GENERAL ASSEMBLY

NEURAL NETWORKS

LAST TIME - WEB TECHNOLOGIES FOR MACHINE LEARNING

- HOW ARE MODELS TRAINED ON LARGE CLUSTERS?
- HOW DOES TENSORFLOW WORK WITH AWS?

MARCH 9-15, 2016

ALPHA VS LEE SEDOL
1V1 - AI VS GRANDMASTER

DEEPMIND LINK: HTTP://DEEPMIND.COM/ALPHA-GO.HTML

LIVE STREAM LINK: HTTPS://WWW.YOUTUBE.COM/C/DEEPMINDAI

I. NEURAL NETWORKS AND HISTORY
II. LOGICAL OPERATORS
III. NEURAL NETWORK VARIETIES
IV. COST FUNCTION
V. BACKPROPAGATION
VI. CONSIDERATIONS
VII. RESOURCES

I. NEURAL NETWORKS AND HISTORY

*Neural networks are **graph based** machine learning algorithms that are very good at creating **complex non-linear decision boundaries** in very **high dimensional feature space***

*Neural networks are **graph based** machine learning algorithms that are very good at creating **complex non-linear decision boundaries** in very **high dimensional feature space***

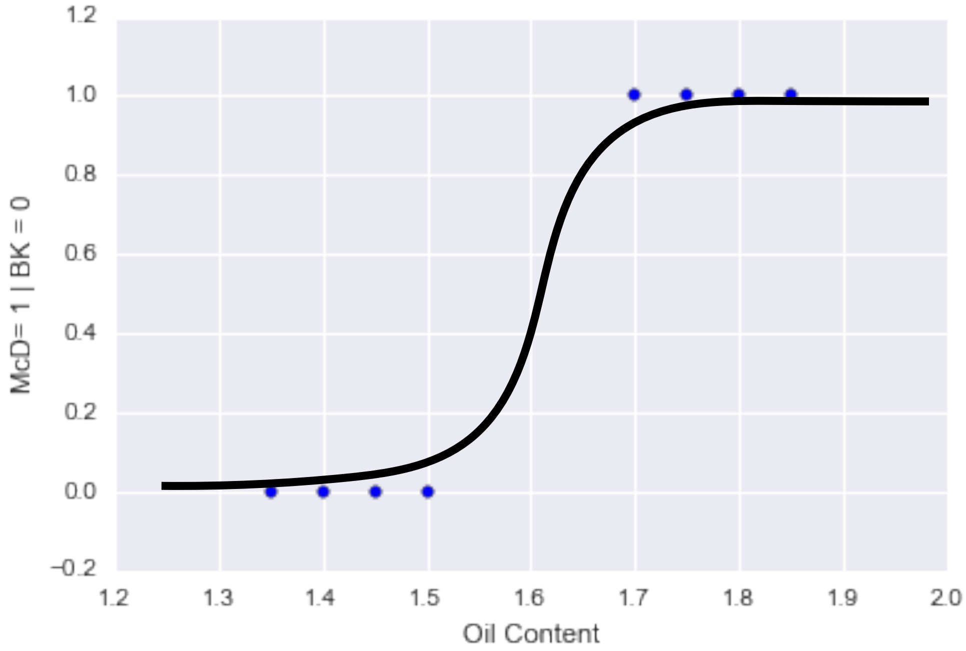
Sound familiar?

- Biologically motivated and developed in the 1970s
- Very widely used in the 1980s and early 90s
- Then **Support Vector Machines** (i.e. kernel trick + slack variables) became the more powerful technique
- Recent resurgence (post 2006) as previous limitations became understood and computational power became available

- Neural networks required heavy computational power which was not widely available
- Most of the focus was on one type of network, the multi-layered perceptron (MLP)
- GPUs and CUDA support helped propel the research into consumer space

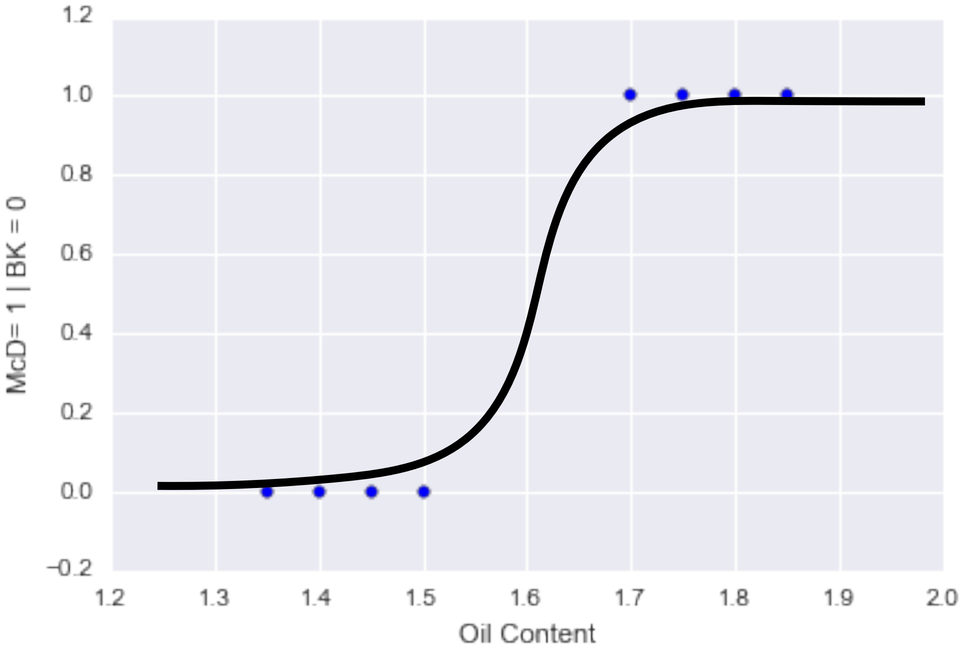
Let's start off with logistic regression

$$y = \frac{1}{1+e^{-(\alpha+\beta_1 x_1+\ldots+\beta_n x_n)}}$$



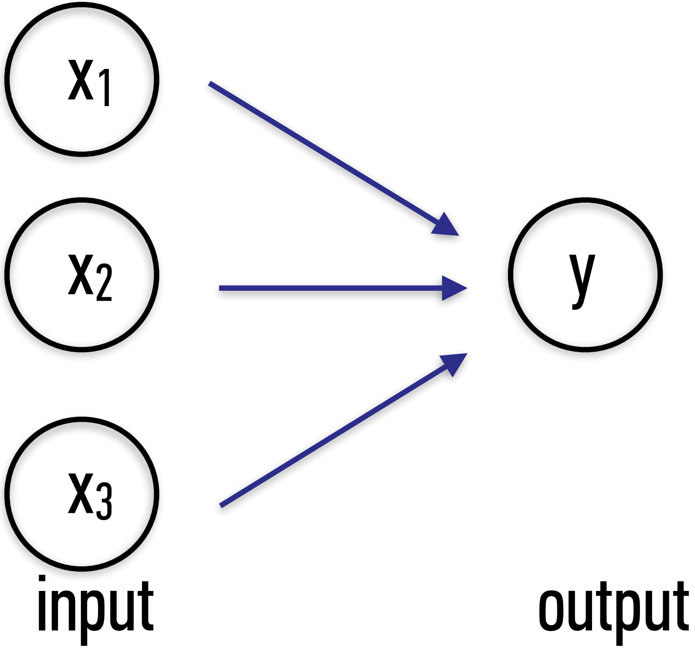
Let's start off with logistic regression

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



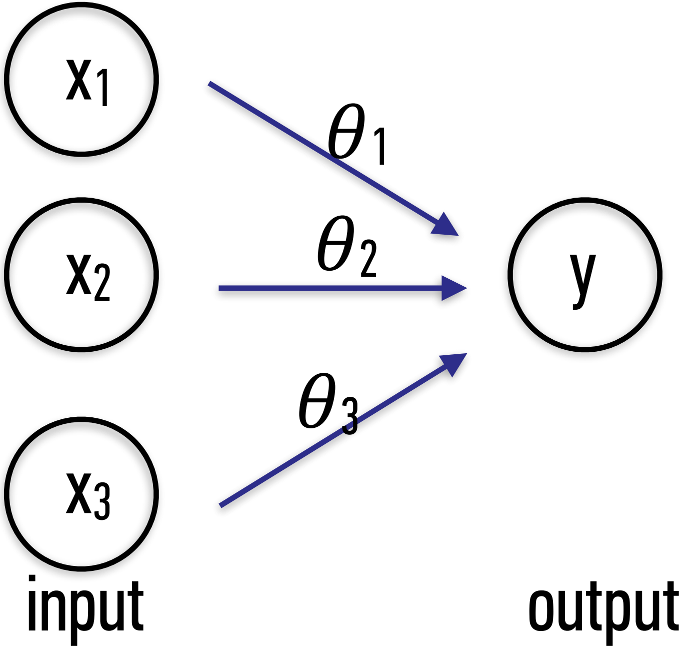
...and reformulate the problem

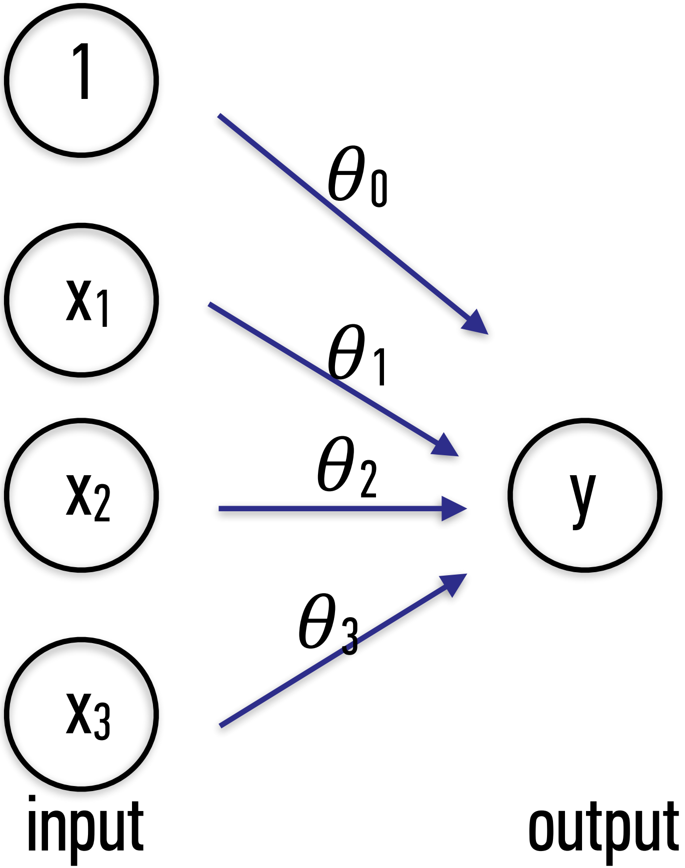
$$y = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)}}$$



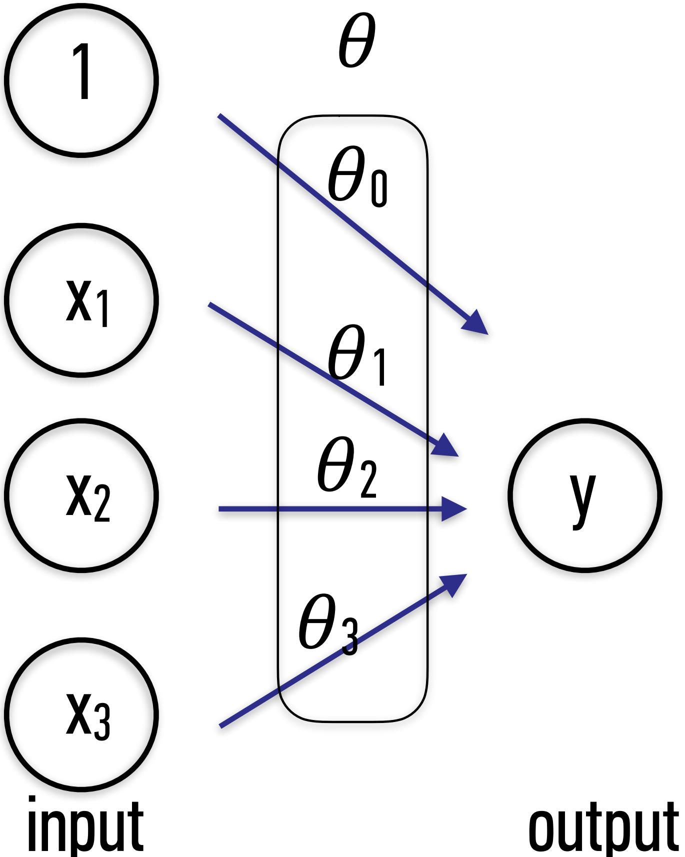
...and reformulate the problem

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



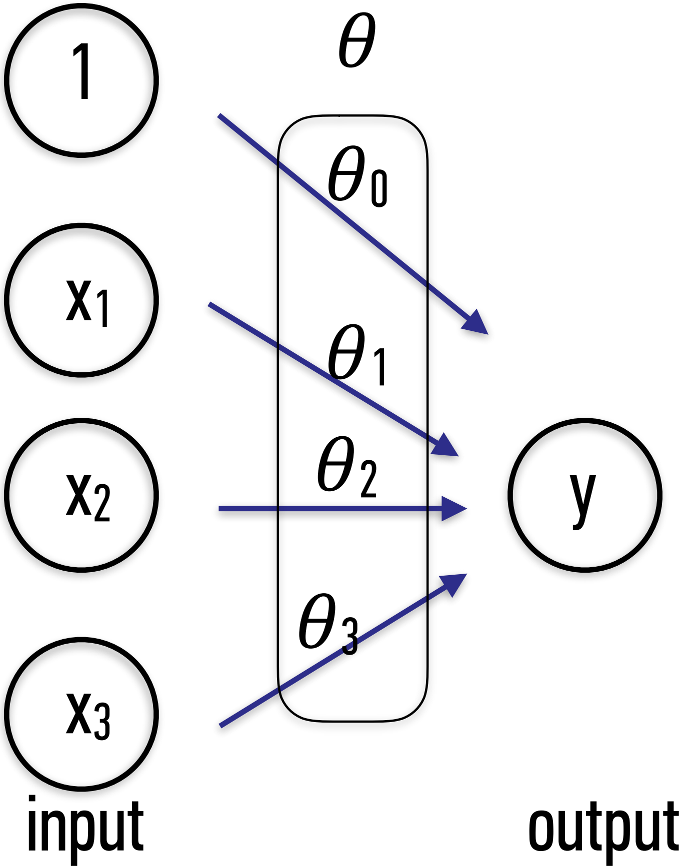


$$y = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3)}}$$



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

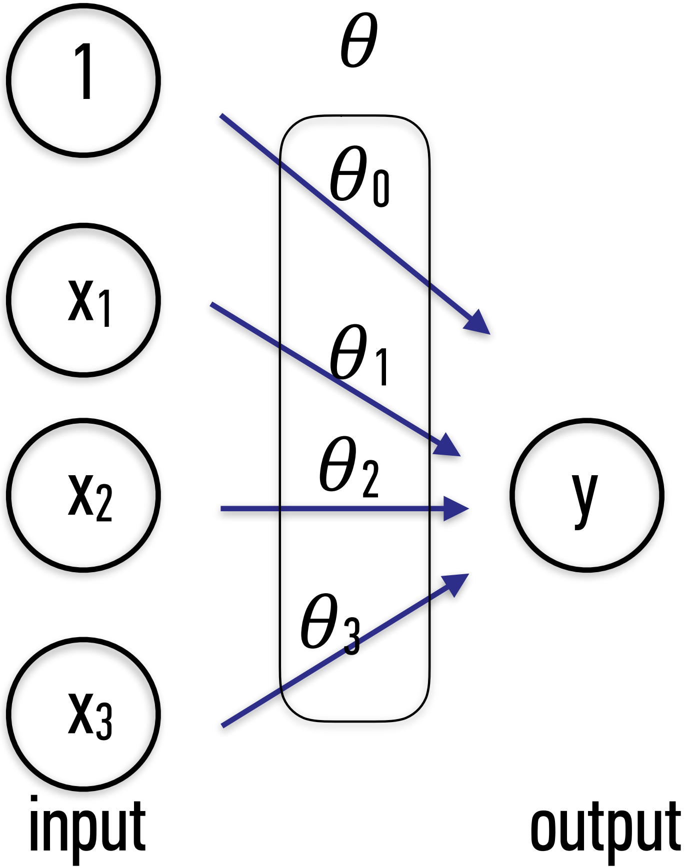
$$\theta = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \theta_3 \end{bmatrix}$$



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

$$y = \sigma(x)$$

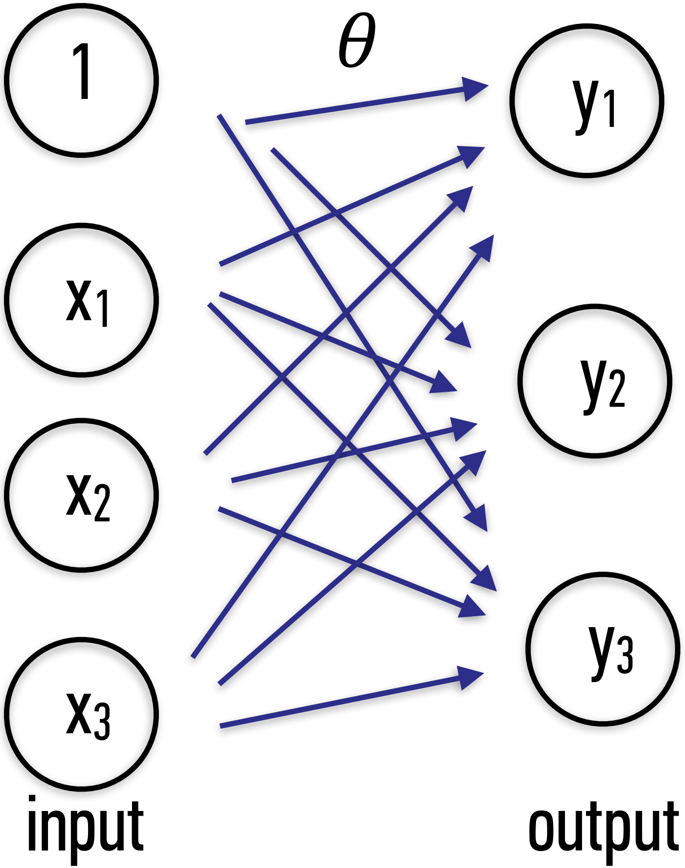
Denote the sigmoid by sigma



$$y = \sigma(x)$$

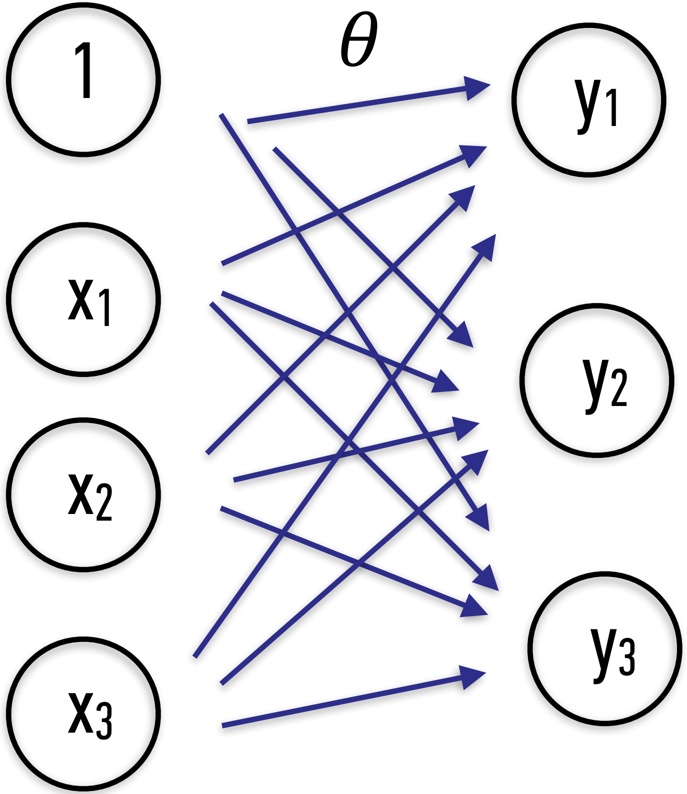
$$y = h_\theta(x)$$

Denote the sigmoid by sigma, or the **activation function**



$$y = h_\theta(x)$$

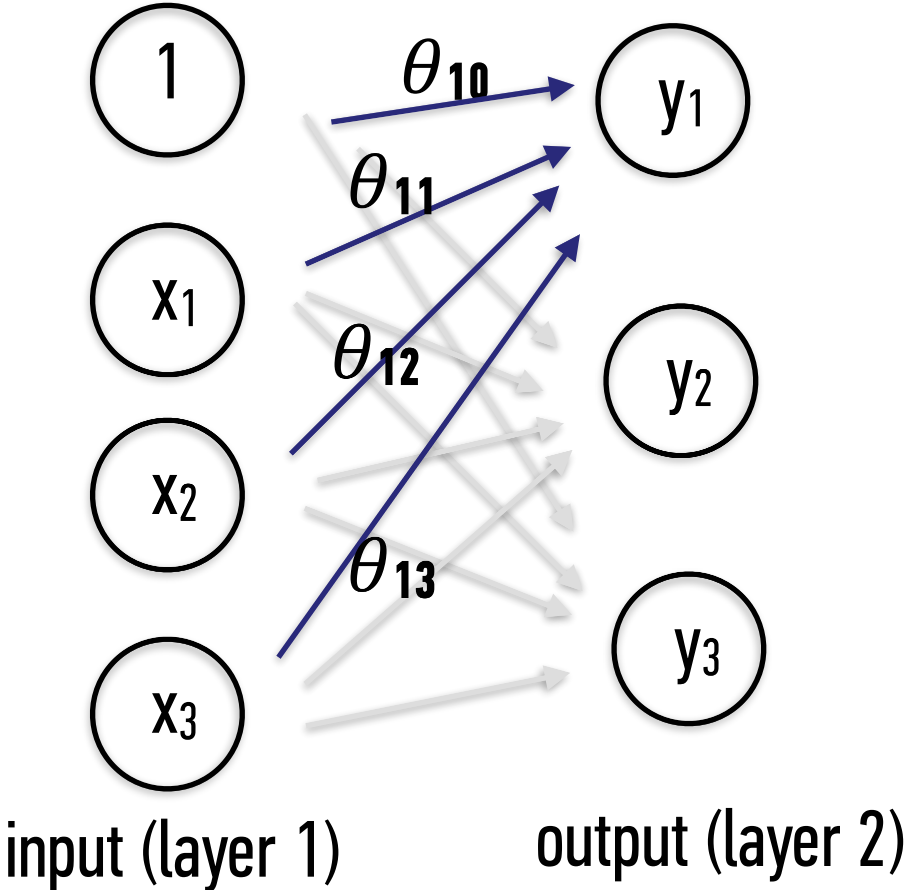
If you have multiple output variables, theta is denoted by a matrix instead of a vector



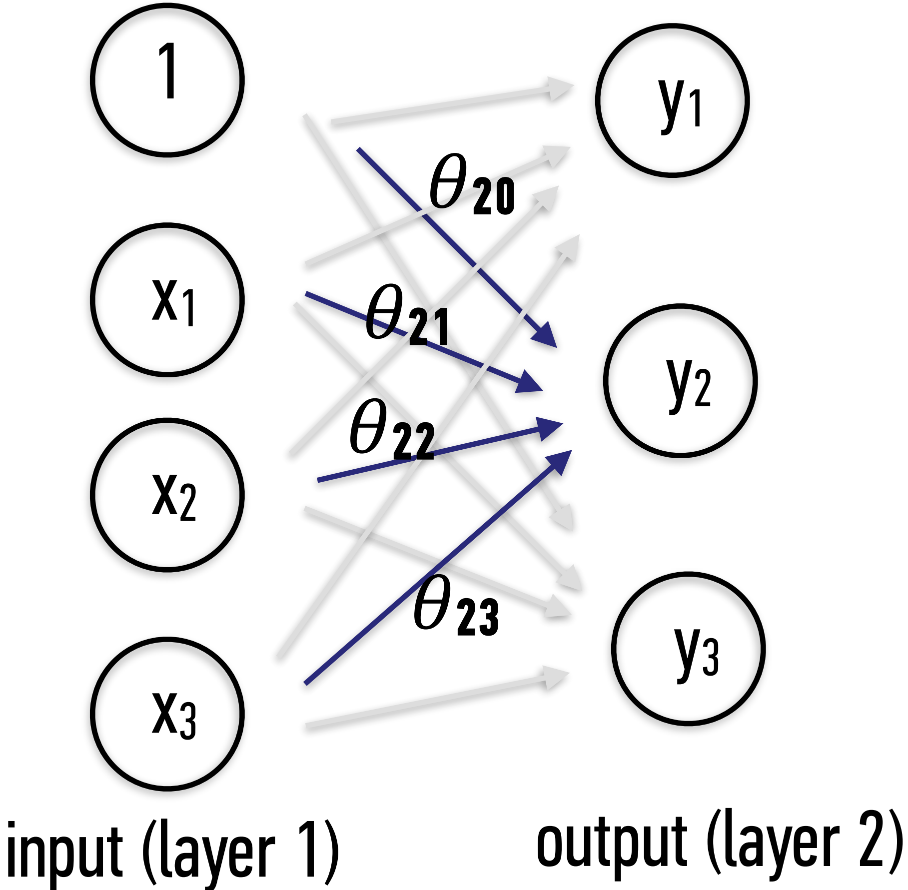
input (layer 1)　output (layer 2)

$$y = h_\theta(x)$$

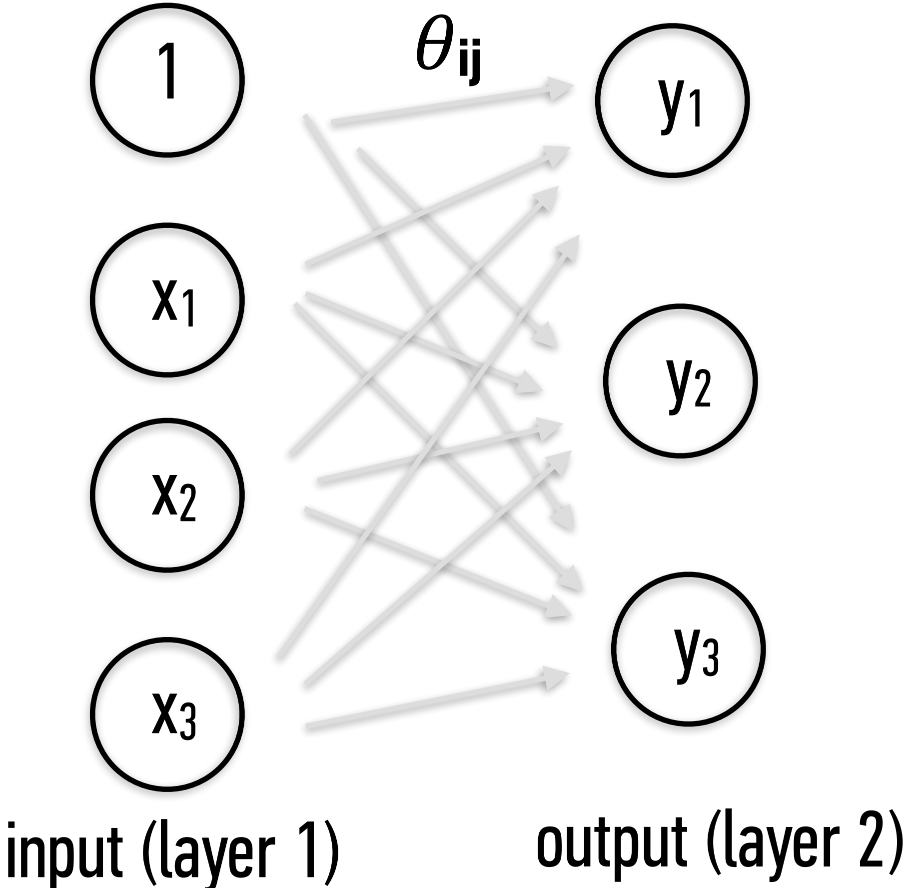
We can now speak of input and output layers



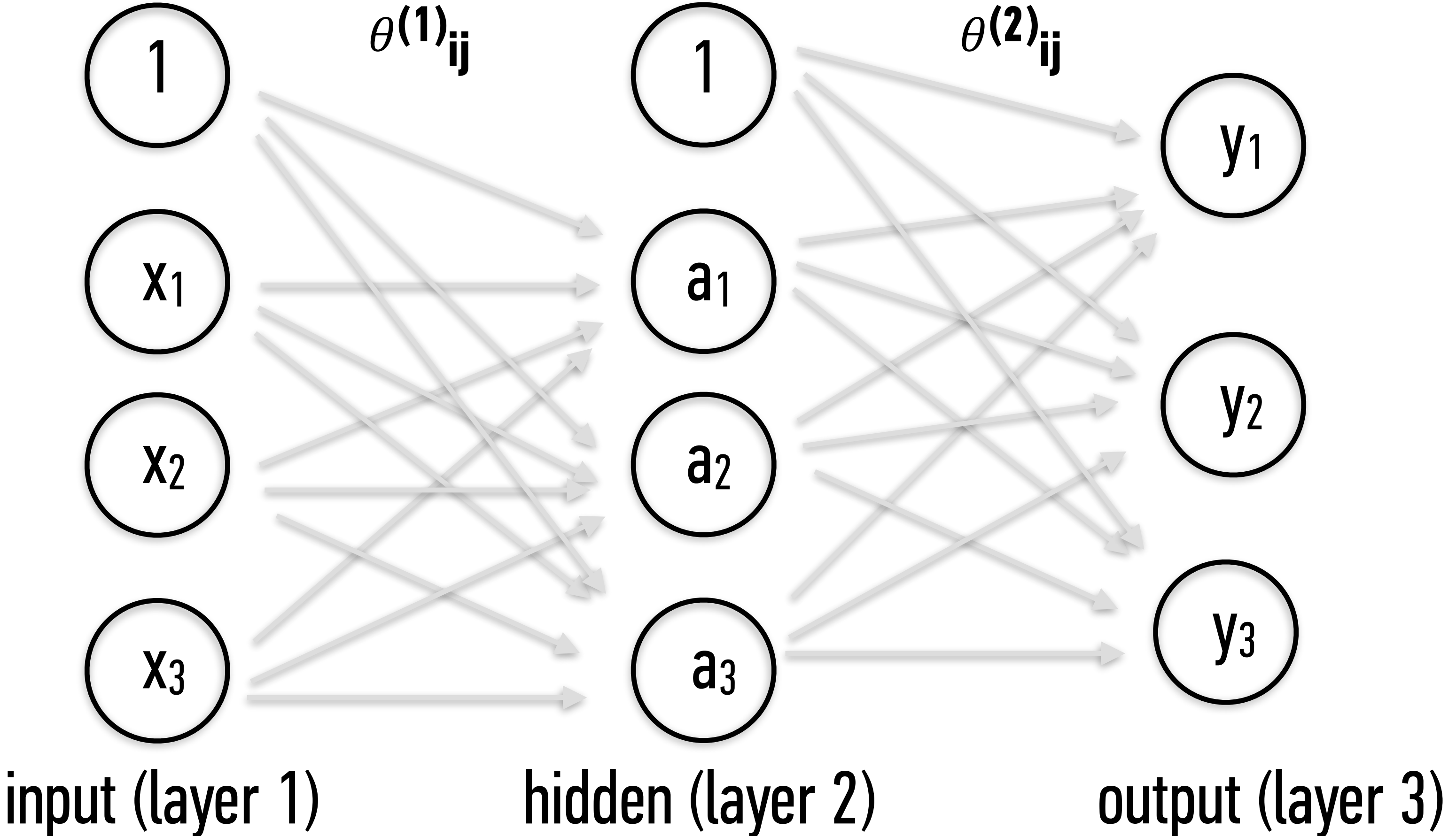
$$y = h_\theta(x)$$



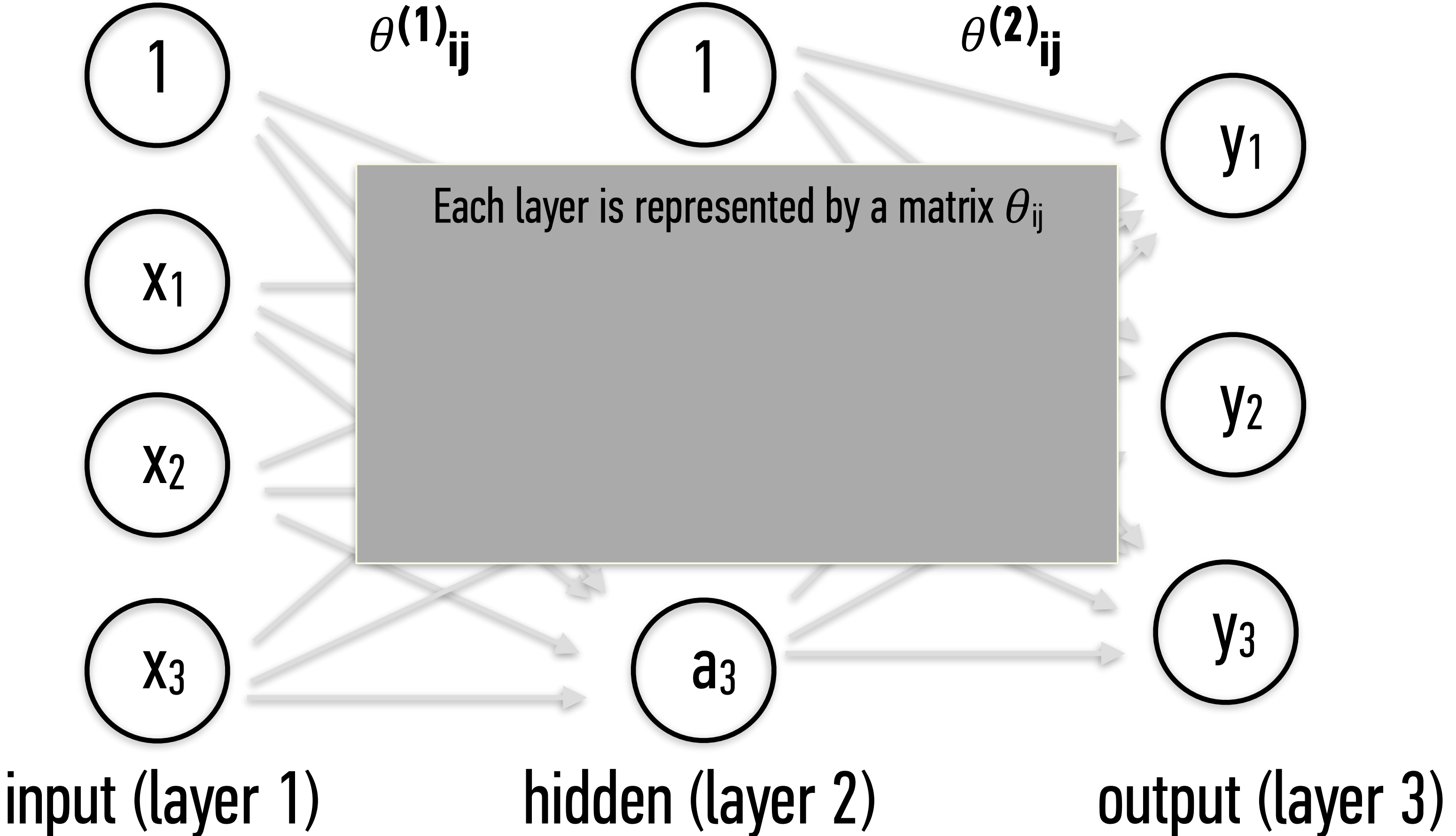
$$y = h_\theta(x)$$



$$y = h_\theta(x)$$

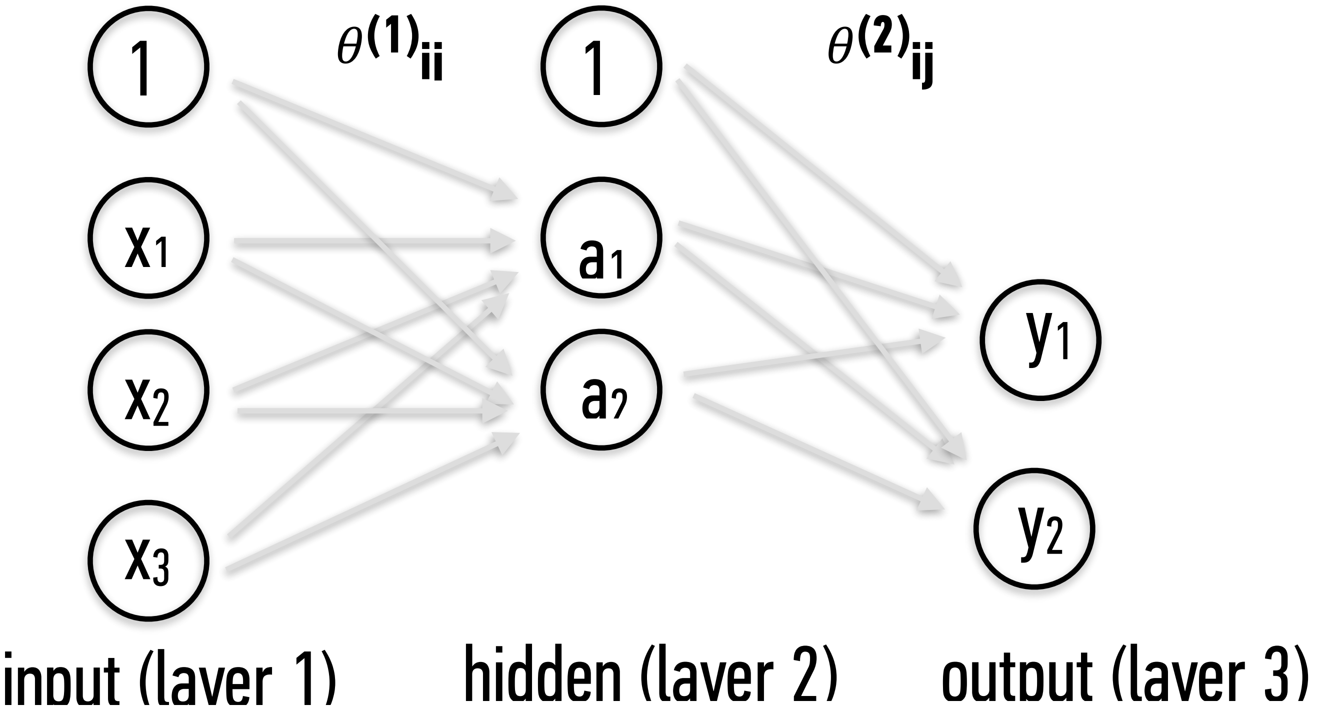


$\theta^{(1)}_{ij}$
$\theta^{(2)}_{ij}$
1
x_1
x_2
x_3
1
a_1
a_2
a_3
y_1
y_2
y_3
input (layer 1)
hidden (layer 2)
output (layer 3)

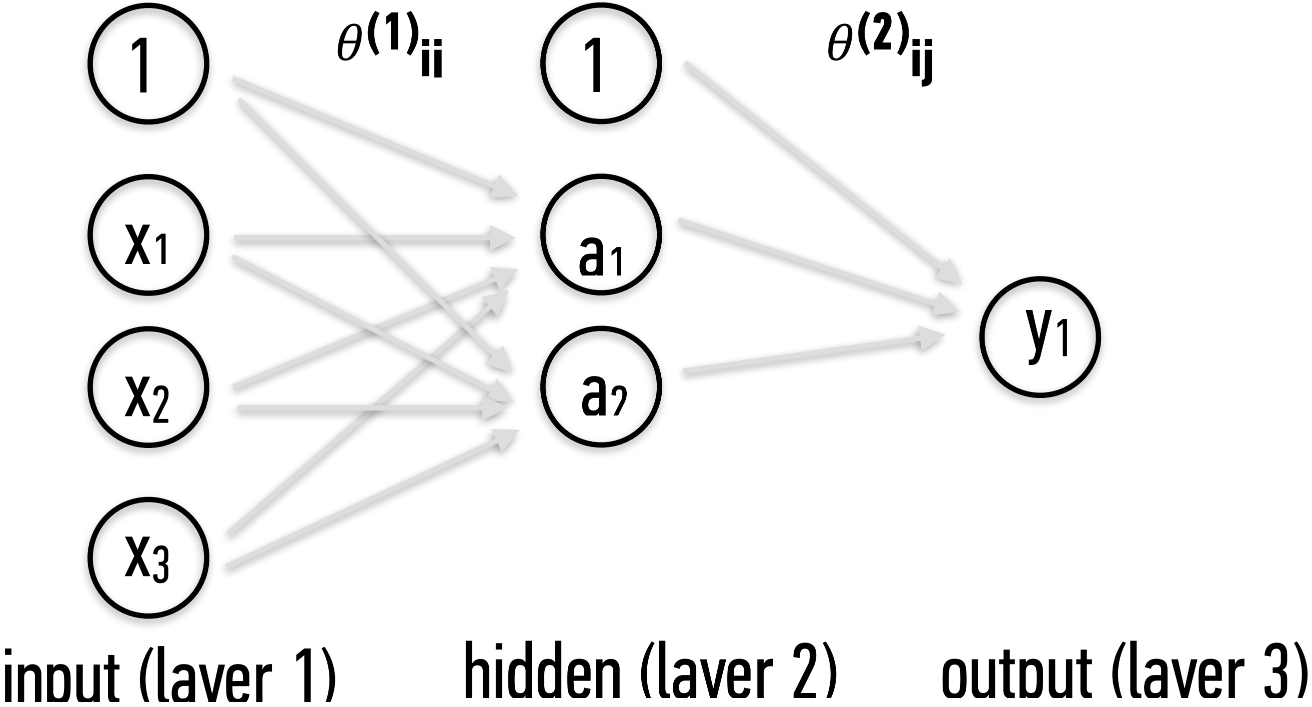


$\theta^{(1)}_{ij}$
$\theta^{(2)}_{ij}$
1
x_1
x_2
x_3
1
a_3
y_1
y_2
y_3
Each layer is represented by a matrix θ_{ij}
input (layer 1)
hidden (layer 2)
output (layer 3)

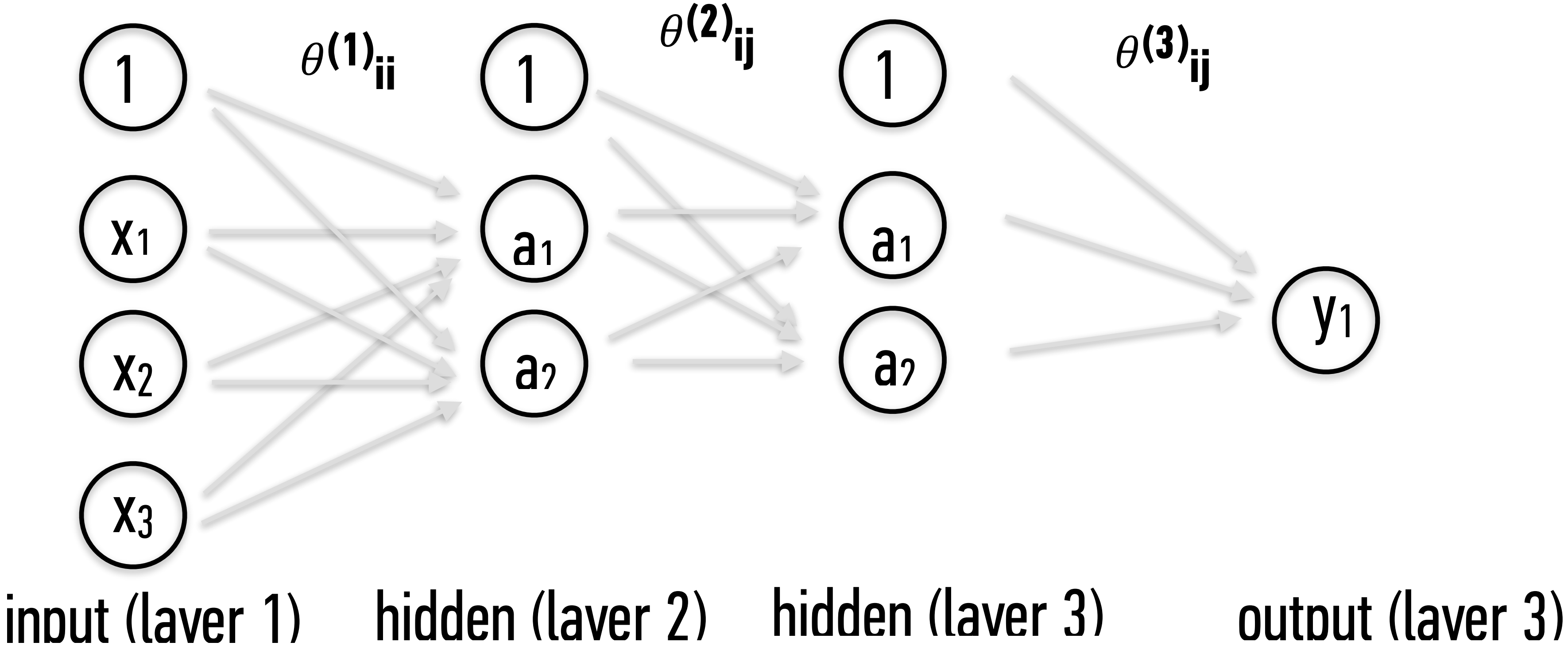
Different formulations of the network are possible



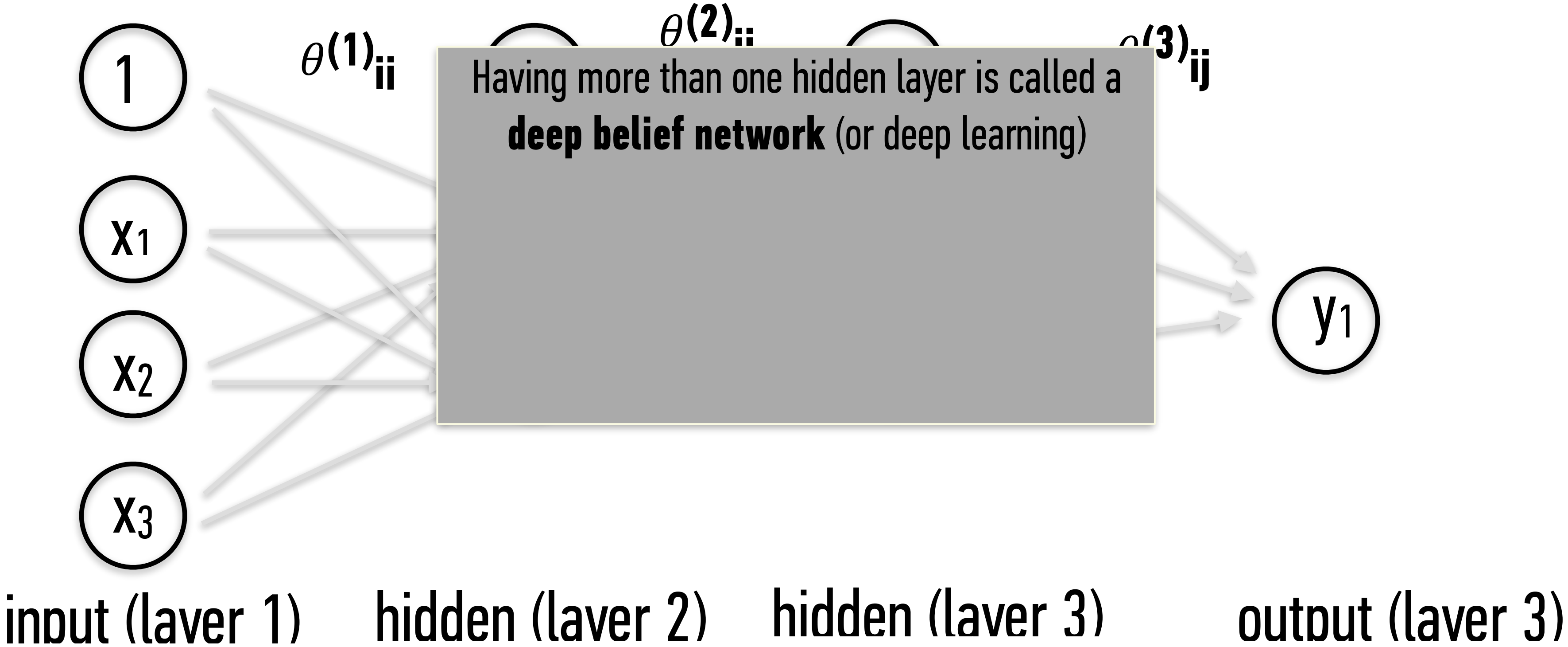
Different formulations of the network are possible



Different formulations of the network are possible



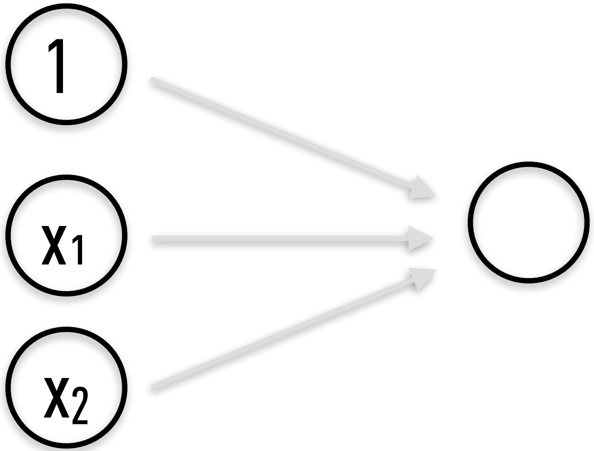
Different formulations of the network are possible



Having more than one hidden layer is called a **deep belief network** (or deep learning)

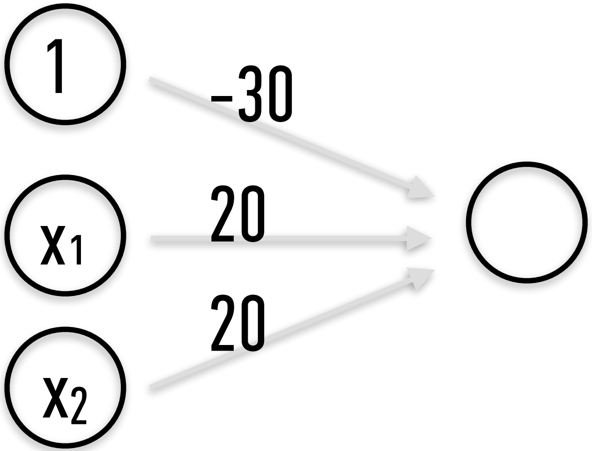
II. LOGICAL OPERATORS

Let's go through an example of a simple network



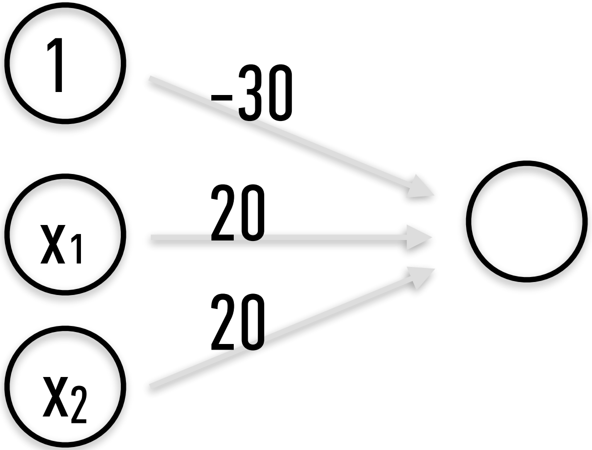
$$y = \sigma(z)$$

Let's go through an example of a simple network



$$y = \sigma(z)$$

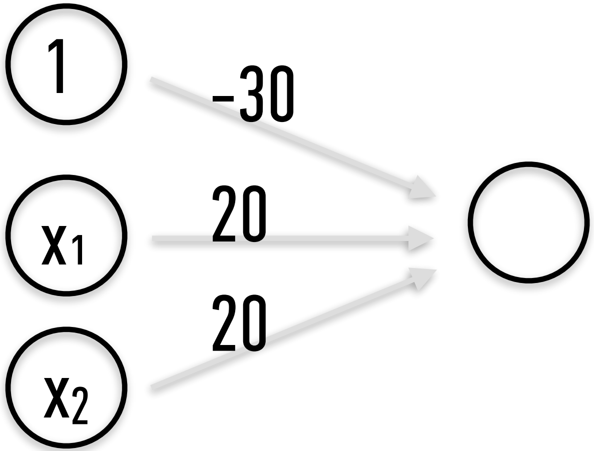
Let's say x_1 and x_2 only take binary values of 0 and 1



x_1	x_2
0	0
0	1
1	0
1	1

$$y = \sigma(z)$$

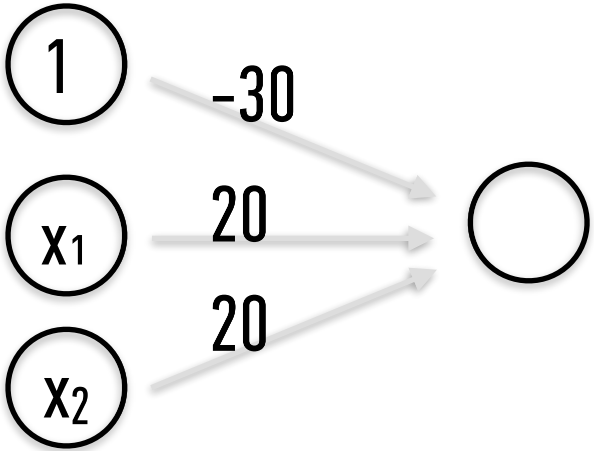
Let's say x_1 and x_2 only take binary values of 0 and 1



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**

$$y = \sigma(z)$$

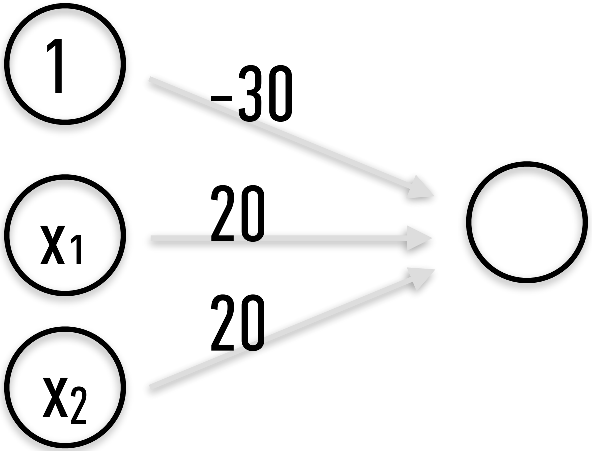
Let's say x_1 and x_2 only take binary values of 0 and 1



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**

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

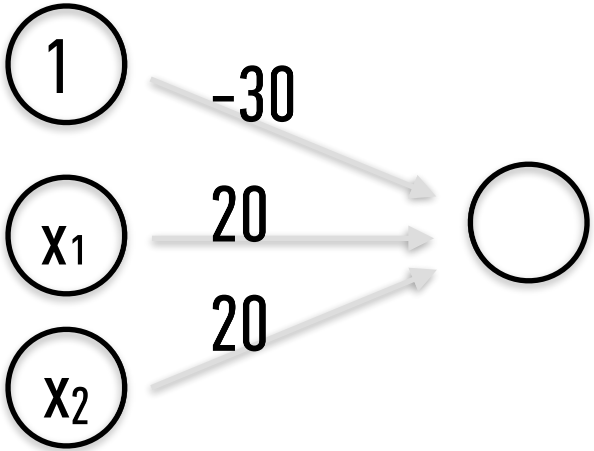
Let's say x_1 and x_2 only take binary values of 0 and 1



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

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

This node represents the logical **AND** operator

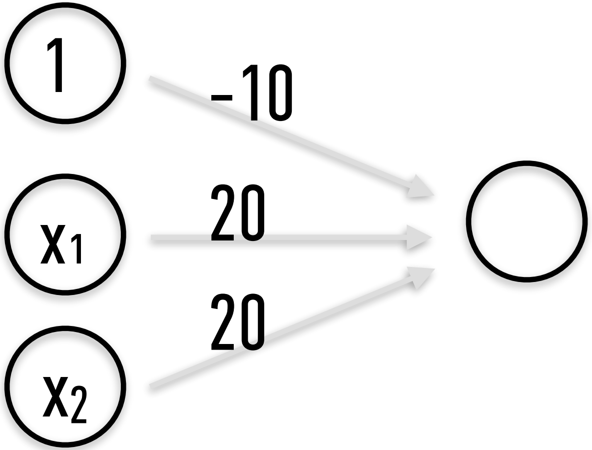


x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

x_1 AND x_2

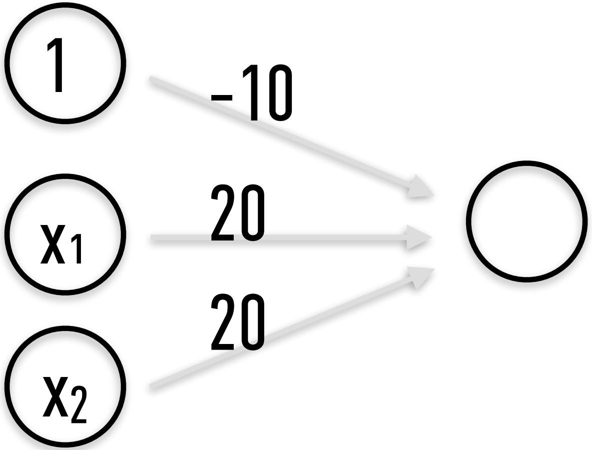
$$y = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\theta x}}$$

Another example



$$y = \sigma(z)$$

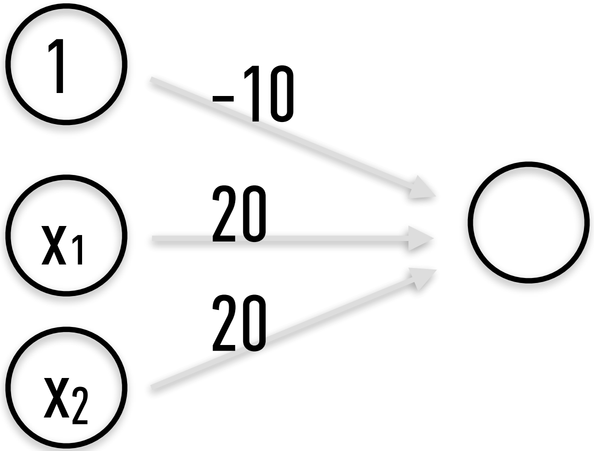
Another example



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=**30**	1

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

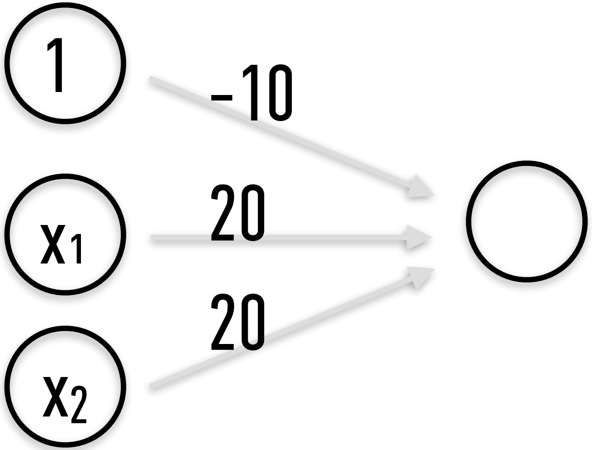
This node represents the logical **OR** operator



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=**30**	1

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

This node represents the logical **OR** operator

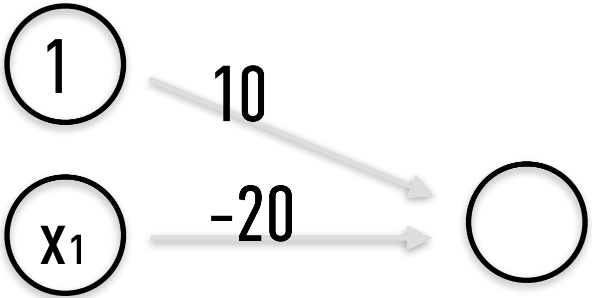


x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

x_1 OR x_2

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

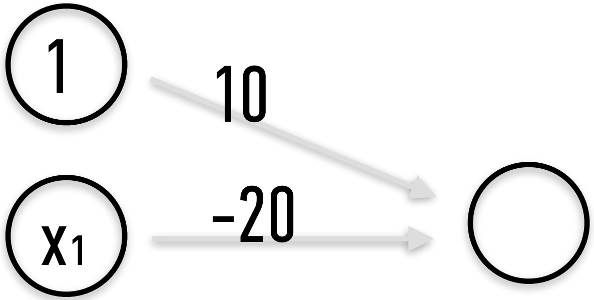
Let's build the logical **NOT** operator



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

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

Let's build the logical **NOT** operator

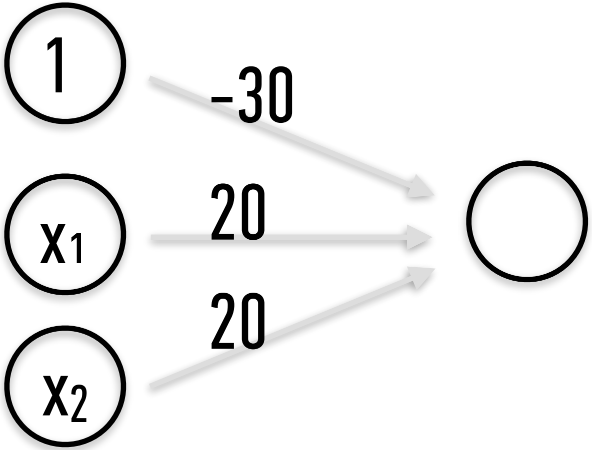


x_1	y
0	1
1	0

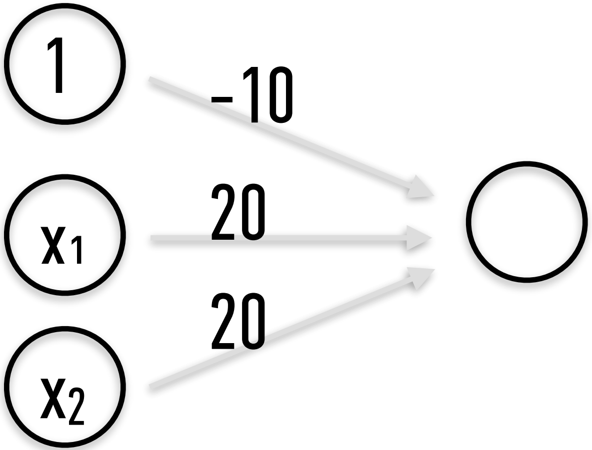
NOT x_1

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

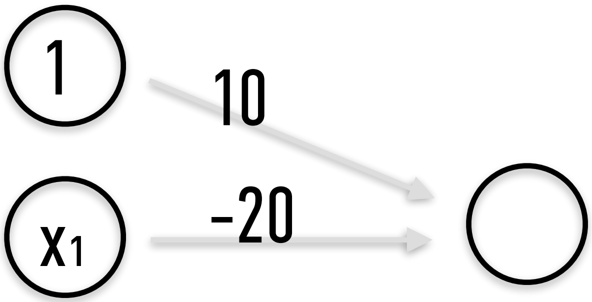
x_1 AND x_2



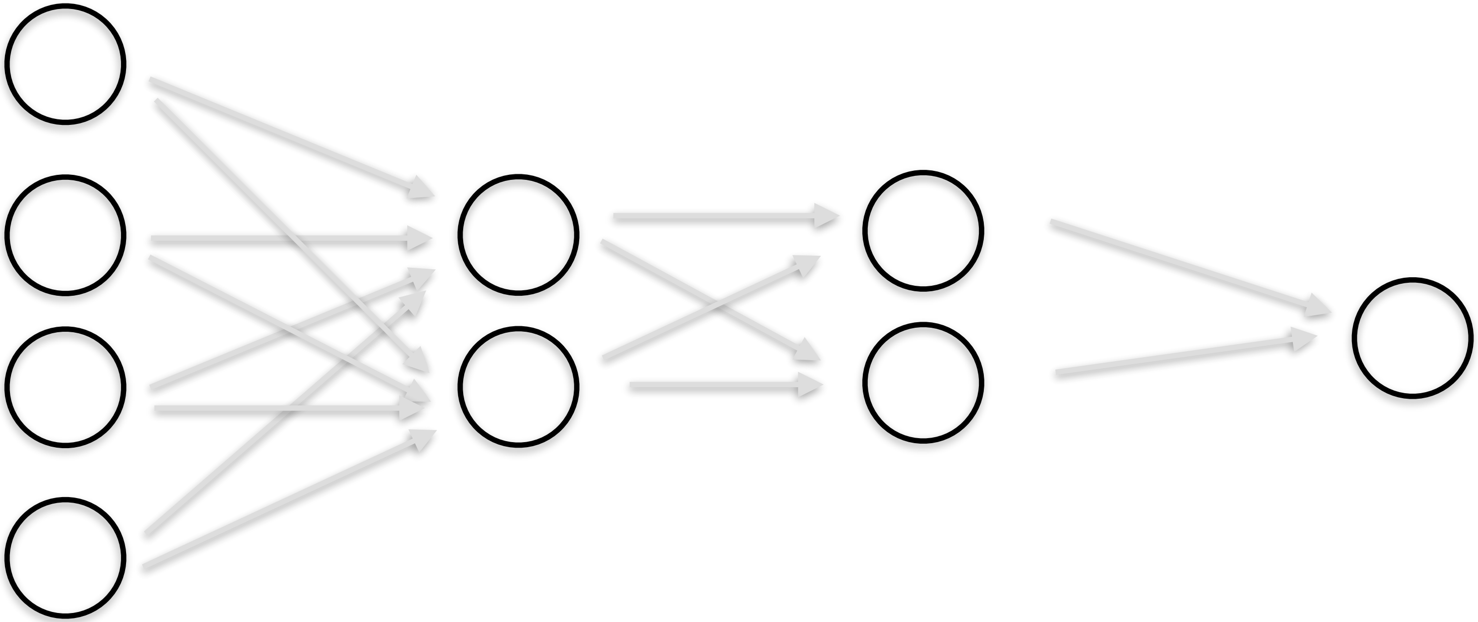
x_1 OR x_2



NOT x_1



We can chain these basic logical operators to create more complex systems and structures

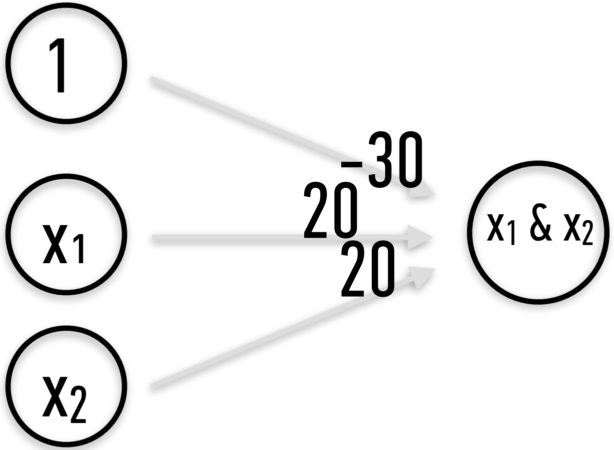


For instance, let's create a neural network representing **x_1 XOR x_2**

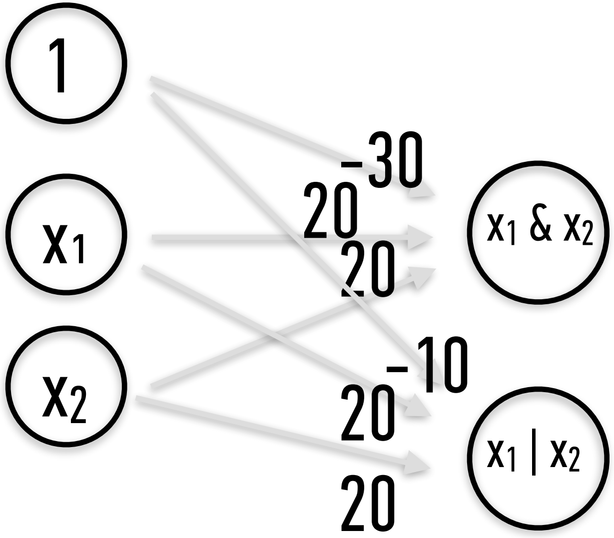
x_1 XOR x_2 = (x_1 OR x_2) AND NOT (x_1 AND x_2)

i.e. either x_1 or x_2, but not both

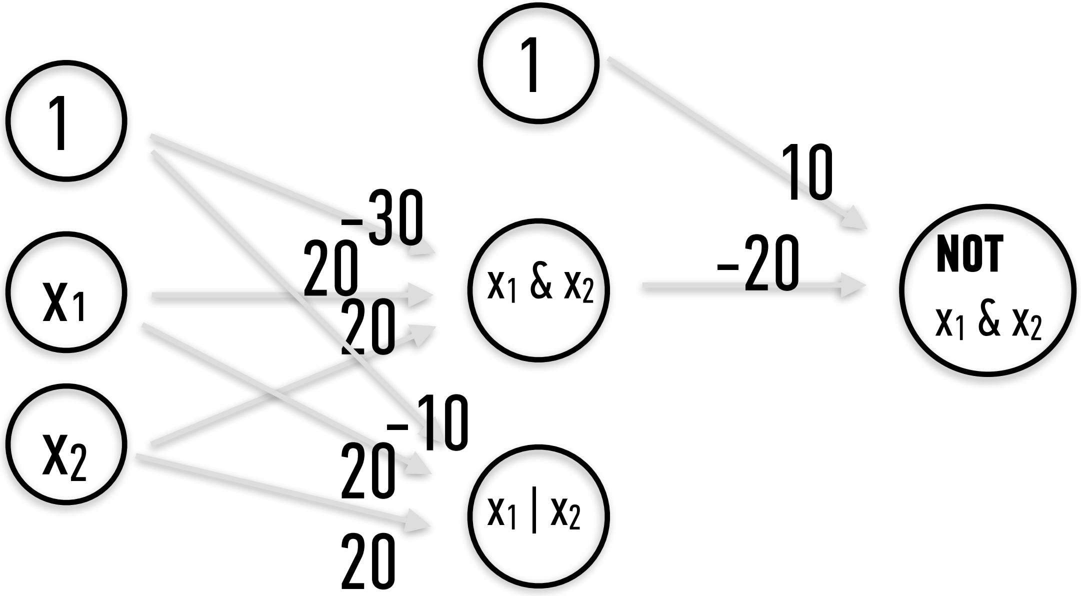
For instance, let's create a neural network representing x_1 **XOR** x_2



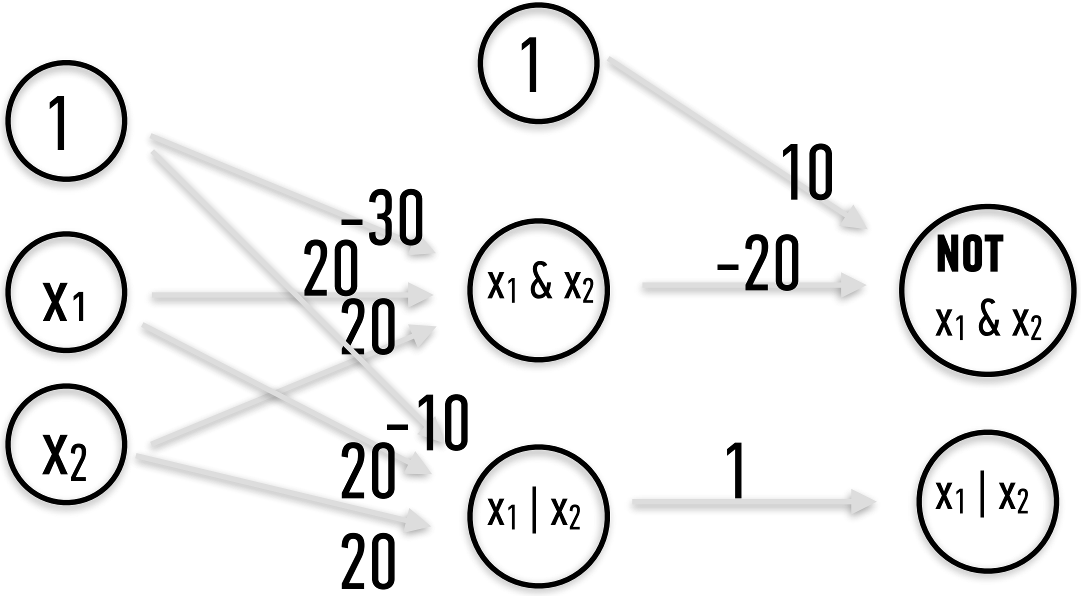
For instance, let's create a neural network representing x_1 XOR x_2



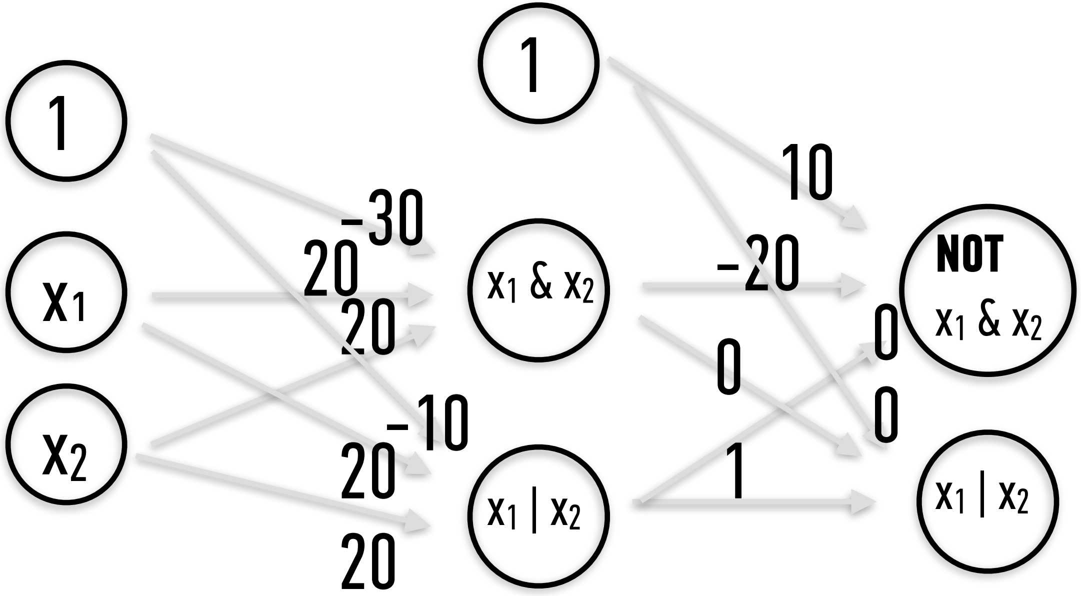
For instance, let's create a neural network representing x_1 XOR x_2



For instance, let's create a neural network representing **x_1 XOR x_2**

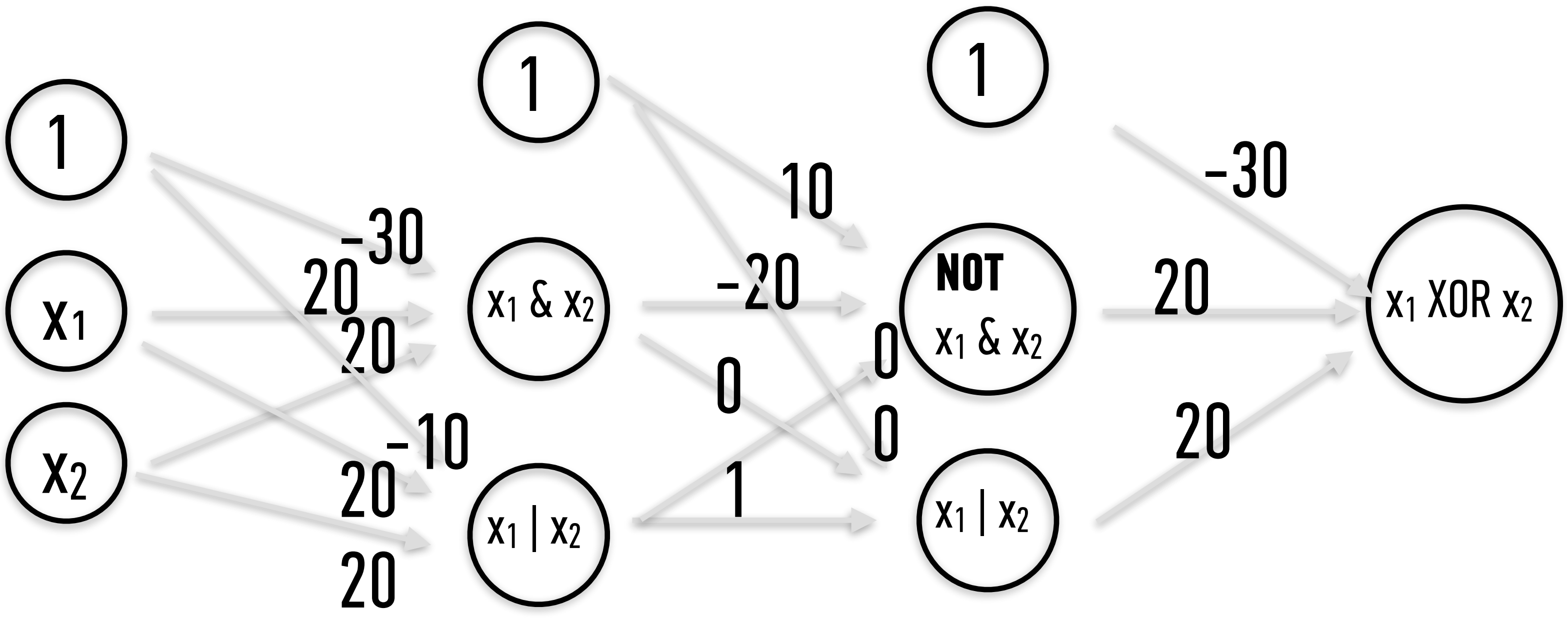


For instance, let's create a neural network representing **x_1 XOR x_2**



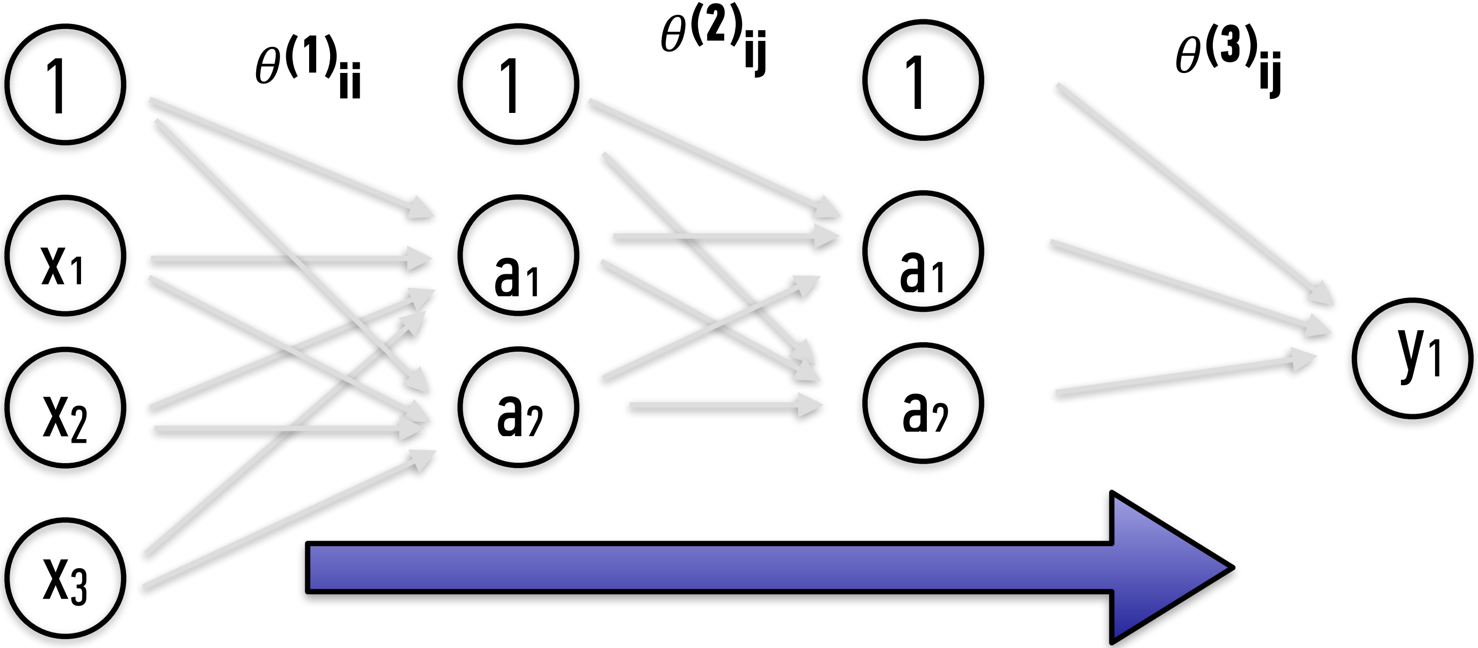
Missing edges are the same as weight coefficients of 0

For instance, let's create a neural network representing x_1 XOR x_2



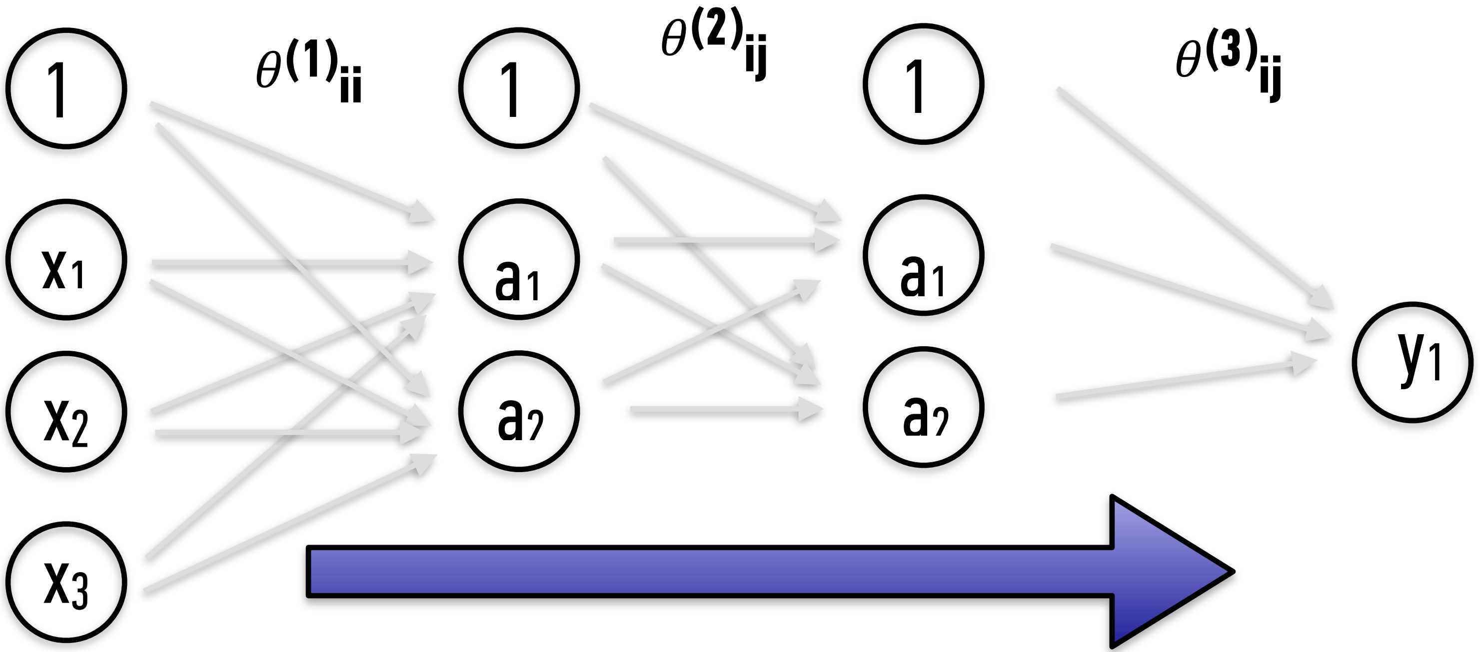
III. NEURAL NETWORK VARIETIES

The aforementioned networks are called ***feed forward networks***

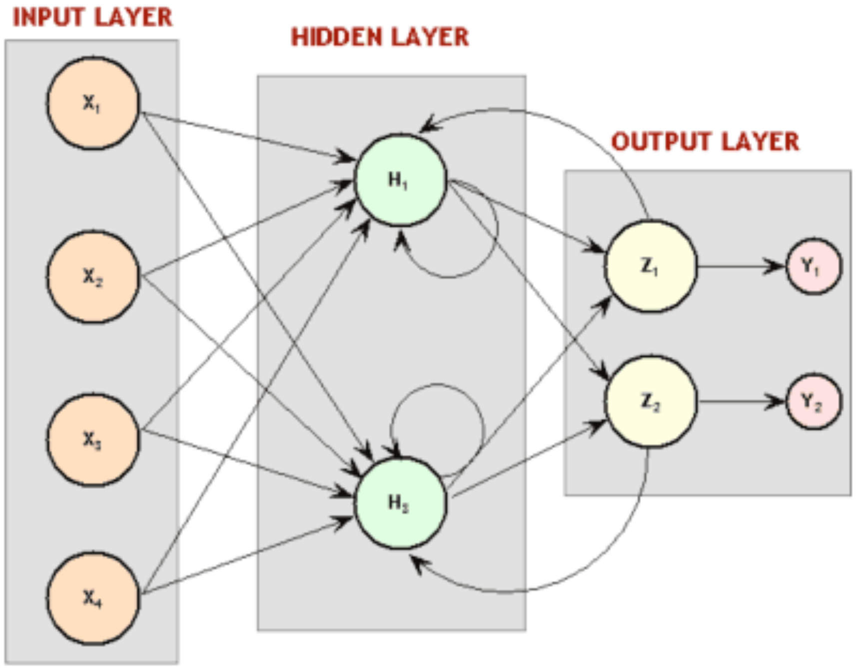


The aforementioned networks are called ***feed forward networks***

Deep learning or deep networks contain more than 1 hidden layer



*Recurrent networks contain **cycles** that help them to retain memory*



Recurrent networks contain ***cycles*** *that help them to retain memory*

-More realistic, but difficult to train

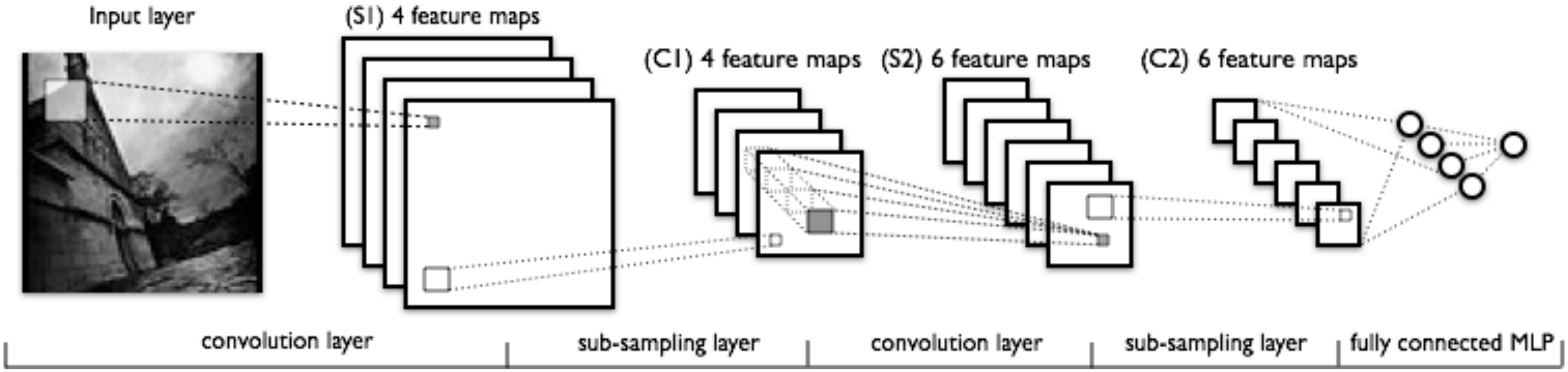
-Ilya Sutskever, 2011 - Recurrent network trained on billion characters from Wikipedia. Model can predict next character in sequence

-Google uses RNN in speech recognition http://googleresearch.blogspot.com/2015/09/google-voice-search-faster-and-more.html

*Recurrent networks contain **cycles** that help them to retain memory*

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.

Convolutional networks contain multiple layers that act as "feature extractors"



Convolutional networks contain multiple layers that act as "feature extractors"

-Typically used on images to reduce the complexity and curse of dimensionality

-Excellent for object detection

Other types:

-Symmetric neural networks - Recurrent networks with symmetric weights

-Hopfield networks - Symmetric network with no hidden units

-Boltzmann machine - Symmetric network with hidden units

IV. COST FUNCTION

Let's go back to a basic feed forward neural network

Let's go back to a basic feed forward neural network

Given some dataset with inputs X, and targets y, how do we solve for the weights and biases?

Let's go back to a basic feed forward neural network

Given some dataset with inputs X, and targets y, how do we solve for the weights and biases?

*Through minimizing the **cost function***

Neural network cost function:

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

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

Neural network cost function:

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

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

Cost J depends on all the matrices theta for each layer

Neural network cost function:

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

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

For all m samples,

For all k output nodes

Neural network cost function:

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

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

why K outputs? We can encode a multi class classification this way...

Neural network cost function:

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

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

wh

enc

cla

For instance, let's say you have a classification problem with 3 classes: **person**, **car**, and **bike**. The first node would be for person, second node for car, and third for bike.

Then for encoding y, we would use the following vector representations:

Person = (1, 0, 0)

Car = (0, 1, 0)

Bike = (0, 0, 1)

This is also called **one hot encoding**

Neural network cost function:

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

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

Cost for each incorrect sample at the output layer

Neural network cost function:

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

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

Cost for having weights that are too large (i.e. regularization)

V. BACK PROPAGATION

*We have this crazy cost function that relates errors to our weights (theta). Now, we can use **gradient descent** to reduce the cost as much as possible*

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

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

The informal way to implement gradient descent in neural networks is through a process called backpropagation

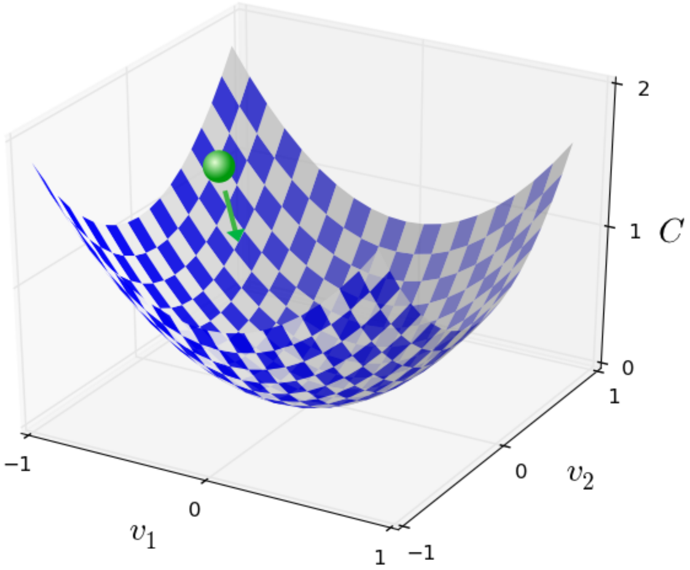
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1 - y_k^{(i)})\log(1 - (h_\theta(x^{(i)}))_k)\right]$$

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

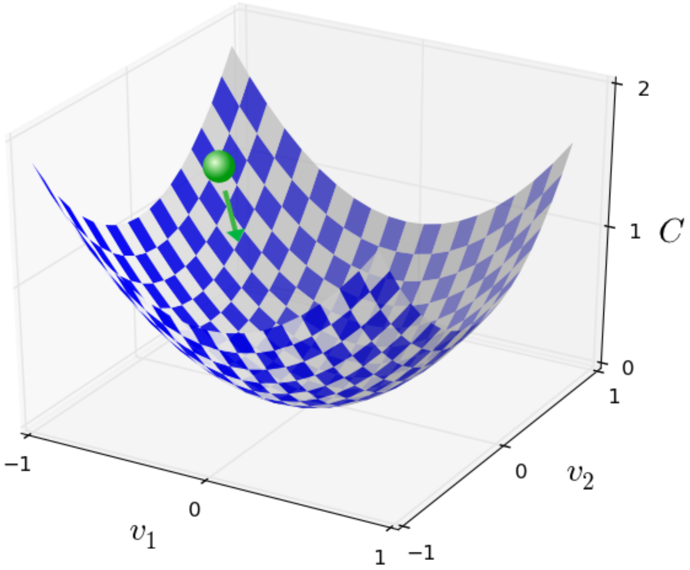
The informal way to implement gradient descent in neural networks is through a process called backpropagation

We won't go into the details, but you can think of backpropagation as a way to back out "which direction" to change a weight or bias based on how wrong an output is

Pretend there are just two variables on the x,y axis and the Cost J is on the z axis. By modulating x and y, we are able to increase or decrease the Cost.



Gradient descent through back propagation is the equivalent of putting a boll on the curve, and letting it roll down to the minimum



VI. CONSIDERATIONS

- *Neural networks are still considered pretty black box*
- *For deeper networks, requires a large amount of computational power (i.e. don't think about computing using a CPU)*
- *Vanishing gradients (somewhat solved with different activation functions)*
- *Choice of network architecture*
- *Hyperparameters (i.e how many nodes, regularization strength, weight initialization)*

VI. RESOURCES

Neural networks have had a (recently) explosive amount of depth to the field.

Learning backpropagation itself can take a few days to understand the gist of.

Then, there's recurrent neural networks, LSTM networks, convolutional networks, autoencoders, etc.

Libraries:

- *Tensorflow – Relatively new, released by Google https://www.tensorflow.org, sklearn wrapper called SKFlow*
- *Theano – More of a general purpose computing library, http://deeplearning.net/software/theano/*
- *Caffe – C++ deep learning mostly focused on vision, with wrappers http://caffe.berkeleyvision.org*
- *Torch – deep learning using Lua language*

*Today, we're going to try the monumental and hopefully successful task of getting **tensorflow** as well as **skflow** running on our systems*

Install Tensorflow:

conda install -c https://conda.anaconda.org/jjhelmus tensorflow=0.6.0

Install SKflow:

pip install skflow

Install Theano:

pip install Theano

More resources:

http://neuralnetworksanddeeplearning.com
http://karpathy.github.io/neuralnets/
http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
http://www.heatonresearch.com/book/
http://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
ftp://ftp.sas.com/pub/neural/FAQ.html

THAT’S IT!

- Exit Tickets: DAT1 - Lesson 19 - Neural Networks