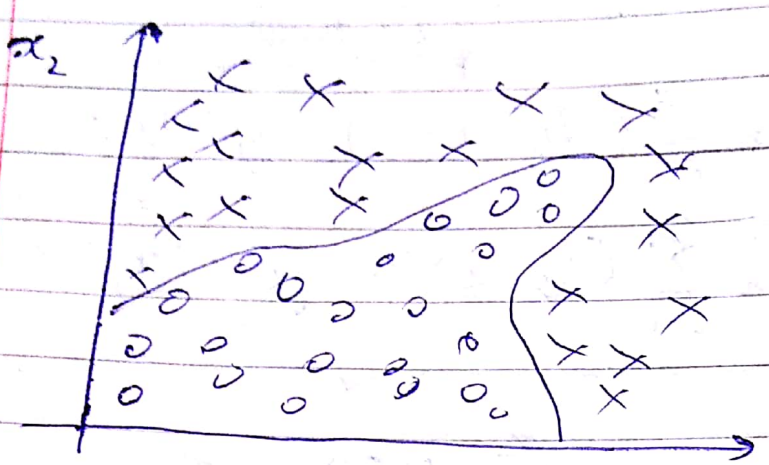
ML

Andrew Ng

Coursera

Week 4

Suppose we need to classify O's:



Many non-linear features will be needed to apply logistic regression:

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2)$

in order to properly fit the decision boundary. The features only increase dramatically in real world situations due to high # of features

Suppose real world feature set of houses:

x_1 = size
x_2 = # of bedrooms
x_3 = # of floors
x_4 = age
...
x_{100}

100 features ($n = 100$).

Counting the second order features from the data set fields.

$x_1 x_2, x_1 x_3 \dots x_1 x_{100}$
$x_2 x_2, x_2 x_3, x_2 x_4 \dots x_2 x_{100}$
$x_3, \dots$

} for $n=100$ ↓ ≈ 5000 features $[O(n^2)]$

↑ asymp. growth rate of quadratic features as the # of features increases

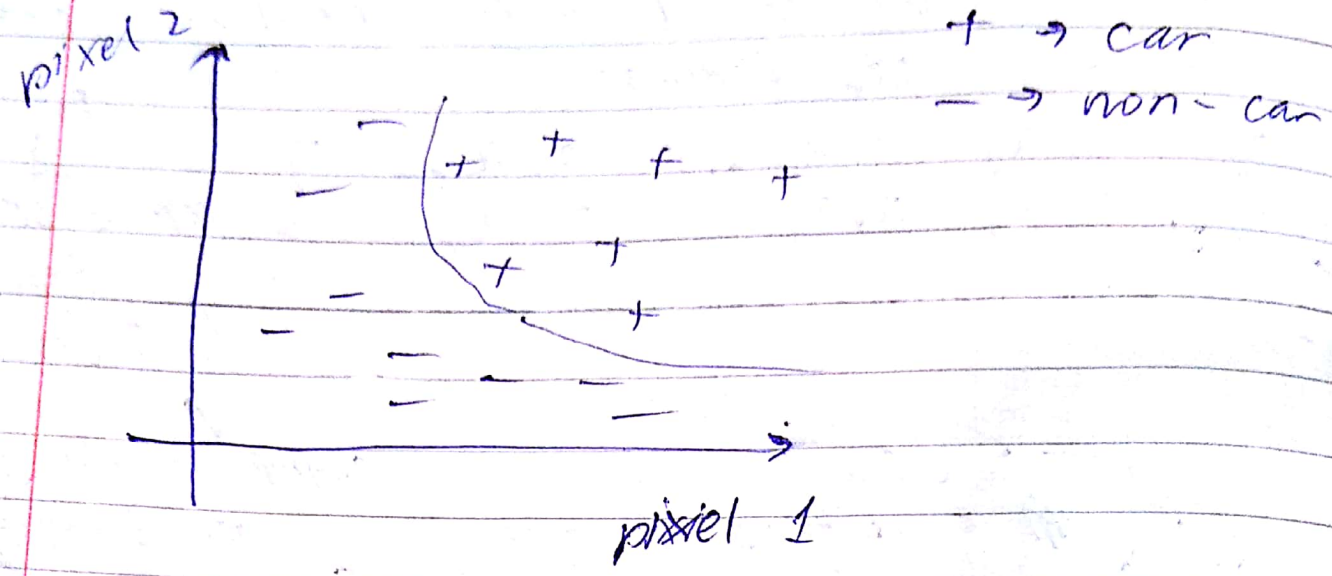
creates overfitting & finding out parameter values begins to become computationally expensive →

Counting the third order features from the data set fields:

$x_1 x_2 x_3, x_1^2 x_2 \dots$ } 170 000 features for $n=100$ $[O(n^3)]$

Too many Feature space is blown up when n increases & data is complex (as the complexity increases, the more higher the order needed becomes)

Suppose computer vision problem where we are trying to understand whether a picture is a car or not



Each pixel constitutes a feature so if 50×50 pixel image then 2500 pixels / features in total
(consider that each pixel is a value of intensity – not RGB)

Considering the Quadratic features of a 2500 pixel image $(x_i \times x_j) \approx$ 3 million features

↳ Thus neural networks are used to learn complex non-linear hypothesis even when n is large

Neural networks

Main goal: have machines that can mimic the working of a brain

Origins: Algorithms that try to mimic the brain

Was very widely used in the 80s & early 90s; popularity diminished in the late 90s

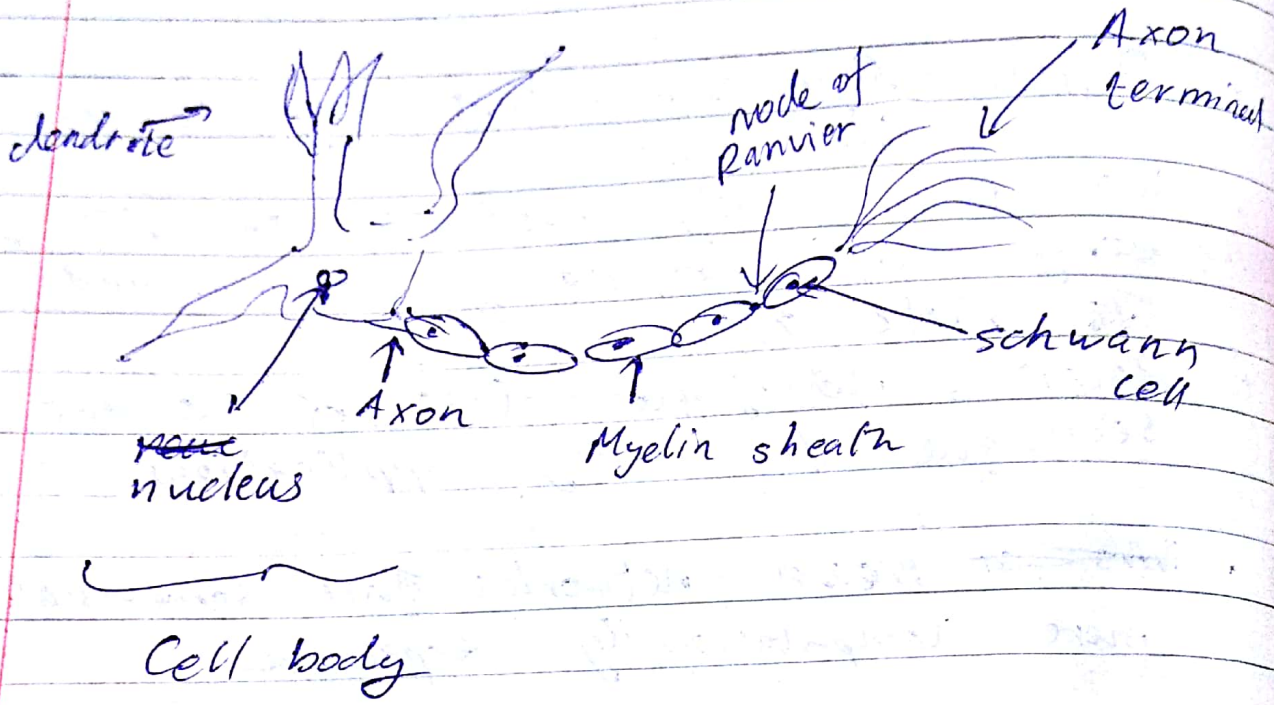
Recent resurgence: state-of-the-art technique for many applications

Neural networks are somewhat more computationally expensive

The "one learning algorithm" hypothesis - brain does all of its tasks (hearing, process sense of touch, consciousness, reasoning) through a single learning algorithm as opposed to many different algorithms running in union

Model representation

Neuron in the brain:



Dendrite - "input wires" - recieve inputs from other locations

Axon - "output wire" - send signals to other neurons

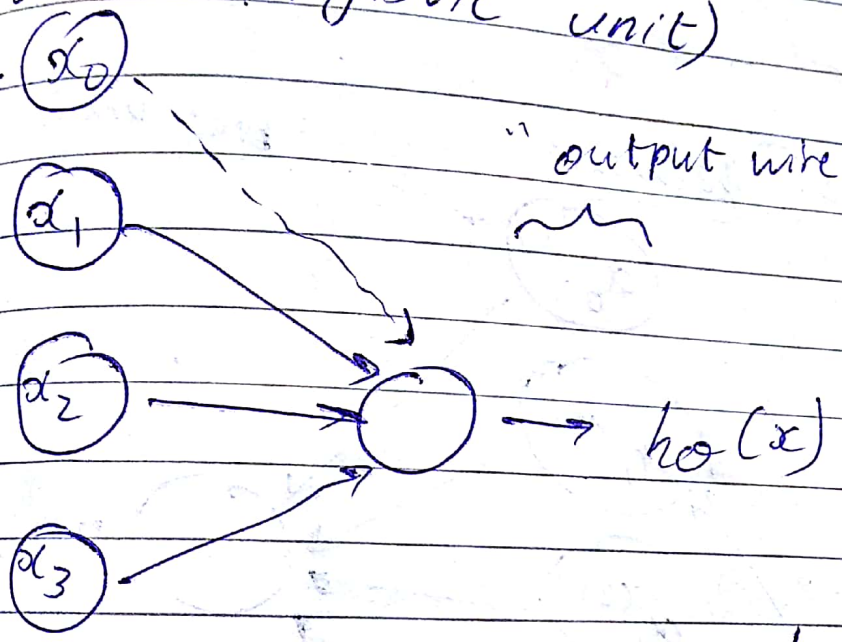
Neuron does the actual computation

- Neuron - a computational unit that gets some inputs & sends output through axon
- Neurons communicate with one another via small pulses of electricity called spikes.

Through these spikes, muscles contract, eyes blinks, all human thought occurs as such

Neron model : logistic unit

↳ a simple model of a neuron (modeled as a logistic unit)



$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

where:

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

dendrites or input wires

body of neuron

"weights" of model

At times this is drawn as well (called the bias unit / bias neuron)

$x_0 = 1$

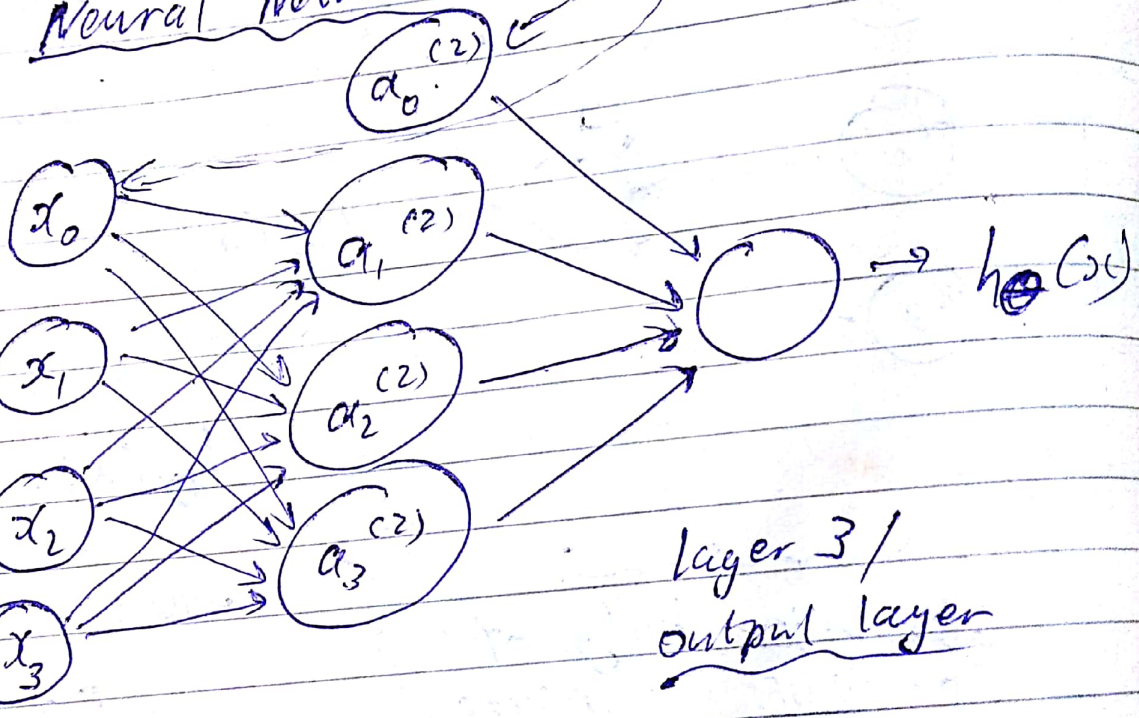
sigmoid (logistic) activation function:

$$g(z) = \frac{1}{1+e^{-z}}$$

↳ used in hypothesis of neural networks

In neural networks the parameters are called "weights" as well

Neural Network



layer 3/ output layer

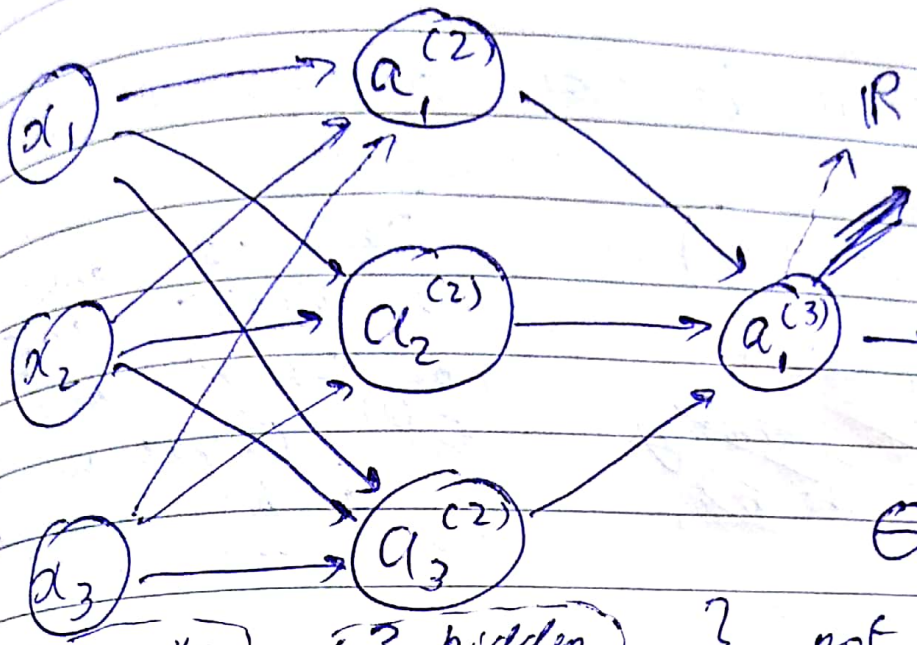
layer 1/ input layer

layer 2/ hidden layer

values while training

Hidden layer called ~~the~~ this way because you can see the input values going into the model & output coming out but you can't see the values in the ~~the~~ hidden layer

NNs can have multiple hidden layers



$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

3 input units | 3 hidden units } not counting the bias

$a_i^{(j)}$ = "activation" of unit i in layer j

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

sigmoid activation function

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

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

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

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

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

$s_{j+1} \times (s_j + 1)$

This +1 comes from the addition in $\Theta^{(j)}$ of the bias nodes x_0 & $\Theta_0^{(j)}$. Output will not include bias nodes while inputs will

not ~~calculating~~ the bias

$\Theta^{(j)}_{30}$ ← indices indicate row & column

$$\Theta^{(1)} = \begin{bmatrix} \Theta_{10} & \Theta_{11} & \Theta_{12} & \Theta_{13} \\ \Theta_{20} & \Theta_{21} & \Theta_{22} & \Theta_{23} \\ \Theta_{30} & \Theta_{31} & \Theta_{32} & \Theta_{33} \end{bmatrix}$$

$$\Theta^{(2)} = \begin{bmatrix} \Theta_{10} & \Theta_{11} & \Theta_{12} & \Theta_{13} \end{bmatrix}$$

Forward propagation:
Vectorised implementation

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

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

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

Hence:

$$a_1^{(2)} = g(z_1^{(2)})$$

$$a_2^{(2)} = g(z_2^{(2)})$$

$$a_3^{(2)} = g(z_3^{(2)})$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

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

$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} = \begin{bmatrix} \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3 \\ \Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3 \\ \Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3 \end{bmatrix}$$

$$= \begin{bmatrix} \Theta_{10} & \Theta_{11} & \Theta_{12} & \Theta_{13} \\ \Theta_{20} & \Theta_{21} & \Theta_{22} & \Theta_{23} \\ \Theta_{30} & \Theta_{31} & \Theta_{32} & \Theta_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \Theta^{(1)} x$$

$$= \Theta^{(1)} a^{(1)}$$

$a^{(2)} = g(z^{(2)})$

$\downarrow$ $\hookrightarrow \mathbb{R}^3$

$a \in \mathbb{R}^3$

The inputs to the neural network can be expressed as x or $a^{(1)}$

~~Each layer has a bias node: $a_0^{(j)}$~~

Each layer has a bias node: $a_0^{(j)} = 1$

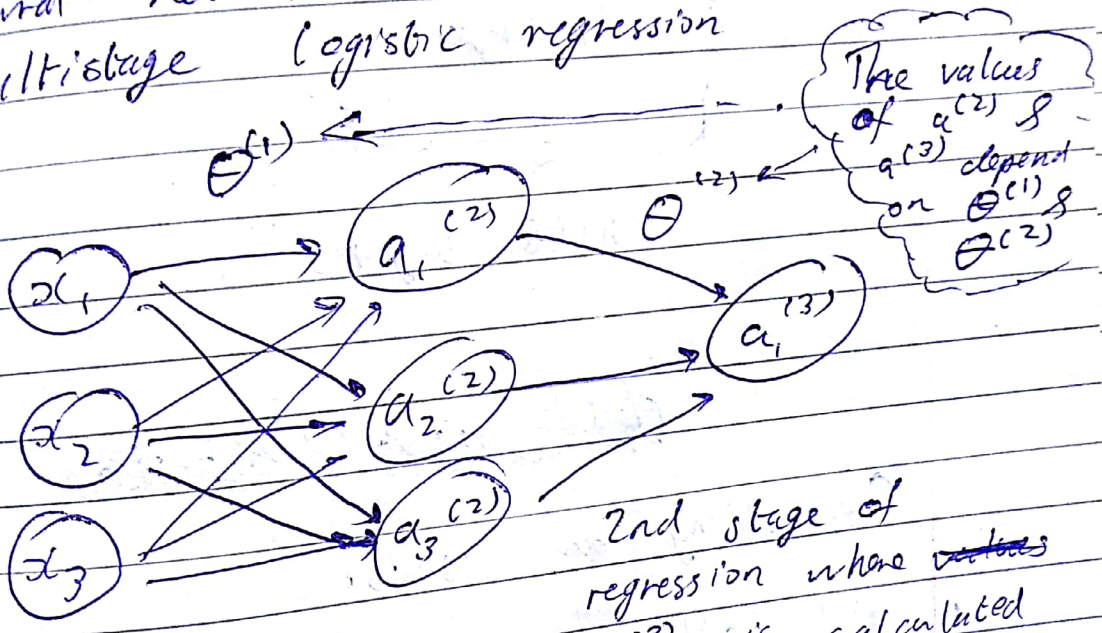
Hence when evaluating $z^{(3)}$ we must add $a_0^{(2)} = 1 \rightarrow a^{(2)} \in \mathbb{R}^4$

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

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

Process of computing $h_\theta(x)$ is called forward propagation because the inputs are propagated forward till the output & activations are evaluated at each stage

Neural networks can be seen as a multistage logistic regression

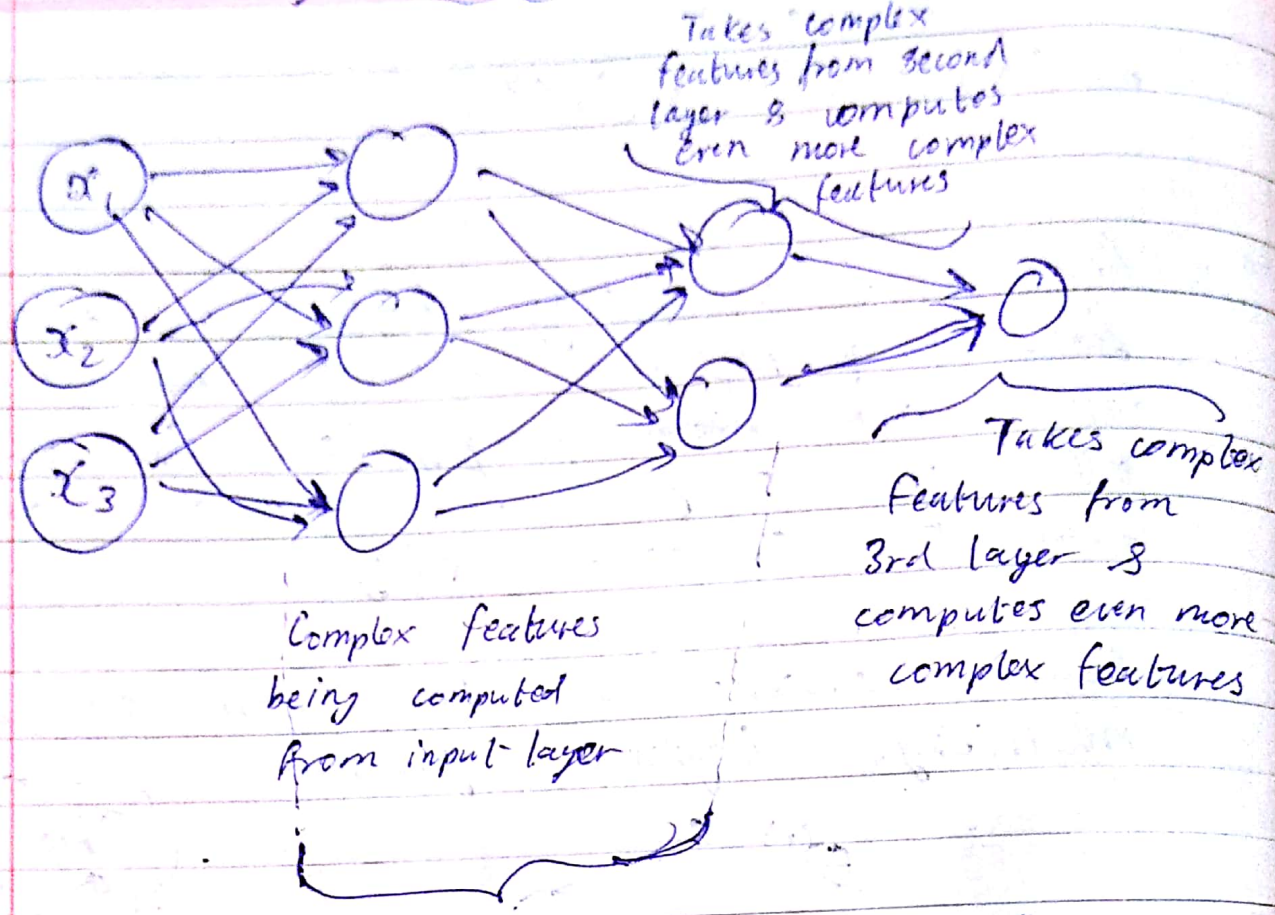


1st stage of regression where values $a^{(2)}$ are calculated

2nd stage of regression where $a^{(3)}$ is calculated from $a^{(2)}$ as inputs

Neural Networks get to learn their own features: $a^{(2)}$ in this example

other network architectures



~~Neural Network Intuition~~

Generalising our results:

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

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad z^{(j)} = \begin{bmatrix} z_1^{(j)} \\ z_2^{(j)} \\ \vdots \\ z_l^{(j)} \end{bmatrix}$$

l : # of nodes in layer j

Given that ~~$z^{(j)} = \Theta^{(j-1)} a^{(j-1)}$~~

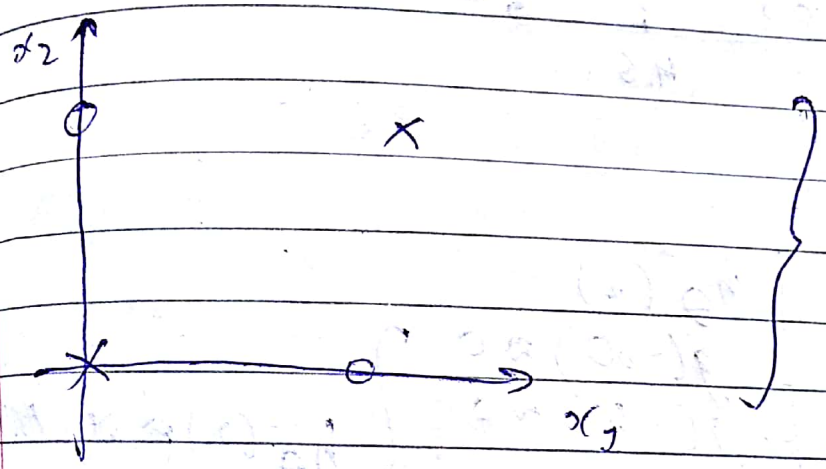
Given that $x = a^{(1)}$ then:

$z^{(j)} = \Theta^{(j-1)} a^{(j-1)}$

Neural Network Intuition

Non-linear classification example: XOR / XNOR

x_1, x_2 are binary (0 or 1)

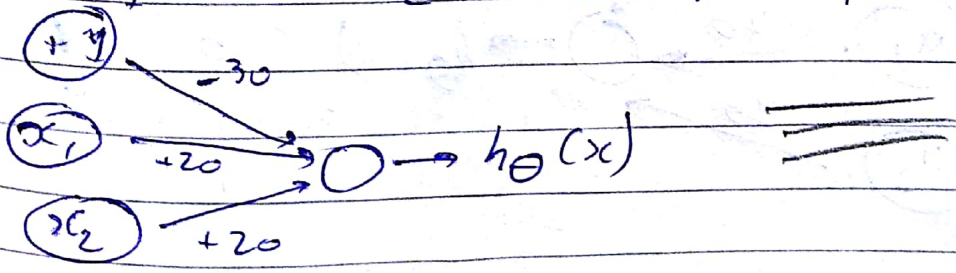


positive values (O):
x_1 XOR x_2
negative values (X):
x_1 XNOR x_2 =
NOT (x_1 XOR x_2)

Simpler example (AND)

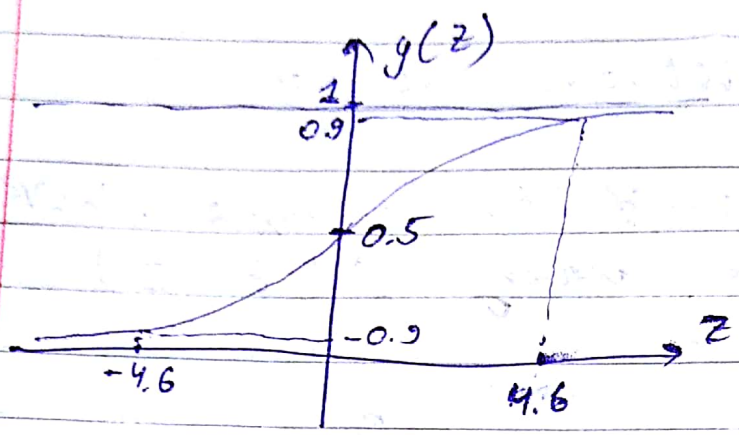
$x_1, x_2 \in \{0, 1\}$

$y = x_1$ AND x_2 ← target function



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

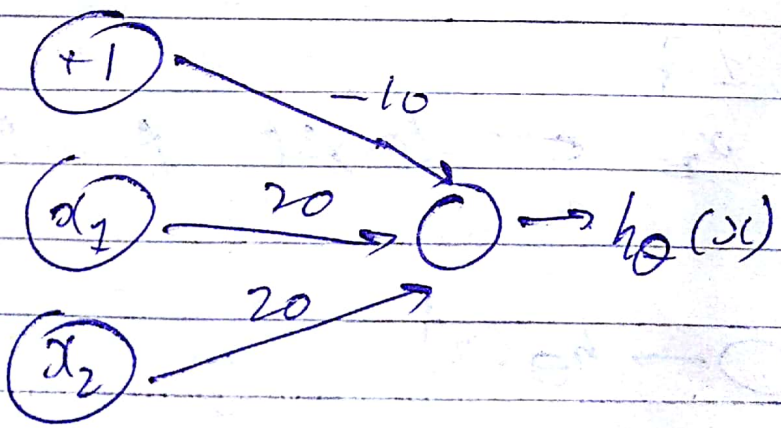
$\theta_{10}^{(1)}$ $\theta_{11}^{(1)}$ $\theta_{12}^{(1)}$



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	$g(+10) \approx 1$

$h_\theta(x) \approx x_1$ AND x_2

Simple example: OR



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 : AND
in mathematical terms

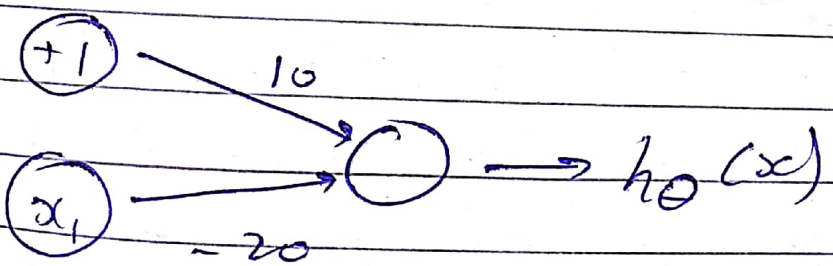
$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow [g(z^{(2)})] \rightarrow h_\theta(x)$$

Graph of function

$$\theta^{(1)} = [-30 \quad 20 \quad 20]$$

Simple example: NOT

Negation: NOT



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

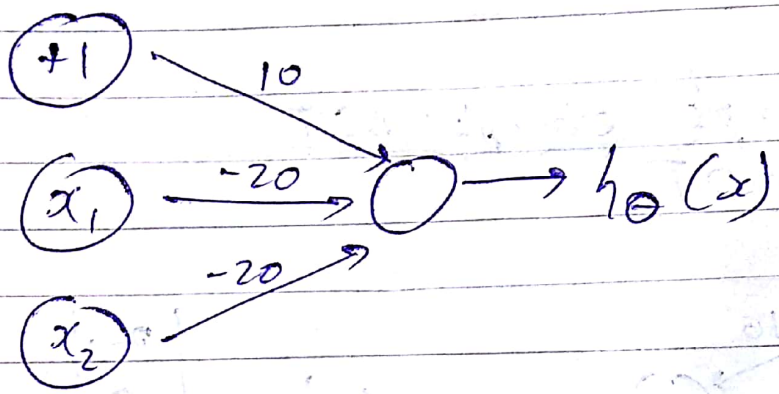
$$h_\theta(x) = g(10 + 20x_1)$$

Simple example: try it yourself

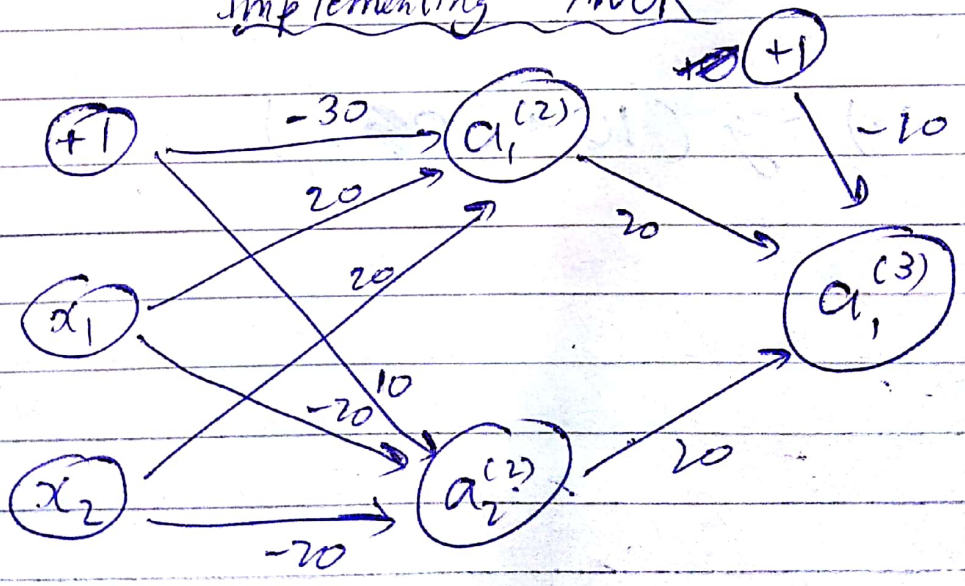
Implement this function: (NOT x_1) AND (NOT x_2)

x_1	x_2	f
0	0	1
0	1	0
1	0	0
1	1	0

$$h_\Theta(x) = 10 - 20x_1 - 20x_2$$



Implementing XNOR



x_1 AND x_2

~~AND form~~

$a_1^{(2)}$ OR $a_2^{(2)}$

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

(NOT x_1) AND (NOT x_2) or x_1 NOR x_2

~~Form~~

In mathematical terms:

AND:

$\Theta^{(1)} = [-30 \quad 20 \quad 20]$

NOR:

$\Theta^{(1)} = [10 \quad -20 \quad -20]$

OR:

$\Theta^{(1)} = [-10 \quad 20 \quad 20]$

XNOR:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \end{bmatrix} \rightarrow [a^{(3)}] \rightarrow h_\Theta(x)$$

where: $\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}$ & $\Theta^{(2)} = [-10, 20, 20]$

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

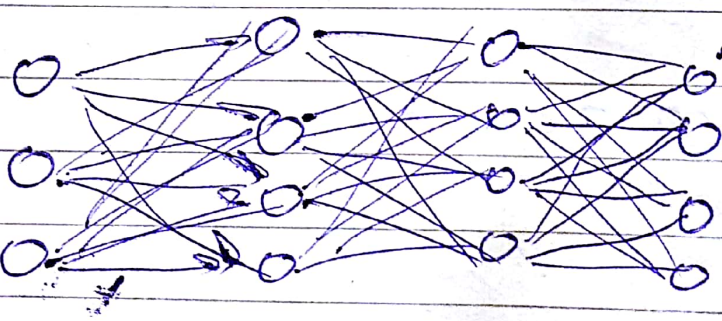
$a^{(3)} = g(\Theta^{(2)} a^{(2)})$

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

Multi-class classification with Neural Networks

The approach used is an extension of one-vs-all method

Example problem: classify pedestrian, car, motorcycle & truck



$h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ when pedestrian

$h_\theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when picture is car

$h_\theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when picture is motocycle

$h_\theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ when picture is truck

Training set : $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots$
$(x^{(m)}, y^{(m)})$

$y^{(i)}$ is one of : $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

pedestrian, car, motorcycle, truck

$h_\theta(x^{(i)}) \approx y^{(i)}$

$\hookrightarrow \mathbb{R}^4$

Mathematically the above problem can be expressed as such:

$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ \dots \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(3)} \\ a_1^{(3)} \\ a_2^{(3)} \\ \dots \end{bmatrix} \rightarrow \begin{bmatrix} h_\theta(x)_1 \\ h_\theta(x)_2 \\ h_\theta(x)_3 \\ h_\theta(x)_4 \end{bmatrix}$