Problem 1:

To derive the update rules, we must derive the 2nd layer update step and the 1st layer update step. The error function we're given is:

$$E(W, v|x) = -\sum_t r^t \log y^t + (1 - r^t) \log(1 - y^t) + \sum_h ||w_h||^2$$

To find the 2nd layer update, we must compute:

$$\frac{dE}{dv_h} = \frac{dE}{dy}\frac{dy}{dv_h}$$

Since the w terms are not a function of v, this is identical to taking the derivative of:

$$-\sum_t r^t \log y^t + (1 - r^t) \log(1 - y^t)$$

With respect to v_h. Then, much like in the sigmoid case, we get:

$$\frac{dE}{dy} = -\sum_{t=1}^{n} \frac{r^t}{y^t} - \frac{1 - r^t}{1 - y^t}$$

And since:

$$\frac{dy}{dv_h} = \frac{dy}{d(v^T z)}\frac{d(v^T z)}{dv_h} = (1 - y^2)z$$

We see that the update value should be:

$$\Delta v_h = \eta \sum_{t=1}^{n} \left(\frac{r^t}{y^t} - \frac{1 - r^t}{1 - y^t}\right)\left(1 - y^{t^2}\right) z^t$$

To find the 1st layers update, we must compute:

$$\frac{dE}{dw_{hj}}$$

However, here we should break the error equation into parts, because the derivative of the additional summation will not be 0 this time. Therefore, let's call the left summation in the error E' and the right summation in the error E''. Then:

$$\frac{dE}{dw_{hj}} = \frac{dE'}{dw_{hj}} + \frac{dE''}{dw_{hj}}$$

And since:

$$\sum_h ||w_h||^2 = \sum_h w_h^T w_h = \sum_h w_{h,1}^2 + w_{h,2}^2 + \cdots + w_{h,d+1}^2$$

We can conclude that:

$$\frac{dE''}{dw_{hj}} = 2w_{hj}$$

Now we compute the derivative of E'. By the chain rule:

$$\frac{dE'}{dw_{hj}} = \frac{dE'}{dy}\frac{dy}{dz_h}\frac{dz_h}{dw_{hj}}$$

Again:

$$\frac{dE}{dy} = -\sum_{t=1}^{n}\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t}$$

But now taking the derivative of y (which is the tanh function) with respect to z_h, we obtain:

$$\frac{dy}{dz_h} = (1-y^2)v_h$$

And now, we must be careful to note the value of the input to z, since we are using LReLU.

When $w_h^T x^t < 0$:

$$\frac{dz_h}{dw_{hj}} = 0.01x_j^t$$

Else:

$$\frac{dz_h}{dw_{hj}} = x_j^t$$

So when $w_h^T x^t < 0$:

$$\frac{dE}{dw_{hj}} = -0.01\sum_{t=1}^{n}\left(\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t}\right)\left(1-y^{t^2}\right)v_h x_j^t + 2w_{hj}$$

And so:

$$\Delta w_{hj} = \eta(-0.01\sum_{t=1}^{n}\left(\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t}\right)\left(1-y^{t^2}\right)v_h x_j^t + 2w_{hj})$$

And when $w_h^T x^t \geq 0$:

$$\frac{dE}{dw_{hj}} = -\sum_{t=1}^{n}\left(\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t}\right)\left(1-y^{t^2}\right)v_h x_j^t + 2w_{hj}$$

And so:

$$\Delta w_{hj} = \eta(-\sum_{t=1}^{n}\left(\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t}\right)\left(1-y^{t^2}\right)v_h x_j^t + 2w_{hj})$$

Thus we actually get two different update rules for w, depending on the value of $w_h^T x^t$.

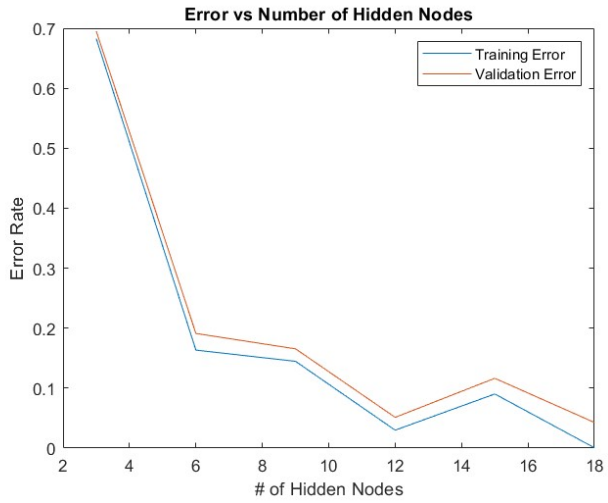
Problem 2:

a. Below is the text output of the script for problem 2a. Since this script runs mlptrain for various hidden layer values, and since each run of mlptrain prints training and validation error, we can compare the errors for varying numbers of hidden nodes.

```
3 Hidden Nodes:
Train error rate:
    0.6829


Valid error rate:
    0.6951


6 Hidden Nodes:
Train error rate:
    0.1634


Valid error rate:
    0.1911


9 Hidden Nodes:
Train error rate:
    0.1447


Valid error rate:
    0.1655


12 Hidden Nodes:
Train error rate:
    0.0299


Valid error rate:
    0.0513


15 Hidden Nodes:
```

```
Train error rate:

    0.0902


Valid error rate:

    0.1164


18 Hidden Nodes:

Train error rate:

   5.3390e-04


Valid error rate:

    0.0427
```

We can see that as the number of hidden nodes increases, the errors tend to decrease. This trend is clearly visible in the graph generated by the script:

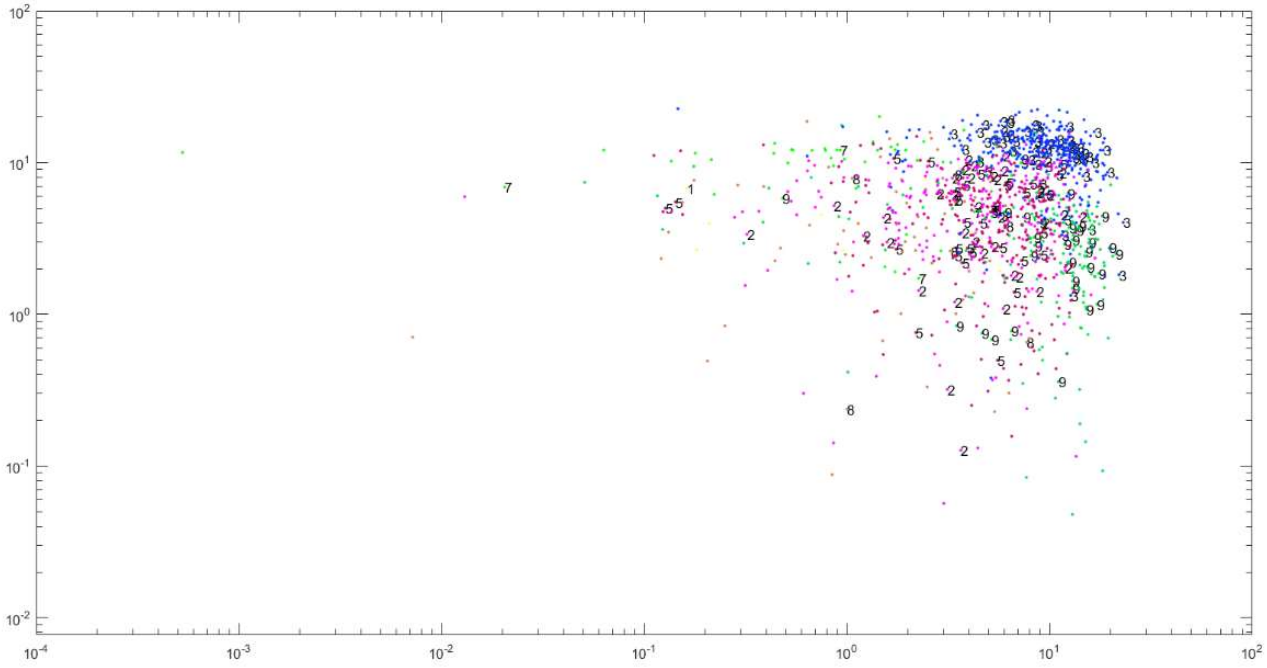


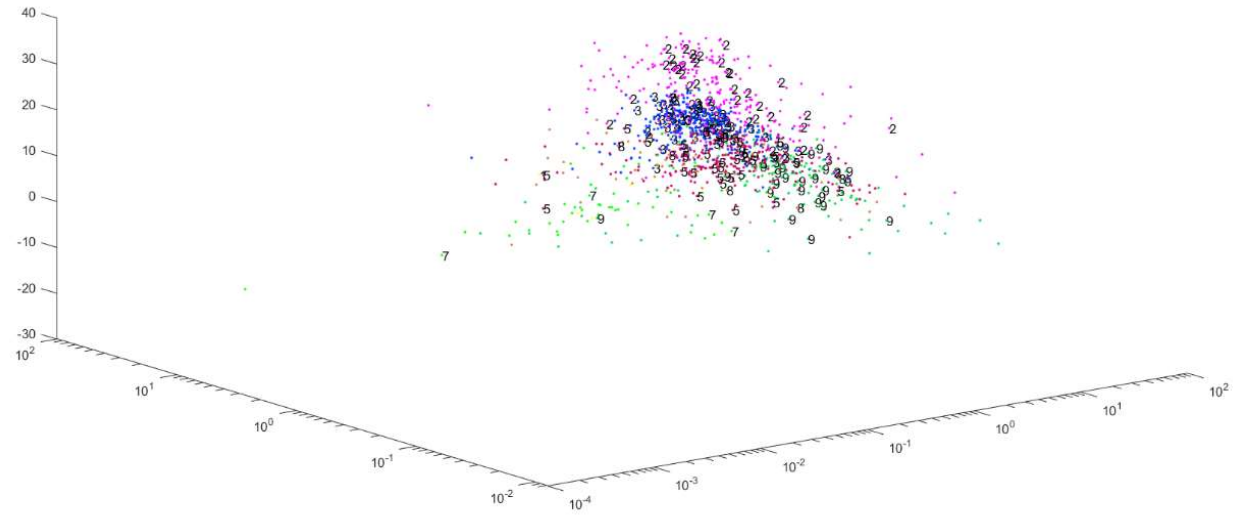
When deciding the best number of hidden nodes, what we're looking for is the lowest validation error, since this is more representative of actual data we'll encounter. From the data we see here, it's clear that 18 hidden nodes produces the best validation error rate (0.0427), so this is the number chosen for part b.

b. For this problem, the network was trained with 18 hidden nodes, since this was determined to be the optimal number from part a. The training and validation sets were combined together to produce a z matrix, which was then projected onto both two and three dimensions using PCA. The error rate of the combined set is printed to the MATLAB terminal, as well as the error rate for the test set:

```
Combined error rate: 0.036572
Test Error Rate: 0.059232
```

The logarithm of the x and y dimensions were taken for both the 2D and 3D plot (as instructed), and points were color coded according to their class. Finally, text labels were added to some of the data to help visualize how digits are distributed (not every point was labeled, since doing so would create crowding, making it difficult to see anything). Below are the resulting 2D and 3D plots:





We can see that even with only two and three dimensions respectively, there are some pretty dense clusters and points appear to be well separated. This is indicative of a well-trained network, which is likely to lead to high accuracy rates.

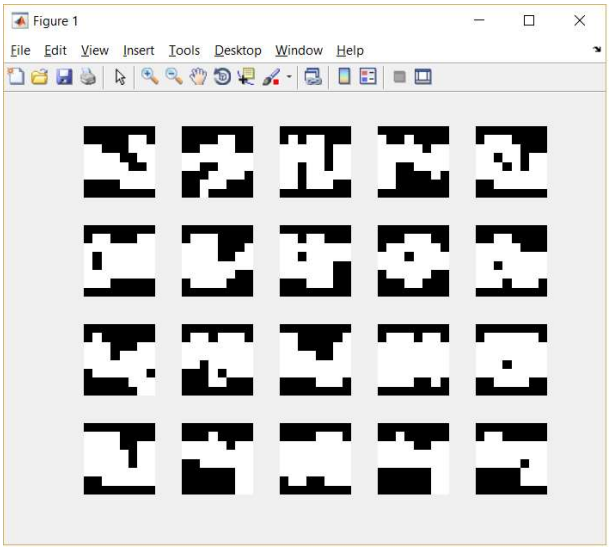
NOTE: Included in my code is a function called mlptest as instructed. However, there was never any explicit instruction on how this function should be used! To compute error, I use my own function called computeMLPError, which is more useful than mlptest because it returns both a z matrix AND error rate, which can then be printed outside of the function rather than inside. In fact, my implementation of mlptest just uses computeMLPError, which is a more general version of mlptest. Mlptest is called only one time, and that's in the problem2b script (to test the accuracy on the network trained with the optimal number of nodes).

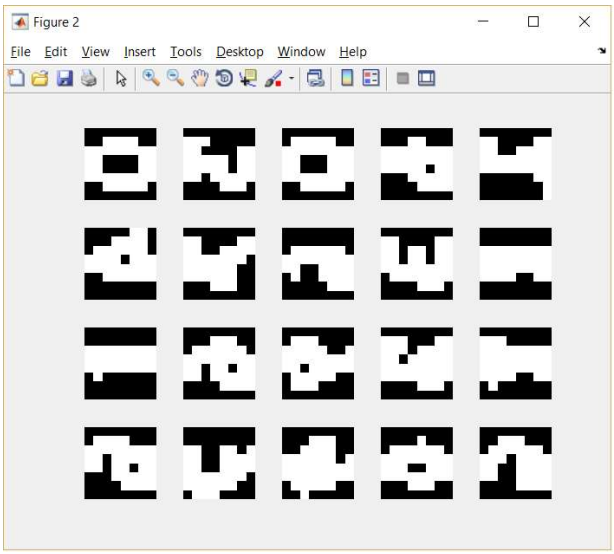
Problem 3:

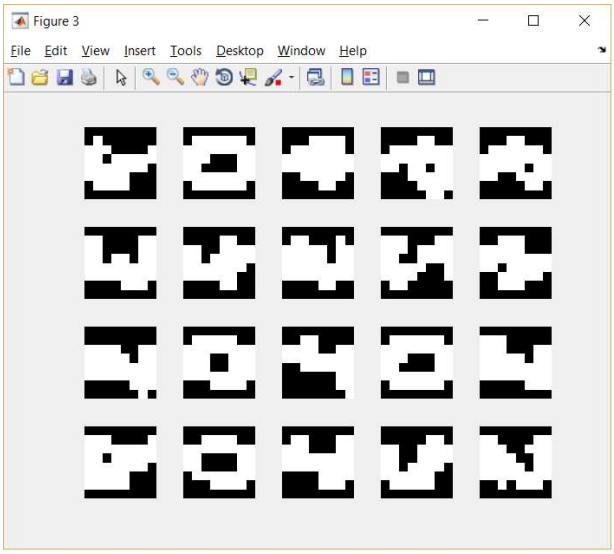
i. Since we are only asked to provide the main file for part 2 of this question, I've decided to include the MATLAB code that defines the layer for part 1 here:

```
layers = [ ...
    imageInputLayer([8 8 1])
    convolution2dLayer(4,1)
    batchNormalizationLayer
    myLReLULayer
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

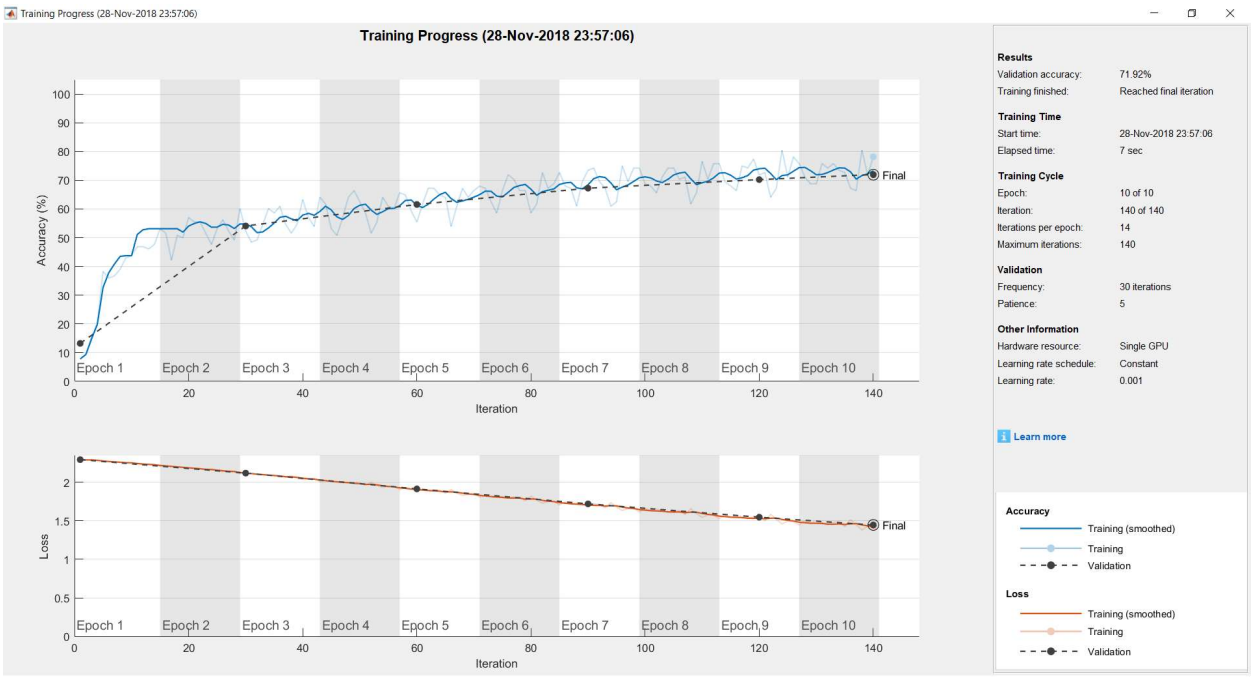
The following images were generated:







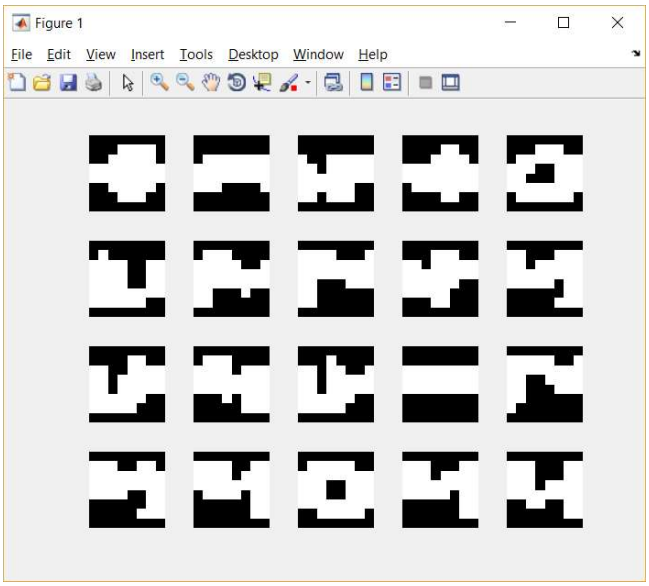
And the code also generated this training plot:

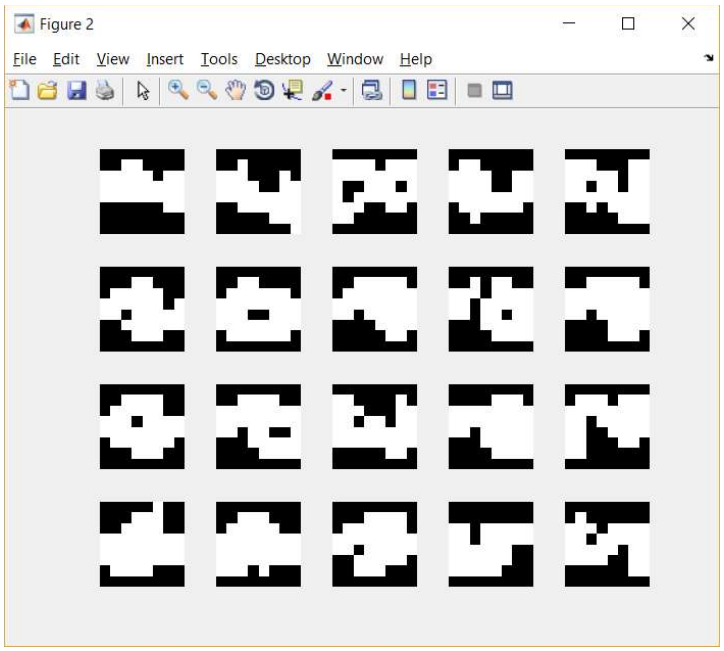


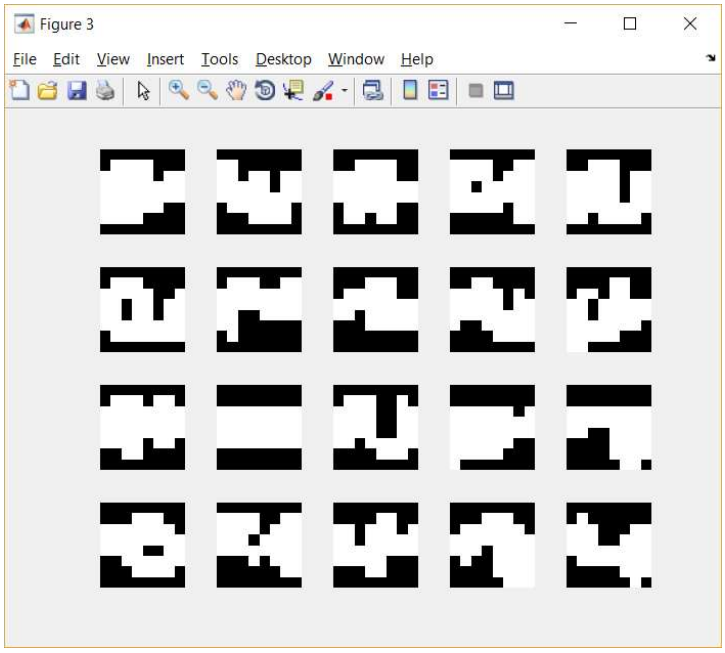
The test accuracy is outputted to the MATLAB terminal as follows:

```
accuracy =

    0.7444
```

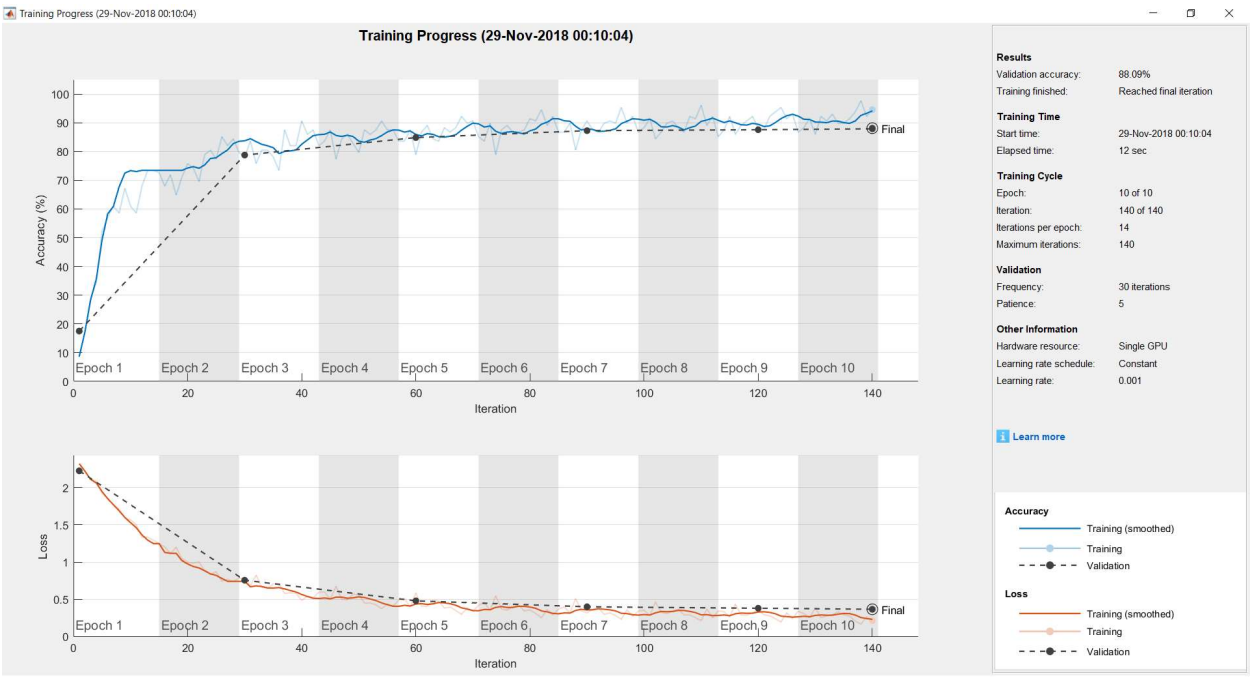
ii. For part 2, the layers array is included in the main file, so I have chosen to omit this from the report. The following images were generated by the code:







Additionally, the following progress report was generated:



The test accuracy is outputted to the MATLAB terminal as follows:

```
accuracy =

    0.8906
```

While an 89.06% classification accuracy is substantially better than the accuracy achieved by the network in part one of this question, this is still not very impressive since it actually performs worse than the MLP I built for problem 2. One good thing about both the models from part 1 and part 2 of this question is that neither seems to suffer from overfitting, because both of them actually perform better on test data than on the training set.