A Neural Network Primer

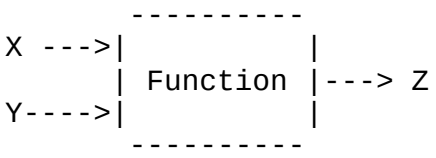
David W. Croft
CompuServe [76600,102]
Internet CroftDW@Portia.Caltech.Edu
modem (818) 793-2426
Pasadena, CA

Revision 2
1994 January 12th

This paper gives the basics of the subject of neural networks. It is recommended for those new to the subject of neural networks and only assumes that the reader has a knowledge of basic algebra.

Please feel free to distribute this paper as you wish but please distribute the entire paper without deletions and limit your corrections and additions to the space below the bottom line after the last paragraph of the original text. If you wish for your correction or modification to be changed in the body of the original text, please send your recommendation to one of my computer addresses above.

A function is a mysterious black box with inputs and an output.



The above function has 2 inputs, X and Y, and one output, Z.

Mysterious black boxes that take in inputs and produce an output are called "functions". Sometimes we don't know what goes on inside the function to produce the output from the inputs so that's why we call them "black boxes".

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

The above table maps out a possible function. It is called the "Or" function. Note that Z is zero when X and Y are both zero and Z is one whenever X "or" Y is a one. Mapping out a function in a table as is shown above is known as creating a "truth-table".

X	Y	Z
0	0	0

```
0  1  |  0
1  0  |  0
1  1  |  1
```

The function mapped out in the truth-table above is known as the "And" function. Note that Z equals 1 only when X "and" Y both equal 1.

Note that the numerical values we have used so far for X, Y, and Z have been limited to only "0" and "1". When you only use two values like this you are using a "binary" or "boolean" number system. "Binary" and "boolean" basically mean "two values". Different ways of expressing boolean values are "1" and "0", "On" and "Off", "High" and "Low", "True" and "False", and "Firing" and "Resting".

Let's say we wanted to make the mysterious black box function "And" not so mysterious. We could do so by describing the inner guts of the box so we could tell how the output is mathematically produced from the output.

```
Z := ( 0.75 * X + 0.75 * Y >= 1.0 )
```

In the above equation,
"+" is addition,
"*" is multiplication,
">=" is an inequality symbol which means "is greater than or equal to",
"()" means "calculate everything in the parentheses first", and
":=" means "is set to the value".

Note that multiplication is always done before any addition in an equation so that 0.75 * 1.0 + 0.75 * 1.0 equals 1.5 and not 1.3125 which it would be if we added 1.0 to 0.75 first then multiplied by 0.75 and 1.0. That is, "0.75 * X + 0.75 * Y" is the same as "(0.75 * X) + (0.75 * Y)", but "0.75 * X + 0.75 * Y" is not the same as "0.75 * (X + 0.75) * Y".

The part of the equation in the parentheses is either true or false. That is, 0.75 * X + 0.75 * Y is either greater than or equal to 1.0 or it ain't. Since that part is either true or false depending on our inputs X and Y, Z will be "set to the value" of true or false. If Z gets set to "true", we'll call it a "1" and if it gets set to "false", we'll call it a "0". This brings us back to our boolean/binary number system of only two values, one and zero.

```
Z := ( 0.75 * X + 0.75 * Y >= 1.0 )
```

Let's try values of zero for X and Y in the above equation and see what the output Z becomes.

```
Z := ( 0.75 * 0.0 + 0.75 * 0.0 >= 1.0 )
Z := ( 0.0 + 0.0 >= 1.0 )
Z := ( 0.0 >= 1.0 )
Z := false (since 0.0 is not greater than or equal to 1.0)
Z := 0
```

Now let's try X = 0 and Y = 1.

```
Z := ( 0.75 * 0.0 + 0.75 * 1.0 >= 1.0 )
Z := ( 0.0 + 0.75 >= 1.0 )
Z := ( 0.75 >= 1.0 )
```

```
Z := false
Z := 0
```

Now let's try X = 1 and Y = 0.

```
Z := ( 0.75 * 1.0 + 0.75 * 0.0 >= 1.0 )
Z := ( 0.75 + 0.0 >= 1.0 )
Z := ( 0.75 >= 1.0 )
Z := false
Z := 0
```

Now let's try X and Y equal 1.

```
Z := ( 0.75 * 1.0 + 0.75 * 1.0 >= 1.0 )
Z := ( 0.75 + 0.75 >= 1.0 )
Z := ( 1.5 >= 1.0 )
Z := true
Z := 1
```

Here is a table of our results.

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

As you can see, this is the truth-table of the "And" function we saw earlier. Thus, the equation Z := (0.75 * X + 0.75 * Y >= 1.0) describes the "And" function. This equation is called the "transfer function" because it "transfers" the inputs X and Y into the output Z.

In our equation, X and Y are both multiplied by 0.75. The multipliers of 0.75 are known as "weights" because they "weight" (as in make lighter or heavier) the inputs. In this particular equation, the "weights" could be said to be making the inputs "lighter" because they are multiplying their values by three-fourths, or 0.75.

In our equation, our inputs X and Y are both multiplied by weights of 0.75 then added together as in "0.75 * X + 0.75 * Y". This is known as the "weighted sum of the inputs".

In our equation, the weighted sum of the inputs ("0.75 * X + 0.75 * Y") had to be greater than or equal to a value of 1.0 in order for Z to be "true" or "1". We'll call the value of 1.0 the "threshold value" because anything less than the threshold value causes Z to be "false" or "0".

In our equation, the output Z can either be true or false, 1 or 0. In neural terms, we'll say that the output Z is "firing" if it equals true (1) and that it is "resting" if it equals false (0).

Our equation, Z := (0.75 * X + 0.75 * Y >= 1.0) can now be re-worded as "the output is firing if the weighted sum of the inputs is greater than or equal to the threshold value. Otherwise, it is resting." This is what neurons do -- fire or not fire depending on whether the sum of their weighted inputs is greater than some threshold value.

Now lets take a look at what happens if we change the values of the

weights in our equation from 0.75 to 1.5.

```
Z := (1.5 * X + 1.5 * Y >= 1.0 )
```

Let's try values of zero for X and Y in the above equation and see what the output Z becomes.

```
Z := ( 1.5 * 0.0 + 1.5 * 0.0 >= 1.0 )
Z := ( 0.0 + 0.0 >= 1.0 )
Z := ( 0.0 >= 1.0 )
Z := false (since 0.0 is not greater than or equal to 1.0)
Z := 0
```

Now let's try X = 0 and Y = 1.

```
Z := ( 1.5 * 0.0 + 1.5 * 1.0 >= 1.0 )
Z := ( 0.0 + 1.5 >= 1.0 )
Z := ( 1.5 >= 1.0 )
Z := true
Z := 1
```

Now let's try X = 1 and Y = 0.

```
Z := ( 1.5 * 1.0 + 1.5 * 0.0 >= 1.0 )
Z := ( 1.5 + 0.0 >= 1.0 )
Z := ( 1.5 >= 1.0 )
Z := true
Z := 1
```

Now let's try X and Y equal 1.

```
Z := ( 1.5 * 1.0 + 1.5 * 1.0 >= 1.0 )
Z := ( 1.5 + 1.5 >= 1.0 )
Z := ( 3.0 >= 1.0 )
Z := true
Z := 1
```

Here is a table of our results.

```
X  Y  |  Z
-----------
0  0  |  0
0  1  |  1
1  0  |  1
1  1  |  1
```

As you can see, this is the truth-table of the "Or" function we saw earlier. Thus, the equation Z := (1.5 * X + 1.5 * Y >= 1.0) describes the "Or" function.

Note that we changed our function from the "And" function to the "Or" function just by changing the weights from 0.75 to 1.5.

```
X  Y  |  Z
-----------
0  0  |  1
0  1  |  0
1  0  |  0
1  1  |  0
```

The above is the truth-table for the "NOr" function. Note that the output in each case is the opposite of what it would be for the "Or" function given the same inputs. That is why we call it the "Not Or" or "NOr" function.

The neural transfer function equation for the "NOr" function is Z := (1.5 * X + 1.5 * Y <= 1.0). Note that this equation is exactly the same as the "Or" function equation with only one difference: the "greater than or equal to" inequality symbol (">=") has been replaced by the "less than or equal to" inequality symbol ("<="). That is, for this function, the output Z is only true/1/firing when the weighted sum of the inputs is less than or equal to, not greater than or equal to, the threshold value of 1.0.

We can change the "<" to a ">" in the equation simply by multiplying both sides of the inequality equation in the parentheses by a negative value.

```
Z := (          1.5 * X +  1.5 * Y   <=        1.0 )
Z := ( -1.0 * ( 1.5 * X +  1.5 * Y ) >= -1.0 * 1.0 )
Z := (         -1.5 * X + -1.5 * Y   >=       -1.0 )
```

You could confirm that the first equation above with the "<" is just the same as the last equation above with the ">" by trying various values of X and Y in both equations and seeing that they both give the same Z each time.

But now suppose we add a constant of 2 to both sides of the inequality.

```
Z := (       -1.5 * X + -1.5 * Y >=       -1.0 )
Z := ( 2.0 + -1.5 * X + -1.5 * Y >= 2.0 + -1.0 )
Z := ( 2.0 + -1.5 * X + -1.5 * Y >=       +1.0 )
```

Note that we have not changed the function by adding 2 to both sides since the output Z for each given X and Y input is still the same but that we have changed the threshold value from -1.0 to +1.0. The extra term in the weighted sum of the inputs of 2.0 can be considered as a constant, non-variable input of 1.0 with a weight of 2.0.

```
Z := ( +2.0 * 1.0 + -1.5 * X + -1.5 * Y >= +1.0 )
```

The reason that we manipulated the algebra to change the "<" to a ">" and the "-1.0" threshold value to a "+1.0" is because our equation must be constrained to a format which neurons "like". That is, the biological/physical implementation of neurons is generally that they only "fire" when the weighted sum of the inputs is greater than, not less than, or equal to some positive, not negative, threshold value.

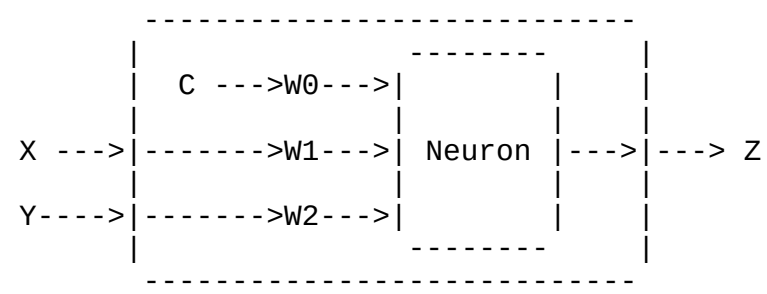
We'll use the symbol "C" for the constant input to distinguish it from the variable inputs of X and Y.

```
Z := ( +2.0 * 1.0 + -1.5 * X + -1.5 * Y >= +1.0 ) becomes
Z := ( +2.0 *   C + -1.5 * X + -1.5 * Y >= +1.0 )
```

While we're at it, we make up some symbols for the multiplying weights for each of the inputs. Let's use "W0" for the weight on the constant input "C" and "W1" and "W2" for the weights of the variable inputs "X" and "Y" respectively.

```
Z := ( W0 * C + W1 * X + W2 * Y >= 1.0 )
```

Our neural black box now looks like the following where the inputs are multiplied by the weights when they enter the function.



We'll then go on to make up a symbol for the threshold value of 1.0 and call it "T".

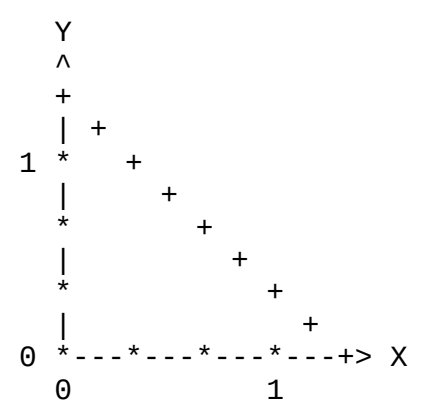
```
Z := ( W0 * C + W1 * X + W2 * Y >= T )
```

You may have noticed the inequality equation (the part of our neural equation in the parentheses) maps out the equation of a line. Consider that

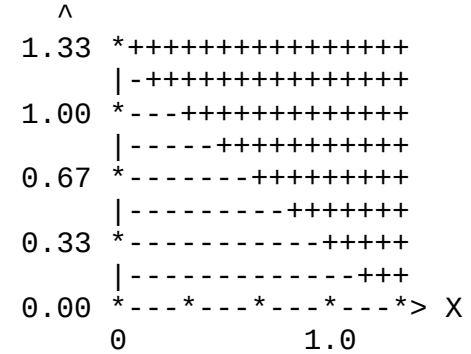
```
W0 * C + W1 * X + W2 * Y >= T becomes
W2 * Y >= -W1 * X - W0 * C + T
Y >= ( -W1 / W2 ) * X + ( -W0 / W2 ) * C + T / W2
Y >= M * X + K
  where M, the slope of the line, is ( -W1 / W2 )
    and K, the Y-axis intersection of the line, is ( -W0 / W2 ) * C + T /
```

If we plot out this equation for the "And" function which has an equation of

```
Z := ( W0 * C +   W1 * X +   W2 * Y >=   T )
Z := (    0.0 + 0.75 * X + 0.75 * Y >= 1.0 )
Z := ( 0.75 * Y >= -0.75 * X - 0.0 + 1.0 )
Z := ( Y >= -1.0 * X + 1.33 )
```



But since we're dealing with an inequality equation, Z is "true" whenever Y is greater than or equal to -X + 1.33. This creates a shaded region above the line in the X-Y plane where Z is "true" as symbolized by "+" and a shaded region below the line where Z is "false" as symbolized by "-".

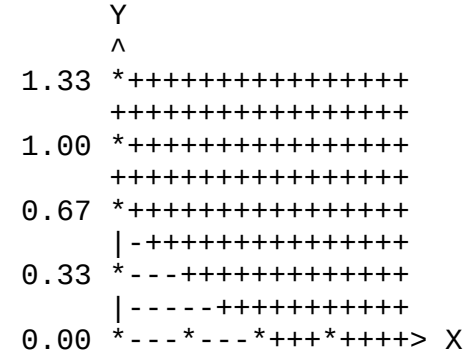


Note that Z is "true" for the coordinates (X,Y) = (1,1) but "false" for (0,0), (0,1), and (1,0). This agrees with the truth-table for the "And" function that we saw earlier.

(X,Y)	Z
(0,0)	0
(0,1)	0
(1,0)	0
(1,1)	1

Similarly, the graph of the "Or" function follows.

```
Z := ( W0 * C +  W1 * X +  W2 * Y >=   T )
Z := (    0.0 + 1.5 * X + 1.5 * Y >= 1.0 )
Z := ( 1.5 * Y >= -1.5 * X - 0.0 + 1.0 )
Z := ( Y >= -1.0 * X + 0.67 )
```



Note that Z is "false" for the coordinates (X,Y) = (0,0) but "true" for (0,1), (1,0), and (1,1). This agrees with the truth-table for the "Or" function that we saw earlier.

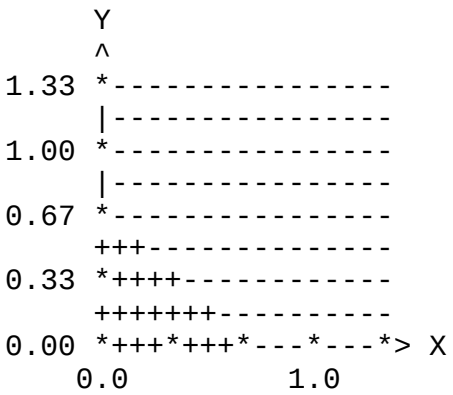
(X,Y)	Z
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	1

Similarly, the graph of the "NOr" function follows.

```
Z := (  W0 *   C +   W1 * X +   W2 * Y >=   T )
Z := ( 2.0 * 1.0 + -1.5 * X + -1.5 * Y >= 1.0 )
```

```
Z := ( -1.5 * Y >= 1.5 * X - 2.0 + 1.0 )
Z := ( Y <= -1.0 * X + 0.67 )
```

Note that the inequality flipped from ">" to "<" because we multiplied both sides by a negative number (1/W2 = 1/-1.5).



Note that Z is "true" for the coordinates (X,Y) = (0,0) but "false" for (0,1), (1,0), and (1,1). This agrees with the truth-table for the "NOr" function that we saw earlier.

(X,Y)	Z
(0,0)	1
(0,1)	0
(1,0)	0
(1,1)	0

Below is the truth-table for the "NAnd" or "Not And" function. Note that the output Z is the opposite of what we had for the "And" function given the same inputs.

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

The equation for the "NAnd" function is

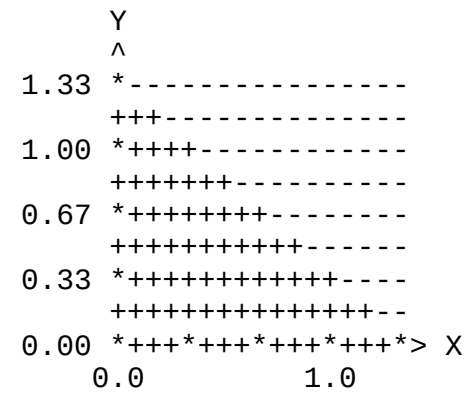
```
Z := ( 0.75 * X + 0.75 * Y <= 1.0 )
```

which is the same as that of the "And" function except that the inequality symbol is flipped from ">=" to "<=". We now manipulate the equation to get it into our neural equation format of Z := (W0 * C + W1 * X + W2 * Y >= T).

```
Z := ( 0.75 * X + 0.75 * Y <= 1.0 )
Z := ( -0.75 * X + -0.75 * Y >= -1.0 )
Z := ( 2.0 * C + -0.75 * X + -0.75 * Y >= +1.0 )
```

We then manipulate the neural equation for the "NAnd" function to put it into a format which makes it easy to graph.

```
Z := ( 2.0 * C + -0.75 * X + -0.75 * Y >= +1.0 )
Z := ( -0.75 * Y >= +0.75 * X + -2.0 * C + 1.0 )
Z := ( -0.75 * Y >= +0.75 * X + -1.0 )
Z := ( Y <= -1.0 * X + 1.33 )
```



Note that Z is "false" for the coordinates (X,Y) = (1,1) but "true" for (0,0), (0,1), and (1,0). This agrees with the truth-table for the "NAnd" function that we saw earlier.

(X,Y)	Z
(0,0)	1
(0,1)	1
(1,0)	1
(1,1)	0

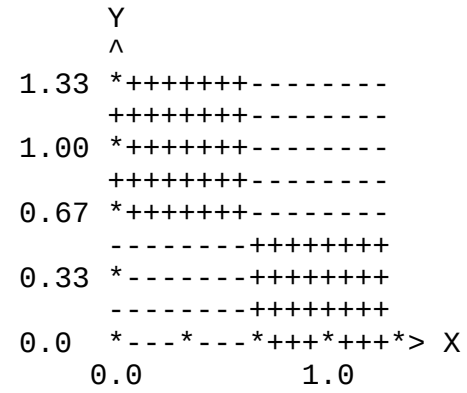
Below is the truth-table for the "XOr" function. Note that the output Z is only "1" when one the inputs X or Y is exclusively (by itself, one and only one of the two inputs) "1". That is why it is called the "exclusive or" or "XOr" function. You will notice that when both of the two inputs are "1" the output is "0".

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

Unfortunately, this equation cannot be implemented using our neural equation format of "the output is firing if and only if the weighted sum of the inputs is greater than or equal to some positive threshold value". To see this, you might try to pick combinations of W0, C, W1, X, W2, Y, and T in the equation

```
Z := ( W0 * C + W1 * X + W2 * Y >= T )
```

to produce the "XOr" function. This would be impossible as will be explained by the graph of "XOr" below.

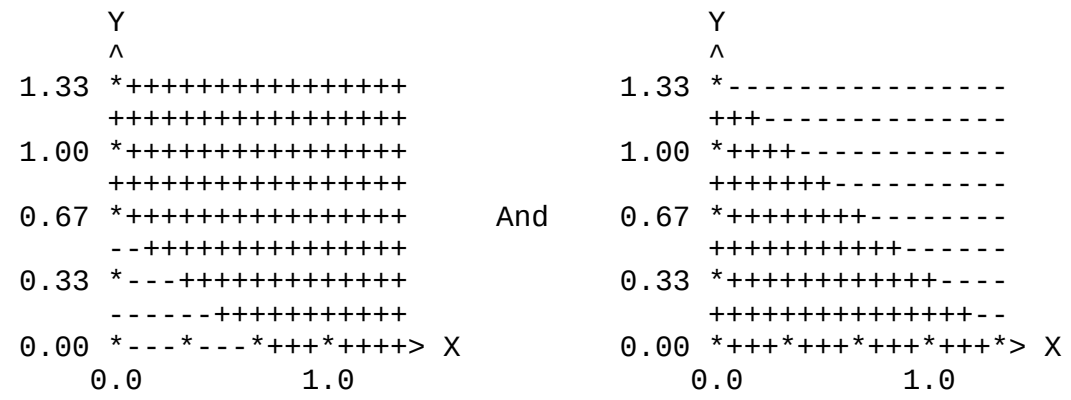


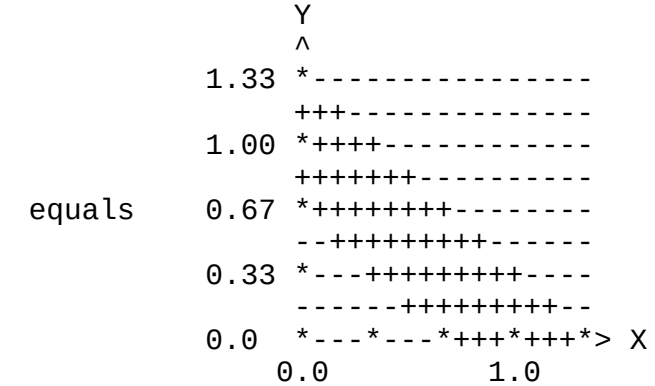
Looking at this graph, we see that we cannot divide the regions where Z is true ("+") from the regions where Z is false ("-") by a single line. That is, no matter where we place a line on the graph, we cannot have all of the "+"s on one side of the line and all of the "-"s on the other side. This is known as the problem of linear inseparability. That is, you cannot separate the regions by a single line. Thus, you cannot use our neural equation of

Z := (W0 * C + W1 * X + W2 * Y >= T)

to generate linearly inseparable functions.

The solution to problem of linearly inseparability is to create these functions using combinations of neural equations. Below are the graphs of the "Or" function and the "NAnd" function. They have certain regions where in both graphs Z is firing ("+"). This intersection of where both are firing is shown in the third, merged graph below them.





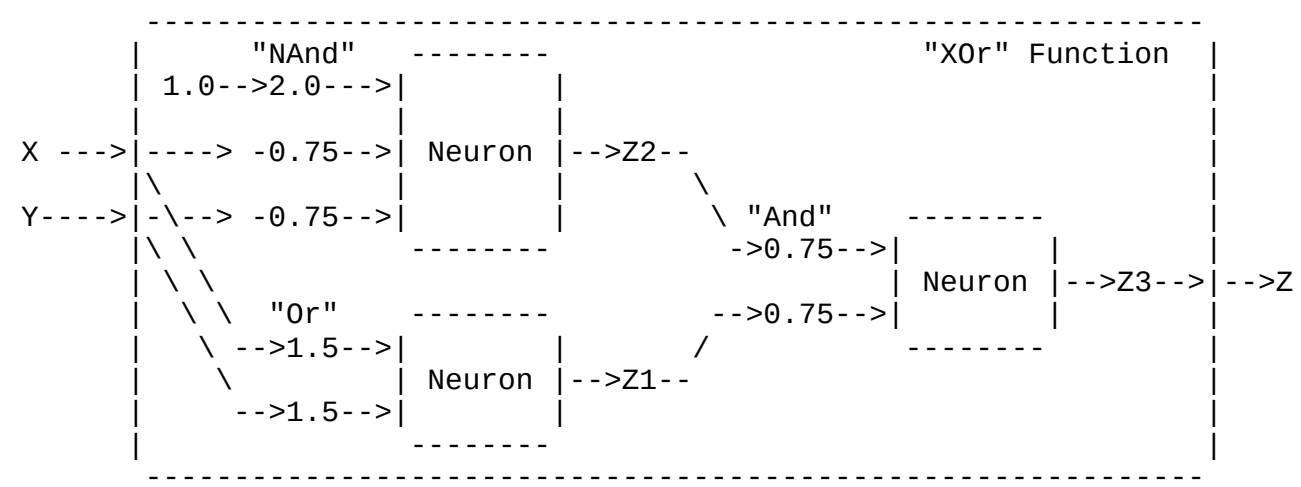
Note that we used the "And" function to merge the two functions together. We chose the "And" function to do this because the output of the "And" function is "1" if and only if both inputs are "1". Thus, the merged output graph has z Z of "1" at a location if and only if both of the input graphs had a Z of "1" at that location.

Note also that while the merged graph does not look exactly like the graph we originally presented for the "XOr" function, it does produce the same output Z for the possible inputs of (X, Y) equals (0,0), (0,1), (1.0), and (1,1).

This can also be seen using truth-tables. Note that the final result is the truth-table for the "XOr" function.

```
 X  Y | Z1          X  Y | Z2        X  Y | Z3
----------         ----------       ----------
 0  0 | 0           0  0 | 1         0  0 | 0
 0  1 | 1    And    0  1 | 1    =    0  1 | 1
 1  0 | 1           1  0 | 1         1  0 | 1
 1  1 | 1           1  1 | 0         1  1 | 0
```

Our black box for the "XOr" function now has three neurons in it. A collection of neurons connected together is a "network" of neurons. Thus, the "XOr" function has been created using a "neural network".



Our neural network equation can be created by combining neural equations.

```
Z1 := X  "Or"   Y
Z2 := X  "NAnd" Y
Z := Z3 := Z1 "And" Z2
Z := ( X "Or" Y ) "And" ( X "NAnd" Y )
Z := ( 1.5 * X + 1.5 * Y >= 1.0 ) "And"
     ( 2.0 + -0.75 * X + -0.75 * Y >= 1.0 )
Z := ( 0.75 * ( 1.5 * X + 1.5 * Y >= 1.0 ) +
       0.75 * ( 2.0 + -0.75 * X + -0.75 * Y >= 1.0 ) >= 1.0 )
```

Let's try X and Y = "1".

```
Z := ( 0.75 * ( 1.5 * 1.0 + 1.5 * 1.0 >= 1.0 ) +
       0.75 * ( 2.0 + -0.75 * 1.0 + -0.75 * 1.0 >= 1.0 ) >= 1.0 )
Z := ( 0.75 * ( 1.5 + 1.5 >= 1.0 ) +
       0.75 * ( 2.0 + -0.75 + -0.75 >= 1.0 ) >= 1.0 )
Z := ( 0.75 * ( 3.0 >= 1.0 ) +
       0.75 * ( 0.5 >= 1.0 ) >= 1.0 )
Z := ( 0.75 * ( true  ) +
       0.75 * ( false ) >= 1.0 )
Z := ( 0.75 * 1 + 0.75 * 0 >= 1.0 )
Z := ( 0.75 + 0.0 >= 1.0 )
Z := ( false )
Z := 0
```

This agrees with our truth-table for the "XOr" function where inputs of X and Y equal "1" gives an output Z equals "0".

As you can see, we can duplicate just about any function that does computation by choosing the appropriate weights and number of neurons. Furthermore, you could say that is simply a matter of choosing the appropriate weights alone if you consider that you can have weights of a zero value. To explain this, assume that in the "XOr" function neural network we actually had four neurons instead of three but that the weights from the first three neurons to the fourth neuron were all zero and that the weights from the fourth neuron to the first three neurons were all zero. Any inputs to the fourth neuron would be multiplied by zero and any output from the fourth neuron would be multiplied by zero. Thus, it would be like the fourth neuron was not even connected. That is how we can say that just about any function that does computation can be duplicated by choosing the appropriate weights alone when given an unlimited number of neurons.

Choosing those weights is hard, however. If you consider that a neural network can duplicate any one of an almost unlimited number of computational functions, you will see that you must pick the right set of weights from an almost unlimited number of possible weights to produce the one function that you want. This is where the training algorithm comes in. A training algorithm is a step-by-step procedure for setting the weights to appropriate values to produce your desired function. By applying the training algorithm to the neural network, you are "training" the weights to your desired values.

For example, suppose that we had a neural network with just one neuron in it that we want to train to duplicate the "And" function. Suppose also that we do not initially know that the appropriate weights for this network should be W1 = 0.75 and W2 = 0.75 to produce the "And" function. We might initially set the weights W1 and W2 to 0.0 and hope that by applying our training algorithm the weights will be trained to the appropriate values for the "And" function.

One possible training algorithm that we might use follows.

- A. Set the input (X,Y) to some possible value such as (0,0), (0,1), (1,0), or (1,1).
- B. Calculate the output Z for the given input (X,Y).
- C. If the output Z is too low, increase the weights which had inputs that were "1". If the output Z is too high, decrease the weights which had inputs that were "1".

- D. Continue looping through this process until each possible input combination gives the right output.

If we try this training algorithm for the "And" function with weights set to zero initially, we get the following truth tables. As a reminder,

Z := (W1 * X + W2 * Y >= T) where T := 1.0.

Desired "And" Function	Loop 1 W1=W2=0 Function	Loop 2 W1=W2=0.375 Function	Loop 3 W1=W2=0.75 Function
X Y | Z	X Y | Z	X Y | Z	X Y | Z
0 0 | 0	0 0 | 0	0 0 | 0	0 0 | 0
0 1 | 0	0 1 | 0	0 1 | 0	0 1 | 0
1 0 | 0	1 0 | 0	1 0 | 0	1 0 | 0
1 1 | 1	1 1 | 0	1 1 | 0	1 1 | 1

As you can see, the weights were increased by 0.375 each time that the output for the input of (X,Y)=(1,1) gave an output Z less than the desired output Z of "1". For (X,Y) of (0,0), (0,1), or (1,0), the output Z was always the desired value "0" so we did not need to increase or decrease the weights.

Now let's change our neural network from an "And" function to a "NOr" function by applying the training algorithm again. Note that the "Nor" function required a constant input C of 1.0 with a weight W0 of 2.0. Let's assume that our neural network already had a constant input C=1.0 but that its weight was fixed at 0.0 so we did not bother to show it previously since its weighted input would have been 1.0 * 0.0 = 0.0. We will now include C and allow W0 to be trained away from 0.0.

Z := (W0 * C + W1 * X + W2 * Y >= T) where T := 1.0.

Desired "NOr" Function	Loop 1 W0=0.0 W1=W2=0.75 Function	Loop 2 W0=0.0 W1=W2=0.375 Function	Loop 3 W0=0.375 W1=W2=0.375 Function
C X Y | Z	C X Y | Z	C X Y | Z	C X Y | Z
1 0 0 | 1	1 0 0 | 0	1 0 0 | 0	1 0 0 | 0
1 0 1 | 0	1 0 1 | 0	1 0 1 | 0	1 0 1 | 0
1 1 0 | 0	1 1 0 | 0	1 1 0 | 0	1 1 0 | 0
1 1 1 | 0	1 1 1 | 1	1 1 1 | 0	1 1 1 | 1

You may note that in Loop 2, W0 was still at 0.0 while W1 and W2 were decreased by 0.375. That is because W0 was both raised by 0.375 and lowered by 0.375 in Loop 1 for a net change of 0.0. W0 was raised in Loop 1 when the input was (C,X,Y)=(1,0,0) and the output was Z=0 when we desired a higher output of Z=1. W1 and W2 were not raised for the same input because our algorithm specifies that we increase the weight when we want a higher output only when the input to that weight was a "1". When the input (C,X,Y) was (1,1,1) for Loop 1, all three weights were decreased since the desired output of Z=0 was

lower than the actual output of Z=1 and all three weights had a "1" as an input.

Loop 4	Loop 5	Loop 6	Loop 7
W0=0.375	W0=0.75	W0=1.125	W0=0.0
W1=W2=0.0	W1=W2=0.0	W1=W2=0.0	W1=W2=-0.75
Function	Function	Function	Function

C X Y	Z	C X Y	Z	C X Y	Z	C X Y	Z
1 0 0	0	1 0 0	0	1 0 0	1	1 0 0	0
1 0 1	0	1 0 1	0	1 0 1	1	1 0 1	0
1 1 0	0	1 1 0	0	1 1 0	1	1 1 0	0
1 1 1	0	1 1 1	0	1 1 1	1	1 1 1	0

Loop 8	Loop 9	Loop 10
W0=0.375	W0=0.75	W0=1.125
W1=W2=-0.75	W1=W2=-0.75	W1=W2=-0.75
Function	Function	Function

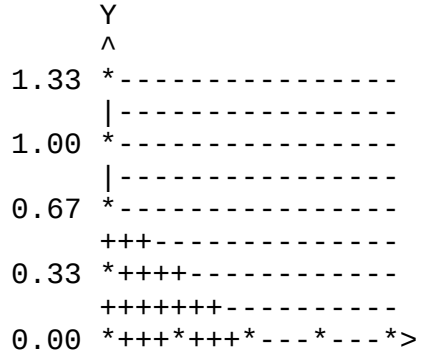
C X Y	Z	C X Y	Z	C X Y	Z
1 0 0	0	1 0 0	0	1 0 0	1
1 0 1	0	1 0 1	0	1 0 1	0
1 1 0	0	1 1 0	0	1 1 0	0
1 1 1	0	1 1 1	0	1 1 1	0

In Loop 10, we finally arrived at weights of (W0,W1,W2)= (1.125,-0.75,-0.75) that duplicated the truth-table for the "NOr" function. Note that earlier we had presented the "NOr" function with weights of (W0,W1,W2)=(2.0,-1.5,-1.5). Both the sets of possible weights with give the same "NOr" function truth table although their graphs are slightly different.

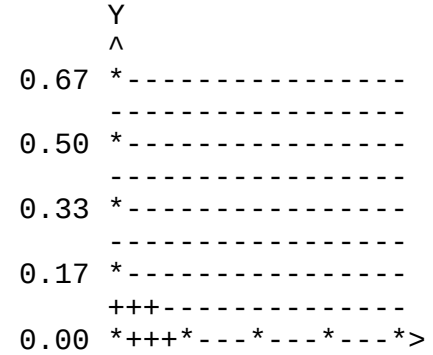
The graph equation for our new "NOr" function is

```
Z := ( 1.125 * 1.0 + -0.75 * X + -0.75 * Y >= 1.0 )
Z := ( -0.75 * Y >= +0.75 * X + -1.125 * 1.0 + 1.0 )
Z := ( -0.75 * Y >= +0.75 * X + -0.125 )
Z := ( Y <= -X + 0.17 )
```

"NOr" Function with (W0,W1,W2)=(2.0,-1.5,-1.5)



"NOr" Function with (W0,W1,W2)=(1.125,-0.75,-0.75)



As you can see, both of these graphs are "on" in the (0,0) corner and

"off" at the coordinates (0,1), (1,0), and (1,1).

Whereas this training algorithm worked for these two very simple neural networks of just one neuron, it may or may not work for other neural networks. Ongoing research into finding an all-purpose training algorithm that will produce the desired weights for any possible function has been unsuccessful so far. However, you can do a lot with a collection of training algorithms which work for some but not all possible neural network functions.

For more information on neural networks and their training algorithms, I highly recommend the book "Neural Computing: Theory and Practice" by Philip D. Wasserman, 1989. This book gives a comprehensible examination of neural networks and describes the equations in such a manner that anyone with basic programming skills can implement them in a computer program without having to know more mathematics than one would have with just basic programming skills. Furthermore, the equations are not written in the syntax of any give computer programming language so it is readable by programmers of any computer language and non-programmers. While the book is currently four or more years old, the subject material within still comprehensively covers the current neural network basics.

APPENDIX A -- A Collection of Boolean Neura Functions

Neuron N0 is constantly firing true which is required for some functions such as the "Not" function.

The threshhold is 1.0 for all neurons.

N-Input NAND

N-Input NOR

2-Input "XOr"

N1	N2	N5
0	0	0
0	1	1

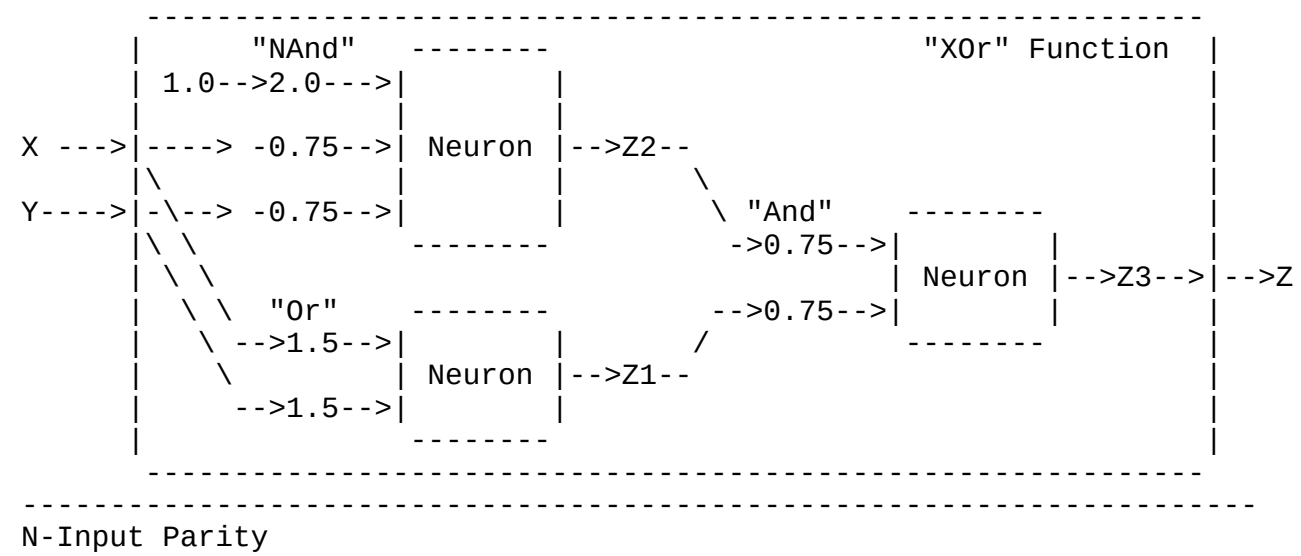
```
 1  0 |  1
 1  1 |  0
```

```
     W0 W1 W2 W3 W4 W5
    ------------------
N0 |  0  0  0  0  0  0
N1 |  0  0  0  0  0  0
N2 |  0  0  0  0  0  0
N3 |  0 -1 +1  0  0  0
N4 |  0 +1 -1  0  0  0
N5 |  0  0  0 +1 +1  0
```

```
     W0 W1 W2 W3 W4 W5
    ------------------
N0 |  0  0  0  0  0  0
N1 |     0  0  0  0  0
N2 |        0  0  0  0
N3 |           0  0  0
N4 |              0  0
N5 |                 0
```

```
     W0 W1 W2 W3 W4 W5
    ------------------
N0 |
N1 |
N2 |
N3 |
N4 |
N5 |
```

Our black box for the "XOr" function now has three neurons in it. A collection of neurons connected together is a "network" of neurons. Thus, the "XOr" function has been created using a "neural network".



End of original text.

Transcribed to HTML on 1997-10-27 by David Wallace Croft .