Taking a Turn on Perceptron Curve Fitting

Steven Bos (#1319671, steven@stevenbos.com)
Martijn Haak (#1345389, m.i.j.haak@student.tudelft.nl)
Sacha Panic(#1367803, a.s.panic-1@student.tudelft.nl)

[ABSTRACT] Neural networks have been a suitable candidate when it comes to curve fitting. But how exactly do the different features of the curve influence the performance of a single layer neural network? We have looked at three types of curves within the range [-1,1]. The first is a third order curve rotated around the origin, the second a sixth order curve symmetric in the line $x = 0$ and the third is an asymmetric curve. All the features of the curve impact the performance or size requirements of the neural network in one way or another, sometimes in intuitive ways, sometimes more subtly. Hence no network will fit all situations.

1. INTRODUCTION

Using neural networks for function or curve fitting seems to be mostly a scientific toy problem. On the other hand it has been applied in practical situations, such as stock market predictions [1]. In fact, neural networks seem to be fairly good at it, as this paper will show. Apart from requiring sometimes large amounts of training data and some tweaking, the only thing neural networks they seem to have going against them is the *fuzziness* of the solution. That is, it is often, if not always, hard to tell *why* it works.

This paper will outline the experiment and its results, where we try to show several aspects of a neural network, and how they influence its performance. This include varying the complexity of the curve to fit, symmetry of the curve, sampling rate of the curve and interpolating and extrapolating data missing from the training dataset.

1-1. TOOLS

We experimented with both Java NNS, Neuro Solutions 5 (with interactive book) and MATLAB (with the Neural Network Toolbox) in order to see which tool best fit our requirements and experience. MATLAB soon showed to be a better fit, as it proved to be more flexible in many ways. MATLAB allows easier importing of data from external sources as well as data generation. It also performed better in the area of (ease of) configurability, even though all tools allow extensive tweaking of the neural network.

Attempts to make their features accessible to novices have failed for all applications, but those familiar with MATLAB, and that applies to many scientists, will have a significant advantage when it comes to understanding the interface. MATLAB even provides different modes of interaction, ranging from a click'n'go GUI targeted specifically to function fitting, to the command line interface commands which should be familiar to use for most MATLAB adepts. And MATLAB's features with regards to visual presentation are hard to match.

MATLAB also beats Java NNS when it comes to the price tag, albeit in a negative way. Of course the extra cash gets you more than just a Neural Network simulation kit, and the toolkit comes with some very good documentation, but unfortunately all that money doesn't buy you a bug-free to play around with.

2. MODEL

For this experiment, three polynomials have been taken of varying degrees, to see how the neural networks would tackle them. A great benefit of these functions is that each input value maps to single output value. We've restricted our experiment to this polynomials within the interval [-1,1]. Each of the polynomials have (local) extremes within the [-1,1] interval, and vary in symmetry. They are displayed in figure 0.

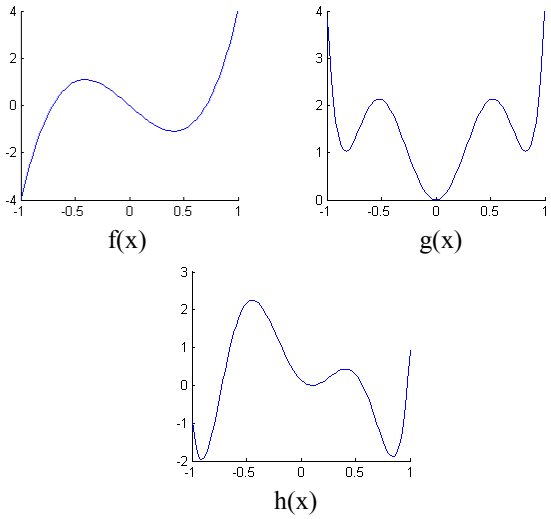

Figure 0 *– The curves used for the experiments.*

$$f(x) = 8x^3 + 4x$$
$$g(x) = 32x^6 - 46x^4 + 14x^2$$
$$h(x) = 32x^6 - 46(x - \tfrac{1}{50})^4 + 14(x - \tfrac{1}{10})^2$$

The sampling rate is changed for one of the experiments, but most experiments are performed with the curves samples at 50 uniformly distributed points (along the x-axis) within the interval [-1,1] inclusive. The neural network is then tested at 200 sample points, also uniformly distributed along the same interval.

3. Set-up of Experiment

As described in the introduction, MATLAB's Neural Network Toolbox has been the tool of choice for carrying out the experiments. Figure 1 shows the set-up of our network. The hidden layer contains *n* neurons, with *n* varying for the different experiments, and a single neuron in the output layer.

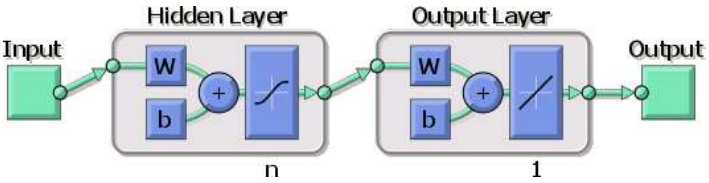

Figure 1 *– Set-up of the experiment: The input layer is connected to all neurons in the hidden layer. The hidden layer uses a tangent sigmoid transfer function. There is only one neuron in the output layer, which uses a pure linear transfer function.*

For our experiment, most of the network's variables are kept the same and only the number of neurons in the hidden layer is varied. These parameters are listed in the table 1.

Parameters	
Network type	Feedforward
Neurons	1 input layer (size 1) 1 hidden layer (size *n*) 1 output layer (size 1)
Training algorithm	Levenberg-Marquardt
Adjustment algorithm	Gradient descent
Performance measure	Mean Squared Error
Transfer function Hidden layer Output layer	 Hyperbolic tangent Linear

Table 1 *– Configuration of the network used in this experiment. Only the amount of neurons in the hidden layer is varied for the experiments.*

The training sets were generated using MATLAB. The base dataset consists of *n* values uniformly distributed between -1 and 1 inclusive. Next, datasets were generated with carefully chosen gaps. This way we can see how well the networks will still be able to generate the original function when interpolating and extrapolating. The test and validation dataset are randomly extracted from the input dataset, each taking up about 10% of the input samples.

4. Results

The following sections outline the results for each of the sub-experiments.

4-1. Complexity

It may be little surprising that the complexity of polynomial directly affects the requirements of the network. As expected, higher order polynomials require a larger hidden layer. Figure 2 shows the results of the trained networks with differently sized hidden layers.

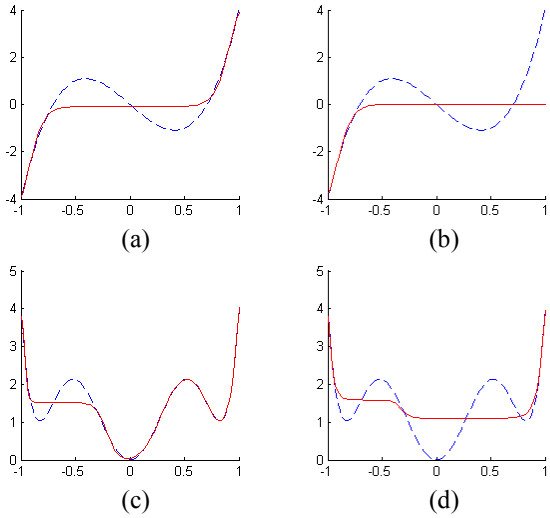

Figure 2 *– Approximation of curve with insufficiently large hidden layers: third order curve approximated with 2 neurons (a) and 1 neuron (b); sixth order curve approximated with 5 neurons (c) and 3 neurons (d).*

More surprisingly, only a few neurons are required in the hidden layer to accomplish near-exact approximation. Three and six neurons in the hidden layer suffice for the third order and sixth order polynomial respectively, which are both accurate up to three decimal points.

4-2. Symmetry

Intuitively it may make sense, but it is still a little surprising that the symmetry of the curve affects the network's performance. The approximation of a symmetric curve with the same complexity (that is, the same degree of polynomial, and the same number of extremes) is more accurate. In fact, where 6 neurons were able to create an almost exact approximation of the curve, as shown in the

previous paragraph, the same preciseness of the asymmetric curve requires 8 or more neurons, although figure 3 below shows that both 6 and 7 neurons are able to do an acceptable job.

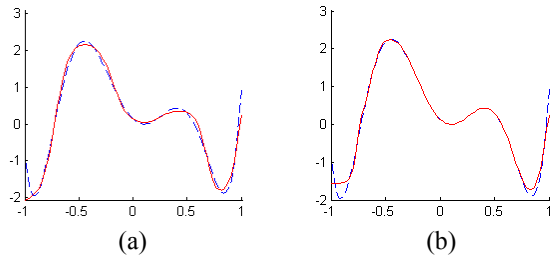

Figure 3 *– Approximation of an asymmetric curve using 6 neurons (a) and 7 neurons (b).*

Interestingly, even with a large number of neurons in the hidden layer, the network seems to come to an (locally) optimal solution much faster for the asymmetric curve. The network almost without exception takes up all 1000 epochs (which is the set limit) for fitting itself to the symmetric curve, while the asymmetric curve rarely requires more than 100 epochs.

4-3. SAMPLE RATE

As long as we are concerned with uniformly samples curves, it appears that the sample rate does not have much influence on the performance network. We tested at 30, 50 and 100 samples. Generally, the higher the sample rate, the longer the training algorithm attempts to further tweak the curve.

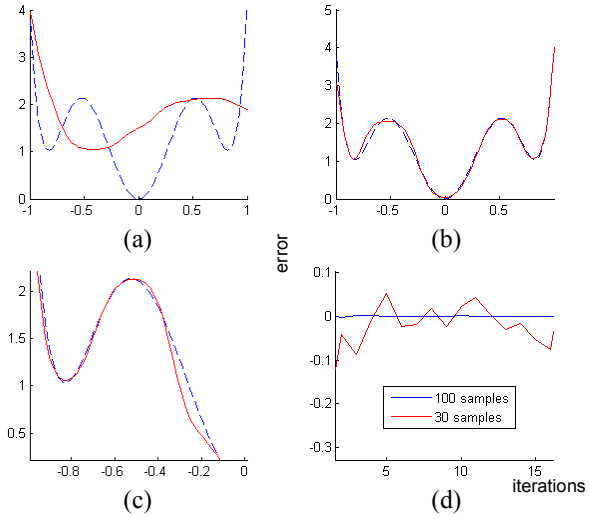

Figure 4 *– Approximation using 30 sample points with 6 neurons (a), 7 neurons (b), and 8 neurons (fragment, c). The error for sampling at 30 and at 100 points is also shown (d).*

The experiment showed that feeding about 50 unique, uniformly distributed samples for the symmetric 6th order curve causes the algorithm to take anywhere between 25 and 40 epochs, but if the number of samples is doubled, the number of epochs greatly exceeds 1000. Just as in the first experiment, this could be done with a hidden layer of 6 neurons, regardless of the sampling rate.

There is a caveat, though, as figure 4 shows. If the number of samples drops below a certain threshold, for the 6th degree symmetrical polynomial this threshold lies somewhere between 20 and 30, the network will not have enough to data to be able to properly generalize the curve.

The issue of sampling rate is related to extrapolation, as discussed in the next section, because sampling at a lower sampling rate causes more gaps, and may miss certain features of the curve altogether.

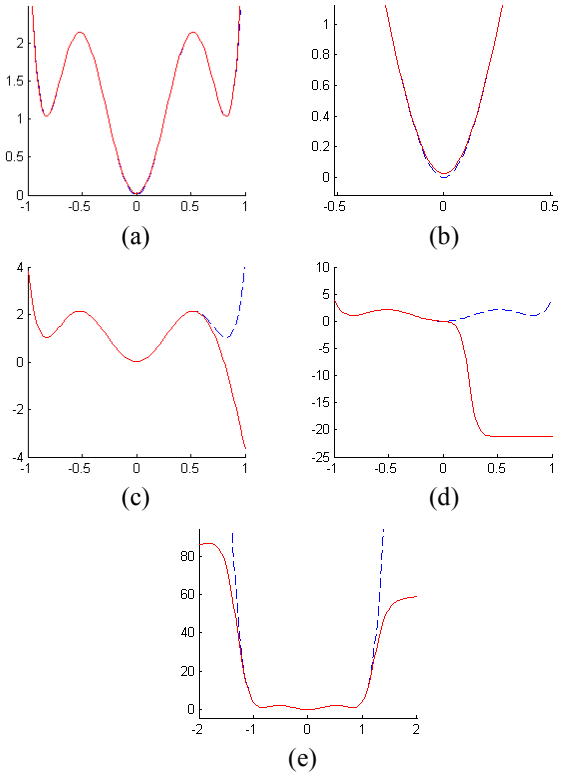

Figure 5 *– Extrapolating a sixth degree symmetric curve; missing the middle section from the input data (a, b), missing parts at one end of the data (c,d), and the behavior of the curve on a larger range than the [-1,1] used for training (e).*

This is worsened further by the fact that 20% of the samples were used for testing and validation. Creating separate sets for testing and validation improved performance in two ways. First of all, removing data points from the original data crippled the uniform nature of the input. Creating an extra data set for testing and validation ensures that this crippling will not occur. Secondly, and

more obviously, all of the input data can now be used for training, and none of it has to be sacrificed for.

Note that we have only shown the results for the symmetric 6th order curve. Behavior changes are more dramatic when symmetry (or the lack thereof) comes into play. The effects on the asymmetric curve of changes in sample rate, especially at the lower rates, were far more erratic and caused us to have to discard these measurements. To get meaningful results, this particular part of the experiment would have to be redone with a separate validation and test set, and larger input sets.

4-4. Extrapolation

The results for the asymmetric and symmetric curves of the same order are quite different. This was also the case for the sample rate, and given the similar nature of the problem, this could be expected. Cutting half the samples (e.g. the upper 25) causes the curve fitting to go completely astray for the asymmetric case. This is not the case for the symmetric curve, as the results show.

Figure 5 shows the resulting curves for different extrapolations of the symmetric curve. Again, the curve was fitted using the minimum of six neurons determined in the first paragraph of this section. If the middle 10 points (out of 50) are omitted, there really doesn't seem to be any problem.

Omitting the upper 10 points causes problems, probably because it just misses the local minimum, and there is nothing to hint the neural network that it should be going back up (figure 5c). Omitting half of the points on one site makes matter even worse, as the downward slope just before the end of the data causes the network to continue this downward trend, rather than going back up (the fact that exactly 50 points were taken will not have helped, as the (0,0) point is not sampled, meaning the last sample still has a derivative f'(x) to be less than 0.

Looking at the behavior of the network completely outside the trained [-1,1] range, we see that even though neural networks seem to have less trouble with symmetric curves, this symmetry is not retained outside of the trained area.

Figure 6 shows the results for the asymmetric sixth degree curve. Performance is sometimes equal to, but usually worse than for a symmetric curve. One thing that is obviously different is the fact that the effects are much less local, and the curve is effected even in areas still available in the source. This probably also explains why fewer neurons had more trouble with the asymmetric curve at the edges, as shown in paragraph 4.2.

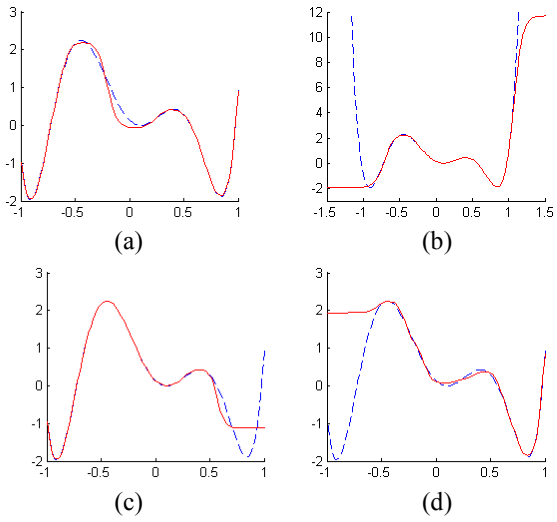

Figure 6 *– Extrapolating a sixth degree asymmetric curve; missing the middle section from the input data (a),missing parts from the upper (c) and lower (d) end of the data, and the behavior of the curve on a larger range than the [-1,1] used for training (b).*

5. Conclusion and Discussion

The first thing that occurs to the novice neural network enthusiast is the low number of neurons that is required for a decent simulation of a curve. One may conclude that the properties of the curve directly influence the requirements on the network. As such, one network will not fit all situations, and careful tweaking and experimentation is required to find a network that fits a particular situation best.

When running the experiments, the outcome wasn't exactly the same every time we ran it[†]. This may be partly contributed to random factors in the training algorithm. Another important issue is that samples are randomly extracted from the original dataset, to be used as test and validation data. This causes to the dataset used for training to consist of a different subset of the original input dataset each time it is run.

The set-up of the experiment was as such that only one thing was changed at a time, to isolate their effects. This, of course, overlooks the situation where two variables are dependent, although we believe that if there are any differences, that they will be small. Thus, our conclusions should hold.

The next logical step would be to redo all experiments in a multi layer perceptron feed forward network. Expected is that the total amount of neurons in the whole network can be decreased, whilst keeping equal or better performance.

[†] Apart from the causes described in the text, MATLAB would sometimes cause erroneous training due to a bug in the Neural Network Toolkit, causing it to accept its initial weights and biases. These runs have been removed from the experiment.

REFERENCES

[1] Goldengem
http://www.goldengem.co.uk

[2] Roel Smits and Louis Ten Bosch, *"The single-layer perceptron as a model of human categorisation behaviour"*, *Speech Hearing and Language: work in progress 1996, Volume 9,* 1996
http://www.phon.ucl.ac.uk/home/shl9/contents.htm