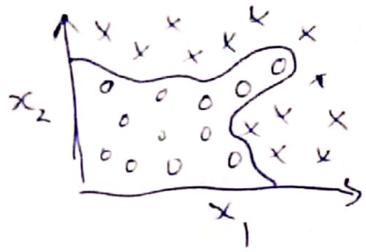
Week 4

Why do we need neural networks?

i) Non-linear classification



To fit a non-linear curve to the data, we need many ~~factors~~ features

$x_1^2, \; x_1x_2, \; x_1x_3 \; \ldots \; x_1x_{100}$

$x_2^2, \; x_2x_3 \; \ldots \; \ldots$

where x_1 = size
x_2 = bedroom
x_3 = floors
⋮
x_{100}

} input set $n = 100$

$\frac{n^2}{2} = 5000$ features (quadratic)

ii) Computer vision:

This as an example of ~~why~~ how n will usually be large.

If we have a 50×50 pixel image
$= 2500$ pixels – in grayscale

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

– varies from 0-255

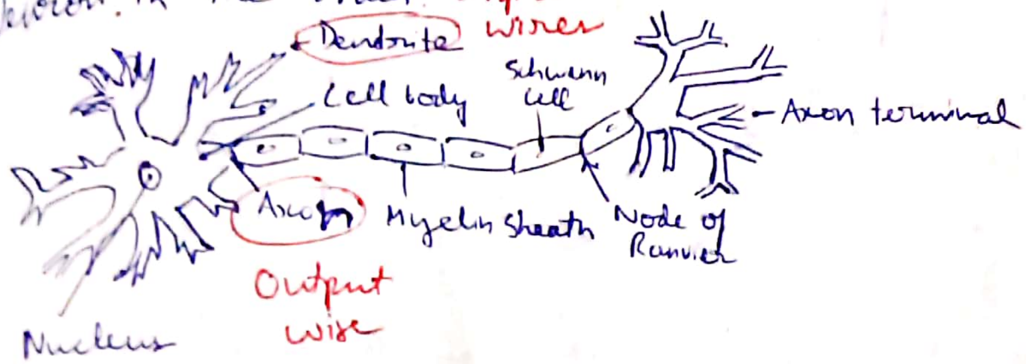
- For this we will have 3 million quadratic features

∴ Therefore it isn't a good idea to use logistic regression by adding quadratic or cubic features since it results in too many features. So we use neural networks which are better for complex non-linear hypothesis that may have a large input set n

Neural Networks

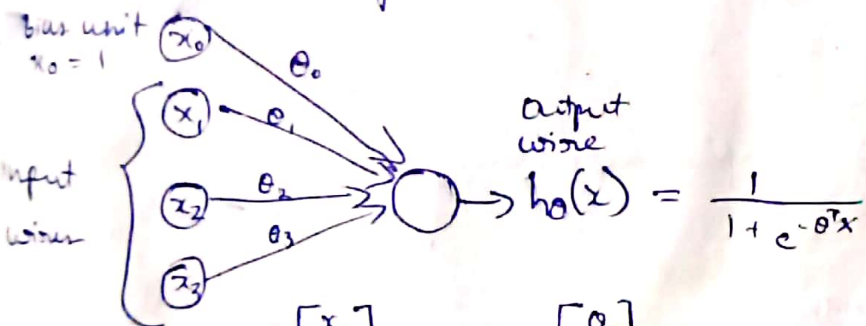
- They are algorithms that mimic the brain.

Neuron in the brain



- It gets input through the dendrites, performs some computation and sends an output through the axon from where it goes to other neurons
- The signals are sent through pulses

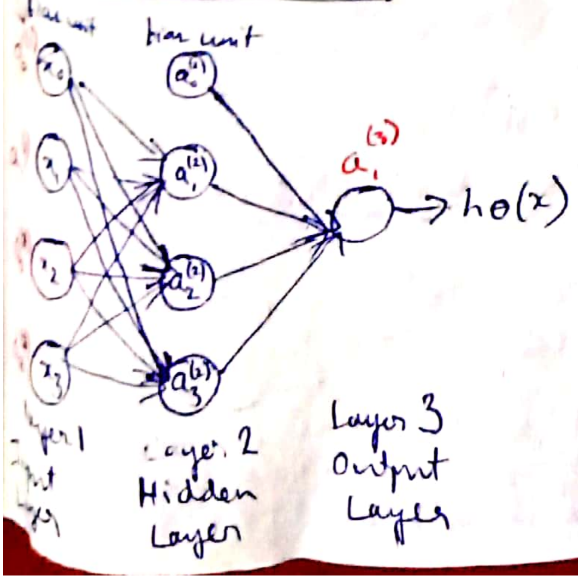
Neuron model : Logistic unit



where $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$, $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$

can also be written as $a_0^{(1)}$ - a_0 of layer 1

Neural Network



$a_i^{(j)}$ - activation of unit i in layer j (the value that is computed)

$\theta^{(j)}$ - matrix of weights controlling function mapping from layer j to $j+1$

$$a_1^{(2)} = g\big(\underbrace{\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3}_{z_1^{(2)}}\big)$$

$$a_2^{(2)} = g\big(\underbrace{\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3}_{z_2^{(2)}}\big)$$

$$a_3^{(2)} = g\big(\underbrace{\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3}_{z_3^{(2)}}\big)$$

activation values are sigmoid of z

$$h_\Theta(x) = a_1^{(3)} = g\big(\underbrace{\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}}_{z^{(3)}}\big)$$

Note: Here Θ_{10} for example mean $a_0^{(1)}$ to $a_1^{(2)}$

$\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$

where s_j is units in layer j

and s_{j+1} is units in layer $j+1$

Exa, in the previous neural network it is 3×4 (Ignoring bias unit)

Vectorized implementation:

Let $z_1^{(2)} = \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3$

and so forth for $z_2^{(2)}$, $z_3^{(2)}$ and $z^{(3)}$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = a^{(1)} = \begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{bmatrix} \quad \text{and} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} x = \Theta^{(1)} a^{(1)}$$

$a^{(2)} = g(z^{(2)}) \leftarrow$ size $\mathbb{R}^3$

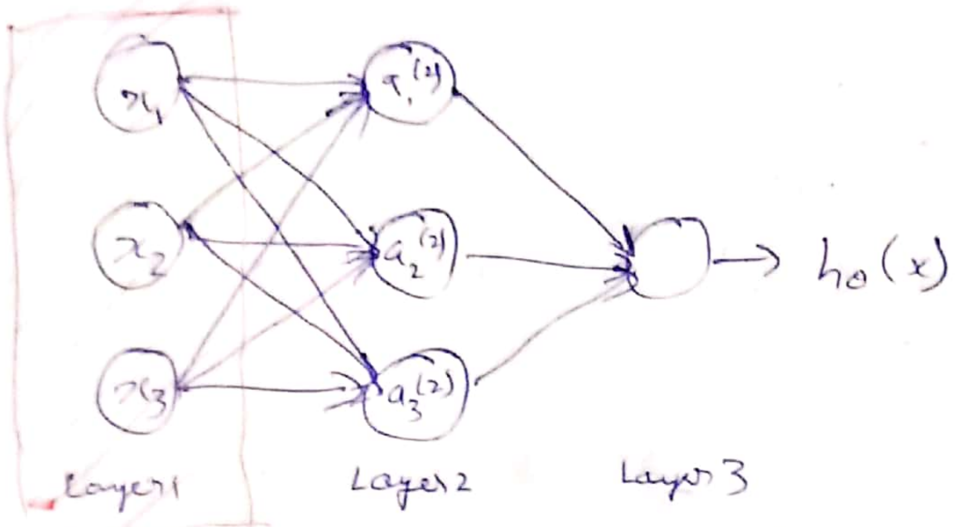
If we add $a_0^{(2)} = 1$, $\leftarrow$ Adding this size $\mathbb{R}^4$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

Its called forward propogation, since we first start of with the activations of the input units then we forward propogate to the hidden layer, compute the activation of the hidden layer and then forward propagate that and compute the activations of the output layer.

Neural Network learning its own features



$$h_\theta(x) = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)$$

If we cover the 1st layer we get a normal logistic regression unit, that uses $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$ instead of x_1, x_2, x_3

But $a_1^{(2)}, a_2^{(2)}$ & $a_3^{(2)}$ are learned from x_1, x_2, x_3 determined by $\Theta^{(1)}$.

Therefore instead of being constrained to x_1, x_2, x_3, the neural network learns its own features $a_1^{(2)}, a_2^{(2)}$ & $a_3^{(2)}$

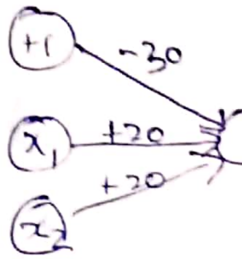
Therefore we don't need to use polynomials of x_1, x_2, x_3 (like before) but rather the NN learns the feature it needs and feeds it to the sigmoid unit in layer 3.

Example and intuition

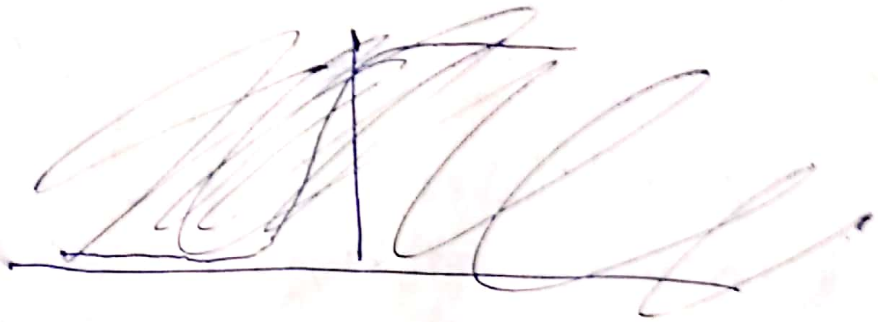
Simple example : AND

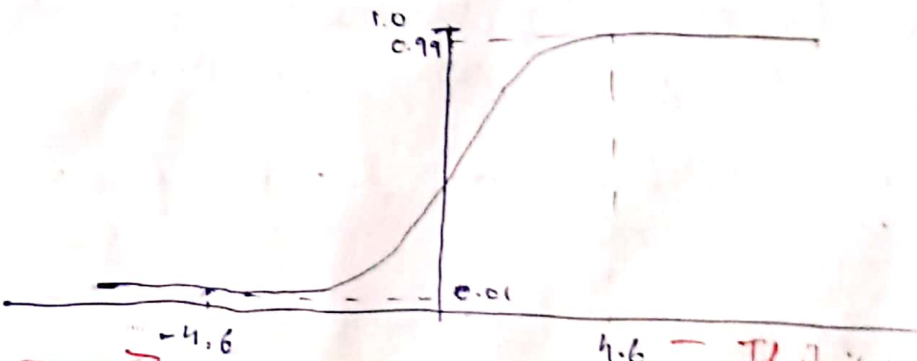
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



$\rightarrow h_\theta(x) = g(-30 + 20x_1 + 20x_2)$





If it is less than this then 0

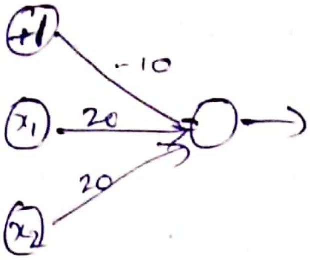
If it is more than this then 1

x_1	x_2	$h_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Simple example: OR

$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ OR } x_2$



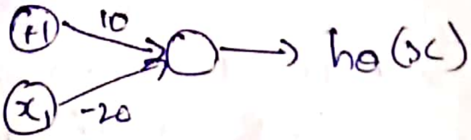
$h_\theta(x) = g(-10 + 20x_1 + 20x_2)$

x_1	x_2	$h_\theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

Simple example: NOT

$x_1, x_2 \in \{0, 1\}$

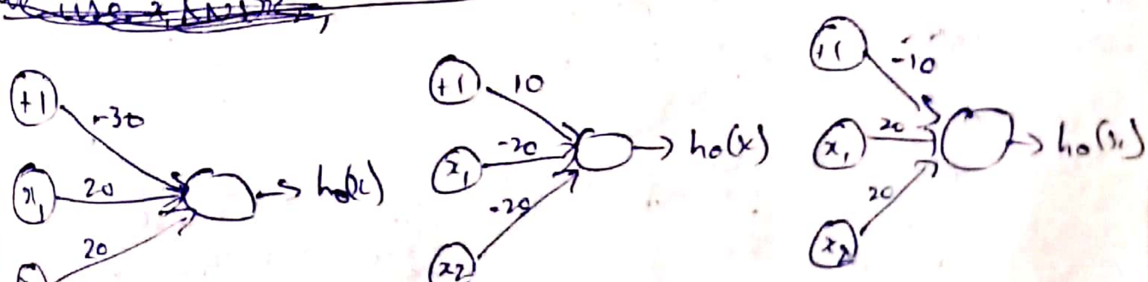
$y = x_1 \text{ NOT } x_2$



x_1	$h_\theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

$h_\theta(x) = g((10) - 20x_1)$

Note: Put large negative value before x_1 for negation

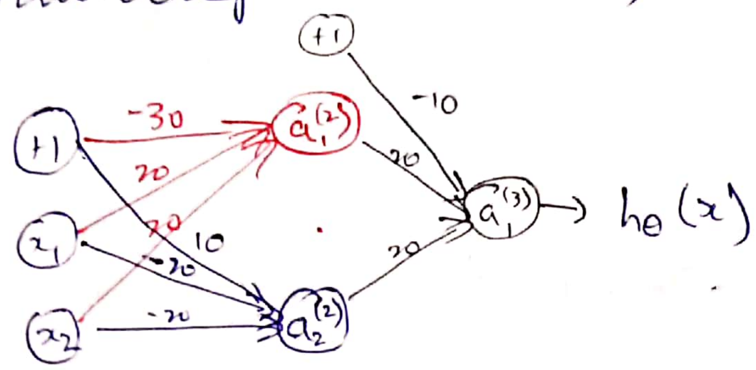


x_1 AND x_2

(NOT x_1) AND (NOT x_2)

x_1 OR x_2

combining the three,



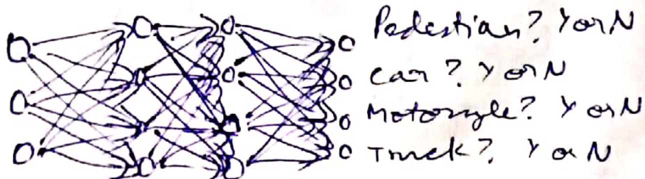
x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

$\therefore$ This is x_1 XNOR x_2

Hence we use simple function to build a more complex function.

Multiclass Classification

It is like one-vs-all method
Take example, our NN will identify pedestrian, car, motorcycle and truck



$h_\theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ when pedestrian $\quad h_\theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ when car $\quad \dots$

The input will be given as,

$$y^{(i)} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \Leftarrow$$ These are the set of resulting classes