

Clothing Image Identification

Below is a model that uses dataset taken from pytorch.datasets module to train a 2 layer neural network in order to predict different clothing images(10 types).

Importing all relevant Modules

```
In [1]: import torch
        from torch import nn, optim
        import matplotlib.pyplot as plt
        import helper
        from torchvision import datasets, transforms
```

The helper module is not generic to pytorch, it was developed by Udacity course on Deep learning.

Below we extract FashionMNIST dataset from pytorch.datasets module which contains the file for raw data containing clothing images of size (1, 28, 28).

The transform key_word transform the image into a tensor with normalisation using pytorch.transforms module

We only take 64 dataset at a time

```
In [2]: transform = transforms.Compose([transforms.ToTensor(),
                                     transforms.Normalize((0.5,), (0.5,))
                                     ])

        trainset = datasets.FashionMNIST('F_MNIST_data/', download = True, train = True, transform = transform)
        trainloader = torch.utils.data.DataLoader(trainset, batch_size = 64, shuffle = True)
```

Creating the feedforward model in pytorch.nn

Below we create an feedforward object for the neural network containing 2 hidden layers . The first layer has 784 nodes or perceptrons, second hidden layer has 128, 3rd hidden layer has 64 and the output layer has 10 nodes or perceptrons.

In the feedforward model we use ReLU(Rectified Linear Unit) as the activation function.

We use `logsoftmax` function for output from feedforward and use `NLLLoss` for calculating loss, just because it is faster.

```
In [3]: input_size = 784
        hidden_layer1_size = 128
        hidden_layer2_size = 64
        output_size = 10

        model = nn.Sequential(nn.Linear(input_size,hidden_layer1_size),nn.ReLU(),
                               nn.Linear(hidden_layer1_size,hidden_layer2_size),nn.ReLU(),
                               nn.Linear(hidden_layer2_size,output_size),nn.ReLU(),
                               nn.LogSoftmax(dim =1))

        criterion = nn.NLLLoss()
```

Then We create an optimizer from `torch.optim` module, in order to update the weights while back propagating through the network and calculating loss and gradient.

For updating the weights `SGD(Stochastic Gradient Descent)` function is used.

The learning rate is taken as 0.01.

```
In [4]: optimizer = optim.SGD(model.parameters(),lr =0.01)
```

Training the data

Below we use a for loop to loop through the `trainloader` dataset and pass images and labels through the feedforward network. After that we backpropagate and update the weights with `optim` module.

We calculate the loss at every step and then display the loss at the end. We can see the losses are decreasing with each epoch .

```
In [5]: epoch = 20
        train_losses = []
        for i in range(epoch):
            rn_loss = 0
            for images,labels in trainloader:
                inputs = images.view(images.shape[0],-1)

                optimizer.zero_grad()
```

```

        output = model.forward(inputs)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()
        rn_loss += loss.item()

    else:
        print(f"Training loss: {rn_loss/len(trainloader)}")
        train_losses.append(rn_loss/len(trainloader))

```

```

Training loss: 1.4631797289416226
Training loss: 0.9957203385291069
Training loss: 0.9183321554205819
Training loss: 0.8830152005274922
Training loss: 0.8612740070009028
Training loss: 0.8458520594054956
Training loss: 0.8329476550189671
Training loss: 0.8224763173157218
Training loss: 0.8126967021905537
Training loss: 0.8043765098110699
Training loss: 0.7965948650641228
Training loss: 0.7903258902177628
Training loss: 0.7840665582972549
Training loss: 0.7782179351046141
Training loss: 0.772432480952633
Training loss: 0.7676274513384935
Training loss: 0.7623370508713
Training loss: 0.757367528196591
Training loss: 0.7530687401162536
Training loss: 0.7492325120389081

```

Testing the model with testdata

We then extract data from pytorch.datasets untrained in order to test our model accuracy.

```

In [6]: testset = datasets.FashionMNIST('~/.pytorch/F_MNIST_data/', download=True, train=False, transform=transform)
        testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=True)

```

Checking the output of 1 image in the testdata and the probability prediction of trained model

```

In [7]: %matplotlib inline
        %config InlineBackend.figure_format = 'retina'

        import helper

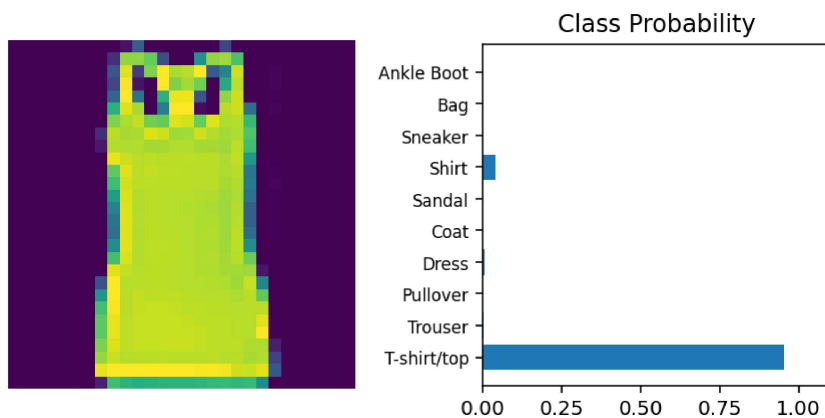
        # Test out your network!

        dataiter = iter(testloader)
        images, labels = dataiter.next()
        img = images[1]
        # Convert 2D image to 1D vector
        img = img.resize_(1, 784)
        with torch.no_grad():
            logps = model(img)

        # TODO: Calculate the class probabilities (softmax) for img
        ps = torch.exp(logps)

        # Plot the image and probabilities
        helper.view_classify(img.resize_(1, 28, 28), ps, version='Fashion')

```



Accuracy:

To calculate accuracy we take the highest scoring probability from the model for each image through `topk` function, which returns a tuple with index and value of probability.

```

In [8]: images = images.view(images.shape[0], -1)

```

```
ps = torch.exp(model(images))
top_p,top_class = ps.topk(1,dim =1)
```

We calculate the matching cases of model with the actual labels as below:

```
In [9]: equal = top_class == labels.view(*top_class.shape)
```

Result:

The accuracy of model then calculated by taking mean of the `equal` tensor (because the inputs are all zeros and ones).

```
In [10]: accuracy = torch.mean(equal.type(torch.FloatTensor))
print(f"Accuracy of Model: {accuracy.item()*100} %")
```

Accuracy of Model: 76.5625 %

The accuracy is 76% but can be improved on further epochs to 96.875%

Including Dropout:

The traing and validation looks like:

```
In [ ]: test_losses = []
        for i in range(epoch):
            test_loss = 0
            with torch.no_grad():
                for images,labels in testloader:
                    img = images.view(images.shape[0],-1)
                    logps = model(img)
                    test_loss += criterion(logps,labels)
            test_losses.append(test_loss/len(testloader))
```

```
In [ ]: plt.plot(train_losses,label = "Training Loss")
        plt.plot(test_losses,label = "Testing Loss")
        plt.legend()
        plt.show()
```

Another approach

```
In [22]: import torch.nn.functional as F
```

```
In [23]: class Network(nn.Module):

    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784,128)
        self.fc2 = nn.Linear(128,64)
        self.fc3 = nn.Linear(64,10)
        self.dropout = nn.Dropout(p =0.2)

    def forward(self,x):

        x = x.view(x.shape[0],-1)
        x= self.dropout(F.relu(self.fc1(x)))
        x= self.dropout(F.relu(self.fc2(x)))
        x = F.log_softmax(self.fc3(x),dim =1)

        return x
```

```
In [24]: model = Network()
         model
```

```
Out[24]: Network(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=10, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

```
In [25]: criterion = nn.NLLLoss(reduction = 'sum')
```

```
In [ ]: epoch = 5
```

```

train_losses, test_losses = [], []

for i in range(epoch):
    rn_loss = 0
    for images, labels in trainloader:

        optimizer.zero_grad()

        output = model.forward(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()
        rn_loss += loss.item()
    else:
        tot_test_loss = 0
        test_correct = 0 # Number of correct predictions on the test set

        # Turn off gradients for validation, saves memory and computations
        with torch.no_grad():
            for images, labels in testloader:
                log_ps = model(images)
                loss = criterion(log_ps, labels)
                tot_test_loss += loss.item()

                ps = torch.exp(log_ps)
                top_p, top_class = ps.topk(1, dim=1)
                equals = top_class == labels.view(*top_class.shape)
                test_correct += equals.sum().item()

        # Get mean Loss to enable comparison between train and test sets
        train_loss = rn_loss / len(trainloader.dataset)
        test_loss = tot_test_loss / len(testloader.dataset)

        # At completion of epoch
        train_losses.append(train_loss)
        test_losses.append(test_loss)

    print("Epoch: {}/{}.. ".format(i+1, epoch),
          "Training Loss: {:.3f}.. ".format(train_loss),
          "Test Loss: {:.3f}.. ".format(test_loss),
          "Test Accuracy: {:.3f}".format(test_correct / len(testloader.dataset)))

```

```

In [ ]: plt.plot(train_losses, label = "Training Loss")
        plt.plot(test_losses, label = "Testing Loss")

```

```
plt.legend()  
plt.show()
```

```
In [ ]: torch.save(model.state_dict(), "Clothing_classifier.pth")
```

```
In [ ]:
```