Alexander Freed
CS 545 Machine Learning
Professor Melanie Mitchell
Homework 2 – Neural Network Digit Classifier
1-29-19

The program for this assignment is heavily based on my program for the perceptron assignment. There are relatively few changes.

Setup, Build, and Run

Linux

Put *NeuralNet.zip* in an empty directory then enter these commands.

1. `unzip NeuralNet.zip`
2. `mkdir release`
3. `cd release`
4. `cmake .. -DCMAKE_BUILD_TYPE=Release`
5. `make`
6. `./NeuralNet "../data"`

Alternative Setup With No CMake

Put *NeuralNet.zip* in an empty directory then enter these commands.

1. `unzip NeuralNet.zip`
2. `g++ src/*.cpp -std=c++14 -I eigen/ -o NeuralNet -O2 –march=native`
3. `./NeuralNet "data"`

You can build in Windows with CMake and Visual Studio the same as in Linux. The major difference is that you set Release mode through Visual Studio. I think the 32-bit version should have enough memory, but just in case, in Windows you can also specify a 64-bit build, e.g.

`cmake .. -G "Visual Studio 15 Win64"`

This program is written in ISO C++14.

Usage:

```
./NeuralNet [dataPath] [numEpochs] [numHidden] [learningRate] [momentum] [defaultSeed] [writePlotData]
```

- `dataPath` – Path to data file directory. Type: string. Default: `"../../data/"`
- `numEpochs` – Number of epochs. Type: unsigned. Range: >0. Default: 50
- `numHidden` – Number of nodes in the hidden layer. Type: unsigned. Range: >0. Default: 20
- `learningRate` – The learning rate. Type: double. Range: >0. Default: 0.1
- `momentum` – Coefficient of previous weight change. Range: [0, ~0.97]. Default: 0.9
- `defaultSeed` – Helps with reproducibility when debugging. 1: use default seed. 0: use clock. Default: 0
- `writePlotData` – Write plot data to file "plotdata.csv". 0: don't write. 1: write. Default: 0

Eigen
Just like the first assignment, this program uses a C++ header-only library called Eigen to do optimized vector and matrix operations. Eigen is open source and licensed under MPL2.
Note: I couldn't include some of the Eigen package because Gmail wouldn't let me attach certain files.

Folder Layout

- data/
  - the *mnist_test.csv* and *mnist_train.csv* datasets.
- eigen/
  - The Eigen source
- python/
  - *plot.py* for plotting accuracy and *splitdata.py* for shortening the datasets
- src/
  - My Neural Net program source code

Class Descriptions

- `NUM_INPUTS` = 785 (defined in *Trainer.h*)
- `NUM_OUTPUTS` = 10 (static member of class `NerualNetDigitClassifier`)
- `InputType` is a typedef for a dynamically sized row-wise vector (defined in *Trainer.h*)
- `OutputType` is a typedef for a matrix with 1 row and `NUM_OUTPUTS` columns
  - Technically a row-wise vector, but was made a matrix to make certain function calls easier. (located in class `NerualNetDigitClassifier`)
- `WeightsType` is a typedef for a dynamic matrix (located in class `NerualNetDigitClassifier`)
- `WeightsCollection` is a typedef for an array of `WeightsType`, size 2 (located in class `NerualNetDigitClassifier`)

`RawTrainer` and `Trainer` are the same as in the first homework. Class `RawTrainer` is a "plain old data" ("POD") struct that holds 785 inputs (as an array) and a correct answer ("target"). The first input is the bias input and is always set to 1. `RawTrainer` is used for fast serializing/deserializing. Class `Trainer` also holds 785 inputs and a target, but the inputs are in the form of `InputType` which is usable by the program.

Class `NeuralNetDigitClassifer` has a few members.

- `m_numHidden` is the number of nodes in the hidden layer. This can only be set at construction.
- `m_weights` is of type `WeightsCollection`, that is a size-2 array of dynamically sized matrices. The first element is a matrix with 785 rows and `m_numHidden` columns. The second element has `m_numHidden` rows and 10 columns. These are the weights from input->hidden and hidden->output. Every element is initialized randomly
- `m_dWeightsPrev` is the same type as `m_weights`—a size-2 array of matrices with the same shape as `m_weights`. These hold the previous weight delta for use in calculating the momentum. Every element is initialized to 0.

The class also has some member functions for training. The main ones are `TrainFromInput` and `DetermineDigit`.

Training is sequenced by a function called `train` located in *main.cpp*.

Neural Network Setup
There are 784 + 1 inputs just like the perceptron assignment. There is one hidden layer with N neurons (N can be set at run-time). The output layer has 10 neurons. The output with the highest activation is selected as the guessed answer.

The weights are represented as matrices. The weights for the input-to-hidden layers are a 785xN matrix. The weights for the hidden-to-output layers are a (N+1)x10 matrix. The +1 row is for the bias of the hidden-to-output activation, and is always set to 1. The weights are initialized randomly (uniform) in the range [-0.05, 0.05] inclusive. Training is done using back-propagation in stochastic gradient descent with a momentum factor. The training set was shuffled randomly at the beginning of every epoch.

Experiment 1
Vary number of hidden units

For this experiment, learning rate was fixed at 0.1, the momentum was fixed at 0.9, and training ran for 50 epochs. The size of the hidden layer was tested with 10, 20, and 100 hidden neurons. Accuracy plots and confusion matrices for the 3 runs are shown on the next page. Note the scale for accuracy (the y-axis) of graphs is between 0.88 and 1. Graphs with a scale of 0 to 1 for every accuracy plot are at the end of this report.

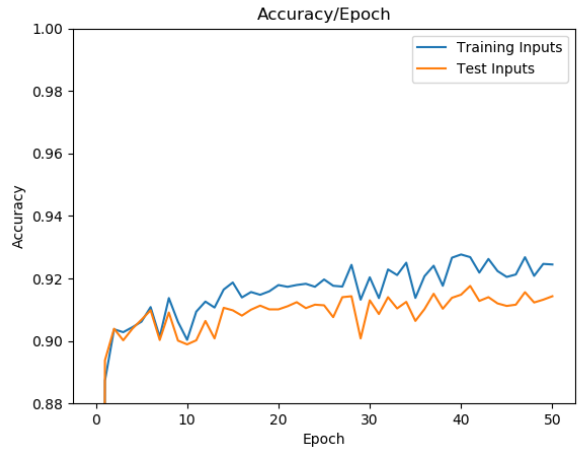

Figure 1: Experiment 1, 10 Hidden Neurons

Target	Guess										Total
	0	1	2	3	4	5	6	7	8	9	
0	957	0	1	4	4	3	4	2	3	2	980
1	0	1113	0	6	1	3	3	1	8	0	1135
2	11	5	928	26	13	7	10	11	20	1	1032
3	5	0	16	904	1	43	2	21	12	6	1010
4	1	0	1	1	893	1	9	6	2	68	982
5	20	1	11	28	14	770	8	10	23	7	892
6	19	2	8	9	6	34	866	0	14	0	958
7	3	8	15	8	4	0	0	954	9	27	1028
8	12	5	11	28	6	28	7	15	855	7	974
9	7	6	0	14	14	5	1	43	16	903	1009

Table 1: Experiment 1, 10 Hidden Neurons

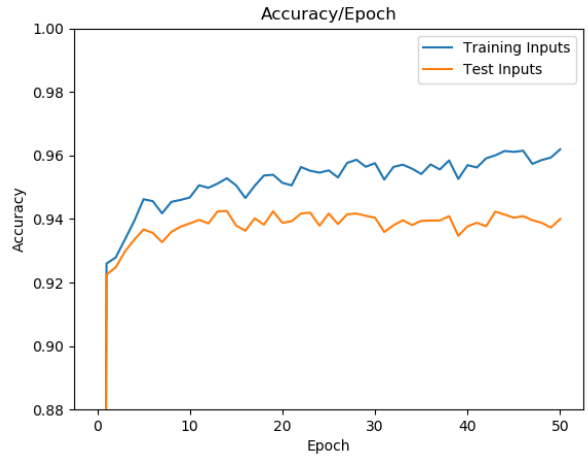

Figure 2: Experiment 1, 20 Hidden Neurons

Target	Guess										Total
	0	1	2	3	4	5	6	7	8	9	
0	954	0	2	1	1	3	5	2	10	2	980
1	0	1114	6	0	1	1	2	1	9	1	1135
2	7	1	965	18	6	1	2	10	17	5	1032
3	0	0	13	959	0	20	0	4	11	3	1010
4	1	1	6	1	909	0	7	0	6	51	982
5	7	1	8	32	0	801	7	5	20	11	892
6	12	3	11	2	7	22	884	0	16	1	958
7	0	4	12	11	3	1	1	967	9	20	1028
8	7	4	7	13	8	15	9	3	898	10	974
9	3	6	1	14	14	5	0	5	12	949	1009

Table 2: Experiment 1, 20 Hidden Neurons

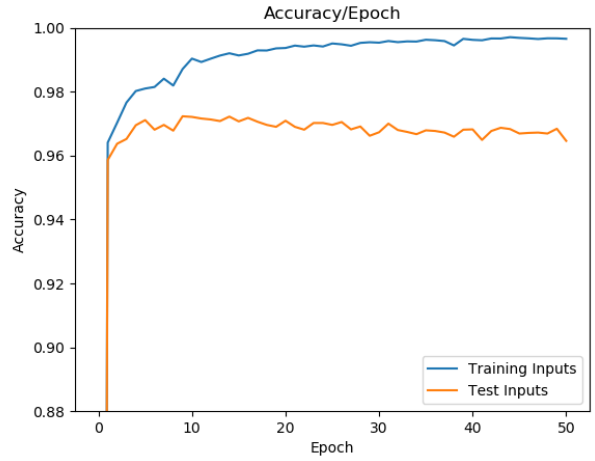

Figure 3: Experiment 1, 100 Hidden Neurons

Target	Guess										Total
	0	1	2	3	4	5	6	7	8	9	
0	968	2	2	0	0	0	2	2	4	0	980
1	0	1120	1	2	0	3	0	1	8	0	1135
2	8	1	985	7	4	2	1	7	15	2	1032
3	0	0	8	967	0	9	0	5	13	8	1010
4	2	0	2	1	953	0	4	3	3	14	982
5	3	1	3	16	3	835	9	2	11	9	892
6	6	4	2	2	0	12	922	0	10	0	958
7	1	3	11	1	2	0	0	986	11	13	1028
8	7	1	2	1	4	5	3	3	945	3	974
9	5	4	1	4	5	1	1	7	16	965	1009

Table 3: Experiment 1, 100 Hidden Neurons

# Hidden Neurons	Final Accuracy (Test Set)	Final Accuracy (Training Set)	Approx. Time to Train 50 Epochs
10	91.43%	92.45%	1m20s
20	94.00%	96.19%	2m47s
100	96.46%	99.66%	13m27s

10 Hidden Neurons: This result had many bumpy oscillations for accuracy on both the test and training data. It was less accurate than the other tests. It has slight over-fitting of the training set. **20 Hidden Neurons:** This result was had less oscillation, was more accurate, and showed more over-fitting than the previous result. **100 Hidden Neurons:** This result was the most accurate and had the smoothest accuracy change of the 3 runs, but it also had the most over-fitting of training data and test accuracy peaked early and trended downwards after 10 epochs.

The neural network is a bit more accurate than the perceptrons from hw1. The perceptrons were between ~84% and ~89% accurate on the test data. The neural net is between ~91% and ~96% accurate. There were similar trends in the confusion matrices in this experiment as with the perceptrons. With 100 hidden neurons, "0" recognition was very accurate, many numbers were still being confused for "8", and "7" and "9" are confused for each other sometimes.

The more hidden neurons, the more accurate the evaluation on the final test set. However, the amount of over-fitting also increased, with almost 100% accuracy on the final training set with 100 hidden neurons.

Training with more hidden neurons increased training time significantly.

It was difficult to see an effect on the number of epochs needed for training to converge. The 10-node example didn't improve much after 5 epochs. The 20-node example didn't improve much after 10 epochs. The 100-node network peaked at 10 epochs then trended downward for the rest of the training! As it became more focused with the training examples, it seemed to lose some of the ability to generalize.

Experiment 2
Vary the momentum

For this experiment, learning rate was fixed at 0.1, the number of hidden nodes was fixed at 100, and the training ran for 50 epochs. The momentum value was tested at 0, 0.5, 0.97, 0.98, and 1.0. Also included is the plot from *Experiment 1* with 0.9 momentum.

Accuracy plots and confusion matrices for the 6 runs are shown on the next 2 pages. Note the scale for accuracy (the y-axis) of graphs is between 0.88 and 1 for most graphs. 2 of the graphs have a scale of 0 to 1 because their values didn't reach 0.88. Graphs with a scale of 0 to 1 for every accuracy plot are at the end of this report.

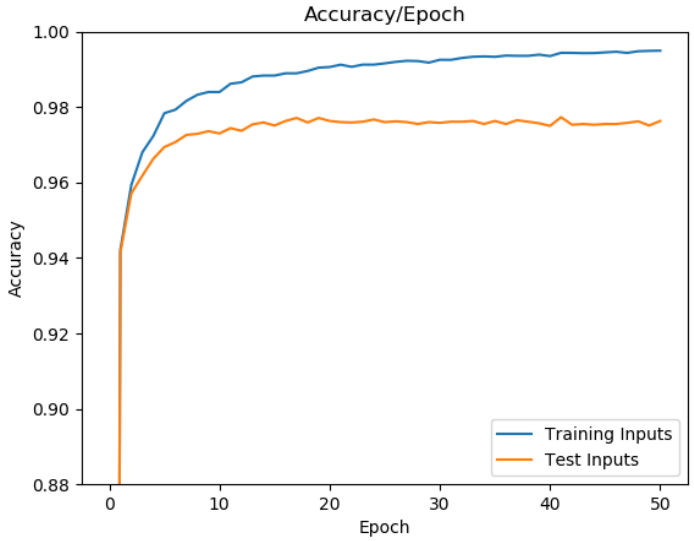

Figure 4: Experiment 2, 0 Momentum

Target	Guess										Total
	0	1	2	3	4	5	6	7	8	9	
0	970	0	0	0	0	1	4	1	4	0	980
1	0	1117	6	2	1	2	2	2	3	0	1135
2	2	2	1007	3	2	0	3	6	6	1	1032
3	0	0	4	992	0	6	0	4	2	2	1010
4	1	0	2	0	959	0	4	2	2	12	982
5	5	0	1	13	0	858	7	1	4	3	892
6	7	3	1	1	0	6	934	0	6	0	958
7	0	2	10	4	1	0	0	998	2	11	1028
8	3	1	3	3	3	1	2	3	954	1	974
9	4	5	0	5	9	2	1	7	2	974	1009

Table 4: Experiment 2, 0 Momentum

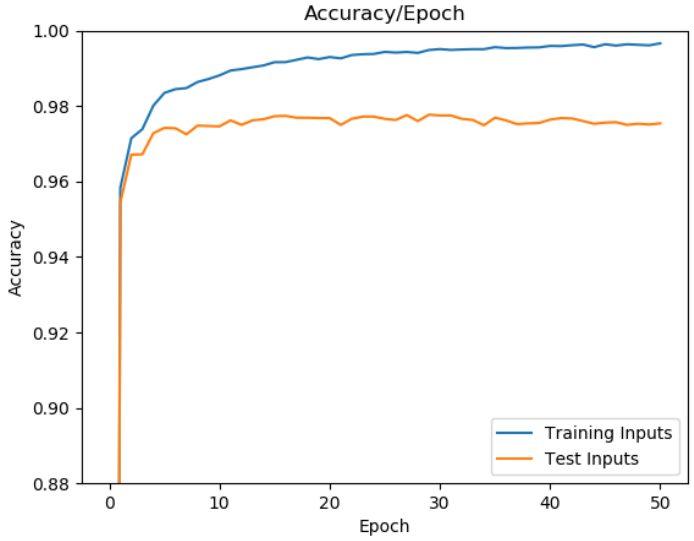

Figure 5: Experiment 2, 0.5 Momentum

Target	Guess										Total
	0	1	2	3	4	5	6	7	8	9	
0	970	1	1	1	0	0	3	1	2	1	980
1	0	1117	3	6	0	1	2	2	4	0	1135
2	2	1	1010	3	0	1	1	7	6	1	1032
3	0	0	5	988	0	6	0	5	4	2	1010
4	4	0	2	0	957	0	5	1	1	12	982
5	3	0	0	11	0	862	4	1	7	4	892
6	5	3	1	1	1	5	935	0	6	1	958
7	1	2	13	4	1	0	0	993	5	9	1028
8	5	0	3	4	3	2	0	4	951	2	974
9	3	4	0	8	8	2	1	3	9	971	1009

Table 5: Experiment 2, 0.5 Momentum

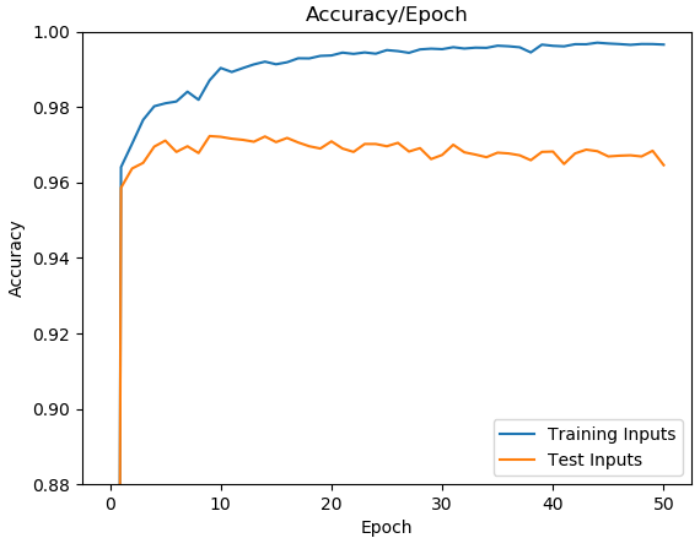

Figure 6: Experiment 2, 0.9 Momentum (Copy of Figure 3)

Target	Guess										Total
	0	1	2	3	4	5	6	7	8	9	
0	968	2	2	0	0	0	2	2	4	0	980
1	0	1120	1	2	0	3	0	1	8	0	1135
2	8	1	985	7	4	2	1	7	15	2	1032
3	0	0	8	967	0	9	0	5	13	8	1010
4	2	0	2	1	953	0	4	3	3	14	982
5	3	1	3	16	3	835	9	2	11	9	892
6	6	4	2	2	0	12	922	0	10	0	958
7	1	3	11	1	2	0	0	986	11	13	1028
8	7	1	2	1	4	5	3	3	945	3	974
9	5	4	1	4	5	1	1	7	16	965	1009

Table 6: Experiment 2, 0.9 Momentum (Copy of Table 3)

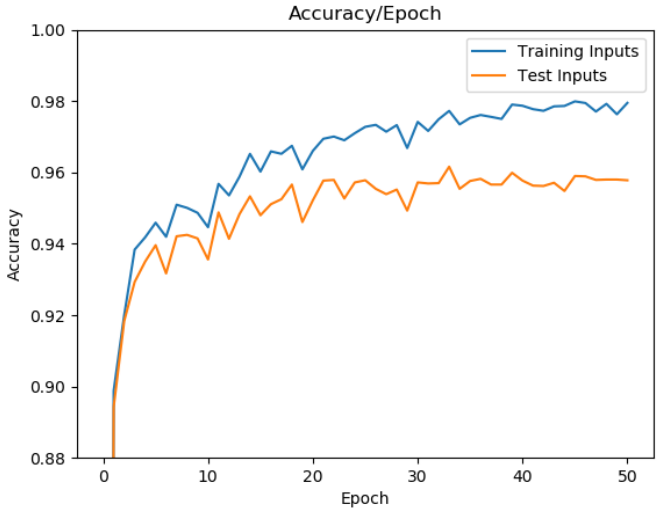

Figure 7: Experiment 2, 0.97 Momentum

Target	Guess										Total
	0	1	2	3	4	5	6	7	8	9	
0	961	1	1	1	0	3	5	1	6	1	980
1	0	1115	2	2	0	1	2	1	12	0	1135
2	6	6	960	5	2	1	3	10	34	5	1032
3	1	1	8	967	0	1	1	5	18	8	1010
4	1	0	1	0	929	0	7	5	6	33	982
5	3	1	1	12	2	838	8	5	18	4	892
6	6	3	2	0	1	5	928	0	13	0	958
7	2	3	10	0	1	0	0	989	6	17	1028
8	8	0	3	5	4	6	4	1	939	4	974
9	3	4	1	16	7	2	0	5	19	952	1009

Table 7: Experiment 2, 0.97 Momentum

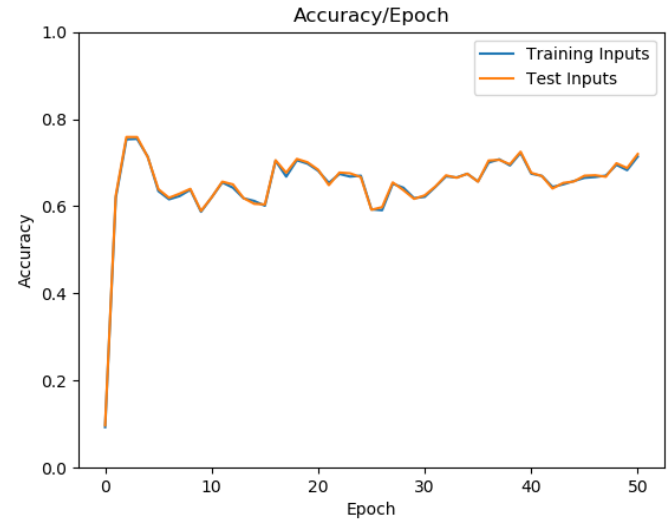

Figure 8: Experiment 2, 0.98 Momentum (Note scale)

Target	Guess										Total
	0	1	2	3	4	5	6	7	8	9	
0	819	1	28	0	0	22	17	8	85	0	980
1	0	1115	7	1	0	3	6	0	3	0	1135
2	18	11	921	7	6	0	19	18	32	0	1032
3	4	19	34	769	0	21	8	23	132	0	1010
4	4	15	11	1	809	0	33	6	103	0	982
5	20	24	30	115	24	343	30	27	279	0	892
6	20	7	24	0	12	0	862	0	33	0	958
7	2	36	30	1	10	1	5	881	62	0	1028
8	14	63	71	58	32	24	24	8	680	0	974
9	19	21	20	9	107	10	16	92	715	0	1009

Table 8: Experiment 2, 0.98 Momentum

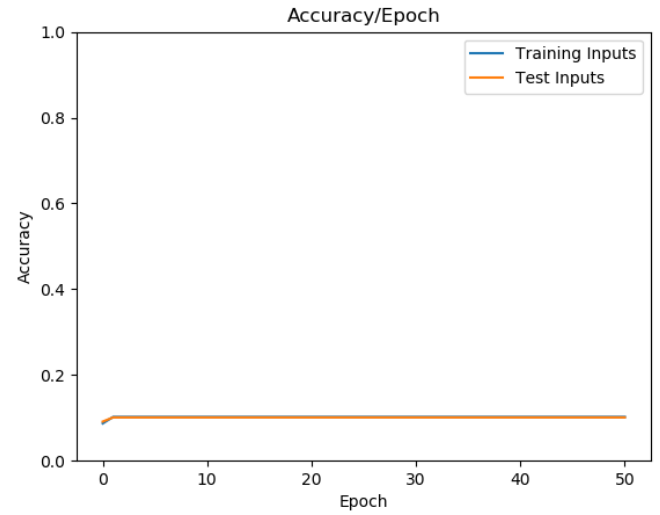

Figure 9: Experiment 2, 1.0 Momentum (Note scale)

Target	Guess										Total
	0	1	2	3	4	5	6	7	8	9	
0	0	0	0	980	0	0	0	0	0	0	980
1	0	0	0	1135	0	0	0	0	0	0	1135
2	0	0	0	1032	0	0	0	0	0	0	1032
3	0	0	0	1010	0	0	0	0	0	0	1010
4	0	0	0	982	0	0	0	0	0	0	982
5	0	0	0	892	0	0	0	0	0	0	892
6	0	0	0	958	0	0	0	0	0	0	958
7	0	0	0	1028	0	0	0	0	0	0	1028
8	0	0	0	974	0	0	0	0	0	0	974
9	0	0	0	1009	0	0	0	0	0	0	1009

Table 9: Experiment 2, 1.0 Momentum

Momentum	Final Accuracy (Test Set)	Final Accuracy (Training Set)
0	97.63%	99.49%
0.5	97.54%	99.66%
0.9	96.46%	99.66%
0.97	95.78%	97.95%
0.98	71.99%	71.42%
1.0	10.1%	10.22%

Momentum 0: This is the best training result of this write-up. The accuracy plot is the smoothest, and the final accuracy of 97.63 is the best in this paper. **Momentum 0.5:** This result is very similar to the previous one in all regards. **Momentum 0.9:** This result is also similar to the previous two, but accuracy on the test set trends downward more in the later epochs. **Momentum 0.97:** This result is has bumpy oscillations for both test and training set accuracy. It is also less accurate in the end on both sets, but there is still similar over-fitting of training data. **Momentum 0.98:** This result never got above 80% accurate. It has bumpy oscillations and it's hard to say whether or not it would have settled given more epochs. **Momentum 1.0:** This result showed no learning. You can see by the confusion matrix that it guessed "3" for everything, achieving ~10% accuracy.

The confusion matrix for 0.98 momentum was the only interesting one in the entire assignment. It never guessed "9". It was very likely to call 9 an 8. It was also somewhat likely to call 9 a 7 or 4. Some numbers were much more accurate than others. 1 was 98.24% accurate! 5 was very inaccurate at 38.45%

Higher momentum seemed to have a negative effect on test set accuracy, with the most accurate example having 0 momentum. Momentum values of 0, 0.5, and 0.9 all had close ending values. Interestingly, having a momentum of 0.9 caused the test set accuracy to peak and then decline over time. 0 and 0.5 momentum seemed to decline over time slightly if at all. **Thus, lower momentums appear to require less time to converge.**

Experiments with 0, 0.5, and 0.9 momentum showed a significant discrepancy between the accuracy of the test set and the training set. The accuracy of the training set continued to improve to over 99% during the 50 epochs, while accuracy of the test set stayed flat or dropped. This indicates over-fitting of the training set. 0 momentum had slightly less disparity between test and training accuracy. 0.5 momentum had more, and 0.9 momentum had even more. **It would appear that higher momentums (within a valid range) slightly increased over-fitting in this experiment.**

A momentum of 1.0 turned out to cause learning to fail. This value causes the error calculation to be small—so small that the calculated weight change is continuously dominated by the previous change (momentum). I don't really understand what causes this. It makes sense intuitively that too high of a momentum would have an amplifying effect and push the weights too far. The interesting thing is that I would expect this to happen with momentums > 1, but this starts to happen somewhere around 0.98 momentum. Even 0.97, although it works, has less accuracy than lower values. Perhaps including the momentum changes the function enough that we should actually use a different derivative function? In any case, one should be careful not to set the momentum too high.

Based on this experiment, momentum doesn't seem to be helpful at all. There is very little difference between momentum values of 0, 0.5, and 0.9 except for slight over-fitting with the higher values. Momentum doesn't seem to reduce oscillations or help find local minima in this experiment.

Experiment 3
Vary the number of training examples

For this experiment, learning rate was fixed at 0.1, the number of hidden nodes was fixed at 100, the momentum value was fixed at 0.9, and the training ran for 50 epochs. A Python program (*splitdata.py*) was written by me to create smaller datasets from the original dataset. The original had 60,000 examples, approximately uniformly distributed between all 10 targets. Another training set was created from the first 50% (30,000) training examples. A third training set was created from just the second quartile of the original training set (15,000, starting with index 15,000 and ending with index 29,999 of the original training set). The second quartile was used because it was more uniformly distributed than the first quartile. The neural net was trained on these smaller training sets.

To run these, you can put smaller sets, *mnist_train25.csv* and *mnist_train50.csv*, in their own folder along with a copy of *mnist_test.csv*. Rename the training set to *mnist_train.csv*. Then when you run the program, specify the new directory instead of the regular data directory. You also have to remove the call to `UnitTest::ValidateLoad` in *main.cpp* lines 144-148 and recompile.

Accuracy plots and confusion matrices for the 2 runs are shown on the next 2 pages. Note the scale for accuracy (the y-axis) of graphs is between 0.88 and 1 for the graphs. Graphs with a scale of 0 to 1 for every accuracy plot are at the end of this report.

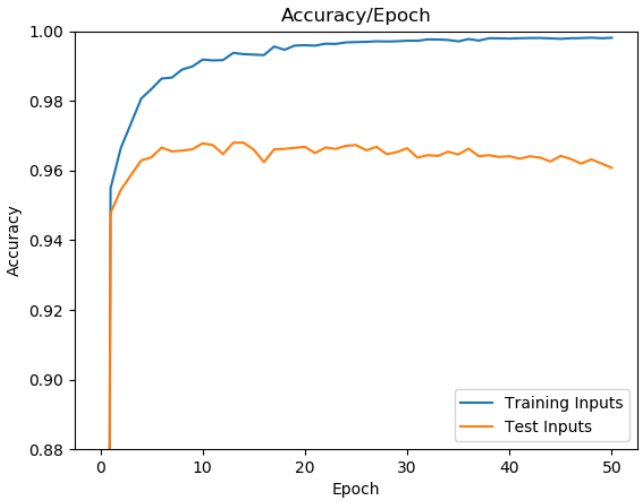

Figure 10: Experiment 3, 50% Training Cases (30k)

Target	Guess										Total
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
0	967	1	0	1	2	0	1	0	5	3	980
1	1	1114	3	1	0	1	5	2	8	0	1135
2	7	4	977	4	1	1	5	10	20	3	1032
3	1	1	11	962	1	11	0	5	11	7	1010
4	1	1	2	0	930	0	4	1	5	38	982
5	4	0	3	7	1	844	6	4	19	4	892
6	6	2	0	1	3	8	926	0	12	0	958
7	0	3	9	2	3	0	0	989	6	16	1028
8	9	0	3	3	5	5	3	6	930	10	974
9	4	3	1	5	6	2	0	6	13	969	1009

Table 10: Experiment 3, 50% Training Cases (30k)

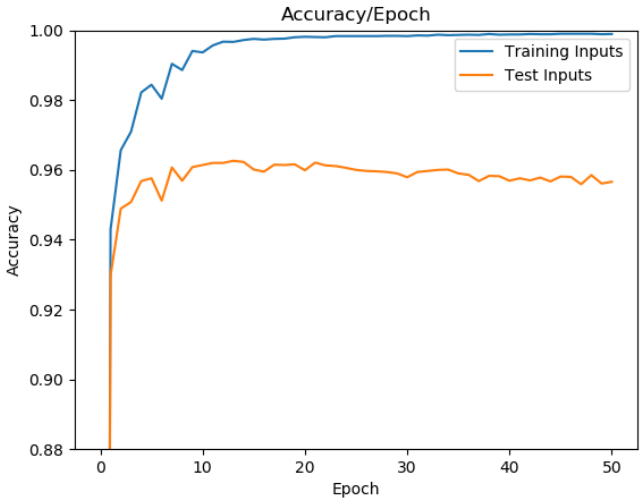

Figure 11: Experiment 3, 25% Training Cases (15k)

Target	Guess										Total
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
0	963	1	5	0	0	0	3	1	3	4	980
1	0	1117	4	1	1	2	5	2	2	1	1135
2	4	2	982	5	2	0	4	12	14	7	1032
3	1	0	13	941	0	9	0	12	28	6	1010
4	1	0	2	0	951	0	8	0	3	17	982
5	5	0	0	14	3	832	8	3	20	7	892
6	14	3	2	0	3	5	921	0	9	1	958
7	1	8	10	2	6	1	0	973	8	19	1028
8	8	1	2	5	5	5	2	4	938	4	974
9	3	4	2	5	16	3	0	5	23	948	1009

Table 11: Experiment 3, 25% Training Cases (15k)

Num Training Examples	Final Accuracy (Test Set)	Final Accuracy (Training Set)
100% (from Exp. 1)	96.46%	99.66%
50%	96.08%	99.81%
25%	95.66%	99.89%

100% Training Examples (taken from Experiment 1, aka Figure 3 and Table 3): This result was the most accurate in the end. **50% Training Examples:** This result was only slightly less accurate. **25% Training Examples:** This was the least accurate, although only slightly. All 3 results looked very similar in terms of accuracy/epoch, over-fitting, size of accuracy oscillations, and accuracy peaking early and trending after about 10 epochs.

Having fewer training examples made the neural network less a little less accurate on the test set. However, the difference isn't a lot. The neural net is surprisingly accurate with just a quarter of the training examples. Both runs show the same convergence, over-fitting, and decline in accuracy over time as we saw in Experiment 1 with 100 hidden nodes.

All 3 plots showed the same over-fitting , seen as a much higher accuracy in the training set and a corresponding decline in test set accuracy**. The size of the training set had the slightest effect on over-fitting, with the smallest training set having a slightly higher disparity between test set and training set accuracy.**

All 3 plots showed similar convergence, with the accuracy rising after the first epoch to 90+%, peaking between 5-15 epochs, and then trending slightly downward. **It was difficult to see the effect of training set size on convergence because of large bumps in accuracy for the 25% and 100% plots.**

Every Graph with the Same Scale

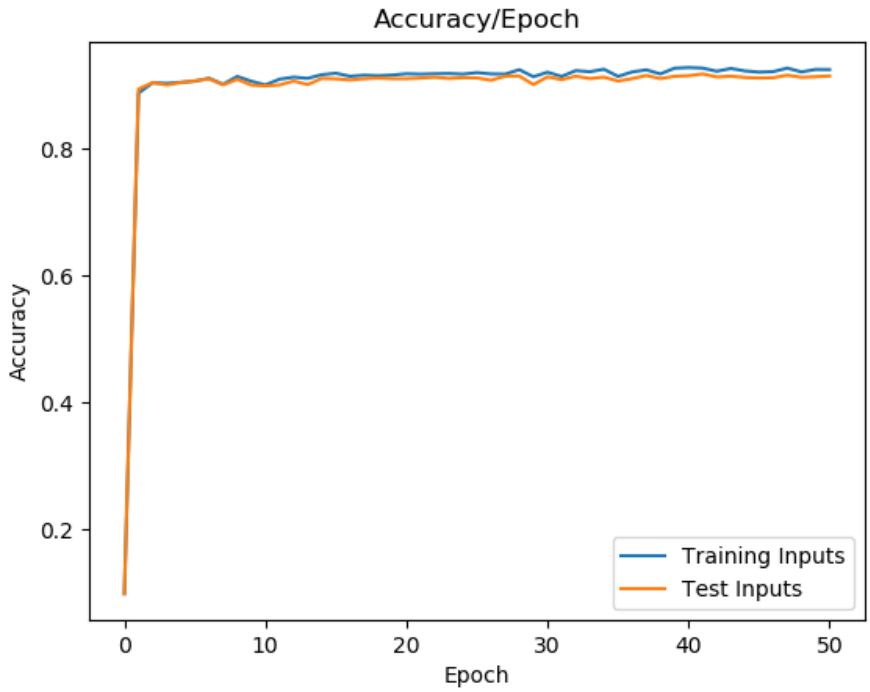

Figure 12: 10 Hidden, 0.9 Momentum

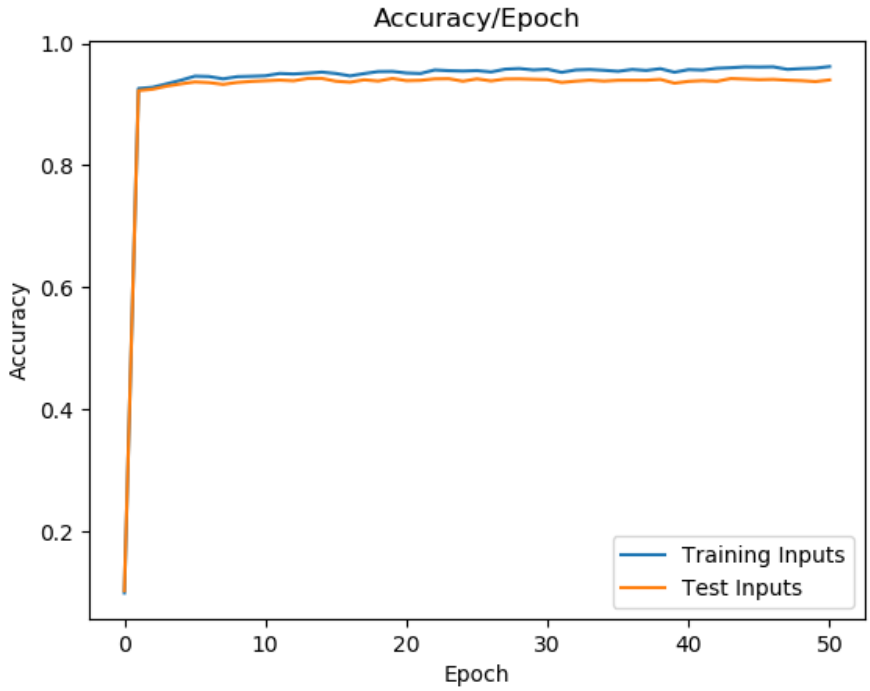

Figure 13: 20 Hidden, 0.9 Momentum

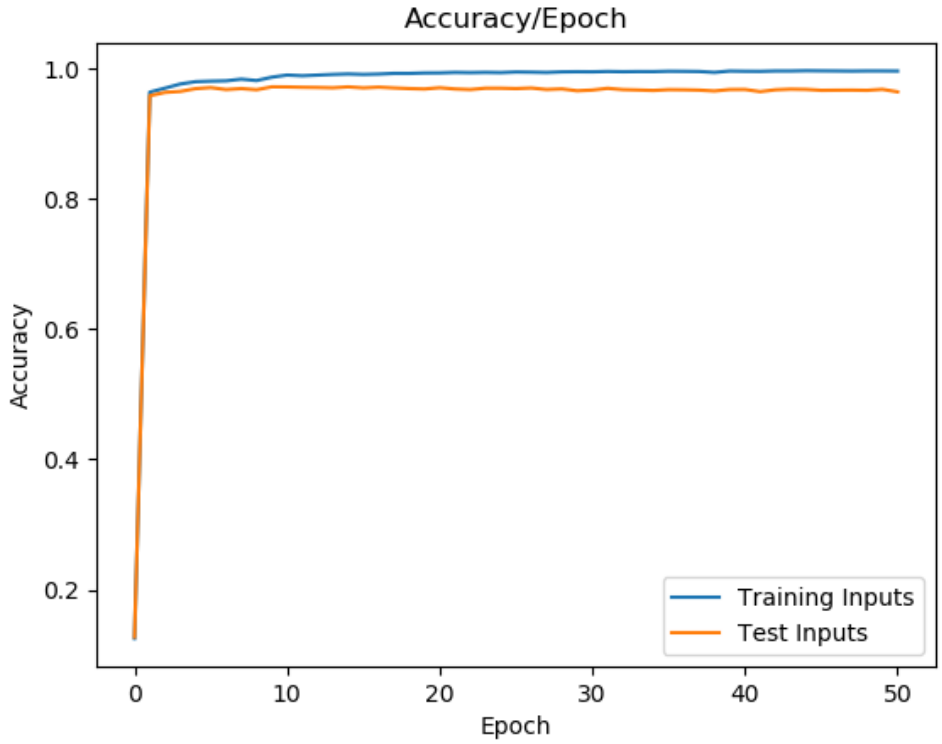

Figure 14: 100 Hidden, 0.9 Momentum

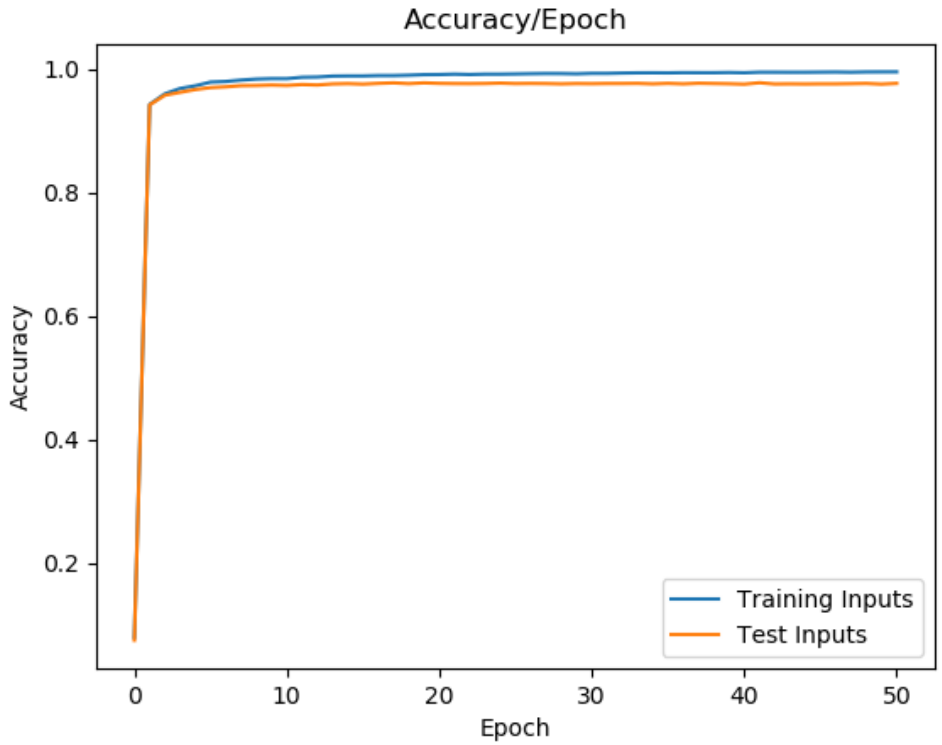

Figure 15: 100 Hidden, 0 Momentum

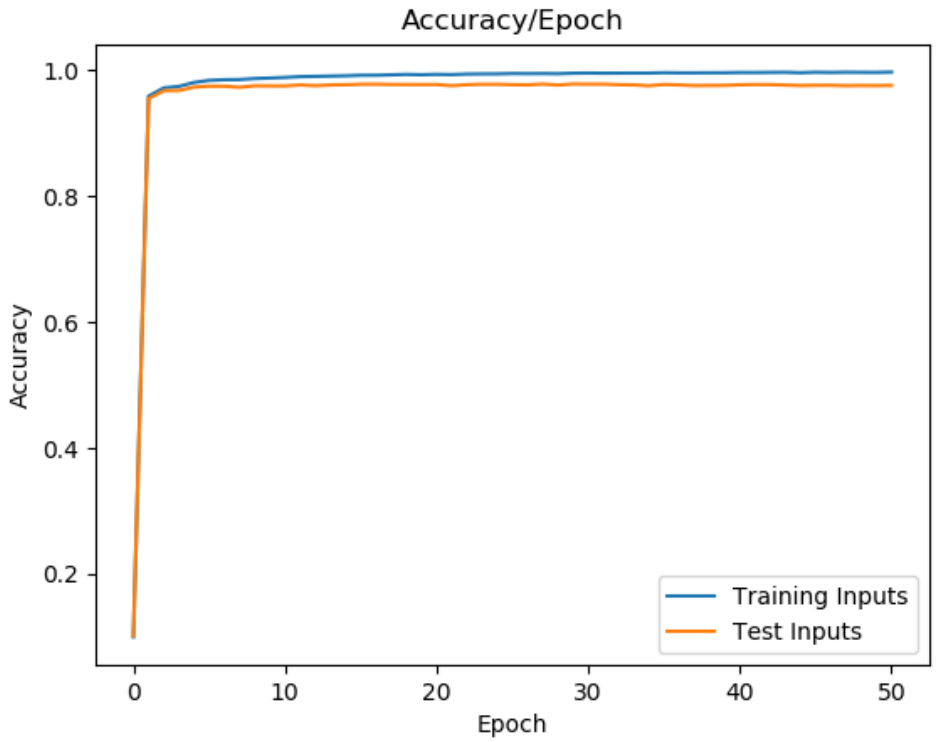

Figure 16: 100 Hidden, 0.5 Momentum

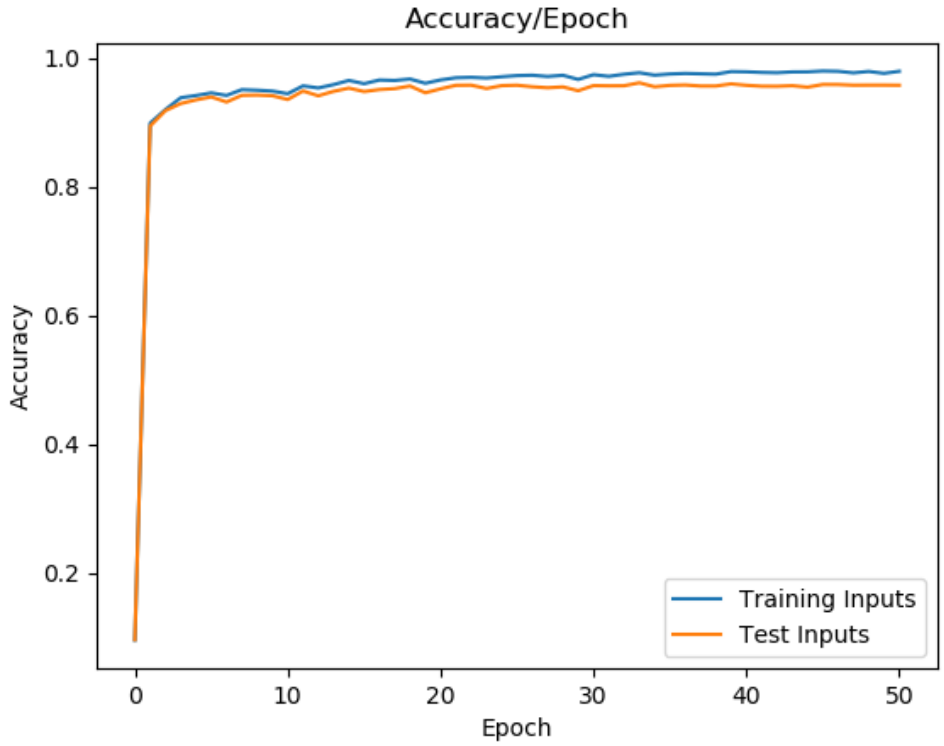

Figure 17: 100 Hidden, 0.97 Momentum

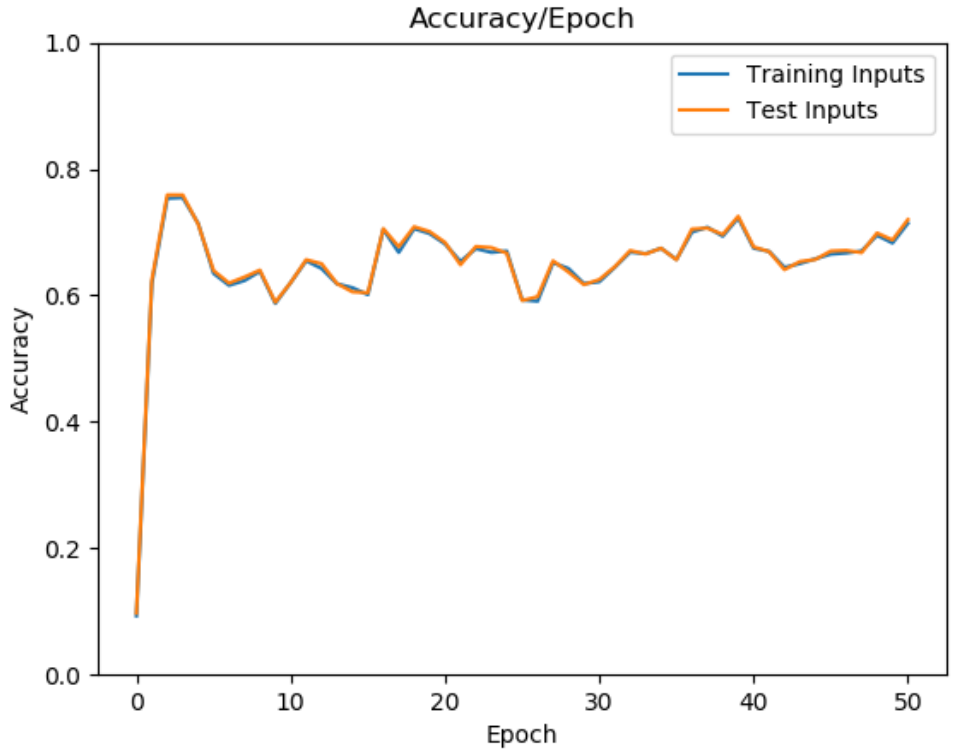

Figure 18: 100 Hidden, 0.98 Momentum

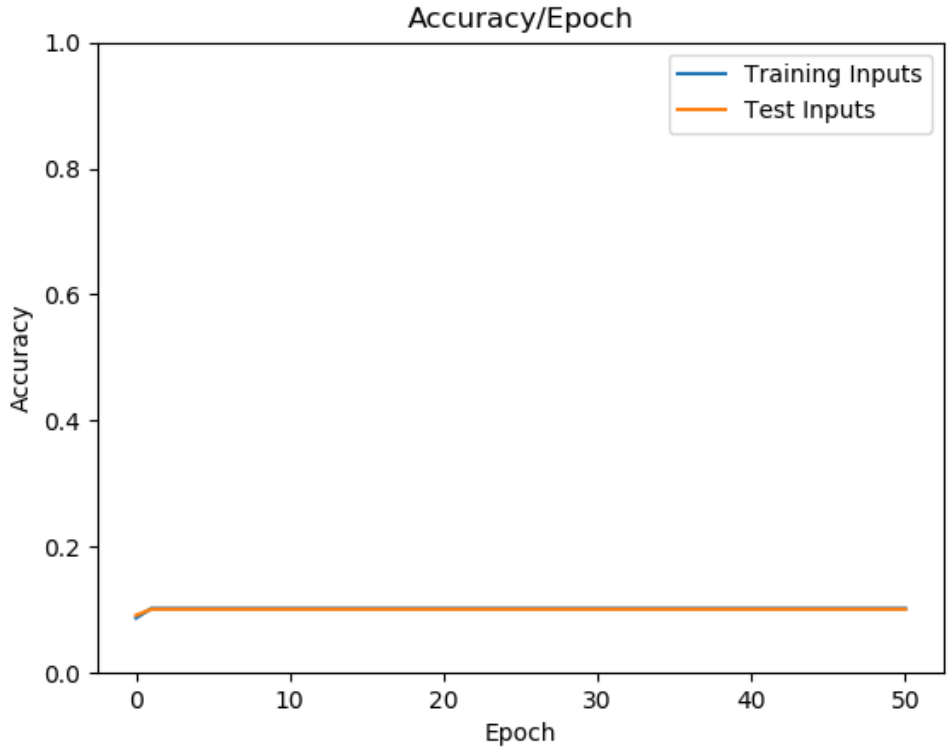

Figure 19: 100 Hidden, 1.0 Momentum

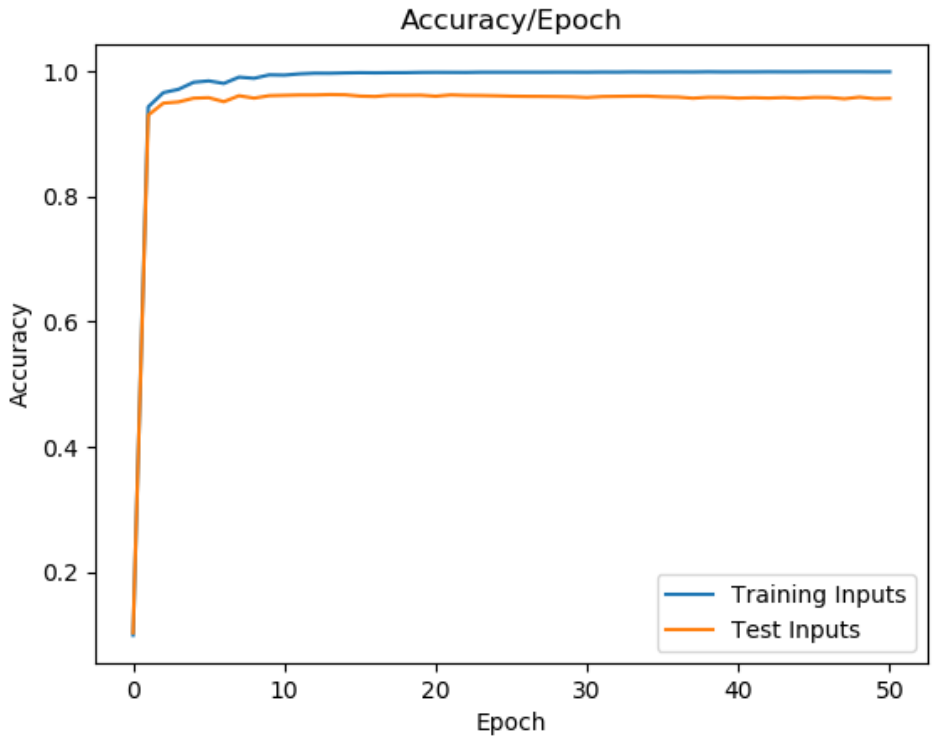

Figure 20: 100 Hidden, 0.9 Momentum, 25% Training Set

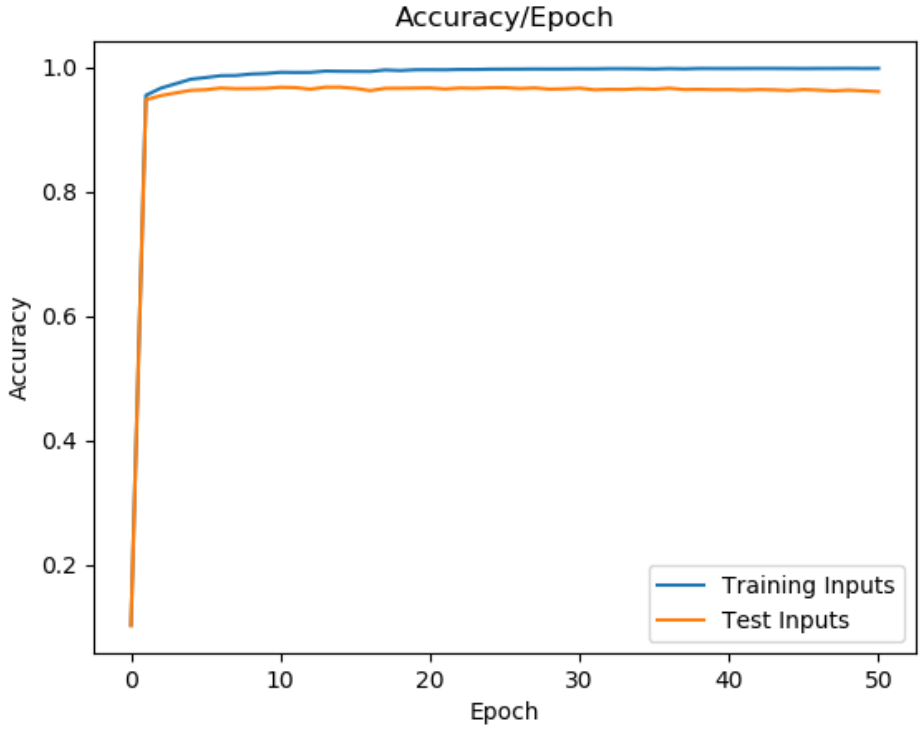

Figure 21: 100 Hidden, 0.9 Momentum, 50% Training Set