An Approach to Evolvable Neural Functionality

Niccolo Capanni, Christopher MacLeod, Grant Maxwell
The Robert Gordon University, Aberdeen AB10 1FR, U.K.

Abstract. This paper outlines a neural model, which has been designed to be flexible enough to assume most mathematical functions. This is particularly useful in evolutionary networks as it allows the network complexity to increase without adding neurons. Theory and results are presented, showing the development of both time series and non-time dependent applications.

Introduction

Research into Artificial Neural Networks (ANNs) has resulted in a diverse range of neuron models that have improved network functionality and expanded applications. Improvements in training methodologies have further increased the possibilities and this has spurred extensive work on the intricacies of improving training time and network robustness.

Examples of the resulting innovative neuron models include Radial Basis, Leaky Integration, Non-linear and Spiking types [1]. Despite this, most widely used ANNs operate on a variation of the classical neural (Continuous Perceptron) model.

Our research team has produced a new neural model based on the idea that a neural unit should be flexible enough to fulfil any differentiable mathematical function required of it [2]. This model is a logical extension of the Perceptron and is particularly useful in evolutionary and control applications.

Basic Power Series Neuron

The most common artificial neural models in current use are those developed from the original McCulloch-Pitts neuron. Ignoring the squashing or activation function, which normalises the output, the activity of this neuron is given by:

$$O = \sum_{i=1}^{n} x_i w_i$$

Where n is the number of inputs, x_i is an input and w_i is the corresponding weight.

For a two input neuron, with input x associated with weight b and input y associated with weight c (as illustrated in figure 1), the activity could be written as:

$$O = bx + cy$$

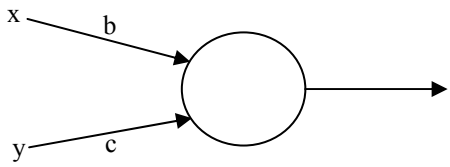

Figure 1. A simple neuron

This, of course, corresponds to a linear separator [3].

We can model any continuous function using an infinite Power Series [4] (for example a Taylor series):

$$f(x) = \alpha x + \beta x^2 + ... + \delta_n x^{n-1}$$

This is the basic series, which is given in most references. However, it can be extended to any number of variables (and hence any number of dimensions). For example, in two dimensions, the series is:

$$O = b_1 x + c_1 y + b_2 x^2 + c_2 y^2 + b_3 x^3 + ..$$

Notice that the first two terms are the same as in the first equation. This could correspond to a two-input neuron with inputs x, and y and weights b_n, c_n. A three input version of this neuron is shown in figure 2.

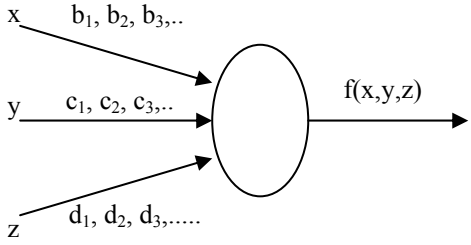

Figure 2. A polynomial neuron

More generally, for γ variables and an χ order series:

$$f(\overline{\sigma}) = \sum_{n=1}^{\chi} \sum_{m=1}^{\gamma} \alpha_{n,m} \sigma_m^{n-1}$$

Which is a non-linear separator. This can also be expanded to include input product terms (e.g. **wxy**) [5], which can enclose areas as discussed below.

Separators in the Second Order Case

To illustrate some of the attributes of higher order separators, let us first consider the second order case of a two input neuron with inputs labelled i_n and weights ω_n.

$$S = \omega_0.i_0 + \omega_1.i_1 + [\omega_2.i_0^2 + \omega_3.i_1^2]$$

Figure 3 shows how a single second order neuron can separate areas requiring many linear separators.

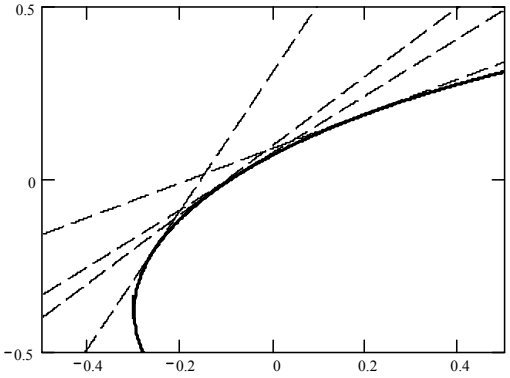

Figure 3. Second order separator

Figure 4 shows in a three-dimensional plot how the neuron can form a separator which can enclose (or exclude) a particular area.

Note that to reduce sensitivity in the neuron, it is often necessary to divide the higher power terms by their factorial.

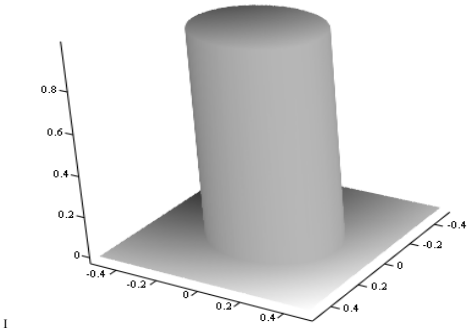

Figure 4. More complex second order separator

In all of these models, the weights of the second powers can evolve to "0", and the Perceptron emerges.

Separators in the Third Order Case

The decision surface can be further complicated through expansion of the power series. The symmetrical limitation of the 2nd order case can be removed by adding a 3rd order as shown in figure 5.

$$S = \omega_0.i_0 + \omega_1.i_1 + \omega_2.i_0^2 + \omega_3.i_1^2 + [\omega_4.i_0^3 + \omega_5.i_1^3]$$

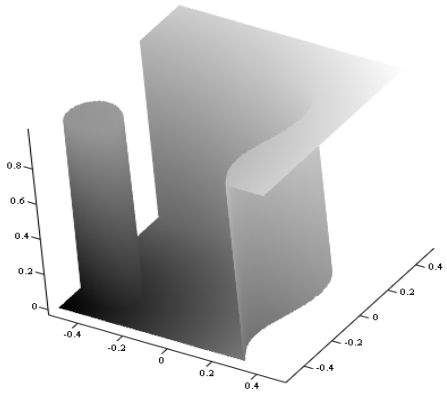

Figure 5. 3D Decision surface of 3rd order neuron

Addition of higher powers increases the complexity of the decision surface and allows greater separation and isolation of decisions.

More Complex Cases

The addition of higher orders of the power series is not the only method of improving the unit functionality. The previous neurons had no interaction between the inputs. However, if interaction is allowed, even more complex behaviour exists (as in sigma-pi units).

$$S = \omega_0.i_0 + \omega_1.i_1 + \omega_2.i_0^2 + \omega_3.i_1^2 + [\omega_4.(i_0^2.i_1) + \omega_5.(i_0.i_1^2) + \omega_6.(i_0.i_1)^2]$$

A 2nd order expansion is shown, with the complex expansions enclosed within brackets.

The addition of these product terms allows greater flexibility in the decision surface without continually expanding the power series. It also introduces a greater element of asymmetrical separation and isolation of distinct regions through the interaction of the inputs. A 3rd order complex neuron produces a separator as shown in figure 6.

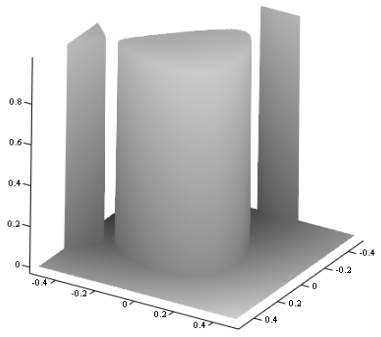

Figure 6. 3D decision surface 3rd order incomplete complex neuron

Applications

Allowing neurons to use higher powers of their inputs allows smooth separators as shown in figure 3. This improves generalisation in the network, although for higher orders generalisation decreases again [5].

These neurons are particularly useful in control systems in a similar way to radial basis units as they can take complex continuous forms without the need for large networks.

In evolutionary networks they allow the complexity of the network to increase by adding extra orders without adding new units to the network structure.

Finally, they can be used with Taguchi Method training [6] by training the first order initially and adding and training each additional order thereafter for a more accurate response.

Some Typical Results

When used with evolutionary training methods, the neurons allow us to reduce the number of epochs required to reach a solution as shown in figure 7.

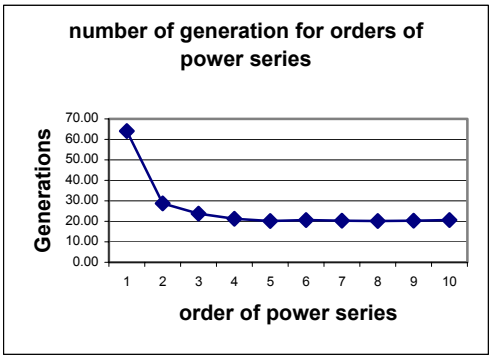

Figure 7. Reduction of training epochs

This network is three layered, consisting of 25 inputs and 60 neurons configured for character recognition. One can see that there is little point in introducing orders above the third. Although the training epochs decrease, the computational power required for training increases - in the case of the three order neuron, by three times. However, there is still a net improvement in training time.

As mentioned above we can also train one order of the neuron at a time using an evolutionary algorithm.

When used in a standard pattern recognition system (of the same type mentioned in relation to figure 7), the use of the higher order neurons allows the system to operate with fewer units, as shown in figure 8.

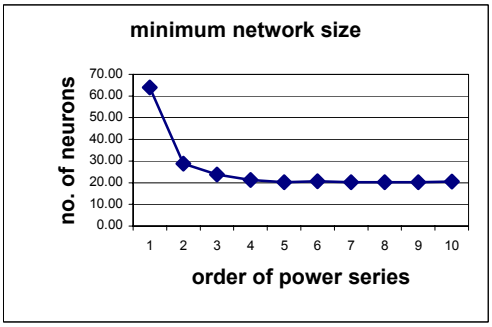

Figure 8. Number of units required in character recognition system

One can see in both cases, that above the fifth order, performance shows little improvement. Indeed there may be disadvantages in using too many orders [5]. In this case, the reduction in number of neurons offsets the increase in multiply and accumulate instructions required for a more complex network.

Future Work - Time Series Models

A time response can also be modelled with another Power Series

$$f(t) = a + bt + ct^2 + ...etc$$

Where t is the time variable. There is no evidence that the temporal properties of neurons are this complex. As a result, it is often easier to simply make an evolvable (or trainable) time decay.

$$f(t) = ae^{bt}$$

A squashing function may be applied to this if necessary, as can another time series representing a delay (refractory period) of the neuron's output.

This response can be multiplied with the power series described earlier to provide a neuron that is capable of mimicking any differentiable function of time.

Conclusions

The neuron described above may prove useful in several application areas including control systems and evolutionary systems. Its principal asset is that it allows the network to evolve with a wide variety of behaviours from a small number of neurons.

References

1. Arbib, M.: The Handbook of Brain Theory and Neural Networks, The MIT Press (1998)
2. MacLeod, C., McMinn, D., et al.: Evolution by devolved action: towards the evolution of systems. In appendix B of: McMinn, D.: Using Evolutionary Artificial Neural Networks to Design Hierarchical Animat Nervous Systems, PhD Thesis, The Robert Gordon University, Aberdeen, UK (2002)
3. Khanna, T.: Foundations of Neural Networks, Addison Wesley (1990)
4. Croft, A., Davison, R., Hargreaves, M.: Engineering Mathematics, Addison-Wesley (1996) 418-440
5. Bishop, C. M.: Neural Networks for Pattern Recognition, Oxford (1995) 9-14.
6. MacLeod, C., Maxwell, G. M.: Using Taguchi Methods to Train Artificial Neural Networks," AI Review, **13,** 3, Kluwer (1999) 177-184