A Basic Neural Net

Neural nets are often compared to neurons in the brain. Even if this is the root of how the concept arose, I don't feel it to be a good example. No one really understands what neurons do. Let's take a more simple example:

Example 1: A Castle Preparing For Seige

Alice (A) wants to let the king know about an oncoming war, she stands on the east front of the castle.
Her colleague, **Bob** (B), stands on the northern front.

At dawn they pass on a message to the **messenger** (M). The messenger is new to the castle and has no idea who to trust.

The messenger takes Alice's message and hears her say 'the war is near!', the messenger quickly goes to Bob who passes on his message and says 'there is nothing to worry about yet'.

The messenger, a little confused, relays the **messages**, the **importance** of *each* **message and** his **personal bias** as to what is happening, to the king.
Note, the messenger took more importance of Alice's message as it seemed more urgent.
The **King**, being rather simple, takes **all** this **information**, *plugs it into his internal* **importance calculator** and decides there is indeed a war!

The King was wrong... all the troops went home, tired from their rushing to the gates.

Alice is known to be scared and to exaggerate things a little, the messenger, now getting to know the people he works for, will **take less importance** to Alice's messages in the future. On the other hand, Bob was telling the truth and is **rewarded with more importance**. His own personal bias of the situation will also change slightly as he gets better at relaying the correct overall message to the king.

Learning:

After he has made the prediction the king needed to see whether he was correct, in order to learn for the future.

note, ***ERROR*** *=* ***difference*** *between* ***PREDICTION*** and **REALITY**
If the king was correct, all importances and biases must be reinforced/ hardened.

However if the king was wrong, the messenger must find those he spoke to and change their importances in proportion to how culpable they were to the prediction, he must also change his own personal bias. All in order to make sure it doesn't happen again!

We now want to start translating the above story into Machine Learning Lingo:

1. This learning process is known as **Backpropagation** (as the messenger propagates his re-evaluations backwards to where he started) and will be covered thoroughly further on.
2. The Kings Calculator is known as the **Activation Function** which is the neurons output and will be covered next.

To put the previous in algebraic terms:

A = Alice's message
B = Bob's message

W_A = importance of Alice's message

W_B = importance of Bob's message
M = personal bias of the Messenger

X = all information
ϕ( ) = the Kings calculator

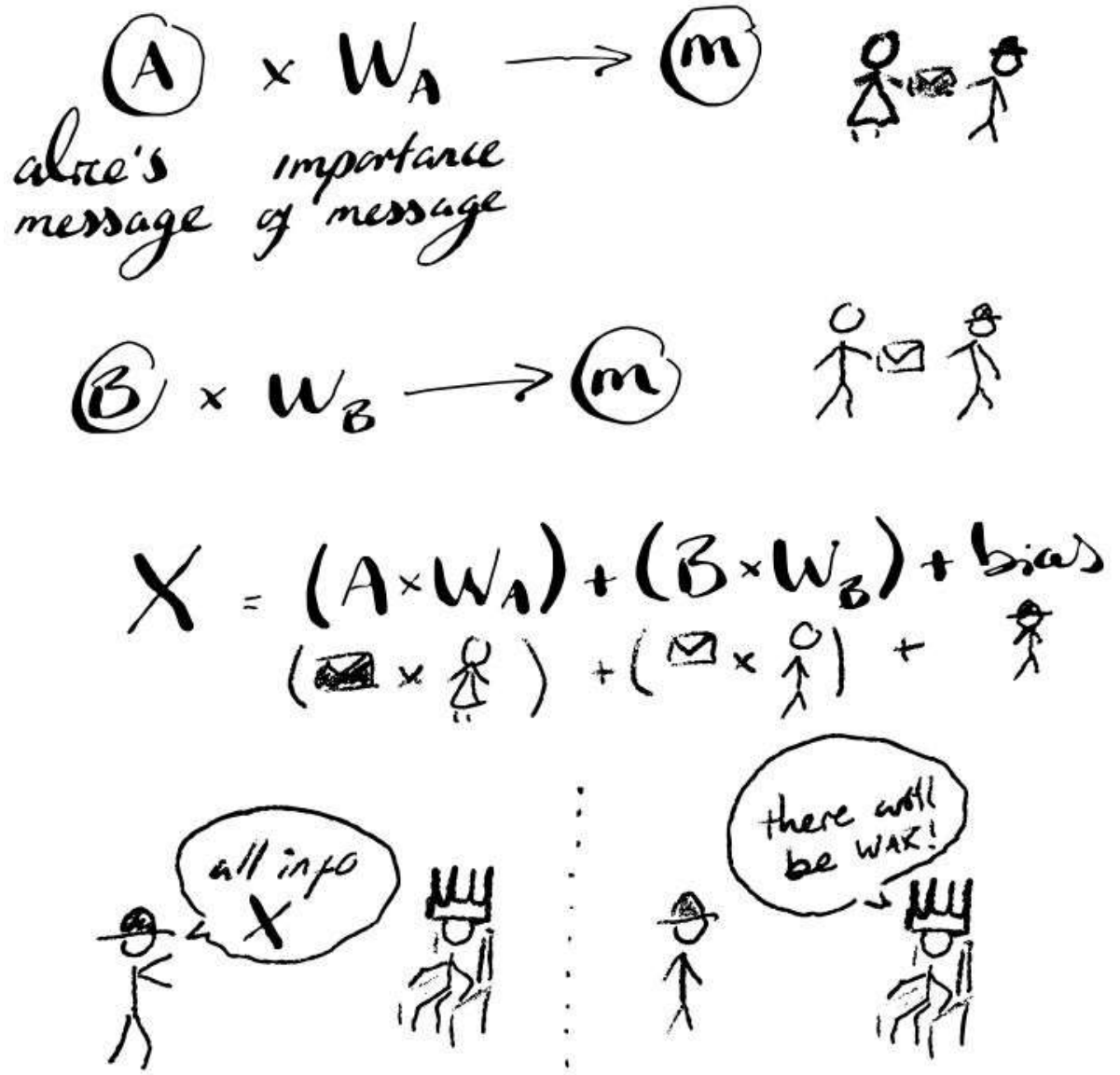
A summary of our story:
Messenger gets Alice's message and Alice's importance and multiplies them together

He then repeats this for Bob

He then adds his own personal bias, now he has all the information

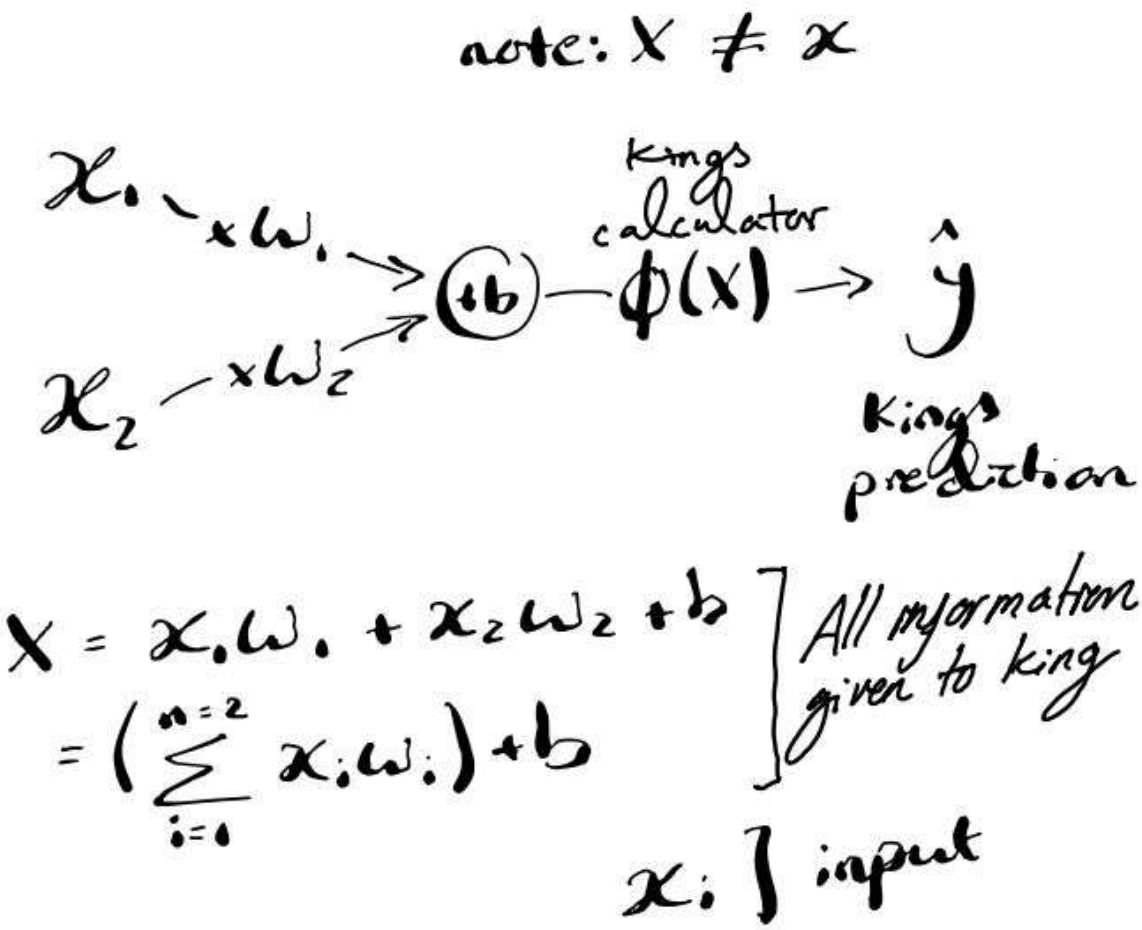
He passes it onto the king who puts it in his calculator and outputs a prediction.

1. Total_A = A * W_A , (multiply Alice's message by its importance)
2. Total_B = B * W_B , (multiply Bob's message by its importance)
3. X = Total_A + Total_B + M , (add them together alongside the bias of the messenger)
4. PREDICTION = ϕ(X), (plug X into the calculator which will spit out whether there is a war or not)



Let's look at the neural network from the story:

It turns out that **Alice and Bob are** equivalent to the **Inputs** of a neural net, the **Messenger** is equivalent to the **Neuron** and the **King's Calculator** is equivalent to the **Neuron's Activation Function**



A re-translation of the above into common mathematical terms you will see in the future

current	common
A = Alice's message	x_1 = input 1
B = Bob's message	x_2 = input 2
W_A = importance of Alice	w_1 = weight 1
W_B = importance of Bob	w_2 = weight 2
M = Messengers bias	b = bias
X = all information	X = all information
ϕ( ) = the Kings calculator	ϕ( ) = the Activation Function
OUTPUT	$\phi(X_1)$ = ϕ_1 = ouput of neuron 1
PREDICTION	$\hat{y}$ = y_{pred} = prediction = output of final neuron
REALITY	y = reality

Summation and Index Notation $\sum_i^n$:
For those that are not used to seeing this please be patient as it will be used a lot throughout

lets take an example where i = 1 = starting index and n = 3 = final index:
$\sum_{i=1}^{n=3}(x_i)$
starting at i=1 we have x_1, then we add 1 to i, giving us x_2 and add this to what we had before,

x_1 + x_2. We then repeat, add one to i and add to what we had before, x_1 + x_2 + x_3. We cannot repeat any further as we have reached our final index, i = n = 3, therefore we have arrived at our answer.

Another example, if (n = 5 and i = 0) => $\sum_{i=0}^{n=5}(x_i) = x_0 + x_1 + x_2 + x_3 + x_4 + x_5$

note, the small number (subscript/ index) after x just denotes its position and has no importance to the value of x, like an id number, Alice could be the 1001th watcher $(x{1001})or the - 20th (x_{-20}). All that matter sis that we can identify her(x_i) and her importance(wi)$ from the others. However, this would make writing the sum equation and many other things unnecessarily difficult so we prefer ordering inputs consecutively

---

Equation summary:

With the tools above, we can re-write everything into a single equation:
In our case, as we have just Alice and Bob as inputs, i = 1 and n = 2

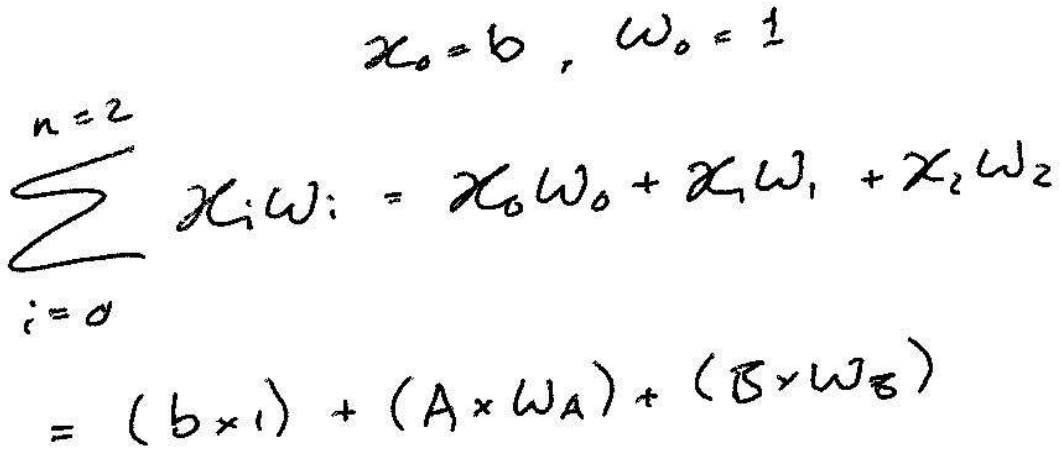
$$\hat{y} = \phi((\sum_{i=1}^{n=2} x_i w_i) + b) = \phi(x_1 w_1 + x_2 w_2 + b) = \phi(X)$$

However, to make things easier for the computer we must simplify even further and move the bias into the $x_i w_i$ part of the equation.

We can create a weight at index 0 and make it = 1 $(w_0 = 1)$
We can also create an input at index 1 and make it = b $(x_0 = b)$

so that:
$\hat{y} = \phi((\sum_{i=1}^{n=2} x_i w_i) + b) = \phi(\sum_{i=0}^{n=2} x_i w_i)$, where $(w_0 = 1$ & $x_0 = b)$

as $x_0 w_0 = b$



Neuron Output Equation:

for any neuron, in any layer, we take the number of inputs, n and work out its output with:

$\hat{y} = \phi(\sum_{i=0}^{n} x_i w_i)$, where $w_0 = 1$ & $x_0 = b$

*The reason we use indexes and obsess about shortening is that it makes writing and processing these equations much easier when we have more inputs. For example in an coloured image of size 1980x1080 we have (1080 * 1980) = 2,092,800 pixels, now taking into account colour. Each pixel takes a red, green and blue input, therefore we have (2,092,800 x 3) = 6,278,400 different inputs.*

Multi-Layered Neural Networks

We have been discussing a single layered neural net, known as a "perceptron". It contains an input layer + a neuron output layer

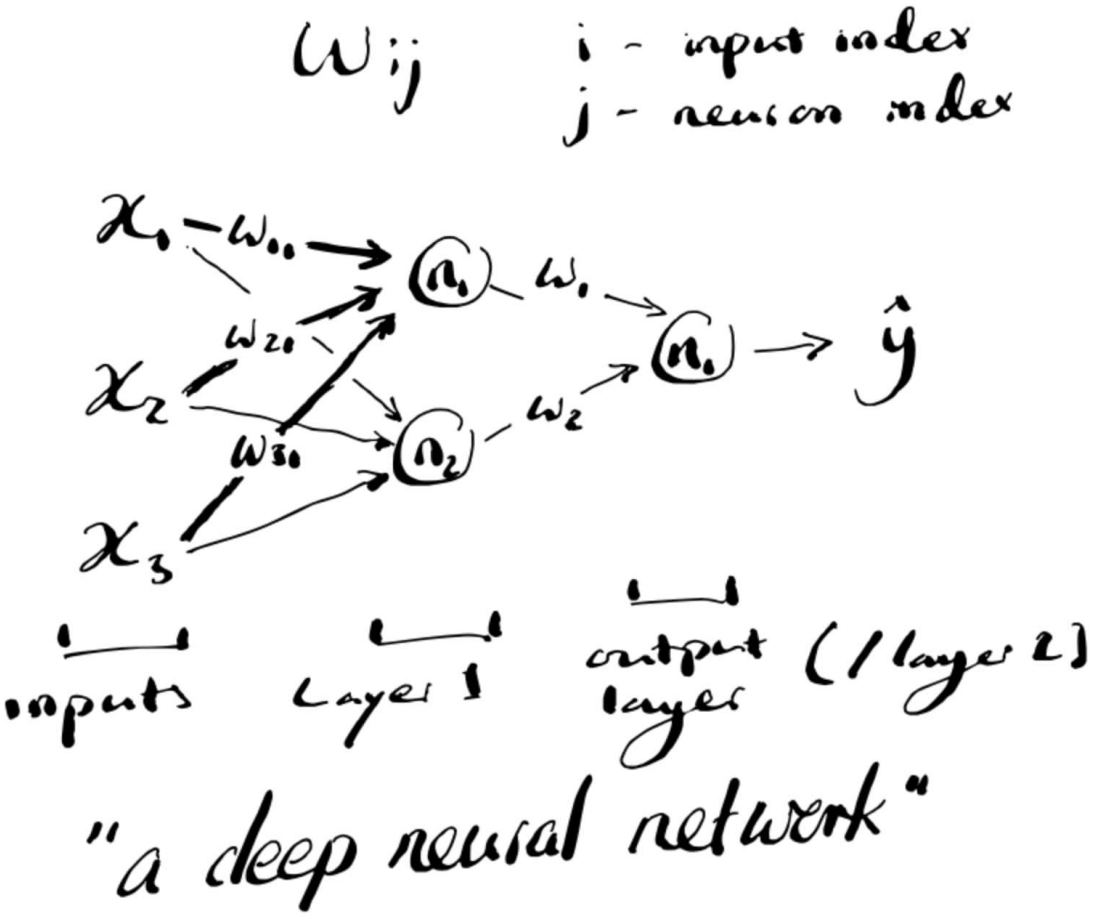
NN's (neural networks) with **more than 1 layer** are known as **"deep"** neural networks
If we were to expand the above and make it "deep", we would do the following:

Say there are 10 watchers (n = 10) and 5 messengers (m = 5) and one **output** important messenger which takes the 5 messengers before it as inputs.

To quickly rewrite the story, for each of the 5 messengers, they go to each individual input, get the message written to all messengers and then multiply it to their specific importance of that watcher, they then add their personal bias and go to the king. This time, the King doesn't trust them enough to make a prediction but nonetheless uses his calculator to hand them back an importance value.

After this first run, we have 5 messengers with 5 different importance values from the King, these, for all intents and purposes will act as watchers or inputs for the final, important, messenger.

The important messenger then goes to each messenger, gets the message (in this case the importance value each individual messenger got from the king) and multiply it with the importance associated to each specific messenger. To top it all off he adds his personal bias and hands it to the king. The King, as before, plugs all this information into his calculator and as he trusts his important messenger, declares a prediction for his kingdom.



Notice, here we have 2 layers of messengers (neurons) and so can call it a "deep" neural network.

Equation Time:

The difference, is that we need another index system for the second layer (the 5 messengers). It is common to use j (as i is already taken).

The next letter (if we had a 3rd layer) would be k, then l, then m. Don't ask me why.

It is also important to understand the importance of indexing, if we look at importance (/weight) we will get 2 indexes, w_{ij}, here this is the importance messenger j gives input i
In the case of w_{jk}, this would mean the importance messenger k (in layer 2) gives messenger j (in layer 1)

In this way, we can see it as the importance j gives k, hence w_{jk}, or the importance i gives j w_{ij}

we will need to use the Neuron Output Equation above

output of neuron 1 (j = 1) in layer 1:
$\phi_1(X_1)$ = $\phi(\sum_{i=0}^{n=10}(x_i w_{i1}))$, (where $x_0 = b_1$ & $w_{01} = 1$)

similarly the output of neuron 2 (j = 2) in layer 1:
$\phi_2(X_2) = \phi(\sum_{i=0}^{n=10}(x_i w_{i2}))$, (where $x_0 = b_2$ & $w_{02} = 1$)
etc for j <= 5

We can then take all these 5 outputs $\hat{y}_1$, $\hat{y}_2$, $\hat{y}_3$, $\hat{y}_4$, $\hat{y}_5$ as inputs for the next layer

output of the last and only neuron 1 (no index required as he is by himself) in layer 2:
$\hat{y}$ = $\phi(\sum_{j=0}^{n=5}(x_j w_j))$, (where $x_0 = b$ & $w_0 = 1$)

note: neuron 1 in layer 1 is NOT equal to neuron 1 in layer 2. The reason we call them neuron 1 is that they are the 1st neuron in their layer

we could rewrite the whole network in a single equation:

$\hat{y} = \phi(\sum_{j=0}^{m=5}(\phi(\sum_{i=0}^{n=10}(x_i w_{ij})w_j)))$

as $\phi(\sum_{i=0}^{n=10}(x_i w_{ij})) = n_j$ the neuron output equation for each messenger j

and $\hat{y} = \phi(\sum_{j=0}^{m=5}(n_j w_j)) = \phi(\sum_{j=0}^{m=5}(\phi(\sum_{i=0}^{n=10}(x_i w_{ij})w_j)))$ = the neuron output equation for the final important messenger.

2-Layered and then 3-Layered Neural Network Output Equation

as follows, a general equation to find the output of a 2 layer NN, with n inputs and m neurons in the second layer

$$\hat{y} = \phi(\sum_{j=0}^{m}(\phi_j w_j)) = \phi(\sum_{j=0}^{m}(\phi(\sum_{i=0}^{n}(x_i w_{ij})w_j)))$$

for a 3-Layered NN we would have the following

$$\hat{y} = \phi(\sum_{k=0}^{m_2}(\phi_k w_k)) = \phi(\sum_{k=0}^{m_2}(\phi(\sum_{j=0}^{m_1}(\phi_j w_{jk}))))$$

$$= \phi(\sum_{k=0}^{m_2}(\phi(\sum_{j=0}^{m_1}(\phi(\sum_{i=0}^{n}(x_i w_{ij}))w_{jk})w_k)$$

ouptuts of layer 1, where j denotes which neuron in layer 1 we are talking about and n is the number of inputs:
$\phi_j = \phi(\sum_{i=0}^{n}(x_i w_{ij})$

outputs of layer 2, where k denotes which neuron in layer 2 we are talking about and m_1 is the number of neurons in layer 1:
$\phi_k = \phi(\sum_{j=0}^{m1}(\phi(X_j)w_{jk})$, where $X_j = \sum_{i=0}^{n}(x_i w_{ij})$

outputs of layer 3, where l denotes which neuron in layer 3 we are talking about and m_2 is the number of neurons in layer 2:
$\phi_l = \phi(\sum_{k=0}^{m2}(\phi(X_k)w_{kl})$, where $X_k = \sum_{j=0}^{m1}(\phi(X_j)w_{jk})$

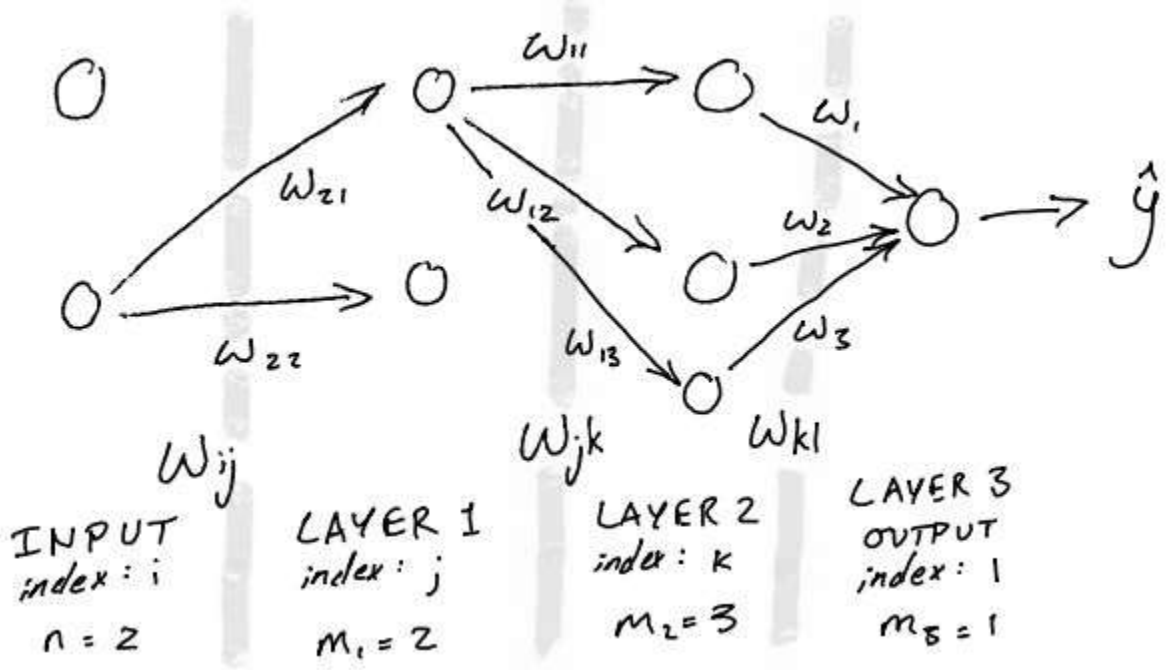
if layer 3 is our final output neuron, $\hat{y} = \phi_l$

We can now see how we can fit this all in one equation, and hopefully you can see the trend that is going on if we were to add more layers:

$$\hat{y} = \phi(\sum_{l=0}^{m_2}(\phi_l w_l m))$$

---

here is a basic multilayered neural net to help you understand the notation used above:



If you haven't understood the idea of summation and indexing, please try these questions to get your head around it. It is a lot to digest at first glance but once you get the hand of it, it will seem much more simple.

http://www.columbia.edu/itc/sipa/math/summation.html
https://courses.lumenlearning.com/ivytech-collegealgebra/chapter/using-summation-notation/