⇒ Week - 4 : Neural networks representation

→ Why do we need neural networks ?

* Says you have a complex supervised learning classification problem.

You can use logistic feature if it has 1-2 features BUT suppose if you have 100 features. It WILL NOT work. Here's why :

eg : Suppose n = training examples = 100

X = [x_0 x_1 x_2 ⋮ x_{100}] { ignoring x_0 }

eg : Suppose n = no. of features = 100

then X ≡ Design matrix = [$(x^1)^T$ $(x^2)^T$ ⋮ $(x^m)^T$]

then where $x^i = \begin{bmatrix} x_0^i \\ x_1^i \\ \vdots \\ x_{100}^i \end{bmatrix}$

* Suppp. to prevent underfitting, we increase our features & include all the quadratic terms (second order). in our ~~Hypothesis~~ hypothesis then :

$$\begin{bmatrix} (x_1^2)^i \\ (x_1 x_2)^i \\ (x_1 x_4)^i \\ x_2^i \\ \vdots \\ (x_1 x_{100})^i \end{bmatrix}$$

for $n = 100$,

$$x^i = \begin{bmatrix} ((x_1)^2)^i \\ x_1 x_2 \\ \vdots \\ x_1 x_{100} \\ ((x_2)^2)^i \\ \vdots \\ (x_2 x_{100})^i \\ \vdots \\ ((x_{100})^2)^i \end{bmatrix}$$

Then our hypothesis will be :

$$h = g(\theta^T x)$$

$$= g(\theta_0 + \theta_1 x_1^2 + \theta_2 x_1 x_2 + \dots)$$

→ So as you can see how many features are added in our hypothesis.

* The exact way to calculate how many features for all polynomial terms is the combination function with repetition: $\binom{n+r-1}{r} = {}^{n+r-1}C_r$

$$= \frac{\lfloor n+r-1}{\lfloor r \quad \lfloor n-1}$$

Refer NPTEL vid. for the formula

→ For 100s features (n) 😊, if we wanted to make then quadratic ($r=2$) we would get $\frac{\lfloor 100+2-1}{\lfloor 2 \quad \lfloor 100-1} = 5050$

resulting new features! We can approximate this as

* We

→ Suppose for 100 features, we wanted

→ We can approximate the growth of the no. of new features we get with all quadratic terms with $O(n^2/2)$.

→ As our decision boundary gets more & more complex, the no. of inputs in our hypothesis will increase. which would be

→ Now, supp. we want to include all the cubic terms in our hypothesis, no. of features $= \frac{\lfloor 100+3-1}{\lfloor 3 \quad \lfloor 100-1} = 171700$, the features grow assymptotically at $O(n^3)$.

These are very steep growths,

so as the no. of our features increase, the no. of quadratic or cubic features in our hypothesis increase very rapidly & becomes quickly impractical.

So, as our hypothesis becomes more & more complex,

→ Neural networks offer an alternate way to perform machine learning when we have complex hypothesis with many features.

As we wouldn't need to "add" more inputs in our hypothesis just to satisfy its (hypothesis) complexity.

⇒ Neurons & the brain

* Neural nets are limited imitations of how our brain works! They've had a big recent resurgence because of advances in computer hardware.

*

⇒ There is evidence that the brain uses only one "learning algorithm" for all its different functions.

→ Evidence for this hypothesis:

* Auditory cortex in brain: Responsible for hearing in humans

  • If you cut the wiring from the ear to the auditory cortex & re-route optic nerve to the auditory cortex.

  • The auditory cortex learns to see !!!!!

* Somatosensory cortex: Responsible for touch processing

  • If you rewire optic nerve to the somatosensory cortex, then it learns to see.

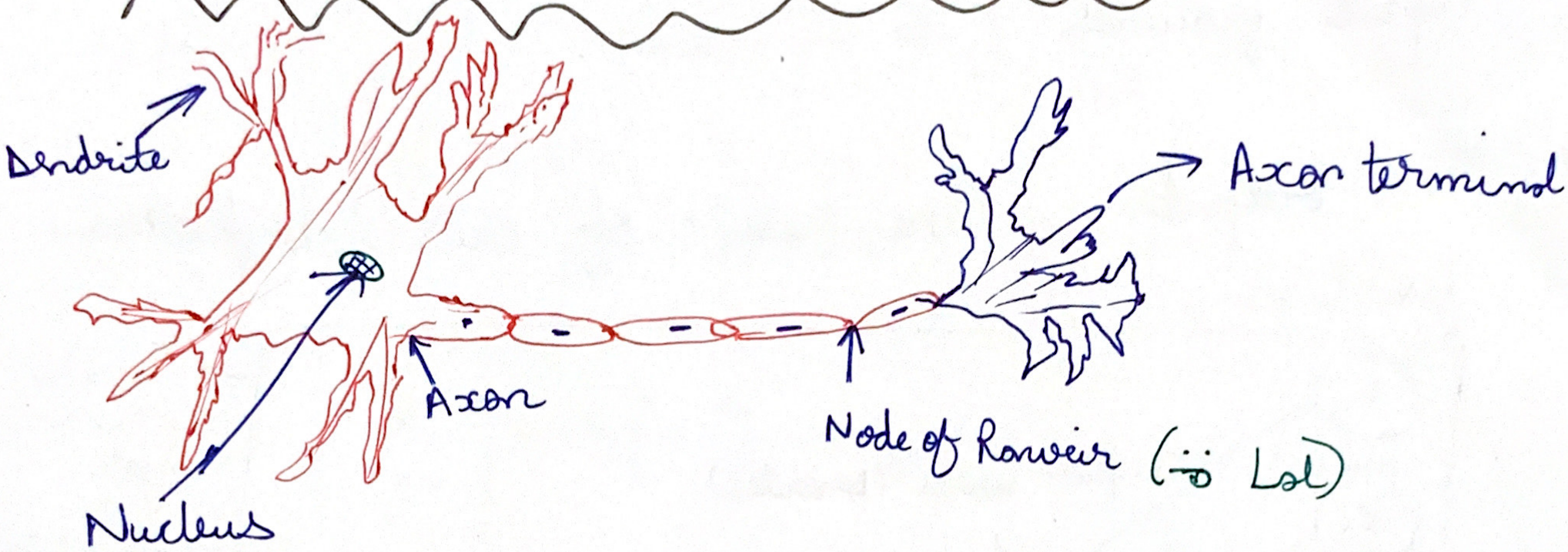
* The principle is called "neuroplasticity" & has many examples & experimental evidence.

→ ~~Eg:~~ More examples:

* Brainport: Seeing with your ~~tounge~~ tongue!
* Human echolocation: Blind people trained to interpret sound & echo (Sonar).

⇒ Model representation

→ What does a neuron look like?



→ Working :

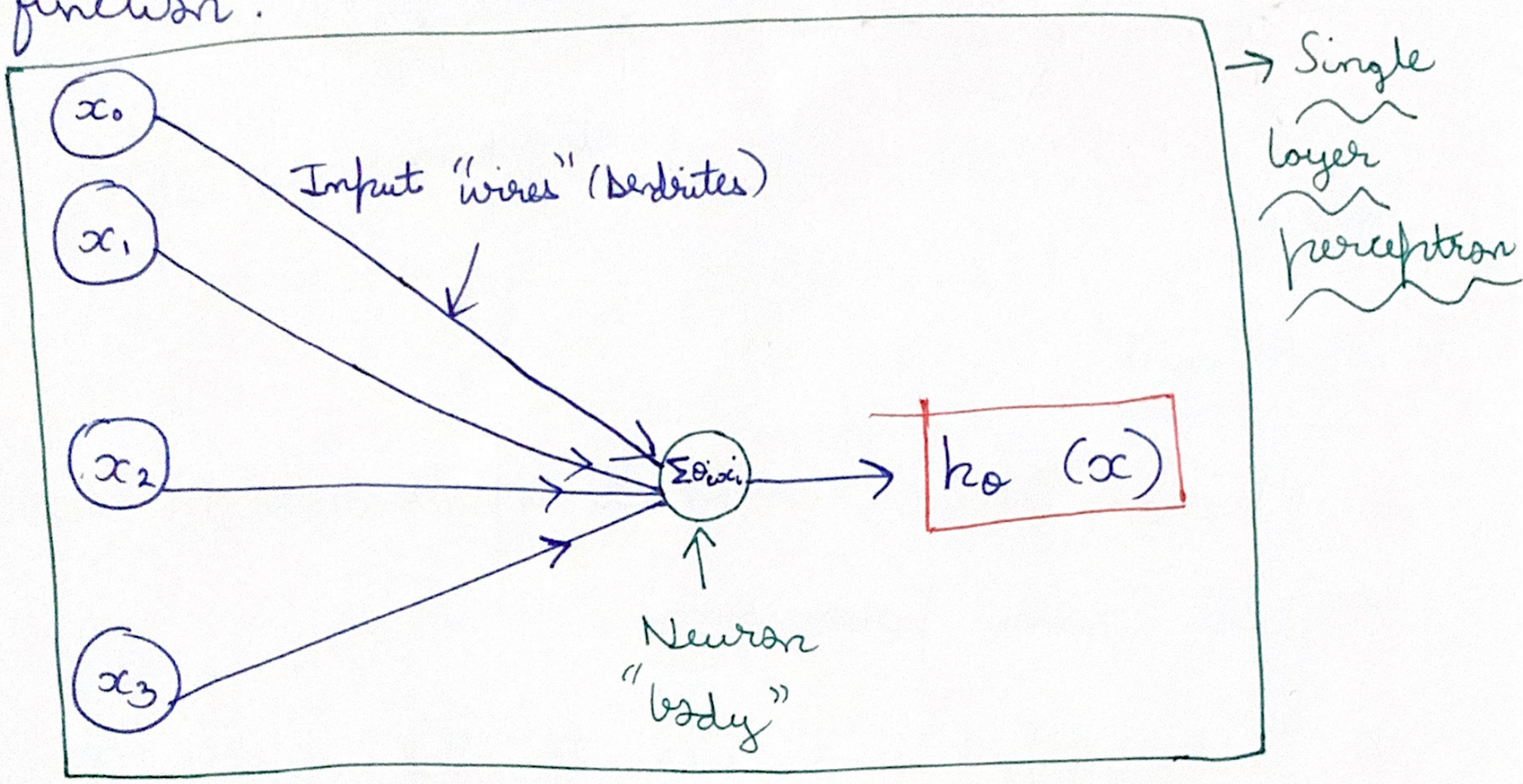
* Neuron gets one or more inputs through dendrites.
* Does some processing
* Sends output down axon to other neuron(s).

→ Neurons communicate through electric spikes.

⇒ Artificial neural network - Representation of a neuron

* In our model, our dendrites are like the input features $(x_1 \dots x_n)$.

* The output is the result of our hypothesis function.



* $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$; In this model our x_0 input node is sometimes called the "bias unit". It is always equal to 1.

* In neural nets, we use the same logistic function as in classification: $\boxed{g(z) = \frac{1}{1+e^{-z}}}$. In neural networks, we sometimes call it a sigmoid (logistic) activation function.

* The parameter $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$ we discussed before are called "weights" in the neural networks model.

* A simplistic representation:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow [\ ] \rightarrow h_\theta(x)$$

⇒ Representing a neural net : Neural net is just a group of ~~networks~~ neurons "strung" together.

* Our inputs nodes (layer 1) go into another node (layer 2), & are output as the hypothesis function.

* Layer-1 : Input Layer

Layer-3 : Output layer

We can have intermediate layers of nodes b/w the input & output layers called the "hidden layer".

→ Notation for hidden layers:

* $a_i^{(j)} \equiv$ activation of unit i in layer j

→ $a_1^{(2)} \equiv$ activation of unit 1 in layer 2

→ Activation: By activation, we mean the value which is computed & output by the node

* $\Theta^{(j)} \equiv$ matrix of weights controlling function mapping from layer (j) to layer $(j+1)$

→ Dimension of $\Theta^{(j)}$:

If network has (s_j) units in layer (j) & $s(s_{j+1})$ units in layer $(j+1)$, then $\Theta^{(j)}$ will be of dimension $(s_{j+1}) \times (s_j + 1)$

The '+1' comes from the addition in $\Theta^{(j)}$ of the "bias nodes" x_0 & $\Theta_0^{(j)}$. In other words, the output nodes will not include the bias nodes

→ Looking at the θ matrix

* Column length → no. of units in the
  fdl. layer

* Row length → no. of units in the current layer + 1
  (because we have to map the bias unit)

→ What are the computations which occur?

* We have to calculate the activation for each node. (the $g(\theta^T x)$ term)

*

* That activation depends on :

  1) the inputs to the node (from the prev. layer)

  2) the parameters associated with the node
  (from the θ vector associated with that layer).

⇒ Note: Every input / activation goes to every node in following layer.

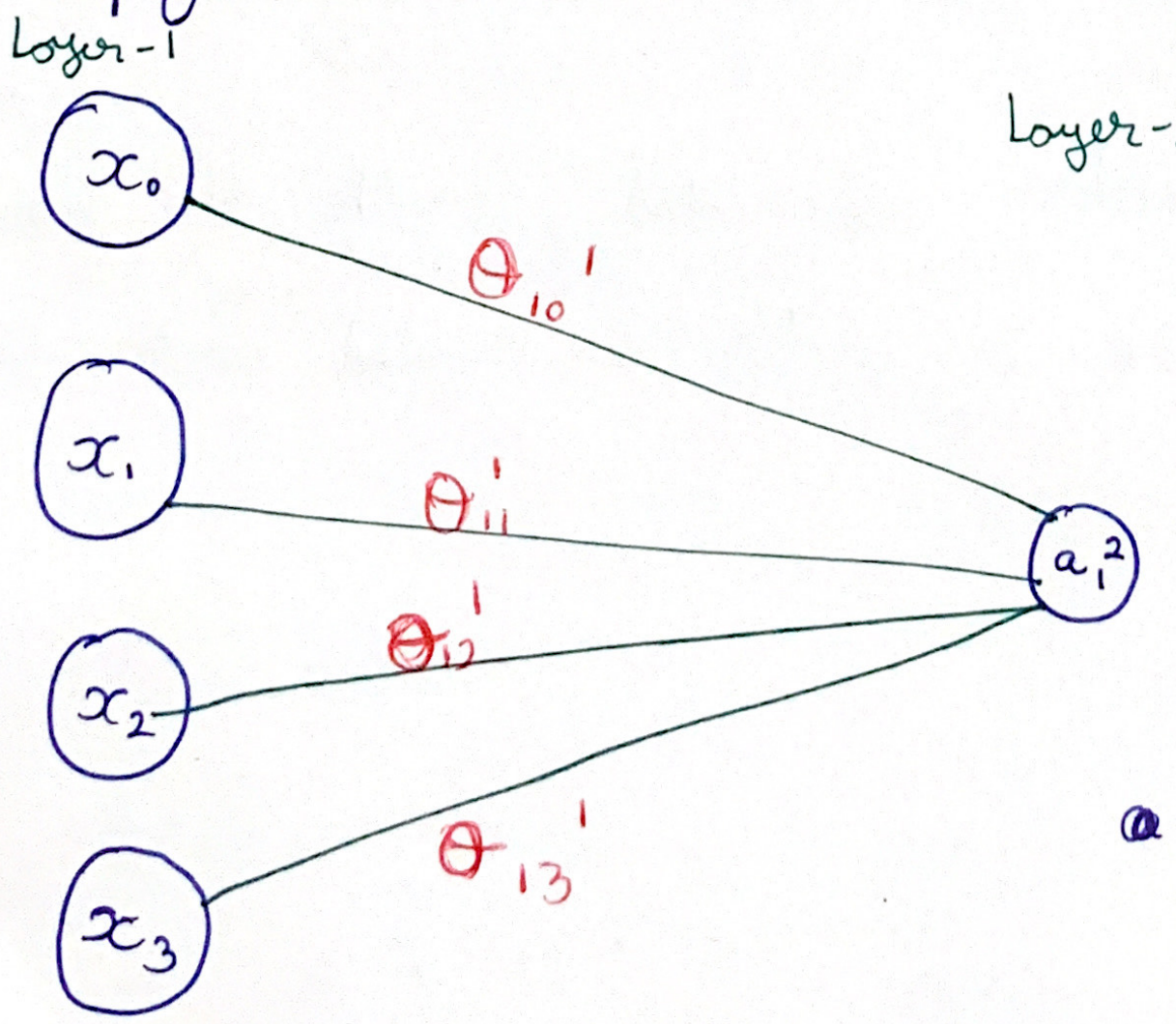
→ θ_{ab}^{c}

* a → Ranges from 1 to the no. of units in layer (c+1)

* b → Ranges from 0 to the no. of units in layer (c)

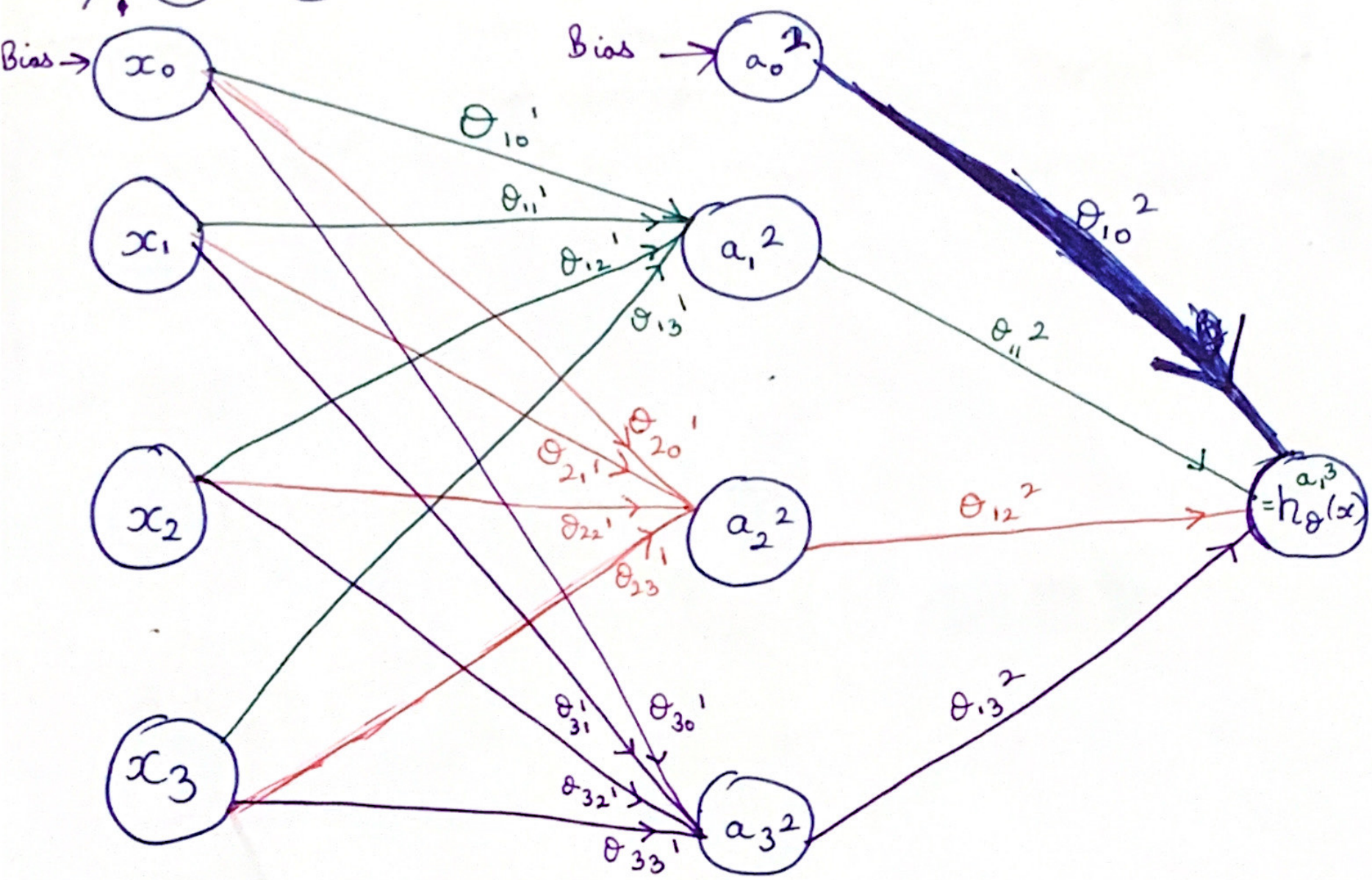
* c → layer which you're moving FROM.

eg: This fll. single layered perceptron will be helpful:



Activation of 1st unit = $a_1^2 = g(\theta_{10}^{1} x_0 + \theta_{11}^{1} x_1 + \theta_{12}^{1} x_2 + \theta_{13}^{1} x_3)$
of Layer 2

⇒ An example of neural net



Layer-1 | Layer-2 (Hidden layer) | Layer-3

* $a_1^2 = g(\theta_{10}^1 x_0 + \theta_{11}^1 x_1 + \theta_{12}^1 x_2 + \theta_{13}^1 x_3)$

* $a_2^2 = g(\theta_{20}^1 x_0 + \theta_{21}^1 x_1 + \theta_{22}^1 x_2 + \theta_{23}^1 x_3)$

* $a_3^2 = g(\theta_{30}^1 x_0 + \theta_{31}^1 x_1 + \theta_{32}^1 x_2 + \theta_{33}^1 x_3)$

* $h_\theta(x) = a_1^3 = g(\theta_{10}^2 a_0^2 + \theta_{11}^2 a_1^2 + \theta_{12}^2 a_2^2 + \theta_{13}^2 a_3^2)$

* $\theta^1 = \begin{bmatrix} \theta_{10}^1 & \theta_{11}^1 & \theta_{12}^1 & \theta_{13}^1 \\ \theta_{20}^1 & \theta_{21}^1 & \theta_{22}^1 & \theta_{23}^1 \\ \theta_{30}^1 & \theta_{31}^1 & \theta_{32}^1 & \theta_{33}^1 \end{bmatrix}_{(3)\times(3+1)}$ → Maps layer-1 to layer-2

(3 input nodes + 1 bias (x_0)) (3 activation nodes)

$*\Theta_2 = \begin{bmatrix} \Theta_{10}^{2} \\ \Theta_{11}^{2} \\ \Theta_{12}^{2} \\ \Theta_{13}^{2} \end{bmatrix}_{1\times(3+1)}$ → Maps layer 2 to layer 3

(3 input nodes) ↓ 1 activation node

+1 bias (a_0^2)

* We can also set $x = a^1$

$$\therefore x = a^1 = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_m \end{bmatrix}$$

$$\therefore a^{j-1} = \begin{bmatrix} a_0^{j-1} \\ a_1^{j-1} \\ \vdots \\ a_m^{j-1} \end{bmatrix} \quad (m+1) \times 1$$

Also, $\theta^{j-1} = \begin{bmatrix} \theta_{10}^{j-1} & \theta_{11}^{j-1} & \theta_{12}^{j-1} & \cdots & \theta_{1 s_{j-1}}^{j-1} \\ \vdots & \vdots & \vdots & & \\ \theta_{s_j 0}^{j-1} & \theta_{s_j 1}^{j-1} & & \cdots & \theta_{s_j s_{j-1}}^{j-1} \end{bmatrix}$

$$\therefore size(\theta^{j-1}) = (s_j) \times (s_{j-1} + 1)$$

$$= (s_j) \times (m+1)$$

We can write $z^j = \theta^{j-1} a$

$$z^j = \theta_{10}^{j-1} a_0^{j-1} + \cdots + \theta_{1 s_{j-1}}^{j-1} a$$

⇒ Vectorized implementation of a neural net

* Define a new variable $z_k^{\,j}$ that encompasses the parameters inside our g function.

If we replace the variable z for all the parameters, we get in our prev. example:

$$a_1^{\,2} = g(z_1^{\,2})$$
$$a_2^{\,2} = g(z_2^{\,2})$$
$$a_3^{\,2} = g(z_3^{\,2})$$

∴ For layer $j = 2$ & node unit $= k$, the variable z will be:

$$z_k^{\,2} = \Theta_{k,0}^{\,1} x_0 + \Theta_{k,1}^{\,1} x_1 + \ldots + \Theta_{k,n}^{\,1} x_n$$

* Vectorizing x & z^j:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix}, \quad z^j = \begin{bmatrix} z_1^{\,j} \\ z_2^{\,j} \\ \cdots \\ z_n^{\,j} \end{bmatrix}$$

Vector of z values from the j^{th} layer

We can rewrite the eqn as :

$$z^{j} = \Theta^{j-1} \; a^{j-1}$$; $z^{j} \rightarrow [s_j \times 1]$ height vector

* Now we can get a vector of our activation nodes for layer j as follows :

$$a^{j} = g(z^{j})$$; here our func^{n} g can be applied element wise to our vector z^{j}

* We can then add a bias unit to layer j after we've computed a^{j}. This will be element a_0^{j} & will be equal to 1.

* Continuing this sequentially, layer by layer & finally we can finally calculate our hypothesis func^{n} as

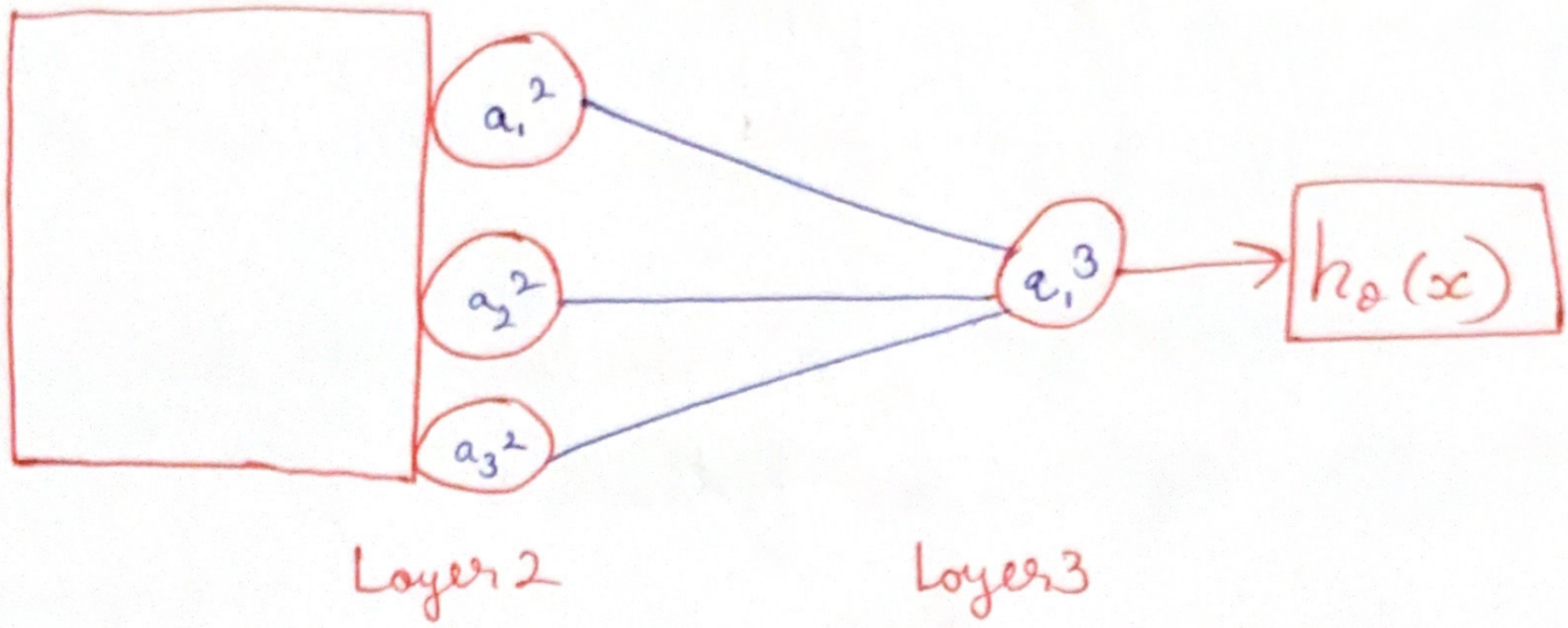
$$h_\Theta(x) = a^{j+1} = g(z^{j+1}) = g(\Theta^{j} a^{j})$$

(Assuming our network has $(j+1)$ layers in total)

(input can + hidden + output)

This process is called forward propagation.

⇒ Neural network learning its own features



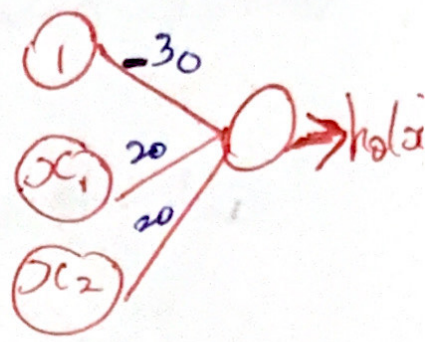
* Notice, that in this last step, between Layer-3 & Layer-2, we are doing exactly the same thing as we did in logestic regression!

The only difference is, instead of inputting a input vector, the features are just values calculated by the hidden layer(s).

* This gives SO, we basically input our values & let the hidden layers learn whatever gives the best final result to feed into the output layer !!!

⇒ Neural network example : AND function

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow [g(z^2)] \rightarrow h_\theta(x)$$



x_0 = bias unit = 1

* Set our first theta matrix as:

$$\theta^1 = [-30 \quad 20 \quad 20]$$

* This will cause the output of our hypothesis to only be +ve if both x_1 & x_2 are 1. In other words:

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

→ Side note : $g(4) \approx 1$; $g(-4.6) \approx 0$

→ Table :

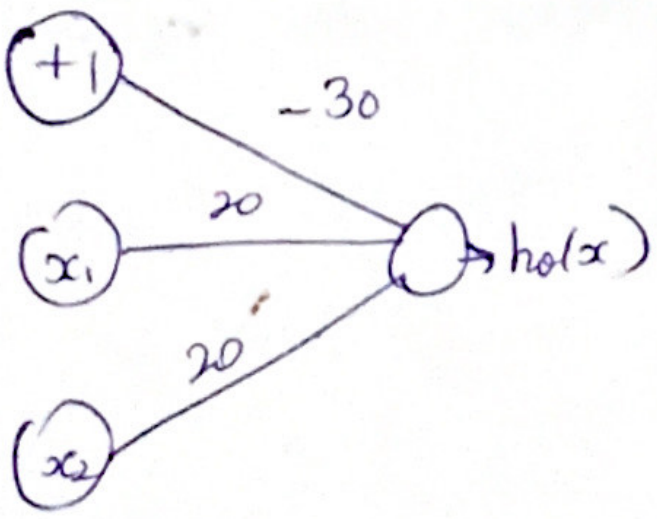
x_1	x_2	$h_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

NOR : $\theta^1 = [10 \quad -20 \quad -20]$

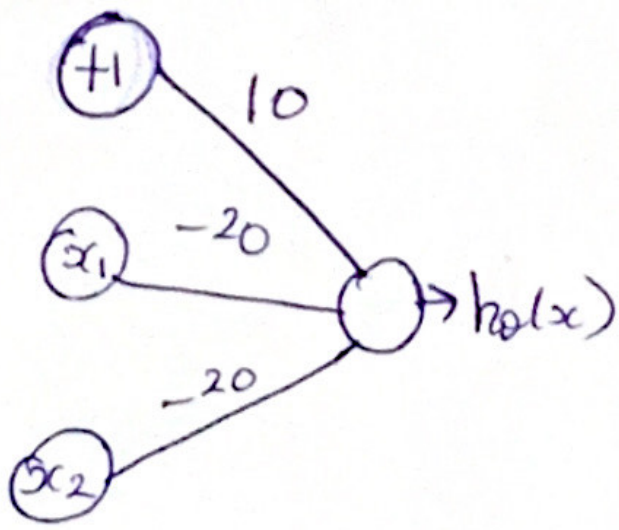
OR : $\theta^1 = [-10 \quad 20 \quad 20]$

---

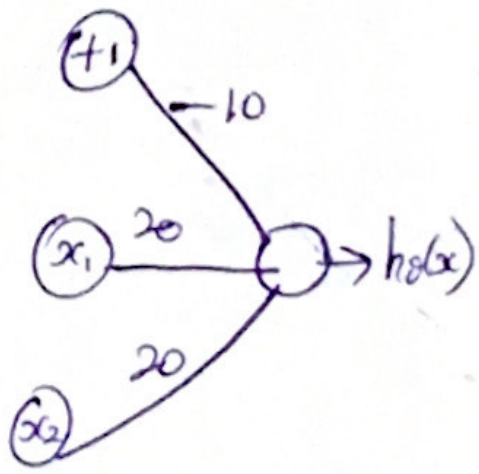
* We combine these 3 into a XOR neural network as shown below:



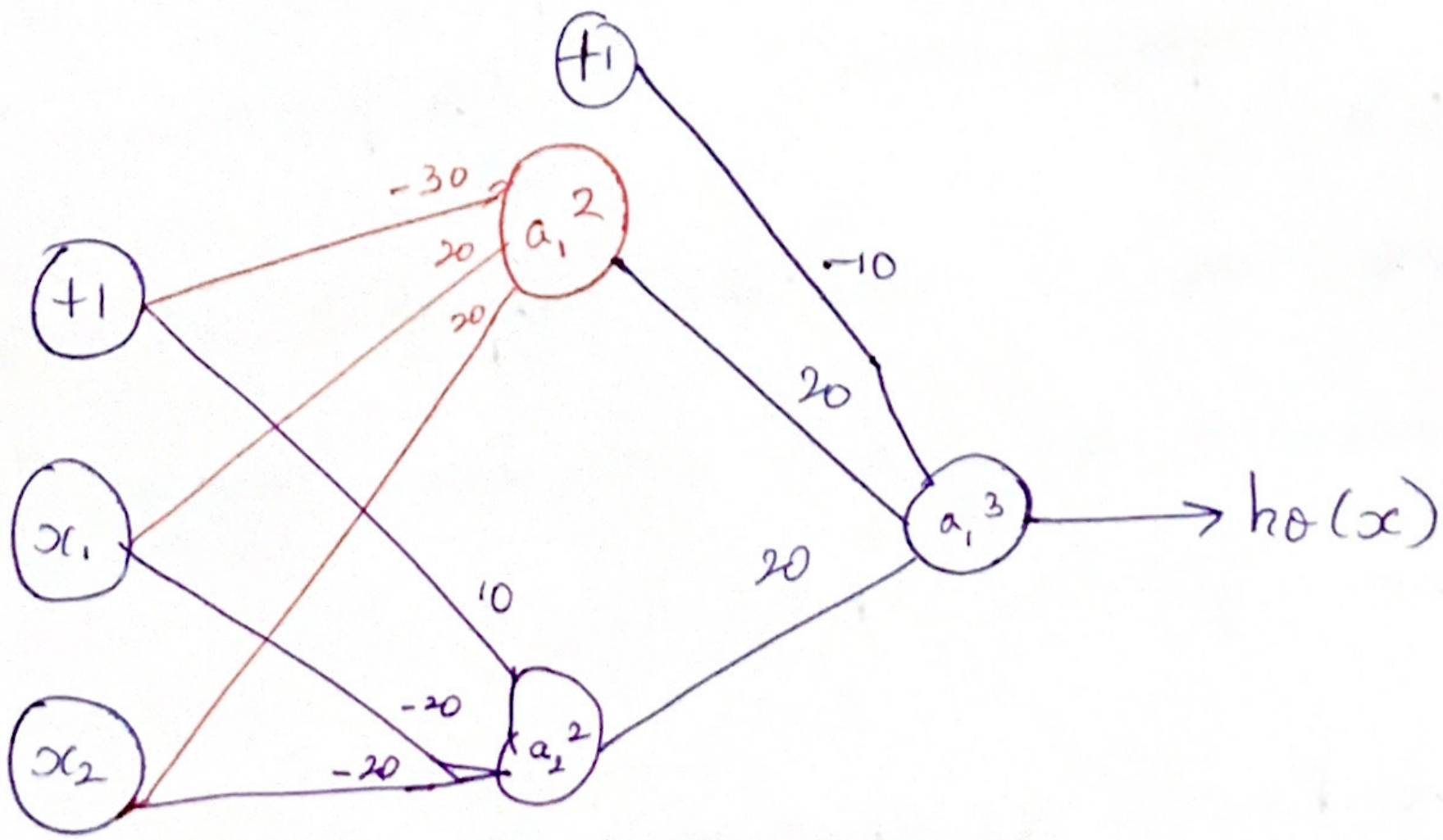
x_1 AND x_2



x_1 XNOR x_2

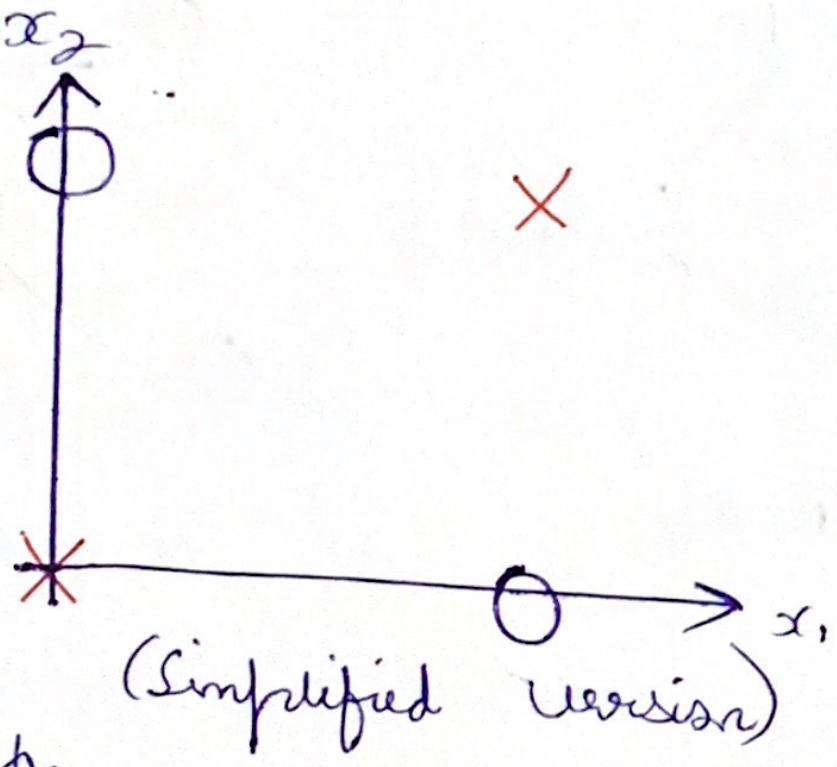


x_1 OR x_2

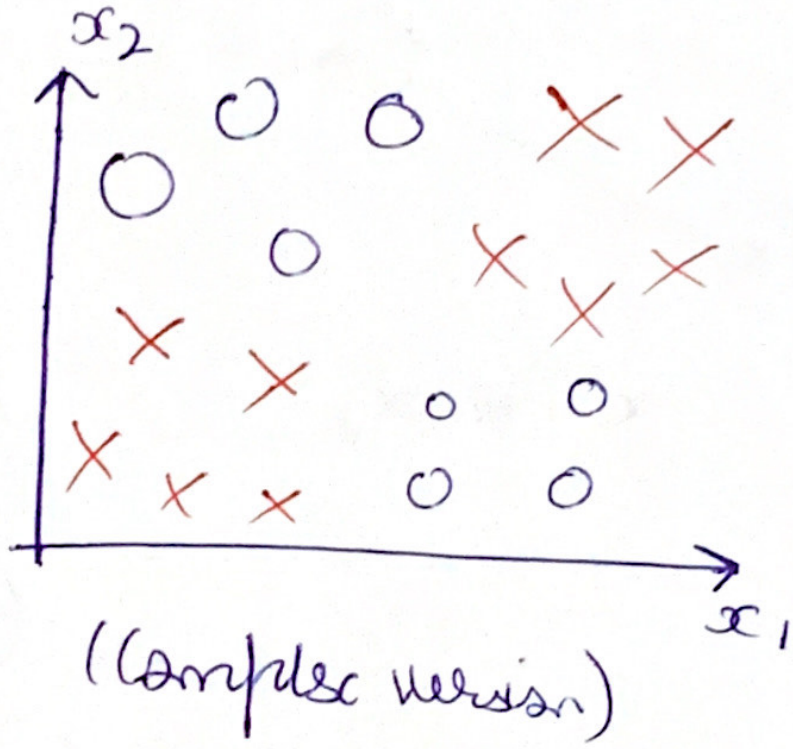


=/ Putting it together : x_1 XNOR x_2

* Supp. we want to classify the foll. dataset :



(Simplified version)



(Complex version)

~~We can see in the simplified version~~

We want to find a non linear decision boundary s.t. $y = x_1$ XNOR x_2

* We know x_1 XNOR $x_2 =$ $\overbrace{x_1 \cdot x_2 + \underbrace{\overline{x_1} \cdot \overline{x_2}}_{\text{NOR}}}^{\text{OR}}$

AND

for

→ ^ the prev. example,

a $\Theta^1 = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}$; $a^1 = x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$\Theta^2 = \begin{bmatrix} -10 & 20 & 20 \end{bmatrix}$

$a^2 = g(\Theta^1 x)$

$a^3 = g(\Theta^2 a^2)$

$h\Theta(x) = a^3$

And there we have the XNOR operator using 2 hidden layers!

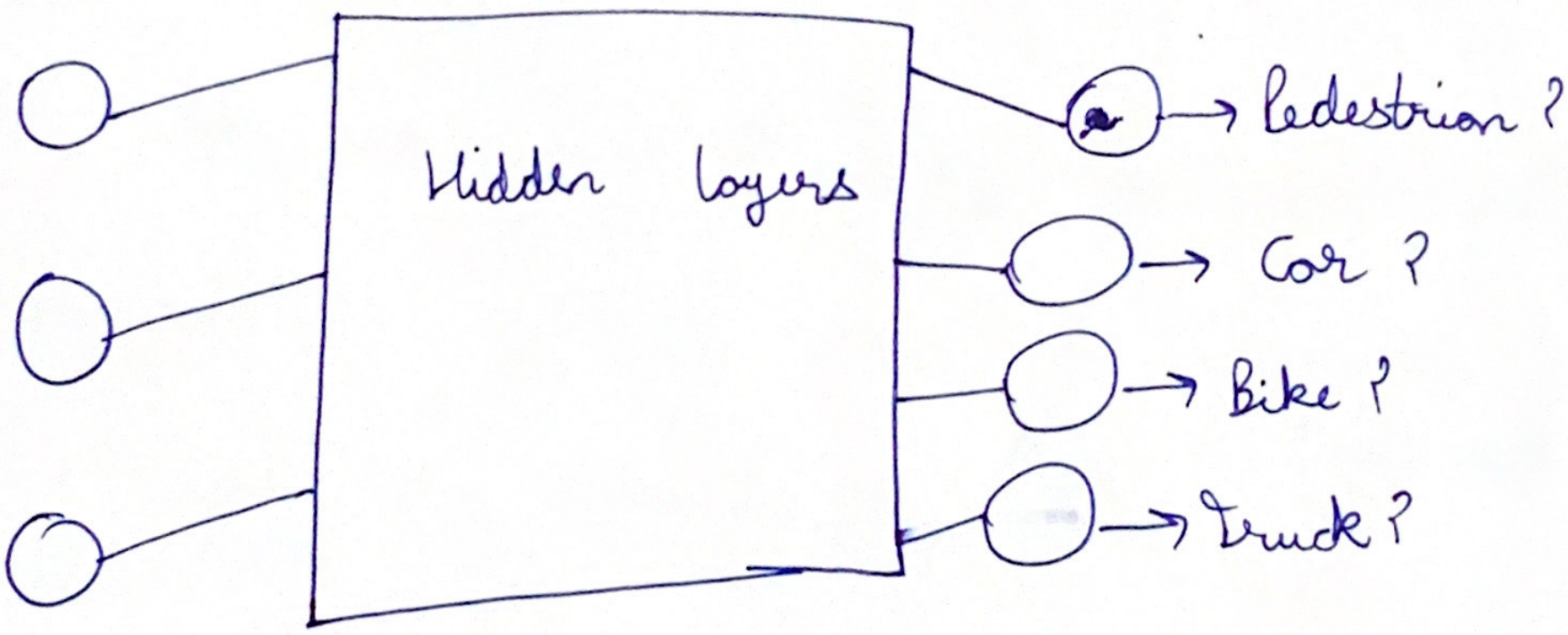
Imagine if we don't even have to "prespecify" the Θs matrices before hand.

That's the power of neural nets! ☺

⇒ Multiclass classification

*To classify data into multiple classes, we let our hypothesis function return a vector of values.

⇒ Multiple output units : one vs all



$$h_\theta(x) \in \mathbb{R}^4$$

Want $h_\theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, when pedestrian; $h_\theta(x) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, when car; $h_\theta(x) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, when bike, etc.

We can define our set of resulting classes as y:

$$y^i = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

* Each y^i represents a diff. image corresponding to either a car, pedestrian, truck, or bike.

The inner layers, each provide us with some new information which leads to our final hypothesis function.

* The setup looks like:

$$\begin{bmatrix}x_0\\x_1\\x_2\\\cdots\\x_n\end{bmatrix} \rightarrow \begin{bmatrix}a_0^{\,2}\\a_1^{\,2}\\\vdots\end{bmatrix} \rightarrow \begin{bmatrix}a_0^{\,3}\\a_1^{\,3}\\\vdots\end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix}h_\theta(x)_1\\h_\theta(x)_2\\h_\theta(x)_3\\h_\theta(x)_4\end{bmatrix}$$