Universal Approximation Theorem

By
Al Bernstein
4/28/2021
http://www.metricmath.com
al@metricmath.com

Introduction

The universal approximation theorem states that an arbitrarily complex neural network, with squashing functions as activation functions, can approximate any continuous function to any degree of desired accuracy. A squashing function is a function that has a finite range – for example $-1 \leq tanh(x) \leq 1$. For an n^{th} order polynomial $P_n(x)$, for $n > 0 \Rightarrow \lim_{x \to \pm\infty} P_n(x) \to \pm\infty$, so polynomials are not squashing functions. A polynomial could be squashed i.e. defined to be within a finite range. For example, a polynomial could be defined in a given interval over x and equal the endpoints outside of that interval. This writeup gives a simple demonstration of the theorem. For practical purposes, the function to be approximated will have a finite domain and range. The approach is to perform a forward calculation – compute the weights and biases analytically and show that it can approximate any finite continuous function. A second approach is presented using the soft exponential activation function (SEU).

Neural Networks implement a function. that map an input to an output. Figure 1 shows a single layer neural network structure with one input and one output.

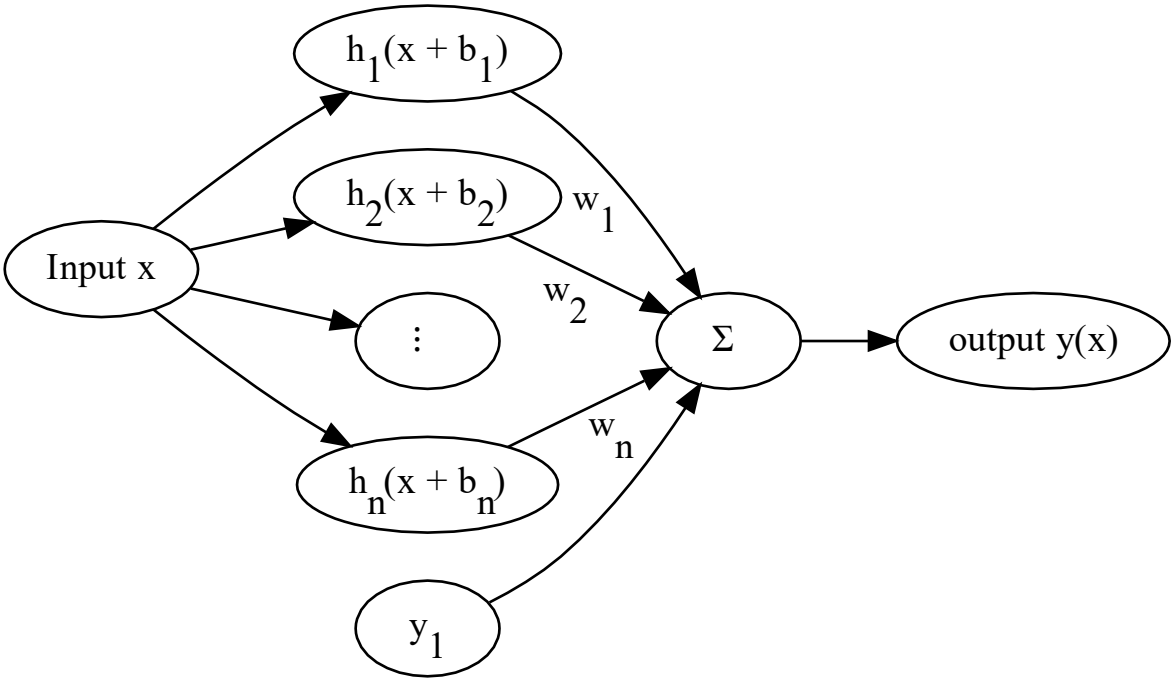

Figure 1

where

$h_i(x)$ are the activation functions
w_i are the weights
b_i are the bias terms.
y_1 is a bias added to the output.

Equation (1) gives the equation for the neural net in Figure 1.

$$y(x) = \sum_i w_i h_i(x - b_i) + y_1 \tag{1}$$

Usually, for a given layer, the activation functions are the same.

Simple Demonstration of the Theorem

We can try to build up the network in Figure 1 to fit any continuous function within a finite interval. We use the rectified linear unit $ReLU(x)$ as the activation function. Figure 2 shows how to add two $ReLU$ functions to produce a line within a given interval i.e. a squashed line.

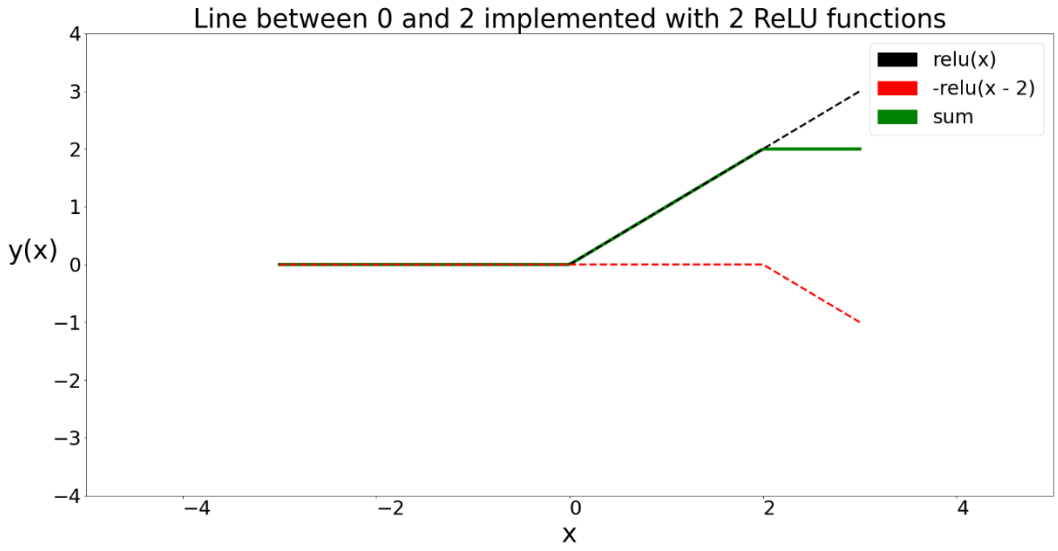

Figure 2

The black dotted line is given by equation $h(x)$ and the red dotted line is given by $-h(x-2)$. The solid green line is the sum of the dotted black and green lines and is given by equation (2).

$$y(x) = h(x) - h(x-2) \tag{2}$$

For $x \geq 2$, the slopes cancel and become 0 with a y – intercept of 2 at $x = 2$.

Now we can simply break any function $f(x)$ into intervals in x, compute the slopes over those intervals and use equation (1) to approximate the function over those intervals. Note that the y-intercept is calculated for the first interval only and is described further below. The function to be approximated - y - is broken into points. The x and y components are shown in equation (3).

$$[(x_0 \quad y_0), (x_1 \quad y_1), \cdots, (x_i \quad y_i)\ ] \tag{3}$$

The slope over each interval is computed as shown in equation (4).

$$m_{i-1} = \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \tag{4}$$

The function is defined as shown in equation (5)

$$f(x) = \begin{cases} m_0 x + b_0, & x_0 \leq x \leq x_1 \\ m_1 x, & x_1 \leq x \leq x_2 \\ \vdots & \\ m_{i-1} x, & x_{i-1} \leq x \leq x_i \end{cases} \tag{5}$$

Notice that the intervals connect – for example x_1 is the end of the first interval and beginning of the second interval. Because we are requiring $f(x)$ to be continuous $\Rightarrow f(x_1)$ has to be the same when approached by the left or right – i.e.

$$f(x_1)^- = f(x_1)^+ \tag{6}$$

After the first point the y – intercept is not needed because the previous line ends where the next line begins.

To illustrate, suppose we want to approximate x^2 over $-2 \leq x \leq 2$

We can start out with three points as shown below.

$[(-2 \quad 4), (0 \quad 0), (2 \quad 4)\ ] \Rightarrow$

Computing the slopes $\Rightarrow$

$$m_0 = \frac{0 - 4}{0 + 2} = -2, \qquad -2 \leq x \leq 0$$

$$m_1 = \frac{4-0}{2-0} = 2, \quad 0 \le x < 2$$

$$m_2 = 0, \quad x \ge 2$$

To produce these slopes at the corresponding x' s ⇒

$$y_r(x) = 4 - 2h(x+2) + 4h(x) - 2h(x-2) \tag{7}$$

where

$y_r(x) \equiv$ reconstructed or approximating function

Going through equation (7) ⇒

At $x = -2 \Rightarrow f(x) = 4$

Since $b_0 = 4$ and $m_0 = -2 \Rightarrow w_0 = -2 \Rightarrow$

$$y_r(x) = 4 - 2h(x+2)$$

At $x = 0$

Since $m_1 = 2$ and $m_0 = -2$, the slope at $x > 0 \Rightarrow w_1 = 4$

So

$$y_r(x) = 4 - 2h(x+2) + 4h(x)$$

For $x \ge 2, m_2 = 0$, and $m_1 = 2 \Rightarrow w_2 = -2$

$$y_r(x) = 4 - 2h(x+2) + 4h(x) - 2h(x-2)$$

Figure 3 shows equation (7) plotted against x^2 for 200 points between -3 and 3.

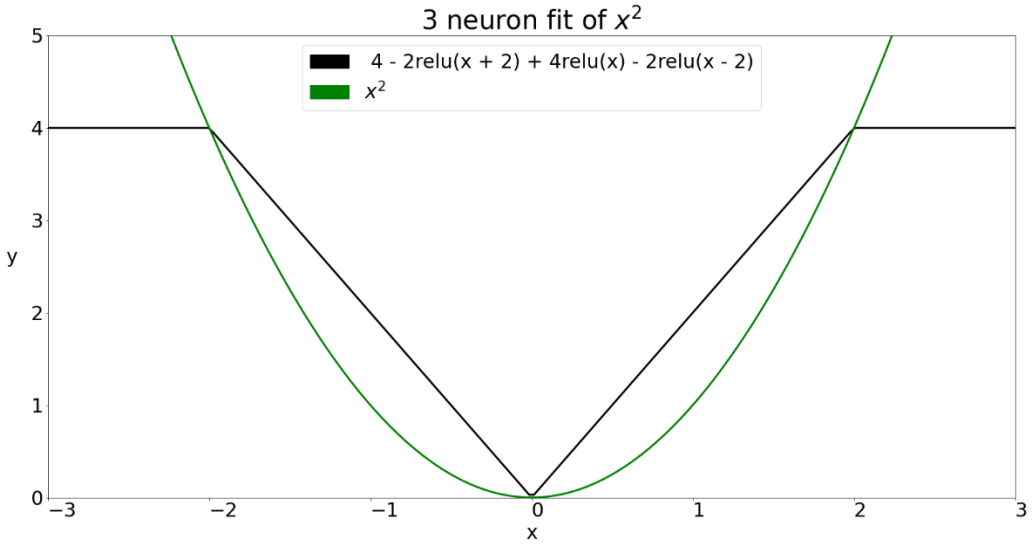

Figure 3

Figures 4, 5, and 6 show 7, 11, and 20 neurons respectively.

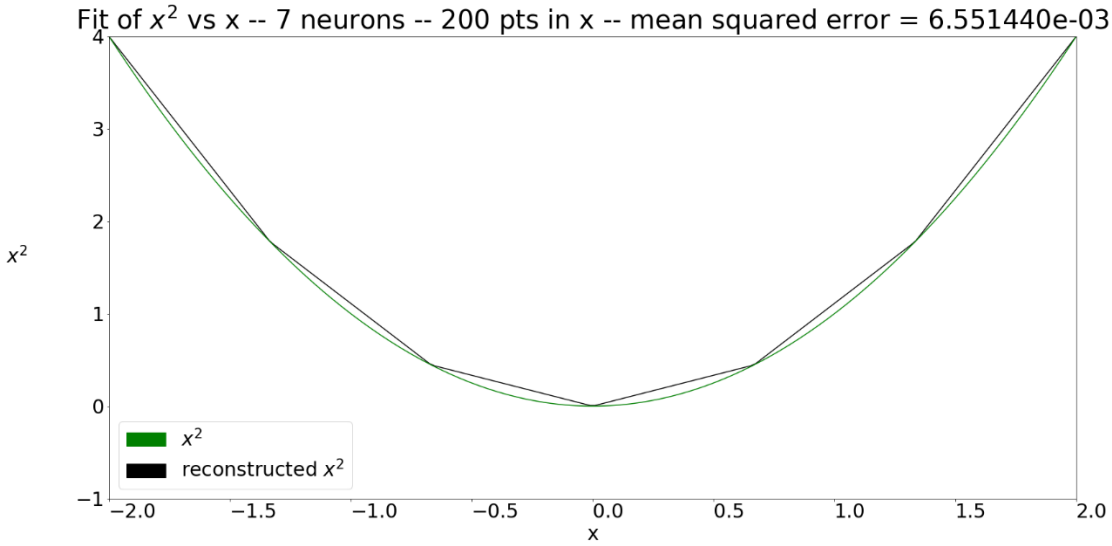

Figure 4

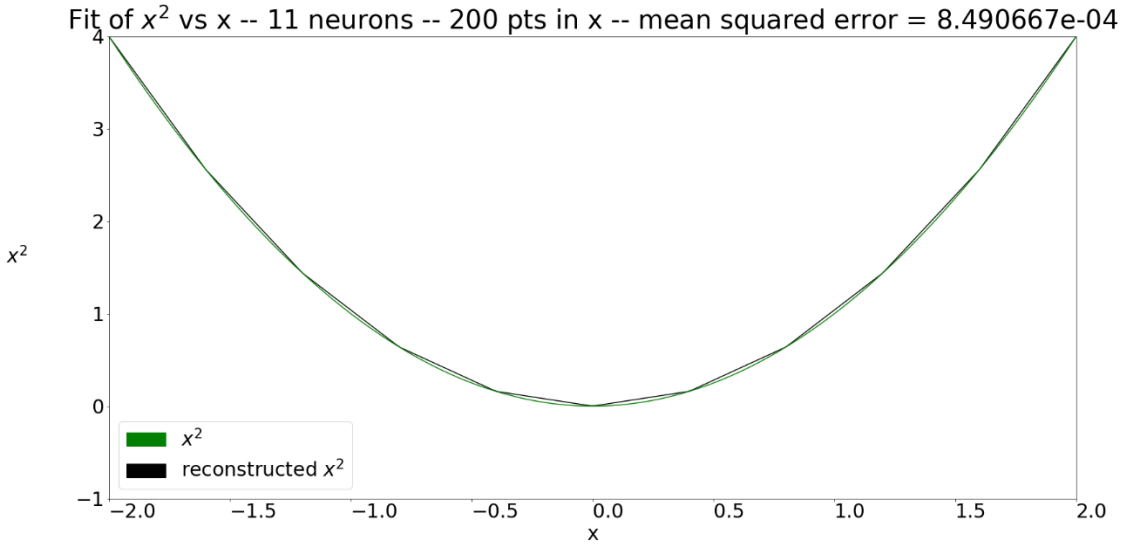

Figure 5

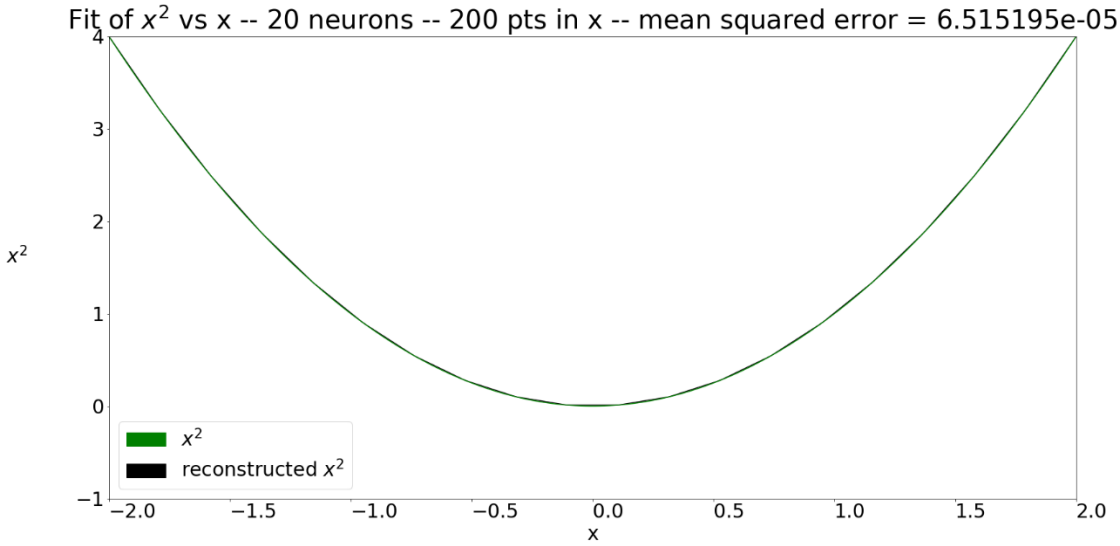

Figure 6

As the number of neurons goes up, the mean squared error goes down which shows that the desired accuracy can be obtained – within numerical precision – by adding more neurons.

Another Function for Comparison

Figure 7 shows another polynomial to show how the approach works with a different function. This polynomial is given by equation (8).

$$\frac{1}{2}(5x^3 - 3x) \;\; [-1 \leq x \leq 1] \tag{8}$$

Outside of the $[-1 \leq x \leq 1]$ range the neural net reconstruction is the value of the function at the endpoints of the range - ± 1 in this case. Note: that the mean square error is computed inside the interval of interest - $[-1 \leq x \leq 1]$.

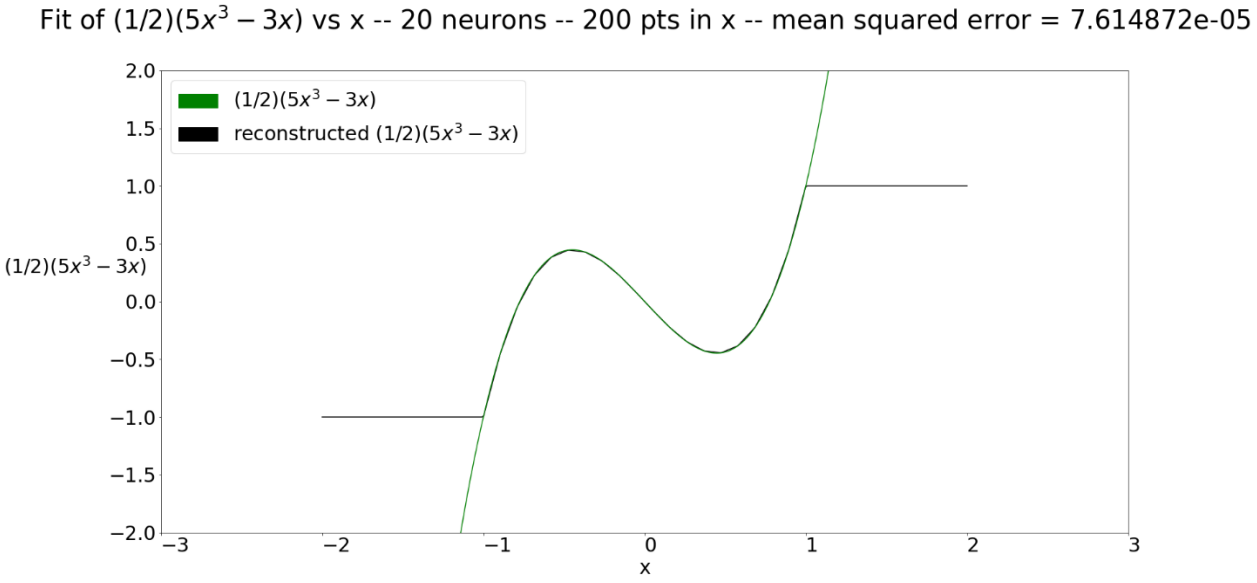

Figure 7

Normalizing the Range and Domain of f

The range of a function – y – values can be normalized by dividing all the values by the maximum value. In machine learning, it is helpful to normalize the input in the domain - x – as well to bound the search space. The x values can be mapped into and out of a normalized range using Equation (9) - x is in the interval $[a \quad b]$ and x_1 is in the interval $[c \quad d]$. The ratio of Δx and Δx_1over their respective intervals remains constant i.e. $^1/_2$ or $^1/_4$ etc. ⇒

$$\frac{\Delta x}{interval} = \frac{x - a}{b - a} = \frac{x_1 - c}{d - c} \equiv constant \Rightarrow$$

$$x_1 = \frac{d-c}{b-a}(x-a) + c \tag{9}$$

The normalization bounds the w_i – range variables - into a range of $[-1 \quad 1]$. The domain variables -b_i-can be bounded into an interval the user feels would get the best results and this could also be a range of $[-1 \quad 1]$.

Normalizing Example

For an example, the approximation function is x^2 for $-2 \leq x \leq 2$ and 2 as shown in Figure 8.

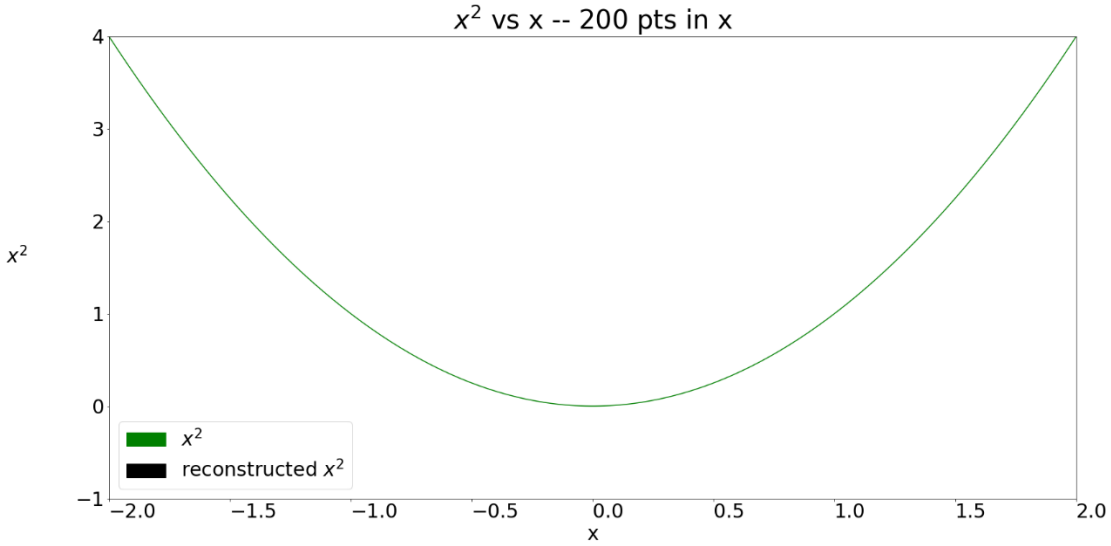

Figure 8

We will normalize it, then fit it, and then de-normalize the fitted function to show that it approximates the original function.

Figure 9 shows the function in Figure 8 after normalization in x and y.

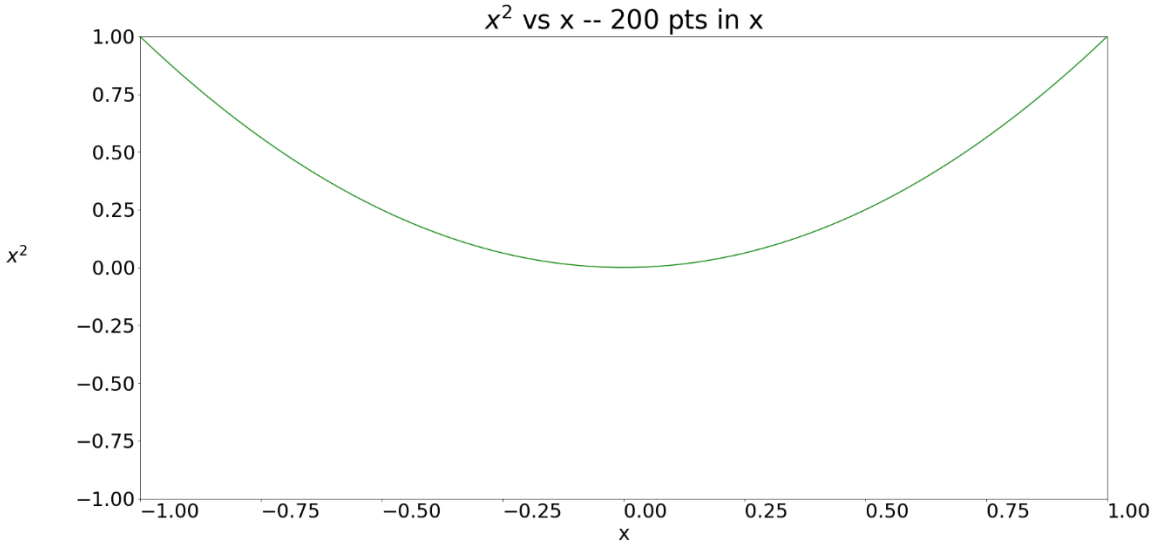

Figure 9

Figure 10 shows a five neuron fit to the normalized function. Normalize y by dividing by the maximum value – in this case by 4.0 – and normalize x using equation (9) with $[a \quad b] = [-2 \quad 2]$ and $[c \quad d] = [-1 \quad 1]$

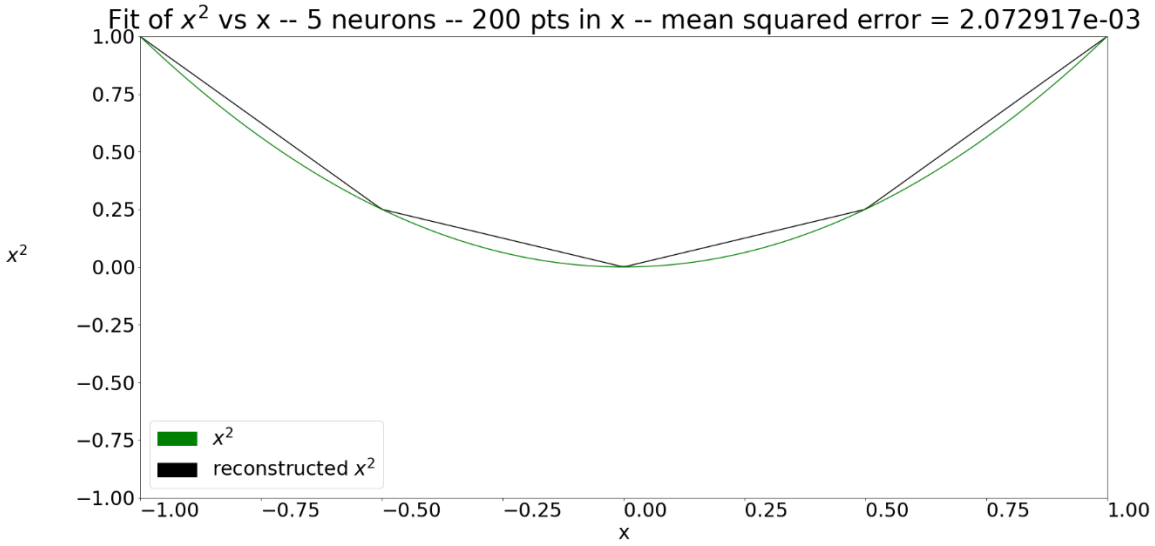

Figure 10

Now de-normalize the data from Figure 10 – for y multiply by 4.0 and x using equation (9) with $[a \quad b] = [-1 \quad 1]$ and $[c \quad d] = [-2 \quad 2]$ to get the five neuron approximation to the original function as shown in Figure 11.

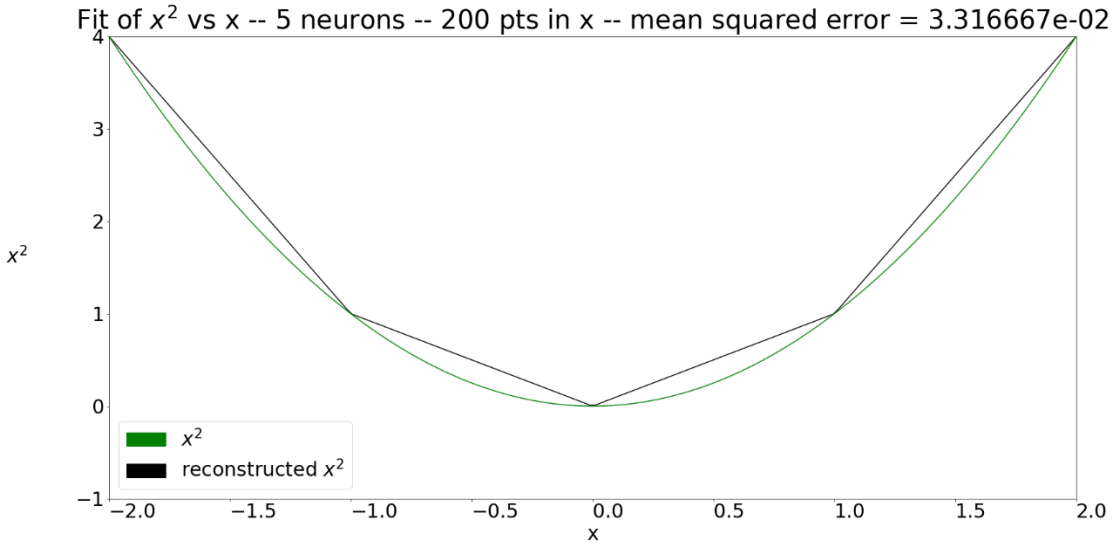

Figure 11

Figure 11 shows that the fit is reasonable given the number of neurons. The importance of this process is that normalization sets boundaries to limit the search space of the optimizer.

Another Approach to Fit Polynomials

The soft exponential activation unit function[1] is shown in equation (10).

$$SEU(\alpha, x) = \begin{cases} \dfrac{-ln\big(1-\alpha(x+\alpha)\big)}{\alpha}, & \alpha < 0 \\ x\,, & \alpha = 0 \\ \dfrac{e^{\alpha x}-1}{\alpha} + \alpha, & \alpha > 0 \end{cases} \tag{10}$$

This activation function is more suited to working with multiplications and polynomials because it can be an e^x, x, or $ln(x)$, with the proper choice of α.

For $\alpha = -1 \Rightarrow SEU(-1,\ x) = ln(x)$

[1] Godfriey L. and Gashler M. , "A Continuum among Logarithmic, Linear, and Exponential Functions, and Its Potential to Improve Generalization in Neural Networks", Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015) - Volume 1: KDIR, pp 481 – 486.

For $\alpha = 0 \Rightarrow SEU(0,\ x) = x$

For $\alpha = 1 \Rightarrow SEU(1,\ x) = e^x$

Note: the above functions have to be bounded in a finite domain and range.

Figures 12 show a neural configuration to implement a polynomial element and the $SEU(\alpha, x)$ implementation.

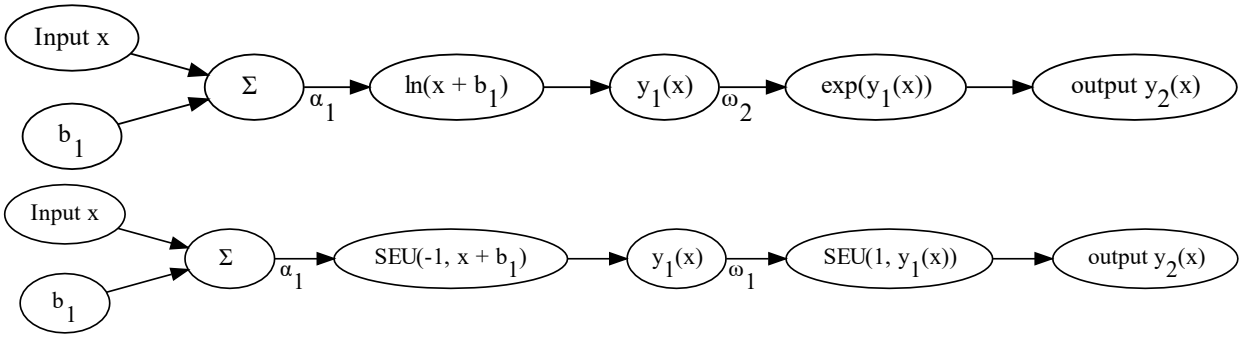

Figure 12

The equation that Figure 12 implements is shown in equation (11)

$$y_2(x) = \omega_1 \exp[\alpha_1 ln(x + b_1)] = \omega_1 \exp[ln(x + b_1)^{\alpha_1}] = \omega_1 (x + b_1)^{\alpha_1} \tag{11}$$

Any continuous function can be approximated to the desired accuracy by using Taylor series around a given point – a - as shown in equation 12.

$$f(x) = f(a) + \sum_n \frac{f^{(n)}(a)}{n!}(x - a)^n \ \ [c \le x \le d] \tag{12}$$

Any continuous function can be approximated in a finite domain - $[c \le x \le d]$ - and range using a set of blocks – from Figure 12 - added together with different α_i coefficients and biases b_i as shown in equation (13).

$$f(x) = \sum_i \omega_i (x - b_i)^{\alpha_i} + y_1 \quad [c \le x \le d] \tag{13}$$