A Gentle Introduction to Backpropagation

Shashi Sathyanarayana
Numeric Insight, Inc (www.numericinsight.com)

A Gentle Introduction to Backpropagation

Shashi Sathyanarayana, Ph.D

July 22, 2014

Contents

1 Why is this article being written?

Neural networks have always fascinated me ever since I became aware of them in the 1990s. I was initially drawn to the hypnotizing array of connections with which they are often depicted. In the last decade, deep neural networks have dominated pattern recognition, often replacing other algorithms in applications

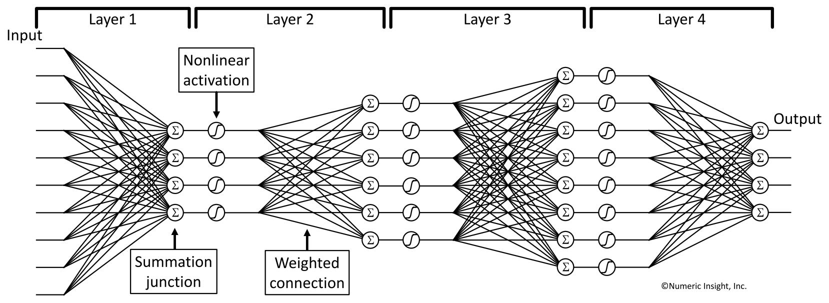

Figure 1: An example of a multi-layered neural network that can be used to associate an input consisting of 10 numbers with one of 4 decisions or predictions.

like computer vision and voice recognition. At least in specialized tasks, they indeed come close to mimicking the miraculous feats of cognition our brains are capable of.

While neural networks are capable of such feats, the very discovery of a method of programming such a computational device is to me, in itself, a miraculous feat of cognition worthy of celebration. My purpose in writing this article is to share my perspective on an amazing algorithm made widely known by a 1986 publication in *Nature*.

I will assume that the reader has some understanding of the meaning and purpose of the various elements of a neural network such as the one shown in Figure 1. I have provided a little bit of background below. Other than that, this article should be an easy read for those with some familiarity of basic calculus.

2 What is so difficult about designing a neural network?

To appreciate the difficulty involved in designing a neural network, consider this: The neural network shown in Figure 1 can be used to associate an input consisting of 10 numbers with one of 4 decisions or predictions. For example, the neural network shown may be used by a bank to determine if credit should be extended to a customer. In this case, the 10 input numbers represent various parameters relevant to an individual's financial responsibility, such as balance in savings accounts, outstanding loan amounts, number of years of employment, and so on. The neural network takes in these 10 numbers, performs calculations and produces an output consisting of 4 numbers. Depending on the position at which the maximum of these 4 numbers appears in the output, the prediction could be one of the following:

1. Excellent creditworthiness with high spending limit
2. Average creditworthiness with moderate spending limit

3. Low creditworthiness with low spending limit
4. High default risk.

Based on this prediction, the bank would take the appropriate decision.

A neural network essentially like the one shown in Figure 1 can perform this miraculous feat of cognition only if it is specifically trained to do so. For the network to work correctly, the weight at each of its $(10\times4)+(4\times6)+(6\times8)+(8\times4) = 144$ connections have to be carefully chosen such that the network classifies every input drawn from a known training set with a high degree of accuracy. This is a classic application of the supervised learning paradigm in machine learning.

There are, of course, no formulas in existence to directly set the values of the 144 weights. The only recourse is to start with some initial values for the 144 weights, check how good the resulting neural network is, and repeatedly refine the weights to progressively make the network more and more accurate. So, what is needed is a method of refining the weights.

If one were to think of the accuracy as some grand function of the weights, it makes sense to refine each weight by changing it by an amount proportional to the partial derivative of that grand function with respect to that weight. Why bring in partial derivatives? Because they, by definition, predict how the accuracy responds to small changes in the weights. In fact, at every iteration, performing a refinement guided by the partial derivatives results in a more advantageous gain in accuracy compared to any other method of refinement. The method of steepest descent does exactly what is suggested here – with the minor, but entirely equivalent, goal of seeking to progressively decrease the error, rather than increase the accuracy.

To keep things straight, let me list the concepts I have described thus far:

1. The weights of the neural network must be set such that the error calculated on a known training set is minimized. Ideally, the network must yield zero error on a large and representative training data set.

2. We adopt the following strategy in order to arrive at the weights that minimize error: Given an arbitrary choice of weights, refine them by changing each weight by a small amount proportional to the partial derivative of the error with respect to that weight. The partial derivatives themselves change after any such refinement, and must be recomputed before the next round of refinement can be applied.

3. We can keep on refining the weights in this manner, and stop refining when the error is zero. In real life, we call it done when the error is low enough. Or refuses to fall anymore. Or we run out of time after a few million rounds of refinements.

Seems simple, right? Yes. As long as there is a simple way of calculating the partial derivative of error with respect to every single weight, at every single iteration. In principle, the partial derivatives can be calculated by systematically perturbing the weights and measuring the resulting changes in error. A word to the wise, don't try this at home. Or even on your neighborhood supercomputer.

3 Backpropagation

In fact, this error minimization problem that must be solved to train a neural network eluded a practical solution for decades till D. E. Rumelhart, C. E. Hinton, and R. J. Williams (drawing inspiration from other researchers) demonstrated a technique, which they called backpropagation, and made it widely known (Nature 323, 533-536, 9 October 1986). It is essentially by building upon their method that today others have ventured to program neural networks with 60 million weights, with astounding results.

According to Bernard Widrow, now Professor Emeritus at Stanford University and one of the pioneers of neural networks, "The basic concepts of backpropagation are easily grasped. Unfortunately, these simple ideas are often obscured by relatively intricate notation, so formal derivations of the backpropagation rule are often tedious." This is indeed unfortunate because the backpropagation rule is one of the most elegant applications of calculus that I have known.

4 Easy as 1-2-3

Once you appreciate the fact that, in order to train a neural network, you need to somehow calculate the partial derivatives of the error with respect to weights, backpropagation can be easily and qualitatively derived by reducing it to three core concepts. It also helps immensely to keep the notation intuitive and easy to connect to the concept being symbolized.

4.1 Boxing

Since training the neural network is all about minimizing the training error, the first step in the derivation involves tacking on an extra computational block to calculate the error between the actual output $\{o_1, o_2, o_3, o_4\}$ and a known target $\{t_1, t_2, t_3, t_4\}$. This is shown as a triangular block in Figure 2. For now, let us think of the output and the target as known and fixed entities. Although we need not concern ourselves with the exact formula to compute the error, I offer the familiar sum-of-squares error as an example

$$e = (o_1 - t_1)^2 + (o_2 - t_2)^2 + (o_3 - t_3)^2 + (o_4 - t_4)^2$$

Next, we choose one of the layers (say Layer 3) and enclose that layer and all following layers (including the error calculating block) in a box, as shown in gray in Figure 2. Keep in mind that this is just one of several nested boxes we can construct in order to compartmentalize the network. Let us resolve not to worry about anything going on inside the box but simply think of the

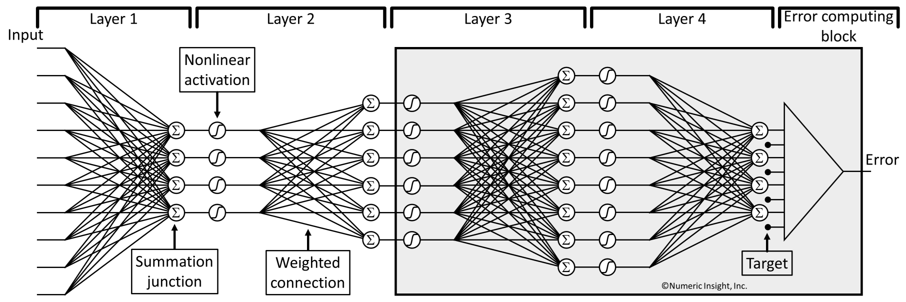

Figure 2: The derivation of the backpropagation algorithm is simplified by adding an extra computational block to calculate the error and also by boxing parts of the network. Compare with Figure 1.

relationship between the input to the box and the output (i.e the error) coming out of the box. We will call this box the current box, and call the input to this box the current input, $\{c_1, c_2, c_3, c_4, c_5, c_6\}$. It is important to recognize that the functional relationship between the current input to the box and the output emanating from the box is completely defined and can be computed. Let us denote this relationship as $e = E(c_1, c_2, c_3, c_4, c_5, c_6)$.

4.2 Sensitivity

As our journey through backpropagation continues, I gently request you to assume that the vector of partial derivatives $\frac{\partial E}{\partial c_1}, \frac{\partial E}{\partial c_2}, \frac{\partial E}{\partial c_3}, \ldots$ of the function $E(c_1, c_2, c_3, c_4, c_5, c_6)$ is known. This might seem like asking for too much! After all, we have set out to find a method to compute some other (equally unrealized) partial derivatives. But I assure you it will all work out in the end. To emphasize the crucial nature of this simple concept, it has been given a name: *Sensitivity*. Let us denote the sensitivity of our current box as $\{\delta c_1, \delta c_2, \delta c_3, \delta c_4, \delta c_5, \delta c_6.\}$ Remember, sensitivity is a vector, not a single number.

With the help of Figure 3 to hold these concepts in our mind, we can concretely think about how the output of the current box responds to a small perturbation applied to any of its current inputs. For example, if the fourth component of the current input changes by the small amount Δc_4, we can expect the error at the output to change by $\Delta c_4 \delta c_4$. Further, in addition to the hypothetical change in component 4, if there is a simultaneous change of Δc_6 in component 6, we can expect the error at the output to change by an additional amount, making the total change $\Delta c_4 \delta c_4 + \Delta c_6 \delta c_6$. The effect of small simultaneous changes in the current input components simply add up at the output.

Knowing the sensitivity of the current box, what can we say about the sensitivity of the preceding box? Keep in mind that the preceding box encloses Layer 2 and all following layers, including the error calculating block. For our

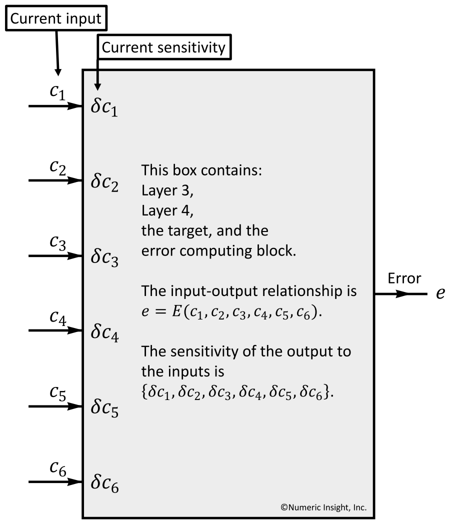

Figure 3: This diagram shows the boxed portion of the neural network of Figure 2. By hiding details of internal connections within the box, this scheme allows us to think about the broad relationship between the input and the output pertaining to this box.

specific example, let us call the input to the preceding box the preceding input, $\{p_1, p_2, p_3, p_4.\}$ It follows quite logically that the sensitivity of the preceding box (which we will naturally denote as $\{\delta p_1, \delta p_2, \delta p_3, \delta p_4\}$) must be related to the sensitivity of the current box and the extra neural network elements making up the difference between the two nested boxes. The extra elements are the very vital nonlinear activation function units, summing junctions and weights.

Figure 4 (top) shows the current box and the extra elements that must be added to construct the preceding box. For clarity, all the elements not relevant to the calculation of the first component of sensitivity (δp_1) have been grayed out. Look closely at the top and bottom parts of Figures 4 to understand how the sensitivity of the preceding box can easily be derived from first principles. Specifically, the bottom part of Figure 4 provides insight into how $\delta p_1 (= \frac{\partial e}{\partial p_1})$ can be computed by allowing the input component p_1 to change by a small quantity Δp_1 and following the resulting changes in the network. Notes: (i) The notation $\mathcal{A}'(p_1)$ has been used for the derivative of the activation function evaluated at p_1. (ii) For clarity, not all changes in signals have been explicitly labeled. Those that are not labeled can easily be determined since they all follow an obvious pattern.

This is indeed a deep result. (And one which we have managed to arrive at without recourse to tedious calculus and numbered equations.) By repeated application, this result allows us to work backwards and calculate the sensitivity of the error to changes in the input at every layer.

The algorithm gets the name backpropagation because the sensitivities are

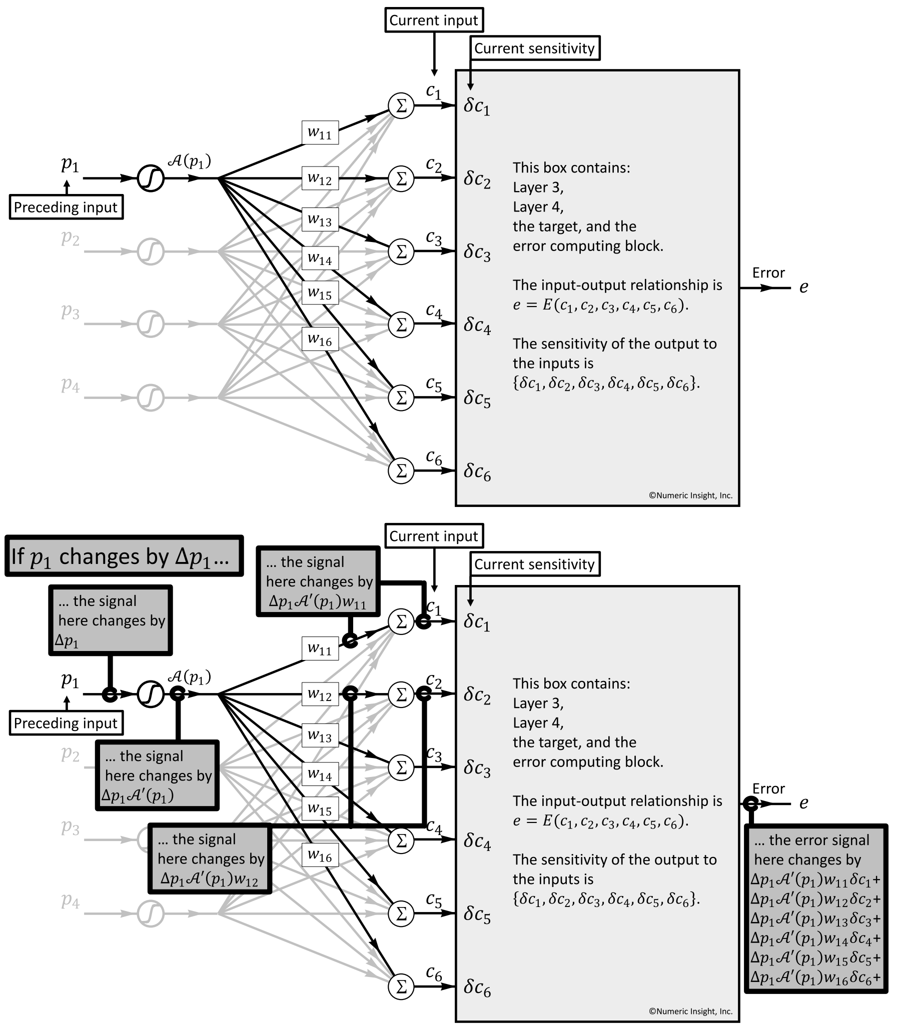
Current input
Current sensitivity
c_1 c_2 c_3 c_4 c_5 c_6
δc_1 δc_2 δc_3 δc_4 δc_5 δc_6
w_{11} w_{12} w_{13} w_{14} w_{15} w_{16}
p_1 p_2 p_3 p_4
$\mathcal{A}(p_1)$
Preceding input
This box contains:
Layer 3,
Layer 4,
the target, and the
error computing block.
The input-output relationship is
$e = E(c_1, c_2, c_3, c_4, c_5, c_6)$.
The sensitivity of the output to
the inputs is
$\{\delta c_1, \delta c_2, \delta c_3, \delta c_4, \delta c_5, \delta c_6\}$.
Error
e
©Numeric Insight, Inc.
If p_1 changes by Δp_1…
… the signal here changes by Δp_1
… the signal here changes by $\Delta p_1 \mathcal{A}'(p_1)$
… the signal here changes by $\Delta p_1 \mathcal{A}'(p_1) w_{11}$
… the signal here changes by $\Delta p_1 \mathcal{A}'(p_1) w_{12}$
… the error signal here changes by
$\Delta p_1 \mathcal{A}'(p_1) w_{11} \delta c_1 +$
$\Delta p_1 \mathcal{A}'(p_1) w_{12} \delta c_2 +$
$\Delta p_1 \mathcal{A}'(p_1) w_{13} \delta c_3 +$
$\Delta p_1 \mathcal{A}'(p_1) w_{14} \delta c_4 +$
$\Delta p_1 \mathcal{A}'(p_1) w_{15} \delta c_5 +$
$\Delta p_1 \mathcal{A}'(p_1) w_{16} \delta c_6 +$

Figure 4: The drawing on the top shows the current box and the extra elements that must be added to construct the preceding box. The drawing on the bottom provides insight into how the sensitivity component $\delta p_1 (= \frac{\partial e}{\partial p_1})$ can be computed by allowing the input component p_1 to change by a small quantity Δp_1 and following the resulting changes in the network.

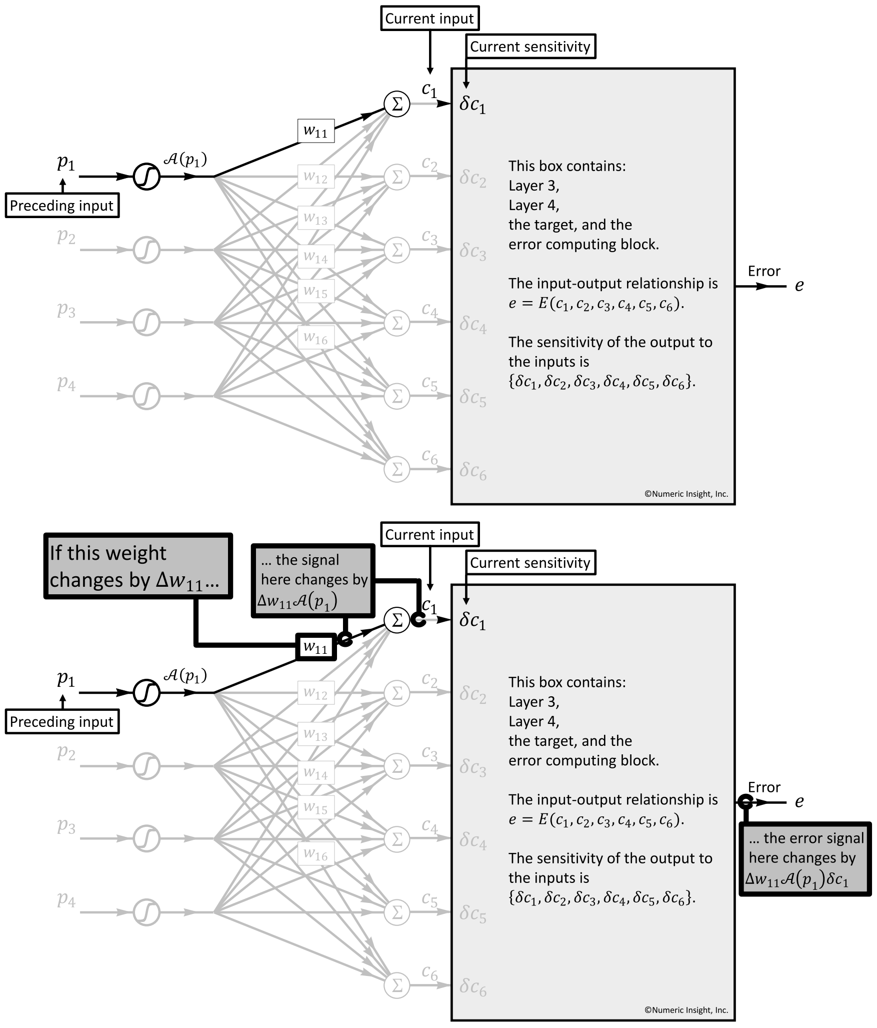
Current input
Current sensitivity
c_1
δc_1
w_{11}
p_1
$\mathcal{A}(p_1)$
Preceding input
p_2
p_3
p_4
w_{12}
w_{13}
w_{14}
w_{15}
w_{16}
c_2 c_3 c_4 c_5 c_6
δc_2 δc_3 δc_4 δc_5 δc_6
This box contains: Layer 3, Layer 4, the target, and the error computing block.
The input-output relationship is $e = E(c_1, c_2, c_3, c_4, c_5, c_6)$.
The sensitivity of the output to the inputs is $\{\delta c_1, \delta c_2, \delta c_3, \delta c_4, \delta c_5, \delta c_6\}$.
Error
e
©Numeric Insight, Inc.
If this weight changes by Δw_{11}…
… the signal here changes by $\Delta w_{11}\mathcal{A}(p_1)$
… the error signal here changes by $\Delta w_{11}\mathcal{A}(p_1)\delta c_1$

Figure 5: Similar to Figure 4, the drawing on the top shows the current box and the extra elements that must be added to construct the preceding box. The drawing on the bottom provides insight into how the partial derivative $\frac{\partial e}{\partial w_{11}}$ (used to guide weight refinement) can be computed by allowing the weight w_{11} to change by a small quantity Δw_{11} and following the resulting changes in the network. Using this intuition, $\frac{\partial e}{\partial w_{11}}$ can be computed as $\mathcal{A}(p_1)\delta c_1$.

propagated backwards as they are calculated in sequence. The textbook formula to express the sensitivity of the preceding layer in terms of the sensitivity of the current layer is easily seen to be

$$\delta p_i = \mathcal{A}'(p_i) \sum_j w_{ij} \delta c_j$$

A starting point is all we need to completely calculate all the sensitivity terms throughout the neural network. To do this, we consider the error computing block itself as the first box. For this box, the input is $\{o_1, o_2, o_3, o_4\}$, and the output is e as given in the sum-of-squares error formula we have seen before. Simple calculus gives us the components of the sensitivity of the error computing block

$$\{2\,(o_1 - t_1)\,, 2\,(o_2 - t_2)\,, 2\,(o_3 - t_3)\,, 2\,(o_4 - t_4)\}$$

4.3 Weight updates

At this point, the last section writes itself. Following the same strategy outlined in the previous figure, look at Figure 5 to intuitively understand how the error changes in response to a small change in one of the weights, say w_{11}. Once again in these figures, details of connections not immediately relevant to this calculation have been grayed out. The much sought after partial derivative of error with respect to the specific weight w_{11} is easily seen to be $\mathcal{A}(p_1)\delta c_1$. Generalizing on this, the textbook formula to compute the partial derivative of the error with respect to any weight is easily seen to be

$$\frac{\partial e}{\partial w_{ij}} = \mathcal{A}\left(p_i\right) \delta c_j$$

Now that we have a formula to calculate the partial derivative of error with respect to every weight in the network, we can proceed to iteratively refine the weights and minimize the error using the method of steepest descent.

In the most popular version of backpropagation, called stochastic backpropagation, the weights are initially set to small random values and the training set is randomly polled to pick out a single input-target pair. The input is passed through the network to compute internal signals (like $\mathcal{A}(p_1)$ and $\mathcal{A}'(p_1)$ shown in Figures 4 and 5) and the output vector. Once this is done, all the information needed to initiate backpropagation becomes available. The partial derivatives of error with respect to the weights can be computed, and the weights can be refined with intent to reduce the error. The process is iterated using another randomly chosen input-target pair.

5 The miraculous feat of cognition

I am in awe of the miraculous feat of cognition that lead early neural network researchers to arrive at the backpropagation algorithm. They clearly had the ability to see patterns and make elegant groupings which ultimately made it possible to train huge networks. Their work not only resulted in the neural network applications we use today, but have also inspired a host of other related algorithms which depend on error minimization.

Although this algorithm has been presented here as a single established method, it should be regarded as a framework. In my experience, appreciating how an algorithm is derived leads to insight which makes it possible to explore beneficial variations. The process of designing robust and efficient scientific algorithms frequently leads us to regard established frameworks as substrates upon which to build new and better algorithms.

The algorithm outline which follows on the next page has been provided to make the concepts presented in this article more concrete, and enable a working multi-layer neural network to be actually programmed and trained in your favorite language. Note that, although the notation used in the preceding pages no longer applies to the algorithm outline on the next pages, we continue to keep the notation intuitive and make it easy to connect variable names to concepts. A table of notations is provided. Also note that the addition of bias terms results in a slight change in the architecture of the layers. This is an important consideration because the use of bias terms makes the training process converge faster. We hope that enabling encouraging results in early experimentation will lead one to try out more complex configurations of the algorithm on practical applications or new and interesting problems.

6 Backpropagation Algorithm Outline. *Training Wheels for Training Neural Networks*

Notation

General Symbols	
L	Number of layers
l	Layer index
N^l	Number of input components going into layer l.
I_i^l	i^{th} component of the input going into layer l. Note that $i = \{0, 1, 2, \ldots, N^l\}$. See Figure 6 to understand why i starts from 0.
N^{L+1}	Number of output components coming out of layer L (i.e. the last layer, or output layer.)
O_j	j^{th} component of the output coming out of layer L. Note that $j = \{1, 2, 3, \ldots, N^{L+1}\}$
$\mathcal{A}()$	The activation function. For example, $\mathcal{A}(I_3^2) = \tanh(I_3^2)$
$\mathcal{A}'()$	The derivative of the activation function. For example, $\mathcal{A}'(I_3^2) = 1 - \tanh^2(I_3^2)$
w_{ij}^l	The weight element connecting the i^{th} input to the j^{th} output in layer l. Note the ranges $l = \{1, 2, 3, \ldots, L\}$, $i = \{0, 1, 2, \ldots, N^l\}$, and $j = \{1, 2, 3, \ldots, N^{l+1}.\}$
δ_i^l	i^{th} component of the sensitivity measured at the input to layer l. Note that $i = \{1, 2, 3, \ldots, N^l\}$
t_j	j^{th} component of the target corresponding to the chosen input. Note that $j = \{1, 2, 3, \ldots, N^{L+1}\}$
g_{ij}^l	The partial derivative of the error e with respect to the weight w_{ij}^l. Note that we have chosen the symbol g to suggest *gradient.*
r	The learning rate.
Training Set	
$\mathcal{I}$	Training set input. Note that $\mathcal{I}$ is a matrix whose rows are selected randomly during training to extract vectors of the form $\{I_1^1, I_2^1, I_3^1, \ldots, I_{N^1}^1\}$ that feed into the network
$\mathcal{T}$	Training set target. Note that at every instance of selection of a row within $\mathcal{I}$, the corresponding row of $\mathcal{T}$ is selected to extract a vector of the form $\{t_1, t_2, t_3, \ldots, t_{N^{L+1}}\}$. It is this vector that is compared with the actual output of the network during training iterations

Table continues on next page ...

Superscripts are used to index layers as in, for example, I_2^3. This is an intuitive notation and it should be clear from context if a quantity is actually being raised to a power.

...table continued from previous page

Stopping Criteria	
g_{Min}	The threshold value of gradient
k	Iteration index
k_{Max}	Maximum number of iterations allowed

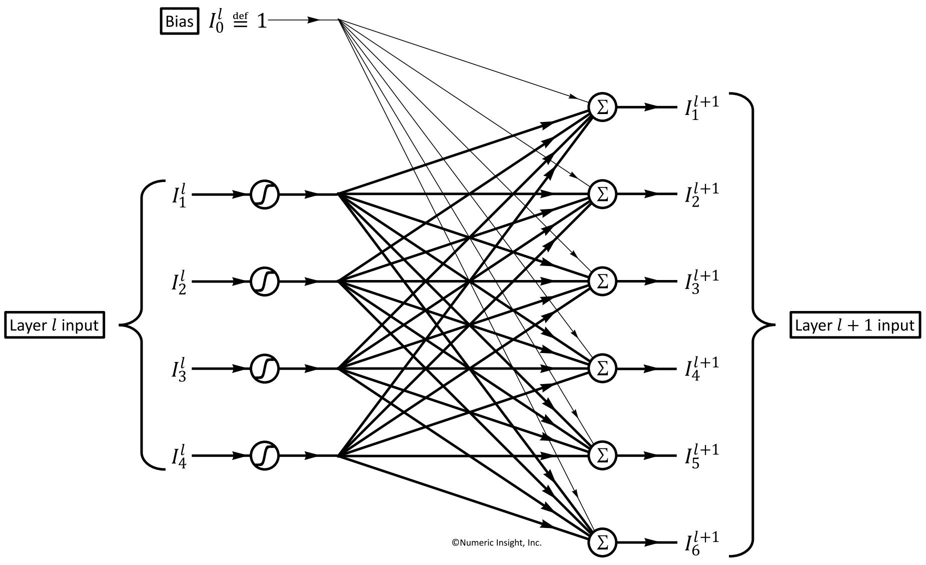

Figure 6: The modified structure of Layer 2 of the multi-layer neural network shown in previous figures. The addition of bias results in a slight change in the architecture, and a slight change in the formulas. The input to each layer is augmented by adding an extra bias element I_0^l, equal to 1 by definition. The use of bias generally results in a faster convergence of the training process. See additional notes in Step 3.

6.1 Summary of the backpropagation algorithm

The overall goal of the algorithm is to attempt to find weights such that, for every input vector in the training set, the neural network yields an output vector closely matching the prescribed target vector.

Step 1 Initialize the weights and set the learning rate and the stopping criteria.
Step 2 Randomly choose an input and the corresponding target.
Step 3 Compute the input to each layer and the output of the final layer.
Step 4 Compute the sensitivity components.
Step 5 Compute the gradient components and update the weights.
Step 6 Check against the stopping criteria. Exit and return the weights or loop back to Step 2.

6.2 Details of the backpropagation algorithm

1. Initialize the weights w_{ij}^l to small random values between -1 and +1.
   Set the learning rate and stopping criteria. Try the following settings for initial experimentation

   $$\begin{aligned} r &= 10^{-2} \\ g_{\text{Min}} &= 10^{-3} \\ k &= 0 \\ k_{\text{Max}} &= 10^5 \end{aligned}$$

2. Randomly choose (with replacement) an input vector $\{I_1^1, I_2^1, I_3^1, \ldots, I_{N^1}^1\}$ from $\mathcal{I}$ and the corresponding target $\{t_1, t_2, t_3, \ldots, t_{N^{L+1}}\}$ from $\mathcal{T}$.

3. Compute the input to each layer, $\{I_1^l, I_2^l, I_3^l, \ldots, I_j^l, \ldots, I_{N^l}^l\}$, and output $\{O_1, O_2, O_3, \ldots, O_j, \ldots, O_{N^{L+1}}\}$ coming out of layer L.

   $$I_j^2 = \sum_{i=0}^{N^1} I_i^1 \; w_{ij}^1 \qquad \text{for } j = 1, 2, 3, \ldots, N^2.$$

   $$I_j^l = \sum_{i=0}^{N^{l-1}} \mathcal{A}(I_i^{l-1})\, w_{ij}^{l-1} \qquad \text{for } l = 3, 4, 5, \ldots, L, \quad \text{for } j = 1, 2, 3, \ldots, N^l.$$

   $$O_j = \sum_{i=0}^{N^L} \mathcal{A}(I_i^L)\; w_{ij}^L \qquad \text{for } j = 1, 2, 3, \ldots, N^{L+1}.$$

   Note that bias (see Figure 6) is implemented by allowing the index i to start from 0 in the above summations. Terms such as $I_0^1, \mathcal{A}(I_0^{l-1})$, and $\mathcal{A}(I_0^L)$ must be set to 1.

4. Compute the sensitivity components $\{\delta_1^l, \delta_2^l, \delta_3^l, \ldots, \delta_i^l, \ldots, \delta_{N^l}^l\}$

   $$\delta_i^{L+1} = 2(O_i - t_i) \qquad \text{for } i = 1, 2, 3, \ldots, N^{L+1}.$$

   $$\delta_i^l = \mathcal{A}^{'}(I_i^l) \sum_{j=1}^{N^{l+1}} w_{ij}^l \delta_j^{l+1} \qquad \text{for } l = L, L-1, L-2, \ldots, 2, \quad \text{for } i = 1, 2, 3, \ldots, N^l.$$

5. Compute the gradient components g_{ij}^l (partial derivative of error with respect to weights) and update the weights w_{ij}^l

$$\left.\begin{aligned} g_{ij}^l &= \frac{\partial e}{\partial w_{ij}^l} = \mathcal{A}(I_i^l)\delta_j^{l+1} \\ w_{ij}^l &= \ w_{ij}^l \ - r\, g_{ij}^l \end{aligned}\right\} \qquad \begin{aligned} &\text{for } l = 1, 2, 3, \ldots, L, \\ &\text{for } i = 0, 1, 2, \ldots, N^l, \\ &\text{for } j = 1, 2, 3, \ldots, N^{l+1} \end{aligned}$$

   Note that terms such as $\mathcal{A}(I_0^l)$ must be set to 1.

6. Check against the stopping criteria

   (a) Increment iteration index, $k = k + 1$.
   If $k > k_{\text{Max}}$, EXIT and return weights. Else,

   (b) Check for convergence
   If $|g_{ij}^l| \leq g_{\text{Min}}$ for evey $l = \{1, 2, 3, \ldots, L\}$, $i = \{0, 1, 2, \ldots, N^l\}$, and $j = \{1, 2, 3, \ldots, N^{l+1}\}$, EXIT and return the weights. Else,

   loop back to Step 2.

7 About the author

Shashi Sathyanarayana Ph.D, the founder of Numeric Insight, Inc (www.numericinsight.com) has several decades of expertise in scientific programming, algorithm development and teaching.

At Boston Scientific Corp, he was responsible for the creation of algorithms that power the companys imaging devices employed to assist detection of atherosclerosis.

He is a named inventor on about a dozen patents in machine learning and ultrasound imaging. His interest in medical imaging began when he was a postdoctoral fellow with Professor Richard Gordon, one of the pioneers of Computed Tomography algorithms. It was further shaped by a research fellowship at the Cleveland Clinic Foundation where he contributed to advancements in echocardiography at the world-renowned Heart Center.

Shashi is passionate about data, algorithms, and teaching. He is an active developer of products and creates highly efficient and effective algorithms. He enjoys creating technical vision and actively engages in understanding customer needs.

Shashi's commitment to judiciously apply common sense and advanced analysis to improve products and processes earned him the Six Sigma Black Belt certification by the American Society for Quality.

Although frequently called upon to work on complex algorithms, he believes in the value of keeping it basic; he recognizes the universal customer need for intuitiveness and simplicity in a product.

Email comments to shashi@numericinsight.com

8 LEGAL NOTICE