

Neural Network Experiments

To illustrate practical techniques, I chose to use the “glass” dataset. This dataset has 214 examples and 6 classes. Here are 4 examples from the original dataset. The last values indicates the class.

```
1.52101 13.64 4.49 1.10 71.78 0.06 8.75 0 0 1
1.51761 13.89 3.60 1.36 72.73 0.48 7.83 0 0 1
1.51618 13.53 3.55 1.54 72.99 0.39 7.78 0 0 1
1.51766 13.21 3.69 1.29 72.61 0.57 8.22 0 0 1
```

Some of the values are relatively large, and some attributes have a small range, so I scaled each attribute to be between 0 and 1, and formatted the examples for my program.

```
0.43 0.44 1.00 0.25 0.35 0.01 0.31 0 0
 1 -1 -1 -1 -1 -1
0.28 0.48 0.80 0.33 0.52 0.08 0.22 0 0
 1 -1 -1 -1 -1 -1
0.22 0.42 0.79 0.39 0.57 0.06 0.22 0 0
 1 -1 -1 -1 -1 -1
0.29 0.37 0.82 0.31 0.50 0.09 0.26 0 0
 1 -1 -1 -1 -1 -1
```

For training and testing purposes, I randomly split the dataset into 114 training and 100 test examples. This split was stratified, i.e., each split has about the same proportion of each class.

Unless stated otherwise, there are output neurons, hidden neurons use tanh activation, output neurons use identity activation, and the goal is to minimize square error/2.

Do not generalize too much from these results. The samples are too small.

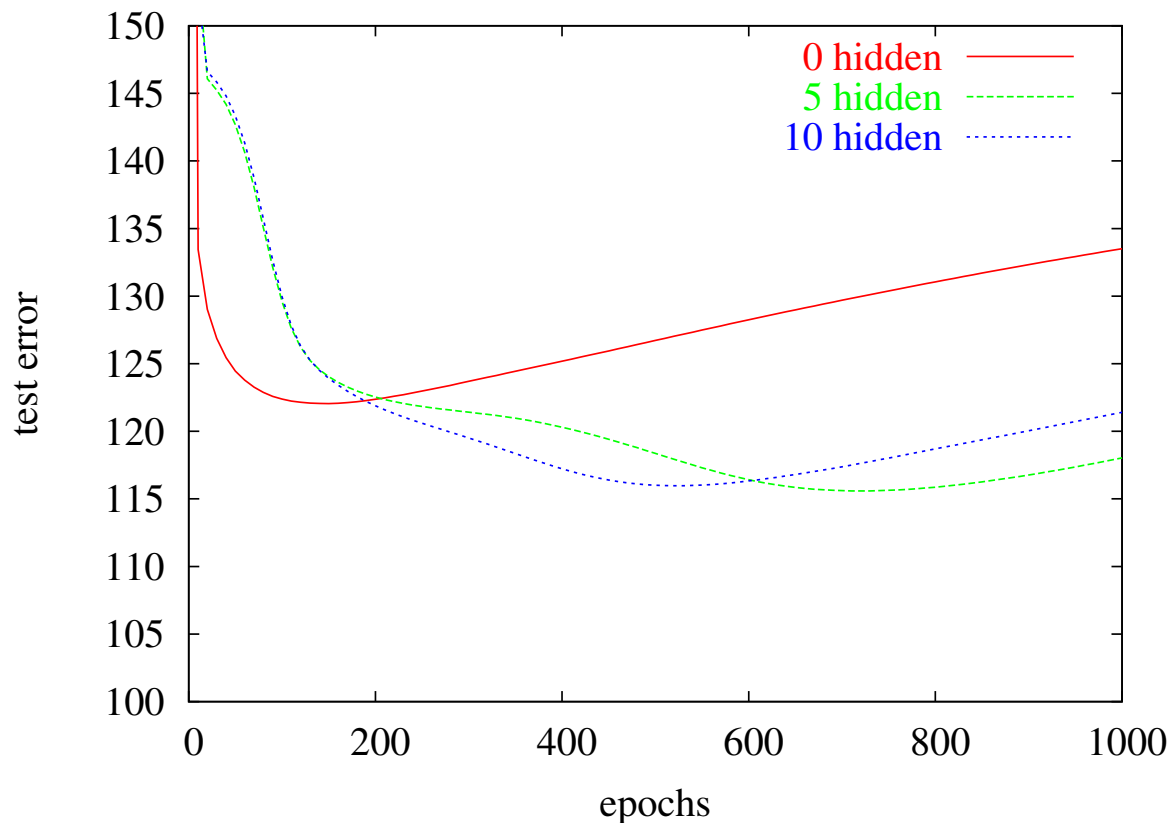
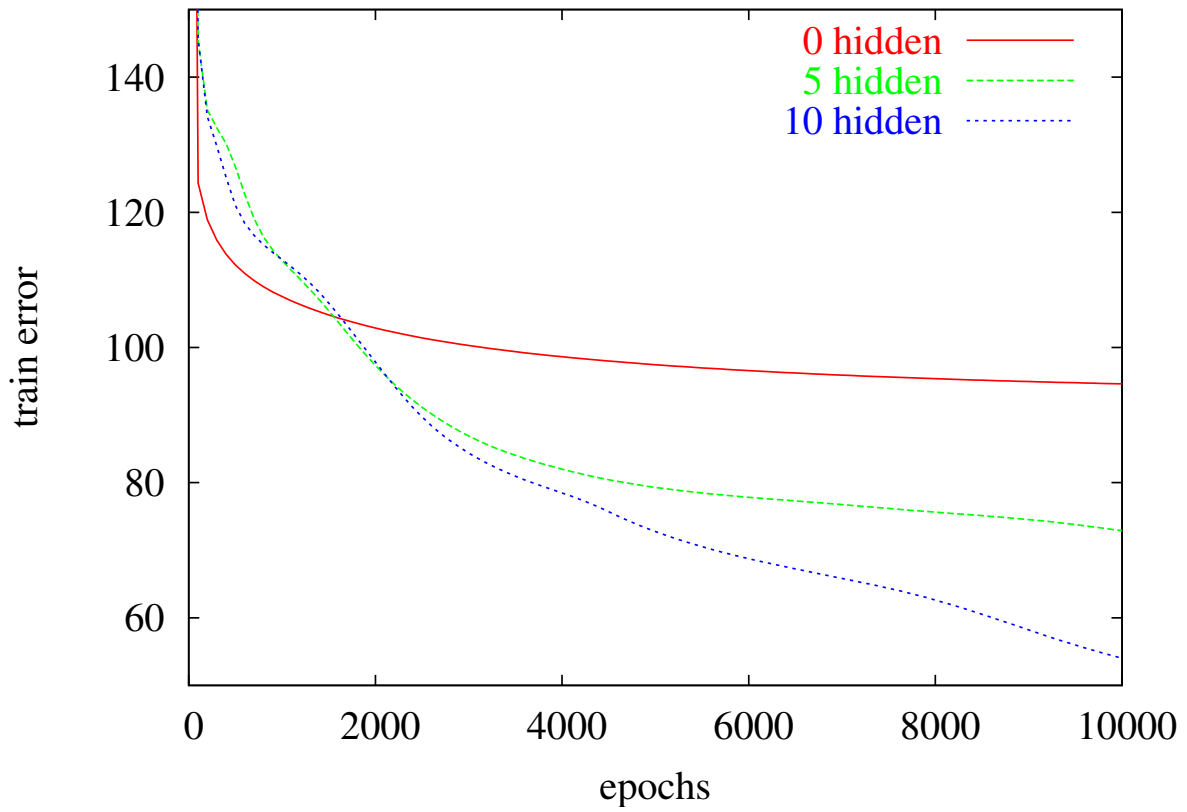
Number of Hidden Neurons

With more hidden neurons, lower training error is expected, but test error will increase at some point. I compared the following.

0 hidden neurons, 0.01 lrate, initial weights 0

5 hidden neurons, 0.001 lrate, random initial weights in $[-0.1, 0.1]$

10 hidden neurons, 0.001 lrate, random initial weights in $[-0.1, 0.1]$



Interpretation of results:

Over 10,000 epochs, 10 hidden units had the lowest training error, and 0 hidden units the highest.

Over 1,000 epochs, all NNs decreased test error up to a point, and then started overfitting. 5 and 10 hidden units had the lowest minimum, but the training algorithm needs a stopping criterion.

Different Output Activation Functions

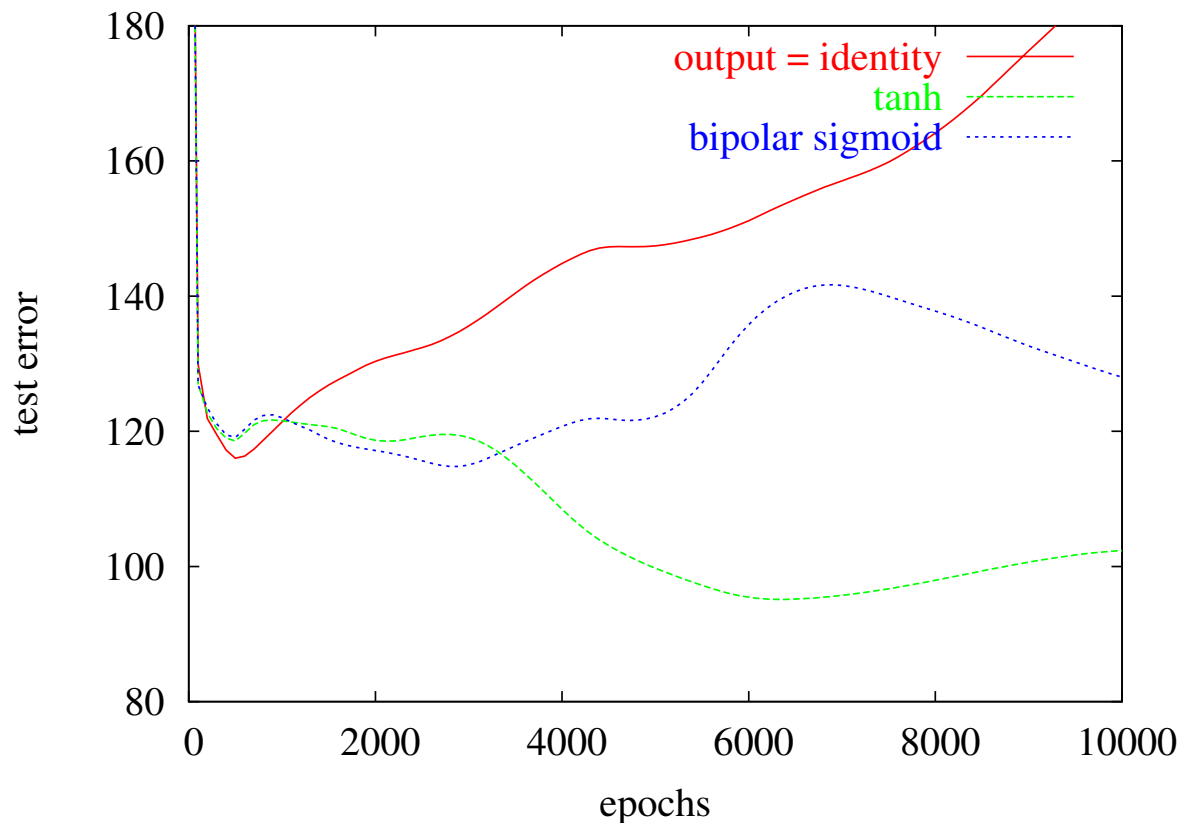
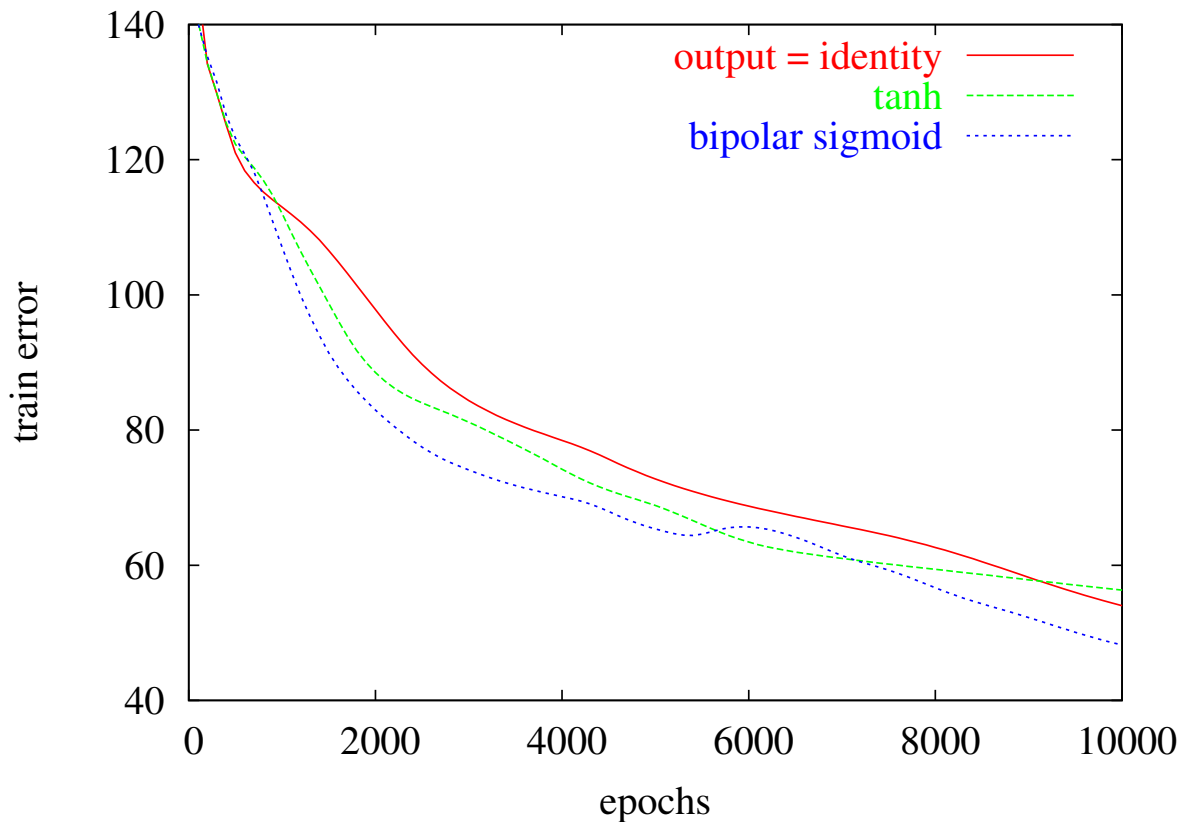
Because the desired outputs are all in $\{-1, 1\}$, an activation that keeps outputs in $[-1, 1]$ might produce much lower error.

Using 10 hidden units, initial weights in $[-0.1, 0.1]$, and tanh for hidden neurons, I considered:

identity activation for output units, 0.001 lrate

tanh activation for output units, 0.002 lrate

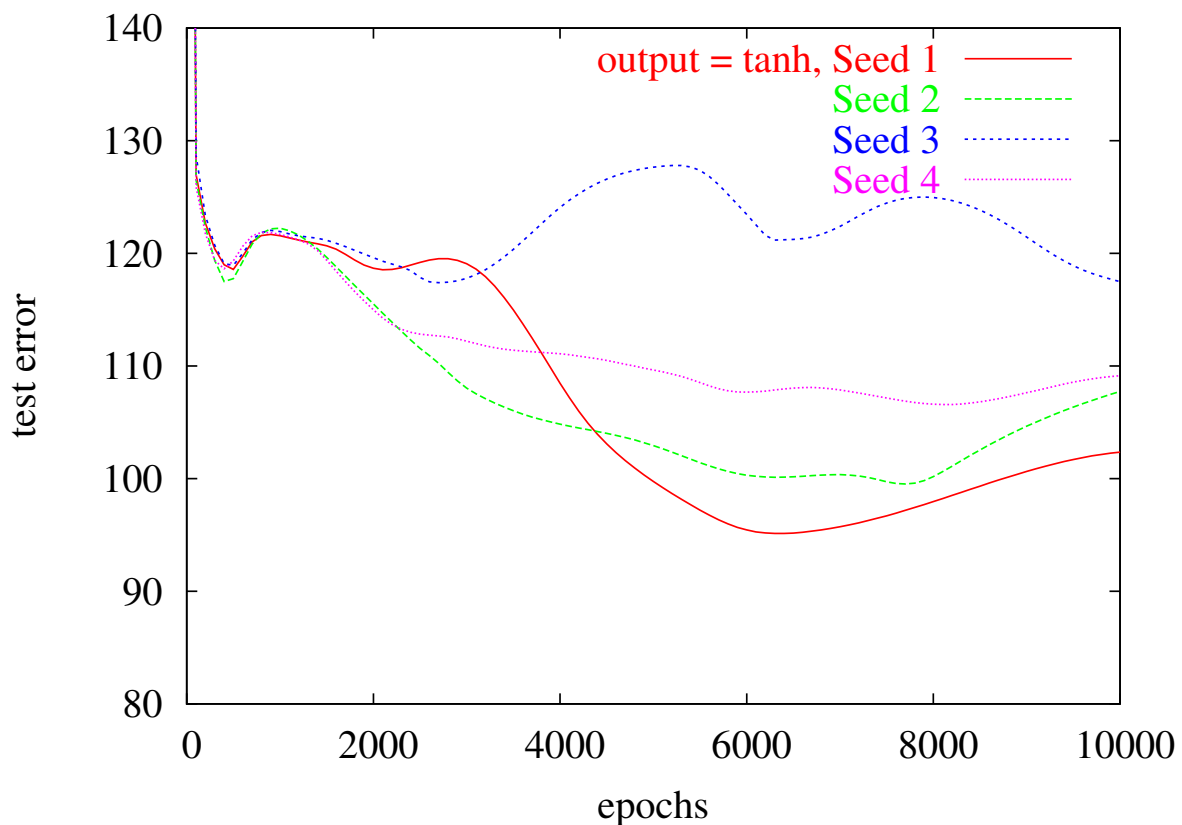
bipolar sigmoid for output units, 0.005 lrate



The three output activation functions are very close in training error, but tanh outperformed the other two on test error.

I ran tanh output activation again, starting the random number generation with different seeds.

The result was a considerable variation in performance though all of them were better than the other two activation functions.



Different Hidden Activation Functions

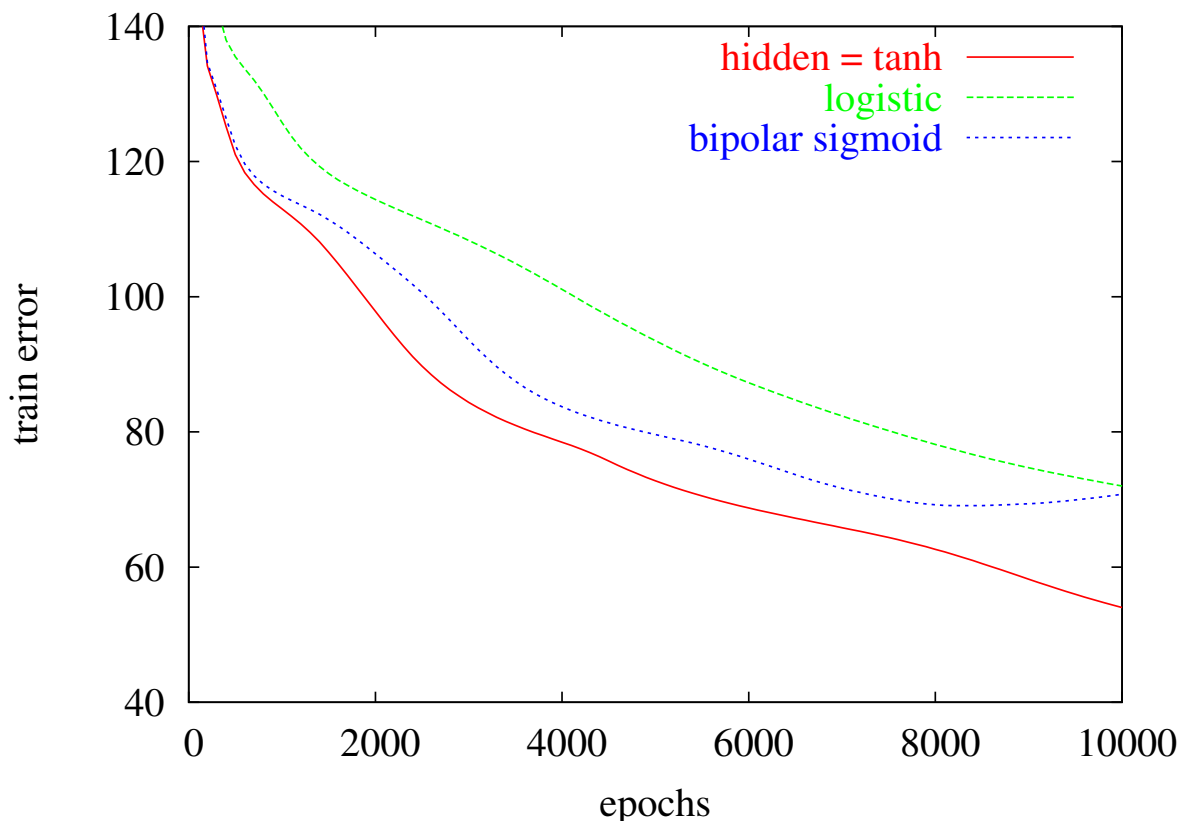
I also tried different activation functions for the hidden units.

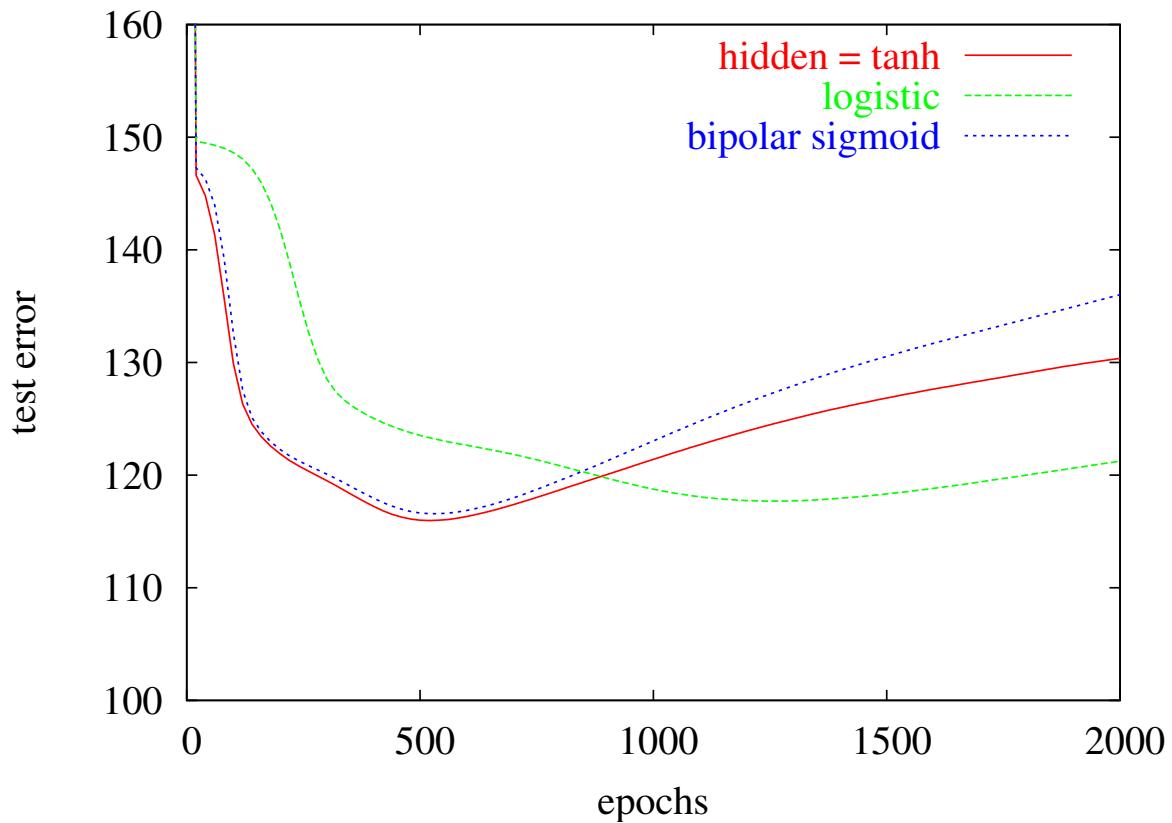
Using 10 hidden units, initial weights in $[-0.1, 0.1]$, and identity for output neurons, I considered:

tanh activation for hidden units, 0.001 lrate

logistic activation for hidden units, 0.002 lrate

bipolar sigmoid for hidden units, 0.002 lrate





tanh for hidden neurons achieved the best training error, but all three had similar results for test error.

In the first 1000-2000 epochs a test error minimum would be reached and then overfitting would take place.

Weights Initialization

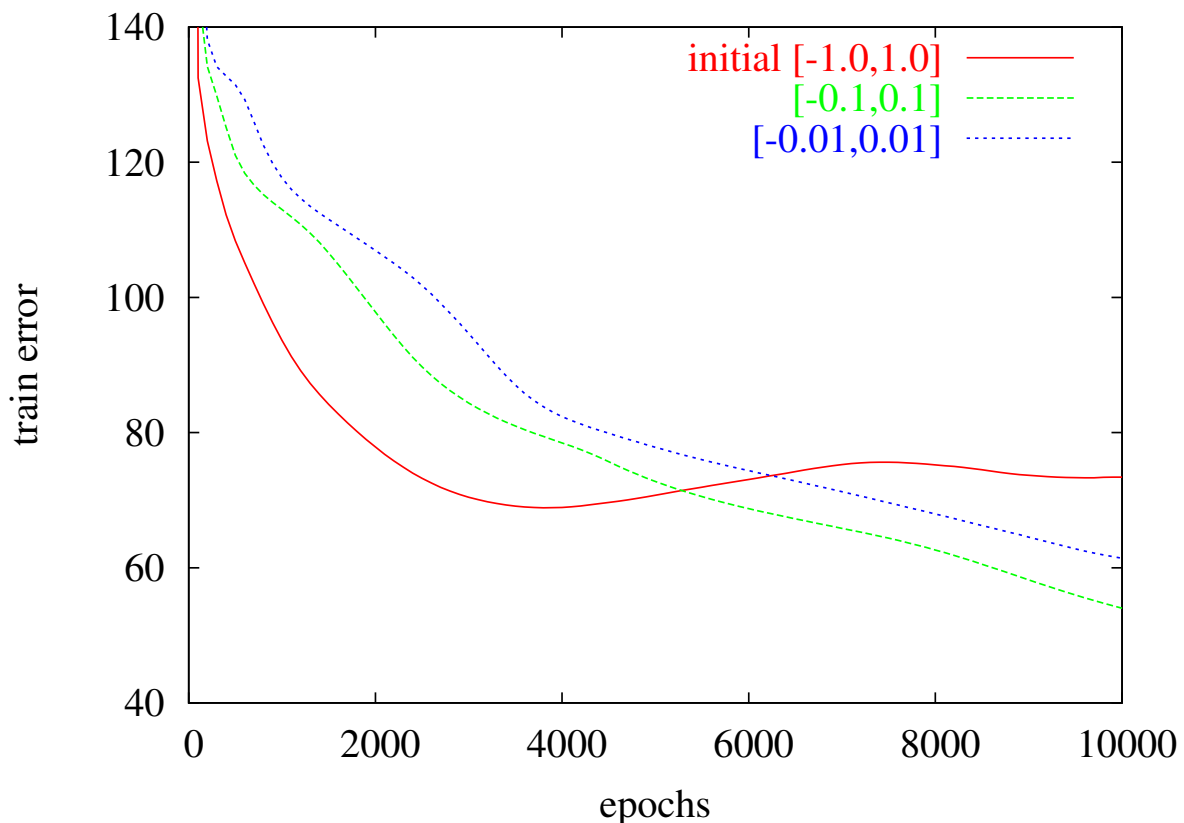
Initializing all weights to zero results in zero learning. With zero weights, a tanh hidden neuron produces a zero output, and, as a result, all error derivatives are zero.

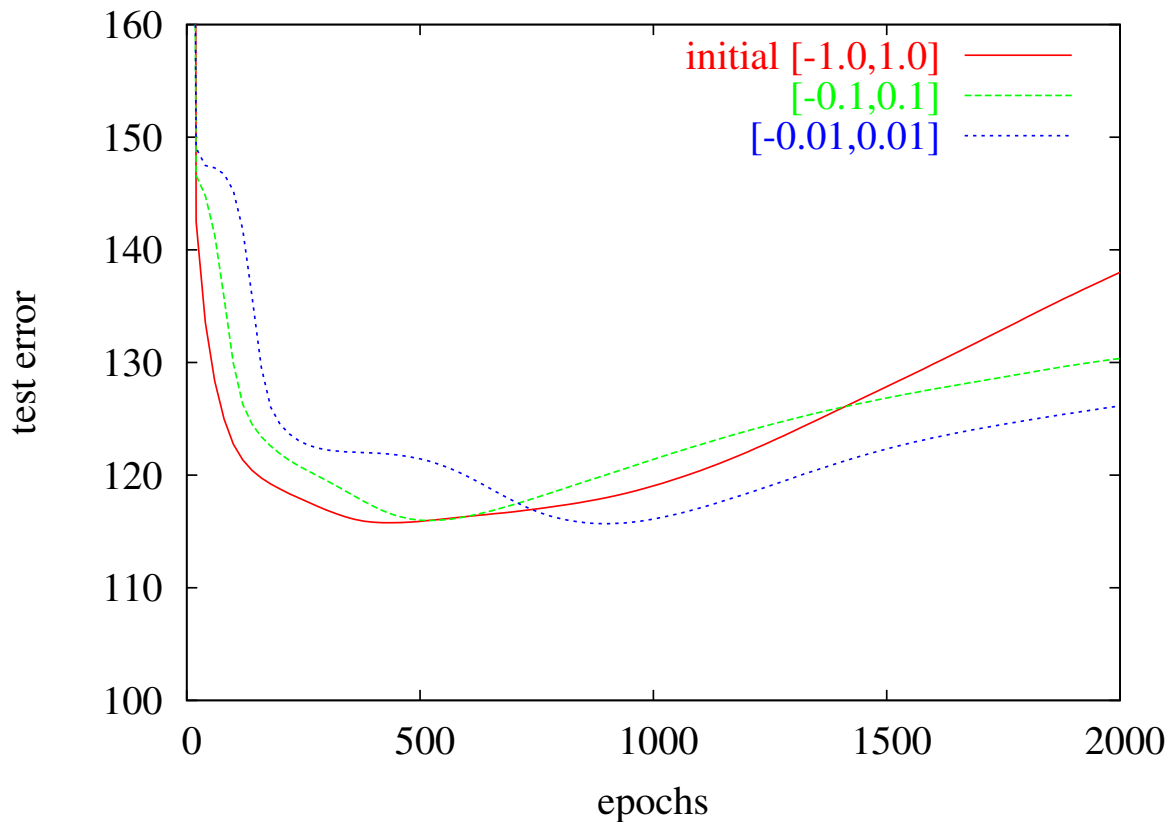
Using 10 hidden units, 0.001 lr rate, tanh for hidden neurons, and identity for output neurons:

random initial weights in $[-1.0, 1.0]$

random initial weights in $[-0.1, 0.1]$

random initial weights in $[-0.01, 0.01]$





If the weights are initially too high, it is more likely to find a bad minimum. If too low, then convergence will be slower.

In this case $[-1.0, 1, 0]$ leads to initially faster convergence, but then hits a minimum after 4000 epochs.

The test error is similar except for faster learning for larger initializations.

Selection Criterion: Validation Set

A NN with more hidden neurons can represent more functions, but is also more likely to overfit the data. I recommend using validation sets as a selection criterion.

Validation Set Method:

Split the training set into a learning set and a validation set.

Train on the learning set.

Choose the weights that minimize error on the validation set.

I split the glass dataset into a test set of 50 examples, a learning set of 114 examples, and a validation set of 50 examples.

test set minimum	validation set method
57.9546	58.6212
52.1225	62.2628
50.3152	55.4330
51.4142	51.9228
48.6549	57.5341

There is considerable variance. This improves with larger datasets.

Momentum

Faster convergence can often be achieved by combining error derivatives over several epochs.

Momentum applies to batch learning.

Let Δw be the previous weight change.

Let $\eta_m \in (0, 1)$ be the momentum parameter.

Then the new weight change is:

$$\Delta w \leftarrow \eta_m \Delta w - \eta \frac{\partial E}{\partial w}$$

This approximates second derivative methods.

Using 10 hidden units, $\eta = 0.1/114$ initial weights in $[-0.1, 0.1]$, tanh for hidden neurons, and identity for output neurons, I evaluated:

$\eta_m \in \{0, 0.2, 0.4, 0.6, 0.8\}$, batch learning

The training error decreases with higher momentum, though 0.8 is a little unstable.

Unfortunately, higher momentum also results in faster overfitting. Note: the x -axis is logscale in the test graph.

