```
import my_utils as mu
import torch
from torch import nn
import numpy as np
```

Elliot Linsey Neural Network ECS659P QMUL (FASHION MNIST)

To reduce space, I have removed most of the actual functions from this pdf report. The full functions are detailed in the submitted jupyter notebook file.

Downloading Data

Firstly, I download the Fashion MNIST dataset using the inbuilt my_utils function

```
# Read training and test data
batch_size = 256
train_iter, test_iter = mu.load_data_fashion_mnist(batch_size)
# type(train_iter)
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarnin
  cpuset_checked))
```

Importing the data and checking the shape

Each batch comes as a tensor of shape [256, 1, 28, 28]. For every 28x28 image we want to cut it into 4 patches.

```
X,y = next(iter(train_iter))
print(X.shape)
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarnin
  cpuset_checked))
torch.Size([256, 1, 28, 28])
```

Patching

To cut into patches, we use the .unfold() function on every batch, this creates a [256, 1, 2, 2, 14, 14] tensor. Our desired shape is [256,4,196]. That is, 256 images with 4 patches, each patch flattened to 196 pixels, per batch. This is achieved using created patching functions.

The Net

The net intakes each batch of [256,1,28,28]. The stem contains the patching function, every batch that is fed into the net is patched first. From here, a linear layer is applied with input 196 and also outputting 196.

The backbone contains 2 blocks, both containing 2 MLPs and two transpose functions. Within the first block the idea is to increase the number of features, then reduce it within the second block.

A mean of all the features is taken, then fed into a Classifier with output 10 to match the number of class labels.

Loss and Optimization Algorithm

I use Cross Entropy Loss and the ADAM optimization algorithm

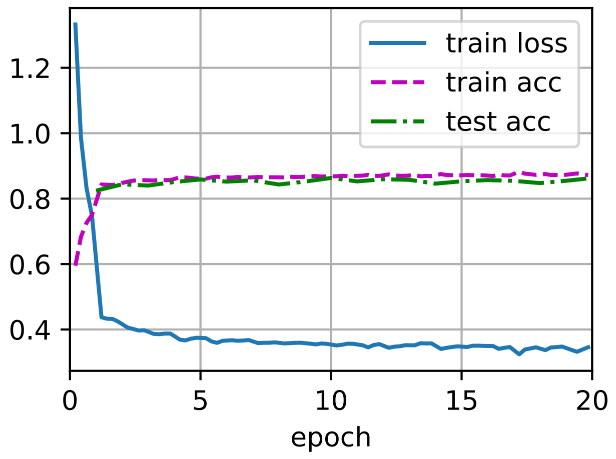
Training

- Using the 'mu.trainf' function that I have put in my edited my_utils file, also in the submitted zip file.

```
num_epochs = 20
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') #choose device: cpu
print('Using device:', device)
if torch.cuda.is_available(): print(torch.cuda.get_device_name(0)) # print the type of the
mu.trainf(net, train_iter, test_iter, loss, num_epochs, optimizer, device)
```

```
    loss 0.347, train acc 0.872, test acc 0.863
    4253.7 examples/sec on cuda
```



Results

After 20 Epochs, we achieve a test accuracy of 0.863. This is close to the training accuracy of 0.872 so it does not appear that we have overfitted. The loss is 0.347 and appears to have plateaued by this point.

Hyperparameters used:

Epochs: 20

Batch size: 256

Patch size: 14x14

Learning rate: 0.01

Weight decay: 0.0005

Optimizer: ADAM

Loss: Cross entropy loss

Weight initialisation: kaiming_normal_

Number of blocks: 2

Thanks for reading!

Elliot Linsey

QMUL

References

All functions from my_utils were taken from the d2l course and website