

# Discontinuous Piecewise Polynomial Neural Networks

**John Loverich**

JOHN.LOVERICH@GMAIL.COM

*N Infinity Computational Sciences*

*638 Beauprez Avenue*

*Lafayette CO 80026*

**Editor:** Somebody

## Abstract

An artificial neural network is presented where each link is represented by a grid of weights defining a series of piecewise polynomial functions with discontinuities between each polynomial. The polynomial order ranges from first to fifth order. The unit averages the input values from each link. A back propagation technique that works with discontinuous link functions and activation functions is presented. The use of discontinuous links function means that only a subset of the network is active for a given input and so only a subset of the network is trained for a particular training example. Unit dropout is used for regularization and a parameter free weight update is used. Better performance is obtained by moving from piecewise linear links to piecewise quadratic and higher order links. The algorithm is tested on the MNIST data set, using multiple autoencoders, with good results.

**Keywords:** artificial neural network, piecewise polynomial, discontinuous, high order, autoencoder

## 1. Introduction

Neural networks and their application to deep learning have become a focus of research due to the success of the algorithms on several types of problems (Mnih et al., 2015; Bengio, 2009; Hinton and Salakhutdinov, 2006). The goal of this work is to create a multi-layer artificial neural network using piecewise discontinuous polynomial functions. The motivation for this work comes from a search for a network that is: (1) easier to analyze than a network based on hyperbolic tangent type activation functions (2); higher order than the rectified linear network; (3) that takes advantage of high order discontinuous polynomial approaches which have had great success in the computational physics community (Hesthaven and Warburton, 2007). The algorithm starts with a simple technique for approximating functions, using piecewise discontinuous polynomials. Typically, discontinuous functions are avoided in artificial neural networks because gradient descent derivatives are not defined at the discontinuities. However, it is shown that the discontinuities act to break up the network into a series of sub networks. Gradient descent can be applied to each of these sub networks separately and good results can be achieved. Note that at least one discontinuity should be used to remain a universal approximator as described by Leshno et al. (1993). The discontinuous network tends to produce over fitted results in many problems. Overfitting can be reduced using the dropout regularization technique described by Hinton et al. (2012); Srivastava et al. (2014). In addition, a parameter free weight update

method as described by Schaul et al. (2013) is used. Lagrange polynomials are used to describe the piecewise polynomial functions and, as such, the weights of the polynomial are the actual value of the polynomial at specific locations. In this paper, the link is the important computational element, but this approach can just as well be applied to the unit. Piecewise polynomial approaches have been investigated in the CMAC architecture (Lane et al., 1992) and more recently the rectified linear unit (Nair and Hinton, 2010), has become a popular activation function which is piecewise linear. High order neural network using higher order weighting terms are described by several authors including Giles and Maxwell (1987); Shin and Ghosh (1992); Huang et al. (2004); Foresti and Dolso (2004); Fallahnezhad et al. (2011) and functional link artificial neural networks (FLANN) by Pao (1989); Patra and Kot (2002); Purwar et al. (2007), which is the approach used in this paper. Discontinuous neural networks have been discussed in many articles, especially with respect to recurrent neural network including Forti and Nistri (2003); Liu and Cao (2010); Gavalda and Siegelmann (1999) focused on convergence state estimation and stability and by Zhang et al. (2002) where a unique recurrent high order algorithm is derived for financial modeling, but where additional free parameters (weights) are added to the unit and a few simulations are performed with piecewise high order elements. There is very little work on multi-layer high order discontinuous polynomial networks. In this paper the algorithm is tested by performing simple curve fitting through the sine wave using a single link, and then classification with the Pima Indians data set and the MNIST data set. The MNIST test is performed using multiple autoencoders. The unique contribution of this paper is the application and development of novel algorithm, using discontinuous piecewise polynomial approximation, in a multi-layer neural network as well as the demonstration of it use with dropout (Hinton, 2002) and the parameter free weight update described by Schaul et al. (2013). The algorithm opens the possibility of using a variety of complicated, discontinuous elements in an artificial neural network using back propagation.

Section describes the algorithm used in this paper including backpropagation with discontinuities, weight initialization and link input range selection. In Section we demonstrate the algorithm on 3 problems: a simple 1D function approximation; the Pima Indians diabetes data set; the MNIST data set. In Section we conclude the paper.

## 2. Algorithm

Definitions used in this paper are defined in table 1. There are two natural ways to apply higher order approximations in artificial neural networks. The first is to replace the single weight at the link with multiple adjustable weights describing a more complicated link function - this is a functional link neural network (FLANN) as described by Pao (1989) and demonstrated by Patra et al. (1999). The alternative is to add adjustable parameters to the unit that describe a changing activation function, in this case adjustable parameters exist in both the unit and the link. In this paper we chose the FLANN approach as it can be written slightly more compactly while maintaining weights defined at the link.

The error correction algorithm used is backpropagation as described by Rumelhart et al. (1988) applied to a FLANN with a minor modification described in Section . The weight update rule is defined by using the parameter free weight update rule described by Schaul et al. (2013). A slight modification to this rule is that the maximum learning rate is set to 0.9. The network description is that of a standard feed forward network, see Figure 1,

Link	Is the connection between two units.
Sub Link	A link is split into sub links.
$\omega_i$	The $i^{th}$ weight of a sub link.
$\omega_{j,\alpha}$	the $\alpha^{th}$ weight of the $j^{th}$ link.
$B_i$	The $i^{th}$ basis function of a sub link.
$N_{in}$	The number of active links into a unit.
$N_{out}$	The number of active links out of a unit.
$N_p$	The number of Chebyshev-Lobatto nodes.
$x_i$	The input to the $i^{th}$ link.
$x_j$	The $j^{th}$ Chebyshev-Lobatto node.
$y_{mi}$	The measured output value for output link $i$
$y_{di}$	The desired output value for output link $i$
$F_i$	The output function for link $i$ .
$y_i$	The output for link $i$ .
$E$	The network error for a single input.
$t_{N_p}$	Time to completion using sub links with $N_p$ nodes

Table 1: Variable definitions used in this paper.

with input links and output links added. Labels for a network element are shown in figure 2. The main differences of the algorithm compared to a standard network (Rumelhart et al., 1988) are: (1) there are multiple weights per link; (2) No bias units are used; (3) the unit averages the input signal to produce an output instead of applying a more complex activation function; (4) input/output links are added which can be used to normalize and shift the data to the desired input/output range.

The weight function of a sub link is described by the following equation,

$$f(x) = \sum_i w_i B_i(x) , \quad (1)$$

with the basis functions given by the Lagrange polynomials

$$B_j = \prod_{0 \leq m \leq k, m \neq j} \frac{(x - x_m)}{(x_j - x_m)} .$$

The Lagrange polynomial  $B_j$  is useful because it has the value 1 at  $x = x_j$  and has the value 0 at all other  $x_i$ . The interpolation nodes are given by the Chebyshev-Lobatto nodes, these differ from pure Chebyshev nodes in that the end points of the domain are included. The Chebyshev-Lobatto nodes are given by,

$$x_j = -\cos\left(\frac{k \pi}{N_p - 1}\right) ,$$

where  $k$  ranges from 0 to  $N_p - 1$  and  $N_p$  is the total number of Chebyshev-Lobatto points. This means that in Equation () the value of the function at  $x_j$  is  $w_j$ . Using this approach we can easily limit the range of a polynomial interpolation by limiting the range of the weights

$w_i$ . For clarity we provide the polynomials for both a linear and quadratic interpolation. In the linear case (assuming  $x_0 = -1$  and  $x_1 = 1$ )

$$B_0 = -\frac{1}{2}(x - 1)$$

$$B_1 = \frac{1}{2}(x + 1)$$

In the quadratic case we assume  $x_0 = -1$ ,  $x_1 = 0$ ,  $x_2 = 1$  the basis functions are

$$B_0 = \frac{1}{2}x(x - 1)$$

$$B_1 = -(x + 1)(x - 1)$$

$$B_2 = \frac{1}{2}(x + 1)x$$

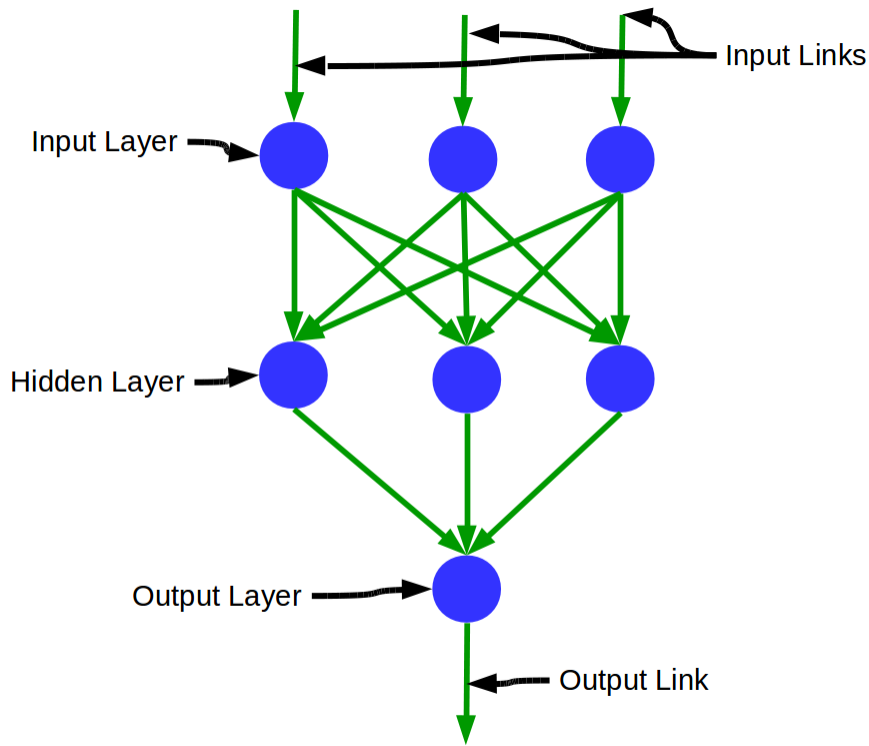


Figure 1: A standard feed forward network. Blue circles represent summing unit and green lines represent links where the signal passes from one neuron to the next along the direction of the arrow.

In addition to the high order weighting, discontinuities are used along with the piecewise polynomial approximation. This means that the function is different depending on the range of the input variable  $x$ . In particular we have the definition

$$F_i(x_i) = \begin{pmatrix} \text{if } [r_{min} \leq x < a_0] & f_{i0}(x_i) \\ \text{if } [a_0 \leq x < a_1] & f_{i1}(x_i) \\ \dots & \dots \\ \text{if } [a_n \leq x \leq r_{max}] & f_{in}(x_i) \end{pmatrix} \quad (2)$$

At the unit, the average of the incoming signals is computed. The unit could be a sigmoid, but it is not needed since the non-linearity is provided by the presence of at least one discontinuity, so only a simple average is used as the activation function of the unit.

$$g = \frac{1}{N_{in}} \left[ \sum_j^N F_j(x_j) \right]$$

The averaging is important because the function  $F_i$  only has a valid solution in a finite range so it is important to guarantee that any signal passed to  $F_i$  is within that range. This can easily be accomplished by choosing initial  $r_{min}$  and  $r_{max}$  correctly.  $N_{in}$  is the number of input signals (not the total number of input sub links).

The idea to use a discontinuous piecewise polynomial comes from a pair of units with multiple links between the units, see Fig. 3. Only a single link is active depending on the output value of the unit. Equation (2) can be described as a set of links between two units where only one link passes an output signal for a given input signal. As a result, the links are grouped together in a bundle (Equation (2)) and shown in figure 4. The standard link of a neural network described in this paper then consists of one or more sub links. The link function can now be thought of as a one dimensional grid with piecewise polynomial elements within each grid cell as shown in Figure 5.

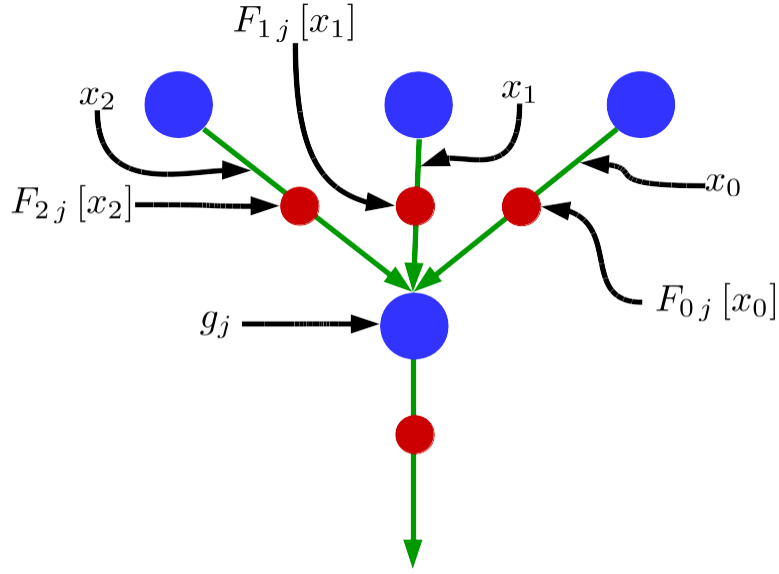


Figure 2: Zoom in of connected units.

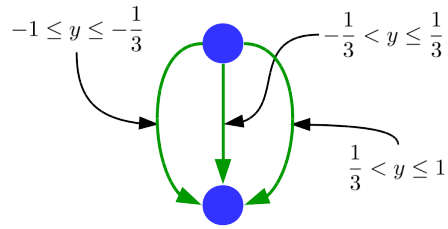


Figure 3: Two units are connected by several links. Each link is only active for a range of possible inputs. This links can be combined into a single link with multiple weights and discontinuities between sub links.

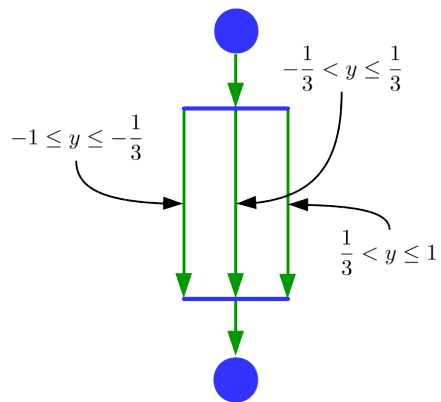


Figure 4: A pair of units connected by a single link with 3 sub links where only one sub link is active for a given input signal. This grouping of links is called a “bundle”. This is the simplified approach taken in this paper.

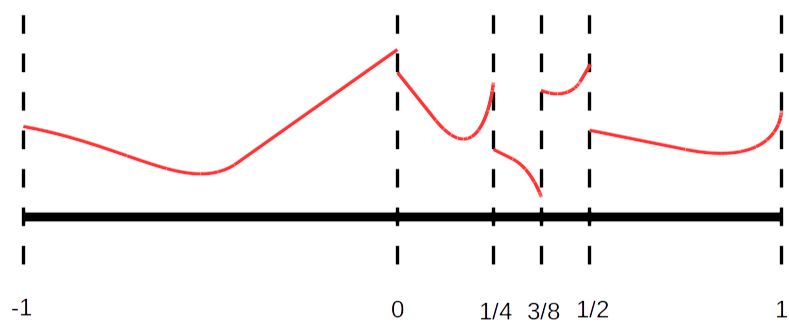


Figure 5: A link can be represented as a grid with a piecewise function valid on each of the ranges of the sub links. The link function is discontinuous at the ends of each range. Notice that in this link the sub link ranges are not equally spaced - and they don't need to be. For the purposes of this paper, all ranges are equally spaced.



## 2.1 Backpropagation with Discontinuities

A network using a link with at least two sub links has the following property: for a given input signal only a subset of the network is active. When training is performed, only the weights of the activated sub links are updated. Figure 6 shows a 3-layer network consisting of two links, each link with two sub links. Figure 7 shows this same network with all the possible paths that a signal can take from input to output. It is evident from Figure 7 that there are 4 networks where each network shares some weights with the other networks. It's easy to over train this type of network since each signal only effects a subset of the weights. To resolve over training the “dropout” regularization technique is used as described in Hinton et al. (2012).

The key to getting an algorithm working with functions that are discontinuous is a backpropagation algorithm that works for discontinuous functions. The implementation is incredibly simple and the only requirement is to remove idle sub links from the back propagation step. After a signal has been propagated through the network, back propagation is performed, but only on the subset of the network active for the input signal. This means that for each link, only one sub link is active, and so back propagation is performed only on that sub link, all other sub links in the link are ignored. When weights are updated, they are only updated in the links that fired during the forward step.

For clarity we include a description of the back propagation applied to this network, recall that backpropagation is only applied to the active sub network.

- Forward propagate input signal
- Record each sub link that is activated for the given input signal
- Back propagate error signal through active sub links
- Update weights of the active sub link
- Ensure that weights are within desired range,  $w_i = \min(w_{max}, \max(w_{min}, w_i))$
- Repeat with a new input signal

The error at the output of the network is measured as

$$E = \sum_i^{N_i} \frac{1}{2} (y_{m,i} - y_{d,i})^2$$

$$\frac{\partial E}{\partial y_{m,i}} = y_{m,i} - y_{d,i}$$

The derivative of a link output with respect to a link input is given by

$$\frac{\partial y_i}{\partial x_i} = \frac{\partial F_i}{\partial x_i}$$

Although,  $F_i$  is discontinuous at points, it is entirely continuous within the range of the derivative. If the derivative happens to be required exactly at the discontinuity, the derivative is taken only in the activated sub link (either left or right of the discontinuity). The

derivative across a unit to one of the unit input links is

$$\frac{\partial y_j}{\partial x_i} = \frac{1}{N_{in}}$$

The derivative of the weight with respect to the link output is

$$\frac{\partial y_j}{\partial w_{j,\alpha}} = \frac{\partial F_j}{\partial w_{j,\alpha}}$$

Error at the top of the link in the output link

$$\delta E_i = \frac{\partial E}{\partial x_i} = \frac{\partial y_i}{\partial x_i} \frac{\partial E}{\partial y_i} = \frac{\partial y_i}{\partial x_i} (y_i - y_{di})$$

Error at the top of the link one layer in

$$\delta E_j = \frac{\partial E}{\partial x_j} = \frac{\partial y_j}{\partial x_j} \frac{\partial E}{\partial y_j} = \frac{\partial y_j}{\partial x_j} \sum \frac{\partial x_i}{\partial y_j} \frac{\partial E}{\partial x_i} = \frac{\partial y_j}{\partial x_j} \sum \frac{\partial x_i}{\partial y_j} \delta E_i$$

In the specific case of an averaging unit, the error is

$$\delta E_j = \frac{\partial y_j}{\partial x_j} \frac{1}{N_{in}} \sum \delta E_i$$

The rule one layer in can be applied to all further layers. The error in the weight is then

$$\frac{\partial E}{\partial w_{\alpha,j}} = \frac{\partial y_j}{\partial w_{\alpha,j}} \frac{\partial E}{\partial y_j} = \frac{\partial y_j}{\partial w_{\alpha,j}} \frac{1}{N_{in}} \sum \delta E_i$$

At this point, all weight update rules that work for standard neural networks will work for this algorithm as well. A simple example is the momentum update

$$w_{\alpha,j}^{n+1} = w_{\alpha,j}^n - \mu \frac{\partial E}{\partial w_{\alpha,j}} + \gamma (w_{\alpha,j}^n - w_{\alpha,j}^{n-1})$$

Although the simple update works for this problem, we instead use Schaul's parameter free update Schaul et al. (2013).

## 2.2 Accelerated Backpropagation

The backpropagation algorithm described is the standard algorithm applied to this model. It was pointed out by Aizenberg and Moraga (2007) that a simple modification to the back propagation algorithm distributes the error more evenly among units. In particular, the algorithm shown suggests that an error at the unit is  $1/N_{in}$  times the error at the top of the output link (the links leaving the unit). The error at the output of the unit is distributed evenly over the input links, and therefore the more input links there are, the smaller the error that each input link receives. A simple way to remedy this problem is to replace the back propagation  $1/N_{in}$  (one over the number of active links entering a unit) with  $1/N_{out}$  (one over the number of links leaving a unit) which will result in a more even distribution of weight change throughout the network and faster convergence. This technique is used in this paper.

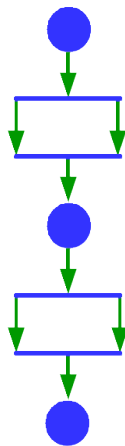


Figure 6: Two stacked bundles and their associated units. In this case each bundle has two sub links and therefore a signal can take one of two paths for each bundle.

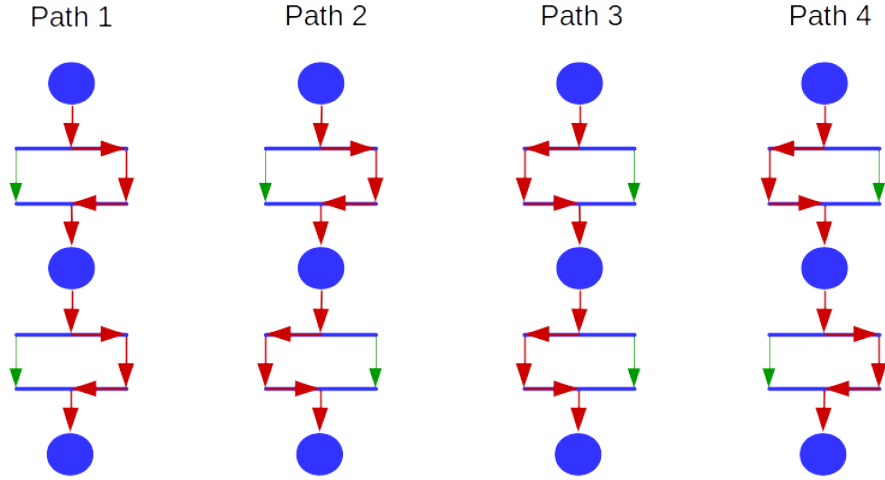


Figure 7: The total number of paths for two bundles stacked on top of one another, with 2 sub links each, is 4. The 4 paths are drawn out in this diagram. Only the active path is used in the back propagation algorithm for each signal. This means that the network is actually 4 separate coupled networks. The coupling occurs because some of the weights are shared between networks (whenever part of a path is shared, the weights are shared).

number of points	2	3	4	5	6	7	8	9	10
$p_{n,\max}$	1	1.25	1.667	1.799	1.989	2.083	2.203	2.275	2.362

Table 2: Maximum overshoot fraction,  $p_{n,\max}$ , given the number of points in the polynomial interpolation (to 4 digits rounded up).

### 2.3 Weight Initialization

Weight initialization is a huge concern for neural networks (Yam and Chow (2000)). In this paper, the weights within a link are initialized such that  $F(x)$  (Equation ) is a line across the link, the equation is given as

$$\omega_i = \left( \frac{x_i - r_{\min}}{r_{\max} - r_{\min}} \right) \omega_a + \omega_a, \quad (3)$$

where  $w_a$  is chosen with a uniform random distribution in the range  $[-1, 1]$ .

### 2.4 Choosing Ranges $r_{\min}$ and $r_{\max}$

Choosing proper ranges for the links is key to getting good solutions. The weights used in the Lagrange polynomial interpolation do not mark the limits of the polynomial value except in the linear case. Figure 8 shows 5 Lagrange polynomials with maximum overshoot for weights within the desired range - note that only in the linear case is the range limited by the values of the weights. Instead, if the weights are in the range  $[-\omega_{\max}, \omega_{\max}]$  then a given choice of weights will produce a maximum overshoot  $p_{n,\max}\omega_{\max}$  where values of  $p_{n,\max}$  are given in Table 2. For a given input, the maximum possible output would be  $p_{n,\max}\omega_{\max}$  which could then be the input to the next link. The input range for each link should be set to  $[-p_{n,\max}\omega_{\max}, p_{n,\max}\omega_{\max}]$  to account for these overshoots. One potential consequence of this choice of range is input value decay. Consider a deep network with only one unit per layer and one link between units. Each layer is initialized using Equation () with  $w_a = 1$  and input range  $[-p_{n,\max}, p_{n,\max}]$ . Suppose the input value is  $x_{\text{in}}$  then as the input value passes through each layer it will be compressed if  $r_{\max} = p_{n,\max} > \omega_{\max}$  by the ratio  $\omega_{\max}/r_{\max}$ . After passing through  $n$  layers, the output signal will be  $(\omega_{\max}/r_{\max})^n x_{\text{in}}$ . As a consequence, the output from each layer is pushed towards the value 0 which means fewer and fewer of the network sub links are used. Fortunately, two things can occur to help the situation: (1) if a discontinuity is at the origin rapid learning can still occur since if a signal is either side of 0 it will adjust disconnected weights; (2) as the network is trained, more and more of the sub links are used. It would seem randomly initializing the weights could alleviate this problem, however, we've found that symmetric initialization as in Equation () works better. In this paper we use  $r_{\max} = -r_{\min} = p_{n,\max}$  where  $p_{n,\max}$  is provided in Table 2 which gives the maximum overshoot for the polynomials when the weights are constrained to be between 1 and -1.

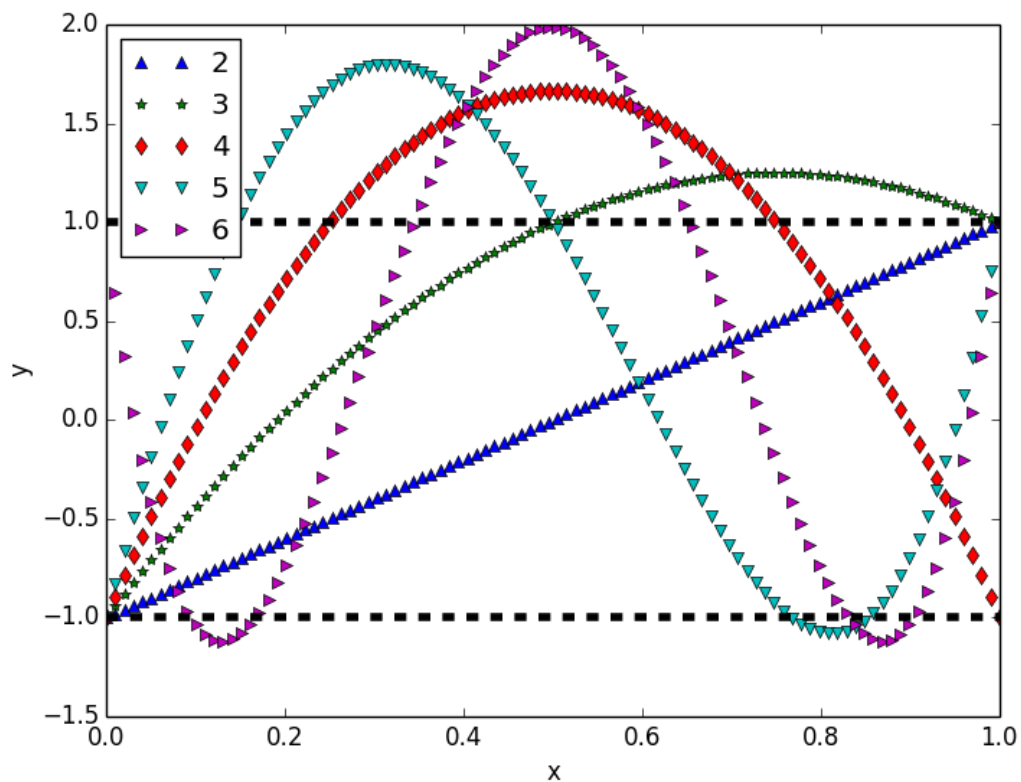


Figure 8: Lagrange polynomials with weights in the range  $[-1, 1]$  chosen for maximum overshoot. Even though the weights are limited to values in the range  $[-1, 1]$  the function can produce values outside this range. The next link then should have an input range that accounts for this overshoot.

## 2.5 Automatic Normalization

Since input links and output links are used in this network and the range and weight is arbitrary, data does not need to be normalized before passing to the network. Normalization automatically occurs for input links by the user's choice of  $r_{min}$  and  $r_{max}$  for the input links, and by the users choice of  $w_{min}$  and  $w_{max}$  for the output links (see Figure 1 for definitions of input and output links). For example, if the MNIST data is being used and image values range from 0 to 255, simply set  $r_{min} = 0$  and  $r_{max} = 255$  in the input link. Similarly, if the output values should be between 0 and 100 then set the weight limits in the output link to  $w_{min} = 0$  and  $w_{max} = 100$ . Although not particularly important, this is a convenient way to deal with input and output without a separate normalization step.

## 3. Results

In this section we use the network on 3 problems. The first is the simple sine wave to illustrate how the functional link approximates this function. This provides clarity for how the link in this network differs from the single weight used in the standard approach. The second problem uses the Pima Indians data set to predict if a patient has diabetes or not. This is a small problem and is used to show the effectiveness of dropout Hinton et al. (2012) for this type of network. Finally, results for the MNIST problem are computed with 10 autoencoders (one for each digit) in a manner similar to that described in Kamyshanska and Memisevic (2013), though using the reconstruction error to determine the digit. All results are computed using online backpropagation. In this problem, all input and output links have fixed weights and the link function is linear with  $\omega_0 = -1$  and  $\omega_n = 1$ . Input links have a range  $[r_{min}, r_{max}]$  dependent on the range of the input parameters.

### 3.1 Sine Wave

A sine wave is approximated using a single link with several sub links showing the function approximation capability of a single link. This problem is illustrative of how the functions in each sub link are combined to produce the full function. Figure 9 shows a sine wave approximated using a bundle with 3 and 5 linear sub links where discontinuity is allowed between the sub links. Figure 10 shows the same sine wave approximated using a link with 2 and 3 quadratic sub links with discontinuity between them. Note that the linear approximation has 6, and 10 degrees of freedom (for 3 and 5 ranges) and the quadratic approximation has 6 and 9 degrees of freedom (for 2 and 3 ranges). Despite having fewer degrees of freedom the quadratic approximation is substantially better than the linear approximation. This just illustrates the fact that higher order polynomial approximations require fewer degrees of freedom (for the same accuracy) than lower order approximations during function approximation.

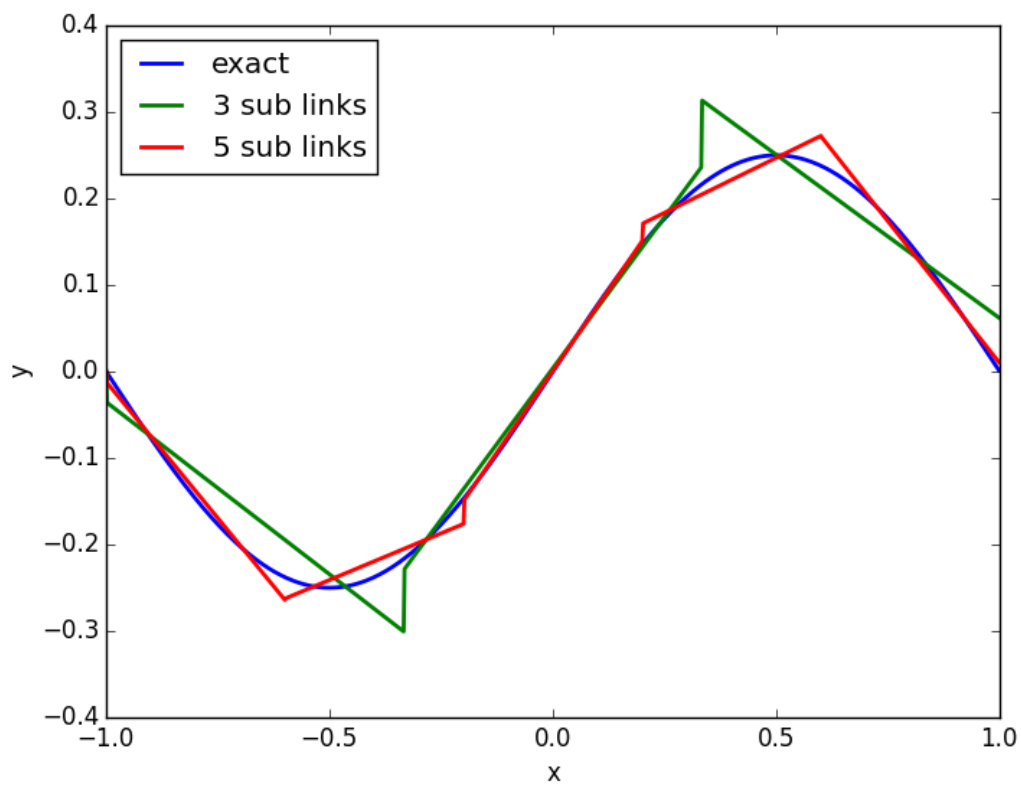


Figure 9: A single link used to approximate a sine wave with piecewise linear sub links ( $N_p = 2$ ). As the number of elements increases the fit improves. Discontinuities are visible at sub link boundaries, the level of discontinuity decreases as the number of sub links is increased.



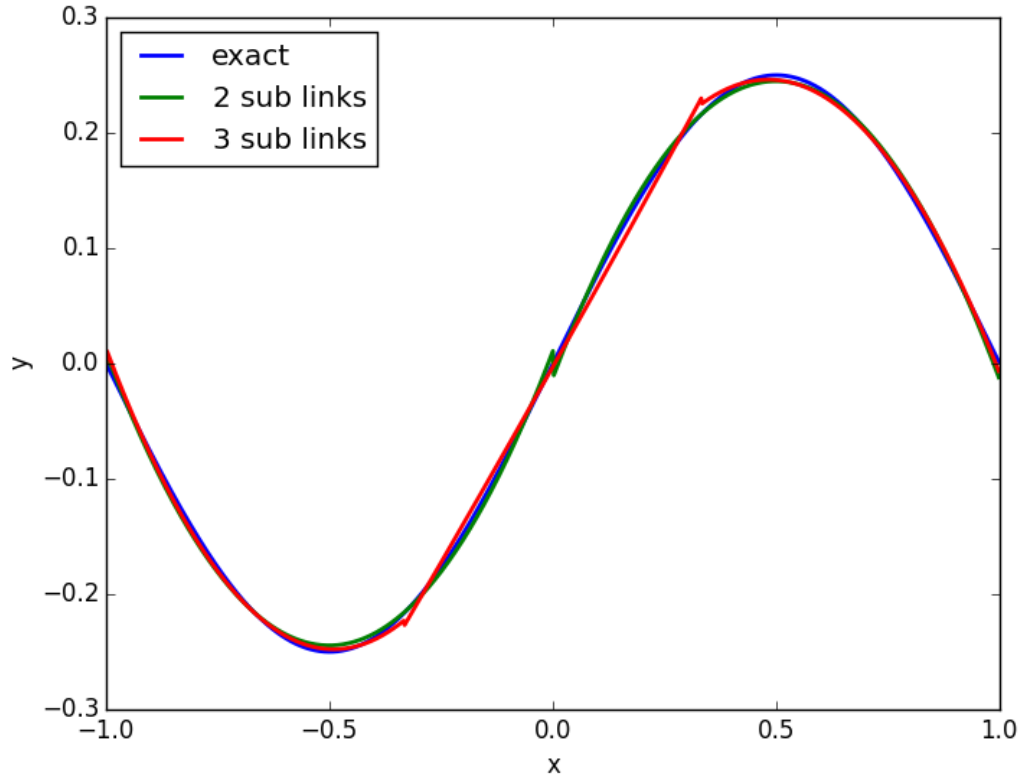


Figure 10: A single link used to approximate a sine wave with piecewise quadratic elements ( $N_p = 3$ ). In this case we get a much better match than the case with piecewise linear elements even with fewer total weights. This is a well-known feature of higher order approximations.

### 3.2 Pima Indians

The Pima Indians test is a simple problem with 8 inputs and 2 outputs and has been a standard benchmark in the past Knowler et al. (1978); Shanker (1996). It's also a very good example of the effectiveness of dropout since it is easy to overfit the problem and so we've chosen to present it here. Table 3 shows the case where unit dropout is used on the hidden layers. 50% of the units are turned off during training and a new 50% are turned off every 10 iterations. Table 4 shows the case where no dropout is used. The error rates presented are averaged over 10 runs where 20% of the Pima data was used as a test set and the other 80% used for training. The training set was selected randomly for each of the 10 runs, but all examples used the same 10 training and test sets. Each run was performed with 8 units in the first layer, 16 in the second and 2 in the last layer. The layers were fully connected. The results show that dropout significantly improves the test set error (generalization) while increasing the training error (reducing over training). In this problem, having one discontinuity (2 sub links) significantly improves the linear ( $N_p = 2$ ) link over the case with no discontinuity. In this particular problem, for links with higher order accuracy ( $N_p > 2$ ) good results can be achieved without any discontinuities. This phenomena is generally not repeated for other problems, such as MNIST where at least 1 discontinuity is required to get good results. The key difference between using a linear polynomial in the link versus a quadratic (3 point) or higher order polynomial is that adding layers to the linear link produces an output that is still linear. If an  $n$ th order polynomial is used as the link function, the output polynomial function is of order  $p^{n \cdot k}$  where  $k$  is the number of layers. If in addition, discontinuities are included in the link, these discontinuities serve to provide multiple interacting polynomial networks.

$N_p$	sub links	% test error	% training error
2	1	28.0	28.5
2	2	23.2	20.2
2	3	24.1	20.0
2	4	23.8	14.4
2	5	25.3	9.5
2	6	25.6	5.6
3	1	26.0	25.3
3	2	23.0	16.5
3	3	23.3	13.2
3	4	24.7	6.2
4	1	21.9	22.6
4	2	23.2	12.9
4	3	24.4	12.4
5	1	22.9	21.5
5	2	24.4	10.6
5	3	23.4	8.8
6	1	22.7	21.3
6	2	25.2	6.5

Table 3: Results on Pima Indians data set using dropout. Each case was run for 200,000 steps on an 8X16X2 fully connected network.

$N_p$	sub links	% test error	% training error
2	1	30.6	28.0
2	2	25.1	20.0
2	3	25.4	19.3
2	4	29.4	9.6
2	5	29.8	5.3
2	6	31.0	2.2
3	1	27.6	25.9
3	2	27.2	12.5
3	3	26.3	11.1
3	4	29.4	2.07
4	1	24.5	21.7
4	2	28.0	10.1
4	3	27.4	7.8
5	1	24.2	21.4
5	2	30.0	5.6
5	3	31.0	4.9
6	1	24.2	21.6
6	2	30.0	3.3

Table 4: Results on Pima Indians data set without dropout. Each case was run for 200,000 steps on an 8X16X2 fully connected network.

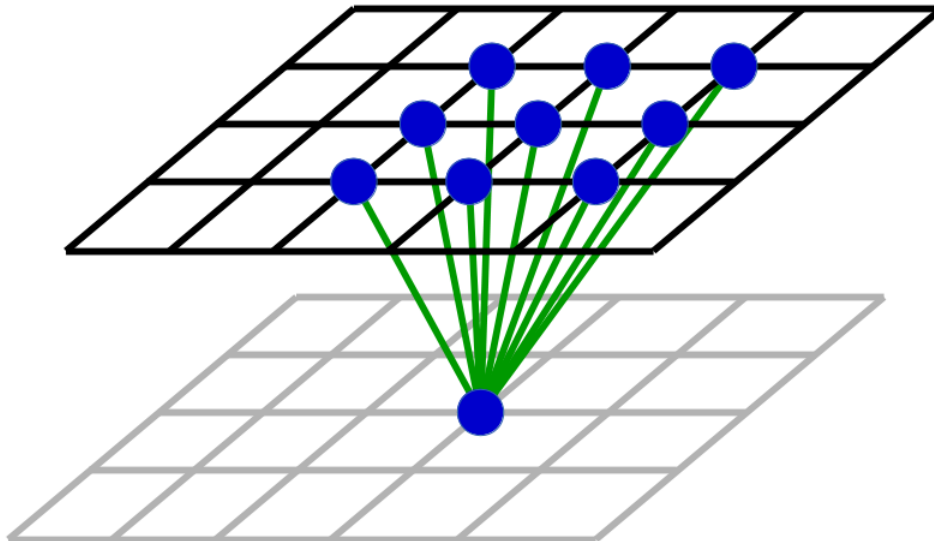


Figure 11: Nearest neighbor connectivity used on the MNIST problem. In this case the unit in the lower layer is connected to the  $N$  nearest neighbors in the previous layer. In the above example there are 9 nearest neighbors and the **width** of the stencil is defined to be 1. A stencil with **width**=2 would have 25 nearest neighbors.

### 3.3 Handwritten Digit Recognition

The MNIST is a standard benchmark of neural network codes on optical character recognition (LeCun et al., 1998). The MNIST data set consists of a 60,000 image training set of 10 digits written from NIST employees and high schoolers. In addition there is a 10,000 image test set that is used to test the generalization capability of the network after the network is trained on the training set. The images consist of 28X28 pixel 1 byte/pixel gray scale images. As such, there are 728 input links (one for each input pixel) and 10 output links (one for each digit). The number of hidden layers and the number of neurons in those hidden layers is variable. The digit recognition problem is solved using

a multi-layer autoencoder for each digit. There are 10 autoencoders trained with each of the 10 digits. The Autoencoder 0 is only trained on the digit 0 examples, autoencoder 1 is only trained on the digit 1 examples etc... At the end, the test set examples are run through each autoencoder, the predicted digit is determined by the autoencoder that produces the smallest error for the given input. The error used in this paper is reconstruction error, improvements in this error measure for this approach to classification are explored in Kamyshanska and Memisevic (2013) which might allow us to obtain even better results. Table 5 and 6 shows results for various networks on the MNIST benchmark. Tests were run with either 200,000 or 400,000 presentations of each networks test set, corresponding to roughly 33 and 67 epochs respectively. The following labels and definitions are used in tables 5 and 6.

- $N_p$  - the number of Chebyshev-Lobatto points in the interpolation.
- layers - the number of neurons in each hidden layer.
- DOF - degrees of freedom which is just  $N_p \times \text{sublinks}$
- sub links - the number of sub links in each link.
- width - the width of the stencil as defined in Figure 11.
- % test error - (% of test images classified incorrectly).
- % training error - (% of training images classified incorrectly).
- steps - the number of times images were presented for training.

Table 5 shows MNIST results for  $N_p = 2$  to 6 varying the number of neighbors used in the multi-layer autoencoder. At a width of 7, a signal from the upper left corner of the input image is able to interact with the signal from the lower right. Test set accuracy improves as the width is increased, but convergence also generally slows down. Results with  $N_p = 3, 4, 5$  are substantially better than those produced for  $N_p = 2$ . In addition, resulting test set accuracy tends to increase with increasing width. Unsurprisingly, the greater connectivity generally results in a better solution. In addition however, greater polynomial order also improves accuracy, especially moving from  $N_p = 2$  to  $N_p = 3$ .

Table 6 shows MNIST results for  $N_p = 2$  to 6 varying the number of sub links. The number of sub links are varied to determine whether the generally improved accuracy is a result of the increased number of degrees of freedom or due to the higher order polynomial representation. In several runs of 6 the performance is very poor, first of all, in all cases with only 1 sub link (no discontinuity) the performance is bad, though this is expected (Leshno et al., 1993). At least 1 discontinuity should be used. In addition, poor results were observed in  $N_p = 4$  with sub links = 3 and  $N_p = 6$  with sub links = 3. It's thought that there are 2 causes for this. (1) In deep networks the initial weight values are random and the output of each link is averaged at the unit which tends to focus that output around the value 0. As the number of layers is increased the output of each unit approaches 0 more closely. If an odd number of sub links are used, the link function is smooth at 0. If an even number of sub links are used then there is a discontinuity at 0. The discontinuity allows for rapid changes of weight near the origin, and the result is less likely to become stuck. In both

$N_p$	layers	sub links	width	% test error	% training error	steps
2	5	2	5	5.70	6.295	4e5
2	5	2	6	6.81	7.355	4e5
2	5	2	7	7.67	8.306	4e5
2	5	6	6	1.97	1.577	4e5
2	5	6	7	2.20	1.717	2e5
2	5	6	7	2.06	1.448	4e5
3	5	2	5	2.41	2.206	2e5
3	5	2	6	2.02	1.945	2e5
3	5	2	7	1.98	1.703	4e5
4	5	2	5	2.29	1.956	2e5
4	5	2	6	2.06	1.836	2e5
4	5	2	7	1.89	1.531	4e5
5	5	2	5	2.10	1.697	2e5
5	5	2	6	1.90	1.475	2e5
5	5	2	7	1.71	1.093	4e5
6	5	2	5	1.87	1.463	2e5
6	5	2	6	1.95	1.435	2e5
6	5	2	7	1.75	0.978	4e5

Table 5: 10 independently trained autoencoder MNIST results. The number of nearest neighbors in the autoencoder is adjusted to see the variation in accuracy. The  $N_p = 2$  case shows low learning for the number of training steps as the training error remains higher than the test error. Cases with  $N_p = 2$  and 6 sub links were run (highlighted in yellow) to show that good learning can be achieved with  $N_p = 2$ . Light green highlights indicate solutions that are better than 98% accurate on the test set.

$N_p$	layers	sub links	DOF	width	% test error	% training error	steps
2	5	1	2	4	17.97	19.221	2e5
2	5	2	4	4	5.07	5.408	4e5
2	5	3	6	4	4.77	5.015	4e5
2	5	4	8	4	2.81	2.628	4e5
2	5	5	10	4	2.86	2.515	4e5
2	5	6	12	4	2.19	1.743	4e5
3	5	1	3	4	18.05	19.218	2e5
3	5	2	6	4	2.40	2.205	2e5
3	5	3	9	4	2.32	2.147	2e5
3	5	4	12	4	2.16	1.788	2e5
4	5	1	4	4	17.95	19.213	2e5
4	5	2	8	4	2.32	2.097	2e5
4	5	3	12	4	17.98	19.19	2e5
5	5	1	5	4	17.97	19.207	2e5
5	5	2	10	4	2.28	1.911	2e5
5	5	3	15	4	3.01	2.615	2e5
5	5	4	20	4	2.19	1.466	2e5
6	5	1	6	4	17.93	19.23	2e5
6	5	2	12	4	2.02	1.565	2e5
6	5	3	18	4	18.01	19.23	2e5
6	5	4	24	4	2.20	1.277	2e5

Table 6: 10 independently trained autoencoder MNIST results. The number of sub links is adjusted to see the variation in accuracy. Blue highlights indicate solutions which performed exceptionally poorly in the number of steps specified (discussed in the text). As seen in Table 5 better overall learning can be achieved by increasing the width. Results with a single sub link show poor learning and so at least one discontinuity should be present. This result is entirely expected as described in Leshno et al. (1993).  $N_p = 2$  tends to learn more slowly (the reason for the larger number of steps), but good results can be achieved by increasing the number of discontinuities.



cases described, the number of sub links is odd so there is no discontinuity at the origin of each link function. (2) For  $N_p > 2$  this problem is exacerbated by input compression (described in Section ) since  $r_{max} > w_{max}$ . A remedy for this problem might include (1) a more sophisticated initialization scheme and (2) an alternative to the averaging occurring at the unit that pushes the incoming signal away from the origin. These approaches are not investigated in this paper.

$N_p$	sub links	$t_{N_p}$ (seconds)	$t_{N_p}/t_{N_2}$
2	2	50	1
3	2	61	1.2
4	2	77	1.5
5	2	90	1.8
6	2	108	2.2

Table 7: Timing results for equal number of links for 1000 inputs.

$N_p$	sub links	$t_{N_p}$ (seconds)	$t_{N_p}/t_{N_2}$
2	6	59	1.0
3	4	65	1.1
4	3	90	1.5
6	2	108	1.8

Table 8: Timing results for equal number of degrees of freedom for 1000 inputs.

### 3.4 Timing Results

A critical question is whether that added complexity and computational time of  $N_p > 2$  is really worth the effort. Both the derivatives and function evaluations become more complex as  $N_p$  increases.

In Tables 7 and 8 below we run the MNIST problem as above with 5 layers, and width 6 with the other parameters specified in the table. The results were computed by running the test case 10 times and averaging the main loop time. Adding additional sub links is not cost free,  $N_p = 2$  with 6 sub links is 12% slower than  $N_p = 2$  with 2 sub links, but with 3 times the degrees of freedom - this is despite the fact the only one sub link is ever active. In addition increasing the number of sub links with  $N_p = 2$  to 12 and the time jumps to 111 seconds (122% slower than with 2 sub links) for 1000 iterations. Table 7 shows that  $N_p = 6$  is only 2.2 times slower than  $N_p = 2$  despite having 3 times the degrees of freedom. Similarly, in Table 8 with the same number of degrees of freedom  $N_p = 6$  is only 1.83 times slower on average. It was observed in section that even for the same number of degrees of freedom  $N_p = 2$  results in a larger error 2.06% compared  $N_p = 6$  at 1.87% in Table 5 this is despite  $N_p = 2$  running for twice as many steps, a similar result is observed in Table 6. For the same number of degrees of freedom, moving to higher order accuracy, versus using lower accuracy with more sub links, is the more efficient approach.

Though performance is highly implementation dependent, the conclusion we draw here is that increasing polynomial order (beyond piecewise linear,  $N_p = 2$ ) is more computationally efficient for the equivalent number of degrees of freedom.

## 4. Conclusion

A novel approach to artificial neural networks is described where the traditional neuronal non-linearity is eliminated in favor of a discontinuous piecewise polynomial discretization of the weights space of each link. The use of discontinuous piecewise polynomial approximations leads to a network, which is the superposition of multiple networks with a set of shared weights as only a subset of the total network is active for each input signal. Standard backpropagation is used for error correction with the modification that sub links that do not fire are not included in the backpropagation step. The dropout technique Hinton et al. (2012) is used to minimize over fitting. It is found that piecewise quadratic polynomials generally produce much better results than piecewise linear for the same number of degrees of freedom and that moving to increasingly higher order polynomials can provide additional improvement. We believe the network described is easier to analyze than standard networks using hyperbolic tangent type non linearities and opens up the possibility of using all sorts of complicated discontinuous elements (both links and units). We have successfully demonstrated good solutions to the MNIST digit recognition test and expect more complicated problems can be solved as well using this algorithm.

## References

- Igor Aizenberg and Claudio Moraga. Multilayer feedforward neural network based on multi-valued neurons (mlmvn) and a backpropagation learning algorithm. *Soft Computing*, 11(2):169–183, 2007.
- Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.
- Mehdi Fallahnezhad, Mohammad Hassan Moradi, and Salman Zaferanlouei. A hybrid higher order neural classifier for handling classification problems. *Expert Systems with Applications*, 38(1):386–393, 2011.
- Gian Luca Foresti and T Dolso. An adaptive high-order neural tree for pattern recognition. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(2):988–996, 2004.
- M Forti and P Nistri. Global convergence of neural networks with discontinuous neuron activations. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 50(11):1421–1435, 2003.
- Ricard Gavalda and Hava Siegelmann. Discontinuities in recurrent neural networks. *Neural computation*, 11(3):715–745, 1999.
- C Lee Giles and Tom Maxwell. Learning, invariance, and generalization in high-order neural networks. *Applied optics*, 26(23):4972–4978, 1987.
- Jan S Hesthaven and Tim Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*, volume 54. Springer, 2007.
- Geoffrey Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

- Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
- Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 985–990. IEEE, 2004.
- Hanna Kamyshanska and Roland Memisevic. On autoencoder scoring. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 720–728, 2013.
- William C Knowler, Peter H Bennett, Richard F Hamman, and Max Miller. Diabetes incidence and prevalence in pima indians: a 19-fold greater incidence than in rochester, minnesota. *American Journal of Epidemiology*, 108(6):497–505, 1978.
- Stephen H Lane, David A Handelman, and Jack J Gelfand. Theory and development of higher-order cmac neural networks. *Control Systems, IEEE*, 12(2):23–30, 1992.
- Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.
- Xiaoyang Liu and Jinde Cao. Robust state estimation for neural networks with discontinuous activations. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 40(6):1425–1437, 2010.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.
- Yoh-Han Pao. *Adaptive pattern recognition and neural networks*. Addison-Wesley Longman Publishing Co., Inc., 1989.
- Jagdish Chandra Patra and Alex C Kot. Nonlinear dynamic system identification using chebyshev functional link artificial neural networks. 2002.
- Jagdish Chandra Patra, Ranendal N Pal, BN Chatterji, and Ganapati Panda. Identification of nonlinear dynamic systems using functional link artificial neural networks. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 29(2):254–262, 1999.

- Shubhi Purwar, Indra Narayan Kar, and Amar Nath Jha. On-line system identification of complex systems using chebyshev neural networks. *Applied Soft Computing*, 7(1):364–372, 2007.
- David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 1988.
- Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *Proceedings of The 30th International Conference on Machine Learning*, pages 343–351, 2013.
- Murali S Shanker. Using neural networks to predict the onset of diabetes mellitus. *Journal of chemical information and computer sciences*, 36(1):35–41, 1996.
- Yoan Shin and Joydeep Ghosh. Approximation of multivariate functions using ridge polynomial networks. In *Neural Networks, 1992. IJCNN., International Joint Conference on*, volume 2, pages 380–385. IEEE, 1992.
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- Jim YF Yam and Tommy WS Chow. A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, 30(1):219–232, 2000.
- Ming Zhang, Shuxiang Xu, and John Fulcher. Neuron-adaptive higher order neural-network models for automated financial data modeling. *Neural Networks, IEEE Transactions on*, 13(1):188–204, 2002.