Project 3

QSR1:

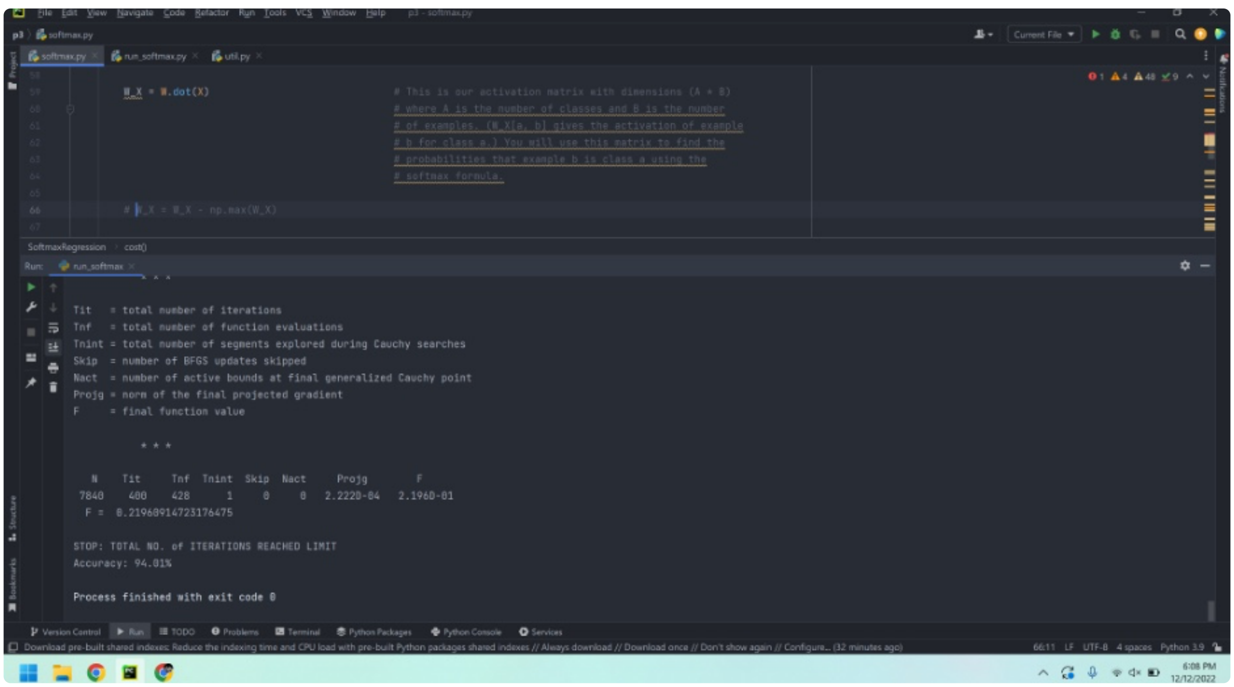
1. Simply by the rules of probability, the sum of the probabilities of different outcomes of an event must be 1. Additionally,

$$\Sigma P(y = i) = \Sigma \frac{e^{w \cdot x}}{\Sigma(e^{w \cdot x})} = \frac{\Sigma(e^{w \cdot x})}{\Sigma(e^{w \cdot x})} = 1$$

1. The dimension of W is n x c, where m is the number of classes that examples can be classified as. The dimension of X is e x n, where e represents the number of examples used. Therefore, the dimension of WX would be e x c, based on the rules of matrix multiplication.

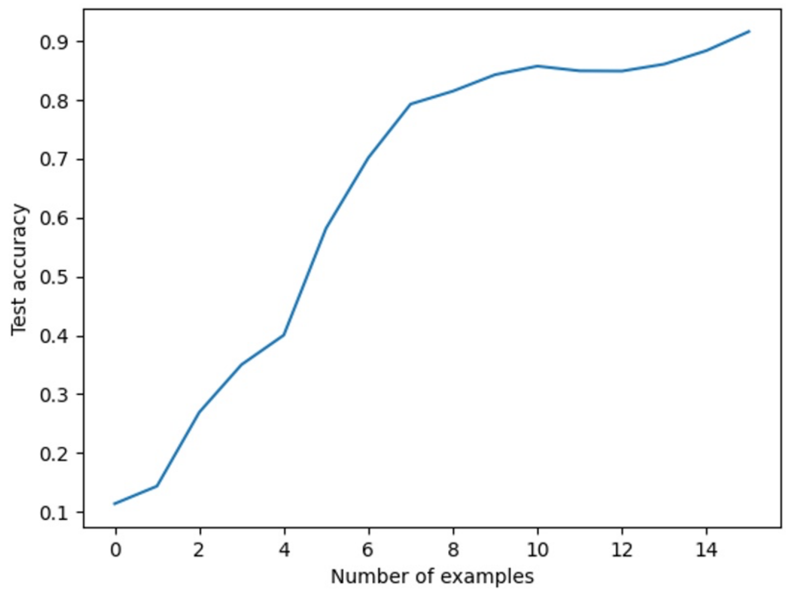
QSR3:

1. By commenting out the line removing the largest entry in the matrix and re-running run_softmax, it's clear that this action had no impact on the probabilities.



2. Since the algorithm uses a lot of exponents, such as $e^{\underline{w} \times \underline{x}}$, having large values used as the exponent could overflow the system and lead to inaccuracies or failure. Removing the largest value from W_X is a safety move to ensure that the numbers calculated aren't too large to handle and continue running the algorithm with.

QSR4:



You can see that the classifier begins to overfit the data a little much between 6-10 examples. This is why the accuracy dips down slightly at around 12 examples. Even though the accuracy decreased slightly, it ensured there wasn't overfitting of the data and eventually led to the highest accuracy observed at 14+ examples.

QNN1:

1. Done
2. The loss does go down as seen below:

```
>>> model, plot_dict = train_1pass(model, training_data, dev_data, learning_rate=1e-2, batch_size=64)
#Samples  6400  loss:0.48971    dev_acc:0.60320
#Samples 12800  loss:0.32823    dev_acc:0.71610
#Samples 19200  loss:0.28400    dev_acc:0.77330
#Samples 25600  loss:0.26458    dev_acc:0.80550
#Samples 32000  loss:0.23976    dev_acc:0.81870
#Samples 38400  loss:0.23340    dev_acc:0.83450
#Samples 44800  loss:0.22094    dev_acc:0.84730
#Samples 51200  loss:0.21095    dev_acc:0.85460
#Samples 57600  loss:0.20088    dev_acc:0.86130
>>> model, plot_dict = train_1pass(model, training_data, dev_data, learning_rate=1e-2, batch_size=64)
#Samples  6400  loss:0.19403    dev_acc:0.86920
#Samples 12800  loss:0.18780    dev_acc:0.87390
#Samples 19200  loss:0.18204    dev_acc:0.87980
#Samples 25600  loss:0.17451    dev_acc:0.88290
#Samples 32000  loss:0.16901    dev_acc:0.88380
#Samples 38400  loss:0.16882    dev_acc:0.88850
#Samples 44800  loss:0.16665    dev_acc:0.88980
#Samples 51200  loss:0.16189    dev_acc:0.89270
#Samples 57600  loss:0.15847    dev_acc:0.89340
>>> 
```

3. The final accuracy of the dev set was 94.71%

```
(base) jaydesmarais@Jays-MacBook-Pro Projects/p3 » python run_nn.py
             1 ↵
activation:Relu
loss function:SquaredLoss
Layer 1 w:(256, 784)    b:(256, 1)
Layer 2 w:(256, 256)    b:(256, 1)
Layer 3 w:(10, 256)     b:(10, 1)
Epoch   1/20    loss:0.21616    dev_acc:0.83710
Epoch   2/20    loss:0.16075    dev_acc:0.88060
Epoch   3/20    loss:0.13646    dev_acc:0.89820
Epoch   4/20    loss:0.15010    dev_acc:0.90810
Epoch   5/20    loss:0.14000    dev_acc:0.91540
Epoch   6/20    loss:0.12648    dev_acc:0.92080
Epoch   7/20    loss:0.14476    dev_acc:0.92460
Epoch   8/20    loss:0.09601    dev_acc:0.92810
Epoch   9/20    loss:0.10001    dev_acc:0.93070
Epoch  10/20    loss:0.13812    dev_acc:0.93280
Epoch  11/20    loss:0.09351    dev_acc:0.93390
Epoch  12/20    loss:0.12926    dev_acc:0.93670
Epoch  13/20    loss:0.09839    dev_acc:0.93850
Epoch  14/20    loss:0.11094    dev_acc:0.94130
Epoch  15/20    loss:0.10212    dev_acc:0.94290
Epoch  16/20    loss:0.09422    dev_acc:0.94450
Epoch  17/20    loss:0.08165    dev_acc:0.94460
Epoch  18/20    loss:0.08029    dev_acc:0.94600
Epoch  19/20    loss:0.09619    dev_acc:0.94640
Epoch  20/20    loss:0.12173    dev_acc:0.94710
```

4. It is good to initialize a weight matrix to small random numbers rather than 0s because neural networks are sensitive to initialization. When you use all the same number, 0s, to initialize the weight vector, the neurons will all compute the same output for a give input regardless of the weights. If you want the network to actually work to learn, it is best to have randomness other than 0 os the network can become more flexible.