Artificial Neural Network Algorithms

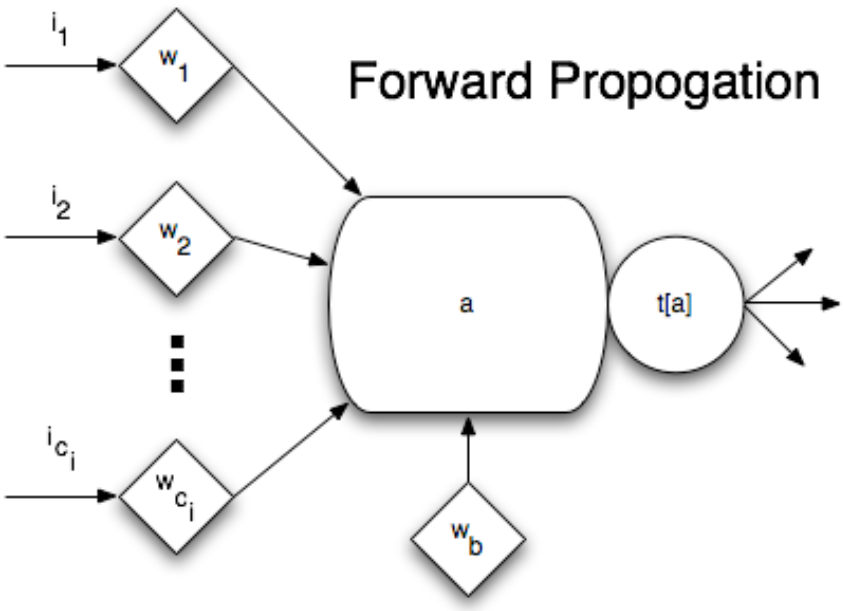
By : Jeffrey Phillips Freeman

This paper hopes to lay out the mathmatical processes a basic artificial neural network will use. It contains the bare minimum for a hobbiest to start playing.

Before you begin processing inputs and learning, the neural network must be created and intialized. The precise structure of the network varies greatly on the task. Usually determining the proper layout is a lot of trial and error and gut instinct. The easiest structure to work with is a "Feedforward Network". In this kind of network the neurons are connected in layers. Each neuron in one layer connects only to neurons in the following layers. No neurons will ever connect out to a neuron in a previous layer. The number of neurons for each layer will have to be guessed, there is no way to get a perfect value. There are some techniques and rules of thumb though. They will be discussed in later volumns. Once a network is created all the weights should be initialized to random, small values.

Several stages must take place for each new input. First the neuron activity is caltulated, then the transfer function calculates the output of a neuron, then each subsequent neuron from the input to the output is executed in this manner. Once the input has propogated to the output the system must learn before processing the next input data. Learning takes place reverse of the output. Ideal output is calculated, and the weights updated, then each previous neuron modifies its weights in this manner from the output back to the input. This process repeats continually as the network learns to predict the ideal outputs.

Processing



$$a = \left(\sum_{s_i=1}^{c_i} \left(w_{s_i} * i_{s_i} \right) \right) + w_b$$

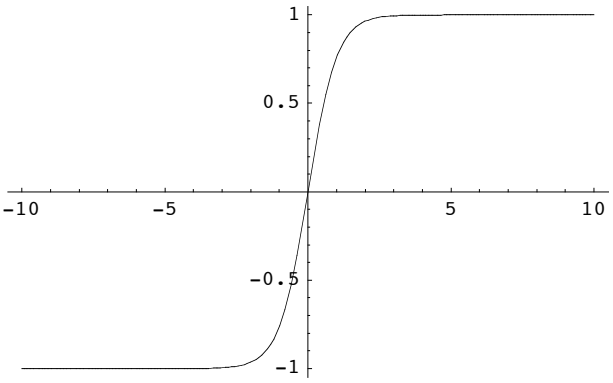
a = Activity
c_i = Synapse Count
s_i = Synapse Index
w_s = Weight @ Index
i_s = Input @ Index
w_b = Bias Weight

Transfer Functions

a = Activity

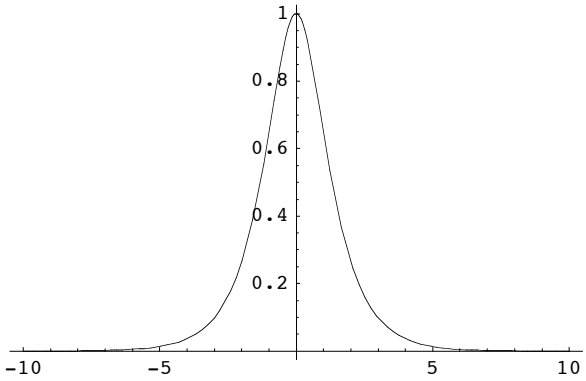
Hyperbolic Tangent

```
tht[a_] := Tanh[a];
Plot[tht[x], {x, -10, 10}];
```



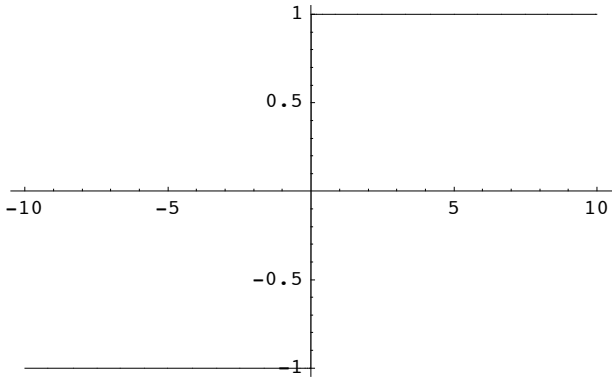
Hyperbolic Secant

```
t_hs[a_] := Sech[a];
Plot[t_hs[x], {x, -10, 10}];
```



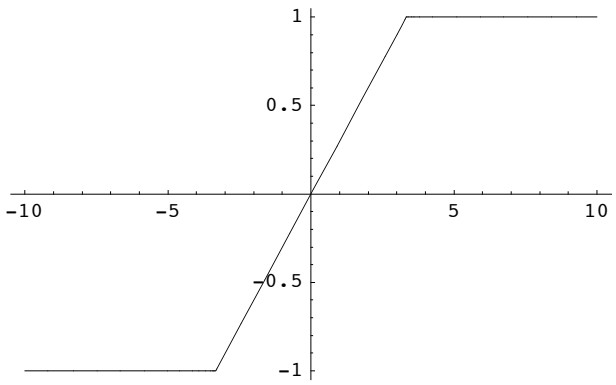
Piecewise

```
t_p[a_] := If[a < 0, -1, If[a > 0, 1, 0]];
Plot[t_p[x], {x, -10, 10}];
```



Ramping

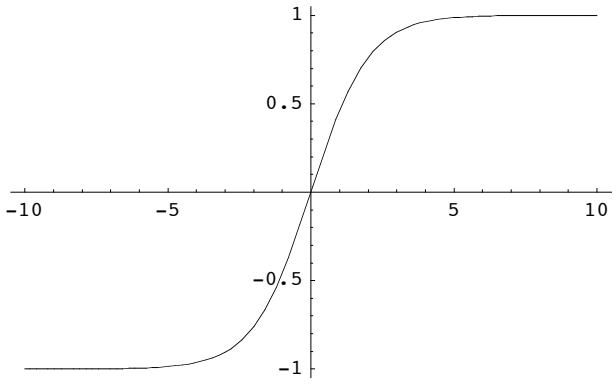
```
S := .3;
tr[a_] := If[a < (-1 / S),
   -1, If[a > (1 / S), 1, a * S]];
Plot[tr[x], {x, -10, 10}];
```



Sigmoid

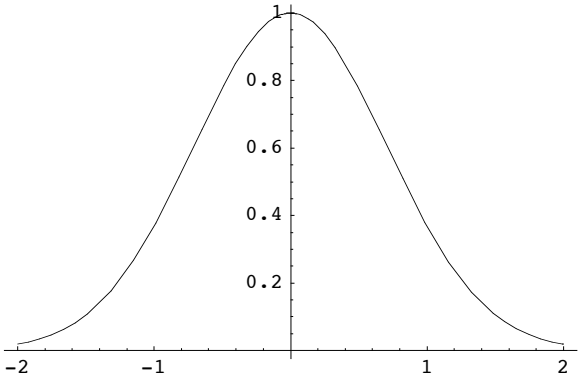
$$t_s[a\_] := \frac{-1 + e^a}{1 + e^a};$$

```
Plot[ts[x], {x, -10, 10}];
```



Gausian

```
t_g[a_] := e^(-a*a);
Plot[t_g[x], {x, -2, 2}];
```

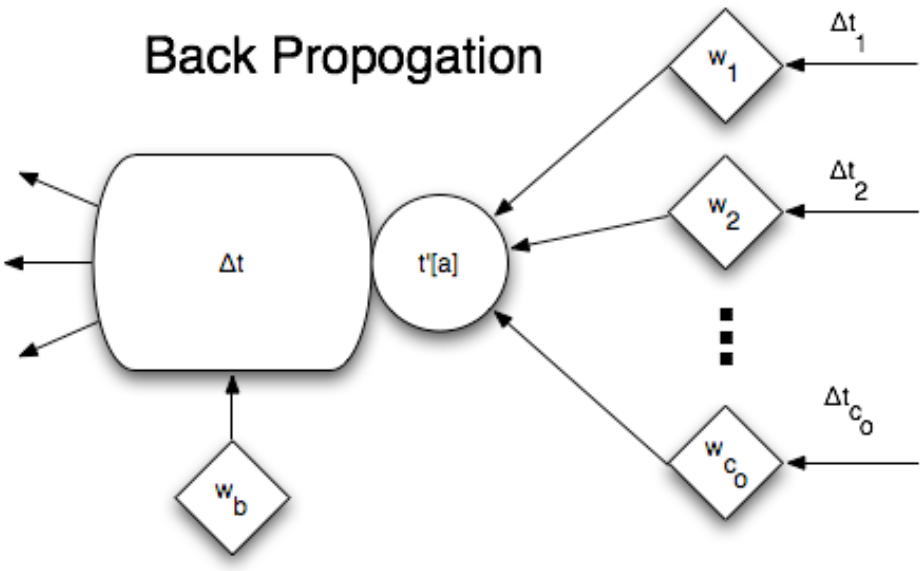


Learning Functions

The learning functions are used to calculate the expected activity for a neuron. The learning function is always the derivative of the transfer function and represented as:

```
t' ()
```

Learning



Non-output Neurons

$$\triangle t = \left(\sum_{s_o=1}^{c_o} (w_{s_o} * \triangle t_{s_o}) \right) * t'[a]$$

$\triangle t$ = Difference between
ideal output and actual output
c_o = Outgoing Synapse Count
s_o = Outgoing Synapse Index
w_{s_o} = Weight @ Index
$\triangle t_{s_o}$ = $\triangle t$ @ Index Neuron
a = Last Activity

Output Neurons

$$\triangle t = \left(\left(\sum_{s_o=1}^{c_o} (w_{s_o} * \triangle t_{s_o}) \right) + (d - t[a]) \right) * t'[a]$$

$\triangle t$ = Expected Activity
d = Desired Output
c_o = Outgoing Synapse Count
s_o = Outgoing Synapse Index
w_{s_o} = Weight @ Index
$\triangle t_{s_o}$ = $\triangle t$ @ Index Neuron
a = Last Activity

Learning Weight

Normal Synapses

$$\triangle w = L * i * \triangle t_o$$

$\triangle w$ = Change in Synapse Weight
L = Learning Rate
i = Synapse Input
$\triangle t_o$ = $\triangle t$ @ Output Neuron

Bias Synapses

$$\triangle w_d = L * \triangle t_o$$

$\triangle w$ = Change in Synapse Weight
L = Learning Rate
$\triangle t_o$ = $\triangle t$ @ Output Neuron

Self Organizing Maps

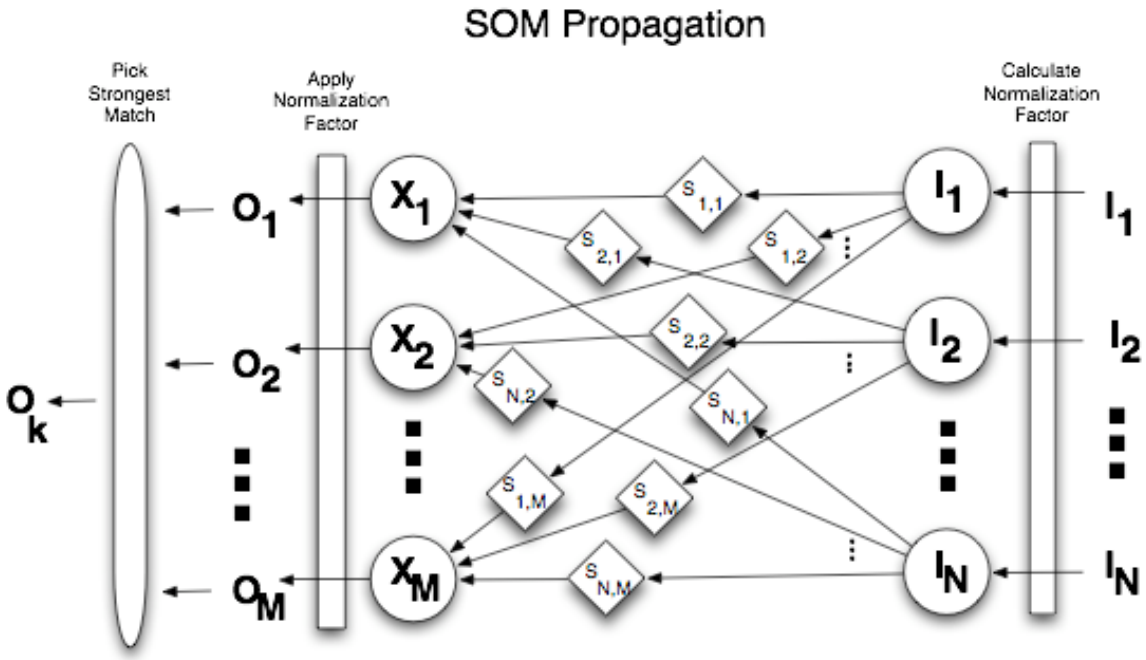
By: Jeffrey Phillips Freeman

Self organizing maps, also refered to as SOM or Kohonen Artificial Neural Networks, are a special type of ANN which uses unsupervized learning to accomplish pattern recognition and static filtering. Since they are unsupervised they only require a properly formated dataset to work from, it needs no feedback fromt he user to learn from. Over time they learn to identify patterns in the data it is presented with which can be used to detect trends that deviate significantly from the normal deviation.

Typically a SOM will have a number of inputs and outputs, the inputs represent the dataset (each input ranging from -1 to 1) and the output is a set of patterns inidicating the certanty that a particular pattern is present for each output, (also ranging from -1 to 1). A good example to visualize this is OCR (optical character recognition) where an image is fed into the input and the output (consisting of 26 output neurons) each represents the chance that a particular letter was presented. So if the neurons for 'a', 'b', and 'c' provided outputs of 0.999, 0.950, and -0.214 then the input image was most likely an 'a', good chance its a 'b', and probably not a 'c'. In order for the network to learn all you need to do is feed it lots of images of letters and over time the outputs will give accurate answers.

A SOM is similar to a basic multi-layer Perceptron except for a few major difference. First, of course, the learning is unsupervized, so you dont need to know the ideal answer. Next, there are no transfer functions, and the prime of the transfer function is therefore not part of the learning algorithm. Also there are no hidden layers, all SOMs only have an input and output layer. Also, unlike the basic multi-layer ANN you must normalize the outputs to the inputs. Aside from these differences youll ind the process is fairly similar.

Processing



Calculate Normalization Factor

The first step for every set of input data is to calculate the normalization factor which is a value used to normalize the outputs in the last step.

$$V = \frac{1}{\sqrt{\prod_{n=1}^{N} (I_n^2)}}$$

$V = \text{normalization factor}$
$N = \text{number of inputs}$
$n = \text{current input index}$
$I_n = \text{Current Input}$

Calculate Output Layer

The next step is to apply the inputs, along with the synapse weights, to calculate the output neurons values.

$$X_m = \sum_{n=1}^{N} (I_n * S_{n,m}), \text{ for } m = 1..M$$

X_m = current Output Neuron
N = number of inputs
n = current input index
m = current output index
I_n = current Input
$S_{n,m}$ = current weight

Apply Normalization

Next we must normalize each output using the normalization factor we calculated earlier. We will also need to convert the output from a range of 0 - 1 to -1 - 1.

$$O_m = 2 * (X_m * V) - 1, \text{ for } m = 1..M$$

O_m = current output
X_m = current output neuron
V = Normalization factor
m = output index

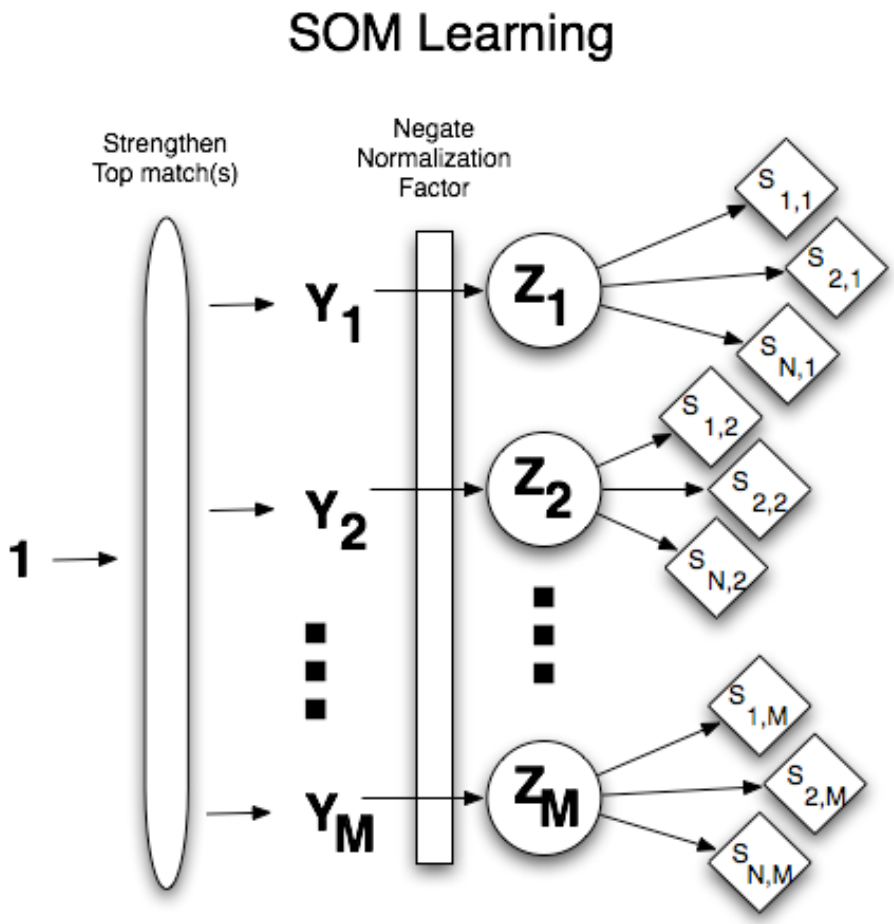
Pick Strongest Match

Next we pick the strongest match of all the outputs. This is simply done by picking the output who's absolute value is the highest. So if the outputs are 0.874, 0.1765, -0.976 then the strongest match will be -0.976.The neuron which produced that value, and the pattern it represents, is the pattern assumed to be present int he input data. This value is also necessary for the learning algorithms for a SOM.

$$O = O_k \text{ where } k = \text{argmax}_m \{ | O_m | \}, \text{ for } m = 1..M$$

Learning

SOM learning is very similar to the Multilayer Perceptron, except the ideal answer is chosen automaticly.Essentially the strongest matching neuron is given a ideal of 1, and all other neurons are given an ideal of -1. This can also be expanded on by giving neighbors to the strongest match some ideal on a gradient between -1 and 1. The normalization factor is then removed, and the strengths are adjusted using a learning equation.



Strengthen Top Match

In this step we strengthen the top match neuron and weaken all other neurons. This is the simplest implmentation; the other approach is to use a gradient by calculating neuron neighbors.

$$Y_m = -1.0, \text{ for } m = 1..M, m \neq k$$
$$Y_k = 1.0$$

Y = Ideal Output
m = Output Index
k = Best Match Output Index
M = Number of Outputs

Negate Normalization Factor

Now we need to remove the normalization factor that was applied to derive the output. To do this we simply do the inverse.

$$Z_m = (Y_m + 1) / (2 * V), \text{ for } m = 1..M$$

Z = Denormalized Ideal Output
Y = Ideal Output
V = Normalization Factor
m = Output Index
M = Number of Outputs

Adjust Weights

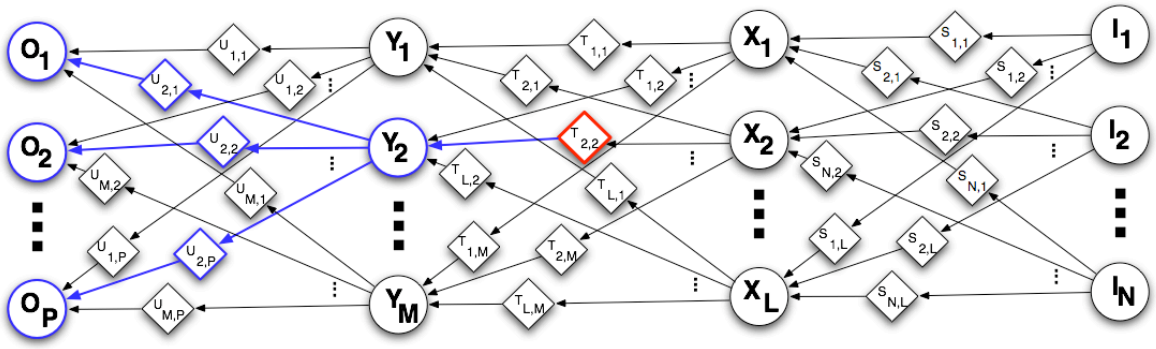
The final step is to adjust the weights, this is done in the same manner as the weights on a perceptron except there is usually no bias weight.

$$\triangle S_{n,m} = L * I_n * (Z_m - O_m),$$
$$\text{for } n = 1..N, \text{ for } m = 1..M$$

$\triangle S$ = Change in Synapse Weight
L = Learning Rate
I = Input Value
O = Output Value
Z = Ideal Output Value
N = Input Count
M = Output Count
n = Input Index
m = Output Index

Connection Optimization

Connection Optimization works on the principle of creating a significane rating greater than 0 for each synapse in the artificial brain. The significance rating indicates how important the particular synapse is to the final output of the brain.



In order to calculate the significance of a synapse there are two general steps. First you must calculate the minium and maximum possible values or the synapse's source neuron calculated from the input forward. Second you must calculate the significance from the target synase forward to the output. In the figure above the red synapse indiciates the synapse for which we are calculating the significance, and the blue area represents the significants values to be calculated.

Each synapse has two different types of significance ratings. The first is called the local significance ratting and is represented by te lower case lambda (λ). The other is the overall significance rating represented by the upper case lamda (Λ). The local significance rating is the significance of a particular synapse compared only with its peer synapses on the input on the same neuron (in the figure above that would be all the T synapses). The overall significance rating is the final value you get at the end of the entire process.

In order to calculate the minimum and maximum values for each neuron you start at the input and multiply each synapse weight by both the maximum and minimum value for each input it is associated with. You wind up with a pair of values for each weighted input however the pair of values for the bias synapse are always the same since the bias only allows for a default input of 1. Take the lowest of each pair and sum them all together, do the same for the highest of each pair.This now gives you the highest and lowest possible activation run each value through the neuron's activation function and you arrive at that neurons minimum and maximum values. Repeat this process for each subsequent neurons.

When representing minimum and maximum values mathmatically the minimum value is represented with the greek letter α and the maximum value by the greek letter Ω. In the example above the allowed input value range will be I_{n_α} to I_{n_Ω} therefore we would calculate the minimum and maximum value of X_l (representing any neuron in the X layer above where l is the destination neuron's index and n is the source input's index) as follows:

$$X_{l_\Omega} = \sum_{n=1}^{N} \begin{cases} S_{n,l} * I_{n_\Omega} & \text{if } S_{n,l} * I_{n_\alpha} < S_{n,l} * I_{n_\Omega} \\ S_{n,l} * I_{n_\alpha} & \text{if } S_{n,l} * I_{n_\alpha} \geq S_{n,l} * I_{n_\Omega} \end{cases}$$

$$X_{l_\alpha} = \sum_{n=1}^{N} \begin{cases} S_{n,l} * I_{n_\alpha} & \text{if } S_{n,l} * I_{n_\alpha} < S_{n,l} * I_{n_\Omega} \\ S_{n,l} * I_{n_\Omega} & \text{if } S_{n,l} * I_{n_\alpha} \geq S_{n,l} * I_{n_\Omega} \end{cases}$$

The next step is to calculate the target synapse's local significance rating as well as the local significance rating of every synapse between it and the output of the network. To calculate the local significance rating of a synapse you take the absolute value of the weight of the target synapse and divide it by the sum of all the absolute values of the wieights of all the synapses on the same neuron. You can represent this mathmatically as for the above example as:

$$\lambda = \left(\frac{|T_{x,m}|}{\sum_{l=1}^{L} \left| T_{l,m} * \begin{cases} X_{l_\alpha} & \text{if } |X_{l_\Omega}| < |X_{l_\alpha}| \\ X_{l_\Omega} & \text{if } |X_{l_\Omega}| \geq |X_{l_\alpha}| \end{cases} \right|} \right)$$

The final step is calculating the overall significance rating for the target synapse. In order to do this first you must determine every possible path between the target synapse and any output neuron. Once that is done multiple the target synapse's local significance rating by that of each synapse in a single path between the target and the output. Then add together all the results or all the paths. The result will be your overall significance rating for the target synapse. For the example above youd get the overall signiicance rating for the red synapse as follows:

$$T_{2,2_\Lambda} = \sum_{p=1}^{P} (T_{2,2_\lambda} * U_{2,p_\lambda})$$

Once youve aquired all the overall significance ratings for all the synapses within a network you can then eliminate the synapses with the lowest significance and ensure that you will effect the accuracy of the network minimally. By constantly

removing the least significant synapses and creating new random synapses the network can eventually learn to adapt the most effiecient set of connections possible without having useless synapses that slow down the time to process but add very little to the accuracy of the final result.