Neural networks - ECS659P

Task 1 – Read dataset and create data loaders

We read the dataset for Fashion MNIST and loaded the data into a design matrix X which has the ground truth outputs Y. The first hyperparameter we set is the batch size and we chose 256.

Task 2 – Stem and Backbone

After we created the stem and put the 28x28 image into patches, we needed to put it through a linear layer to get the feature vector of the patches. Next we put the outputs through the block which contains two MLPS (hidden layers). Each weight (w1,…w4) corresponds to a linear transformation. We put the inputs (patches) through each layer.

The activation function creates non linearity. We chose the ReLu as it is the most popular and performs well on most models. The activation functions allows us to train a model to complex data and if we don't use this we are essentially running a linear model regardless of how many layers we use. We use the ReLu twice so we have 2 hidden layers (MLPS). We defined the amount of perceptron's in these layers and tuning this setting determines the accuracy also. This defines how many features we will extract.

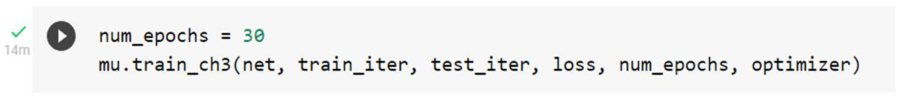
The last step of the classifier is SoftMax regression and we actually have this built into our loss function so there was no need to define this in the model.

Task 3 – Loss and optimiser

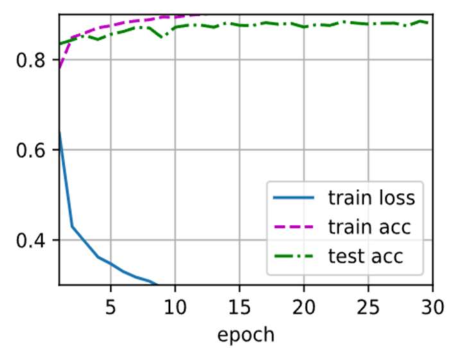
We chose cross entropy loss as the SoftMax regression classifier is built in to the loss function, so we did not need to implement a softmax activation function in our model. This allows us to change the outputs into probabilities automatically.

The first optimiser we chose was SGD. This didn't give a great accuracy as we only got 83%. We changed our optimisation algorithm to Adam which gave a much higher test accuracy of over 85%. Adam is better for more complex data such as images we discovered.

Task 4 – Training the model



```
num_epochs = 30
mu.train_ch3(net, train_iter, test_iter, loss, num_epochs, optimizer)
```



4.1. The curves for evolution of loss

We can see from the graph above, the curve for loss decreases as the amount of epochs increases. The loss has already reached close to it's minimum at less than 10 epochs so not much will change in terms of accuracy after this point. We may need to consider using less epochs next time as this may have been unnecessary excessive. At each batch we update our model to reduce loss and we can see the test and train accuracy increase as loss decreases.

4.2. The curves for the evolution of training and test accuracies

We can see from the graph that the curves for evolution of training and test accuracy are increasing as the amount of epochs increase and the loss decreases. This shows the power of gradient descent as we keep optimising our parameters which we initialised of weights and biases to gwt the best combination which allows us to keep the error as close to 0 as possible. By the 10th epoch training and test accuracies were near enough at their peak. Train accuracy always remained a lot higher than test accuracy which is good as the data has been seen before, however 94% accuracy on the training dataset shows possible overfitting.

4.3. Training details including hyperparameters

Hyper parameters are manually pre specified. We experimented with different numbers to see what brought us the best results. We tuned our hyperparameters based on the best test accuracy and least overfitting simultaneously.

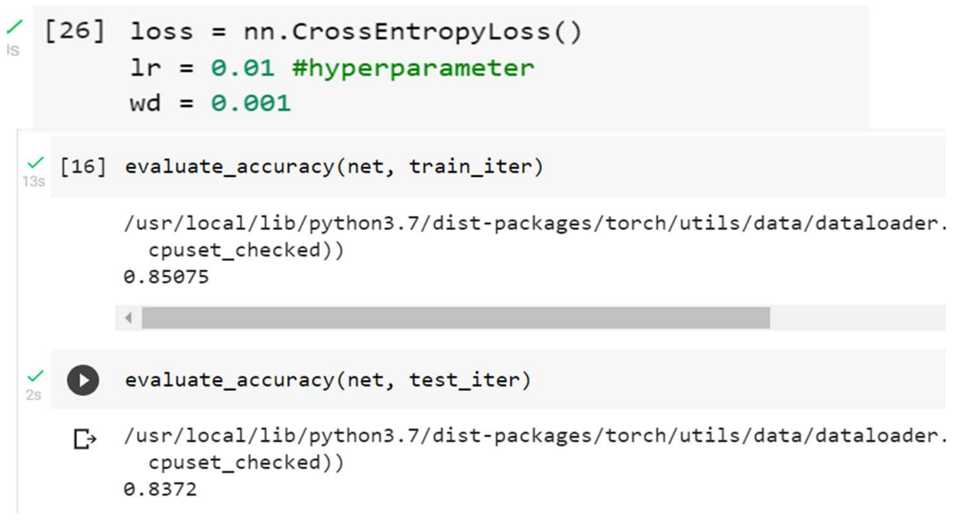
1. Batch size
   We chose a batch size of 256. The smaller the batch size the more updates of the gradient we get. Too large a batch size for example the full dataset would take far too long to update the gradient. Lower batch size and more Epochs means we go over more batches and update more often which brings us closer to the optimum weights and biases.

   After the first epoch we have gone through the entire dataset using the minibatches which allows us to more frequently update the parameters we initialised.

2. Epochs
   Each epoch is one run over the dataset. The more epochs we have, the more chances we have to update the models to optimal parameters after each batch. This allows better accuracy as we get closer to the global optimum.

   We chose 30 epochs to get closer to the optimal parameters. We may get to a point of convergence and we would then know we can stop iterating as the loss between the ground truth values and our predictions should be very minimal and close to 0. When we chose a low value of epochs such as 3, our model ran into problems such as low accuracy as it didn't have enough time to reach the global optimum.

3. Learning rate
   Lower learning rates ended up being vital. After tuning, we discovered learning rates over 0.05 could result in spurrious results. We chose a lower learning rate of 0.001 which yielded the best accuracy. 0.01 gave a decent test accuracy of 83.72%, however the steps in gradient descent were still too large. Lowering the training rate to 0.001 takes longer to train but it gave the best accuracy. If we lower the learning rate too low than it will take too long to run, we need a balanced the trade-off for accuracy is time.



```
[26] loss = nn.CrossEntropyLoss()
     lr = 0.01 #hyperparameter
     wd = 0.001
```

```
[16] evaluate_accuracy(net, train_iter)

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.
  cpuset_checked))
0.85075
```

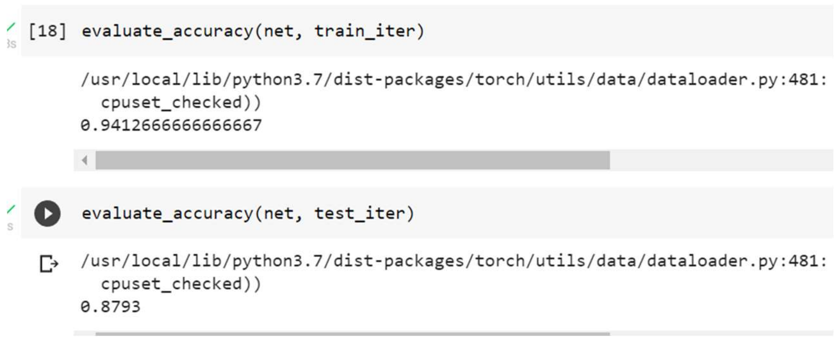
```
evaluate_accuracy(net, test_iter)

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.
  cpuset_checked))
0.8372
```

Task 5 – Final model accuracy

Too high training accuracy can mean overfitting. This means the model may not generalise well to unseen data as it has learnt the patterns of the training data too well and accounted for all the complex patterns. We would rather a model with 90% accuracy than 99% training accuracy as this means it accounts for noise and outliers and should be able to better generalise to unseen data.

Our model is shown to be effective, however slightly overfits as the training accuracy is 94% but the test accuracy is quite high at over 87%. However we could have gotten above 90% accuracy if the model generalised better. Overall our model is displayed to be an accurate predictor of images on the fashion MNIST dataset.



```
[18] evaluate_accuracy(net, train_iter)

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481:
  cpuset_checked))
0.9412666666666667
```

```
evaluate_accuracy(net, test_iter)

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481:
  cpuset_checked))
0.8793
```

References

Jupyter Notebook (Neural networks - Week06_Lab_with_solutions), ECS659P, 2022, Queen Mary University London, QMPLUS [Accessed April 8, 2022].

How to split tensors with overlap and then reconstruct the original tensor? PyTorch Forums. Available at: https://discuss.pytorch.org/t/how-to-split-tensors-with-overlap-and-then-reconstruct-the-original-tensor/70261/2 [Accessed April 8, 2022].