Machine Learning Practical (MLP)
Coursework 2
Georgios Pligoropoulos - s1687568

MNIST Digit Classification Problem

Our goal is to classify digits, from the MNIST dataset, as accurately as possible using a classic neural network which takes as input the flat representation of all the 784 (28x28) pixels of each grayscale image.

Grand Experiment 1: Exploring the optimal system architecture in numbers of layers and number of neurons of each layer

Why choosing this experiment and what we are trying to achieve

We are **uncertain of how much complexity is required** of the model of our neural network and thus we want to find the sweet spot between a very simple or a very complex neural network.
We will explore where the neural network breaks/fails for too few layers or too few features and where for too many of them.
In other words **we variate the dimensionality** and also **we variate the number of neurons**.

Note that in this grand experiment **we are are not optimizing for accuracy** neither optimizing for performance. Rather we want to see how the complexity of the model generally affects the accuracy and performance on a dataset as complex as MNIST. In other words we care for comparison rather than absolute values.

Constants in our Neural Network Architecture

We are using the **Gradient Descent Learning Rule with constant learning rate of 1.1** because it performed relatively well, with good accuracy and performance without making the system too unstable.
Note that we are using a learning rate that produced good enough results within 20 epochs for the three layer architecture of 100 dimensionality each in the previous coursework.

$$\Delta wi(t) = -\eta Di(t)$$

$$wi(t) = wi(t-1) + \Delta wi(t)$$

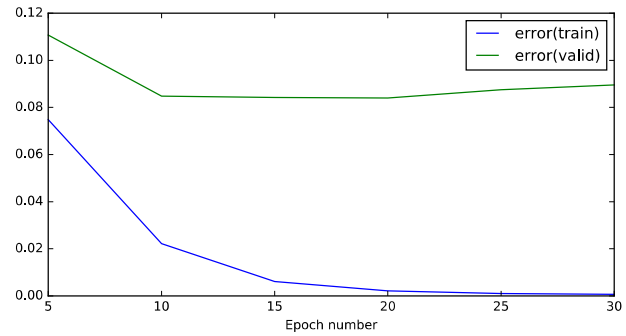
We are **not going to use regularization** rather we will do early stopping if we see that the validation accuracy and validation error have reached optimum levels.

Here we are using **glorot initialization**, a special parameter initialisation scheme for the weights which makes the scale of the random initialisation dependent on the input and output dimensions of the layer, with the aim of trying to keep the scale of activations at different layers of the network the same at initialisation.
We do this because we do not want the weights to get too big too soon and saturate the network.
We also initialise the **biases to zero** as this is not going to affect the gradients.

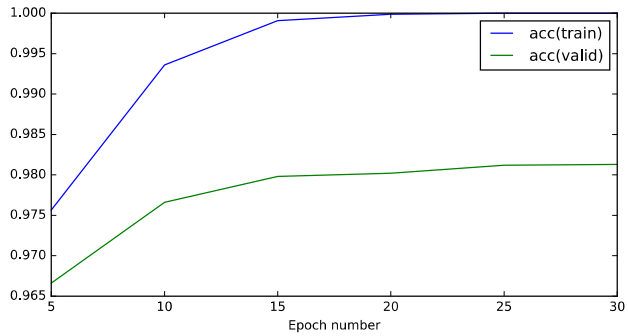
For all experiments **we use a batch size of 50** and we use **affine transformations** which are interleaved with **logistic sigmoid** nonlinearities, and at the end we always have a **softmax output layer**.

Note: When in our explanations refer to **accuracy** we mean the final validation accuracy and when we refer to **performance** we mean how fast we reached the optimal accuracy for the current experiment.

Baseline: Three Layers and dimensionality of 100 for each hidden layer



Graph 1: Training & Validation Error - Three Hidden Layers - 100 dimensionality for each hidden layer



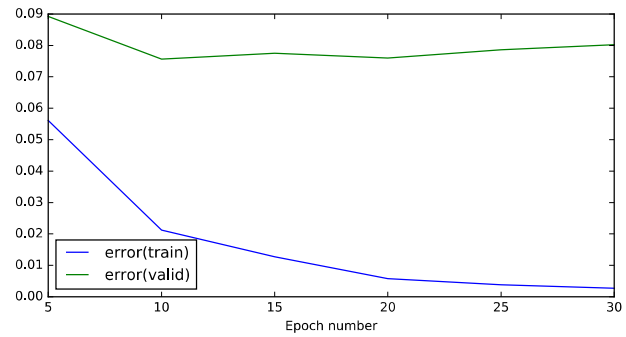
Graph 2: Training & Validation Accuracy - Three hidden layers - 100 dimensionality for each hidden layer

Runtime: `43.7089149952 seconds`

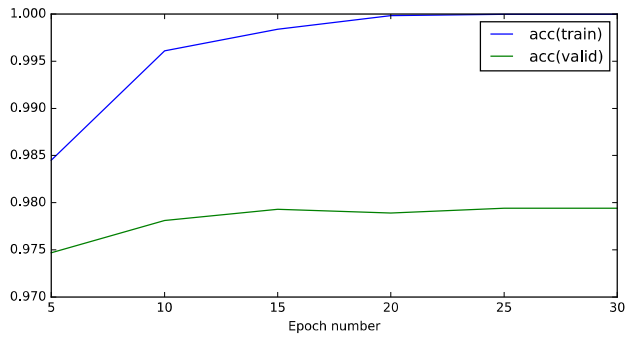
`Final Testing Accuracy (percentage 0 to 1)`	`0.98129999999999906`
`Final Testing Error`	`0.089556102164988771`
`Final Training Accuracy (percentage 0 to 1)`	`1.0`
`Final Training Error`	`0.00070516598919551218`

This is our base reference from first coursework. Within 20 epochs we overfit but as we saw in previous coursework even for 100 epochs the performance does not get significantly better.

Two Layers and dimensionality of 100 for each hidden layer



Graph 3: Training & Validation Error - Two Hidden Layers - Dimensionality 100 for each hidden layer

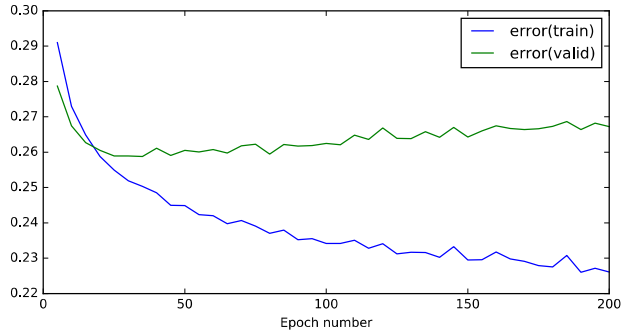


Graph 4: Training & Validation Accuracy - Two Hidden Layers - Dimensionality 100 for each hidden layer

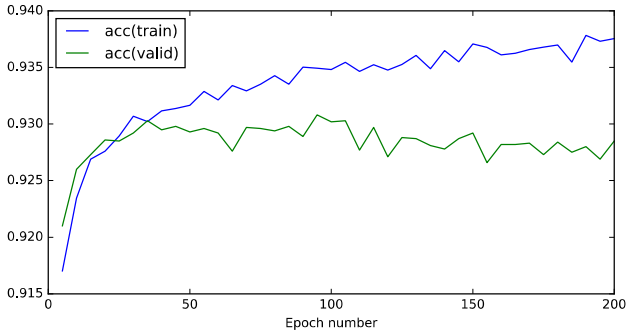
`Final Testing Accuracy (percentage 0 to 1)`	`0.97939999999999872`
`Final Testing Error`	`0.08024213423911615`
`Final Training Accuracy (percentage 0 to 1)`	`0.99997999999999998`
`Final Training Error`	`0.0027497674489311871`

For two layers the accuracy and the performance does not seem so bad. The accuracy has slightly dropped from ~98.1% to 97.9%. The best accuracy is achieved within 15 epochs. Meaning that we could consider keeping this much faster implementation.

One Layer without hidden layers



Graph 5: Training & Validation Error - One Layer



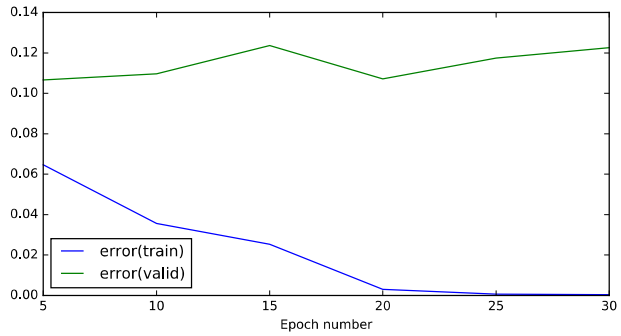
Graph 6: Training & Validation Accuracy - One Layer

Final Testing Accuracy (percentage 0 to 1)	0.92849999999999921
Final Testing Error	0.26725893060469291
Final Training Accuracy (percentage 0 to 1)	0.9375400000000047
Final Training Error	0.22611458848848595

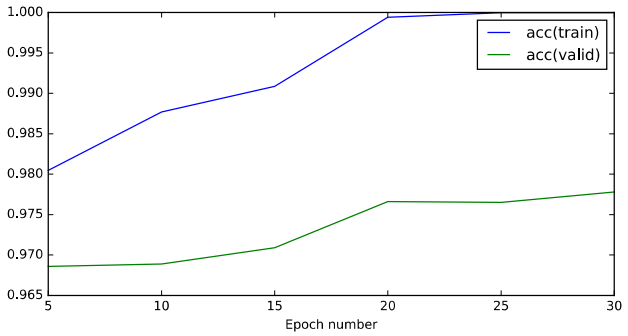
The single layer network is not possible to capture the complexity of the model, even after 200 epochs the training accuracy does not seem to be able to grow larger than 93.7% and the validation error is increasing. Moreover it is apparent that the validation accuracy has even start to decrease. The best validation accuracy achieved is ~93% which is much lower than before. We will consider this a failed case.

Let's see if adding the layers to go from 3 layers to 4 layers will have a positive impact or not or it will have a neutral effect.

Four Layers and dimensionality of 100 for each hidden layer



Graph 7: Training & Validation Error - Four Hidden Layers - dimensionality 100 for each hidden layer



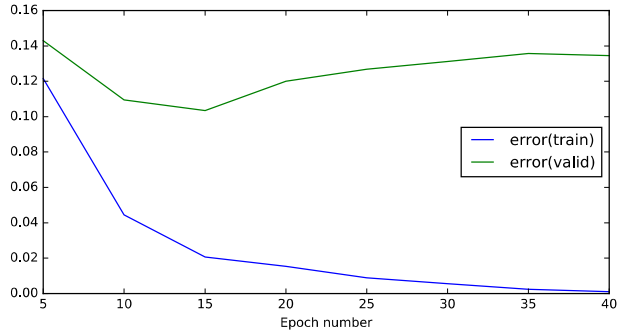
Graph 8: Training & Validation Accuracy - Four Hidden Layers - dimensionality 100 for each hidden layer

Final Testing Accuracy (percentage 0 to 1)	0.97779999999999878
Final Testing Error	0.12262454645853889
Final Training Accuracy (percentage 0 to 1)	0.99997999999999998
Final Training Error	0.00039507227075968177

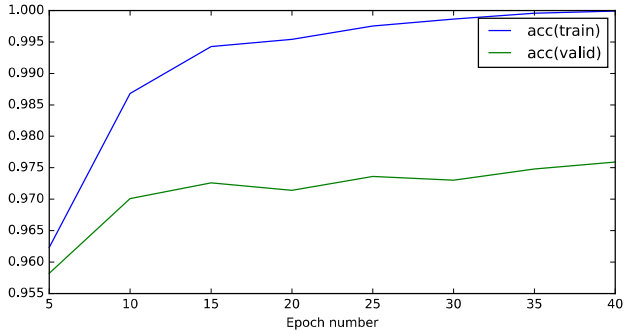
By adding one more layer we do not seem to have achieved anything significant. We are actually overfitting again since the beginning because the validation error is constantly increasing. We have not done anything better neither

in terms of performance or accuracy.

Five Layers and dimensionality of 100 for each hidden layer



Graph 9: Training & Validation Error - Five Hidden Layers - dimensionality 100 for each hidden layer

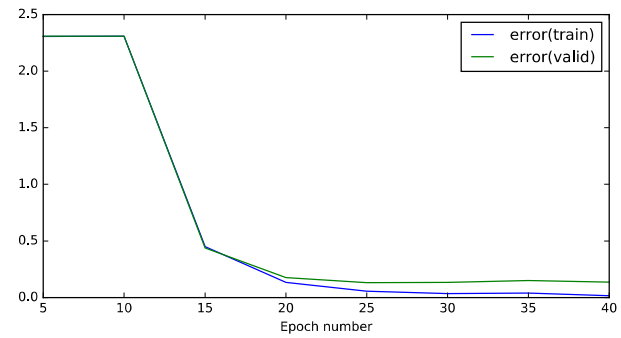


Graph 10: Training & Validation Accuracy - Five Hidden Layers - dimensionality 100 for each hidden layer

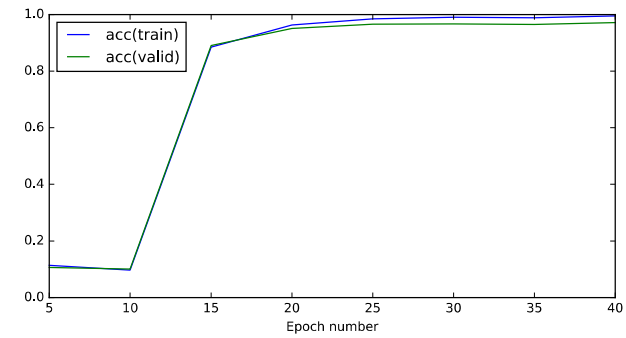
`Final Testing Accuracy (percentage 0 to 1)`	`0.97589999999999943`
`Final Testing Error`	`0.13445038749109026`
`Final Training Accuracy (percentage 0 to 1)`	`0.99990000000000012`
`Final Training Error`	`0.00098406888507099766`

For five layers again slower speed of algorithm and the accuracy has not improved. The validation error starts increasing after 15 epochs as well. We have done worse overall.

Six Layers and dimensionality of 100 for each hidden layer



Graph 11: Training & Validation Error - Six Hidden Layers - dimensionality 100 for each hidden layer



Graph 12: Training & Validation Accuracy - Six Hidden Layers - dimensionality 100 for each hidden layer

`Final Testing Accuracy (percentage 0 to 1)`	`0.97219999999999929`
`Final Testing Error`	`0.13549318830813875`
`Final Training Accuracy (percentage 0 to 1)`	`0.99558000000000246`
`Final Training Error`	`0.016967568263340702`

And finally with six layers we see a different picture. The neural network has a hard time to adjust all the weights to achieve something useful at the first ten epochs. The model is slowly learning as the weights are adjusting trying to find a balance. Afterwards we see that the accuracy is at the same terms as before. This means that even though we have all this extra complexity to our model of 6 layers we are not taking advantage any of them. It seems that some

features are just being replicated from layer to layer and the model has already found its minimum on the third layer.

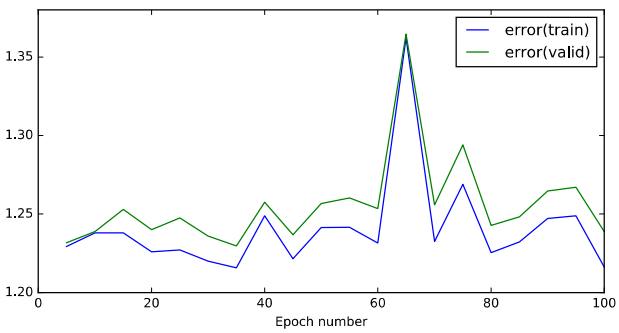
Comparing Layers with dimensionality of 100 for each hidden layer

	Final Testing Accuracy	Final Testing Error	Final Training Accuracy	Final Training Error
One Layer	0.92849999999999	0.26725893060469	0.93754000000000	0.22611458848848
Two Layers	0.97939999999999	0.08024213423911	0.99997999999999	0.00274976744893
Three Layers	0.98129999999999	0.08955610216498	1.000000000000000	0.00070516598919
Four Layers	0.97779999999999	0.12262454645853	0.99997999999999	0.00039507227075
Five Layers	0.97589999999999	0.13445038749109	0.99990000000000	0.00098406888507
Six Layers	0.97219999999999	0.13549318830813	0.99558000000000	0.01696756826334

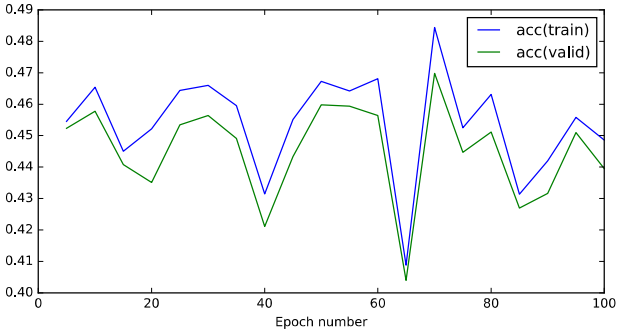
Taking into account both our plots above, where the behavior of the neural network can be displayed, and this comparison table we conclude that adding more than three layers have helped nothing else but slowing the execution time. Even the difference between the two layers and three layers does not seem that significant.

Let's go back to our experiment with 2 layers that was fast enough and with good enough accuracy and let's try to play with the dimensionality to see what we can achieve with less or more dimensionality on the hidden layer.

Two Layers and dimensionality of 2 for the hidden layer



Graph 13: Training & Validation Error - Two Layers - dimensionality of 2 for the hidden layer

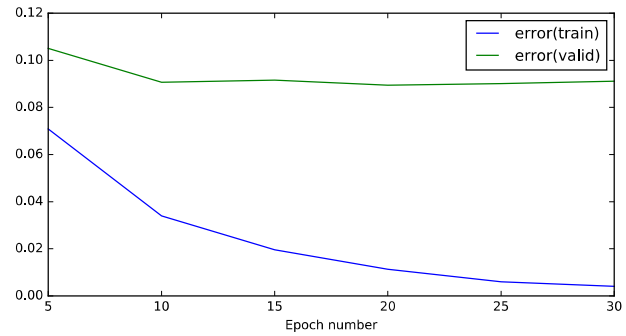


Graph 14: Training & Validation Accuracy - Two Layers - dimensionality of 2 for the hidden layer

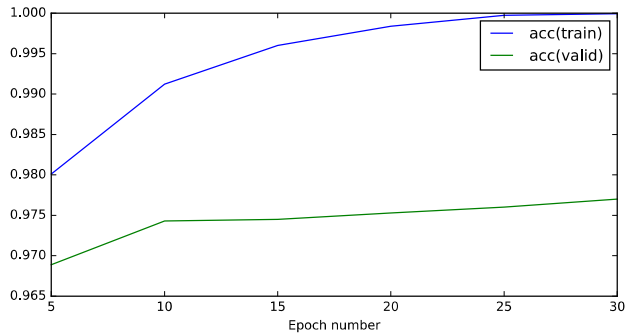
Final Testing Accuracy (percentage 0 to 1)	0.43949999999999995
Final Testing Error	1.238846999618503
Final Training Accuracy (percentage 0 to 1)	0.44859999999999839
Final Training Error	1.2162632009070424

Obviously when the features were reduced to only 2 features it would be silly of us to believe that only two features could represent 10 digits from 0 to 9. We executed this experiment to verify our intuition. And the numbers agree with us with a final validation accuracy of ~43%

Two Layers and dimensionality of 1000 for the hidden layer



Graph 15: Training & Validation Error - Two Layers - dimensionality of 1000 for the hidden layer



Graph 16: Training & Validation Accuracy - Two Layers - dimensionality of 1000 for the hidden layer

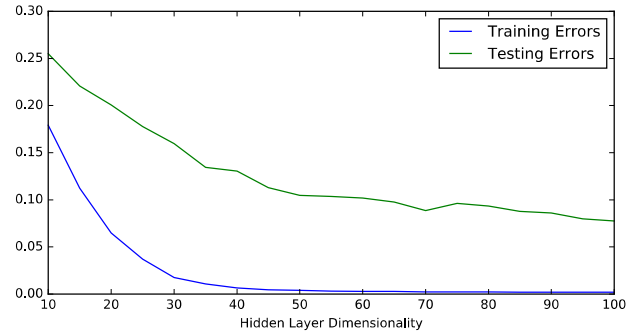
Final Testing Accuracy (percentage 0 to 1)	0.97699999999999931
Final Testing Error	0.09109709503057789
Final Training Accuracy (percentage 0 to 1)	0.99994000000000005
Final Training Error	0.0040190342463076881

When we increased the dimensionality to 1000 we got an accuracy of ~97.7% which means that we didn't really do any better than the dimensionality of 100 and we overfitted at ~25 epochs. The system also became extremely slow because of the number of connections. Therefore in this experiment achieved nothing special.

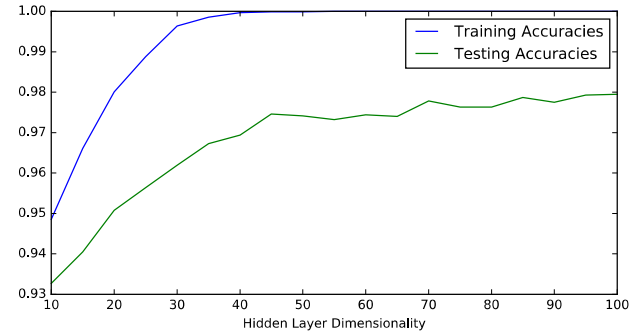
Next we are going to do a grid search to try and find the best dimensionality for a neural network of two layers.

Two Layers and dimensionality of 10 to 100 for the hidden layer with steps of 5

We are executing the experiment from 10 to 100 neurons for the hidden layer of a two layer network to see how the final validation accuracy behaves. Each experiment runs for 40 epochs because 40 epochs for a learning rate of 1.1 is enough to get an estimate if the neural network is capable to be trained or not.



Graph 17: Final Training and Validation Error for Two Layers by varying the dimensionality from 10 to 100 with steps of 5



Graph 18: Final Training and Validation Accuracy for Two Layers by varying the dimensionality from 10 to 100 with steps of 5

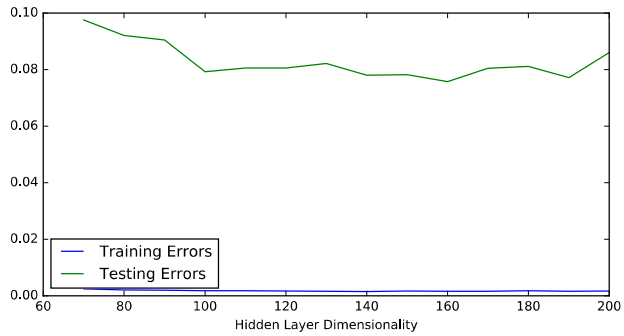
From the above two plots we can see how the model becomes more capable of capturing the complexity of the input when the dimensionality of the hidden layer is increasing

By following the exact same procedure it worths to try and see if for larger dimensionalities, again for one

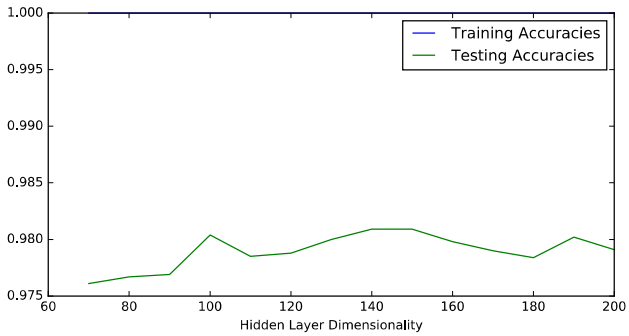
hidden layer the accuracy is increased.

Two Layers and dimensionality of 70 to 200 for the hidden layer with steps of 10

We are repeating the previous experiment but now we are ranging from 70 to 200 neurons for the hidden layer with steps of 10 mainly to see how the final validation accuracy behaves. Again we are using 40 epochs for each experiment.



Graph 19: Final Training and Validation Error for Two Layers by varying the dimensionality from 70 to 200 with steps of 10

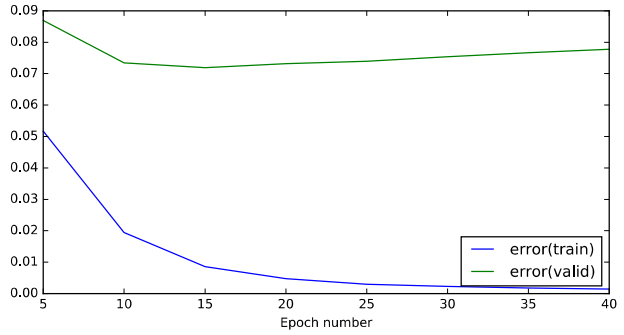


Graph 20: Final Training and Validation Accuracy for Two Layers by varying the dimensionality from 70 to 200 with steps of 10

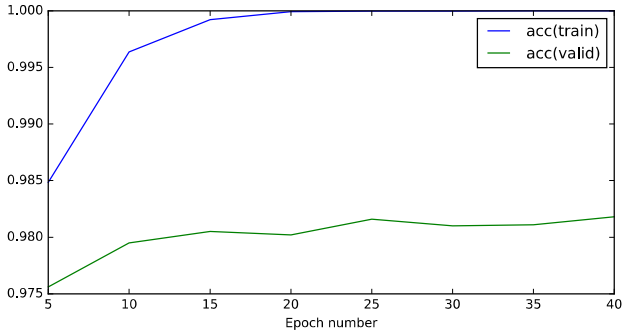
We see that for larger dimensionality we have a better capture of the complexity of the model. We peaked at hidden dimensionality = 190 and 140 has almost equal performance.

We are going to repeat that experiment for this particular dimensionality of 190 to compare its speed with the three layers when the dimensionality was 100 for each layer.

Two Layers and dimensionality of 190 for the hidden layer



Graph 21: Training & Validation Error - Two Layers - dimensionality of 190 for the hidden layer



Graph 22: Training & Validation Accuracy - Two Layers - dimensionality of 190 for the hidden layer

Runtime: `81.0337300301 seconds`

`Final Testing Accuracy (percentage 0 to 1)`	`0.98179999999999878`
`Final Testing Error`	`0.077739857983879238`
`Final Training Accuracy (percentage 0 to 1)`	`1.0`
`Final Training Error`	`0.0014331900425222032`

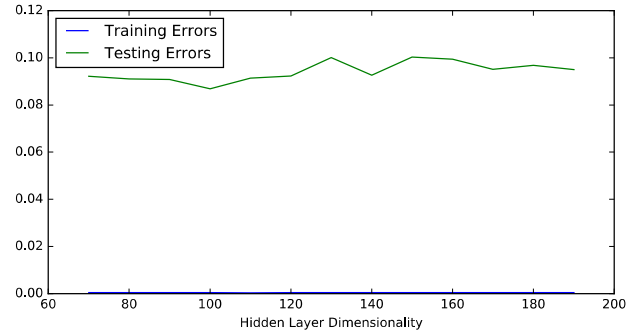
We got an optimal training accuracy of approximately ~98.1% within 25 epochs. However the 40 epochs run took in

total ~81 seconds which is almost half the speed of the calculation in comparison to the 100x100 two hidden layers (three layers in total) which took only ~43 seconds for a total of 100 epochs.

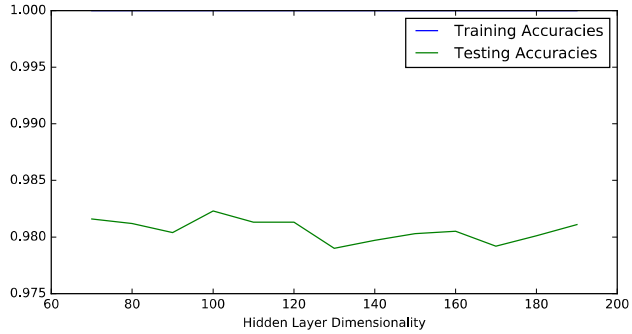
We will do one final experiment with two hidden layers where the dimensionality of the first hidden layer is going to be 140 which had a good enough accuracy, only slightly smaller than the experiment with 190 neurons. This is because it seems that too many neurons are required to get a good performance for the two layer model. We might get better accuracy if we bring in more levels in the neural network to get more levels of abstractions and since we had seen that three levels was working better than two levels in the experiments above.

Then for the second hidden layer we are going to do a grid search for the dimesionality within the range of 70 to 190. But this time we are going to keep track both of the accuracy/error and the runtime because apart of the accuracy we also care for the speed on which we can perform the experiments. Because the faster we can iterate through our experiments the faster we are going to be able to reach to meaningful conclusions.

Three Layers with 140 dimensionality of first hidden layer and a range of 70 to 200 for the second hidden layer with step of 10



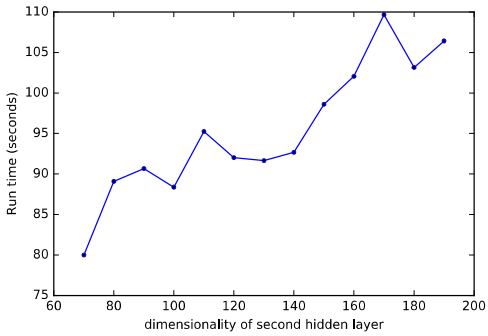
Graph 23: Final Training and Validation Error for Three Layers with first hidden layer of dimensionality of 140 and by varying the dimensionality of the second hidden layer from 70 to 200 with steps of 10



Graph 24: Final Training and Validation Accuracy for Three Layers with first hidden layer of dimensionality of 140 and by varying the dimensionality of the second hidden layer from 70 to 200 with steps of 10

We see that we have achieved the best accuracy for 100 dimensions in the second hidden layer.

Let's see how is performance, in terms of running time, for each case.



Graph 25: Runtime for Three Layers with first hidden layer of dimensionality of 140 and by varying the dimensionality of the second hidden layer from 70 to 200 with steps of 10

Comparing Metrics of best of Two Layers with best of Three Layers

Dimensionality of hidden layers	**190 (two layers)**	**140x100 (three layers)**
Runtime (seconds) for 40 epochs	81.0337300301	88.37469506263733
Final Testing Accuracy (percentage 0 to 1)	0.98179999999999878	0.98229999999999873
Final Testing Error	0.077739857983879238	0.08683500828069636
Final Training Accuracy (percentage 0 to 1)	1.0	1.0
Final Training Error	0.0014331900425222032	0.00037837813292187177

We have found out that an architecture of three layers has not provided any significant advantage in comparison with the two layers.

Training error for three layers is smaller than training error for two layers which means that the three layer model captures better the complexity of the training data.

However the performance of the accuracy has only increased ~0.1% from ~98.18% to ~98.23%

By combining two hidden layers we achieved similar results with the two layers model (only one hidden layer). More particularly the final validation accuracy for three layers is 98.12% which is very similar to 98.17% of the two layers with 190 neurons.

However the performance of the neural network with three layers is worse than our most-computationally-expesive case of the two layers of 190 neurons. We expect the two layers neural network to be faster for 140 or 150 neurons where the validation accuracy was comparable to the 190 neurons.

Conclusions

It depends of whether you are looking for maximum performance or you are looking for maximum accuracy.

For the MNIST problem you may choose 2 or 3 layers (one or two hidden layers respectively)

When choosing 2 layers network the best range is between 100 to 150 neurons and again the criteria of accuracy vs performance is going to determine the exact number of neurons.

On the other hand when choosing 3 layers and you have optimized the first hidden layer to have 140-150 neurons then you need to pick between 100-120 neurons for the second hidden layer to have an extra advantage on terms of final error and accuracy.