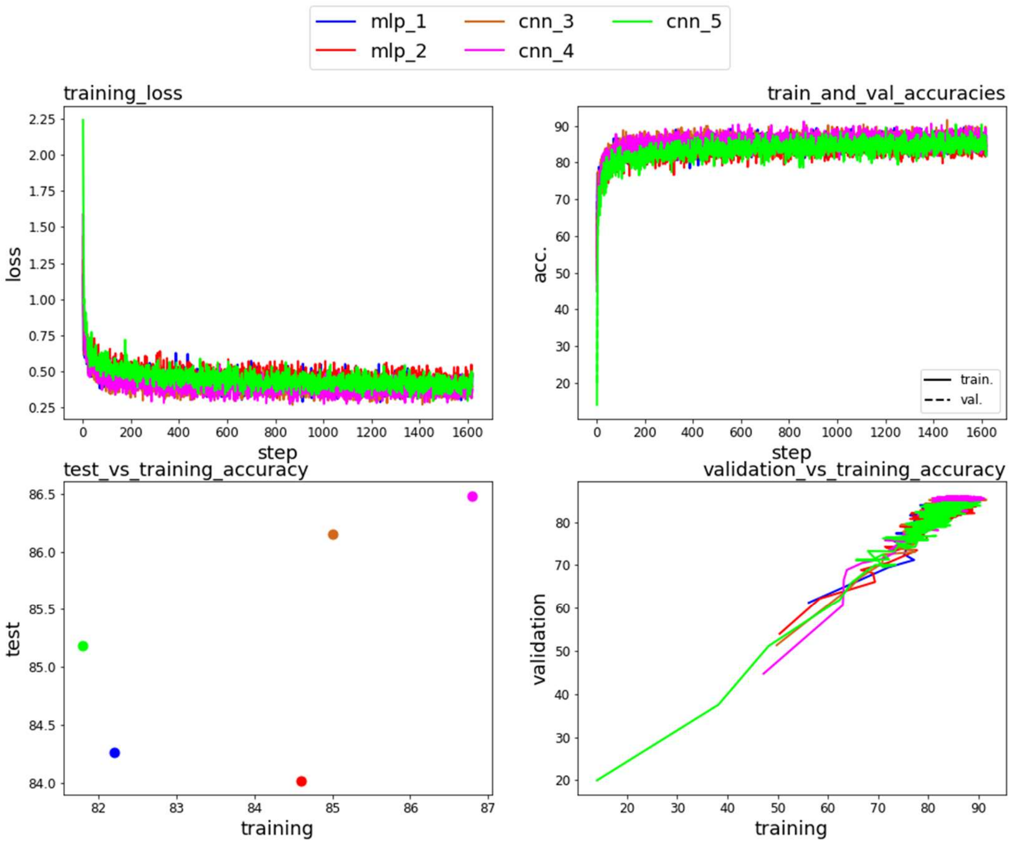


Figure 1: Result parameters for 5 different models

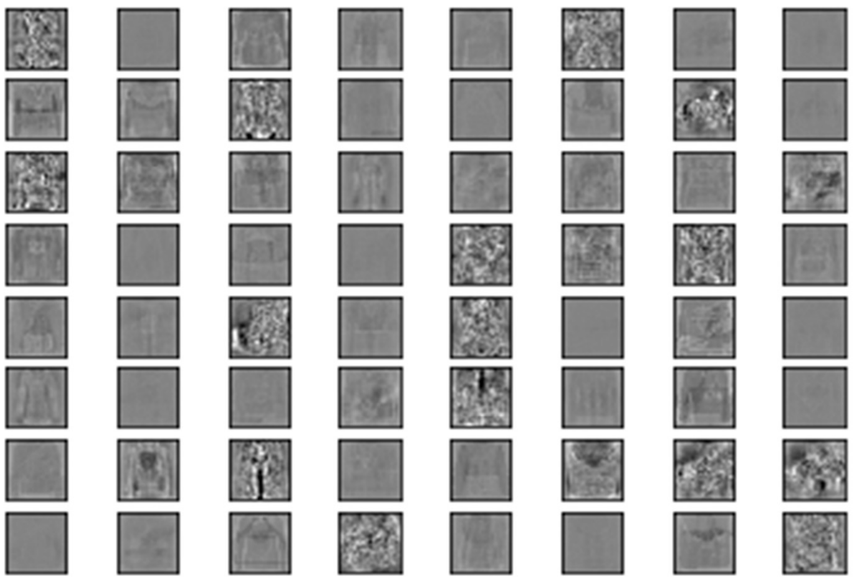


Figure 2: Weight (with highest test accuracy) in “mlp_1” model

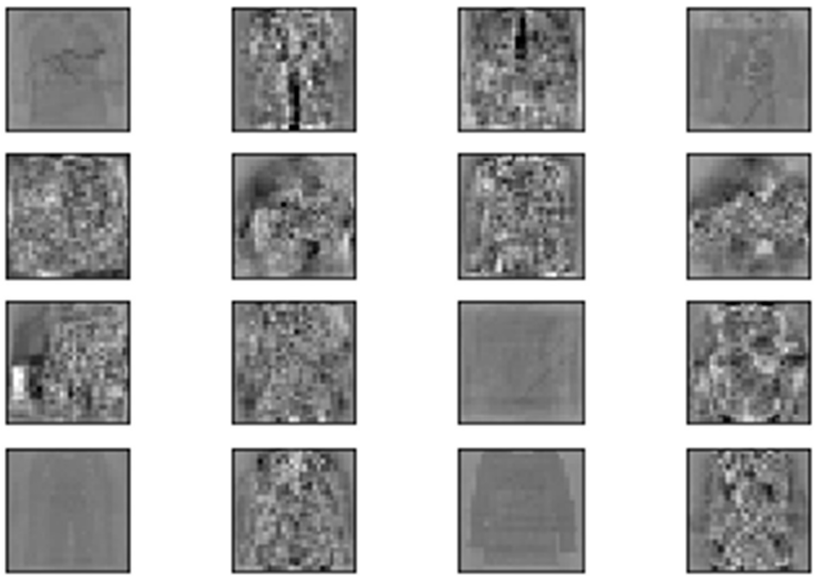


Figure 3: Weight (with highest test accuracy) in “mlp_2” model

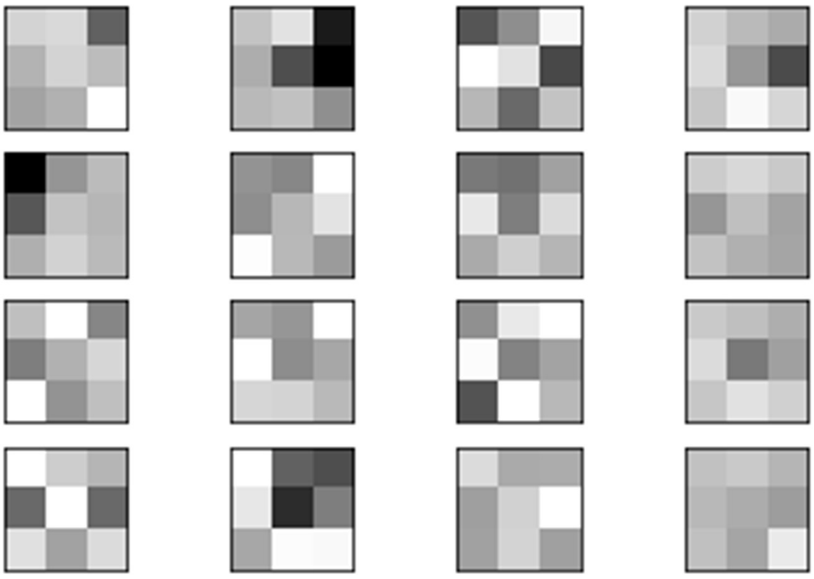


Figure 4: Weight (with highest test accuracy) in "cnn_3" model

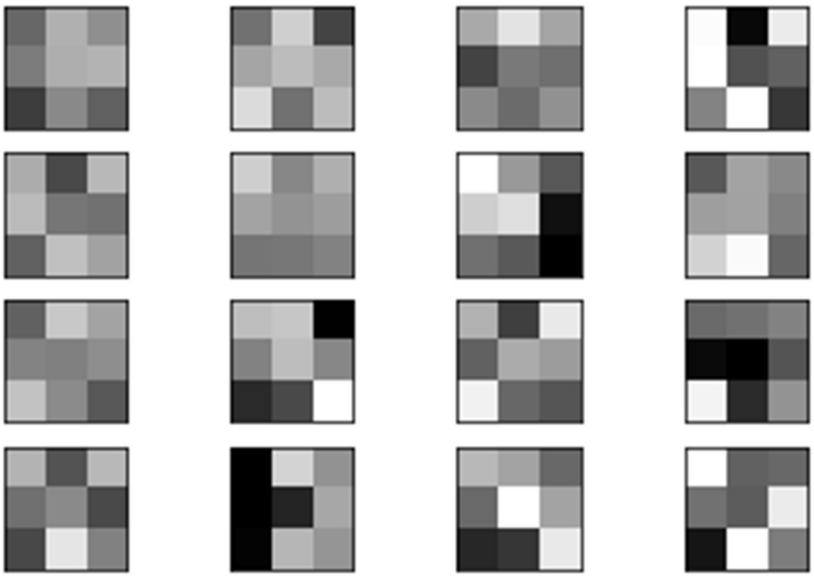


Figure 5: Weight (with highest test accuracy) in "cnn_4" model

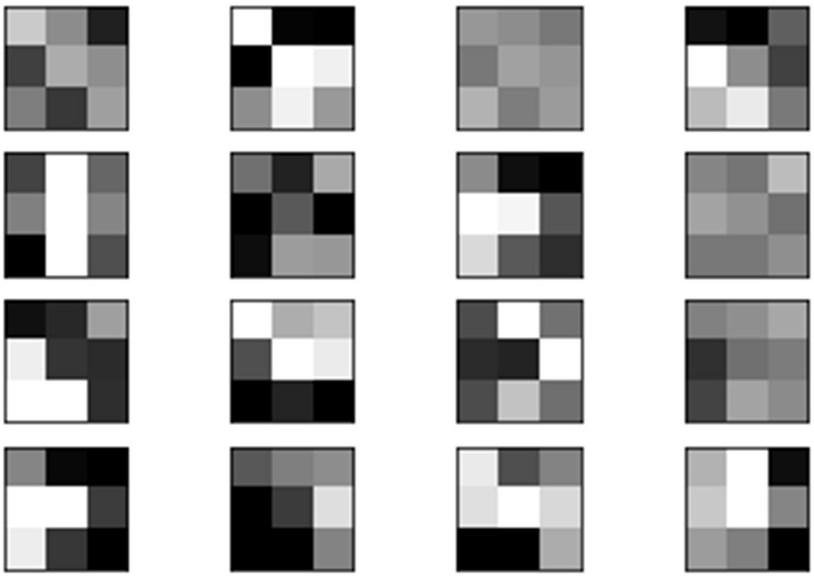


Figure 6: Weight (with highest test accuracy) in “cnn_5” model

2.2 Discussion

1. What is the generalization performance of a classifier?

The generalization performance of a learning algorithm refers to the performance on out-of-sample data of the models learned by the algorithm (Retrieved from https://link.springer.com/referenceworkentry/10.1007/978-0-387-30164-8_329). In other words, generalization performance depends on validation and test results. Train accuracy value has no contribution on generalization performance.

2. Which plots are informative to inspect generalization performance?

As explained in previous question, train accuracy is not a perfect parameter to inspect generalization performance. Therefore, to be able to comment on generalization performance of a model, we should consider the test and valid accuracy values of that model (left bottom plot, and right bottom plot).

3. Compare the generalization performance of the architectures.

Since “CNN_4” has the highest test accuracy result, it has the best generalization performance among all architectures. It also has the highest training accuracy; however, it has no effect on generalization performance.

4. How does the number of parameters affect the classification and generalization performance?

As the number of parameters increases, a neural network would response more than it did. Therefore, it will try to get more accurate result in training accuracy rather than test accuracy. The reason is that it will try to train itself according to train data, so it has no more general features in this model for unseen data. (Reference : https://www.kdnuggets.com/2019/11/generalization-neural-networks.html) Therefore, as number of parameters increase, there is a decrease in generalization performance.

In our case,

Parameters order: mlp_1>mlp_2>cnn_3>cnn_4>cnn_5

Cnn_5 should be perfect for generalization performance. However, in results, cnn_4 is the best (in terms of generalization performance). If we increase epoch size, I think we will get more correct result.

5. How does the depth of the architecture affect the classification and generalization performance?

It has a positive effect on both performances. However, as depth of the architecture increases, training will be harder. (Epoch size should be increased also)

In our case,

Depth order: cnn_5> cnn_4> cnn_3 > mlp_2> mlp_1

Cnn_5 should be perfect for generalization performance. However, in results, cnn_4 is the best (in terms of generalization performance). If we increase epoch size, I think we will get more correct result.

6. Considering the visualizations of the weights, are they interpretable?

Since it is the weights of the first layer, features should be low-level. Therefore, it is little bit hard to be interpretable. However, MLP models has a proper visualization of the weights whereas CNN ones do not have. It can be seen from figures 2-6. Figure 2 and 3 shows some blueprint of clothes (Output Classes). Therefore, MLP visualizations are interpretable while CNN ones are not.

7. Can you say whether the units are specialized to specific classes?

The answer should be no. Since we are in first layer, weights will determine the different features, not for classes. If we took the next layers, it could be pictures belonging to specific classes. For example, if we took last layer's weights, result will tell us specific classes.

8. Weights of which architecture are more interpretable?

As explained in question 6, MLP models are more interpretable.

9. Considering the architectures, comment on the structures (how they are designed). Can you say that some architectures are akin to each other? Compare the performance of similarly structured architectures and architectures with different structure.

As names of them implies, it can be divided into CNN and MLP models.

In CNN, as the number in a model (cnn_"3", cnn_"4" etc.) increases, depth of that model increases, and number of parameters decreases. Therefore, generalization should be increases; however, experimental results show cnn_4 is the best, and honestly, I don't know why. My single suggestion is to increase epoch size and repetition times.

To be able to comment on generalization performance, we should consider the difference between value of test and training accuracy. If testing accuracy is more higher than the training accuracy, then it means we are *overfitting*. If testing accuracy is more lower than the training accuracy, then it means we are *underfitting*. Therefore, this difference should be small

In MLP, the difference between training and test accuracies were small in mlp_2. Therefore, mlp_2 has better generalization performance.

10. Which architecture would you pick for this classification task? Why?

I will choose CNN_5 and increase epoch size or repetition time. If we are not able to increase these parameters, I would choose CNN_4. Since it has the greatest accuracy results for all curves. On the other hand, if increasing epoch size or repetition times are allowed, CNN_5 will be my choice.

3. Experimenting Activation Functions

3.1 Experimental Work

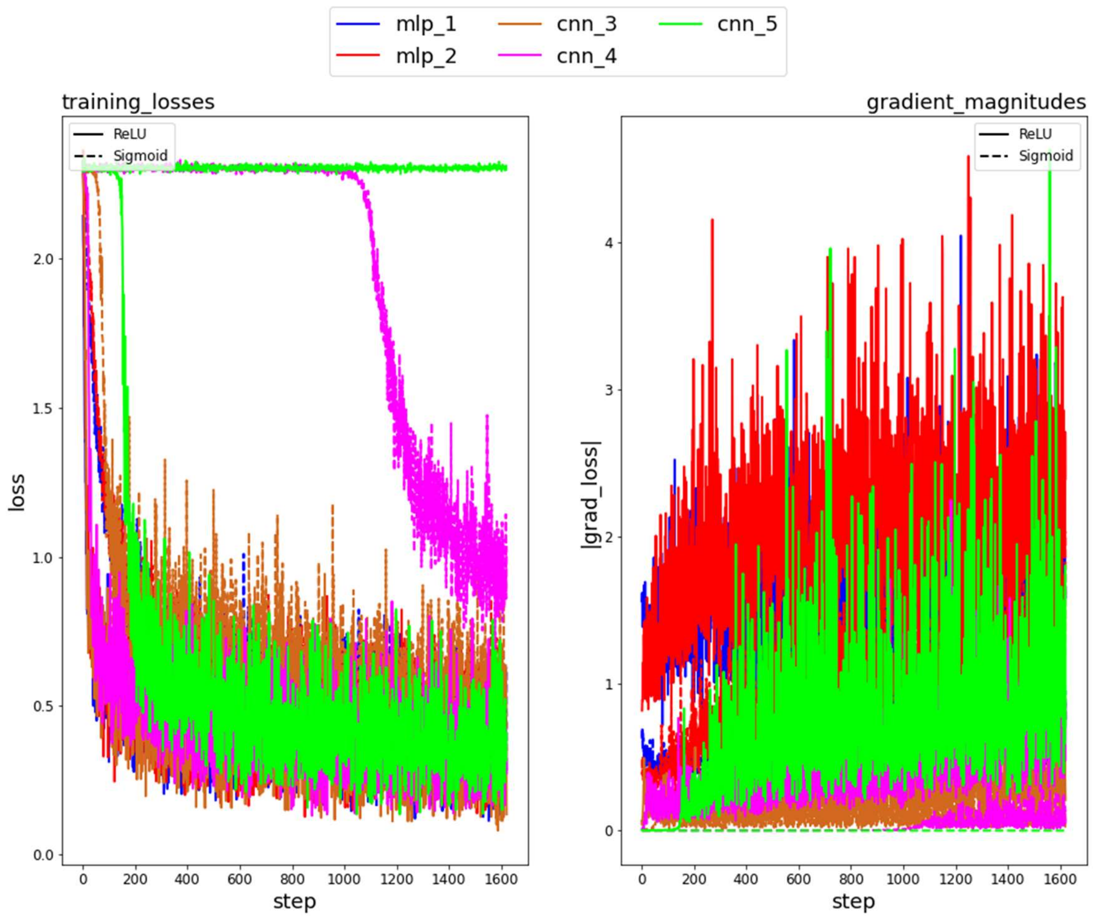


Figure 7: Result of 5 Models with ReLU and Sigmoid Functions

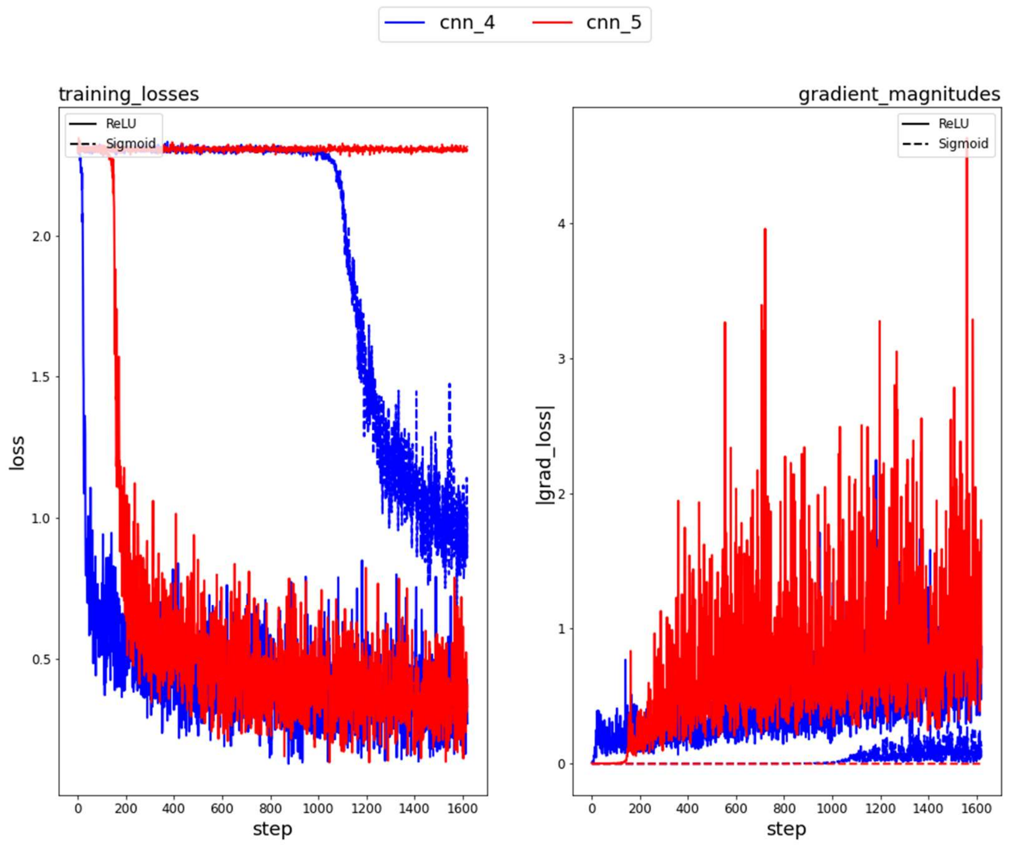


Figure 8: Result of 2 Models with ReLU and Sigmoid Functions

3.2 Discussions

1. How is the gradient behavior in different architectures? What happens when depth increases

Using ReLU causes bigger gradient than using sigmoid function as an activation function for both CNN and MLP architectures.

It is known that in CNN architectures, gradient tends to decrease as depth increases. However, this situation is visible when sigmoid function is used as experimental result shows. On the other hand, in MLP architectures, it is not observed, which is unexpected.

2. Why do you think that happens?

As depth increases, neural network becomes more complex system, which will have more steps. Therefore, our neural network will learn in a faster way, which means it can find the optimum values for weights easily. Therefore, as depth increases it is expected that gradient will become much smaller.

Moreover, for sigmoid function, output is in between 0 and 1 whereas Relu function outputs is in range of 0 and infinity. Therefore, the one model with ReLU has bigger gradient value compared to the one with Sigmoid due to range of output.

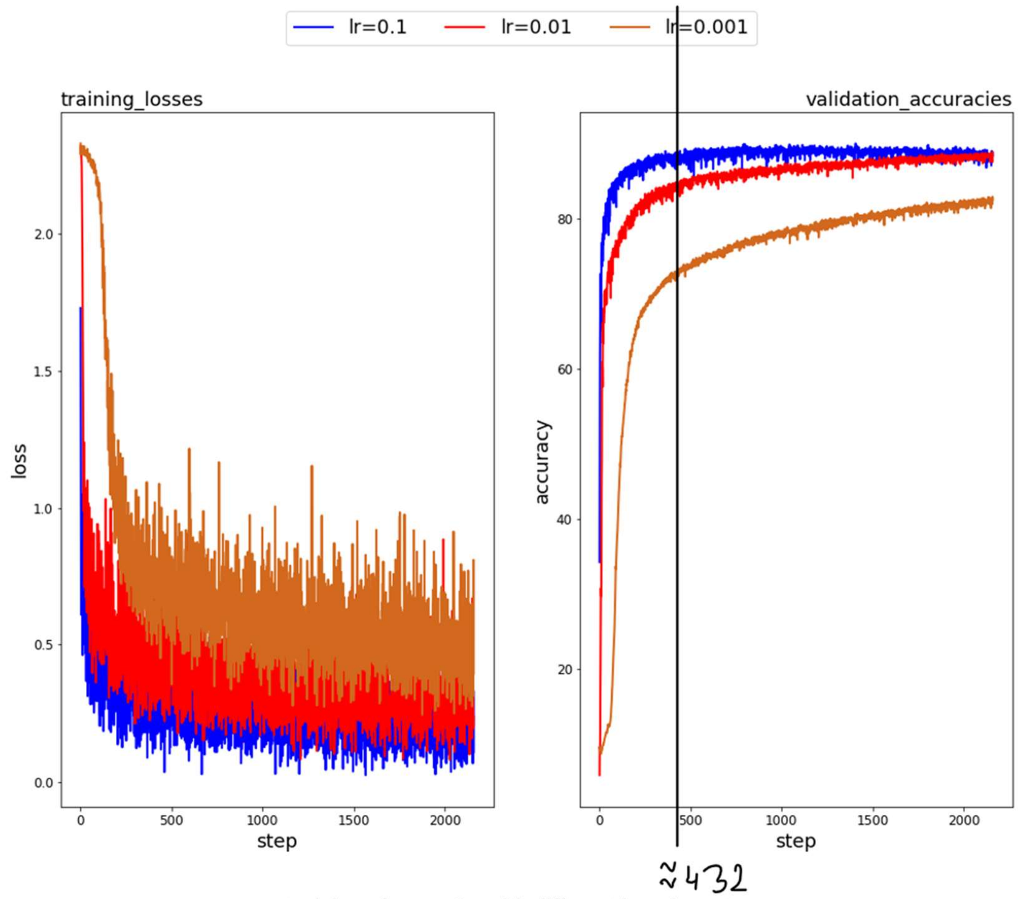
3. Bonus: What might happen if we do not scale the inputs to the range [−1.0, 1.0]?

If we did not scale the inputs, gradient descent will find it’s optimum point in a larger time (https://www.quora.com/Why-is-it-important-to-scale-your-inputs-in-gradient-descent). Therefore, gradient loss would become more than the one we got now. Also, learning rate should be decreased so that training will result in a better network.

4. Experimenting Learning Rate

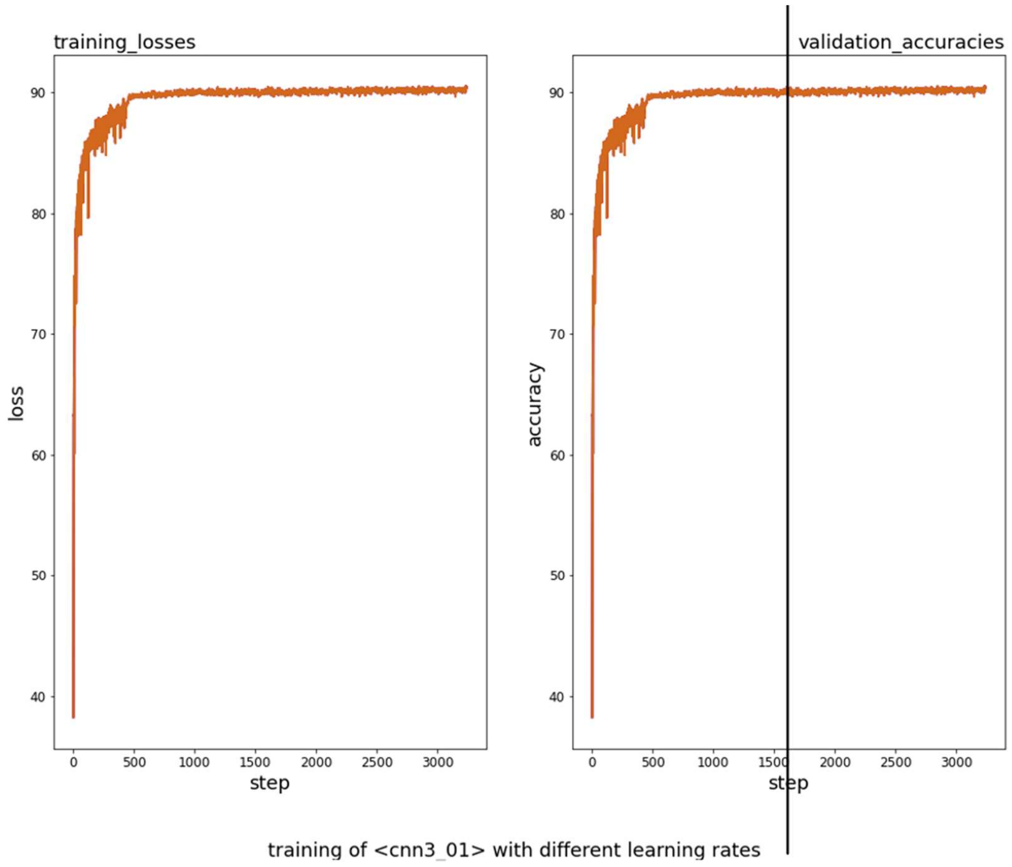
4.1 Experimental Work

Result is shown in below for different learning rates.

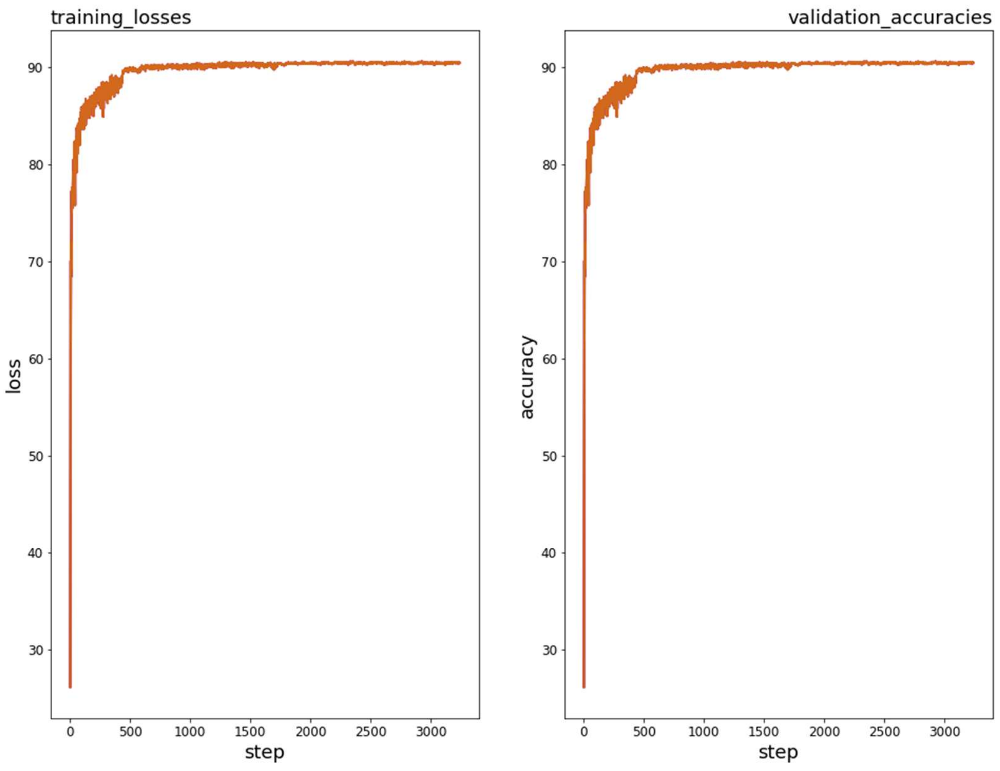


training of <cnn_3> with different learning rates

Turning learning rate from 0.1 to 0.01 at step of 450. (~432 = 4 EPOCH * 108 Step)



Turning learning rate from 0.1 to 0.01 at step of 1750. (~1728= 16 EPOCH * 108 Step)



training of <cnn3_01> with different learning rates

Compared to Adam optimizer, at the end there is in more stable condition.

4.2 Discussion

1. How does the learning rate affect the convergence speed?

As learning rate increases, converge speed is also increasing.

2. How does the learning rate affect the convergence to a better point?

As learning rate increases, it is hard to find better point. The reason is that as learning rate is increases, change of weight increases. Therefore, this change in weights leads us to have unstable accuracy rating. In this scenario, it can go into a local minimum, and it won't be able to go outside of that region. In other words, there is a trade-off between convergence time and stability.

3. Does your scheduled learning rate method work? In what sense?

As mentioned in previous question, stacking and unstable situation was occurred in learning rate of 0.1. On the other hand, when learning rate is equal to 0.001, converging time was very slow. By this method, advantages of these two cases are considered. In first steps, learning rate was high so that we can converge the stable point in earlier steps. As steps are done, learning rate is decreased to stabilize the curve. Although validation accuracy is not changed, convergence time is increased thanks to this method. The trade-off, that is mentioned in previous question, disappeared.

4. Compare the accuracy and convergence performance of your scheduled learning rate method with Adam.

Accuracy of these methods are nearly same. In terms of convergence time, scheduled learning rate is much more perfect as explained in previous question.

```python
import torchvision
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from sklearn.model_selection import train_test_split

import json
from utils import visualizeWeights

BATCH_SIZE=50
epoch_size = 15
TRAIN_SIZE=10
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/@aaysbt/fashion-mnist-data-training-using-pytorch-7f6ad71e96f4
cusTransform =
transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5,),(0.5,),)])

# 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 mlp 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
# 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
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
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        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.relu2 = 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)

#https://stackoverflow.com/questions/53580088/calculate-the-output-size-in-convolution-laye
r
        self.fc4 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #It didin't work ??????
        #x = x.view(-1, self.input_size)
        hidden1 = self.fc1(x)
        relu1 = self.relu1(hidden1)
        hidden2 = self.fc2(relu1)
        relu2 = self.relu2(hidden2)
        pool1 = self.maxpool1(relu2)
        hidden3 = self.fc3(pool1)
        pool2 = self.maxpool2(hidden3)
        #Reshaping linear input
        pool2=pool2.view(BATCH_SIZE,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_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
#Dictionary for json
dictonary ={
    'name': None,
    'loss_curve':None,
    'train_acc_curve':None,
    'val_acc_curve':None,
    'test_acc':None,
    'weights':None,
    }
#Model Types

models=['mlp_1','mlp_2','cnn_3','cnn_4','cnn_5']
for modelselected in models:
    print(f"Training is started for model {modelselected}")
    train_losses_total=[]
    train_accus_total=[]
    valid_accus_total=[]
    weight_best=None
    maxPerformanceTask=0
    for stepX in range(TRAIN_SIZE):
        print(f"Step {stepX+1} is started")
        if modelselected == 'mlp_1':
            model_mlp = mlp1(784,64,10).to(device)
        elif modelselected == 'mlp_2':
            model_mlp= mlp2(784,16,64,10).to(device)
        elif modelselected == 'cnn_3':
            model_mlp = cnn_3(784,10).to(device)
        elif modelselected == 'cnn_4':
            model_mlp = cnn_4(784,10).to(device)
        elif modelselected == 'cnn_5':
            model_mlp = cnn_5(784,10).to(device)
        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model_mlp.parameters(), lr = 0.01)

        #Recorded values for each try
        train_losses=[]
        train_accus=[]
        valid_accus=[]
        test_accus=[]
        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)
                # Forward pass
                outputs = model_mlp(images)
                loss = criterion(outputs, labels.to(device))

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


                if (i+1) % 10 == 0:
                    model_mlp.eval()
                    #Train calculation
                    #Directly taken from
https://discuss.pytorch.org/t/how-does-one-get-the-predicted-classification-label-from-a-py
torch-model/91649/3
                    _, tra_pred=outputs.max(1)
                    tra_size = labels.size(0)
                    tra_corr= tra_pred.eq(labels).sum().item()
                    tra_acc=tra_corr/tra_size
                    train_acc=tra_acc*100
                    running_loss = loss.item()
                    val_total=0
                    val_correct=0
                    #Valid
                    for j, (val_images, val_labels) in enumerate(val_generator):
                        # Accuracy Calculation
                        #
https://androidkt.com/calculate-total-loss-and-accuracy-at-every-epoch-and-plot-using-matpl
otlib-in-pytorch/
                        val_images = val_images.to(device)
                        val_labels = val_labels.to(device)
                        # Forward pass
                        val_outputs = model_mlp(val_images)
                        _, val_predicted = val_outputs.max(1)
                        val_total += val_labels.size(0)
                        val_correct += val_predicted.eq(val_labels).sum().item()
                    val_accu=(val_correct/ val_total)*100
                    #print ('Step {} Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Train
Accuracy {:.4f}, Val Accuracy {:.4f} '
                    #       .format(stepX,epoch+1, epoch_size, i+1, total_step,
running_loss,train_acc,val_accu))
                    train_losses.append(running_loss)
                    train_accus.append(train_acc)
```

```
                  valid_accus.append(val_accu)
        train_losses_total.append(train_losses)
        train_accus_total.append(train_accus)
        valid_accus_total.append(valid_accus)
        #print(train_losses_total)

        #Test
        with torch.no_grad():
            test_generator = torch.utils.data.DataLoader(test_data, batch_size =
BATCH_SIZE, shuffle = False)
            model_mlp.eval()
            correct = 0
            total = 0
            for images, labels in test_generator:
                images = images.to(device)
                labels = labels.to(device)
                outputs = model_mlp(images)
                _, predicted = outputs.max(1)
                total += labels.size(0)
                correct += predicted.eq(labels).sum().item()
            test_accu =(correct/total)*100
            test_accus.append(test_accu)
            if(test_accu > maxPerformanceTask):
                maxPerformanceTask=test_accu
                model_mlp.to('cpu')
                weight_best=model_mlp.fc1.weight.data.numpy()
                model_mlp.to(device)

        print('For Step {} It is finished'.format(stepX+1))
    #https://www.geeksforgeeks.org/python-column-wise-sum-of-nested-list/
    average_train_losses = [sum(sub_list) / len(sub_list) for sub_list in
zip(*train_losses_total)]
    average_train_accu = [sum(sub_list) / len(sub_list) for sub_list in
zip(*train_accus_total)]
    average_valid_accu = [sum(sub_list) / len(sub_list) for sub_list in
zip(*valid_accus_total)]


    #Dictionary for json
    dictonary ={
         'name': modelselected,
         'loss_curve':average_train_losses,
         'train_acc_curve':average_train_accu,
        'val_acc_curve':average_valid_accu,
        'test_acc':maxPerformanceTask,

'weights':weight_best.tolist(),#https://stackoverflow.com/questions/26646362/numpy-array-is
-not-json-serializable
        }
    #JSON Writing a file (geeksforgeeks.com)
    with open("q2_"+modelselected+".json","w") as outfile:
        json.dump(dictonary,outfile)

    visualizeWeights(weight_best, save_dir='resultQ2',
filename='input_weights_'+modelselected)
```

```
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/@aaysbt/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 mu 0 mu
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        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 mu 0 mu
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        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-network-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-network-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-netw
ork-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)
```

```
import torchvision
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from sklearn.model_selection import train_test_split
import json


BATCH_SIZE=50
epoch_size =20
#Faster
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/@aaysbt/fashion-mnist-data-training-using-pytorch-7f6ad71e96f4
cusTransform =
transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5,),(0.5,),)])

# training set
train_data = torchvision.datasets.FashionMNIST('./data', train = True, download = True,
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




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
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        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.relu2 = 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)

#https://stackoverflow.com/questions/53580088/calculate-the-output-size-in-convolution-laye
r
        self.fc4 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #It didin't work ??????
        #x = x.view(-1, self.input_size)
        hidden1 = self.fc1(x)
        relu1 = self.relu1(hidden1)
        hidden2 = self.fc2(relu1)
        relu2 = self.relu2(hidden2)
        pool1 = self.maxpool1(relu2)
        hidden3 = self.fc3(pool1)
        pool2 = self.maxpool2(hidden3)
        #Reshaping linear input
        pool2=pool2.view(BATCH_SIZE,144)
        output = self.fc4(pool2)
        return output

#Model Types
train_losses_total=[]
valid_accus_total=[]
lrs=[0.1,0.01,0.001] #Learning Rates
```

```
for LR in lrs:
    print(f"Training is started for Learining Rate {LR}") # Printing LR state
    model_mlp = cnn_3(784,10).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model_mlp.parameters(), lr = LR, momentum=0.00)#Setting LR
    #Recorded values for each try
    train_losses=[]
    valid_accus=[]

    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)
            # Forward pass
            outputs = model_mlp(images)
            loss = criterion(outputs, labels.to(device))

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


            if (i+1) % 10 == 0:
                model_mlp.eval()
                #Train calculation
                #Directly taken from
https://discuss.pytorch.org/t/how-does-one-get-the-predicted-classification-label-from-a-py
torch-model/91649/3
                running_loss = loss.item()
                val_generator = torch.utils.data.DataLoader(val_set, batch_size =
BATCH_SIZE, shuffle = False)
                val_total=0
                val_correct=0
                #Valid
                for j, (val_images, val_labels) in enumerate(val_generator):
                    # Accuracy Calculation
                    #
https://androidkt.com/calculate-total-loss-and-accuracy-at-every-epoch-and-plot-using-matpl
otlib-in-pytorch/
                    val_images = val_images.to(device)
                    val_labels = val_labels.to(device)
                    # Forward pass
                    val_outputs = model_mlp(val_images)
                    _, val_predicted = val_outputs.max(1)
                    val_total += val_labels.size(0)
                    val_correct += val_predicted.eq(val_labels).sum().item()
                val_accu=(val_correct/ val_total)*100
                train_losses.append(running_loss)
                valid_accus.append(val_accu)
    train_losses_total.append(train_losses)
    valid_accus_total.append(valid_accus)
    #Dictionary for json
dictonary ={
     'name': 'cnn3',
     'loss_curve_1':train_losses_total[0],
     'loss_curve_01':train_losses_total[1],
     'loss_curve_001':train_losses_total[2],
     'val_acc_curve_1':valid_accus_total[0],
     'val_acc_curve_01':valid_accus_total[1],
     'val_acc_curve_001':valid_accus_total[2],
     }
#Recording Results
with open("Q4cnn3_1.json","w") as outfile:
    json.dump(dictonary,outfile)
```

```
import torchvision
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from sklearn.model_selection import train_test_split
import json

BATCH_SIZE=50
epoch_size =30
#Faster
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/@aaysbt/fashion-mnist-data-training-using-pytorch-7f6ad71e96f4
cusTransform =
transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5,),(0.5,),)])

# training set
train_data = torchvision.datasets.FashionMNIST('./data', train = True, download = True,
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)



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 m1 0 m1
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.relu2 = 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)

#https://stackoverflow.com/questions/53580088/calculate-the-output-size-in-convolution-laye
r
        self.fc4 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #It didin't work ??????
        #x = x.view(-1, self.input_size)
        hidden1 = self.fc1(x)
        relu1 = self.relu1(hidden1)
        hidden2 = self.fc2(relu1)
        relu2 = self.relu2(hidden2)
        pool1 = self.maxpool1(relu2)
        hidden3 = self.fc3(pool1)
        pool2 = self.maxpool2(hidden3)
        #Reshaping linear input
        pool2=pool2.view(BATCH_SIZE,144)
        output = self.fc4(pool2)
        return output

#Model Types
lrs=[0.1,0.01,0.001]

print(f"Training is started for Learining Rate {lrs[0]}")
model_mlp = cnn_3(784,10).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_mlp.parameters(), lr = lrs[0], momentum=0.00)
```

```
#Recorded values for each try

valid_accus=[]
flag=True
#Flag for changing Lr from 0.1 to 0.01
for epoch in range(epoch_size):
    if((epoch==4) and (flag)): #(4*108)~=432nd step (See the report)
        optimizer = torch.optim.SGD(model_mlp.parameters(), lr = lrs[1], momentum=0.00)
        flag=False
        print("Learning Rate is changed") #Informing change in LR
    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)
        # Forward pass
        outputs = model_mlp(images)
        loss = criterion(outputs, labels.to(device))

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


        if (i+1) % 10 == 0:
            model_mlp.eval()
            #Train calculation
            #Directly taken from 
https://discuss.pytorch.org/t/how-does-one-get-the-predicted-classification-label-from-a-py
torch-model/91649/3
            running_loss = loss.item()
            val_generator = torch.utils.data.DataLoader(val_set, batch_size = BATCH_SIZE, 
shuffle = False)
            val_total=0
            val_correct=0
            #Valid
            for j, (val_images, val_labels) in enumerate(val_generator):
                # Accuracy Calculation
                # 
https://androidkt.com/calculate-total-loss-and-accuracy-at-every-epoch-and-plot-using-matpl
otlib-in-pytorch/
                val_images = val_images.to(device)
                val_labels = val_labels.to(device)
                # Forward pass
                val_outputs = model_mlp(val_images)
                _, val_predicted = val_outputs.max(1)
                val_total += val_labels.size(0)
                val_correct += val_predicted.eq(val_labels).sum().item()
            val_accu=(val_correct/ val_total)*100

            valid_accus.append(val_accu)


#Dictionary for json
dictonary ={
     'name': 'cnn3_01',
     'loss_curve_1':valid_accus,
     'loss_curve_01':valid_accus,
     'loss_curve_001':valid_accus,
     'val_acc_curve_1':valid_accus,
     'val_acc_curve_01':valid_accus,
     'val_acc_curve_001':valid_accus,
     }
#Recording Results
with open("Q4cnn3_01.json","w") as outfile:
    json.dump(dictonary,outfile)
```

```
import torchvision
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from sklearn.model_selection import train_test_split
import json

BATCH_SIZE=50
epoch_size =30
#Faster
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/@aaysbt/fashion-mnist-data-training-using-pytorch-7f6ad71e96f4
cusTransform =
transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5,),(0.5,),)])

# 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)




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 m1 0 m1
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.relu2 = 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)

#https://stackoverflow.com/questions/53580088/calculate-the-output-size-in-convolution-laye
r
        self.fc4 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #It didin't work ??????
        #x = x.view(-1, self.input_size)
        hidden1 = self.fc1(x)
        relu1 = self.relu1(hidden1)
        hidden2 = self.fc2(relu1)
        relu2 = self.relu2(hidden2)
        pool1 = self.maxpool1(relu2)
        hidden3 = self.fc3(pool1)
        pool2 = self.maxpool2(hidden3)
        #Reshaping linear input
        pool2=pool2.view(BATCH_SIZE,144)
        output = self.fc4(pool2)
        return output

#Model Types
lrs=[0.1,0.01,0.001]

print(f"Training is started for Learining Rate {lrs[0]}")
model_mlp = cnn_3(784,10).to(device)
criterion = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.SGD(model_mlp.parameters(), lr = lrs[0], momentum=0.00)
#Recorded values for each try

valid_accus=[]
flag=True
#Flag for changing Lr from 0.1 to 0.01
flag1=True
#Flag for changing Lr from 0.01 to 0.001
for epoch in range(epoch_size):
    if((epoch==4) and (flag)): #(4*108)~=432nd step (See the report)
        optimizer = torch.optim.SGD(model_mlp.parameters(), lr = lrs[1], momentum=0.00)
        flag=False
        print("Learning Rate is changed") #Informing change in LR
    if((epoch==16) and (flag1)): #(16*108)~=1728th step (See the report)
        optimizer = torch.optim.SGD(model_mlp.parameters(), lr = lrs[2], momentum=0.00)
        flag1=False
        print("Learning Rate is changed") #Informing change in LR
    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)
        # Forward pass
        outputs = model_mlp(images)
        loss = criterion(outputs, labels.to(device))

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


        if (i+1) % 10 == 0:
            model_mlp.eval()
            #Train calculation
            #Directly taken from
https://discuss.pytorch.org/t/how-does-one-get-the-predicted-classification-label-from-a-py
torch-model/91649/3
            running_loss = loss.item()
            val_generator = torch.utils.data.DataLoader(val_set, batch_size = BATCH_SIZE,
shuffle = False)
            val_total=0
            val_correct=0
            #Valid
            for j, (val_images, val_labels) in enumerate(val_generator):
                # Accuracy Calculation
                #
https://androidkt.com/calculate-total-loss-and-accuracy-at-every-epoch-and-plot-using-matpl
otlib-in-pytorch/
                val_images = val_images.to(device)
                val_labels = val_labels.to(device)
                # Forward pass
                val_outputs = model_mlp(val_images)
                _, val_predicted = val_outputs.max(1)
                val_total += val_labels.size(0)
                val_correct += val_predicted.eq(val_labels).sum().item()
            val_accu=(val_correct/ val_total)*100

            #print ('Step {} Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Train Accuracy
{:.4f}, Val Accuracy {:.4f} '
            #       .format(stepX,epoch+1, epoch_size, i+1, total_step,
running_loss,train_acc,val_accu))
            valid_accus.append(val_accu)
```

```
#Dictionary for json
dictonary ={
      'name': 'cnn3_001',
      'loss_curve_1':valid_accus,
      'loss_curve_01':valid_accus,
      'loss_curve_001':valid_accus,
      'val_acc_curve_1':valid_accus,
      'val_acc_curve_01':valid_accus,
      'val_acc_curve_001':valid_accus,
      }
with open("Q4cnn3_001.json","w") as outfile:
    json.dump(dictonary,outfile)
```