Neural Network

Representation

JP @ 2B3E

Neurons & Brain

- Neural networks (NNs) were originally motivated by looking at machines which replicate the brain's functionality
  - Looked at here as a machine learning technique
- Origins
  - To build learning systems, why not mimic the brain?
  - Used a lot in the 80s and 90s
  - Popularity diminished in late 90s
  - Recent major resurgence
    - NNs are computationally expensive, so only recently large scale neural networks became computationally feasible

Neurons & Brain

- Brain
  - Does loads of crazy things
    - Hypothesis is that the brain has a single learning algorithm
  - Evidence for hypothesis
    - Auditory cortex --> takes sound signals
      - If you cut the wiring from the ear to the auditory cortex
      - Re-route optic nerve to the auditory cortex
      - Auditory cortex learns to see
    - Somatosensory context (touch processing)
      - If you rewrite optic nerve to somatosensory cortex then it learns to see
  - With different tissue learning to see, maybe they all learn in the same way
    - Brain learns by itself

Neurons & Brain

[Brainport https://www.youtube.com/watch?v=CNR2gLKnd0g](https://www.youtube.com/watch?v=CNR2gLKnd0g)

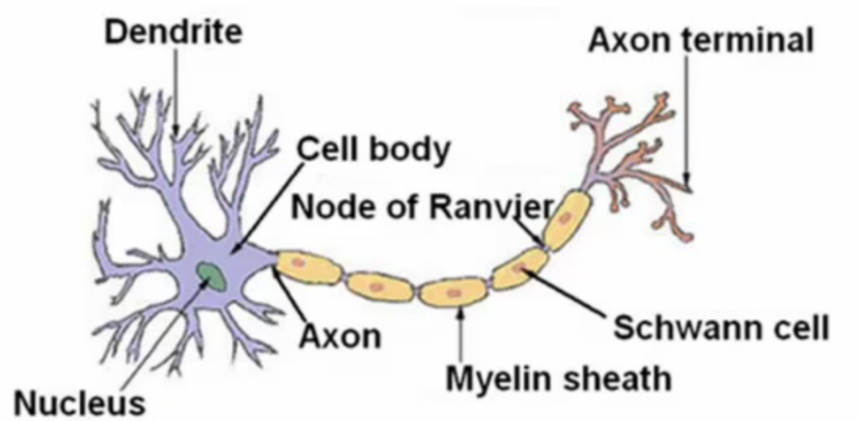
Human echolocation
https://www.youtube.com/watch?v=A8lztr1tu4o

Haptic belt
https://www.youtube.com/watch?v=dqijP8si9rc

- Other examples
    - Seeing with your tongue
        - Brainport
            - Grayscale camera on head
            - Run wire to array of electrodes on tongue
            - Pulses onto tongue represent image signal
            - Lets people see with their tongue
    - Human echolocation
        - Blind people being trained in schools to interpret sound and echo
        - Lets them move around
    - Haptic belt direction sense
        - Belt which buzzes towards north
        - Gives you a sense of direction
- Brain can process and learn from any data source

Neurons

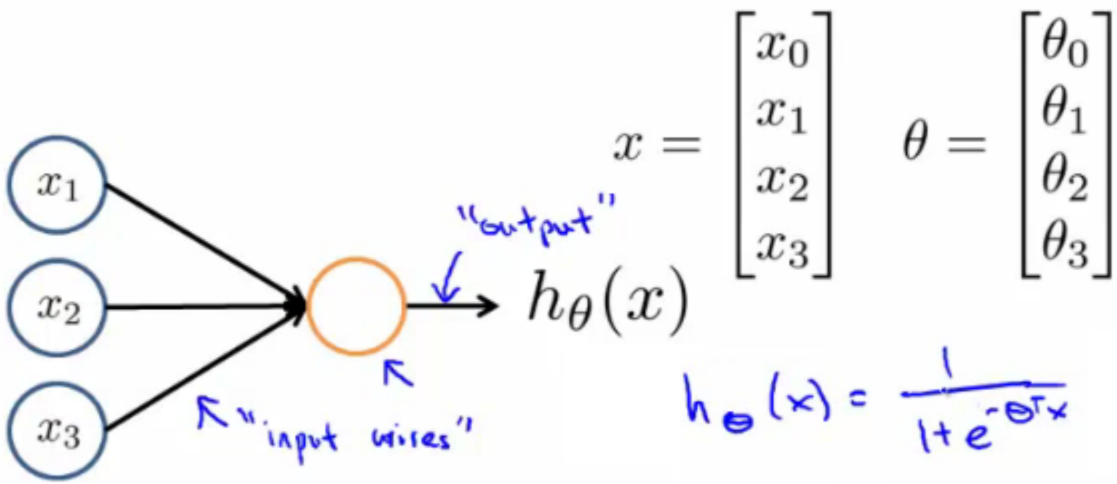
- Components
  - Cell body, # of inputs (dendrites), output wire (axon)
- Simple level
  - Neuron gets one or more inputs through dendrites
  - Does processing
  - Sends output down axon
- Neurons communicate through electric spikes
  - Pulse of electricity via axon to another neuron



Artificial Neurons

주의: notation이 "Deep Learning from Scratch"와 다름

- A neuron is a logistic unit
  - Feed input via input wires
  - Logistic unit does computation
  - Sends output down output wires
- That logistic computation is just like logistic regression hypothesis calculation

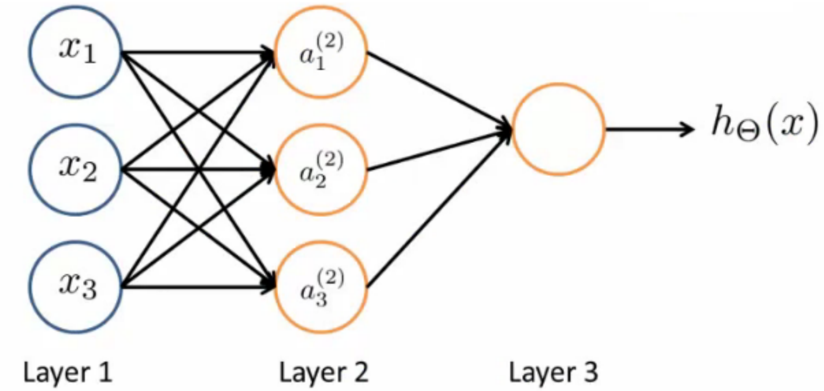


Artificial Neural Network

- Very simple model of a neuron's computation
  - Often good to include an x0 input - the bias unit
    - This is equal to 1
- This is an artificial neuron with a sigmoid (logistic) activation function
  - Ө vector may also be called the weights of a model
- The above diagram is a single neuron
  - Next, we have a group of neurons strung together

Artificial Neural Network

- input is x_1, x_2 and x_3
  - We could also call input activation on the first layer - i.e. (a_1^1, a_2^1 and a_3^1)
  - Three neurons in layer 2 (a_1^2, a_2^2 and a_3^2)
  - Final neuron which produces the output
    - Which again we *could* call a_1^3
- First layer is the **input layer**
- Final layer is the **output layer** - produces value computed by a hypothesis
- Middle layer(s) are called the **hidden layers**
  - You don't observe the values processed in the hidden layer
  - Can have many hidden layers



Neural networks - notation

- **$a_i^{(j)}$ - activation of unit *i* in layer *j***
  - So, a_1^2 - is the **activation** of the 1st unit in the second layer
  - By activation, we mean the value which is computed and output by that node
- **$\Theta^{(j)}$ - matrix of parameters controlling the function mapping from layer *j* to layer *j* + 1**
  - Parameters for controlling **mapping** from one layer to the next
  - If network has
    - s_j units in layer *j* and
    - s_{j+1} units in layer j + 1
    - Then Θ^j will be of dimensions $[s_{j+1} \times s_j + 1]$
      - Because
        - s_{j+1} is equal to the number of units in layer (j + 1)
        - (s_j + 1) is equal to the number of units in layer j, plus an additional unit

Neural networks - notation

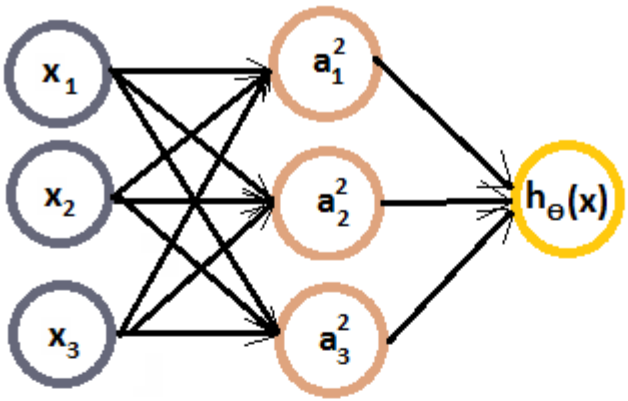
- Looking at the Ɵ matrix
  - Column length is the number of units in the following layer
  - Row length is the number of units in the current layer + 1 (because we have to map the bias unit)
  - So, if we had two layers - 101 and 21 units in each
    - Then $Ɵ^j$ would be = [21 x 102]

Neural networks - calculation

- What are the computations which occur?
  - We have to calculate the activation for each node
  - That activation depends on
    - The input(s) to the node
    - The parameter associated with that node (from the Θ vector associated with that layer)

Neural networks - calculation

- Below we have an example of a network, with the associated calculations for the four nodes below



$$
\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\
a_2^{(2)} &= g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\
a_3^{(2)} &= g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \\
h_\Theta(x) = a_1^{(3)} &= g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})
\end{aligned}
$$

Neural networks - calculation

- calculate each of the layer-2 activations based on the input values with the bias term (which is equal to 1)
  - i.e. x_0 to x_3
- then calculate the final hypothesis (i.e. the single node in layer 3) using exactly the same logic
  - except the input is not x values, but the activation values from the preceding layer

Neural networks - calculation

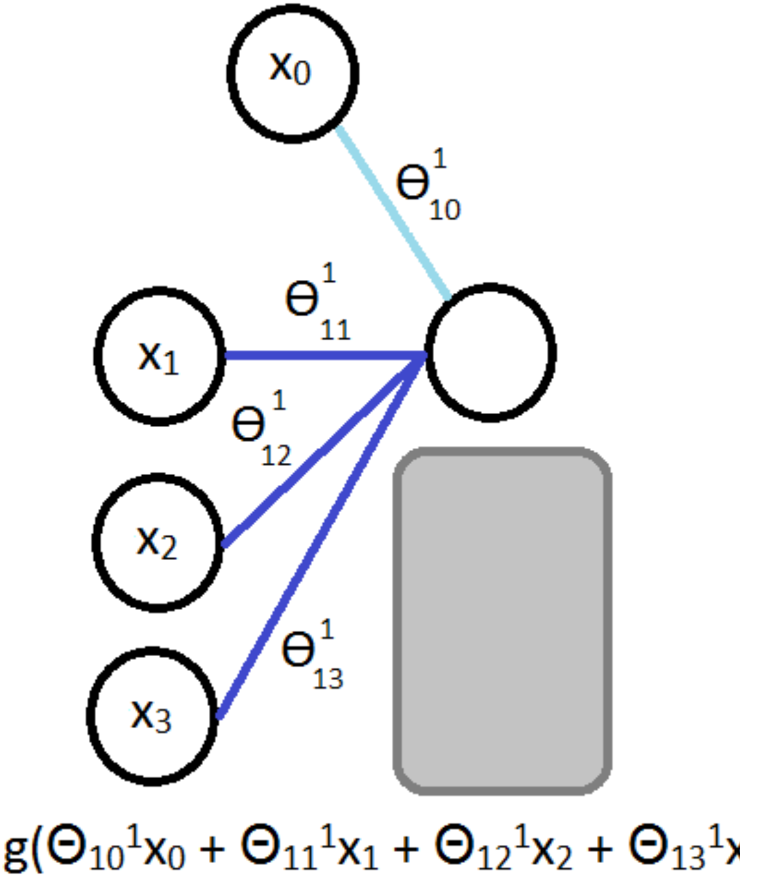
- The activation value on each hidden unit (e.g. a_1^2) is equal to the sigmoid function applied to the linear combination of inputs
  - Three input units
    - So $\Theta^{(1)}$ is the matrix of parameters governing the mapping of the input units to hidden units
      - $\Theta^{(1)}$ here is a [3 x 4] dimensional matrix
  - Three hidden units
    - Then $\Theta^{(2)}$ is the matrix of parameters governing the mapping of the hidden layer to the output layer
      - $\Theta^{(2)}$ here is a [1 x 4] dimensional matrix (i.e. a row vector)
  - One output unit

Neural networks - calculation

- Something conceptually important is that
  - **Every input/activation goes to every node in following layer**
  - Which means each "layer transition" uses a matrix of parameters with the following significance
    - θ_{ji}^{l}
      - j (first of two subscript numbers)= ranges from 1 to the number of units in layer l+1
      - i (second of two subscript numbers) = ranges from 0 to the number of units in layer l
      - l is the layer you're moving FROM

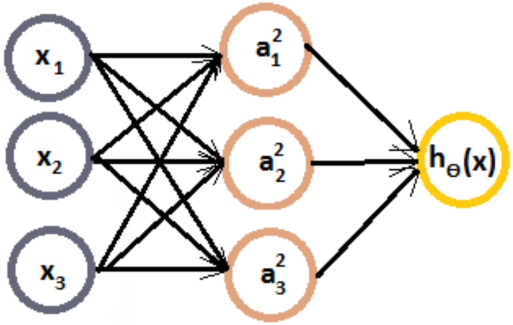
Neural networks - calculation

- For exampleΘ_{13}^{1} = means
  - **1** - we're mapping to node 1 in layer l+1
  - **3** - we're mapping from node 3 in layer l
  - **1** - we're mapping from layer 1



Model Representation

- *Here we'll look at how to carry out the computation efficiently through a vectorized implementation.*
- *We'll also consider why NNs are good and how we can use them to learn complex non-linear things*



$$
\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\
a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\
a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \\
h_\Theta(x) = a_1^{(3)} &= g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})
\end{aligned}
$$

Model Representation

- Define some additional terms
  - $z_1^2 = \Theta_{10}^1 x_0 + \Theta_{11}^1 x_1 + \Theta_{12}^1 x_2 + \Theta_{13}^1 x_3$
  - Which means that
    - $a_1^2 = g(z_1^2)$
  - superscript numbers are the layer associated
- Similarly, we define the others as
  - z_2^2 and z_3^2
  - These values are just a linear combination of the values
- If we look at the block we just redefined
  - We can vectorize the neural network computation
  - So lets define
    - x as the feature vector x
    - z^2 as the vector of z values from the second layer

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

Model Representation

- z^2 is a 3x1 vector
- We can vectorize the computation of the neural network as follows in two steps
  - $z^2 = \Theta^{(1)}x$
    - i.e. $\Theta^{(1)}$ is the matrix defined above
    - x is the feature vector
  - $a^2 = g(z^{(2)})$
    - To be clear, z^2 is a 3x1 vecor
    - a^2 is also a 3x1 vector
    - g() applies the sigmoid (logistic) function element wise to each member of the z^2 vector
- To make the notation with input layer make sense;
  - $a^1 = x$
    - a^1 is the activations in the input layer
    - Obviously the "activation" for the input layer is just the input!
  - So we define x as a^1 for clarity
    - So
      - a^1 is the vector of inputs
      - a^2 is the vector of values calculated by the $g(z^2)$ function

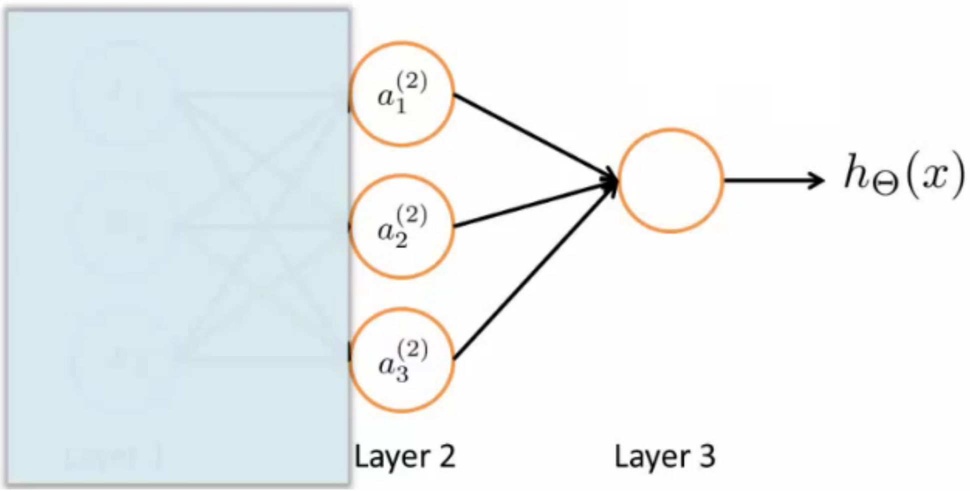
Model Representation

- Having calculated then z^2 vector, we need to calculate a_0^2 for the final hypothesis calculation

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

- To take care of the extra bias unit add $a_0^2 = 1$
  - So add a_0^2 to a^2 making it a 4x1 vector
- So,
  - $z^3 = \Theta^2 a^2$
    - This is the inner term of the above equation
  - $h_\Theta(x) = a^3 = g(z^3)$
- This process is also called **forward propagation**
  - Start off with activations of input unit
    - i.e. the x vector as input
  - Forward propagate and calculate the activation of each layer sequentially
  - This is a vectorized version of this implementation

Learning



- Layer 3 is a logistic regression node
- The hypothesis output = $g(\Theta_{10}^2 a_0^2 + \Theta_{11}^2 a_1^2 + \Theta_{12}^2 a_2^2 + \Theta_{13}^2 a_3^2)$
- This is just logistic regression
  - The only difference is, instead of input a feature vector, the features are just values calculated by the hidden layer

Learning

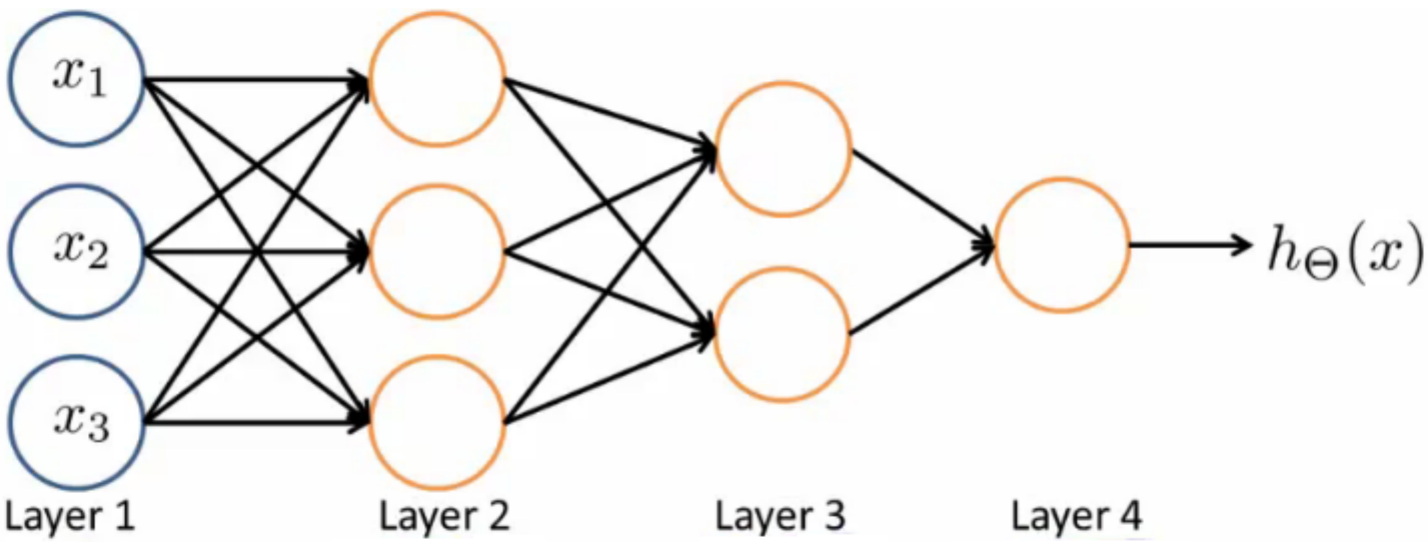
- The features a_1^2, a_2^2, and a_3^2 are calculated/learned - not original features
- So the mapping from layer 1 to layer 2 (i.e. the calculations which generate the a^2 features) is determined by another set of parameters - Θ^1
  - So instead of being constrained by the original input features, a neural network can learn its own features to feed into logistic regression
  - Depending on the Θ^1 parameters you can learn some interesting things
    - Flexibility to learn whatever features it wants to feed into the final logistic regression calculation
      - compared to previous logistic regression, calculate your own exciting features to define the best way to classify or describe something
      - the hidden layers role: feed the hidden layers our input values, and let them learn whatever gives the best final result to feed into the final output layer

Learning

- As well as the networks already seen, other architectures (topology) are possible
  - More/less nodes per layer
  - More layers
  - Once again, layer 2 has three hidden units, layer 3 has 2 hidden units by the time you get to the output layer you get very interesting non-linear hypothesis

Learning

- Some of the intuitions here are complicated and hard to understand
  - In the following lectures we're going to go though a detailed example to understand how to do non-linear analysis

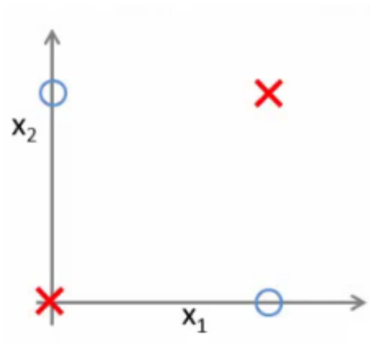


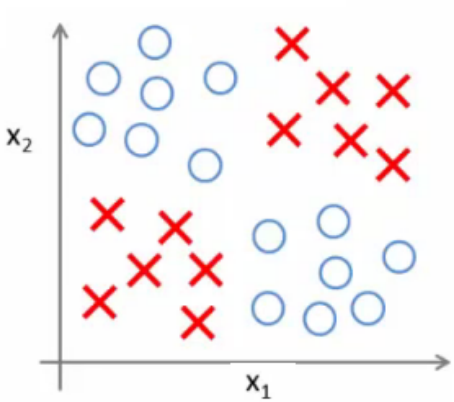
Example – complex, nonlinear function

- **Non-linear classification: XOR/XNOR**
    - **x_1, x_2 are binary**

$y = x_1$ XOR x_2

x_1 XNOR x_2





Where XNOR = NOT (x_1 XOR x_2)

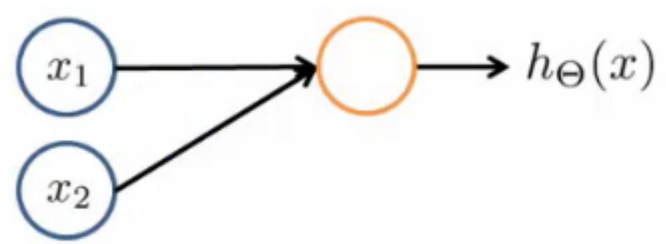
Positive examples when both are true and both are false

Let's start with something a little more straight forward...

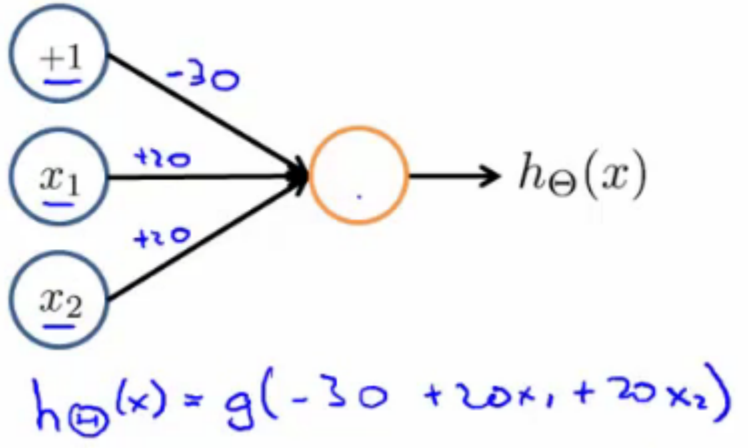
Don't worry about how we're determining the weights (Θ values) for now - just get a flavor of how NNs work

Neural Network example 1: AND function

- Can we get a one-unit neural network to compute this logical AND function? (*probably...*)
    - Add a bias unit
    - Add some weights for the networks
        - What are weights?
            - Weights are the parameter values which multiply into the input nodes (i.e. Θ)



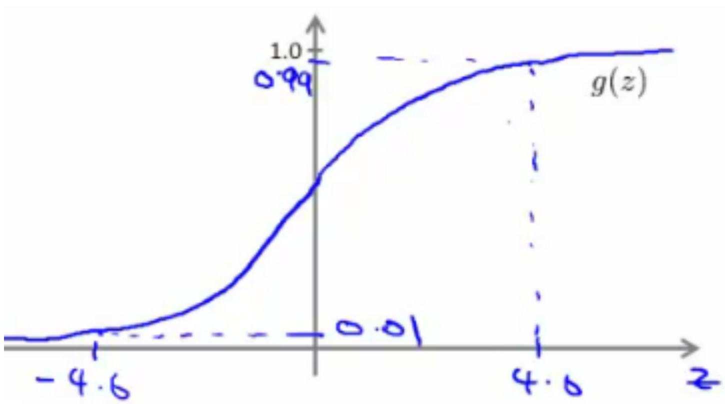
Neural Network example 1: AND function



- Sometimes it's convenient to add the weights into the diagram. These values are in fact just the Θ parameters so
  - $\Theta_{10}^1 = -30$
  - $\Theta_{11}^1 = 20$
  - $\Theta_{12}^1 = 20$
- To use our original notation

Neural Network example 1: AND function

- Look at the four input values



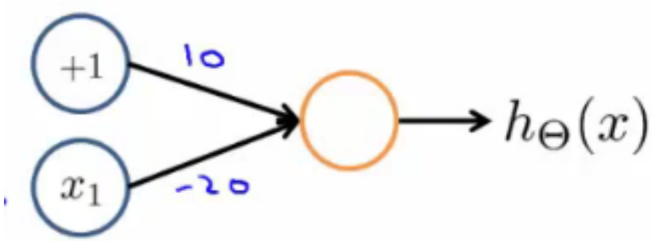
Sigmoid function (reminder)

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

- So, as we can see, when we evaluate each of the four possible input, only (1,1) gives a positive output

Neural Network example 2: NOT function

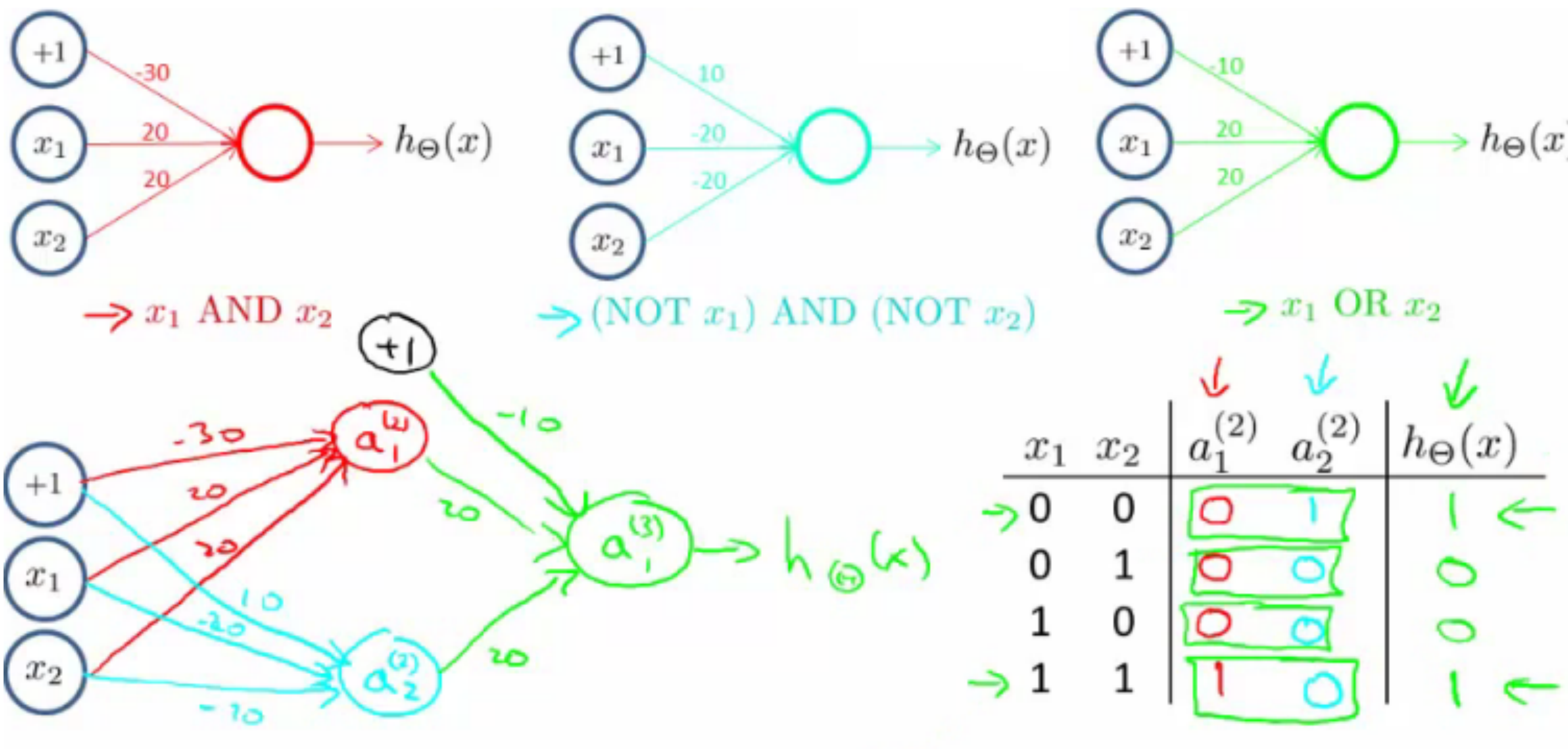


x_1	$h_\Theta(x)$
0	$g(10) \approx \boxed{1}$
1	$g(-10) \approx \boxed{0}$

Neural Network example 3: XNOR function

- So how do we make the XNOR function work? XNOR is short for NOT XOR
  - i.e. NOT an exclusive or, so either go big (1,1) or go home (0,0)
- So we want to structure this so the input which produce a positive output are
  - AND (i.e. both true)
    OR
  - Neither (which we can shortcut by saying not only one being true)

Neural Network example 3: XNOR function

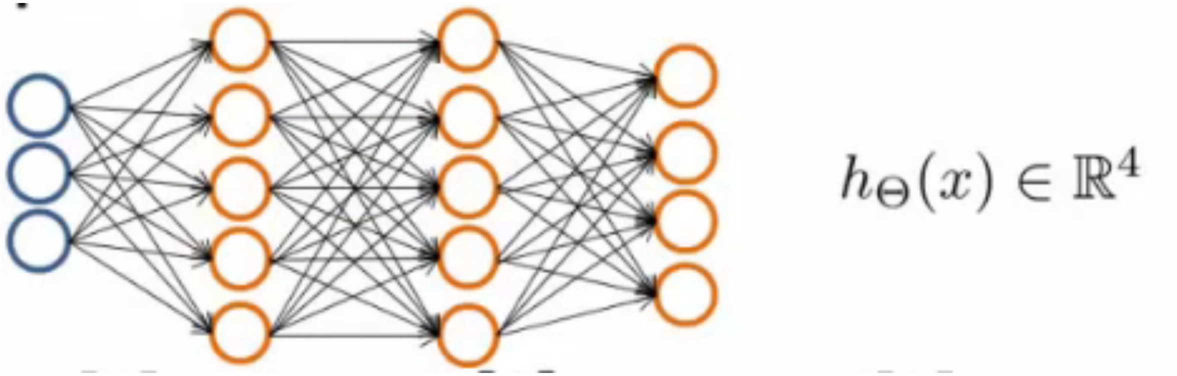


Multi-class classification

- Multiclass classification is, unsurprisingly, when you distinguish between more than two categories (i.e. more than 1 or 0)
- With handwritten digital recognition problem - 10 possible categories (0-9)
  - How do you do that?
  - Done using an extension of one vs. all classification
- Recognizing pedestrian, car, motorbike or truck
  - Build a neural network with four output units
  - Output a vector of four numbers
    - 1 is 0/1 pedestrian
    - 2 is 0/1 car
    - 3 is 0/1 motorcycle
    - 4 is 0/1 truck
  - When image is a pedestrian get [1,0,0,0] and so on

Multi-class classification

- Just like one vs. all described earlier
  - Here we have four logistic regression classifiers



Multi-class classification

- Training set here is images of our four classifications
  - While previously we'd written y as an integer {1,2,3,4}
  - Now represent y as

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