

Robosoup

nourishing technology

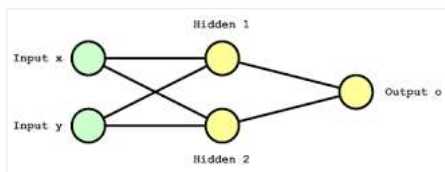
[Blog](#)
[Frameworks](#)
[Contact](#)

Sunday, 7 September 2008

The Sigmoid Function in C#

In my last post, I discussed the Single Layer Perceptron and its limitations regarding solving linearly non-separable problems. If we are to solve this type of problem, then we need a more sophisticated network, and that means adding layers. However, if we do that, how do we train the network?

Until now, we have used the error observed at the output neuron to adjust the weights between it and the input layer. But, what do we do if we add another layer in between, a hidden layer? Well, we still need to feed this error back through the network in order to adjust the weights, and this is a problem.

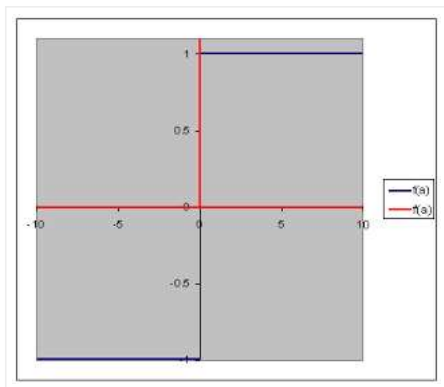


Currently, a neuron is either “on” or “off”, which is defined by the threshold activation function we are using.

$$f(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

```
public int Function(double x)
{
    return (x >= 0) ? 1 : -1;
}
```

When graphed, it looks like this, with the blue line representing the function and the red line its derivative, which at zero is infinity (not very useful).



Any adjustment we make to the weights feeding the hidden layer is going to be rather coarse, so we need to smooth this out. We can do that by replacing our simple threshold function with something better. How about this? This is known as a sigmoid function.

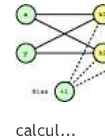
$$f(x) = 2 / (1 + e^{-2x}) - 1$$

```
public double Sigmoid(double x)
{
    return 2 / (1 + Math.Exp(-2 * x)) - 1;
}
```

And its derivative.

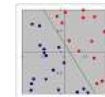
$$f'(x) = 1 - f(x)^2$$

Popular Posts



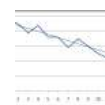
[Training Neural Networks Using Propagation in C#](#)

In my previous post, I show multi layer Perceptrons can be used to solve linearly non-separable problems. In that example, we calculated...



[The Single Layer Perceptron](#)

This is the first in a series of posts which I will write about the evolution of neural networks. I start with the most simple...



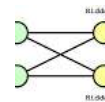
[Linear Regression in C#](#)

When looking at time series data, such as a stream of prices, it is often useful to establish a general trend and represent it with a linear regression...



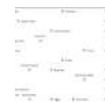
[Flocking Boids \(C#\)](#)

In this post, I shall demonstrate an example of swarm intelligence based on the Boids model. In my ecosystem, there are two different types of boids...



[The Sigmoid Function in C#](#)

In my last post, I discussed the Single Layer Perceptron and its limitations regarding solving linearly non-separable problems. If we use a sigmoid function...



[C# Self Organising Map \(SOM\)](#)

In this post, I shall be discussing Self Organising Maps (SOM), which are also known as Kohonen maps or topographical maps. Until now...



New: Top Business Trends 201

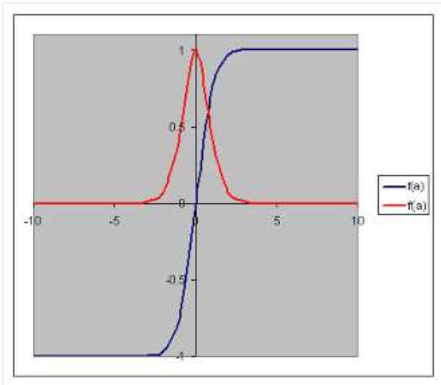
Top 10 Business Intelligence Trends New for 2016. Read Free Whitepaper!



www.tableau.com

```
public double Derivative(double x)
{
    double s = Sigmoid(x);
    return 1 - (Math.Pow(s, 2));
}
```

When graphed, it looks like this.



Now, this looks much more useful. There is a smooth and continuous transition between -1 and 1, and the derivative goes nowhere near infinity. Whilst this function is very popular with many neural net developers, I would like to make one further refinement. I like my neurons to be working in the range of 0 to 1, and to do that I need to modify the function slightly.

$$y = \frac{1}{1 + e^{-x}}$$

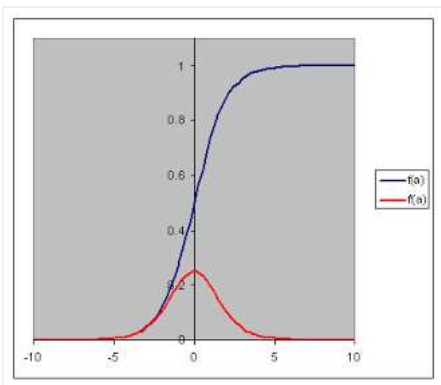
```
public double Sigmoid(double x)
{
    return 1 / (1 + Math.Exp(-x));
}
```

And its derivative.

$$f'(x) = f(x)(1 - f(x))$$

```
public double Derivative(double x)
{
    double s = Sigmoid(x);
    return s * (1 - s);
}
```

When graphed, it looks like this.



Now, that is perfect, and it has the added benefit of being computationally less demanding, which will be important for larger networks.

In my next post, I will show you how to use the sigmoid activation function to build a Multi Layer Perceptron, trained with the Back Propagation Algorithm.


I welcome comments on this post, and suggestions for future posts.

John

Posted by John Wakefield at 07:48:00


 Recommend this on Google

Labels: AI, C#, Neural Network, Perceptron



Effective Dashboards

Create Effective Dashboards
Now. 6 Best Practices. Free
Whitepaper!



[Newer Post](#)

[Home](#)

[Older Post](#)

Simple template. Powered by [Blogger](#).