GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

LECTURE 18: NEURAL NETWORKS

I. RECOMMENDATION SYSTEMS
II. CONTENT-BASED FILTERING
III. COLLABORATIVE FILTERING
IV. MATRIX FACTORIZATION (ILLUSTRATIVE EXAMPLE)
V. THE NETFLIX PRIZE

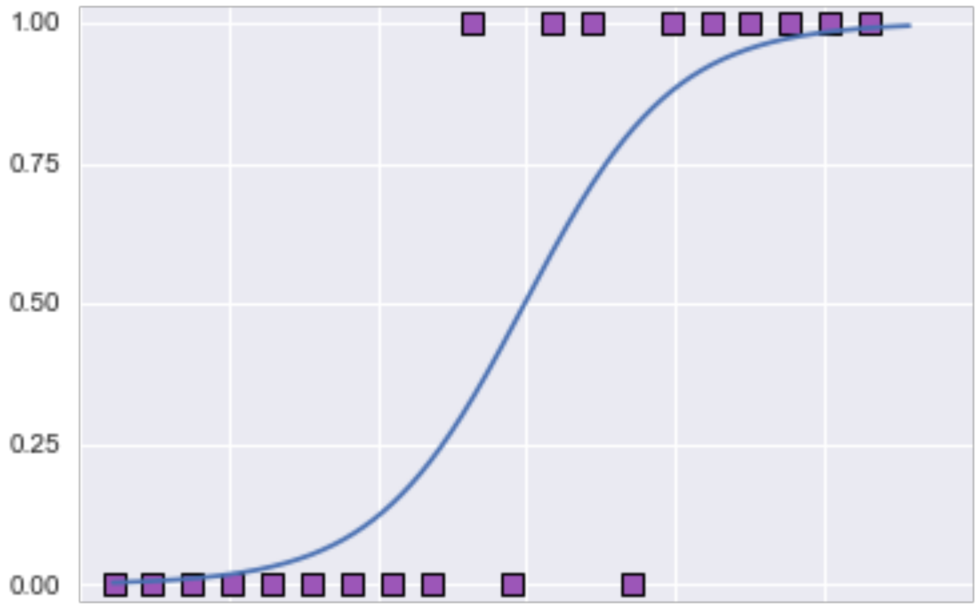
I. NEURAL NETWORKS
II. LOGICAL OPERATORS
III. NEURAL NETWORKS VARIETIES
IV. COST FUNCTION
V. BACKPROPAGATION

- ILLUSTRATE A NEURAL NETWORK ARCHITECTURE
- ILLUSTRATE HOW A NEURAL NETWORK CAN CONSTRUCT A NON-LINEAR DECISION BOUNDARY USING THE XOR PROBLEM
- ILLUSTRATE THE BACKPROPAGATION ALGORITHM
- ILLUSTRATE OTHER TYPES OF NEURAL NETWORKS

I. NEURAL NETWORKS

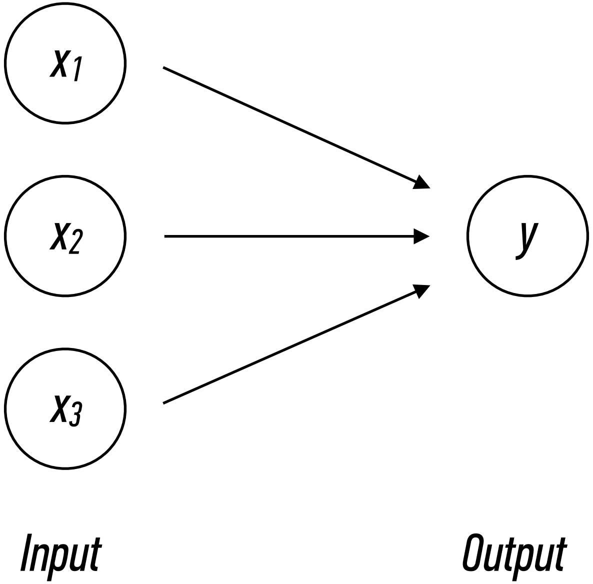
Let's review logistic regression...

$$y = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)}}$$

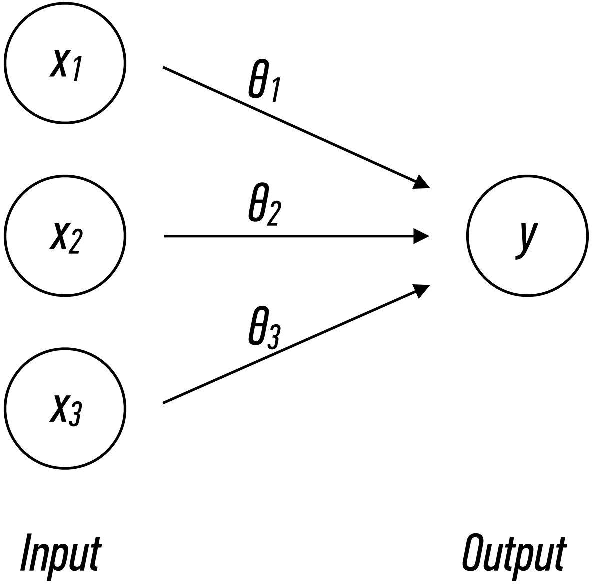


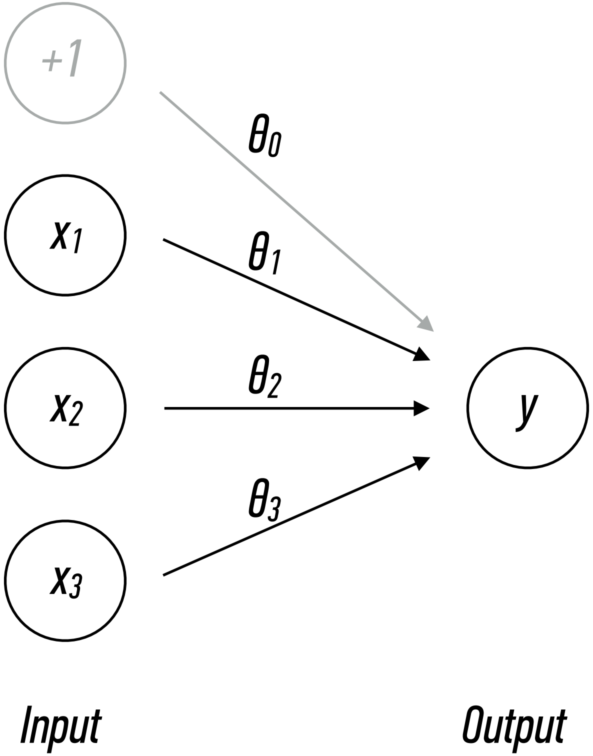
...and introduce some new notation

$$y = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)}}$$

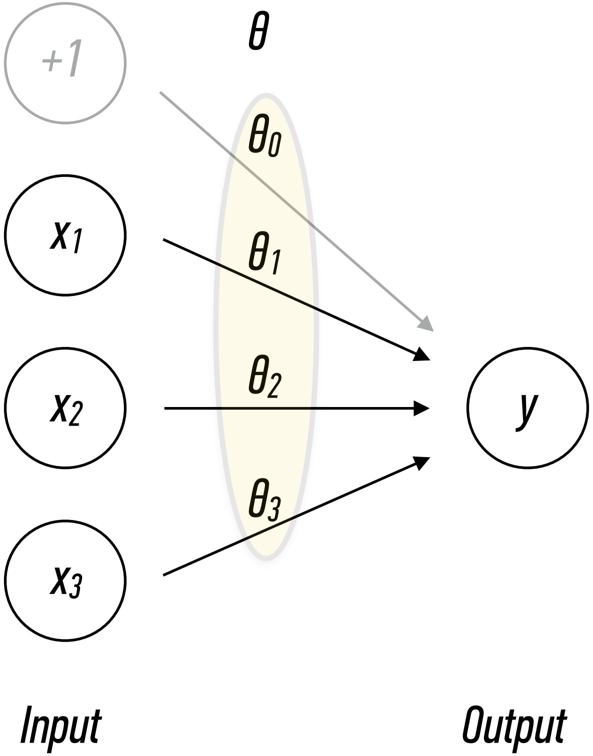


$$y = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)}}$$



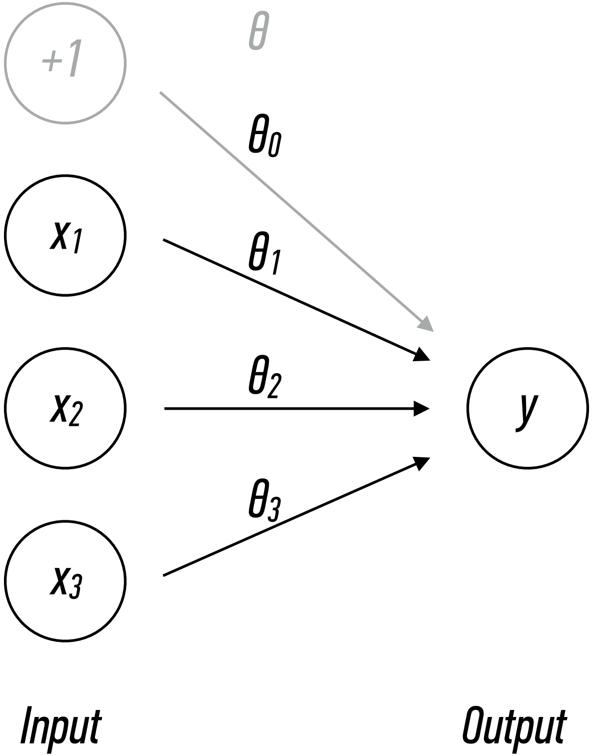


$$y = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)}}$$



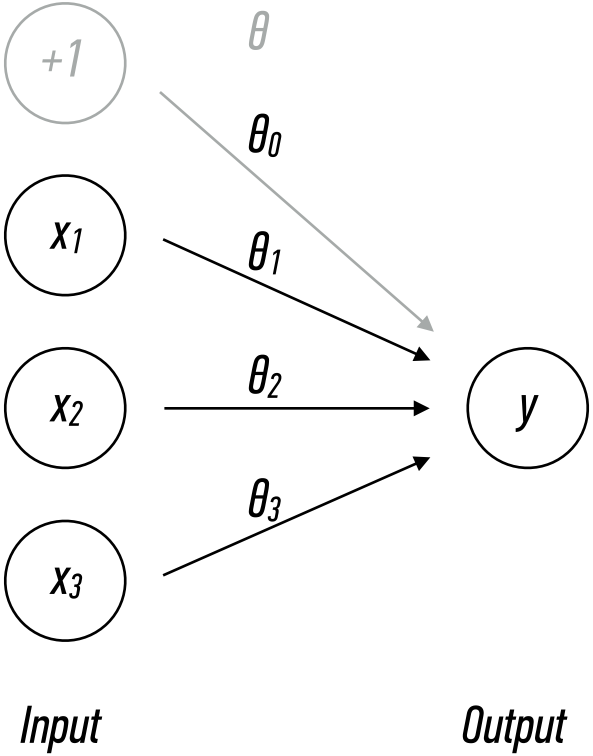
$$y = \frac{1}{1 + e^{-\theta \mathbf{x}}}$$

$$\theta = (\theta_0, \theta_1, \theta_2, \theta_3)$$

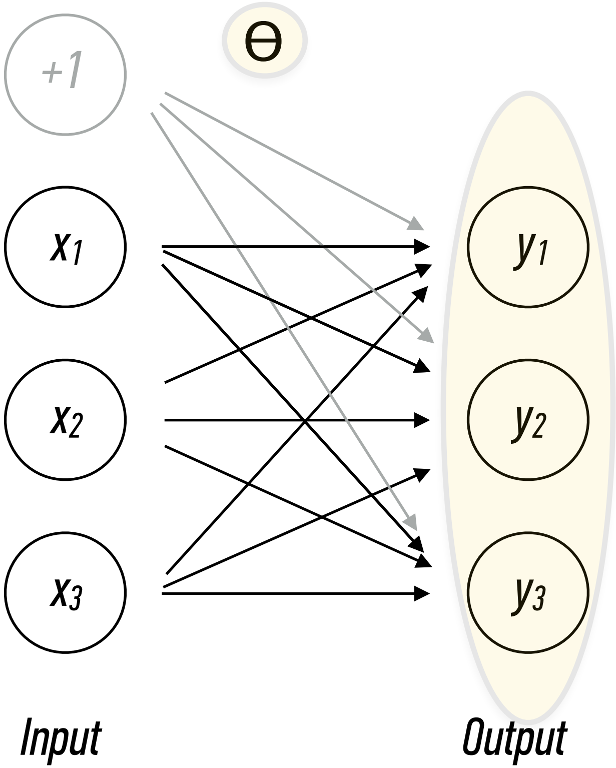


$$y = \sigma(\theta \mathbf{x})$$

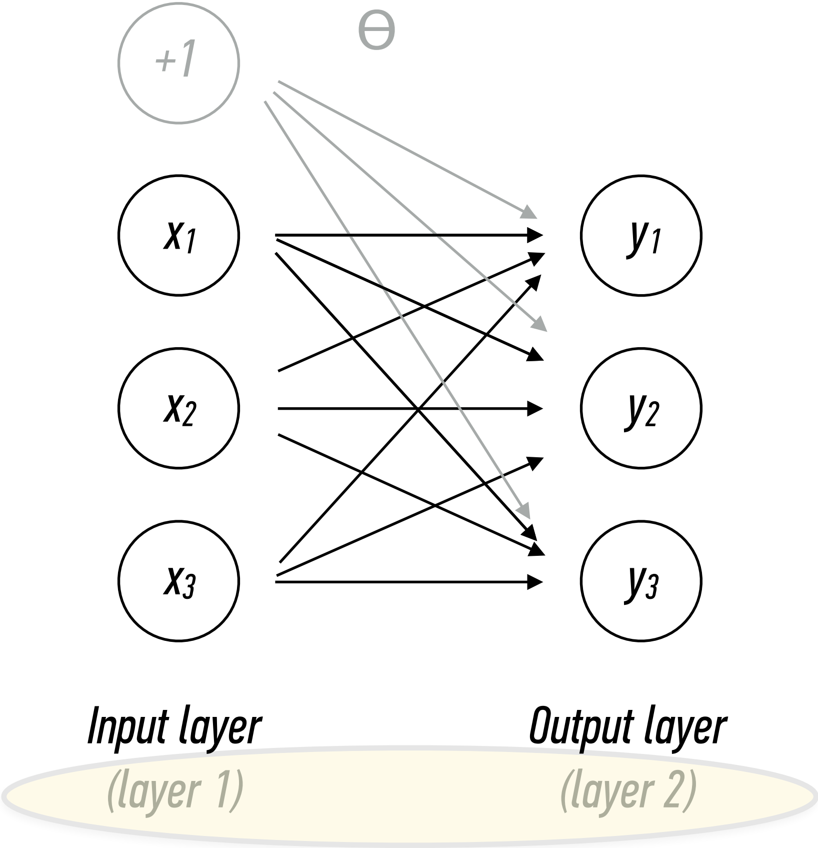
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



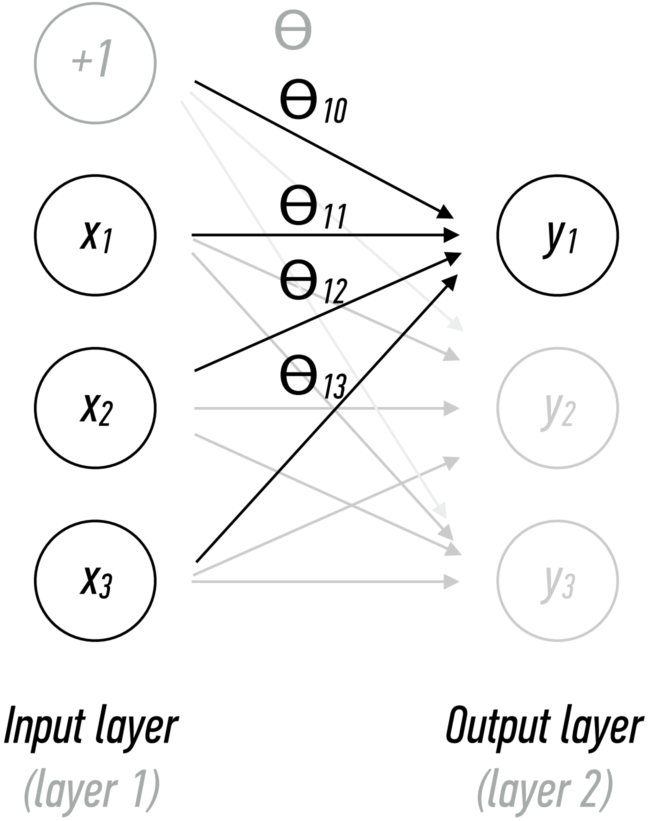
$$y = h_\theta(\mathbf{x})$$



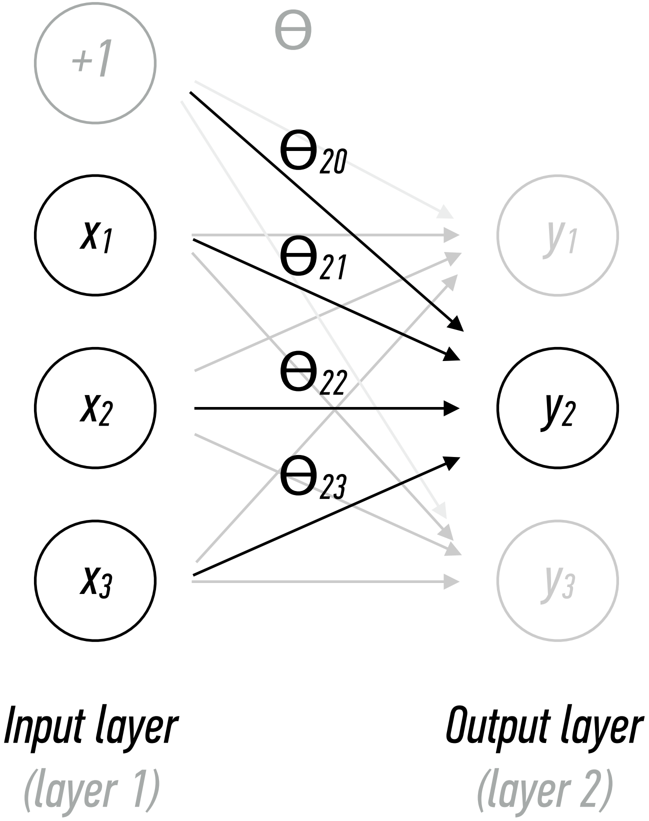
$$\mathbf{y} = h_{\Theta}(\mathbf{x})$$



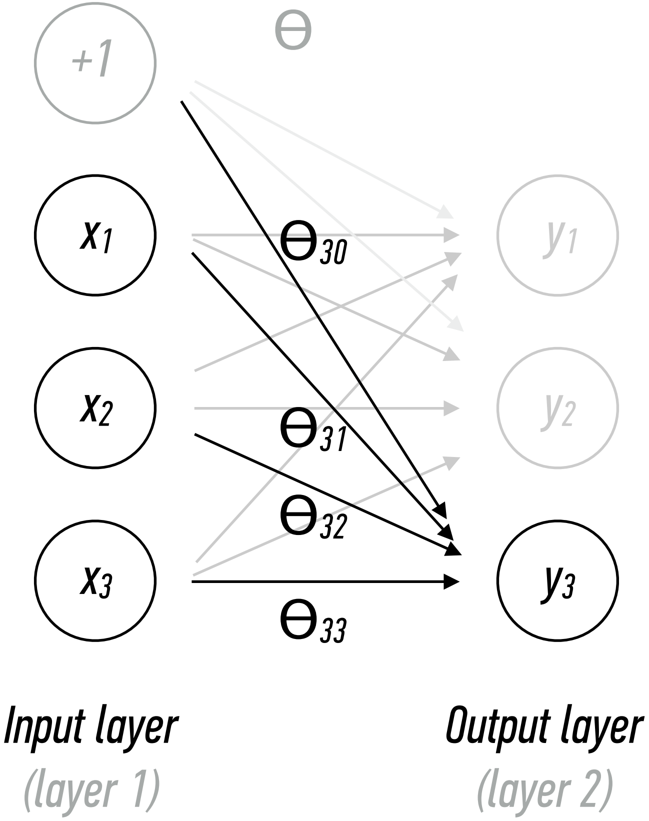
$$\mathbf{y} = h_\Theta(\mathbf{x})$$



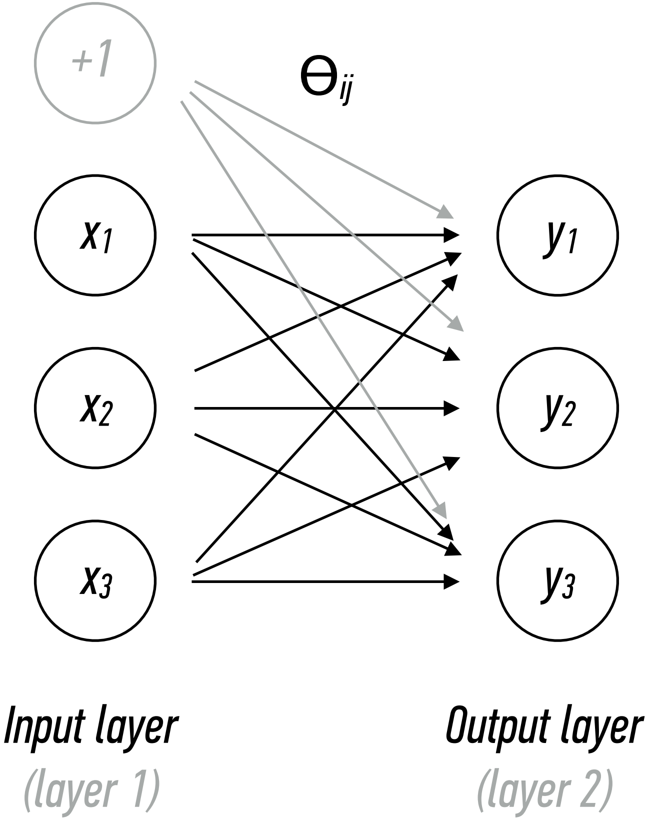
$$\mathbf{y} = h_{\Theta}(\mathbf{x})$$

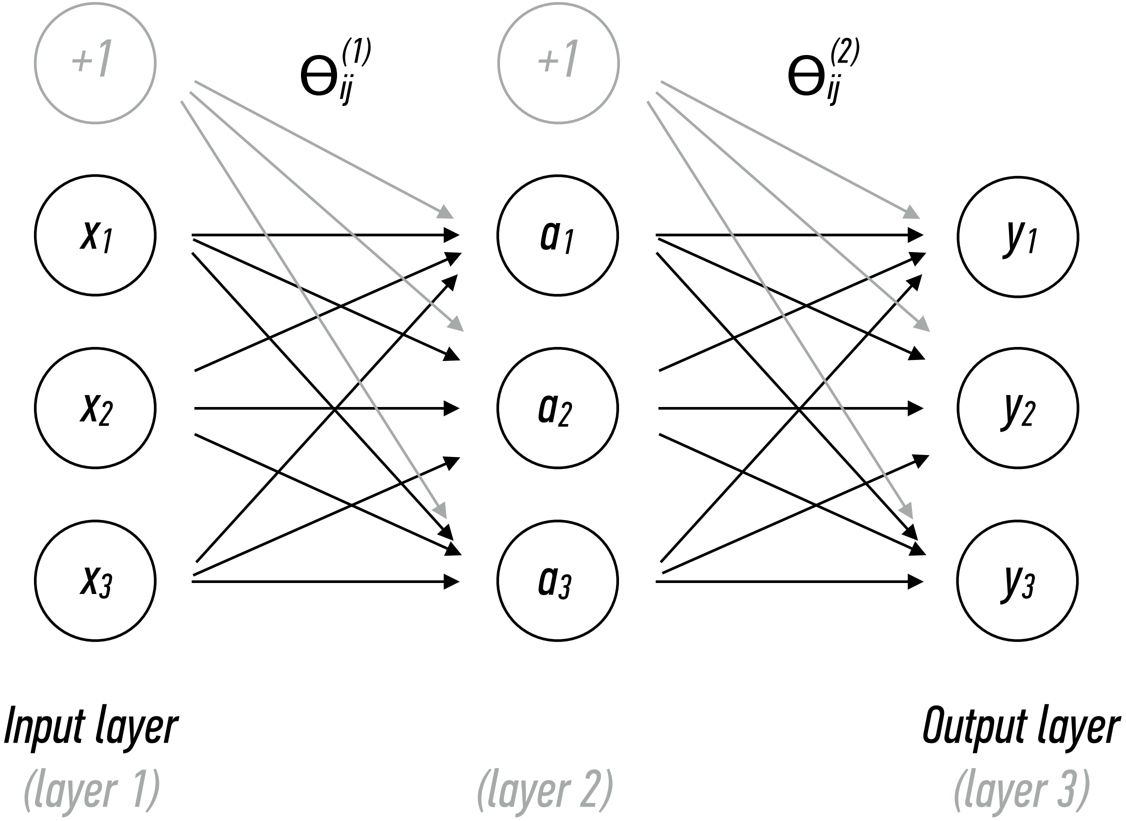


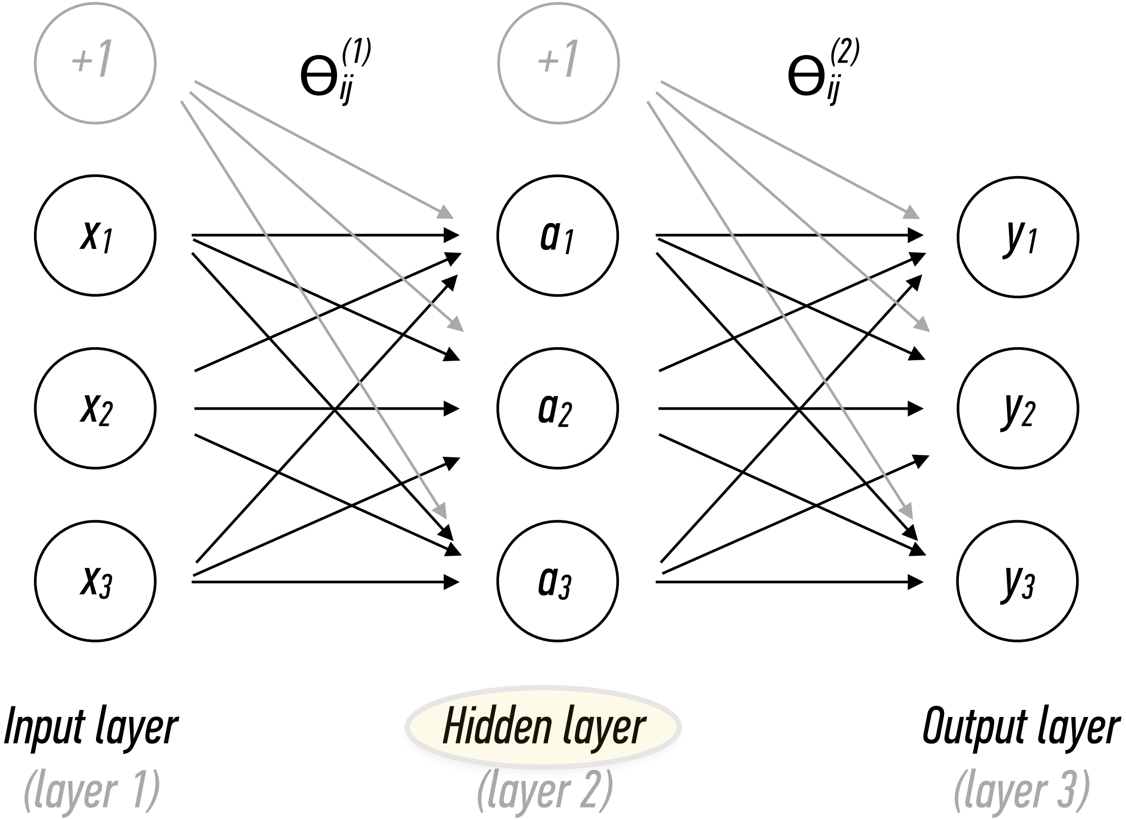
$$\mathbf{y} = h_\Theta(\mathbf{x})$$

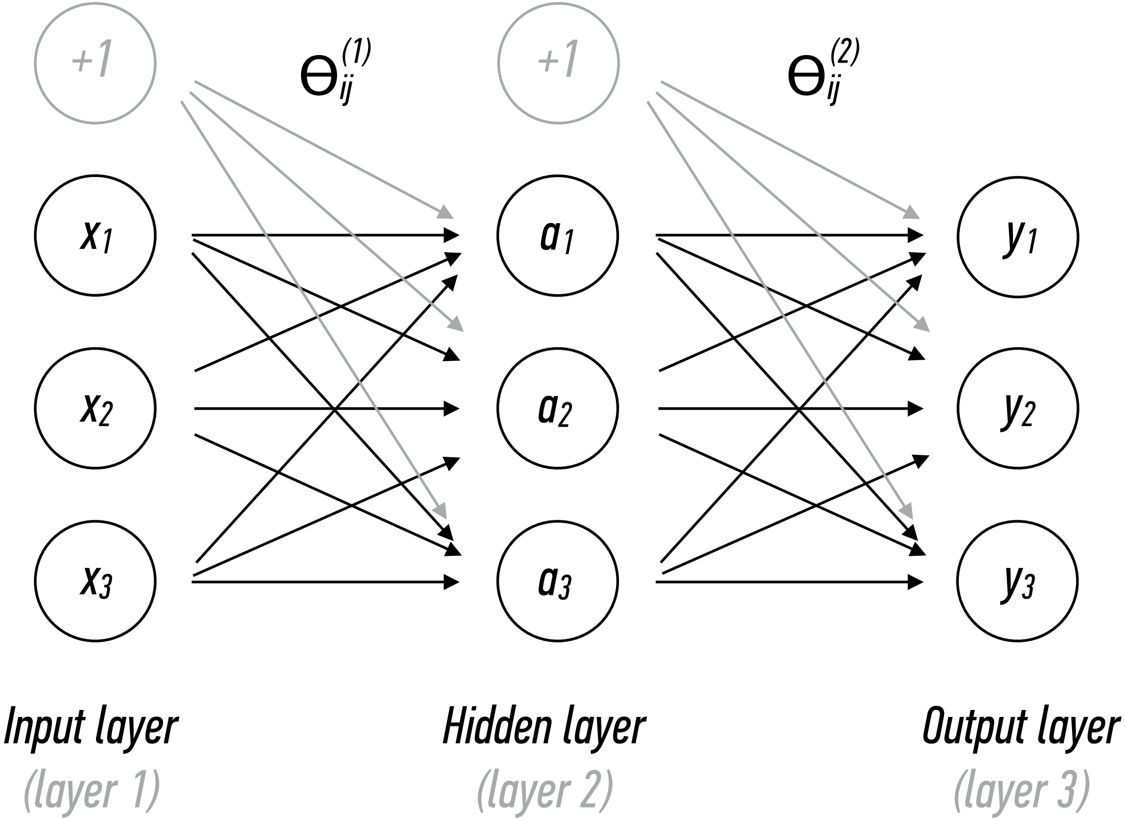


$$\mathbf{y} = h_{\Theta}(\mathbf{x})$$


+1
Θ_{ij}
x_1
x_2
x_3
y_1
y_2
y_3
Input layer
(layer 1)
Output layer
(layer 2)


+1
$\Theta_{ij}^{(1)}$
+1
$\Theta_{ij}^{(2)}$
x_1
x_2
x_3
a_1
a_2
a_3
y_1
y_2
y_3
Input layer
(layer 1)
(layer 2)
Output layer
(layer 3)

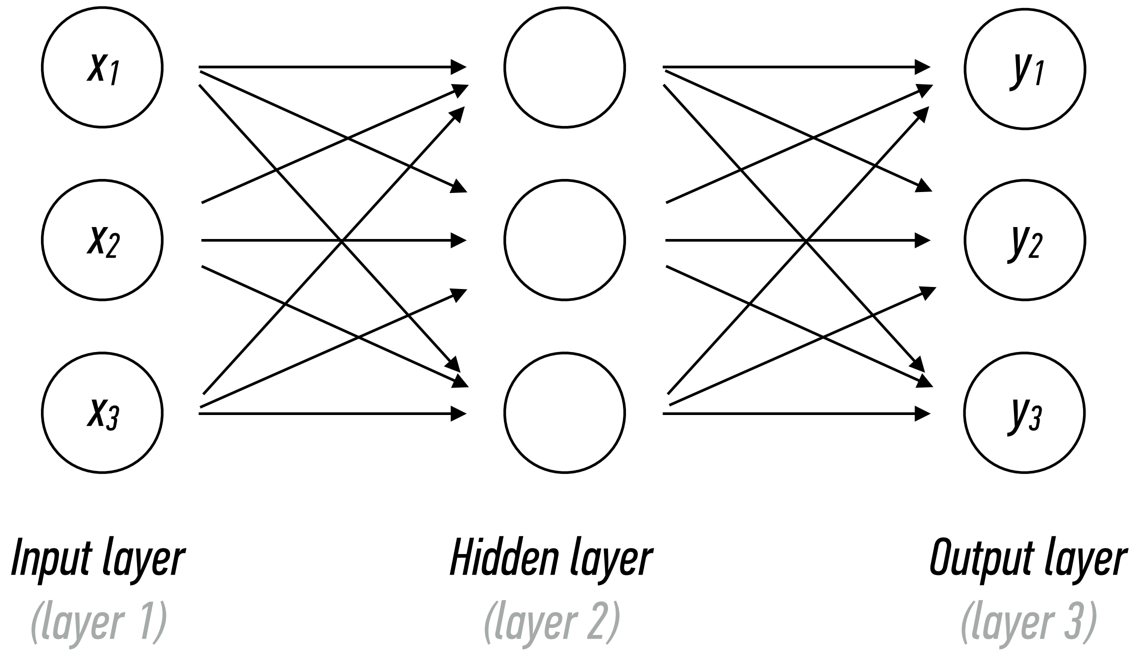

+1
$\Theta_{ij}^{(1)}$
+1
$\Theta_{ij}^{(2)}$
x_1
x_2
x_3
a_1
a_2
a_3
y_1
y_2
y_3
Input layer
(layer 1)
Hidden layer
(layer 2)
Output layer
(layer 3)



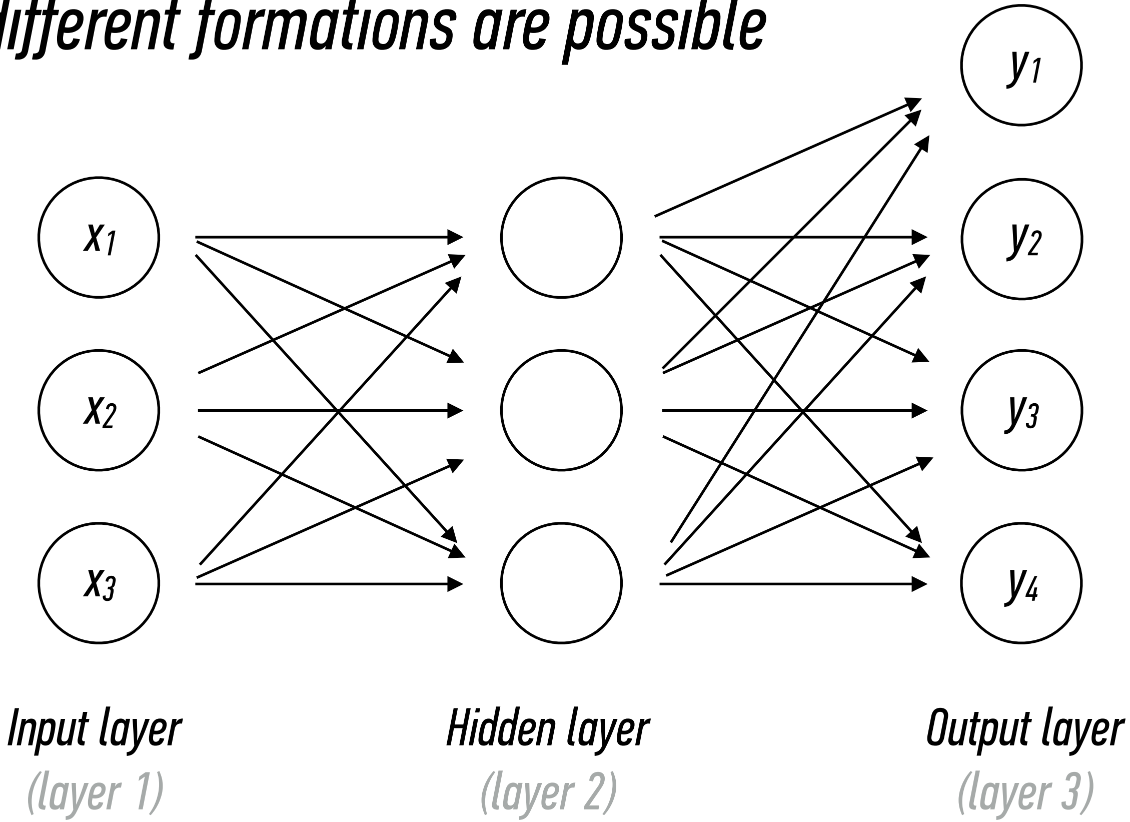
NOTE

Each layer is represented by a matrix Θ_{ij}

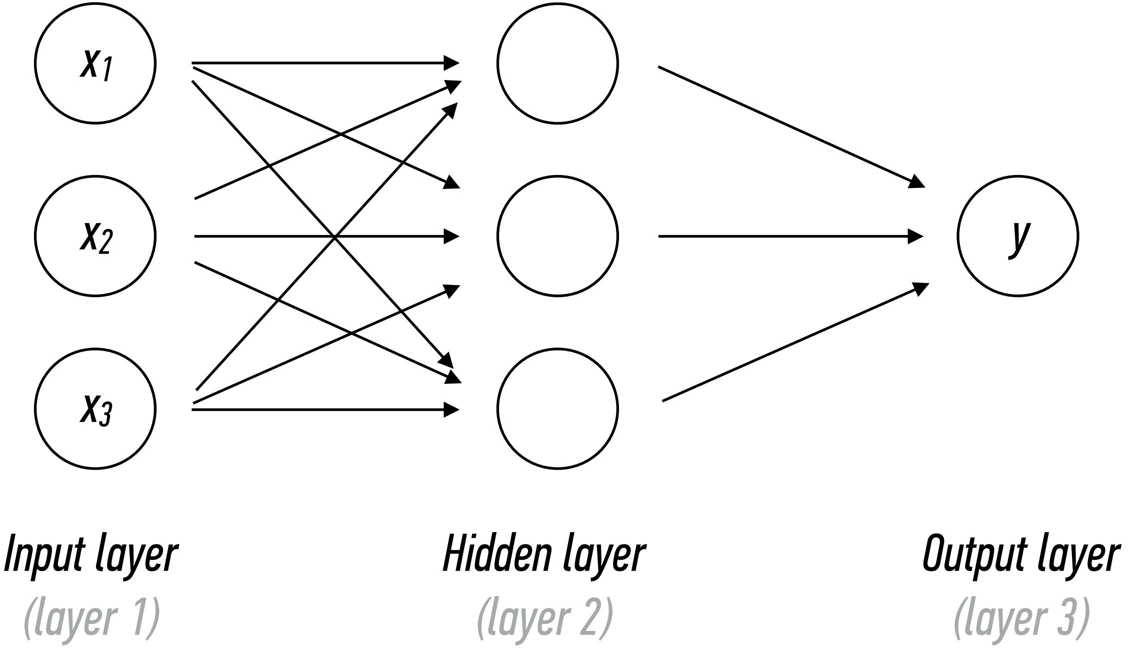
Obviously, different formations are possible

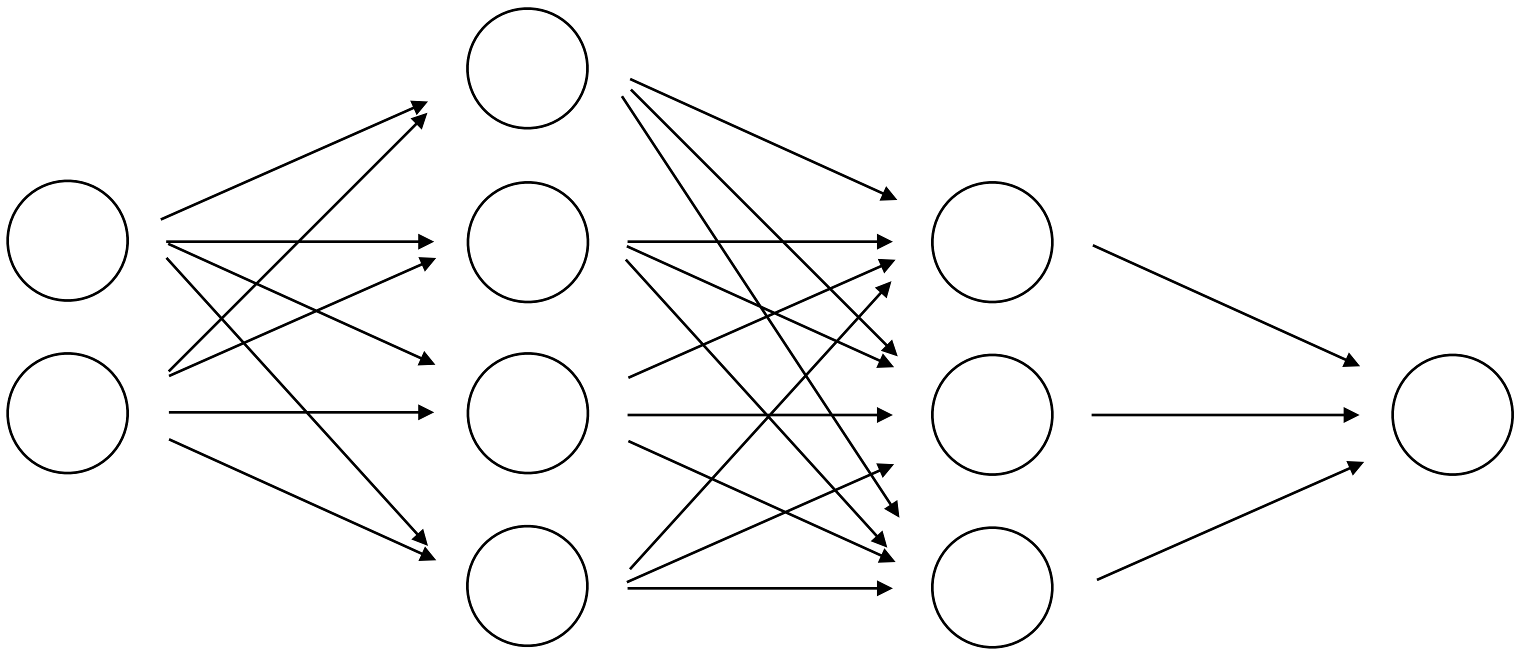


Obviously, different formations are possible



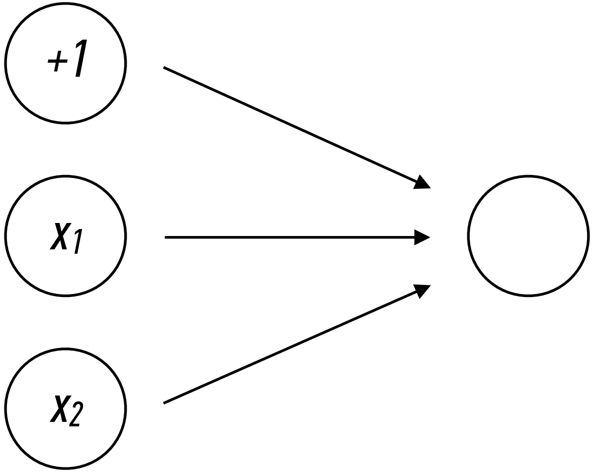
Obviously, different formations are possible

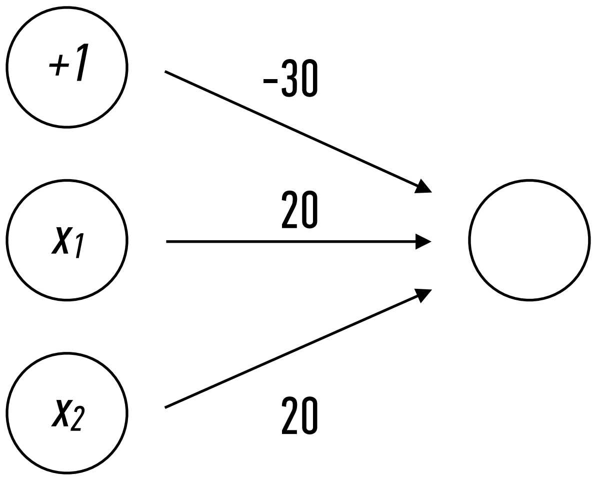


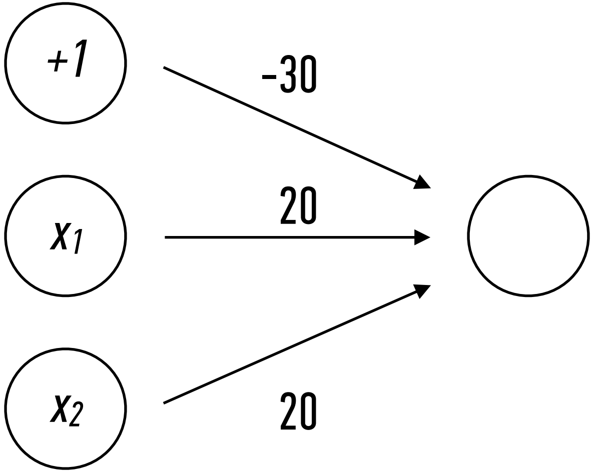


Input layer (layer 1) | *Hidden layer* (layer 2) | *Hidden layer* (layer 3) | *Output layer* (layer 4)

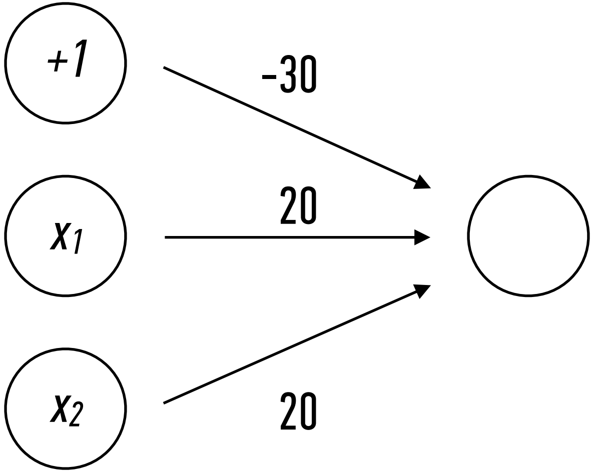
II. LOGICAL OPERATORS


+1
x_1
x_2


+1
x_1
x_2
-30
20
20

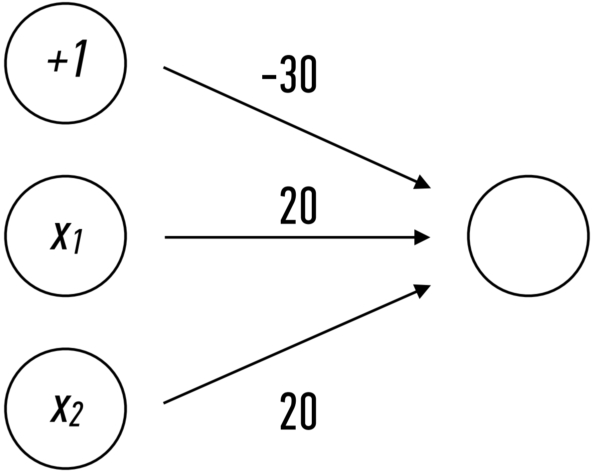


x_1	x_2	
0	0	
0	1	
1	0	
1	1	

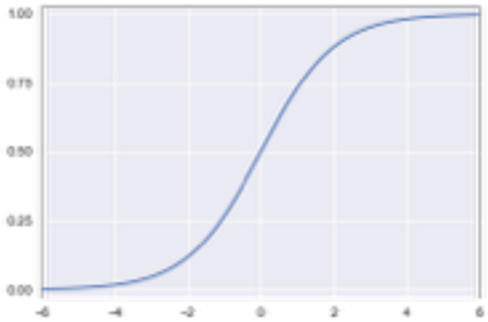


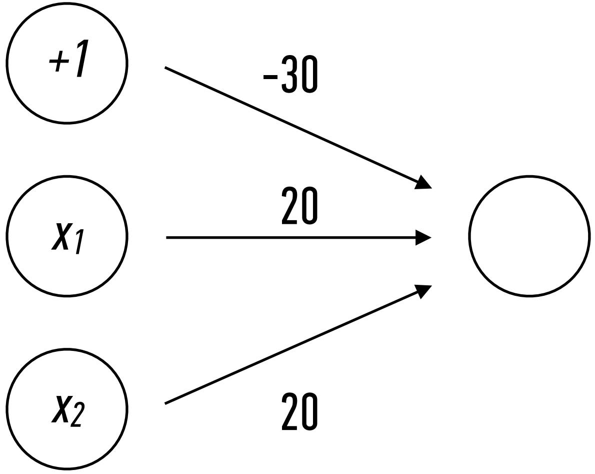
x_1	x_2	z
0	0	−30 + 0 + 0 = −30
0	1	−30 + 0 + 20 = −10
1	0	−30 + 20 + 0 = −10
1	1	−30 + 20 + 20 = 10

LOGICAL OPERATORS

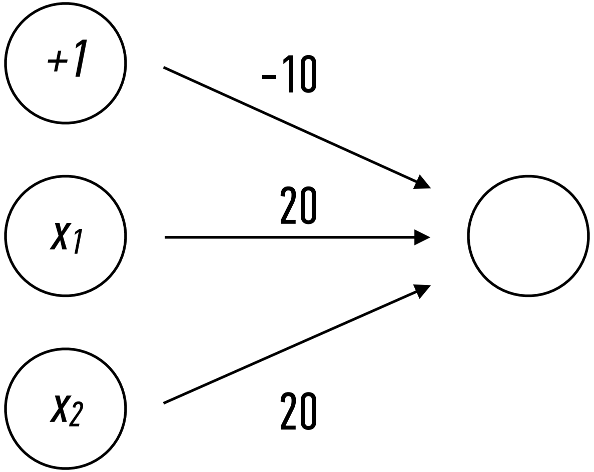


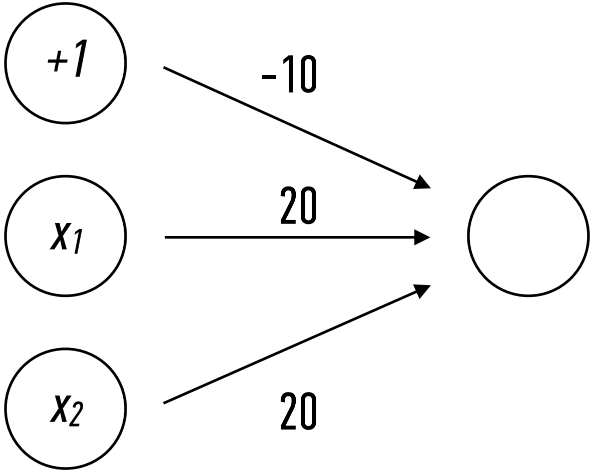
x_1	x_2	z	y
0	0	−30 + 0 + 0 = −30	0
0	1	−30 + 0 + 20 = −10	0
1	0	−30 + 20 + 0 = −10	0
1	1	−30 + 20 + 20 = 10	1



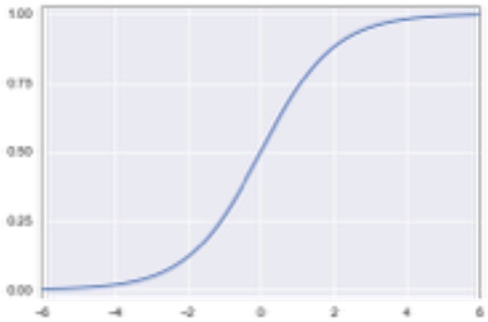


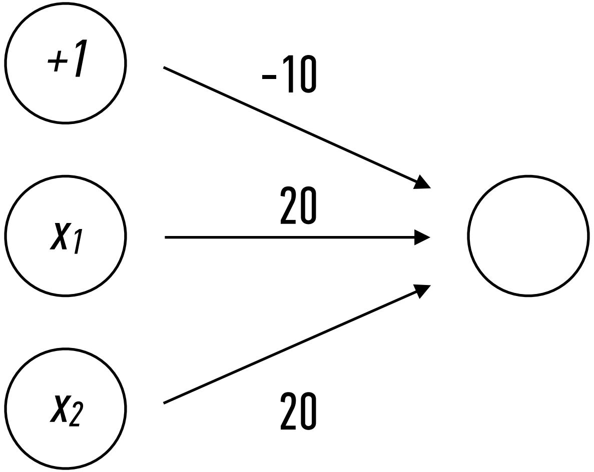
x_1	x_2		y
0	0		0
0	1	x_1 AND x_2	0
1	0		0
1	1		1


+1
-10
x_1
20
x_2
20

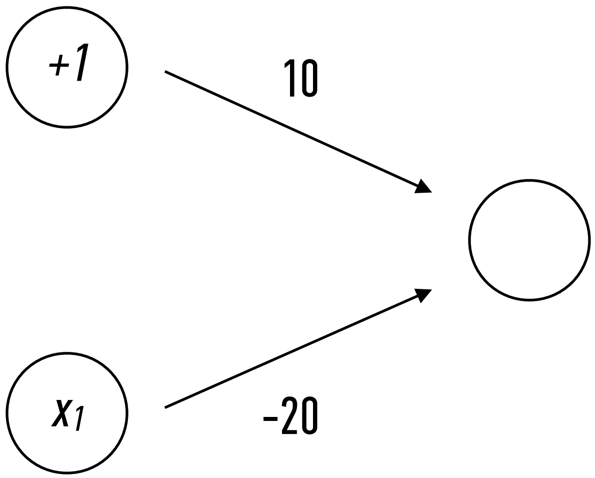


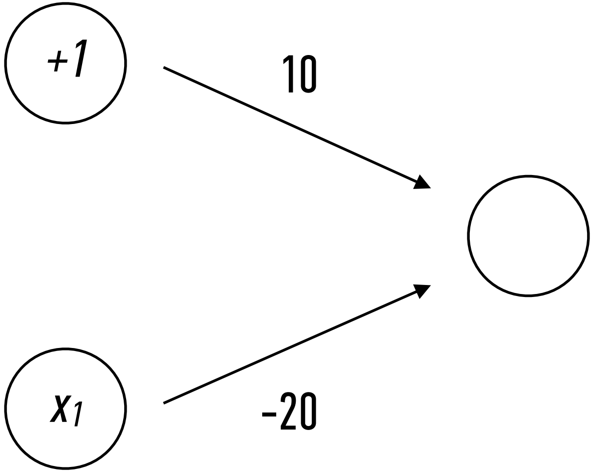
x_1	x_2	z	y
0	0	$-10 + 0 + 0 = -10$	0
0	1	$-10 + 0 + 20 = 10$	1
1	0	$-10 + 20 + 0 = 10$	1
1	1	$-10 + 20 + 20 = 10$	1



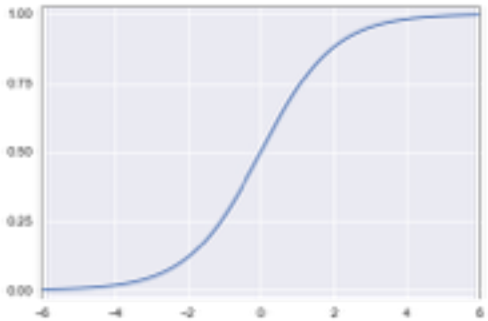


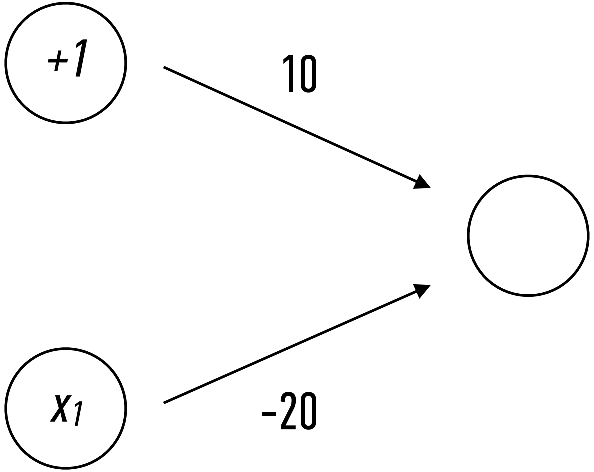
x_1	x_2		y
0	0		0
0	1		1
1	0	x_1 OR x_2	1
1	1		1


+1
10
x_1
-20



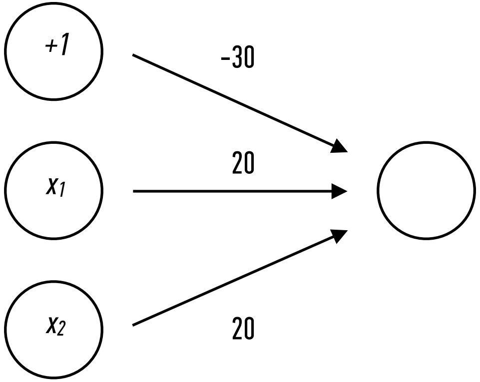
x_1	z	y
0	10 + 0 = 10	1
1	10 - 20 = -10	0



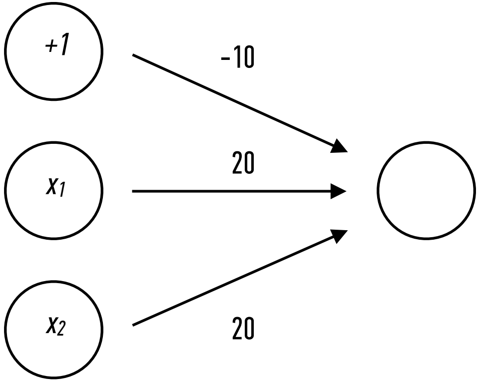


x_1		y
0	**NOT x_1**	1
1		0

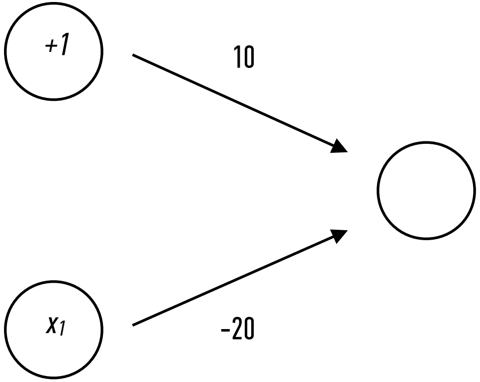
x_1 AND x_2



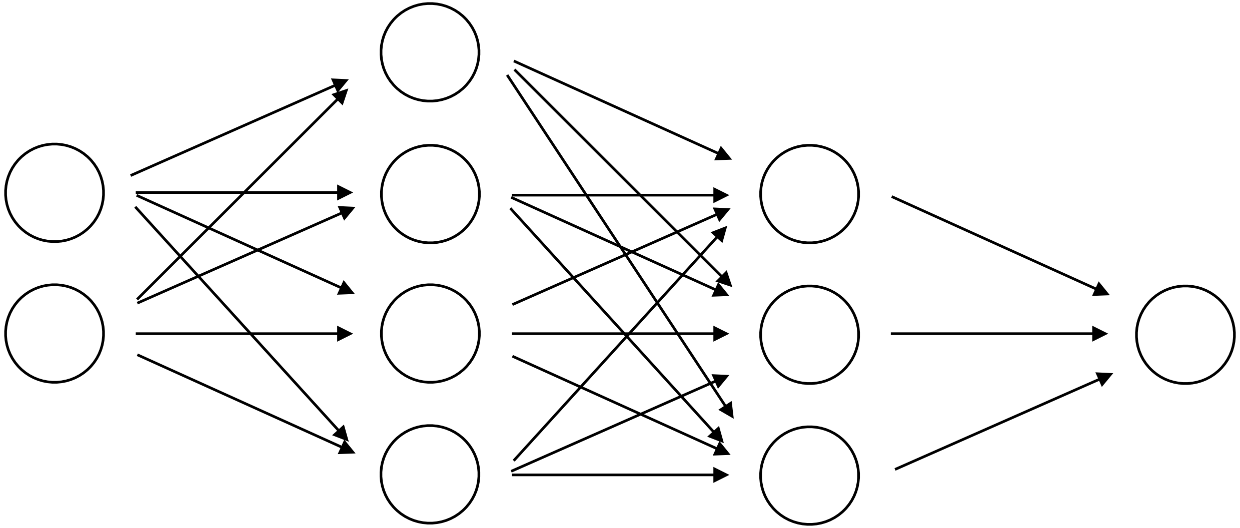
x_1 OR x_2



NOT x_1

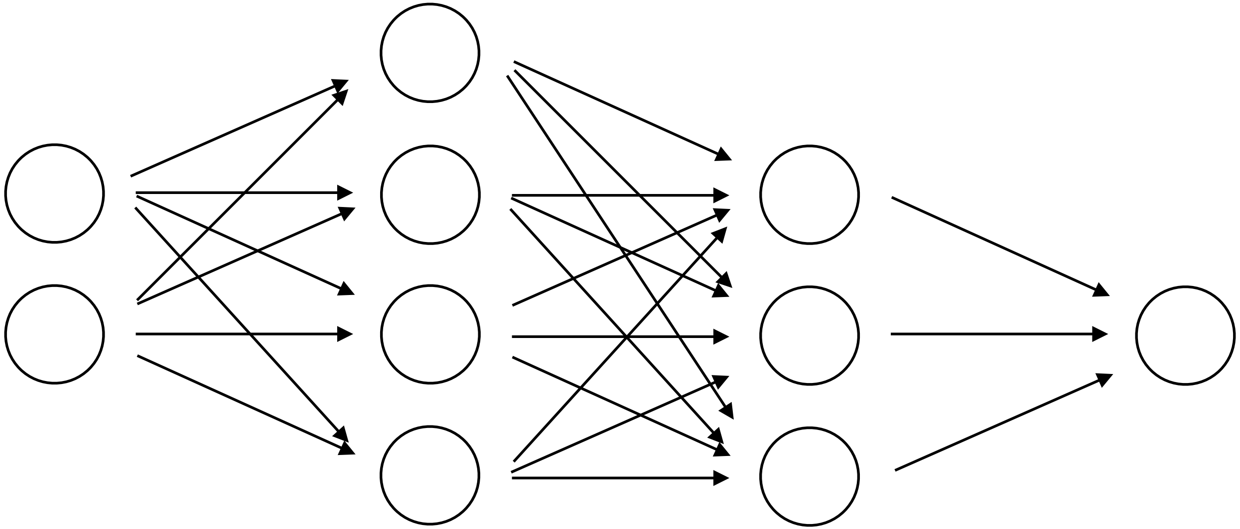


By chaining these basic logical operators, you can create complex logical structures in your model

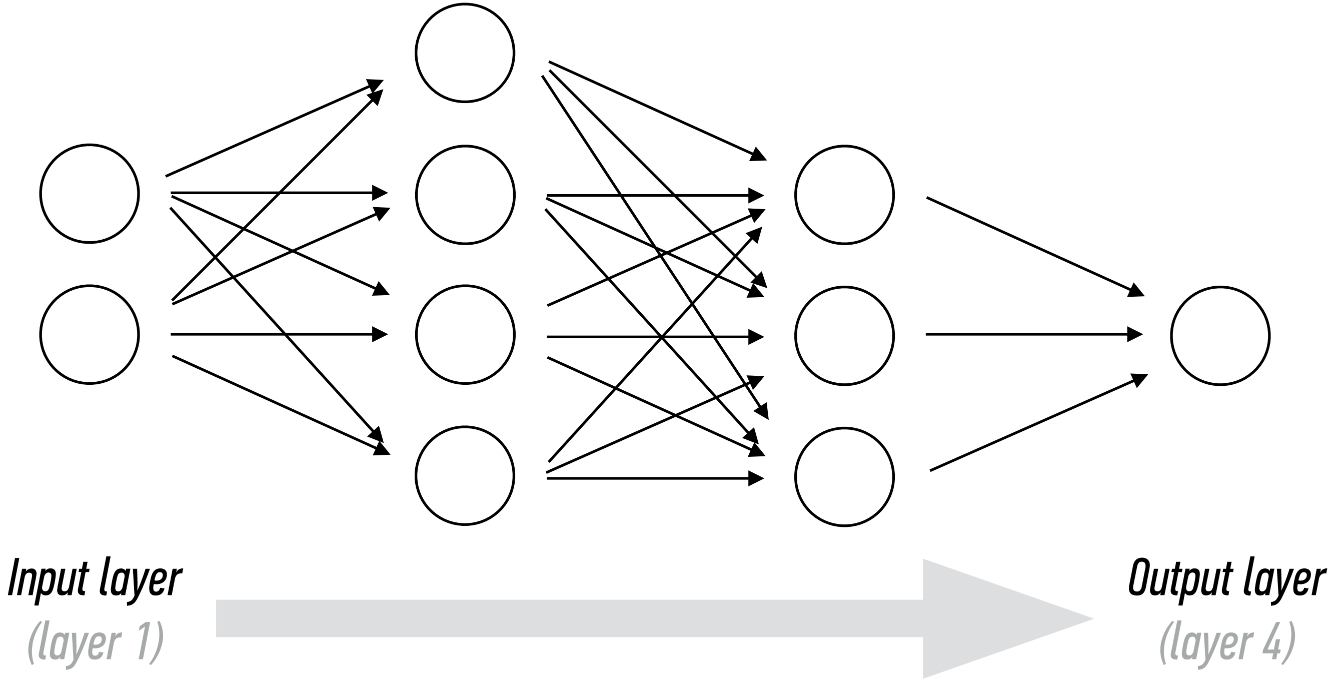


III. NEURAL NETWORK VARIETIES

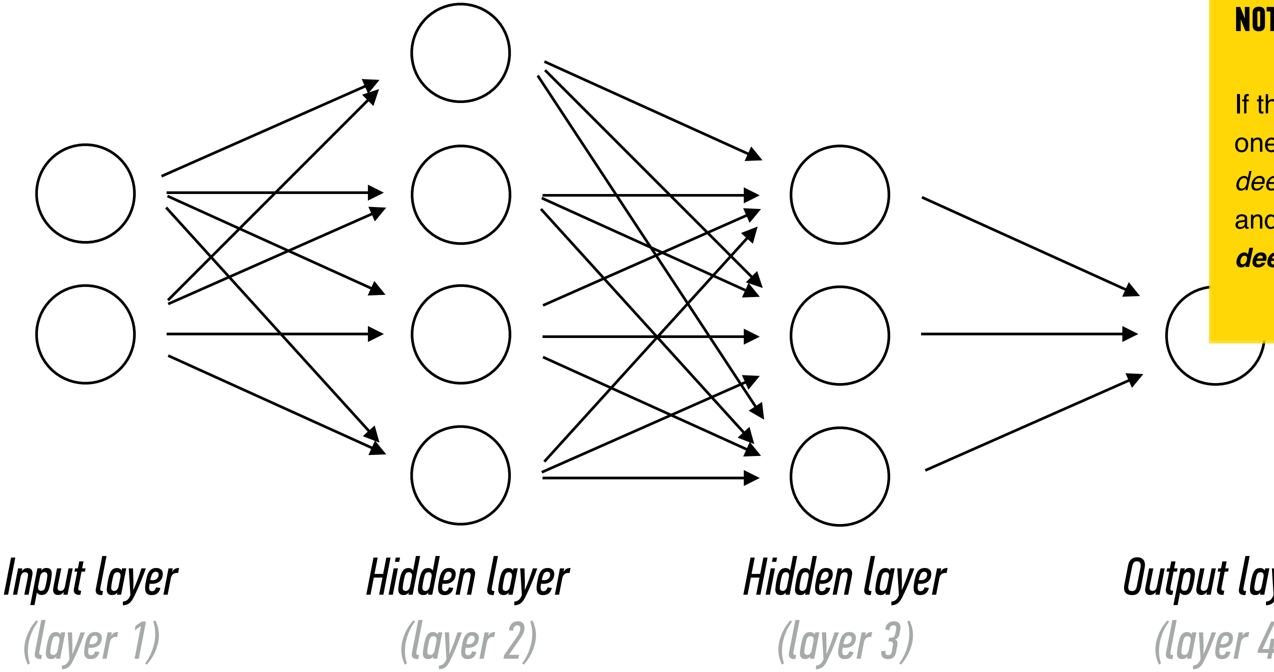
The aforementioned example was a **feed-forward** *neural network*



The aforementioned example was a ***feed-forward*** *neural network*



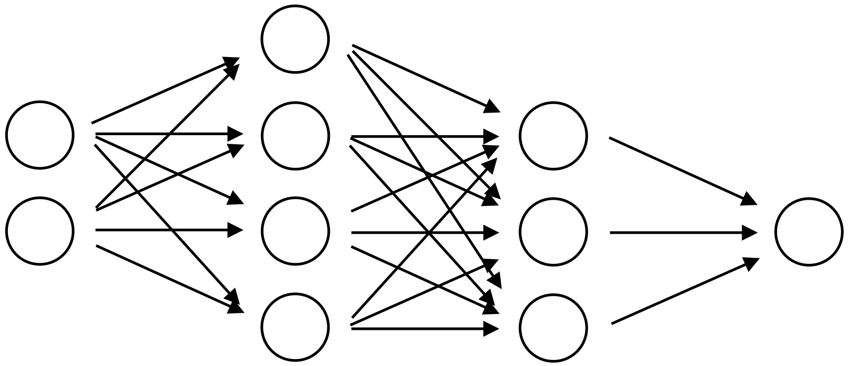
*The aforementioned example was a **feed-forward** neural network*



NOTE

If there is more than one layer, this is a *deep neural network*, and we speak of ***deep learning***.

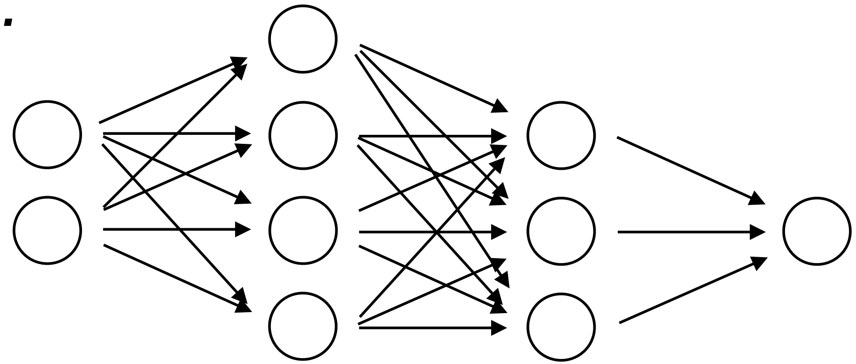
Each node in a neural network is a non-linear function of the nodes in the layer below, which is called an **activation function**.



Each node in a neural network is a non-linear function of the nodes in the layer below, which is called an **activation function***.*

Instead of a logistic regression, one could use other (non-linear) functions, such as the **perceptron***.*

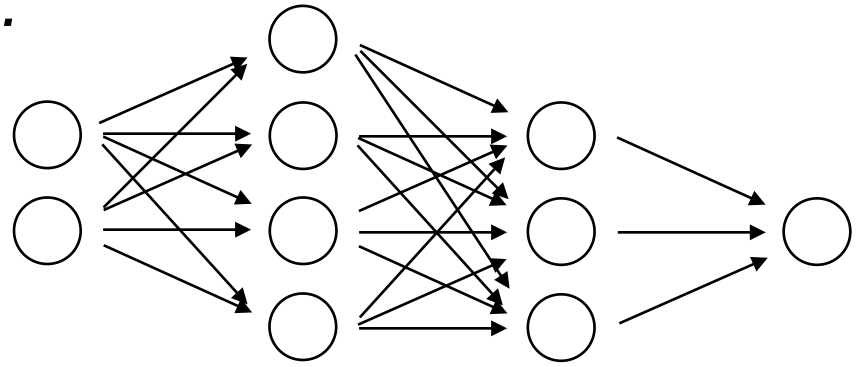
$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$



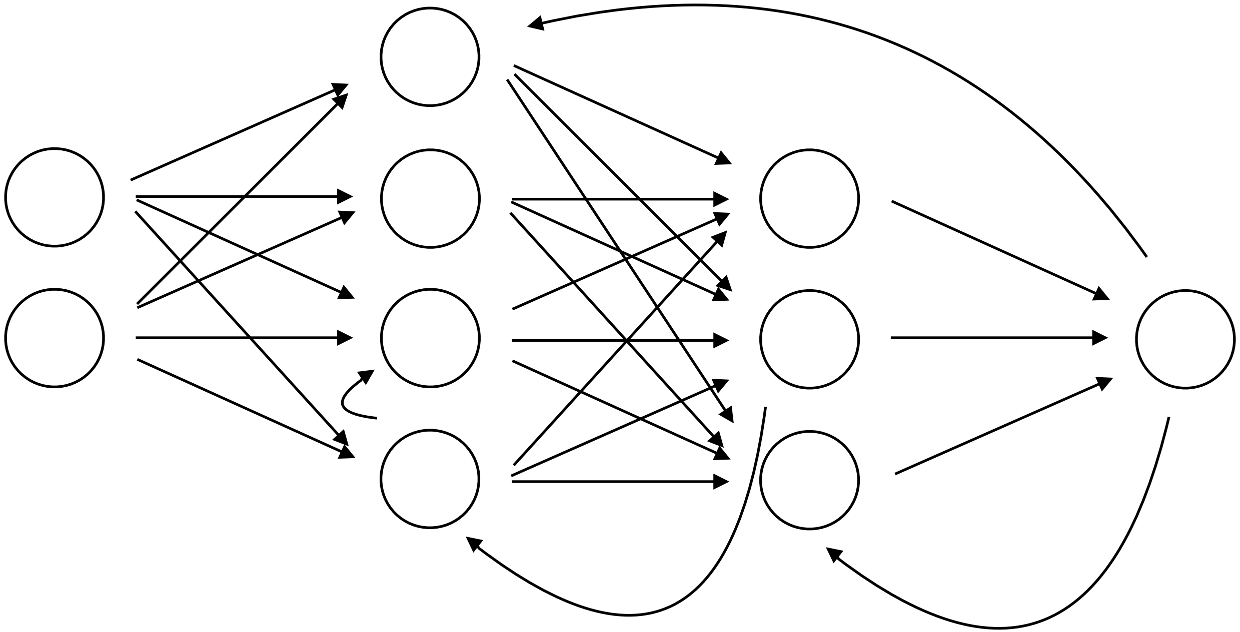
Each node in a neural network is a non-linear function of the nodes in the layer below, which is called an **activation function**.

Instead of a logistic regression, one could use other (non-linear) functions, such as the **perceptron**.

Such a network is called a **multilayer perceptron** *(MLP).*

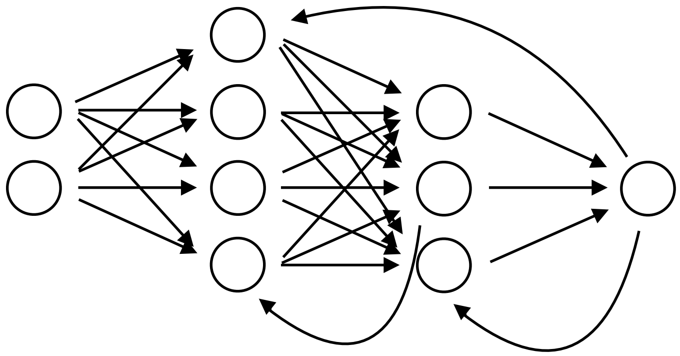


A **recurrent network** *has directed cycles in its connection graph*



A **recurrent network** *has directed cycles in its connection graph*

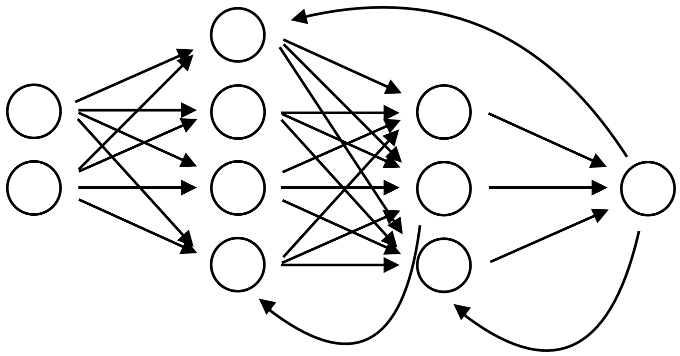
These networks are biologically more realistic, but are difficult to train



A **recurrent network** *has directed cycles in its connection graph*

These networks are biologically more realistic, but are difficult to train

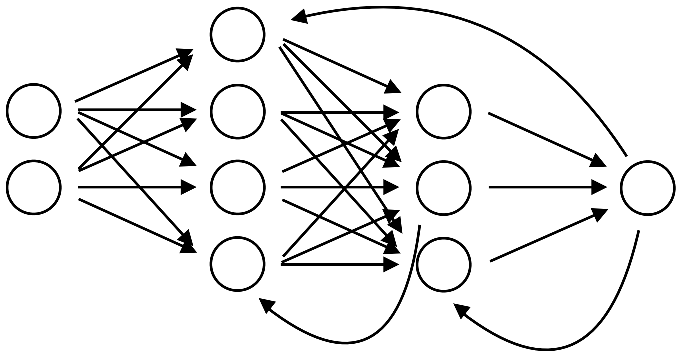
Note that you cannot speak of multiple hidden layers anymore, since we lost the ordering of layers.



A **recurrent network** *has directed cycles in its connection graph*

These networks are biologically more realistic, but are difficult to train

Note that you cannot speak of multiple hidden layers anymore, since we lost the ordering of layers. The "multiple layers" you might see are a special case of missing hidden-hidden connections.



NEURAL NETWORKS – EXAMPLE

Example of a recurrent network (Ilya Sutskever, 2011)

- *Trained on half a billion characters from Wikipedia*
- *Model to predict the next character in a sequence*
  - *generates probability distribution for the next character*
  - *samples a character from this distribution*
  - *repeats process given updated text*

Source: Geoffrey Hinton, *Neural Networks for Machine Learning*, coursera.org

NEURAL NETWORKS – EXAMPLE

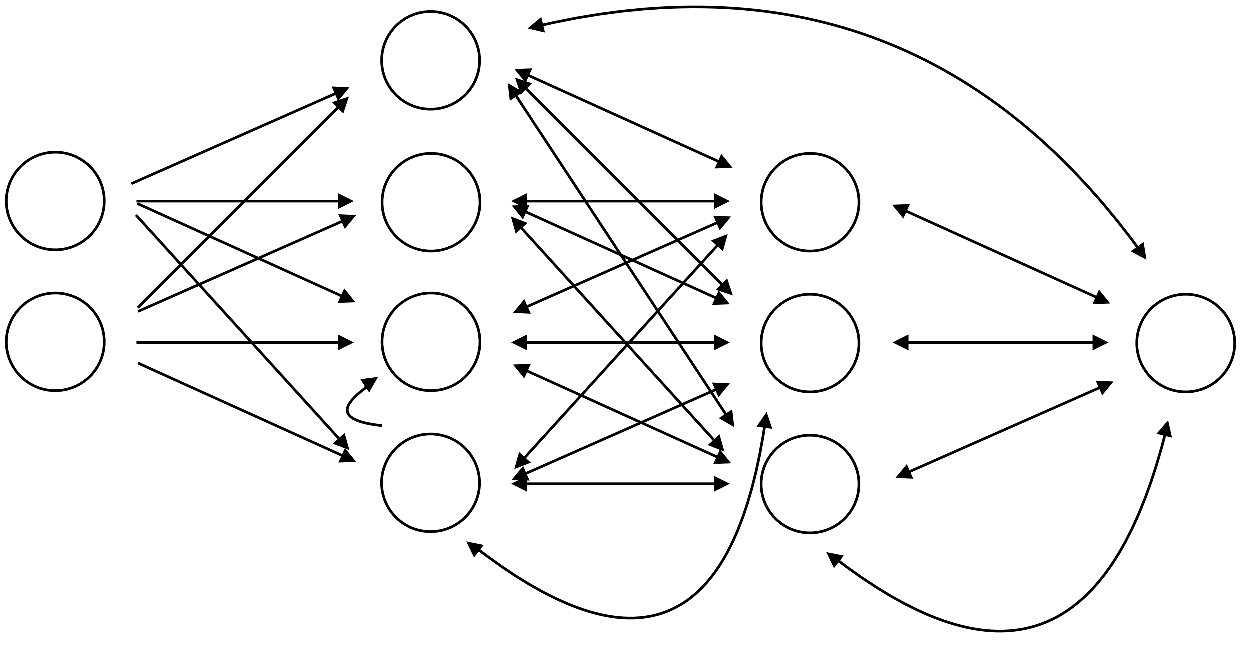
Example of a recurrent network (Ilya Sutskever, 2011)

In 1974 Northern Denver had been overshadowed by CNL, and several Irish intelligence agencies in the Mediterranean region. However, on the Victoria, Kings Hebrew stated that Charles decided to escape during an alliance. The mansion house was completed in 1882, the second in its bridge are omitted, while closing is the proton reticulum composed below it aims, such that it is the blurring of appearing on any well-paid type of box printer.

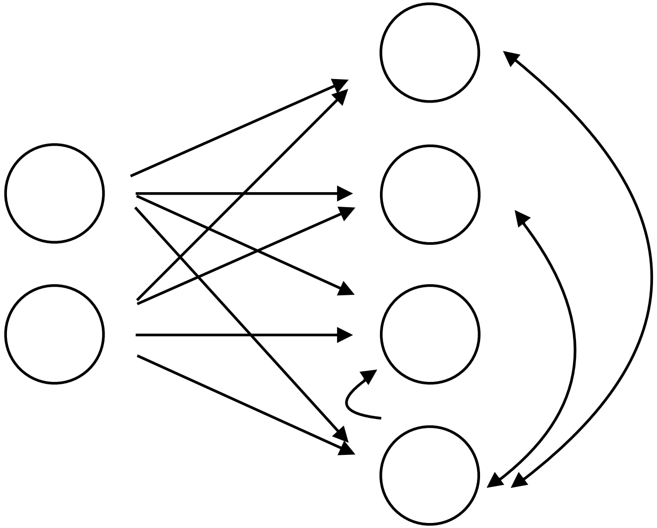
By: Sutskever's recurrent network, one character at a time

Source: Geoffrey Hinton, *Neural Networks for Machine Learning*, coursera.org

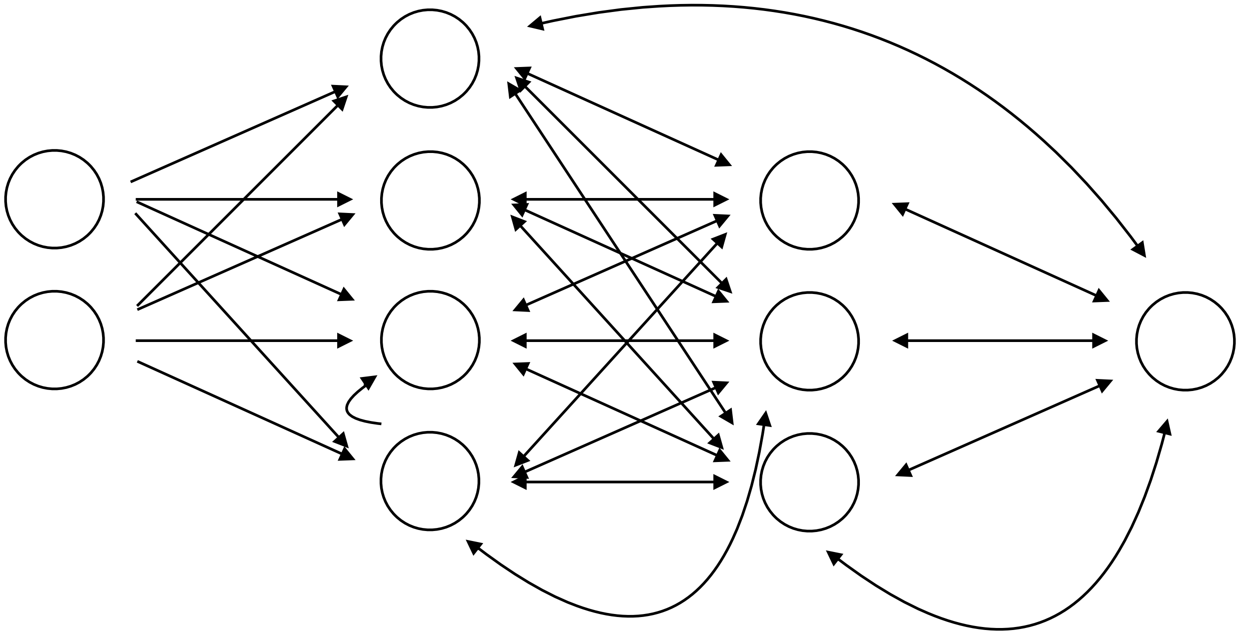
A **symmetrical connected network** *is a recurrent network with equal weights in both directions*



A symmetrical connected network with no hidden units is called a **Hopfield net**

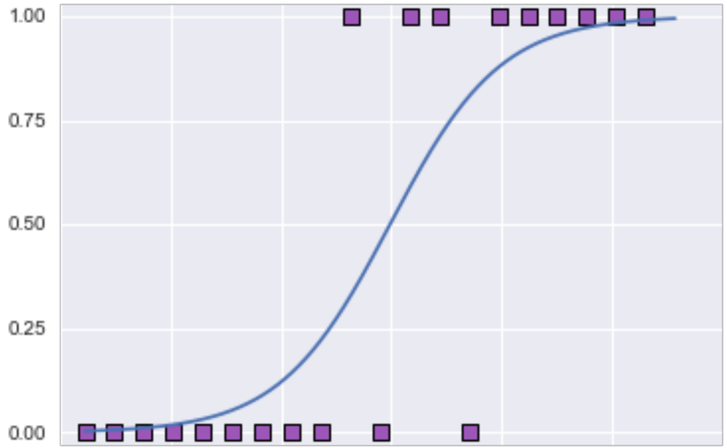


A symmetrical connected network with hidden units is called a **Boltzmann machine**



IV. COST FUNCTION

Let's go back to the logistic regression



$$y = h_\theta(\mathbf{x}) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)}}$$

Recall the cost function of the logistic regression

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

Recall the cost function of the logistic regression

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

m is the number of samples

Recall the cost function of the logistic regression

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

m is the number of samples

regularization term

Recall the cost function of the logistic regression

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

m is the number of samples *cost per observation* *regularization term*

Recall the cost function of the logistic regression

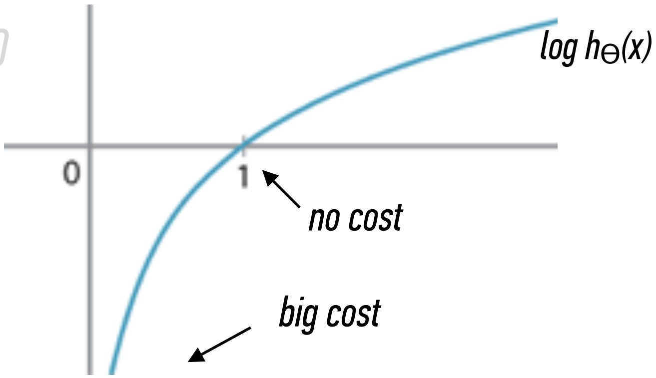
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} \underbrace{y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))}\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$= \begin{cases} \log h_\theta(x^{(i)}) & \text{if } y^{(i)} = 1 \\ \log(1-h_\theta(x^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

Recall the cost function of the logistic regression

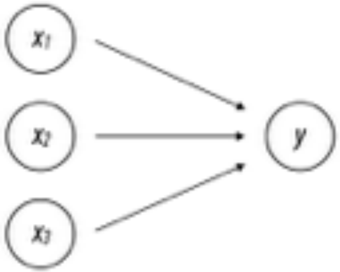
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} \underbrace{y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))}\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$= \begin{cases} \log h_\theta(x^{(i)}) & \text{if } y^{(i)} = 1 \\ \log(1-h_\theta(x^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$



So we now understand logistic regression...

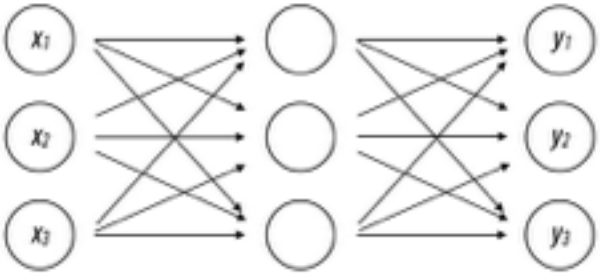
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$



The neural net cost function looks very similar

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

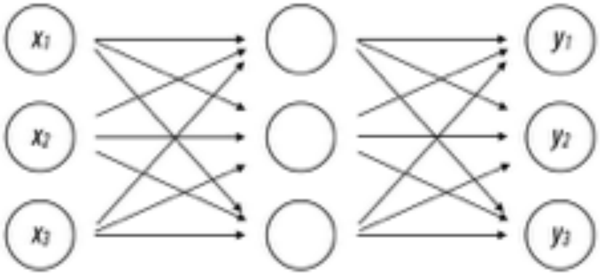
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$



The neural net cost function looks very similar

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

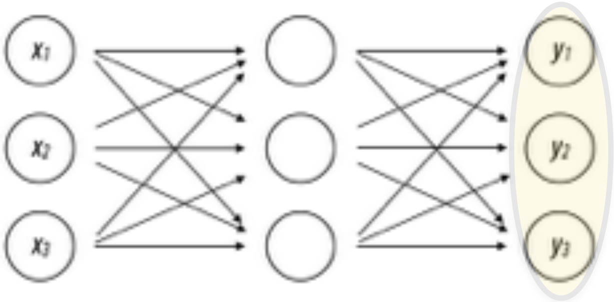
Cost J depends on collection of matrices $\Theta_{ij}^{(l)}$ for all activation functions in each layer



The neural net cost function looks very similar

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

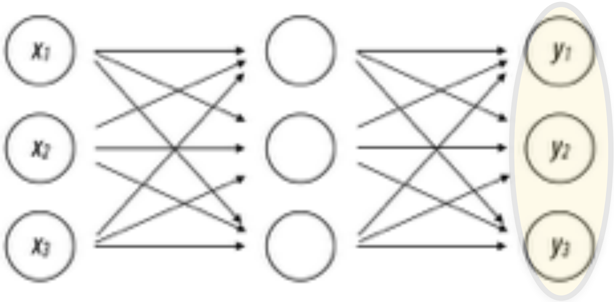
K *is the number of output variables:* $y_1, y_2, \ldots, y_K$



The neural net cost function looks very similar

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

K is the number of output variables: $y_1, y_2, \ldots, y_K$
(You can see this as K different classes)



For example, for $K = 3$,

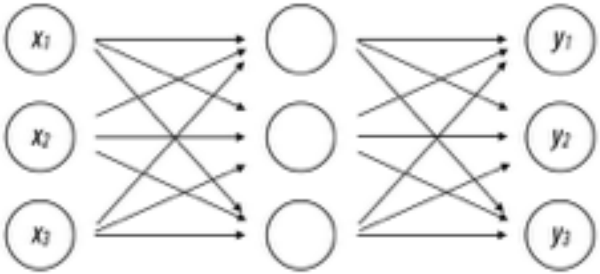
$\begin{pmatrix}1\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\end{pmatrix}$
Car	Pedestrian	Bike

The neural net cost function looks very similar

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

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

Regularization term now includes all coefficients $\Theta_{ij}^{(l)}$ for all activation functions in all L-1 layers

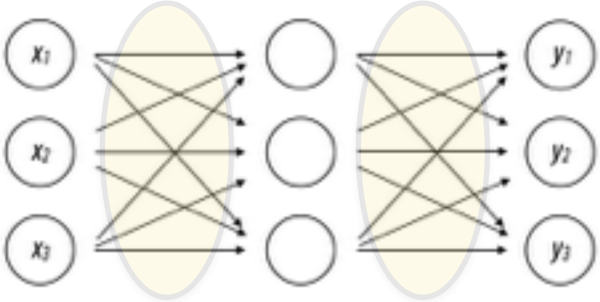


COST FUNCTION

The neural net cost function looks very similar

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

Note that all activation functions in a given layer l have s_l input variables and s_{l+1} output variables



III. BACKPROPAGATION

(MINIMIZING COSTS)

We could minimize the cost function using **gradient descent** *as usual, but this is very slow and inefficient*

We could minimize the cost function using **gradient descent** *as usual, but this is very slow and inefficient*

Instead, we use a method called **back propagation**.

We could minimize the cost function using **gradient descent** *as usual, but this is very slow and inefficient*

Instead, we use a method called **back propagation**.

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

Back propagation is used to approximate the **gradient** *of J (i.e., its derivative in each dimension), to speed up the optimization process.*

We will compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ *by computing all contributions to cost for each sample* i *in each different dimension, independently.*

We will compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ *by computing all contributions to cost for each sample* i *in each different dimension, independently.*

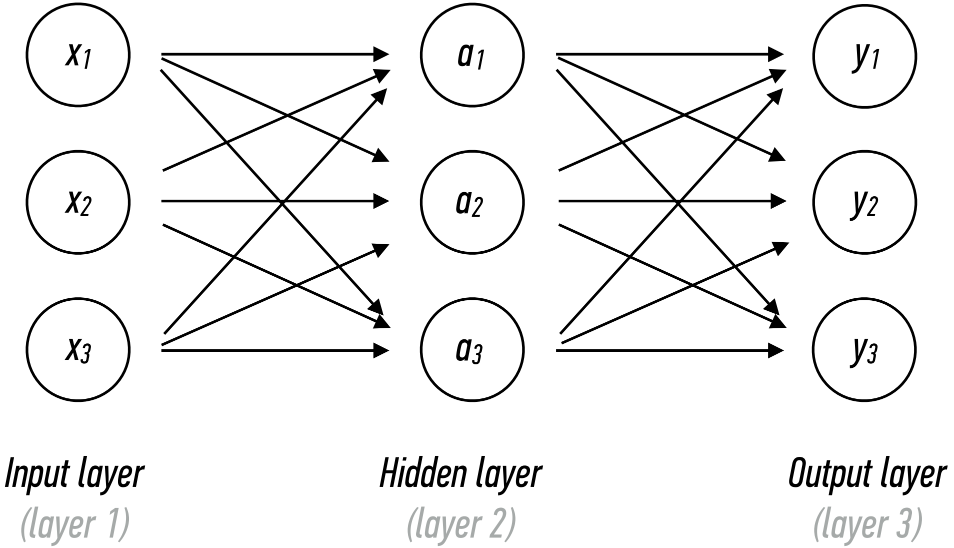
Write cost of sample i = $y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)$

We will compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ *by computing all contributions to cost for each sample* i *in each different dimension, independently.*

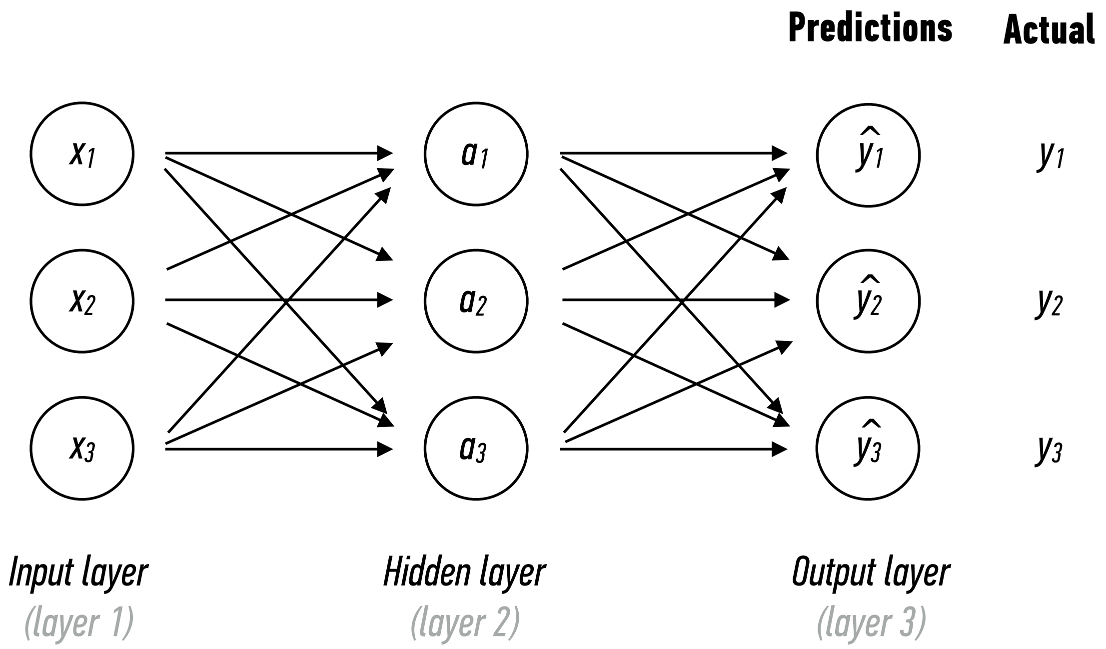
Write cost of sample i = $y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)$

Then we want to compute $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \mathrm{cost(i)}$ *for the error for sample* $(x^{(i)}, y^{(i)})$ *along dimension* z_j *in in layer* l

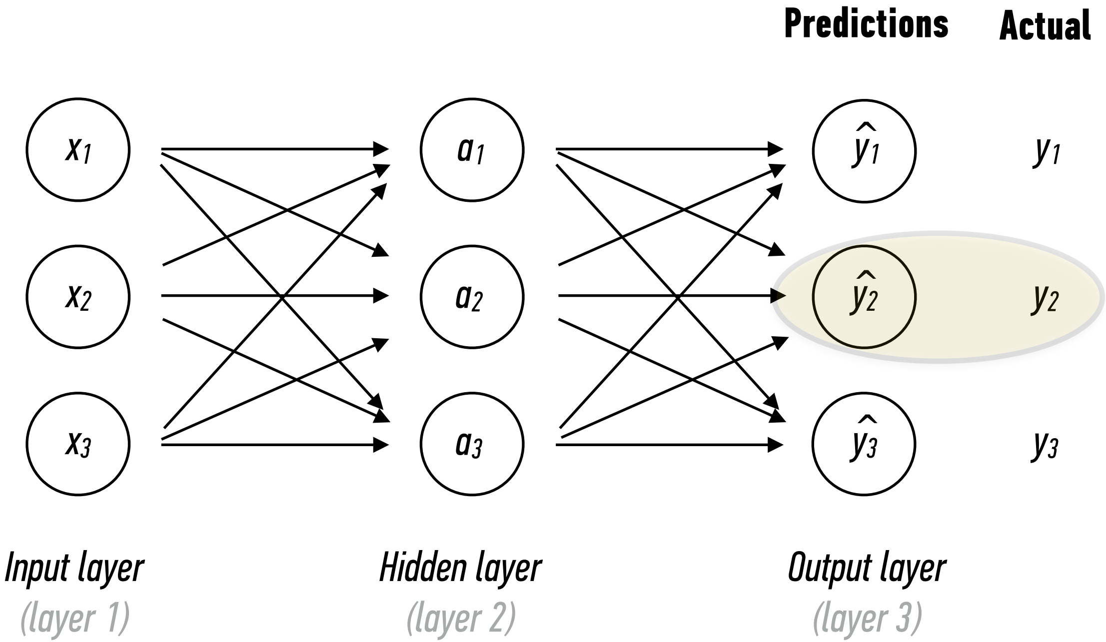
Given some neural network



We compare our predictions with the actual labels

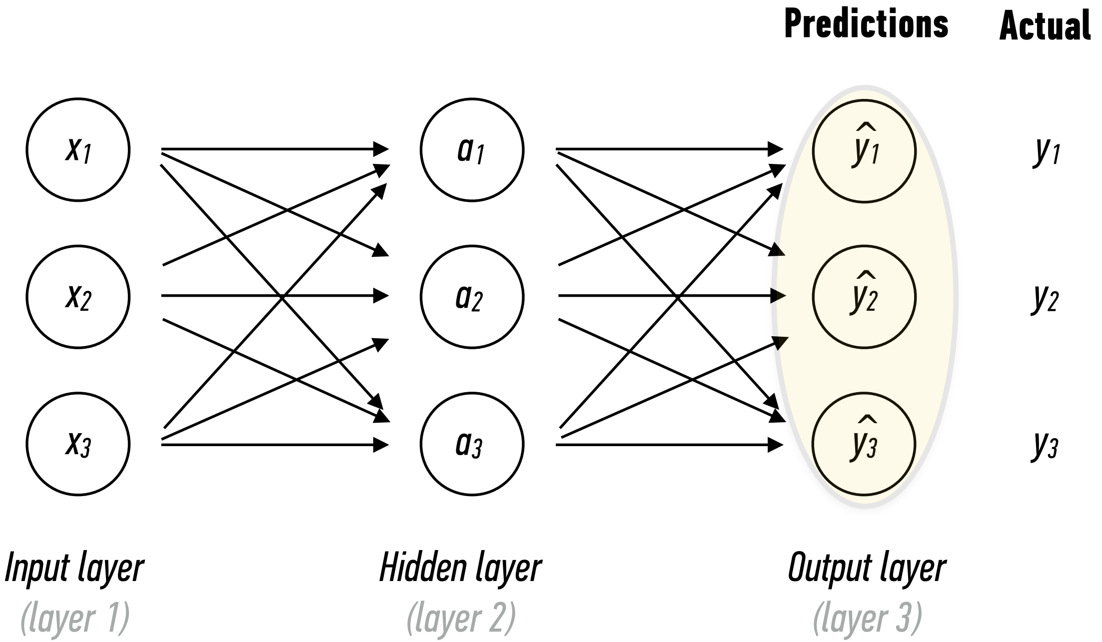


We compute the error for all samples



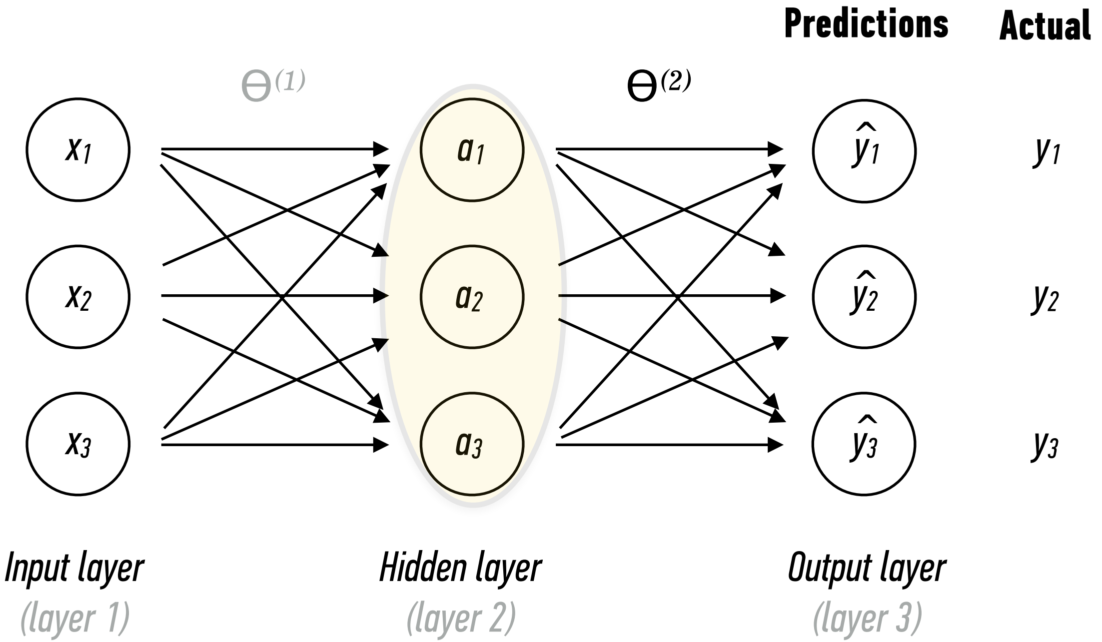
$$\delta_2^{(3)} = \hat{y}_2 - y_2$$

We compute the error for all samples, for all classes



$$\delta^{(3)} = \hat{y} - y$$

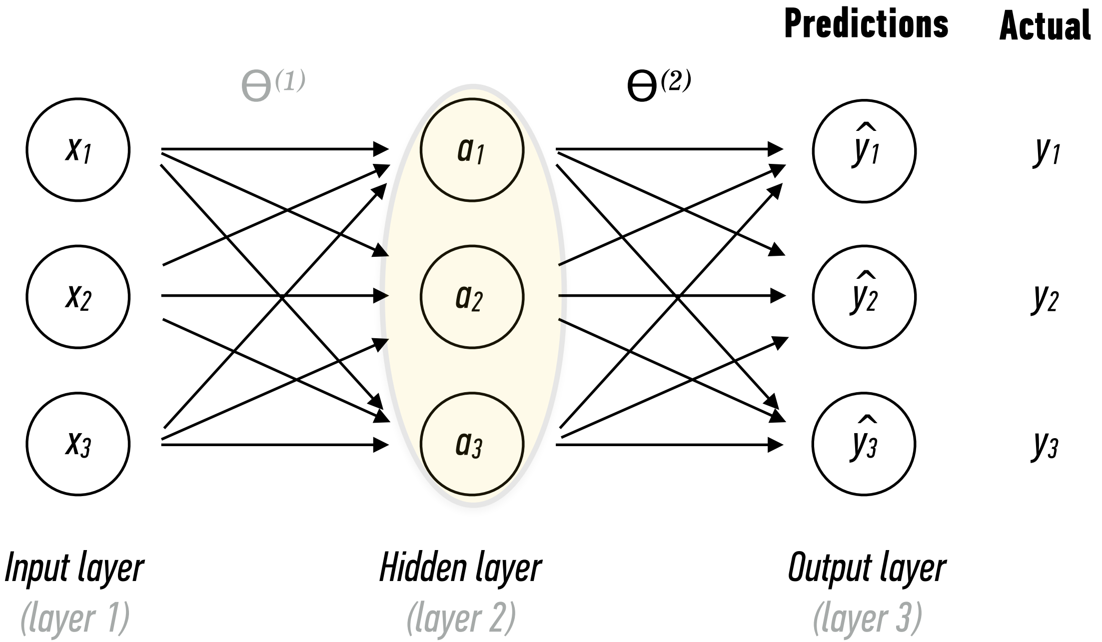
And we **back-propagate** *the error term through the network*



$$\delta^{(3)} = \hat{y} - y$$

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

And we **back-propagate** *the error term through the network*

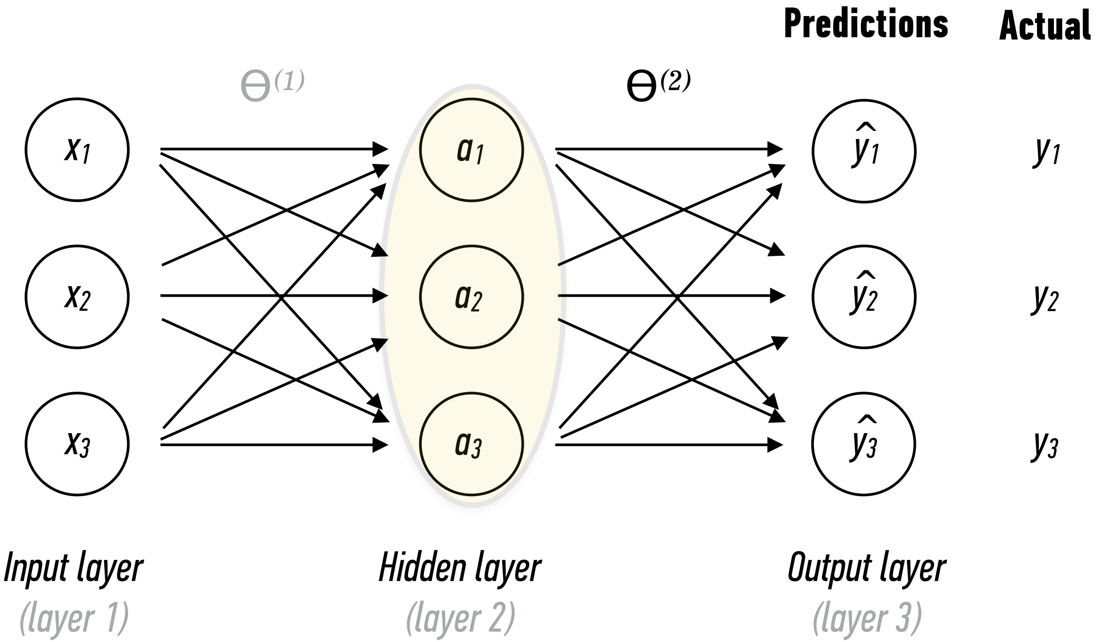


$$\delta^{(3)} = \hat{y} - y$$

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

. is element-wise multiplication

And we **back-propagate** *the error term through the network*



$$\delta^{(3)} = \hat{y} - y$$

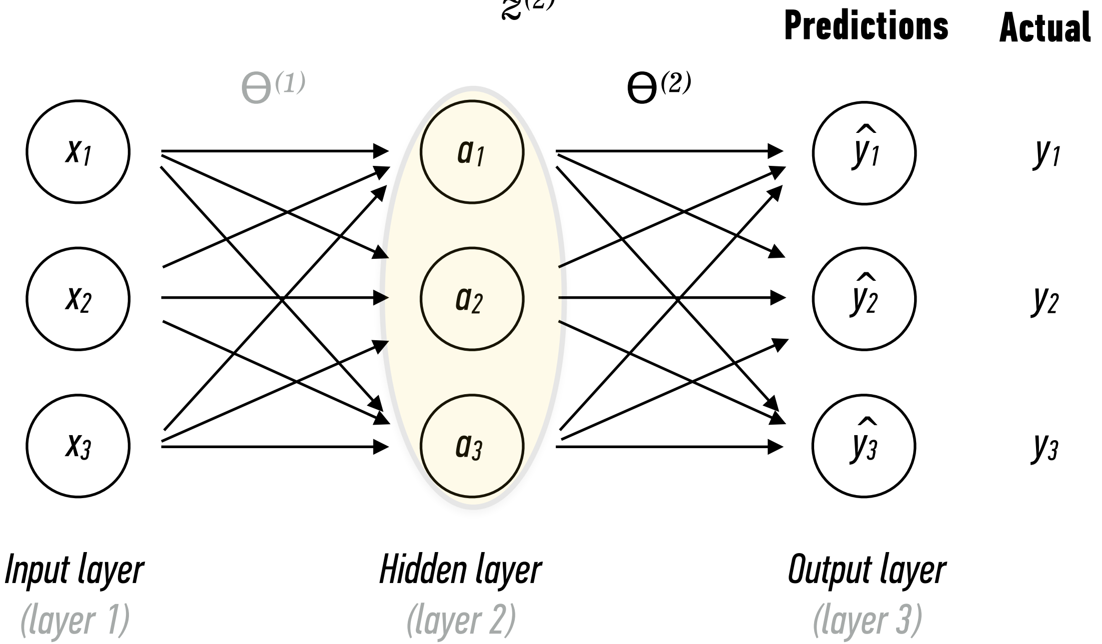
$$\delta^{(2)} = \Theta^{(2)T}\,\delta^{(3)} \,.\, g'(z^{(2)})$$

. is element-wise multiplication

g is activation function

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

$$a^{(2)} = g(\underbrace{\Theta^{(1)T} a^{(1)}}_{z^{(2)}})$$



$$\delta^{(3)} = \hat{y} - y$$

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

. is element-wise multiplication

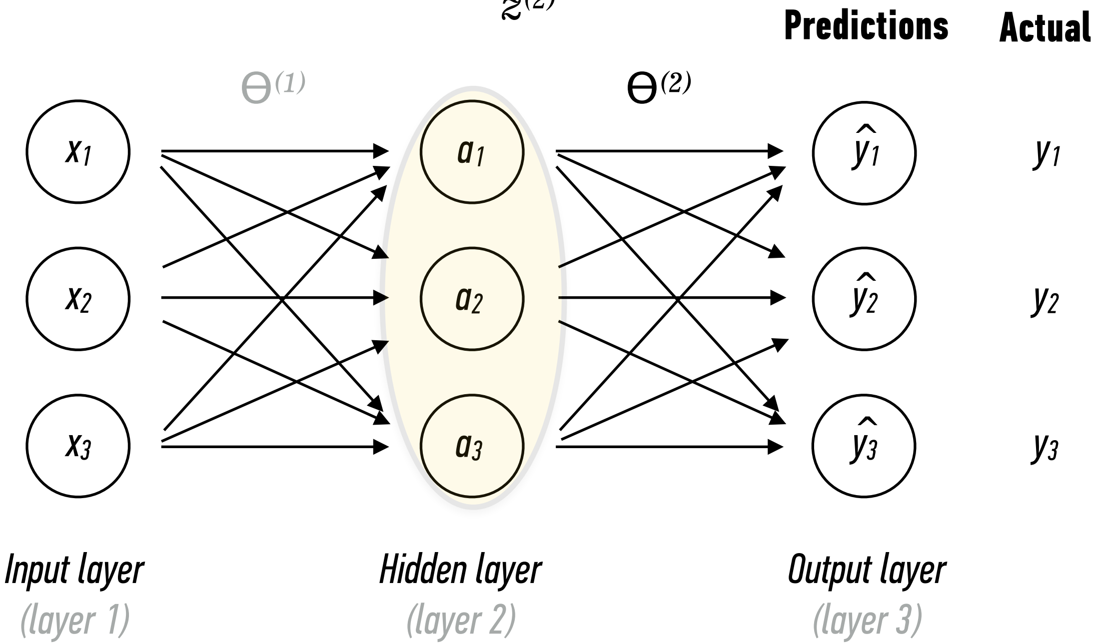
g is activation function

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

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

$a^{(1)} = x$

$a^{(2)} = g(\underbrace{\Theta^{(1)T} a^{(1)}}_{z^{(2)}})$



$\delta^{(3)} = \hat{y} - y$

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

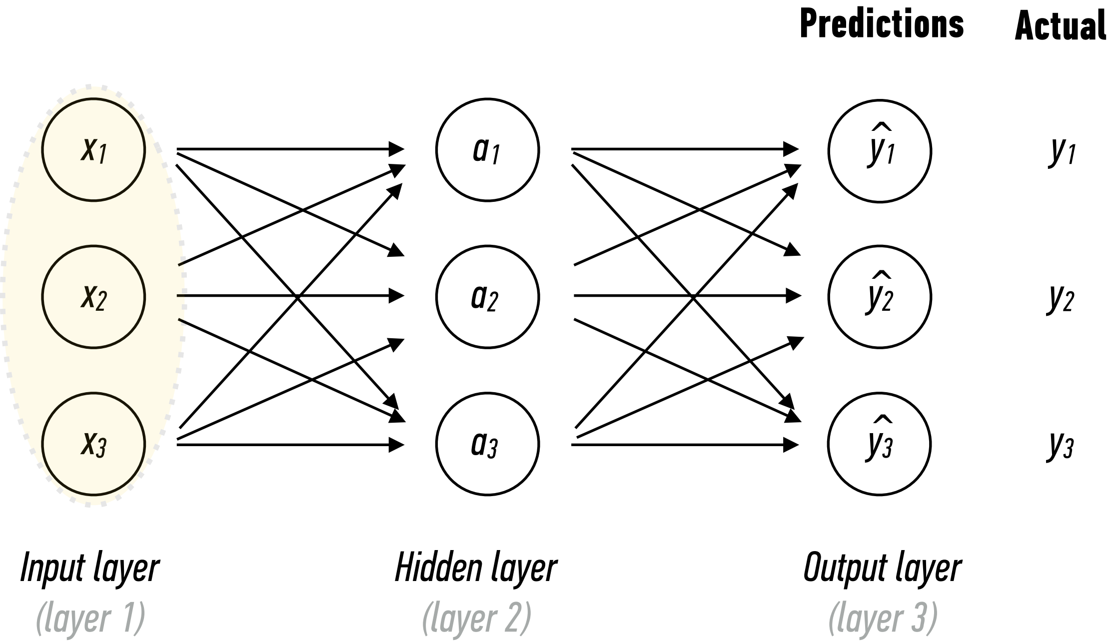
. is element-wise multiplication

g is activation function

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

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

$= \Theta^{(1)T} x$



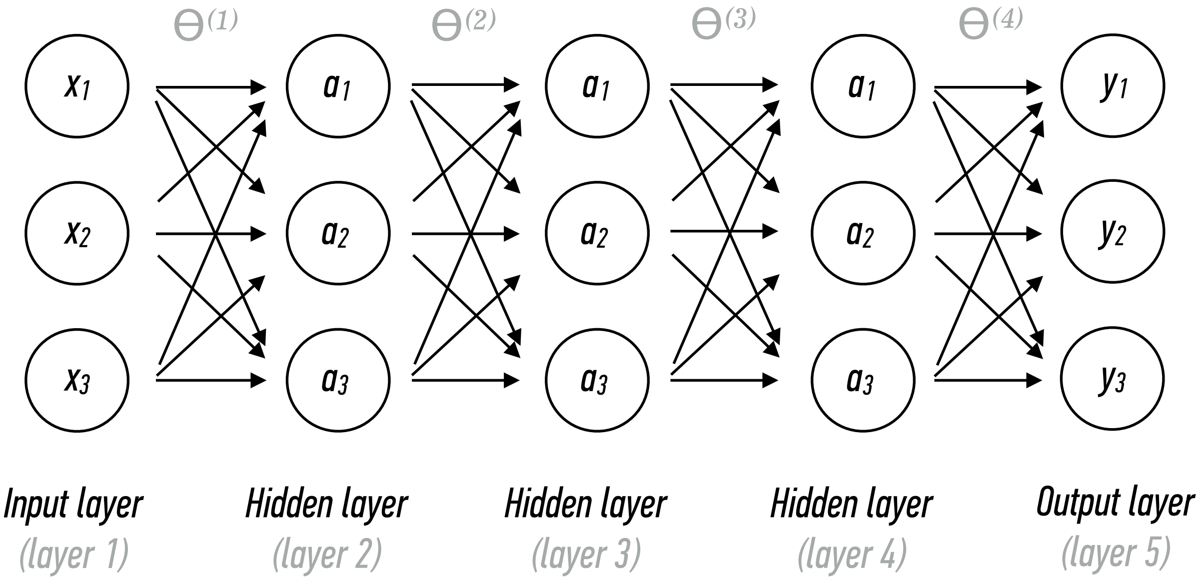
$$\delta^{(3)} = \hat{y} - y$$

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

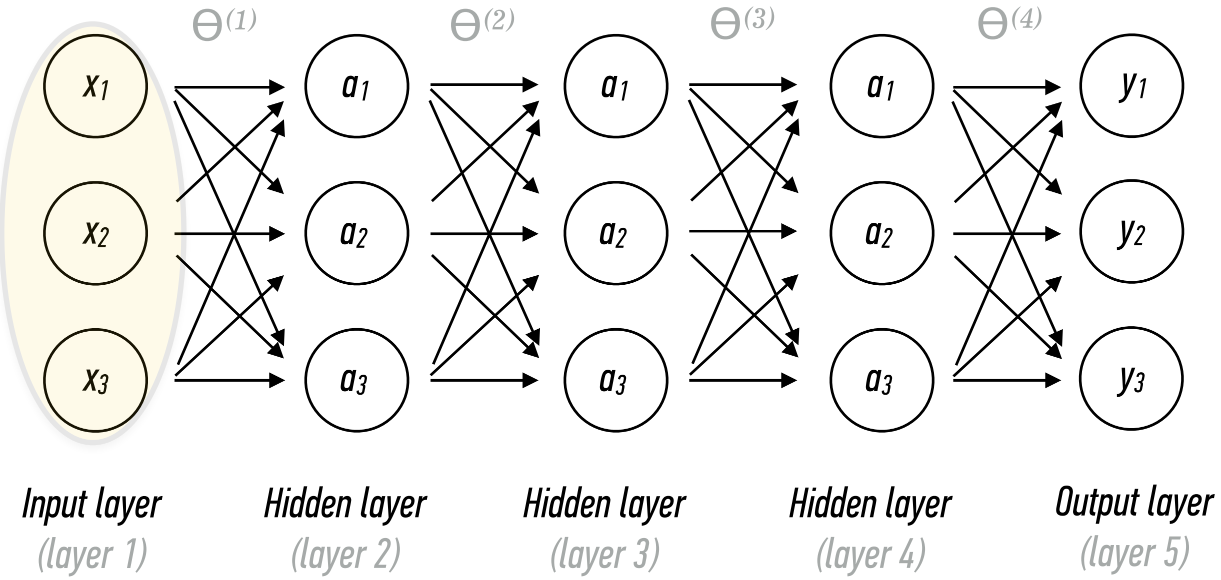
$\delta^{(1)}$ does not exist

(error of input makes no sense)

A deeper network might be more illustrative



$a^{(1)} = x$

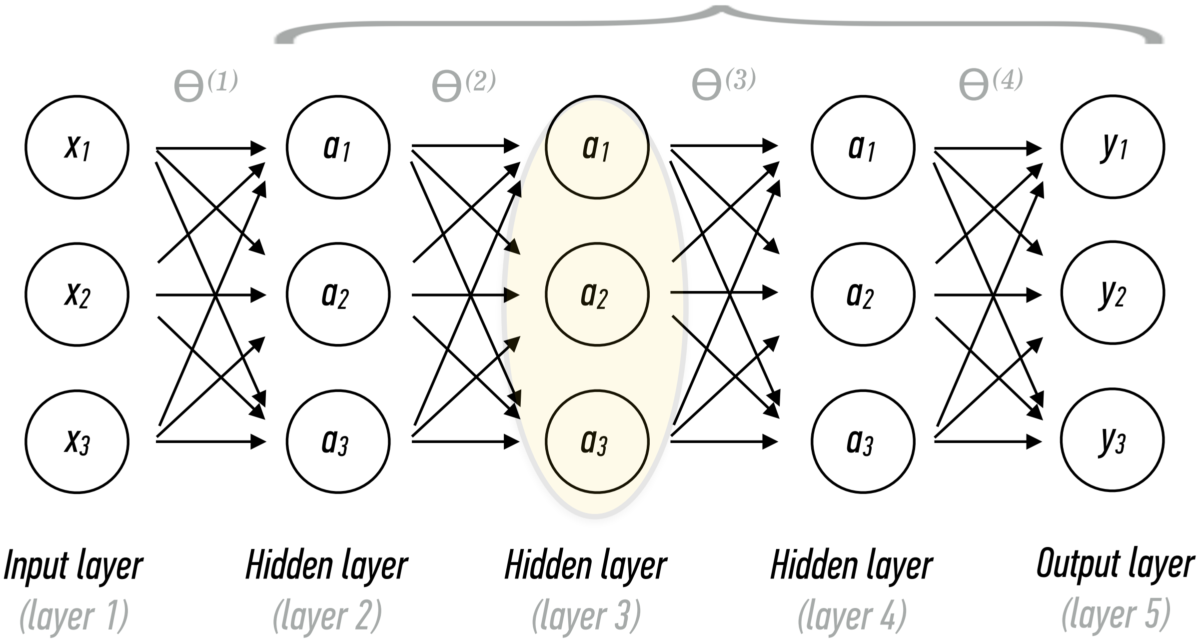


$\Theta^{(1)}$
$\Theta^{(2)}$
$\Theta^{(3)}$
$\Theta^{(4)}$
x_1
x_2
x_3
a_1
a_2
a_3
a_1
a_2
a_3
a_1
a_2
a_3
y_1
y_2
y_3
Input layer
(layer 1)
Hidden layer
(layer 2)
Hidden layer
(layer 3)
Hidden layer
(layer 4)
Output layer
(layer 5)

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

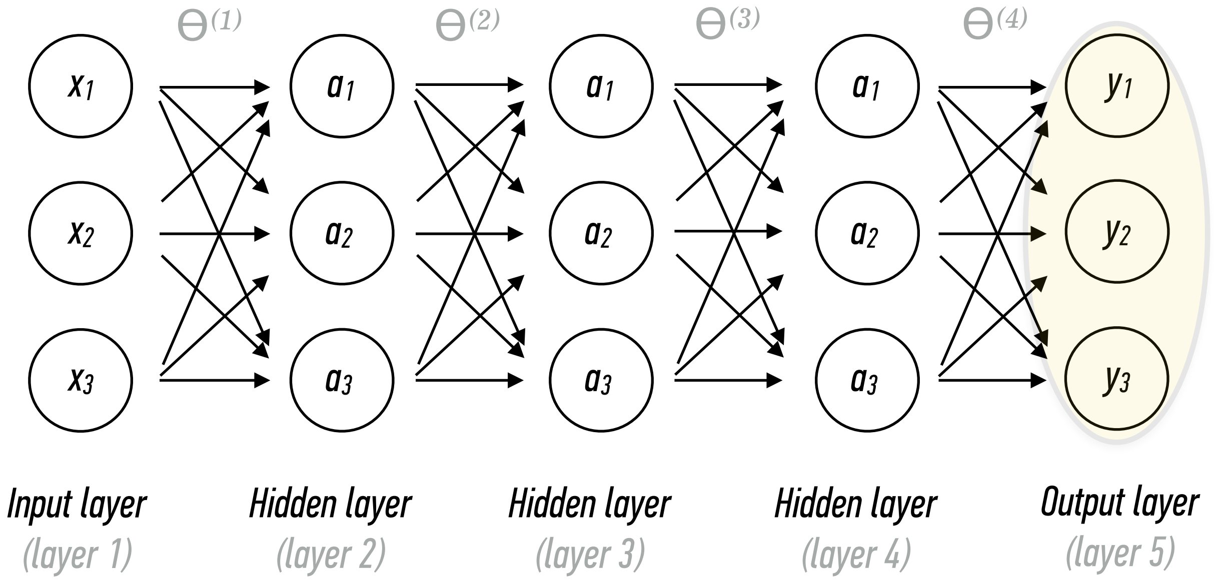
$$z^{(l)} = \Theta^{(l-1)T} a^{(l-1)}$$

$a^{(1)} = x$



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

$$a^{(1)}=x \qquad z^{(l)} = \Theta^{(l\text{-}1)T} a^{(l\text{-}1)} \qquad a^{(5)}=y$$

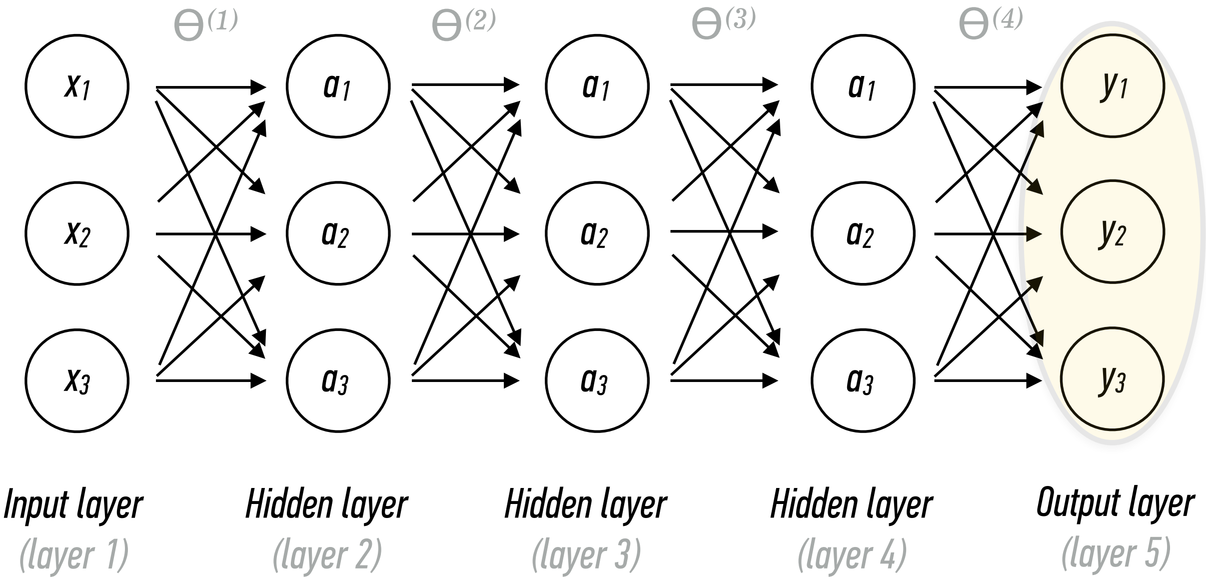


$a^{(l)} = g(z^{(l)})$

$a^{(1)} = x$

$z^{(l)} = \Theta^{(l\text{-}1)T}\, a^{(l\text{-}1)}$

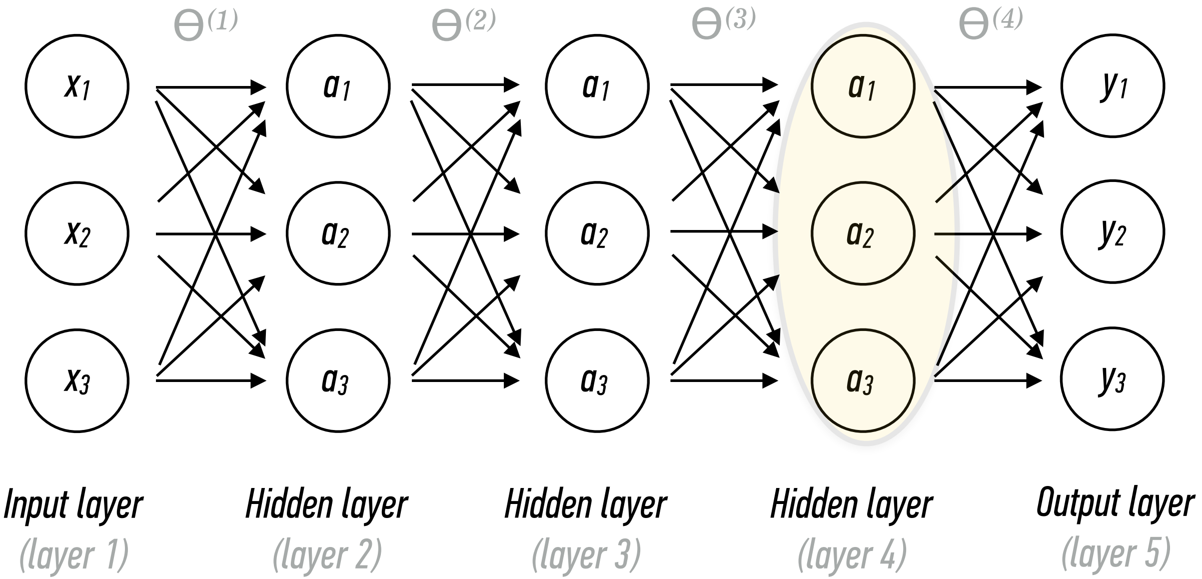
$a^{(5)} = y$



$\delta^{(5)} = \hat{y} - y$

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

$a^{(1)}=x$ $\qquad z^{(l)} = \Theta^{(l-1)T}\, a^{(l-1)}$ $\qquad a^{(5)}=y$

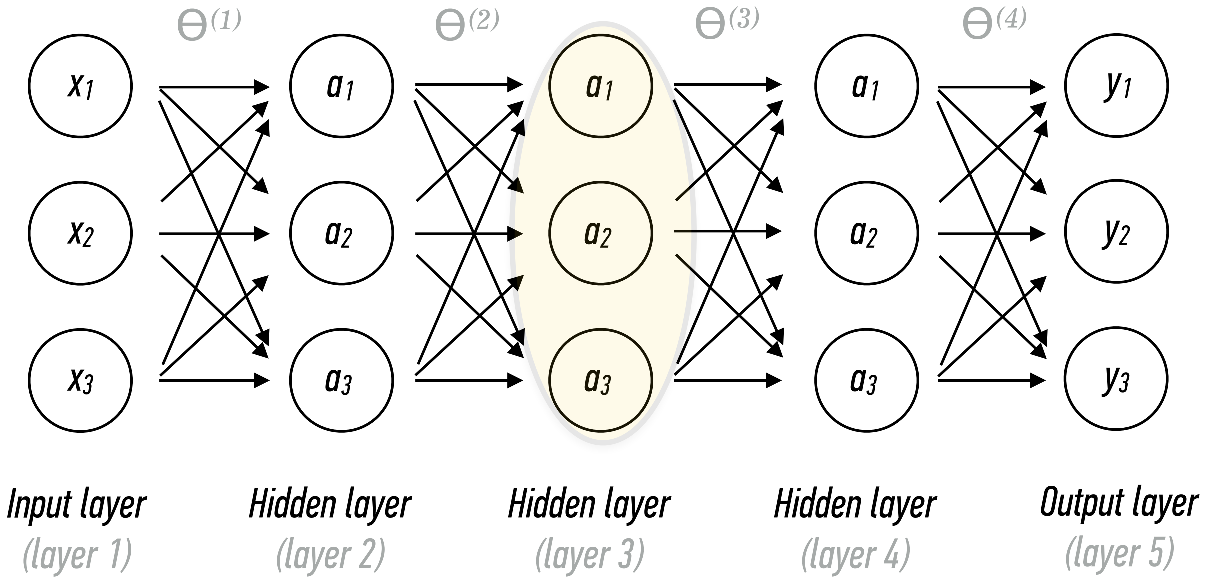


$$\delta^{(5)} = \hat{y} - y$$

$$\delta^{(4)} = \Theta^{(4)T}\, \delta^{(5)} \,.\, g'(z^{(4)})$$

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

$$a^{(1)} = x \qquad z^{(l)} = \Theta^{(l\text{-}1)T}\, a^{(l\text{-}1)} \qquad a^{(5)} = y$$



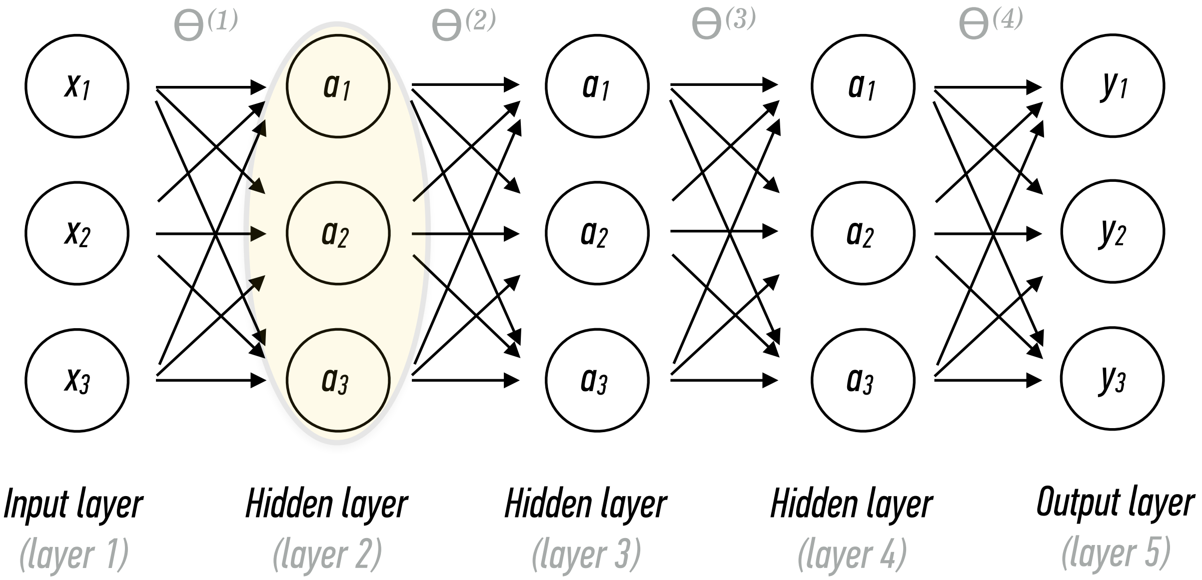
$$\delta^{(5)} = \hat{y} - y$$

$$\delta^{(4)} = \Theta^{(4)T}\, \delta^{(5)} \,.\, g'(z^{(4)})$$

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

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

$$a^{(1)} = x \qquad z^{(l)} = \Theta^{(l\text{-}1)T}\, a^{(l\text{-}1)} \qquad a^{(5)} = y$$



$$\delta^{(5)} = \hat{y} - y$$

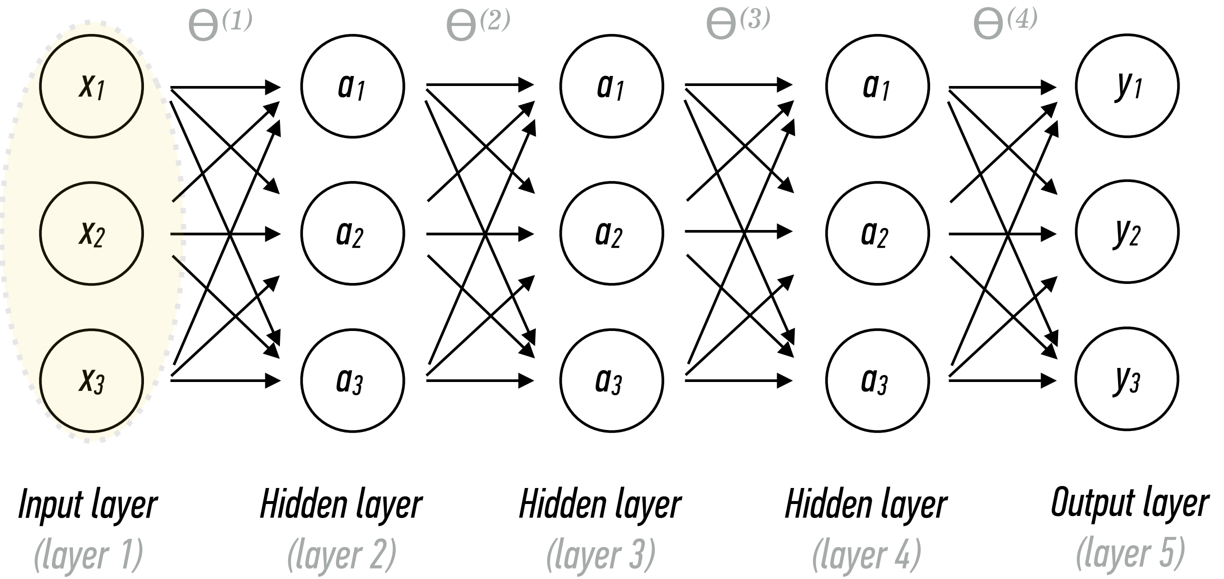
$$\delta^{(4)} = \Theta^{(4)T}\, \delta^{(5)} \,.\, g'(z^{(4)})$$

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

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

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

$$a^{(1)} = x \qquad z^{(l)} = \Theta^{(l-1)T} a^{(l-1)} \qquad a^{(5)} = y$$



$$\delta^{(5)} = \hat{y} - y$$

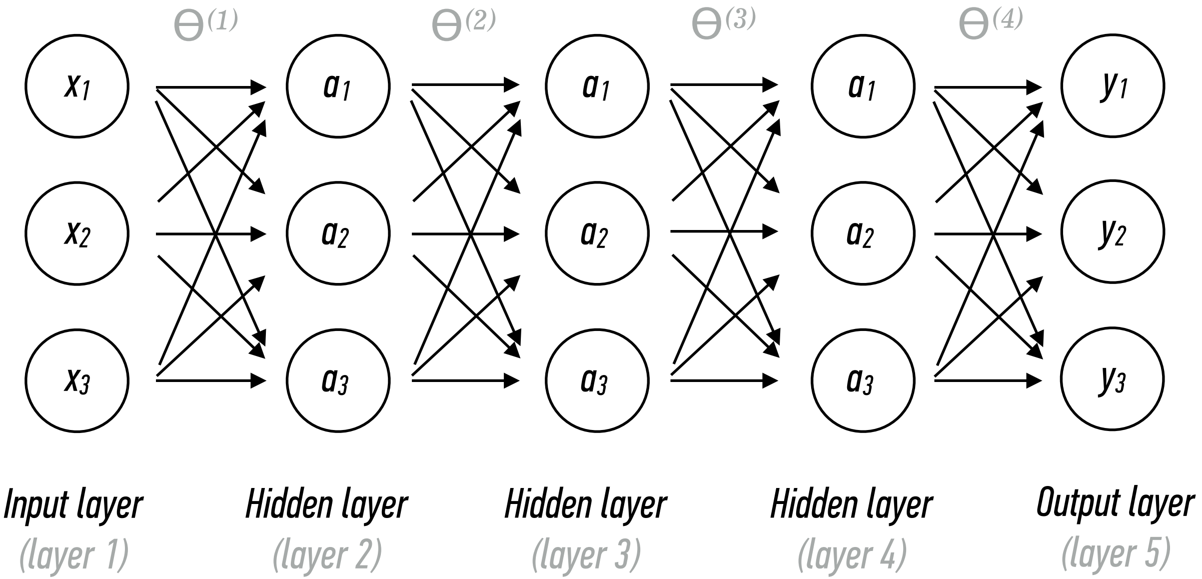
$$\delta^{(4)} = \Theta^{(4)T} \delta^{(5)} \,.\, g'(z^{(4)})$$

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

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

$\delta^{(1)}$ does not exist

We can now compute the partial derivatives by mapping the back-propagated error terms



$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

In the repo you'll find an example of back propagation in a simple neural network python implementation

DISCUSSION