CS395: Selected CS1 (Introduction to Machine Learning)

Associate Prof. Wessam El-Behaidy

Fall 2021

References:

https://www.coursera.org/learn/machine-learning (Andrew Ng)

Machine learning A to Z: **Kirill Eremenko ©superdatascience**

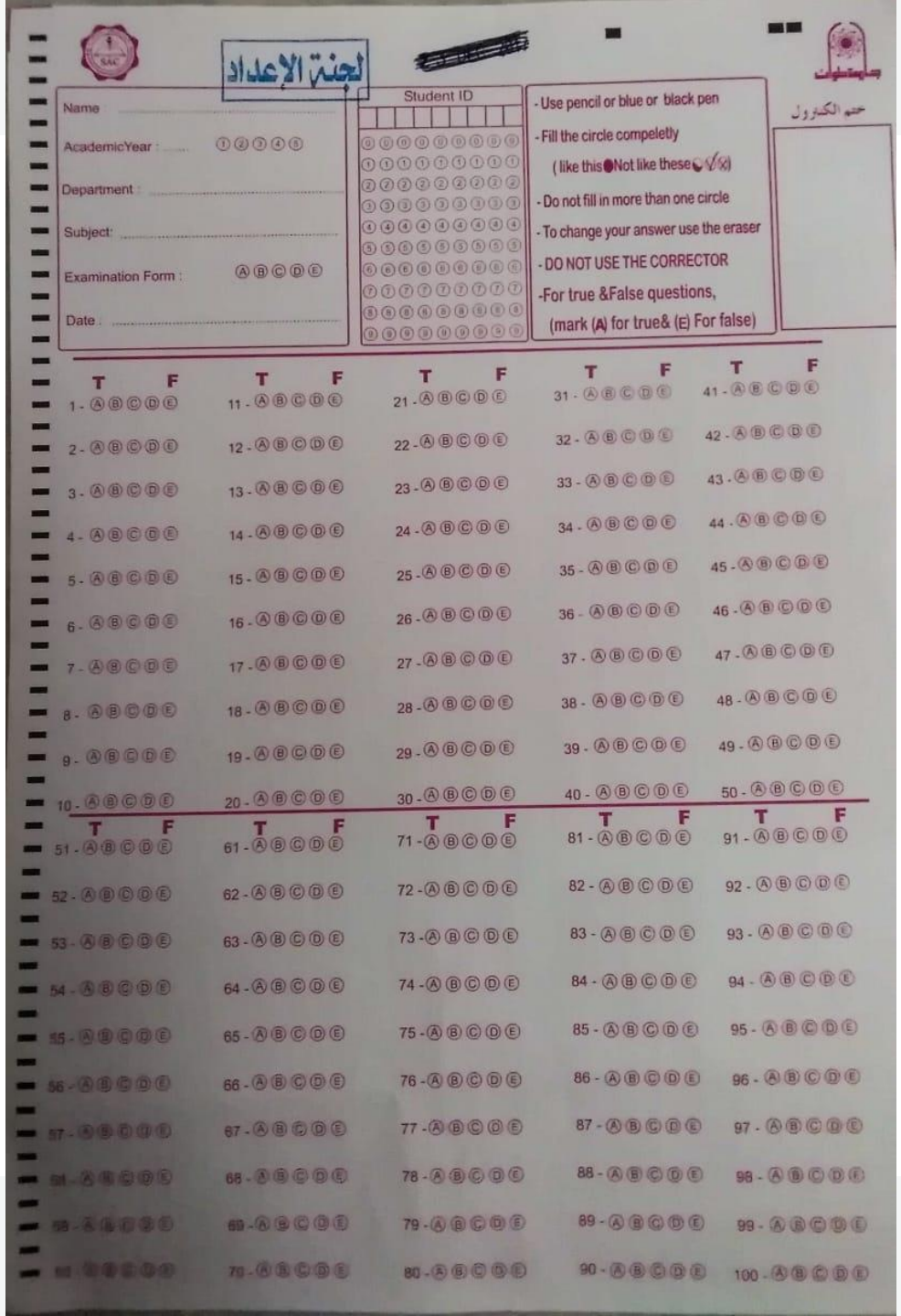
Midterm Exam

- Midterm Exam (based on announced schedule)

Sun. 21 Nov. from 9:00 AM to 9:30 AM

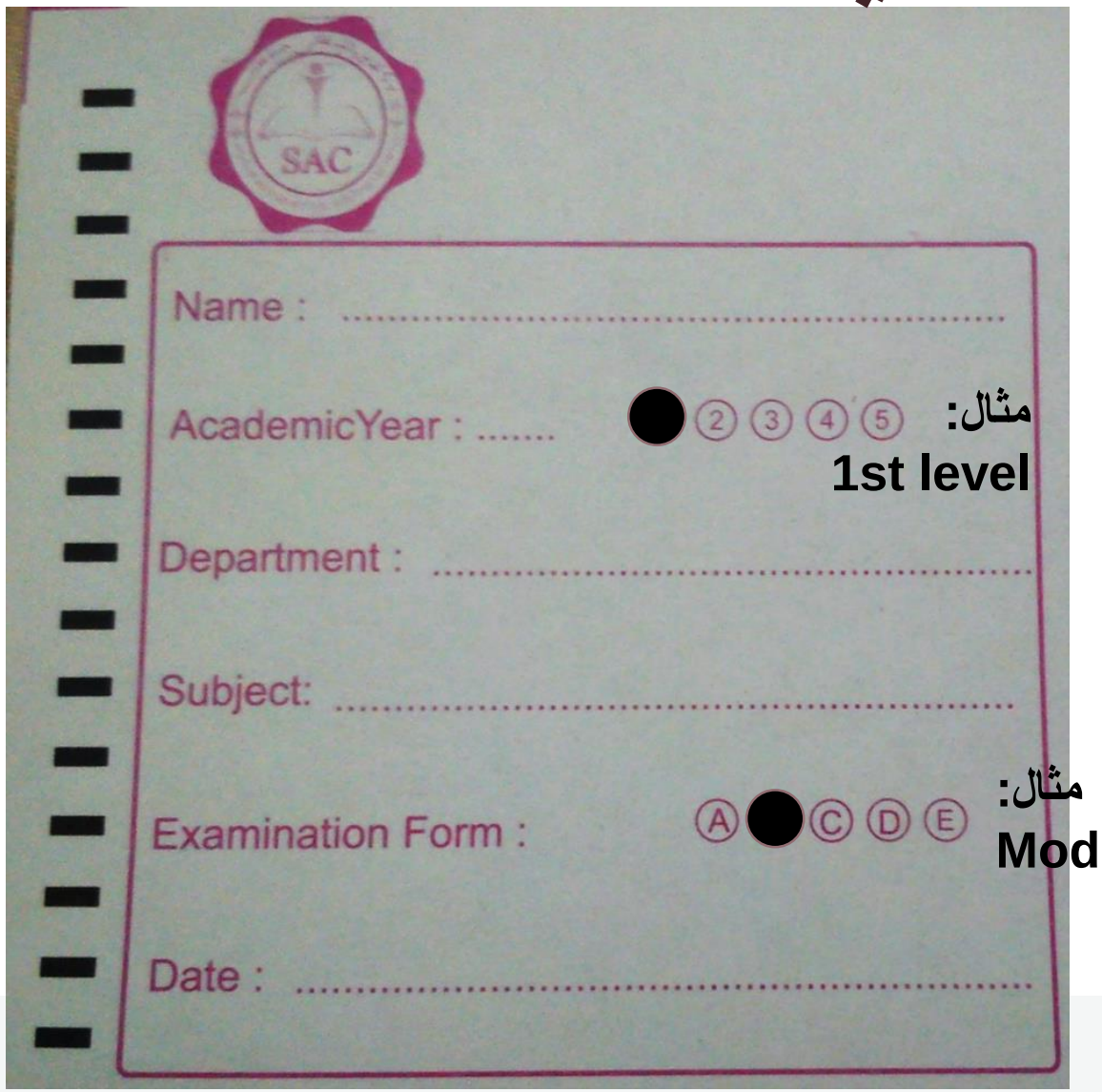
تنبيهات هامة

- الحضور للكلية قبل موعد الامتحان بنصف ساعة على الأقل
- يتواجد معك (قلم جاف- قلم رصاص – استيكة) **هام جدا جدا**
- الاجابة فى ورقة الاجابات الالكترونية بالقلم الرصاص وتحبر فى النهاية
- كتابة البيانات على ورقة الاسئلة وورقة الاجابة
- تسليم كلا من ورقة الاسئلة وورقة الاجابة (**عدم تسليم الورقتين يلغى امتحانك**)
- يجب احضار الة حاسبة للامتحان



الشكل العام لورقة التصحيح الألكترونى

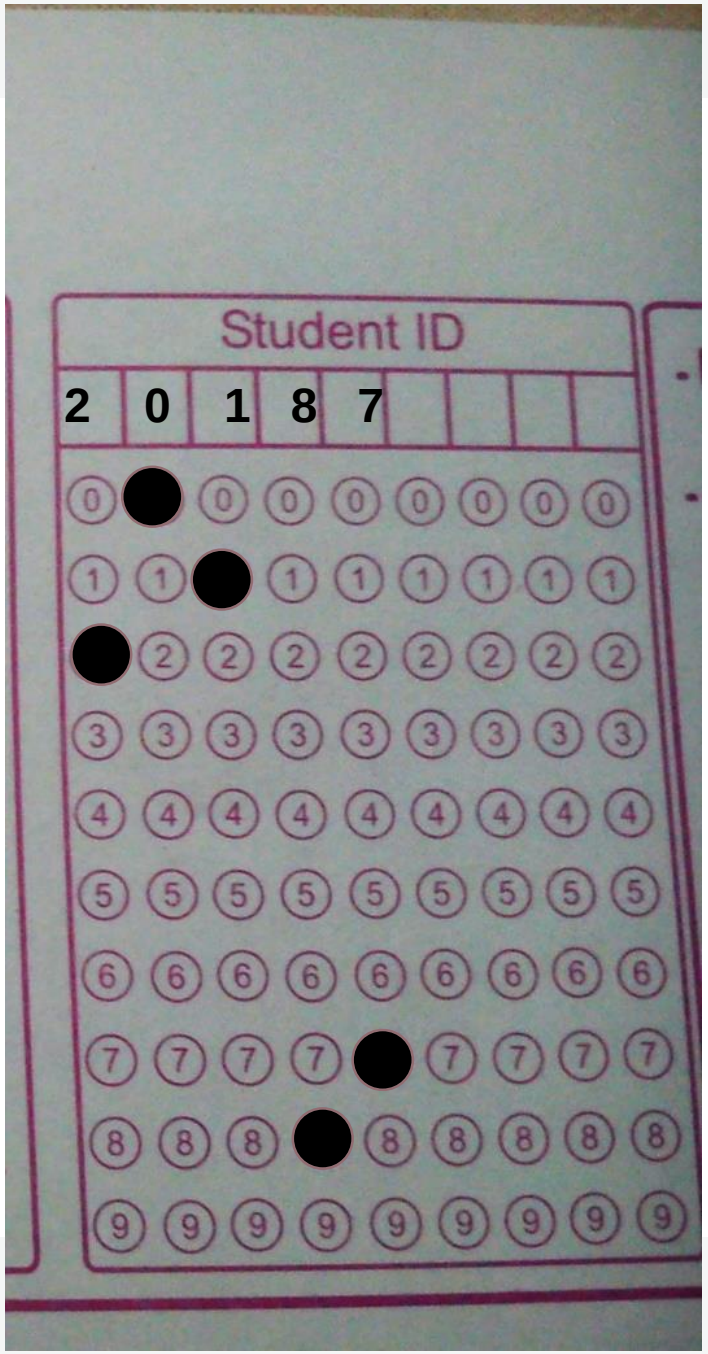
البيانات الأساسية



مثال:
1st level

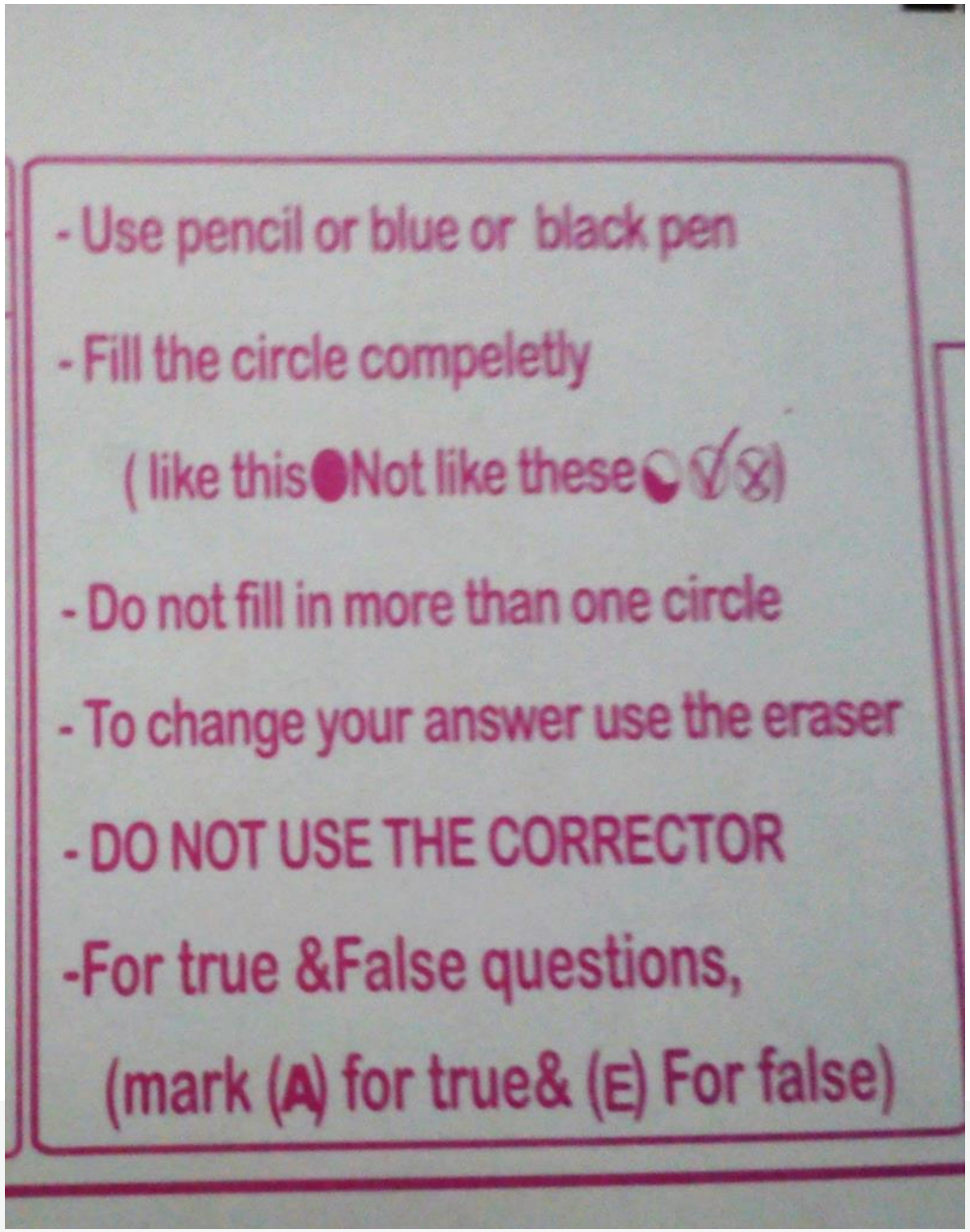
مثال:
Model B

رقم الطالب



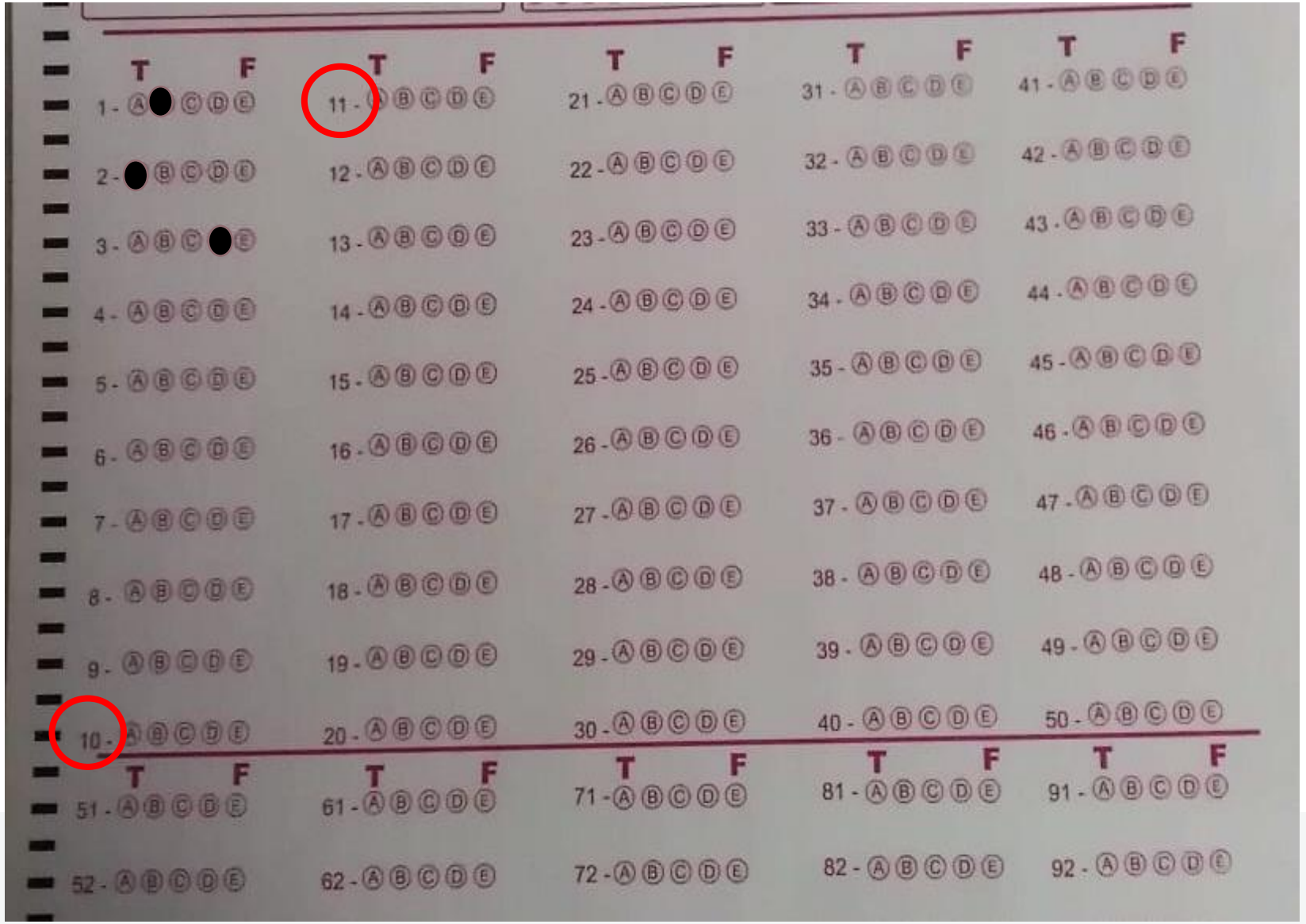
مثال:
رقم الجلوس : 20187

تعليمات هامة

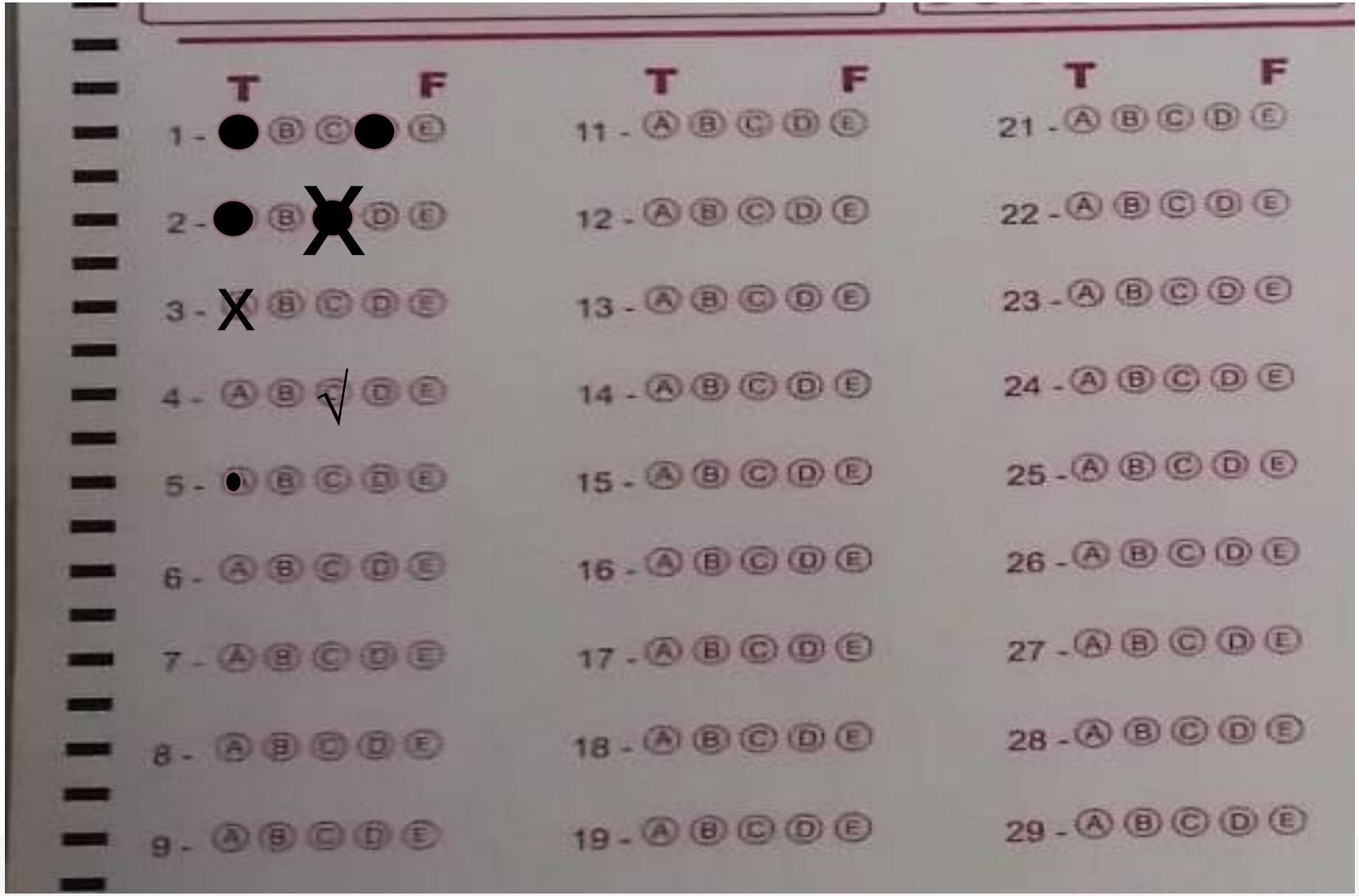


مثال لإجابات مظللة بشكل صحيح

مثال:
اجابة السؤال الأول
B



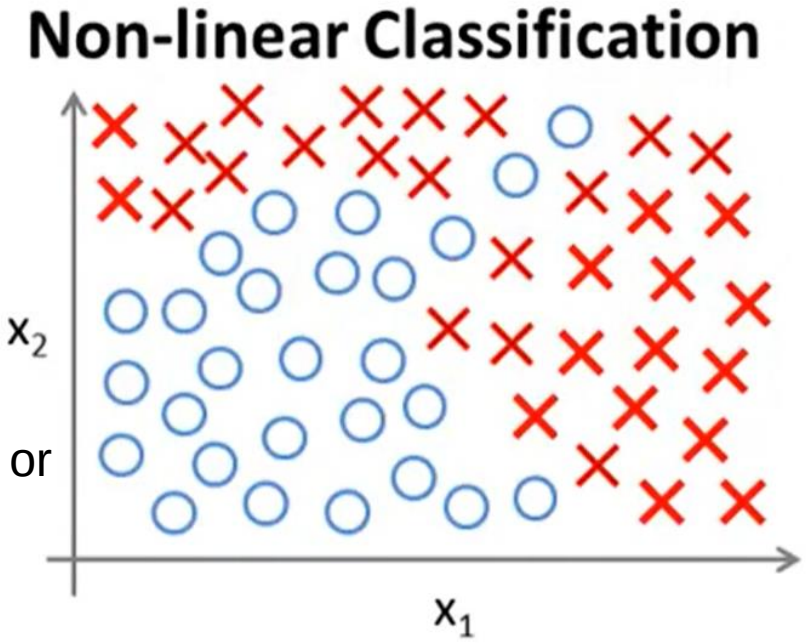
مثال لإجابات مظللة بشكل خطأ



Problems of Logistic Regression with non-linear classification

Non-linear Classification

$$h_\theta(x) = g\,(\theta^T x) = g\begin{pmatrix}\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 \\ +\theta_5 x_2^3 \dots\dots\dots\end{pmatrix}$$



If n=100,
how many **parameters** if we only use second polynomial terms or Quadratic terms??

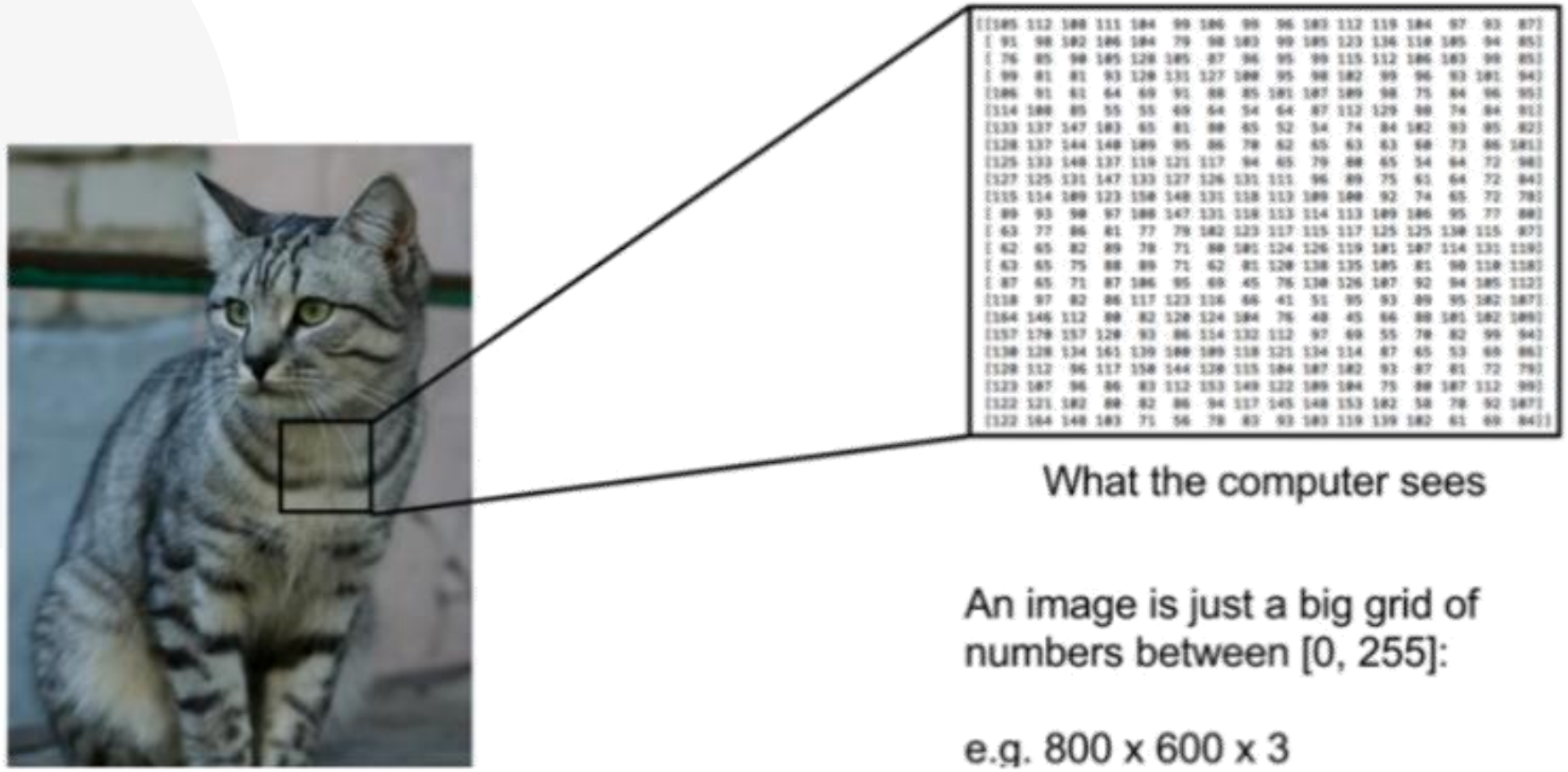
What if we need to have more higher polynomial terms?
how many parameters?

Is this a good way to build a non-linear classifier, **when n is large**?

Image Classification Problem

How many features do we need?

Semantic Gap, What the computer sees?

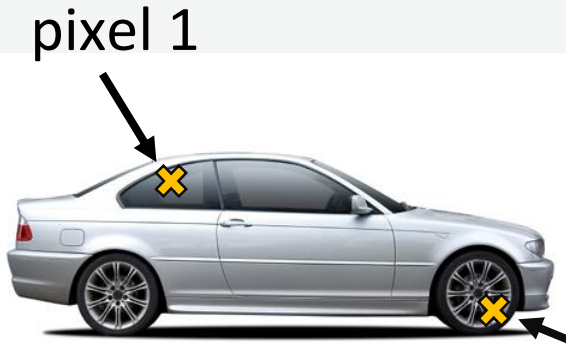


What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3

Is it easy for the machine?



Learning Algorithm

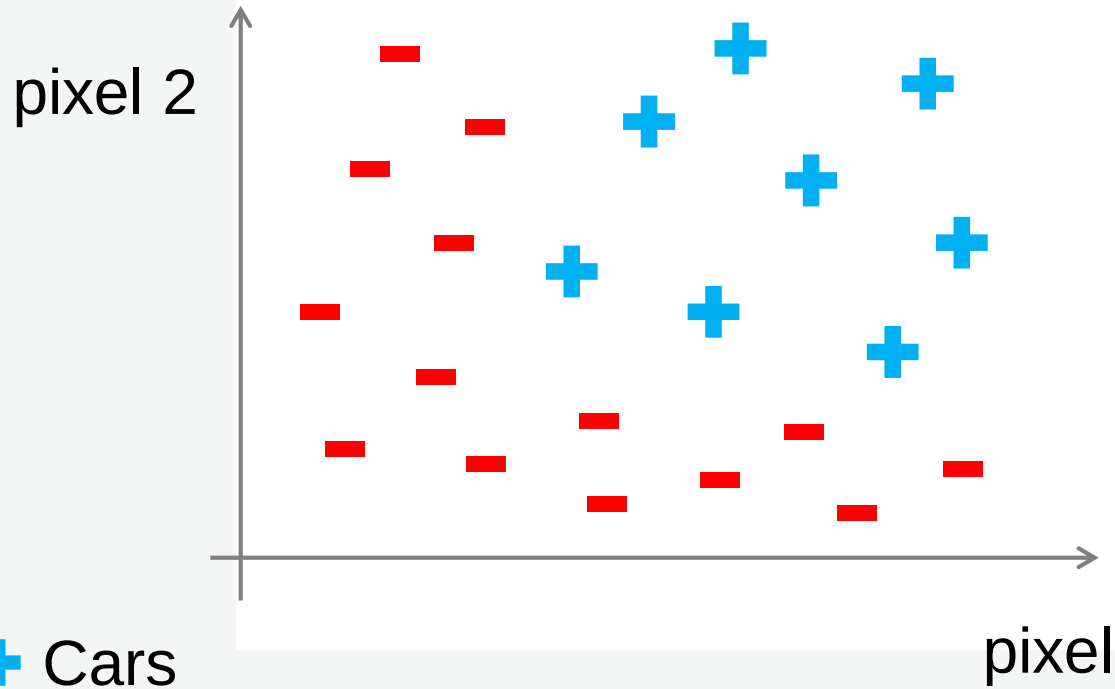
50 x 50 pixel images→ 2500 pixels

$$n = 2500$$

(7500 if RGB)

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

Quadratic features ($x_i \times x_j$): ≈3 million features

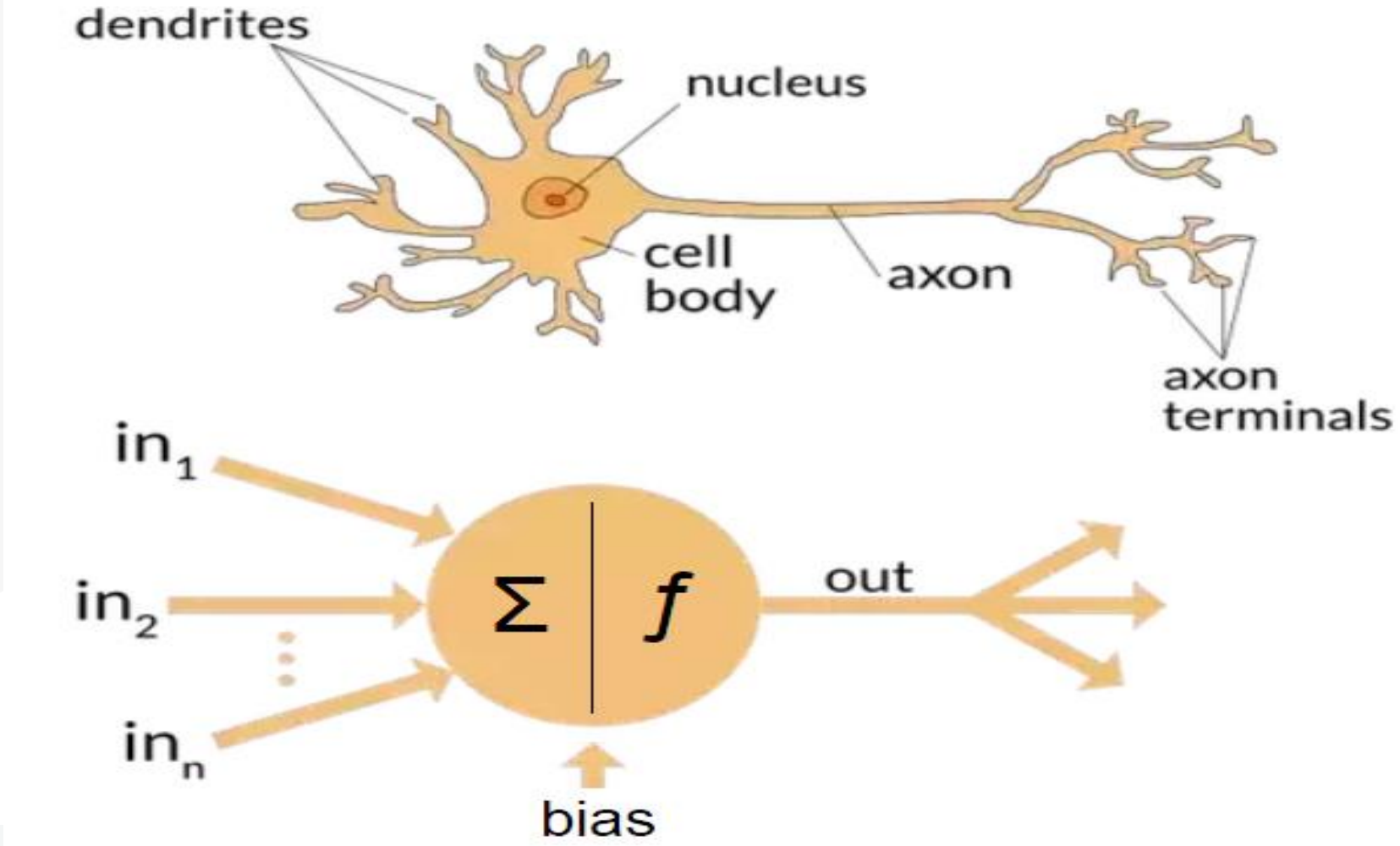


Artificial Neural Network (ANN)

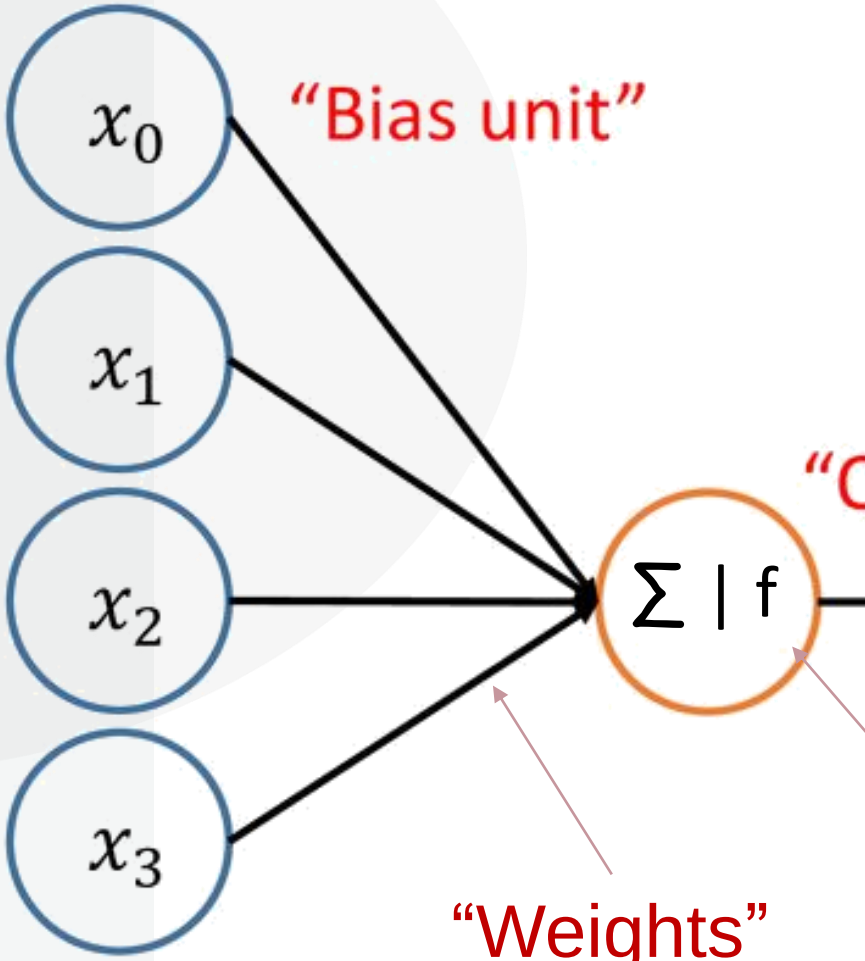
Neural Networks, Neurons and the Brain

- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications

Neural Networks, Neurons and the Brain



An artificial neuron: Logistic unit



"Bias unit"

"Input"

"Output"

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

"Weights"

"Parameters"

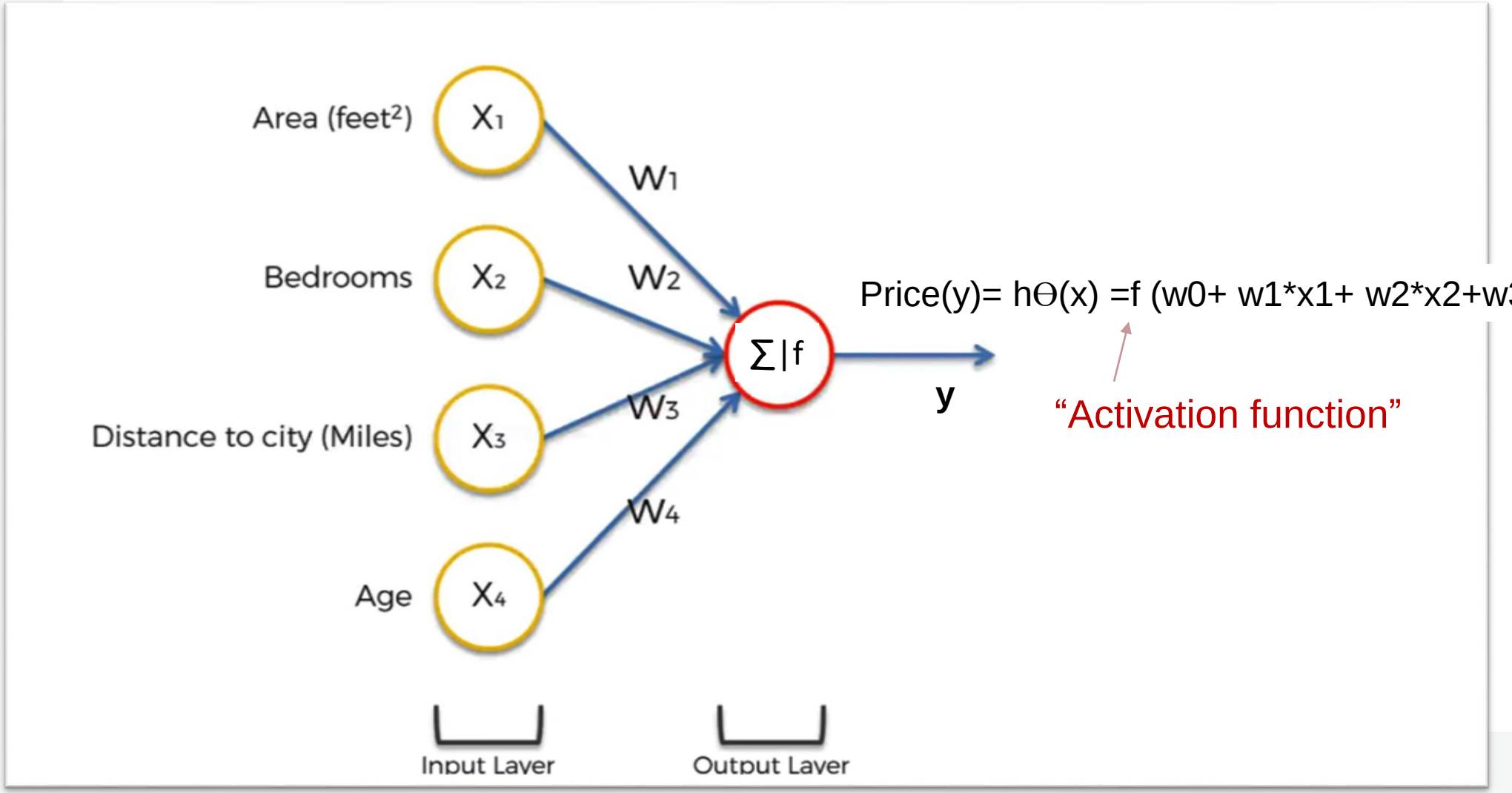
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^\top x}}$$

"Weights"

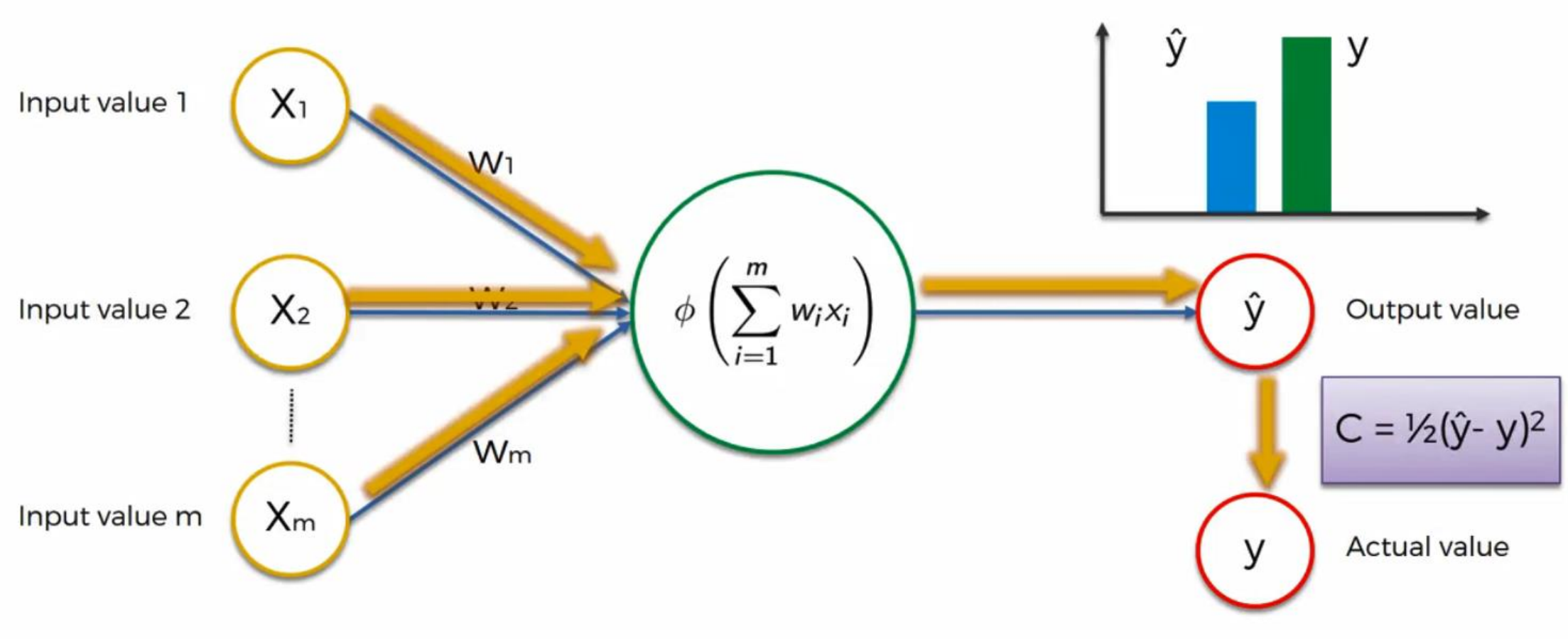
- Sigmoid (logistic) activation function

Slide credit: Andrew Ng

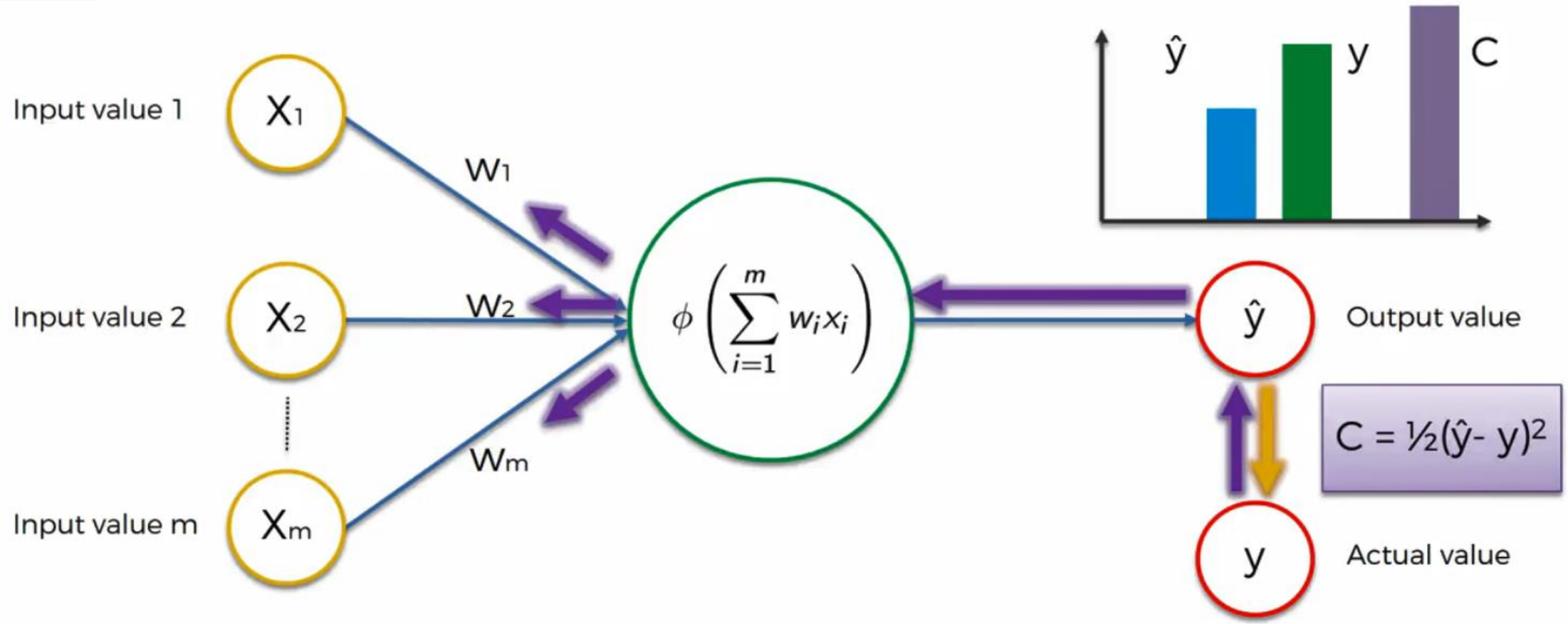
Neural networks: Simple example



Forward-Propagation

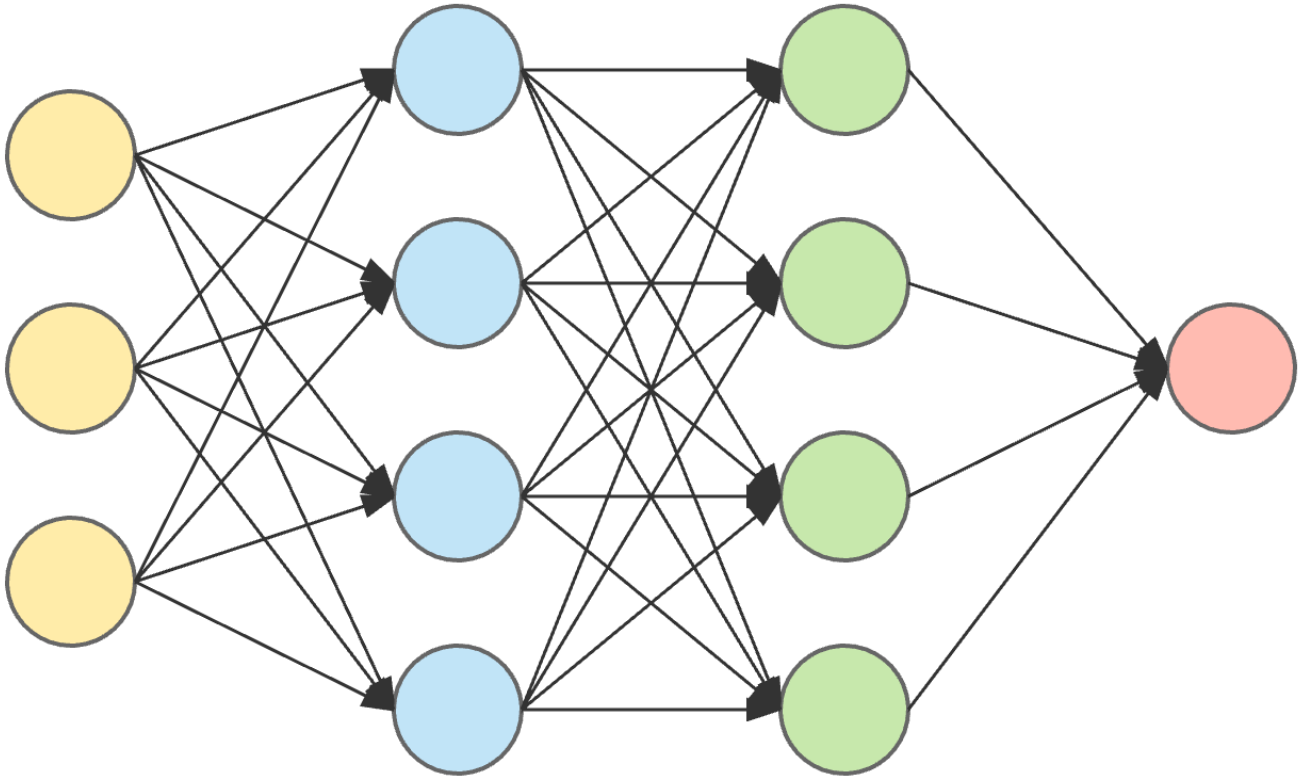


Backward-Propagation



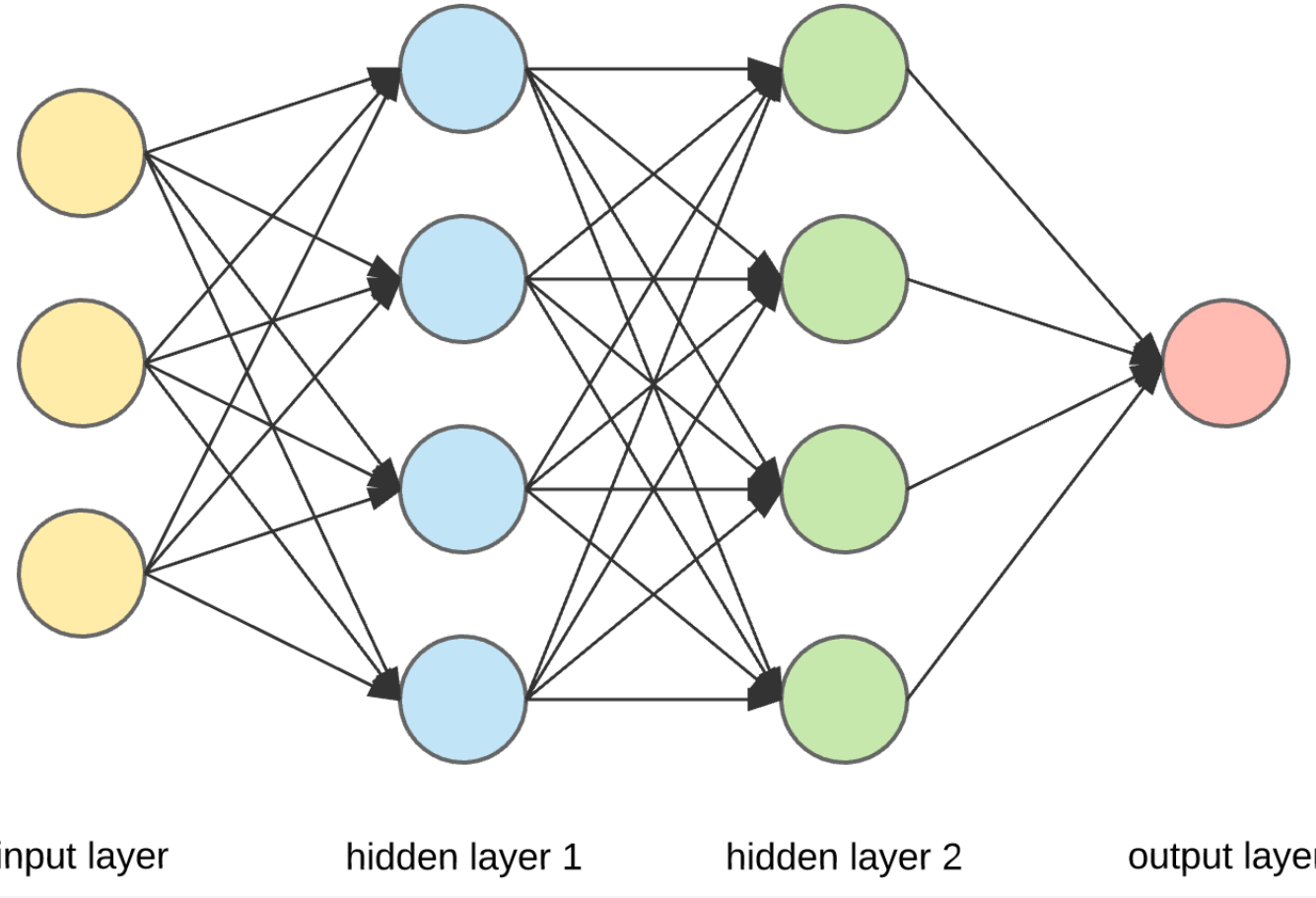
Artificial Neural networks (ANN)

- Artificial Neural Networks (ANN) are multi-layer **fully-connected** neural nets. Every node in one layer is connected to every other node in the next layer.



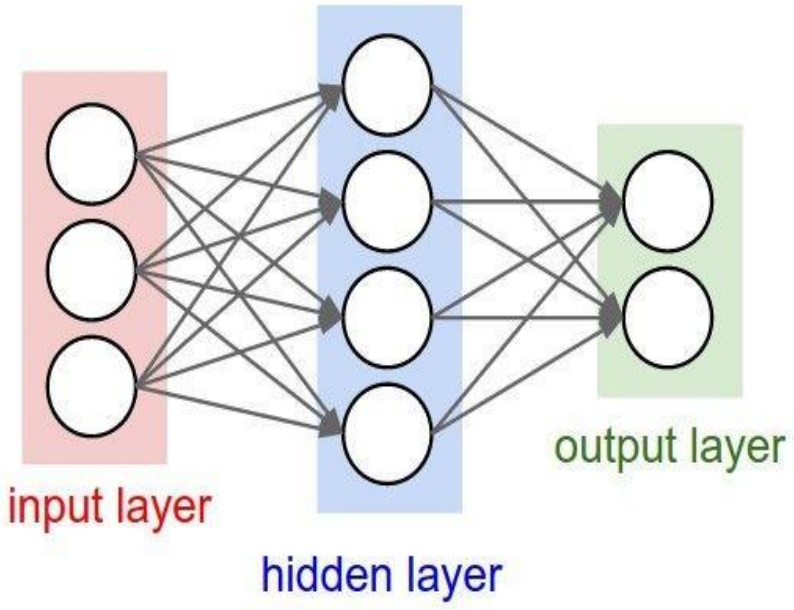
Artificial Neural networks (ANN) (Cont.)

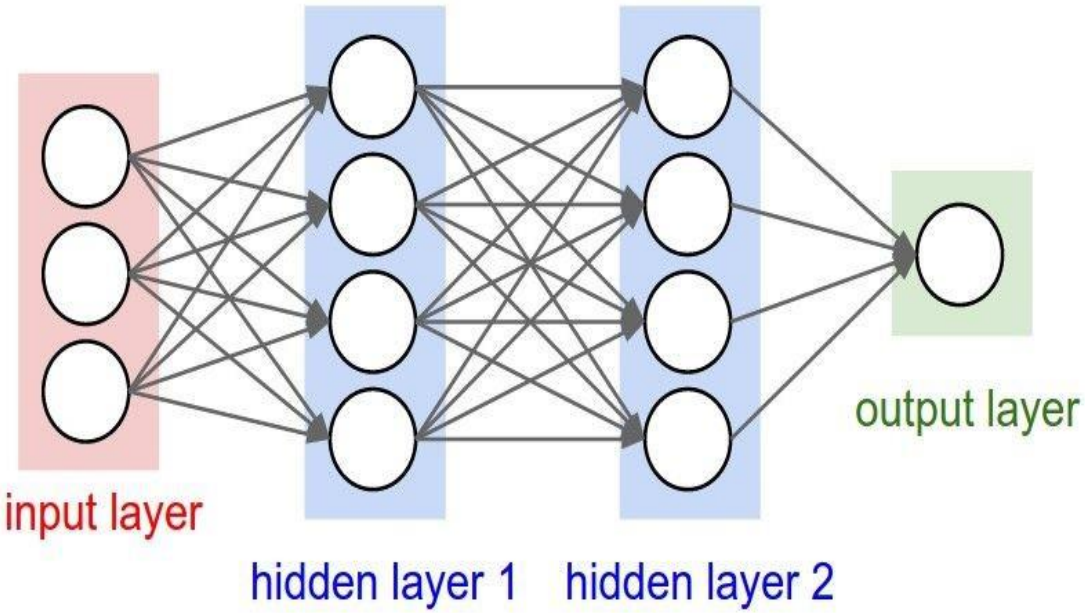
- They consist of an **input layer**, one or multiple **hidden layers**, and an **output layer**. To make the network deeper, the number of hidden layers is increased.



Artificial Neural networks(ANN) Architectures

basically, anything that isn't an input layer and isn't an output layer is called a hidden layer.

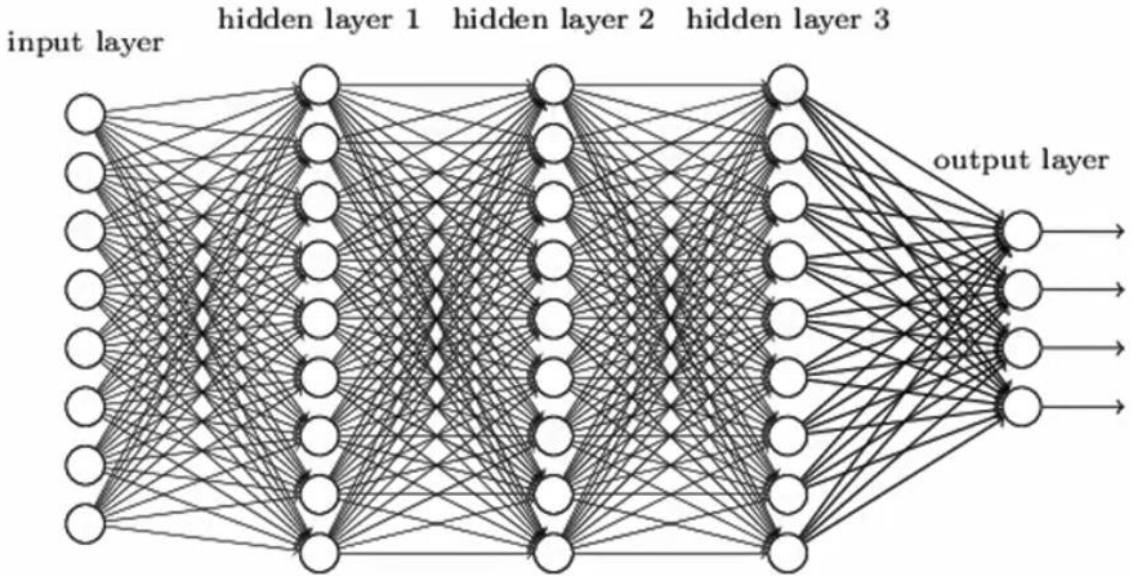




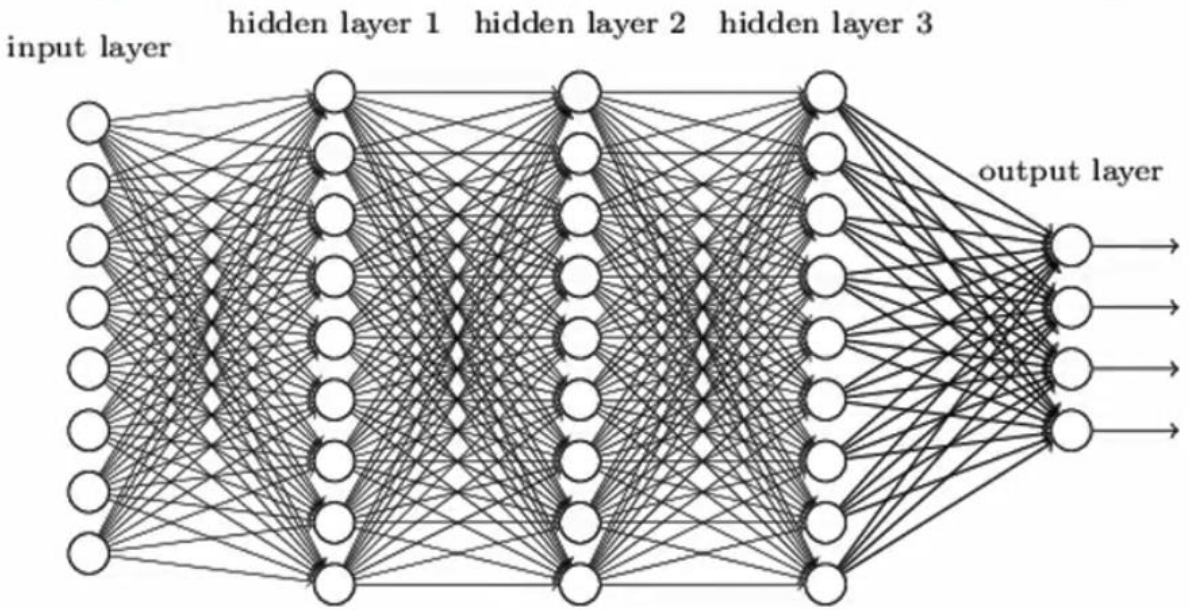
“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

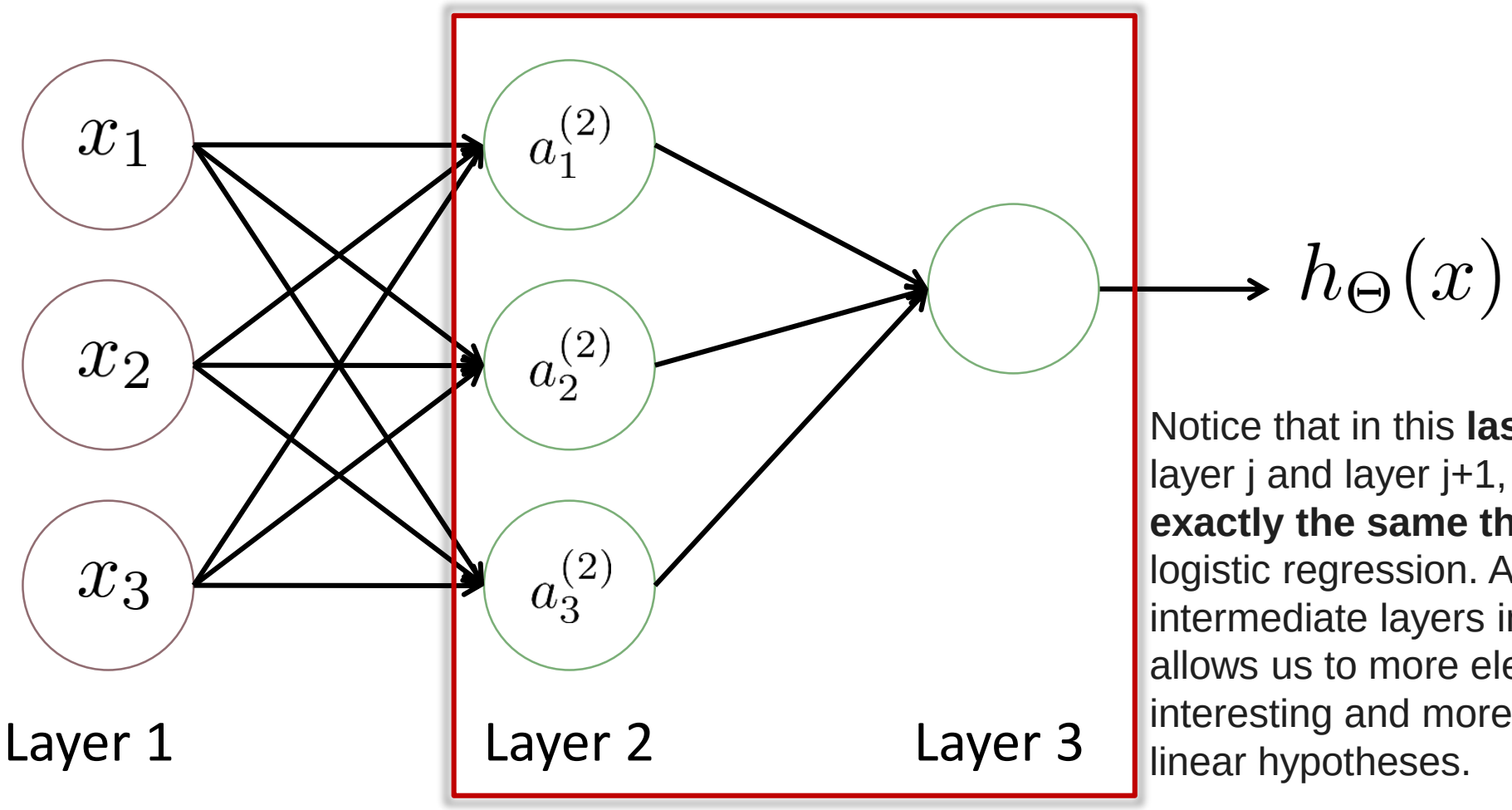
Forward Propagation



Backpropagation



Neural Network learning its own features

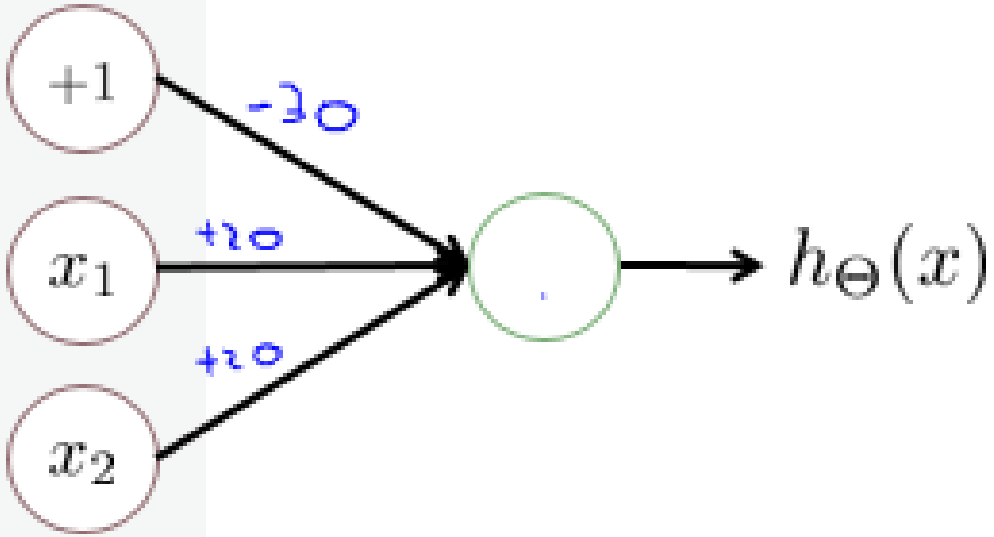


Notice that in this **last step**, between layer j and layer j+1, we are doing **exactly the same thing** as we did in logistic regression. Adding all these intermediate layers in neural networks allows us to more elegantly produce interesting and more complex non-linear hypotheses.

Simple example: AND

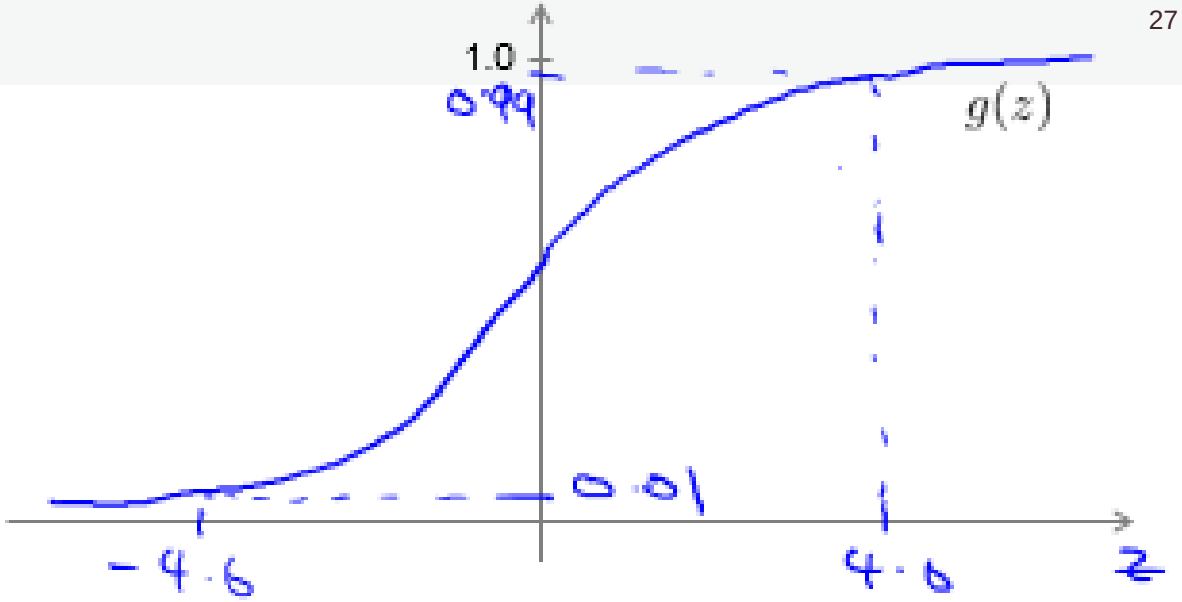
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



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

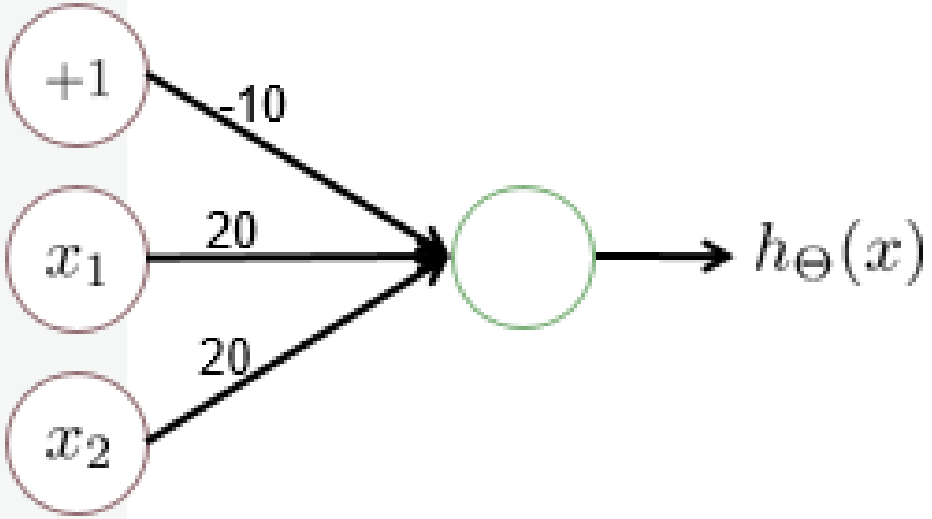
$\Theta^{(1)}_{10}$ $\Theta^{(1)}_{11}$ $\Theta^{(1)}_{12}$



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

Example: OR function

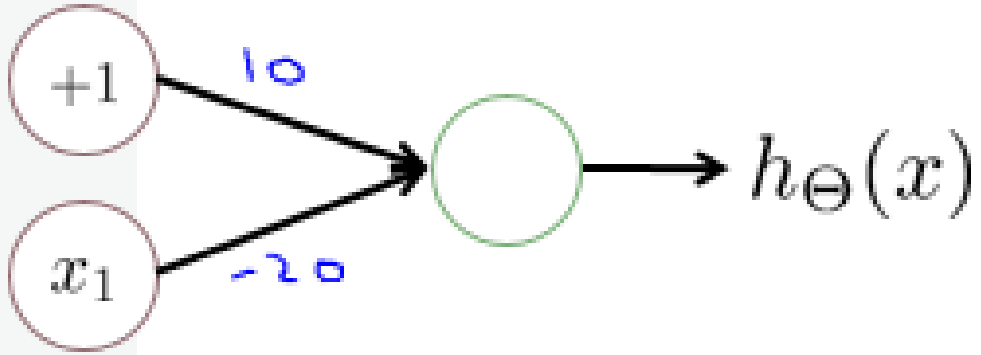


$g(-10 + 20x_1 + 20x_2)$

x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	≈ 1
1	1	≈ 1

Negation:

NOT x_1



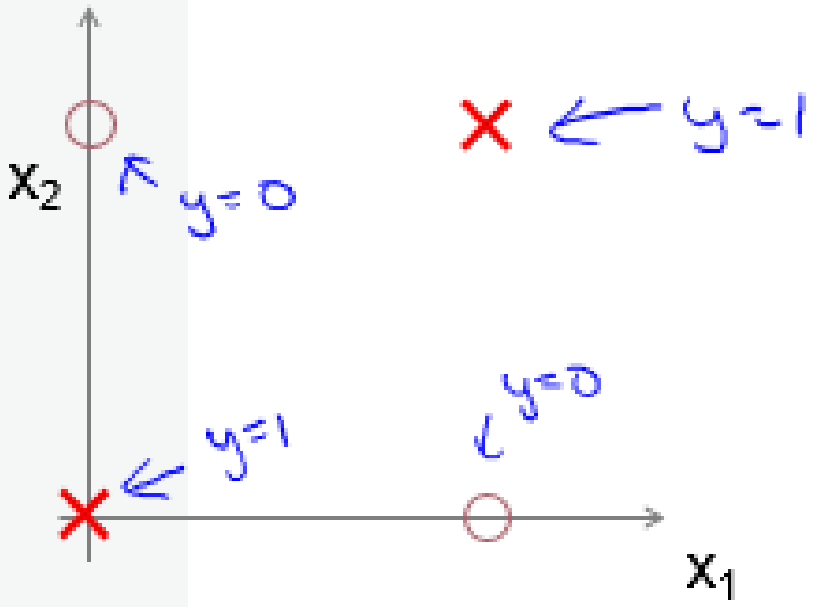
x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

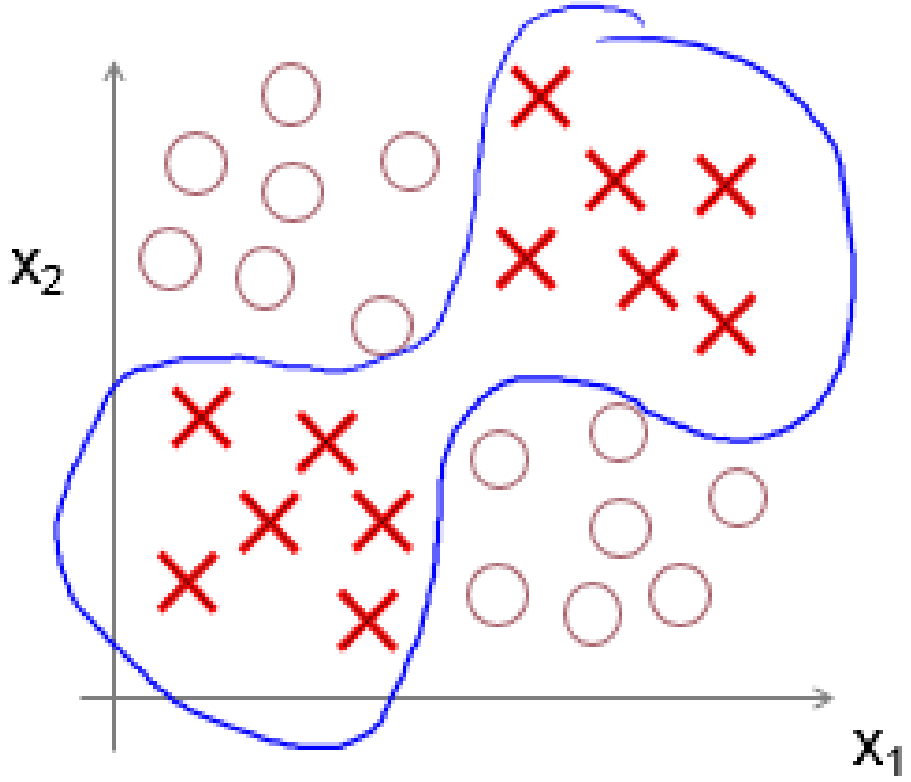
$h_\Theta(x) = g(10 - 20x_1)$

(NOT x_1) AND (NOT x_2) ?

Non-linear classification example: XOR/XNOR

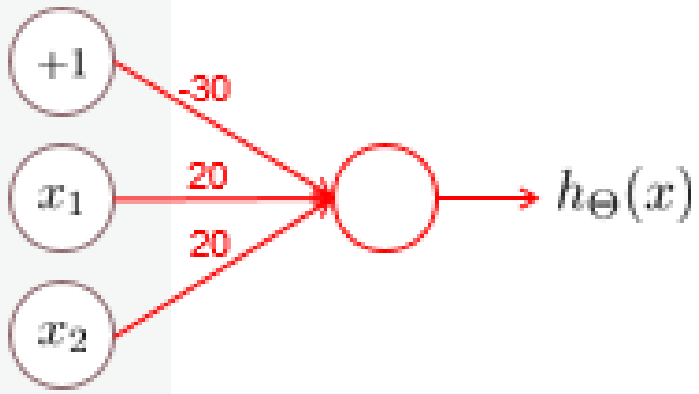
x_1 x_2 , are binary (0 or 1).



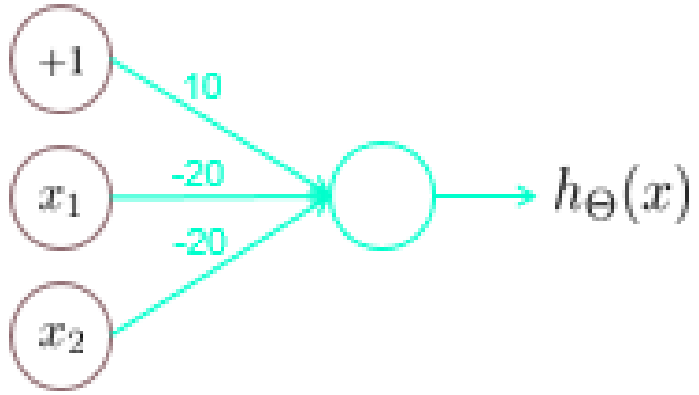


$$y = x_1 \text{ XOR } x_2$$

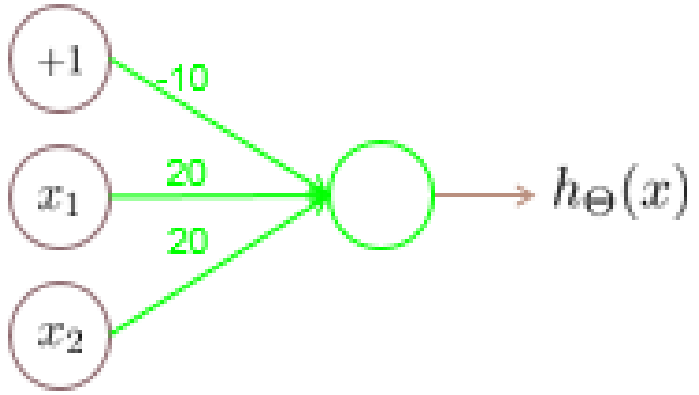
Putting it together: x_1 XNOR x_2



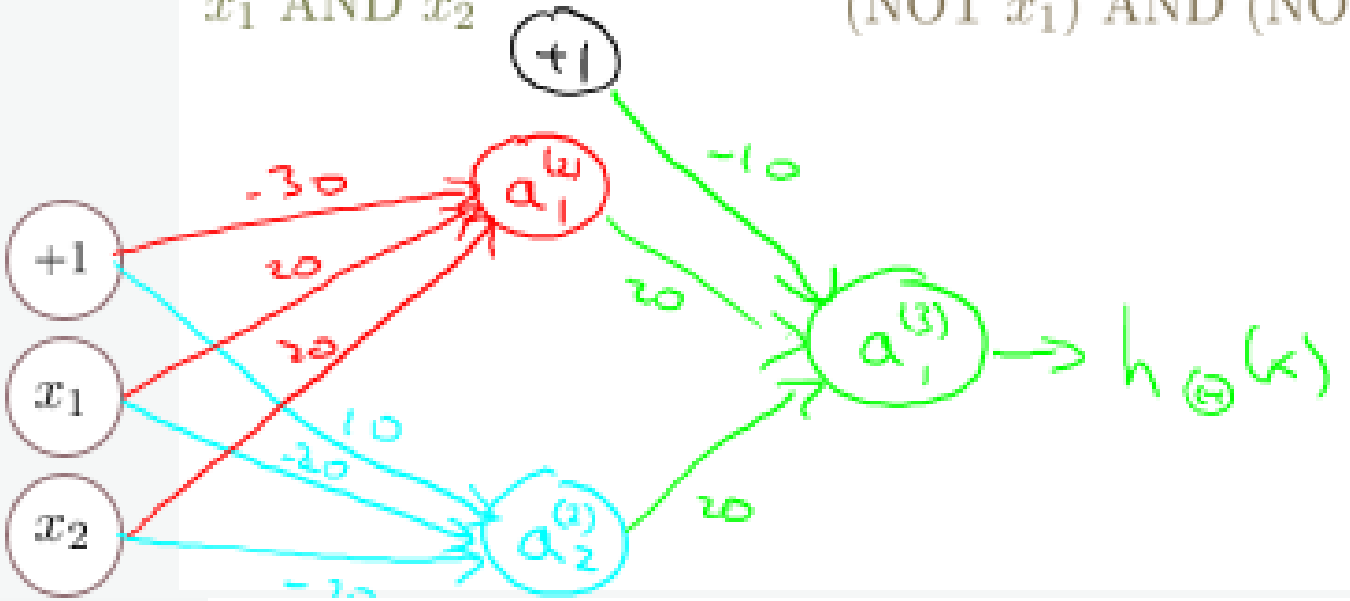
x_1 AND x_2



(NOT x_1) AND (NOT x_2)



x_1 OR x_2

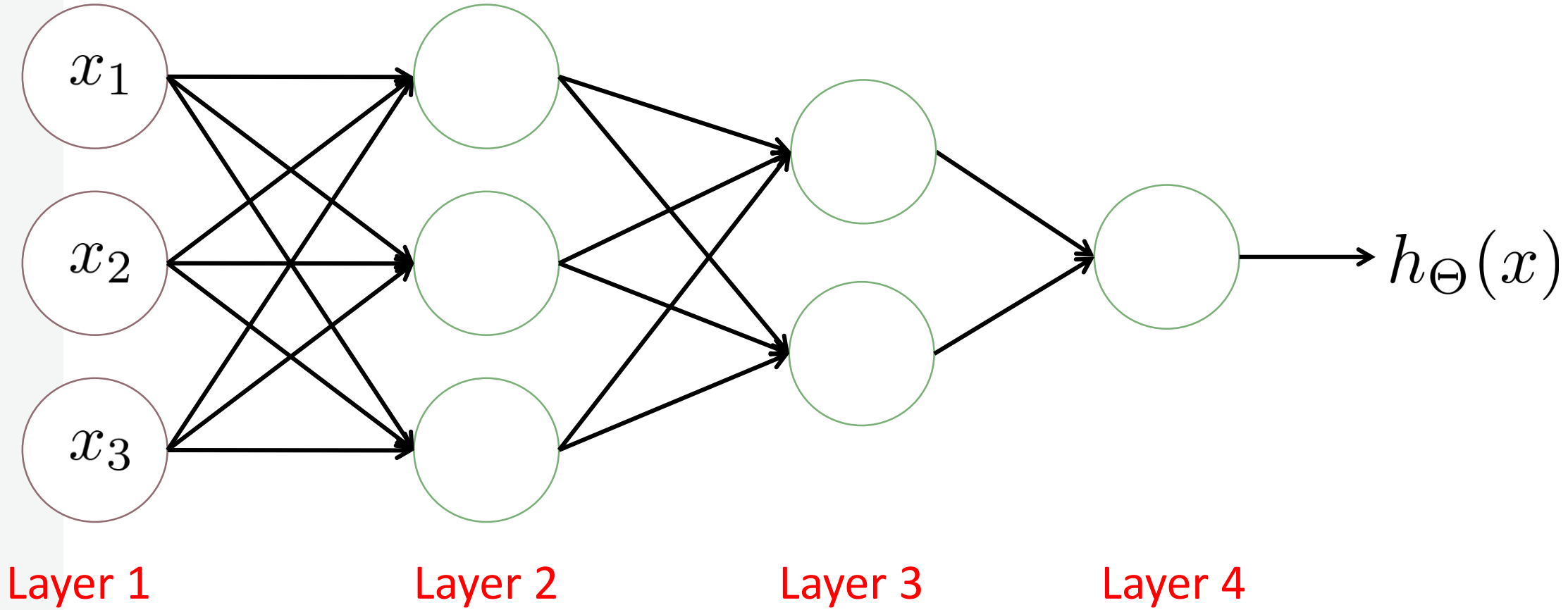


x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

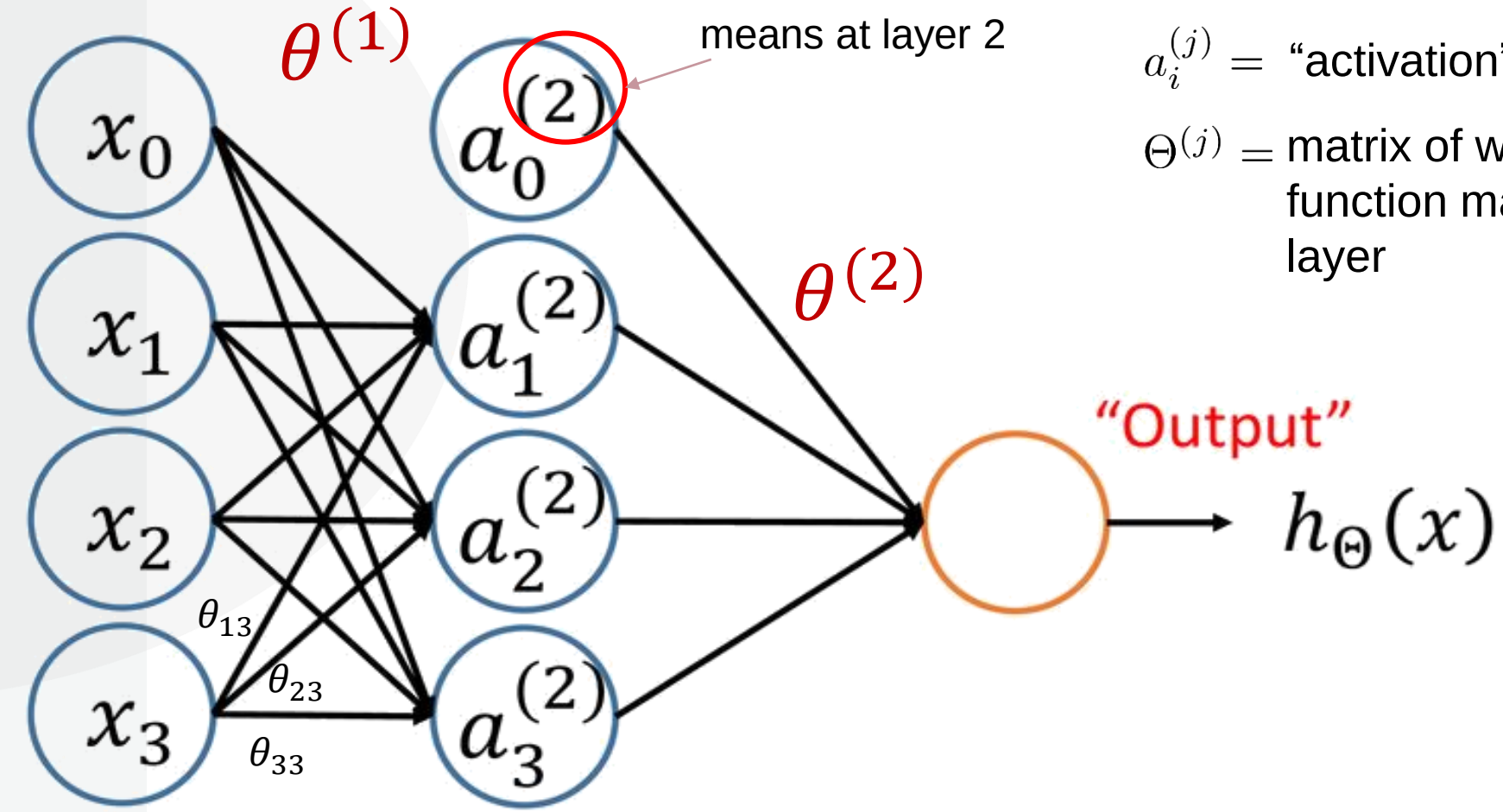
ANN Notation

Forward and Backward propagation

Layers Notation



Neural network – Multilayer



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

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

Slide credit: Andrew Ng

What is the dimension of $\Theta^{(1)}$?

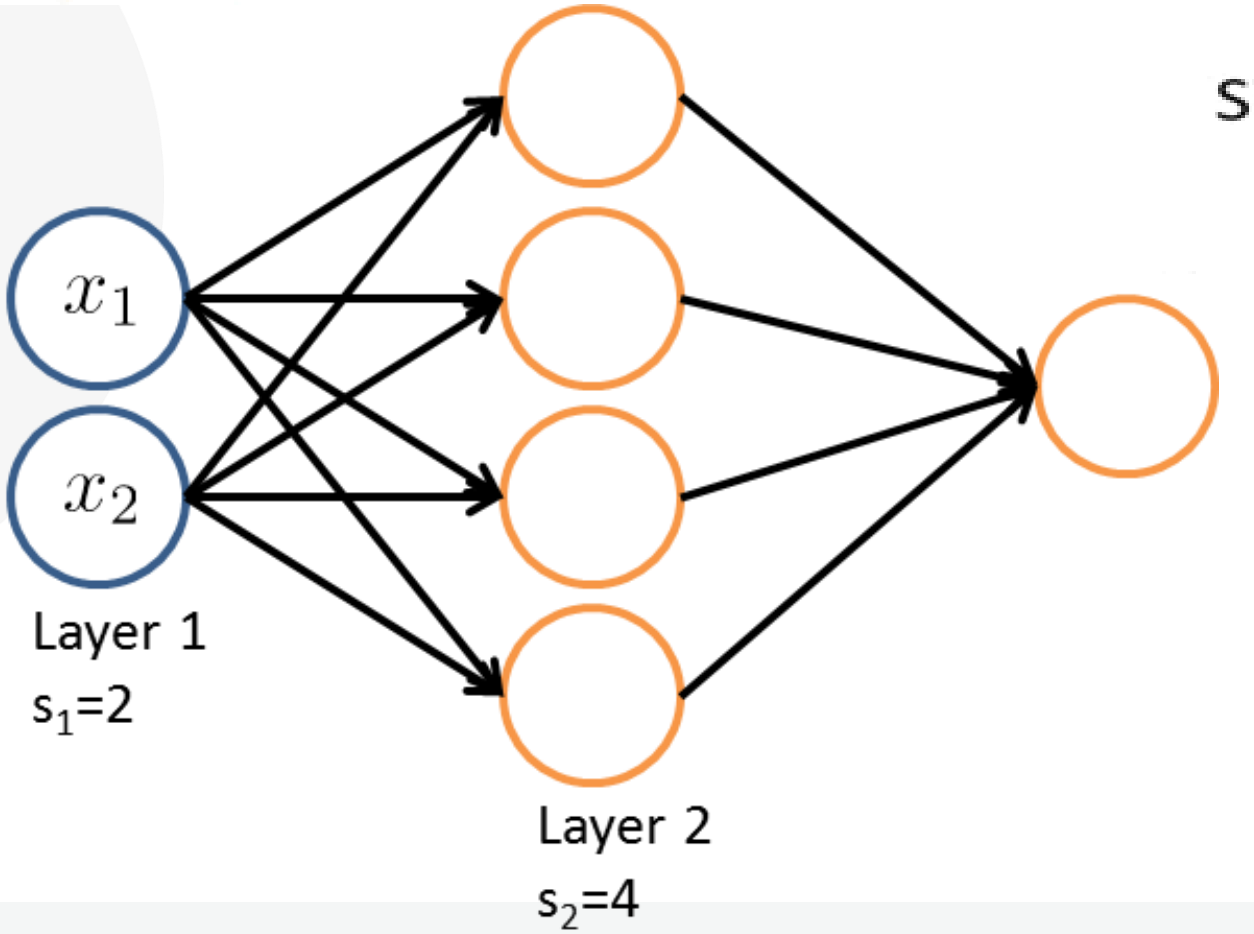
$s_{j+1} \times (s_j + 1)$ = 4 x 3

s_j unit in layer j

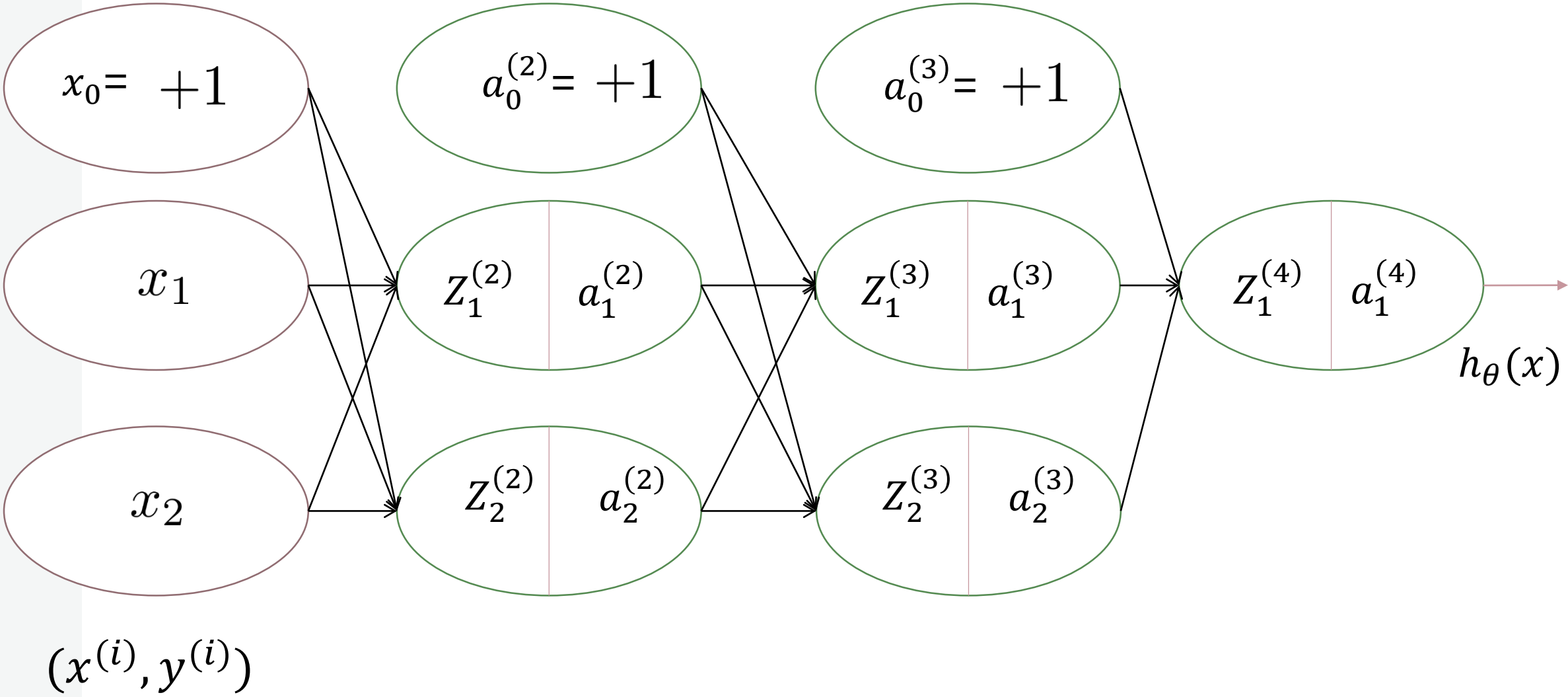
s_{j+1} units in layer $j + 1$

Size of $\Theta^{(j)}$?

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



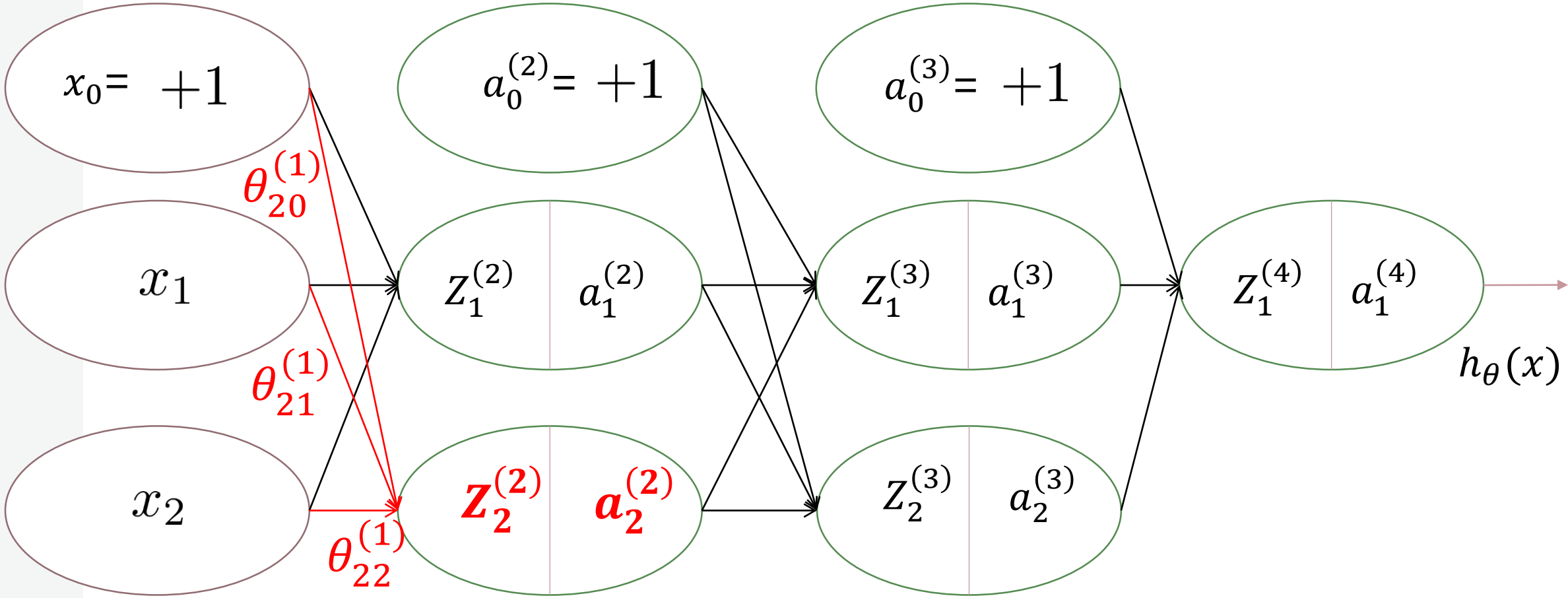
Forward Propagation



$(x^{(i)}, y^{(i)})$

One sample

Forward Propagation

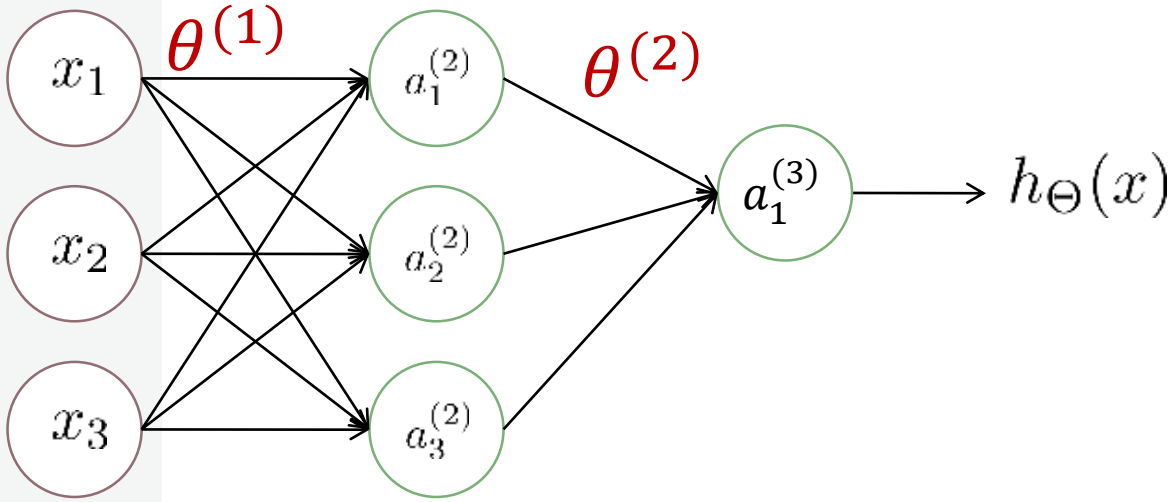


$(x^{(i)}, y^{(i)})$

One sample

$$Z_2^{(2)} = \theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 \qquad a_2^{(2)} = g(Z_2^{(2)}) = \frac{1}{1 + e^{-Z_2^{(2)}}}$$

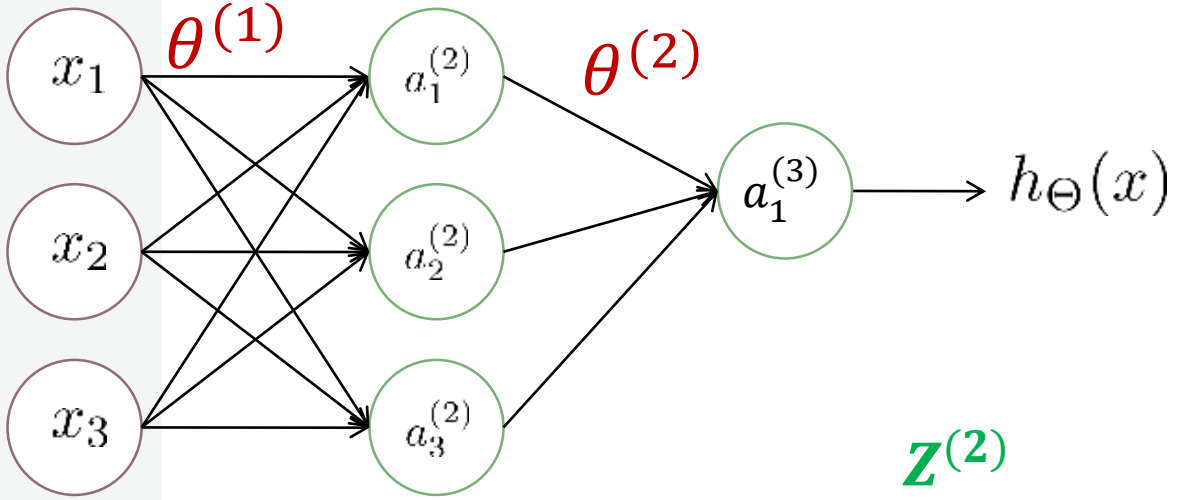
Forward propagation: Vectorized implementation



$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

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

Forward propagation: Vectorized implementation



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

$Z^{(2)}$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \quad Z_1^{(2)}$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \quad Z_2^{(2)}$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \quad Z_3^{(2)}$$

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

$Z^{(3)}$

Forward propagation

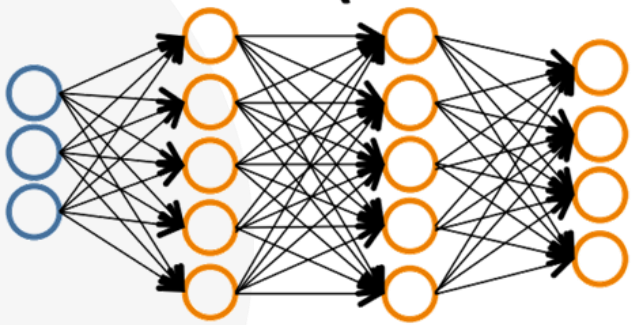
$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$

Add $a_0^{(2)} = 1$

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

Neural Networks: Learning

Neural Network (Classification)



$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$L =$ total no. of layers in network

$s_l =$ no. of units (not counting bias unit) in layer l

$s_1 = 3\,, s_2 = 5\,, s_3 = 5, s_4 = 4$

Binary classification

$y = 0$ or 1

1 output unit

$\mathrm{K} = 2$

$s_L = 1\ (output\ layer)$

Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

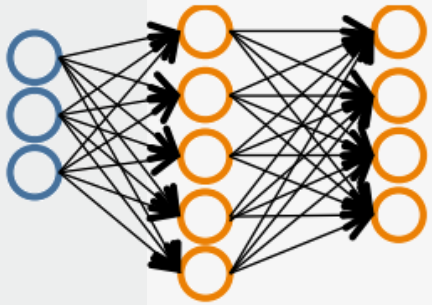
K output units

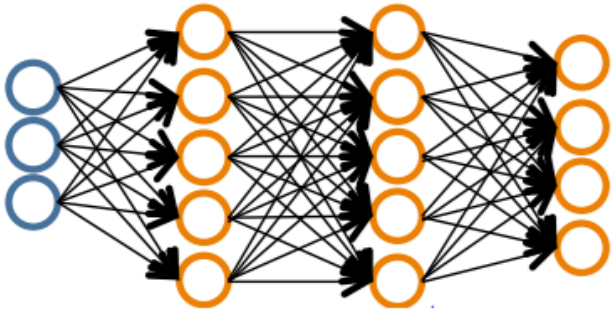
$s_L = K$

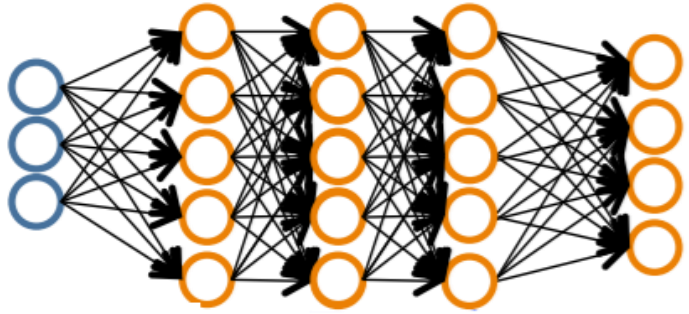
$K \geq 3$

Training a neural network

Pick a network architecture (connectivity pattern between neurons)







No. of input units: Dimension of features $x^{(i)}$

No. output units: Number of classes

Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

Gradient computation

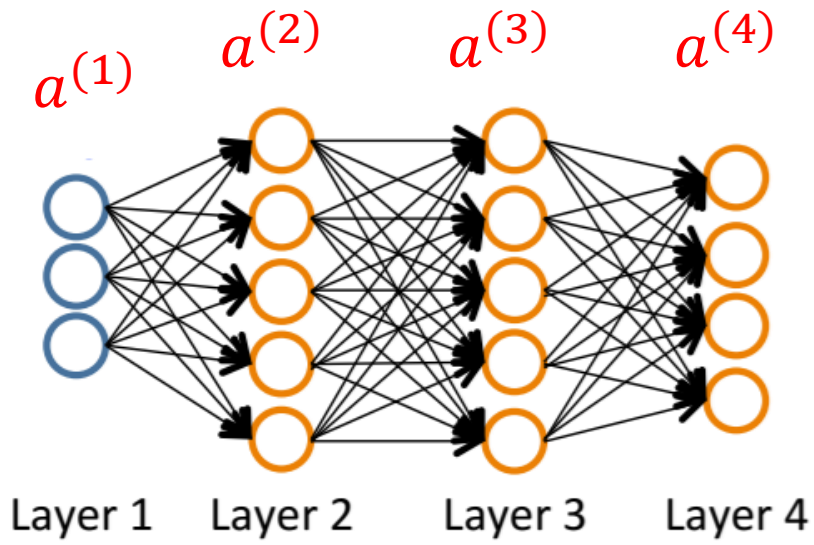
$a_i^{(j)}$ = "activation" of unit i in layer j

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

Given one training example (x, y):

Forward propagation:

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$



Cost function

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log((h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Gradient computation

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Gradient computation: Backpropagation algorithm

Intuition: $\delta_j^{(l)}$ = "error" of node j in layer l.

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost(i)}$ (for $j \geq 0$)

for each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

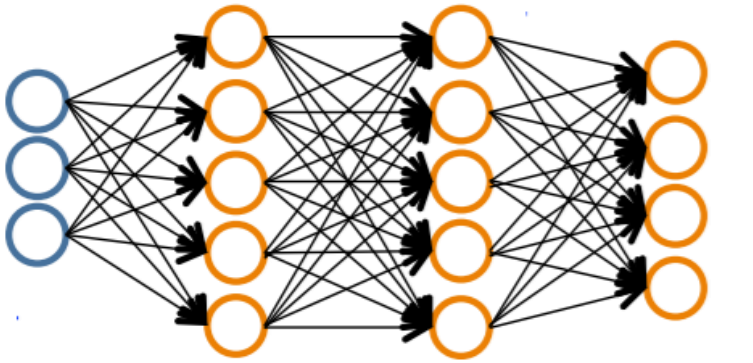
$h_\theta(x)_j$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

element wise multiplication

$\delta^{(2)}$ $\delta^{(3)}$ $\delta^{(4)}$



$$g'(z^{(l)}) = a^{(l)} .* (1 - a^{(l)})$$

Note:
We don't calculate the error of cost for **bias units** and **input units**

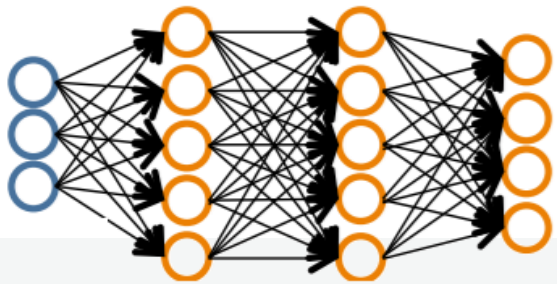
Training a neural network

1. Randomly initialize weights (Don't initialize with zero)
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

```
for i = 1:m
```

Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$

(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).



Training a neural network

5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.
   Then disable gradient checking code.
6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

In next Example

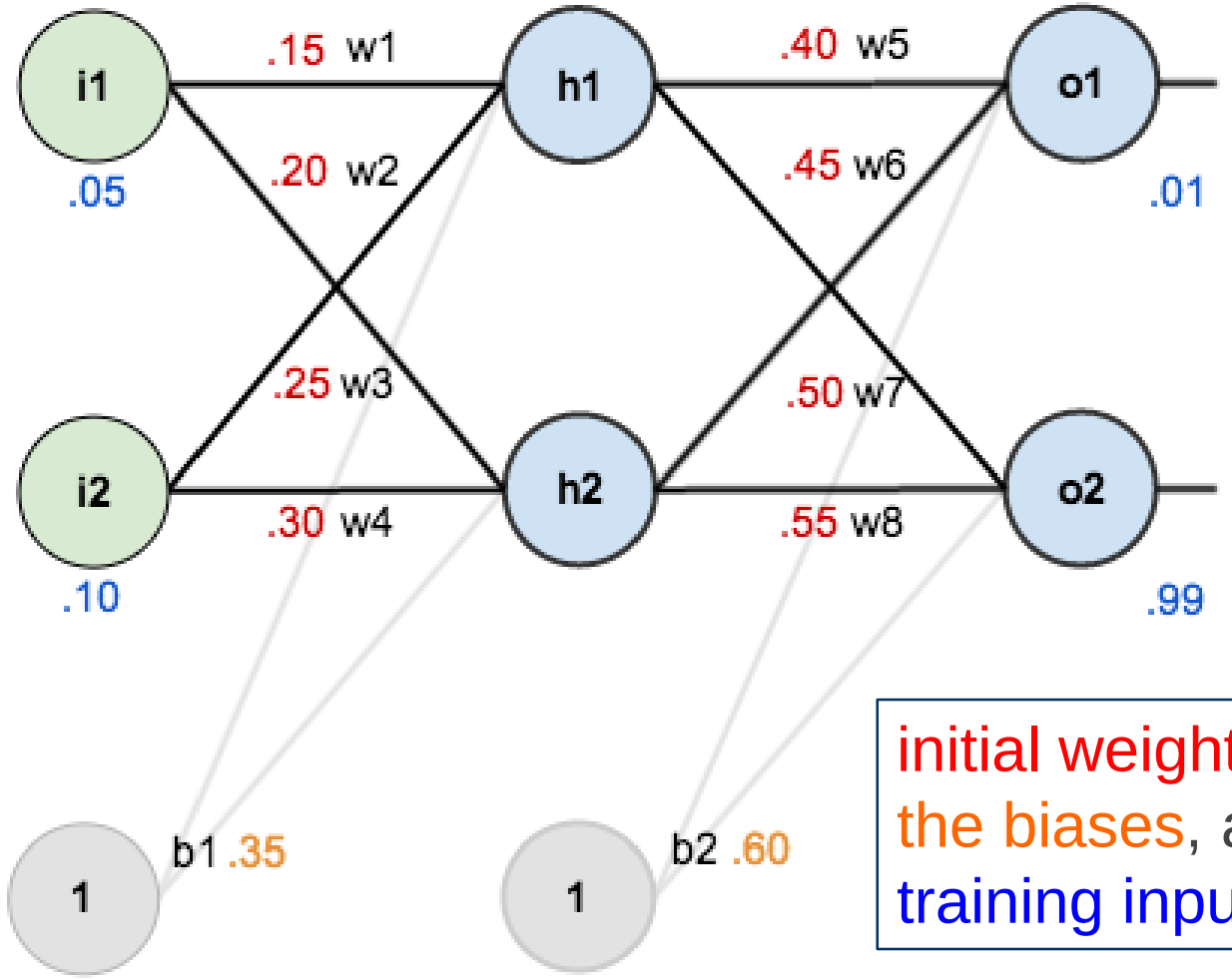
$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\Theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - (h_\Theta(x^{(i)})))\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Focusing on a single example $x^{(i)}$, $y^{(i)}$, and ignoring regularization ($\lambda = 0$),

$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$$

(Think of $\text{cost(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$)

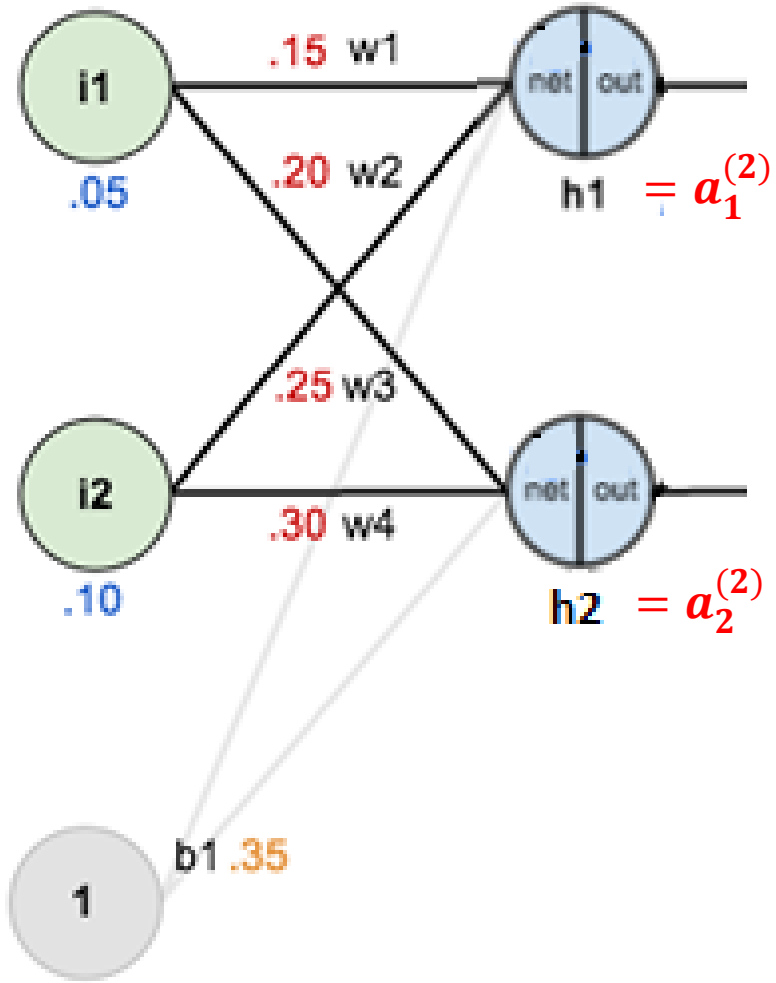
Numerical Example



initial weights
the biases, and
training inputs/outputs

- https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Numerical Example: Forward Pass-hidden layer



- Calculate total net input for h_1 (net_{h1})

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1 \qquad z_1^{(2)}$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

- Apply the sigmoid function to get the output of h_1 (out_{h1})

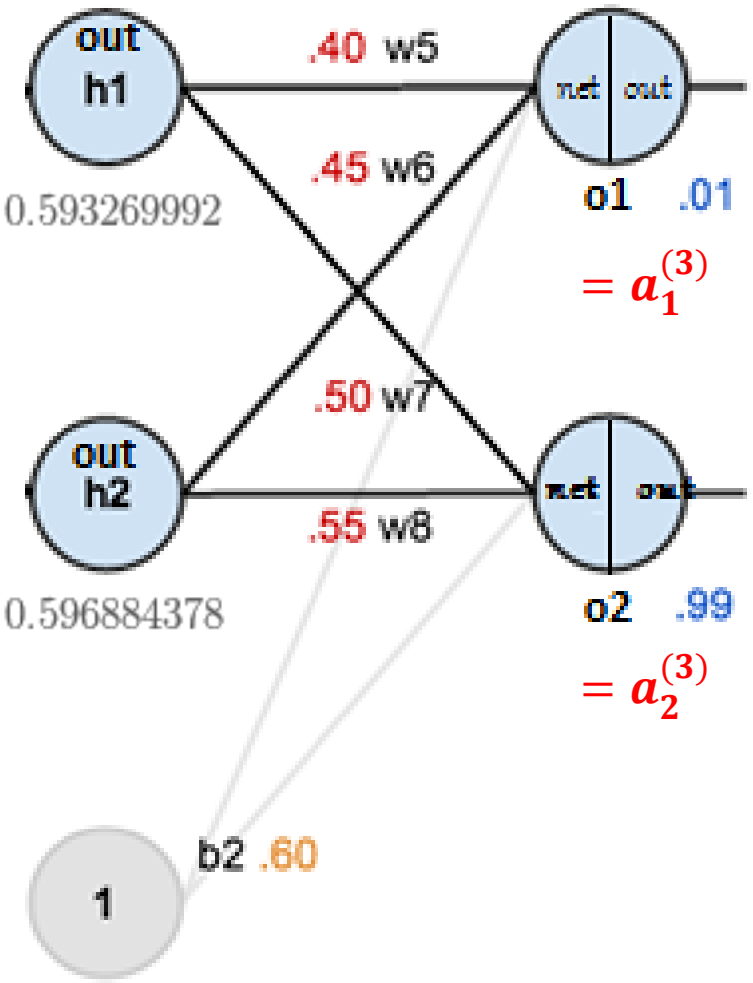
$a_1^{(2)} = g(z_1^{(2)})$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

- Carrying out the same process for h_2, we get:

$$out_{h2} = 0.596884378 \qquad a_2^{(2)}$$

Numerical Example: Forward Pass-output layer



Repeat same process for output layer:

Calculate total net input for o_1 (net_{o1})

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$\boldsymbol{z_1^{(3)}}$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1$$

$$= 1.105905967$$

Apply the sigmoid function to get the output of o_1 (out_{o1})

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$\boldsymbol{a_1^{(3)} = g(z_1^{(3)})}$

Carrying out the same process for o_2, we get:

$$out_{o2} = 0.772928465$$

$\boldsymbol{a_2^{(3)}}$

Numerical Example: Calculate Total Error

- Calculate the error for each output neuron using the squared error function and sum them to get the total error:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Used for an approximate cost for one sample

- For example, the target output for o1 is 0.01 but the neural network output 0.75136507, therefore its error is:

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

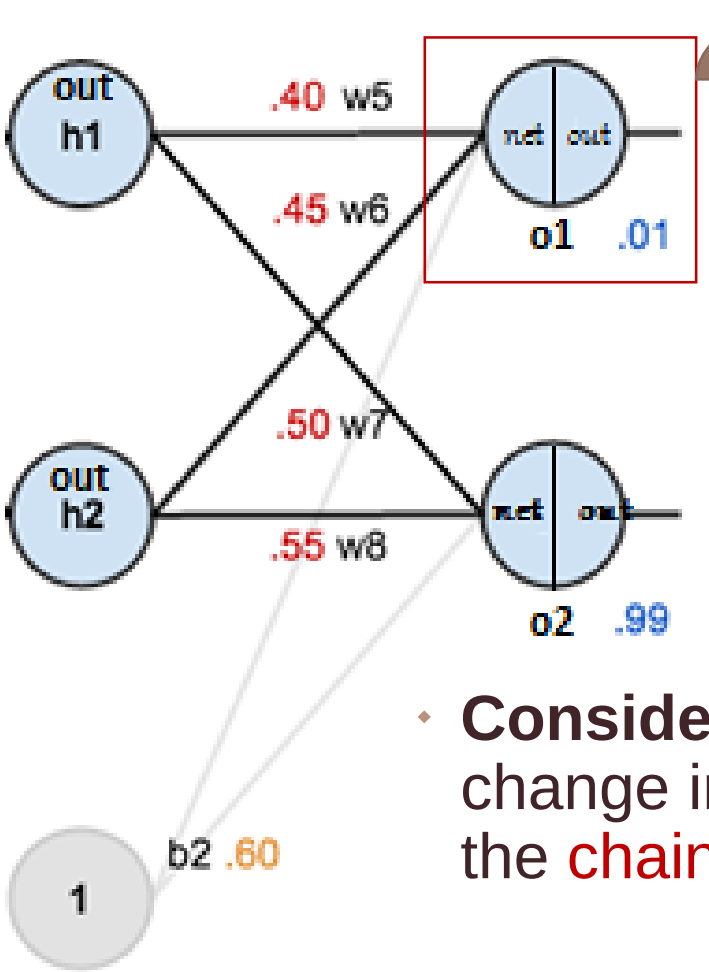
- Repeating this process for o2 (remembering that the target is 0.99) we get:

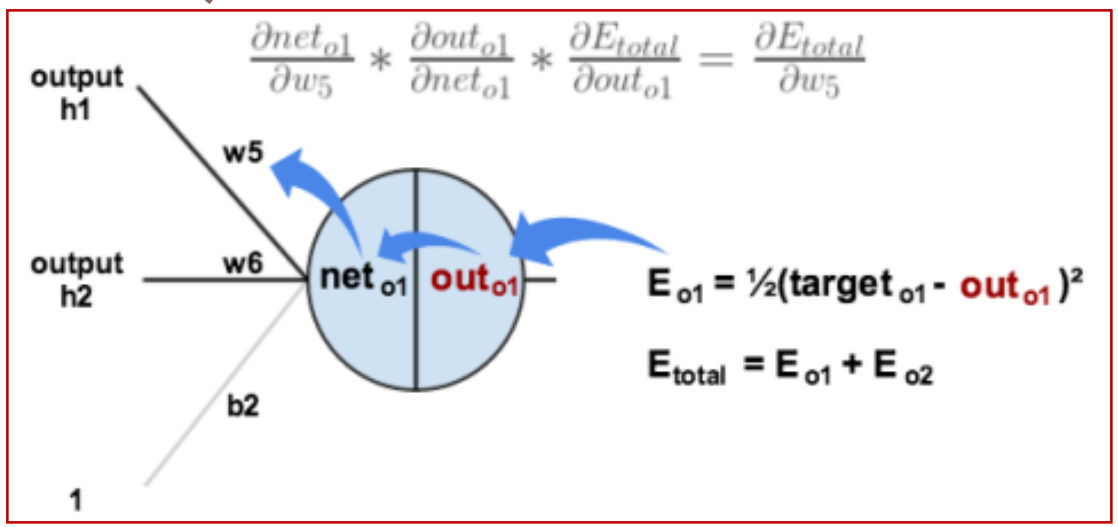
$$E_{o2} = 0.023560026$$

- The total error for the neural network is the sum of these errors:

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

Numerical Example: Backward Pass-output layer

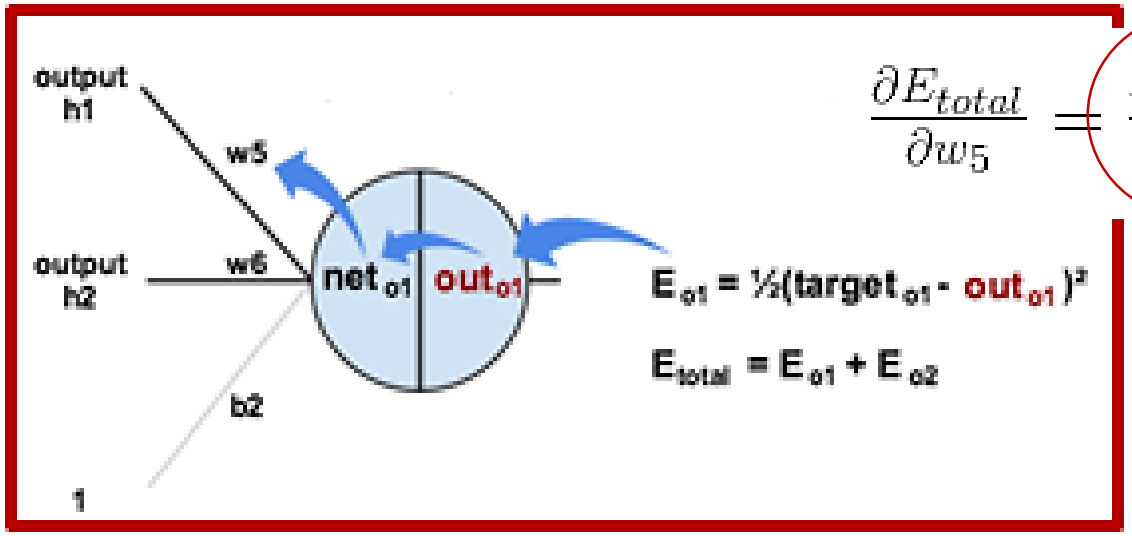




- **Consider w5**, we want to know how much a change in w5 affects the total error. By applying the chain rule we know that:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Numerical Example: Backward Pass-output layer



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

- 1- The total error change with respect to the output o1

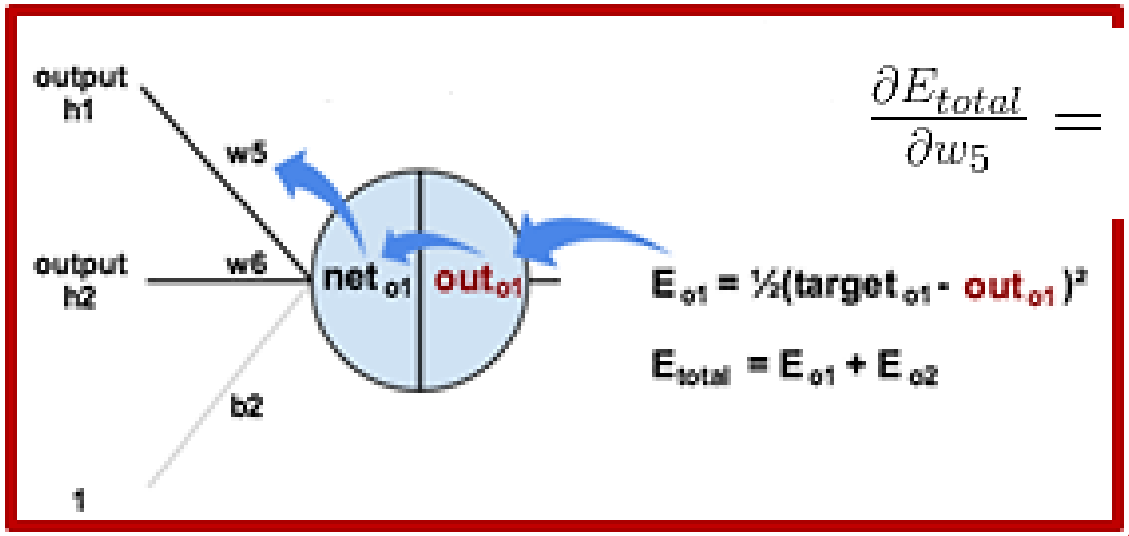
$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$-(target - out)$ is sometimes expressed as $out - target$

Numerical Example: Backward Pass-output layer



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

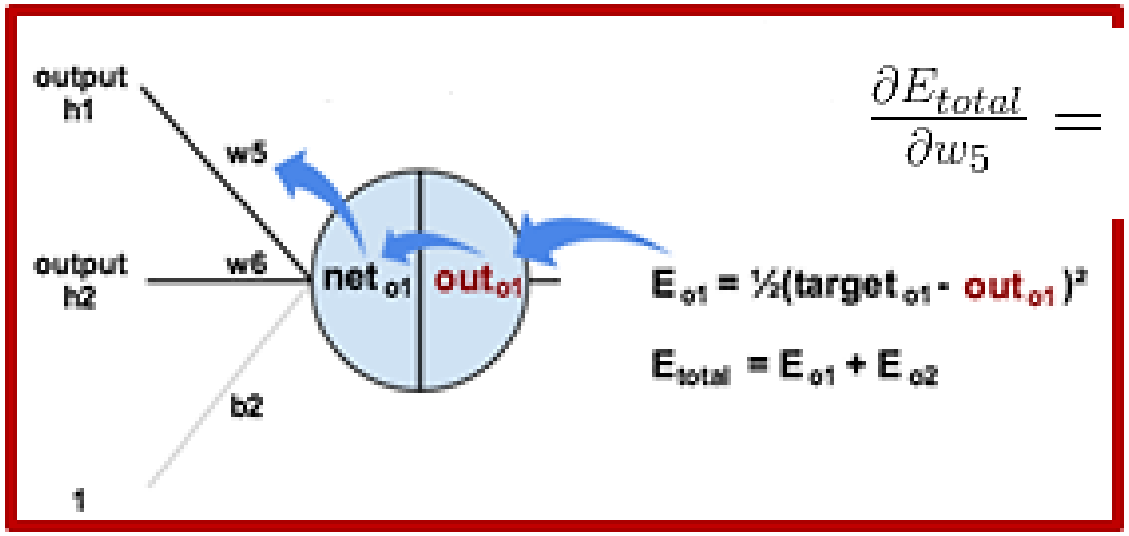
$$\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$$

- 2- The output o1 change with respect to its total net input

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1-out_{o1}) = 0.75136507(1-0.75136507) = 0.186815602$$

Numerical Example: Backward Pass-output layer



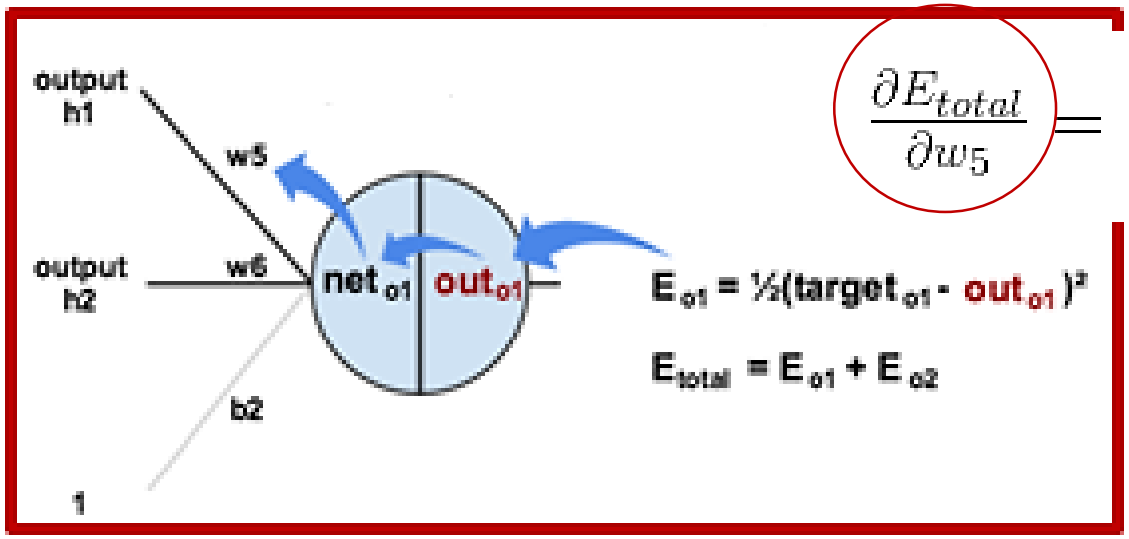
$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

- 3- The total net input of o1 change with respect to w5

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

Numerical Example: Backward Pass-output layer



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

- 4- Putting all together

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

Numerical Example: Backward Pass-output layer

- 5- To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate, eta, which we'll set to 0.5):

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

- We can repeat this process to all weights of output layer to get the new weights w6, w7 and w8:

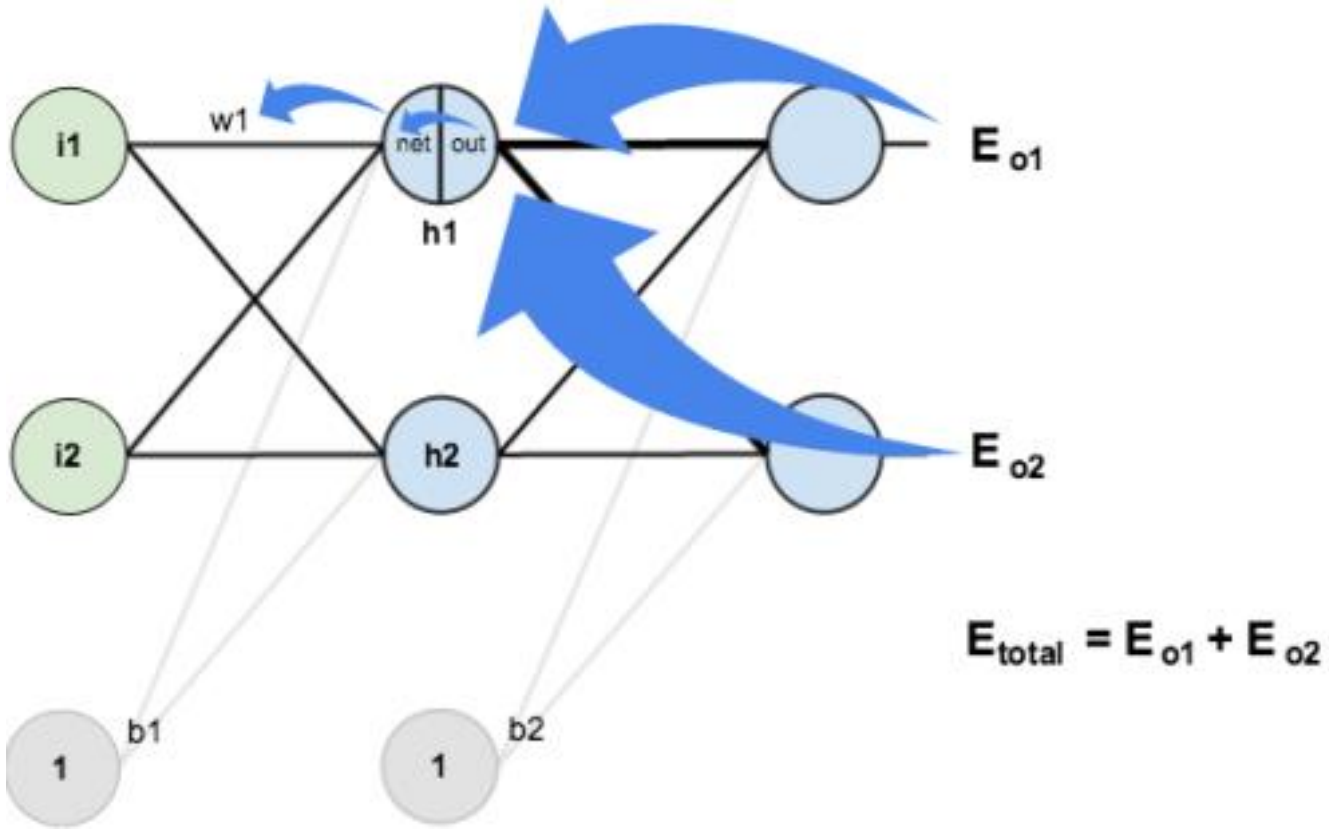
$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

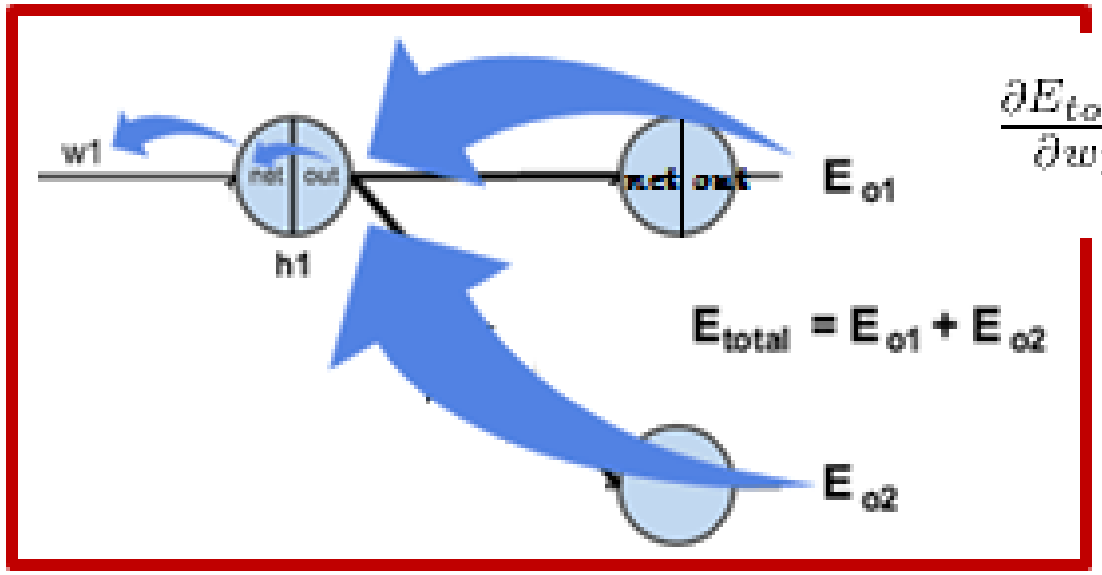
Numerical Example: Backward Pass-hidden layer

- we'll continue the backwards pass by calculating new values for w1, w2,w3 and w4



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$ where, $$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

Numerical Example: Backward Pass-hidden layer



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

Where,

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

- 1- Starting with $\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$

$= 0.74136507 * 0.186815602$

$= 0.138498562$

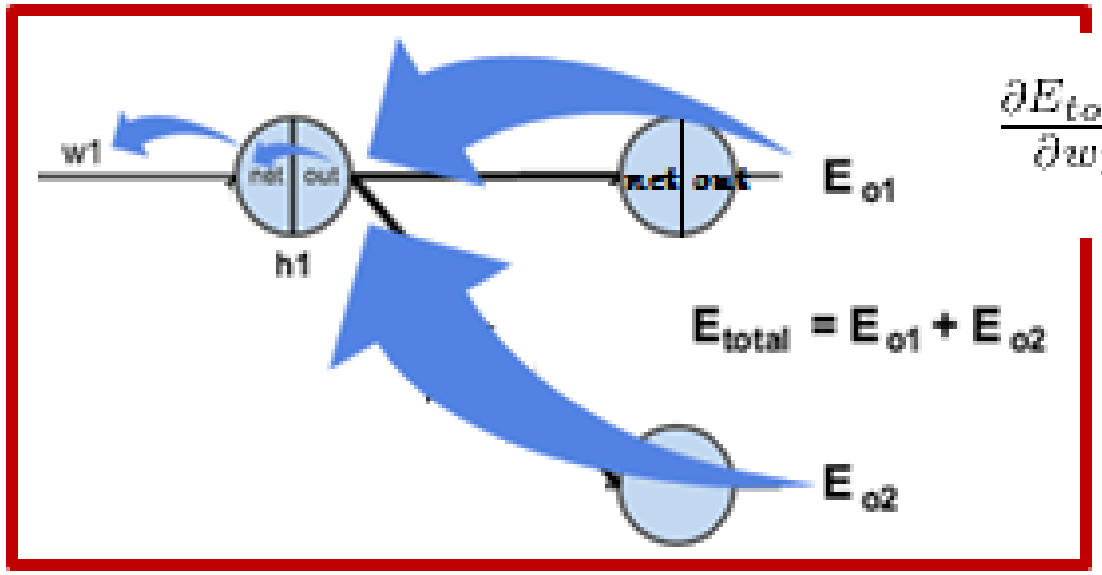
using values calculated earlier

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$ → Not updated weight

$$\frac{\partial E_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

Numerical Example: Backward Pass-hidden layer



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

Where,

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

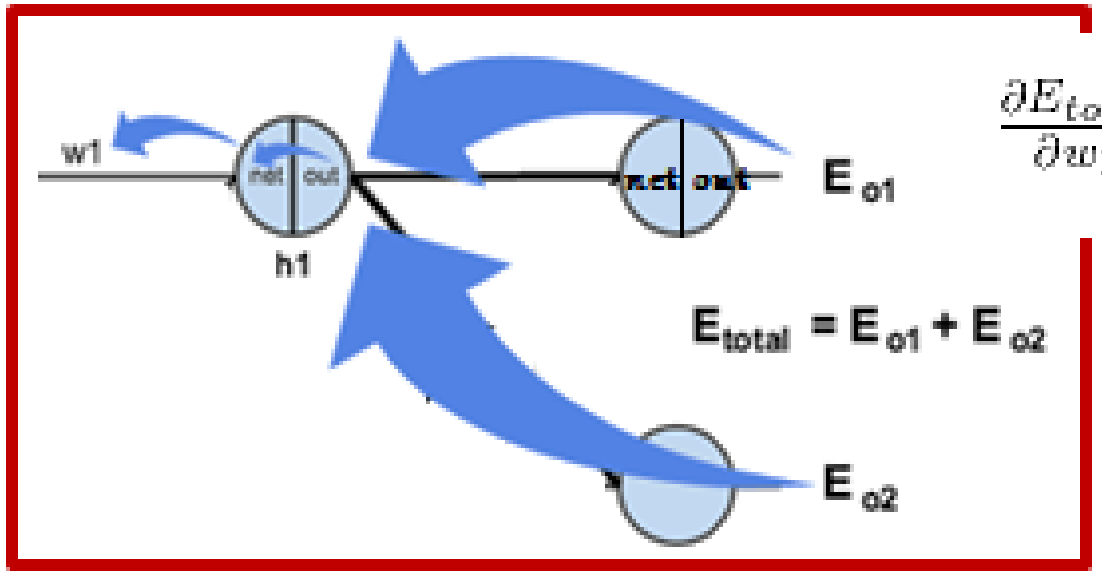
Following the same process for $\frac{\partial E_{o2}}{\partial out_{h1}}$, we get:

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

Therefore:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

Numerical Example: Backward Pass-hidden layer



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

2 3

Where,

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

- 2- $out_{h1} = \frac{1}{1+e^{-net_{h1}}}$

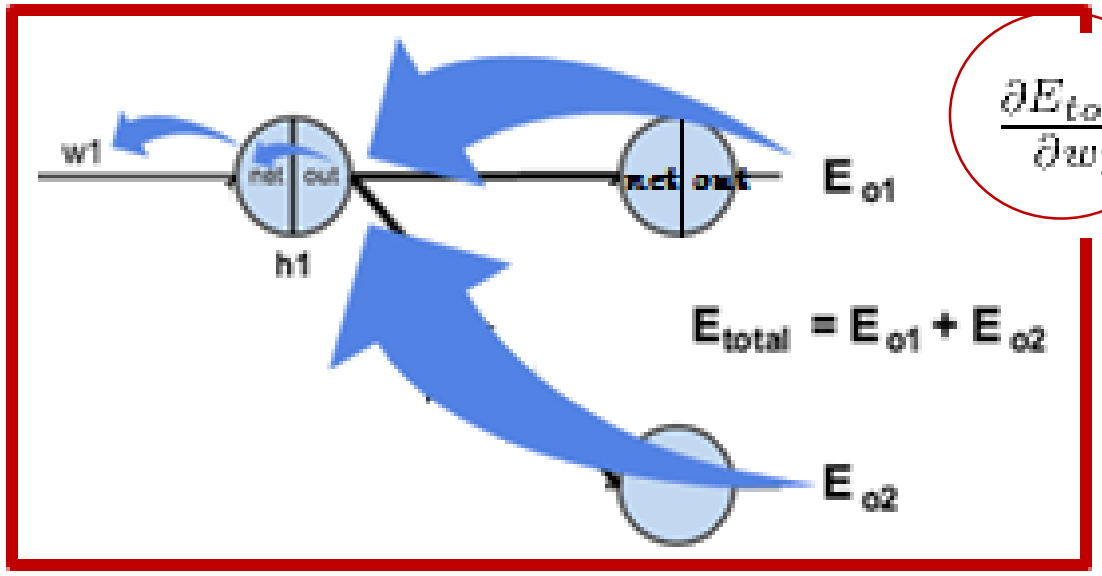
$$\boxed{\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

- 3- $net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

Numerical Example: Backward Pass-hidden layer



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

Where,

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

- 4- Put all together

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

Numerical Example: Backward Pass- hidden layer

- 5- We can update w1:

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

- We can repeat this process to all weights of hidden layer to get the new weights w2, w3 and w4:

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

Finally,

- When we fed forward the 0.05 and 0.1 inputs originally,
  - Total error was 0.298371109.
- After this first round of backpropagation,
  - Total error is now 0.291027924.
- But after repeating this process 10,000 times, for example,
  - Total error drops to 0.0000351085.
- At this point, when we feed forward 0.05 and 0.1,the two outputs neurons generate
  - 0.015912196 (vs 0.01 target) and
  - 0.984065734 (vs 0.99 target).

Derivative Rules

Derivative rules

- **Constant Rule:** $f(x) = c$ then $f'(x) = 0$
- **Constant Multiple Rule:** $g(x) = c \cdot f(x)$ then $g'(x) = c \cdot f'(x)$
- **Power Rule:** $f(x) = x^n$ then $f'(x) = nx^{n-1}$
- **Sum and Difference Rule:** $h(x) = f(x) \pm g(x)$ then $h'(x) = f'(x) \pm g'(x)$
- **Product Rule:** $h(x) = f(x)g(x)$ then $h'(x) = f'(x)g(x) + f(x)g'(x)$
- **Quotient Rule:** $h(x) = \dfrac{f(x)}{g(x)}$ then $h'(x) = \dfrac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$
- **Chain Rule:** $h(x) = f(g(x))$ then $h'(x) = f'(g(x))g'(x)$ $\dfrac{dy}{dx} = \dfrac{dy}{du}\dfrac{du}{dx}$

Derivative rules

- **Trig Derivatives:**
  - $f(x) = \sin(x)$ then $f'(x) = \cos(x)$
  - $f(x) = \cos(x)$ then $f'(x) = -\sin(x)$
  - $f(x) = \tan(x)$ then $f'(x) = \sec^2(x)$
  - $f(x) = \sec(x)$ then $f'(x) = \sec(x)\tan(x)$
  - $f(x) = \cot(x)$ then $f'(x) = -\csc^2(x)$
  - $f(x) = \csc(x)$ then $f'(x) = -\csc(x)\cot(x)$

- **Exponential Derivatives**
  - $f(x) = a^x$ then $f'(x) = \ln(a)a^x$
  - $f(x) = e^x$ then $f'(x) = e^x$
  - $f(x) = a^{g(x)}$ then $f'(x) = \ln(a)a^{g(x)}g'(x)$
  - $f(x) = e^{g(x)}$ then $f'(x) = e^{g(x)}g'(x)$

Derivative rules

- **Logarithm Derivatives**
  - $f(x) = \log_a(x)$ then $f'(x) = \dfrac{1}{\ln(a)x}$
  - $f(x) = \ln(x)$ then $f'(x) = \dfrac{1}{x}$
  - $f(x) = \log_a(g(x))$ then $f'(x) = \dfrac{g'(x)}{\ln(a)g(x)}$
  - $f(x) = \ln(g(x))$ then $f'(x) = \dfrac{g'(x)}{g(x)}$

Thanks