THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

COMP4434 Big Data Analytics

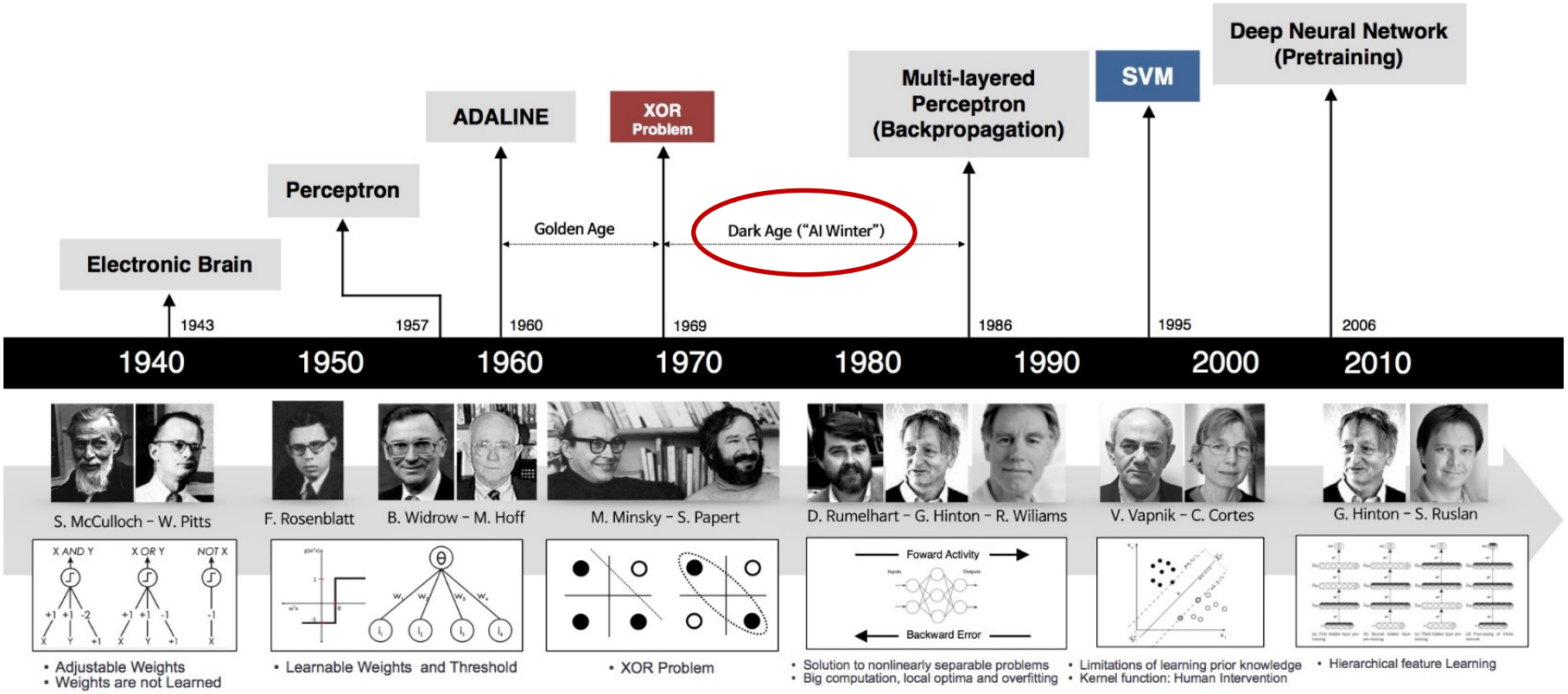
Lecture 8 Multilayer Perceptron & Backpropagation

HUANG Xiao

xiaohuang@comp.polyu.edu.hk

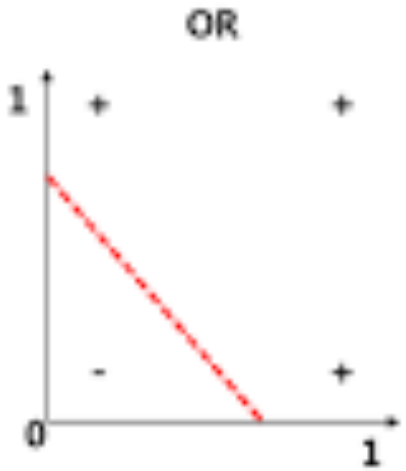
Neural Network

- Very powerful, but dumped for decades because of its complexity and large computation resources
  - First AI Winter (approximately 1974–1980); Second AI Winter (1987–2000)
- Resurrect in recent years because of much higher computation power (e.g., GPU) and much more training examples (big data)

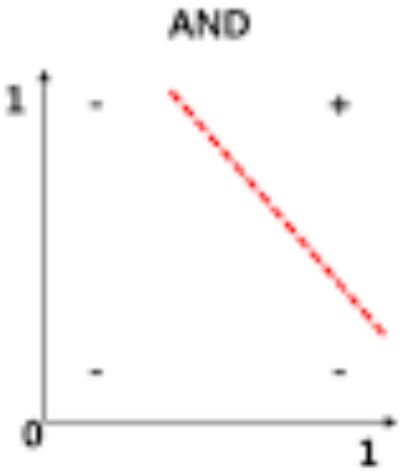


Source: https://sefiks.com/2017/10/14/evolution-of-neural-networks/

XOR Problem



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



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

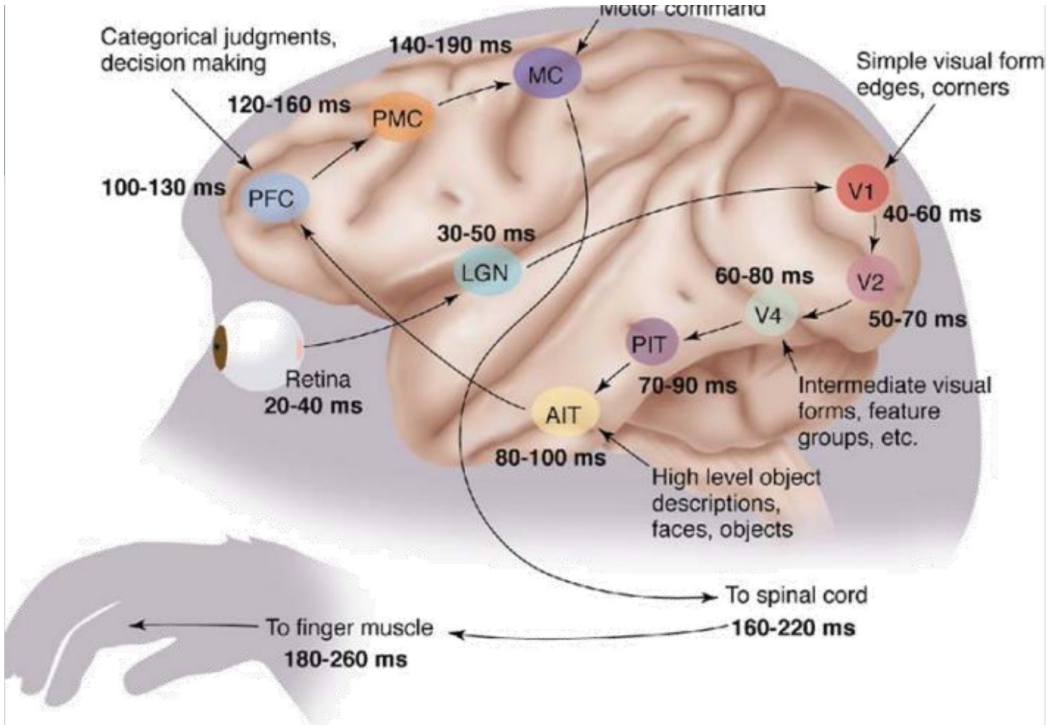
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Applications

- Computer Vision
  - Classifying large number of images in Google Image
  - Search for “cats”
  - Auto labeling an image
  - Face recognition
- Speech Recognition
  - iPhone Siri
- Natural Language Processing
  - ChatGPT

Inspired by Human Brain

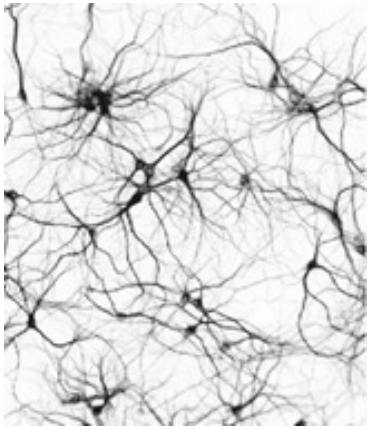
- Our brain has lots of neurons connected together
- Human brain is a graph/network of 100B nodes and 700T edges

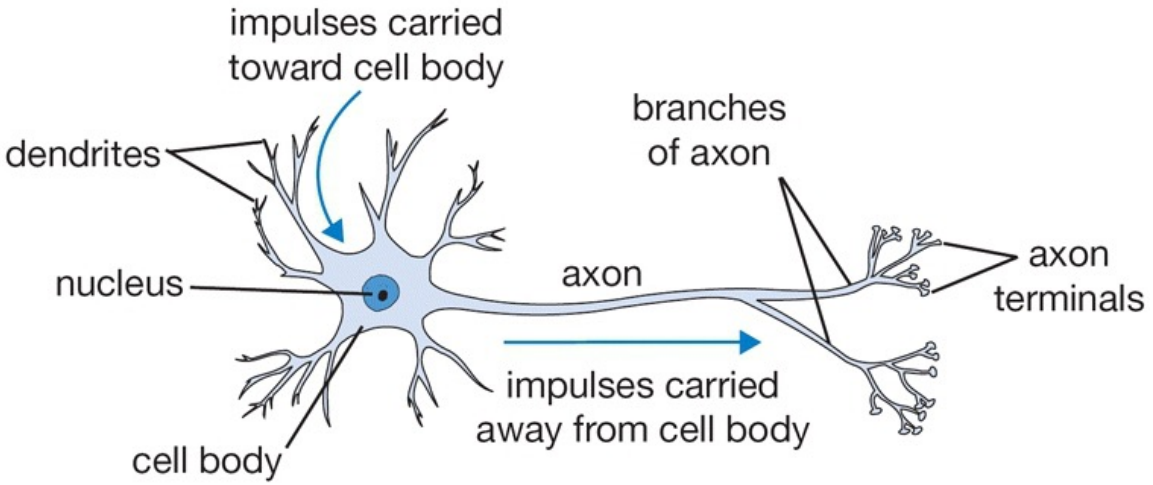


- The strength of the connections between neurons represents long term knowledge

Model

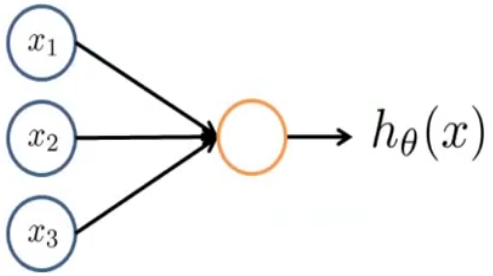
- It learns new features from your input features
- Its architecture is based on our brain structure
- The axon terminal of one neuron connects with the dendrite of another neuron which consists a quite complicated neural network

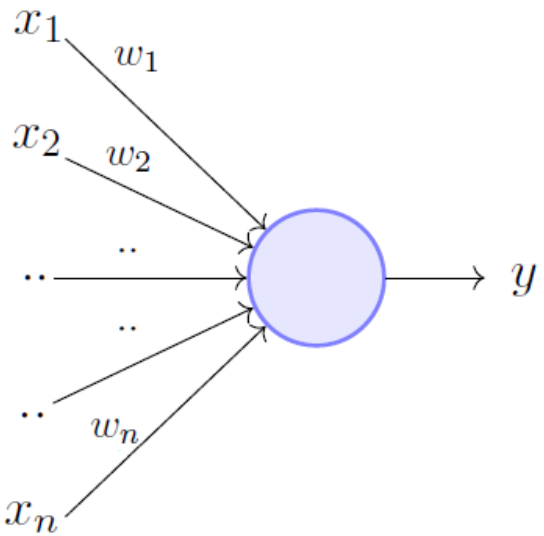




Perceptron: The Artificial Neuron

- x_i are input nodes and $h_\theta(x)$ is the output
- Node that takes the input represents the body of Neuron





$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

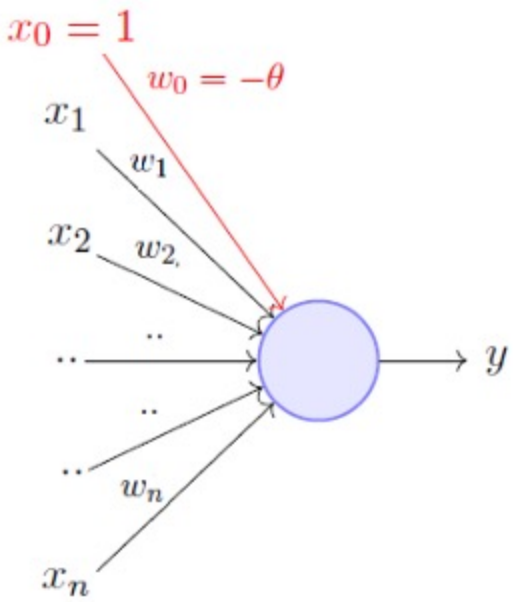
Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

A Linear Neuron

- Take a weighted sum of the inputs and set the output as one only when the sum is more than an arbitrary threshold θ
- Threshold is learn-able by adding an input with the weight $-\theta$



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

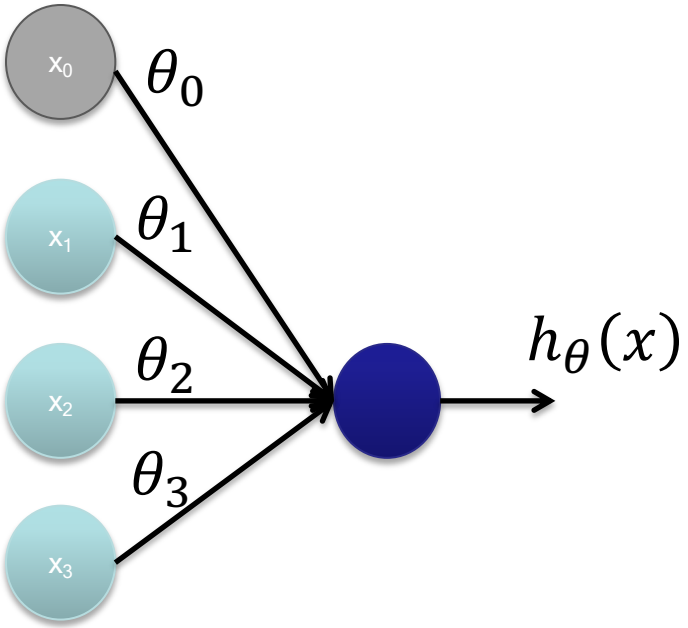
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

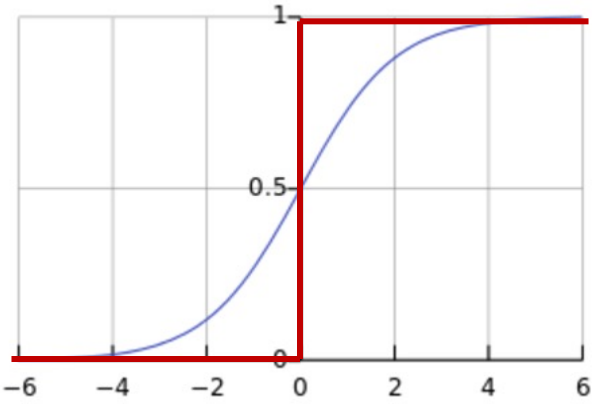
$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

Linear vs Logistic Neuron

- Linear: $h_\theta(x) = \sum(\theta_i x_i) = \theta^T x$ $\quad y = \begin{cases} 1 & h_\theta(x) \geq 0 \\ 0 & h_\theta(x) < 0 \end{cases}$
- Logistic: $h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$ $\quad y = \begin{cases} 1 & h_\theta(x) \geq 0.5 \\ 0 & h_\theta(x) < 0.5 \end{cases}$

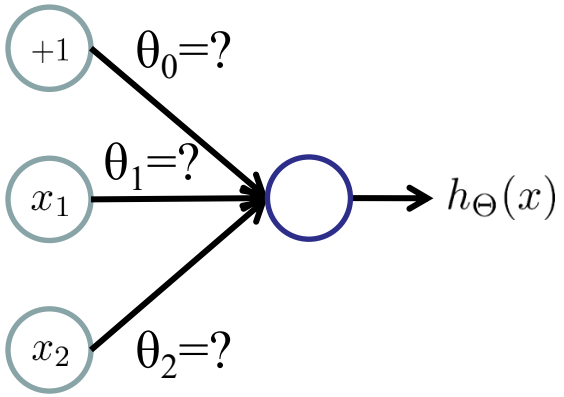
Activation function





AND Example

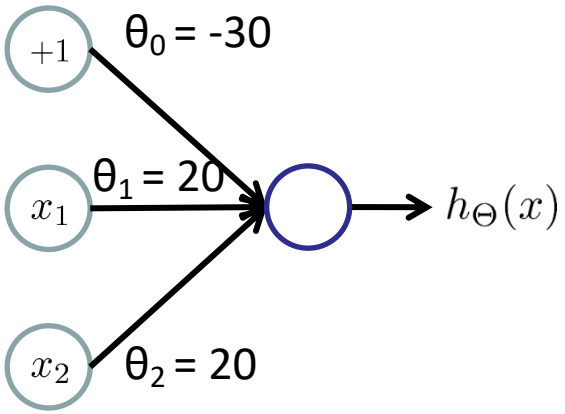
- AND: $y = x_1 \wedge x_2$



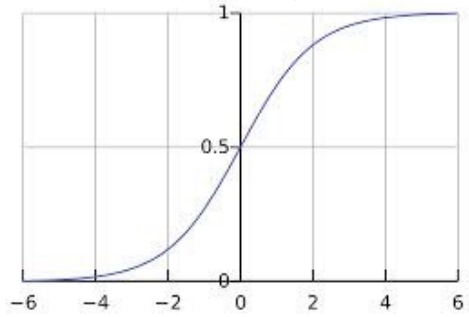
x_1	x_2	h
0	0	0
0	1	0
1	0	0
1	1	1

AND Example

- AND: $y = x_1 \wedge x_2$



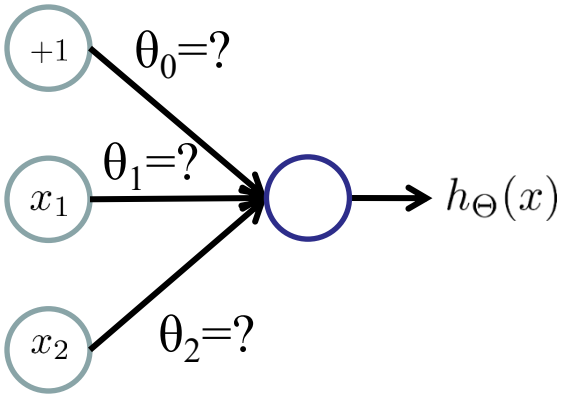
$$g(z) = \frac{1}{1+e^{-z}}$$



x_1	x_2	$g(\theta_0+\theta_1 x_1+\theta_2 x_2)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

OR Example

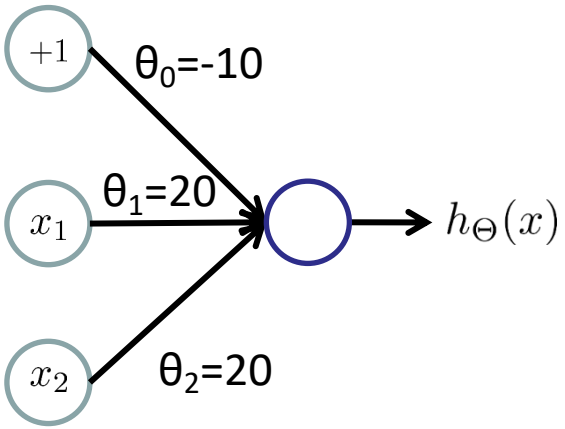
- OR: $y = x_1 \vee x_2$



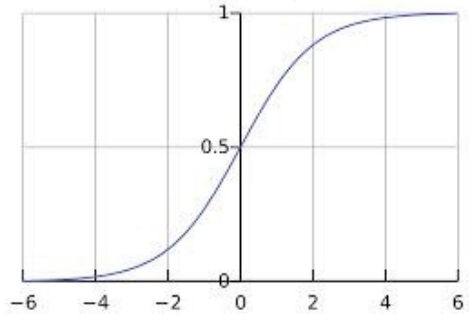
x_1	x_2	h
0	0	0
0	1	1
1	0	1
1	1	1

OR Example

- OR: $y = x_1 \vee x_2$



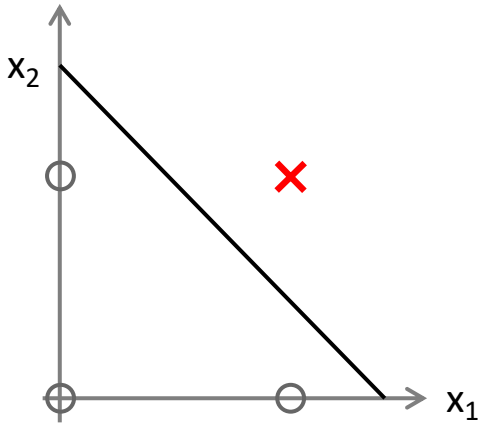
$$g(z) = \frac{1}{1+e^{-z}}$$

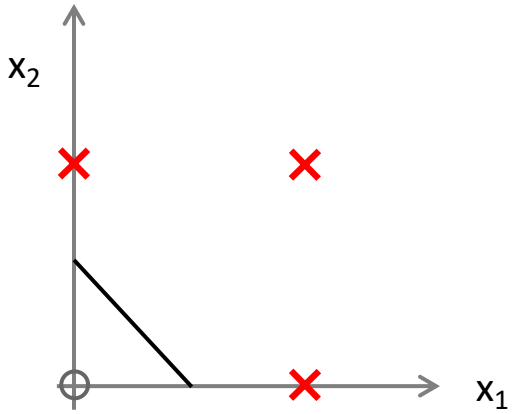


x_1	x_2	$g(\theta_0+\theta_1 x_1+\theta_2 x_2)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

Linear Hypothesis

- AND: $-30+20x_1+20x_2 = 0$
- OR: $-10+20x_1+20x_2 = 0$

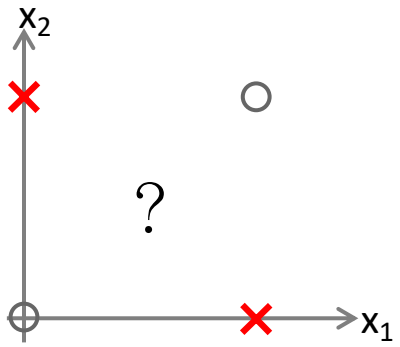




In addition, NOT and a great number of Boolean functions can also be represented by single-layer neural networks.

Non-linear Hypothesis

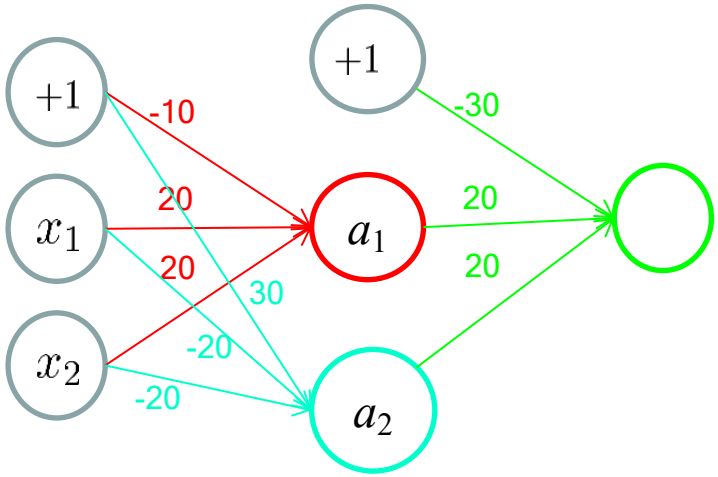
- XOR: However, you can't find a straight line to separate the two classes



x_1	x_2	h
0	0	0
0	1	1
1	0	1
1	1	0

XOR Example

x_1	x_2	a_1	a_2	h
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

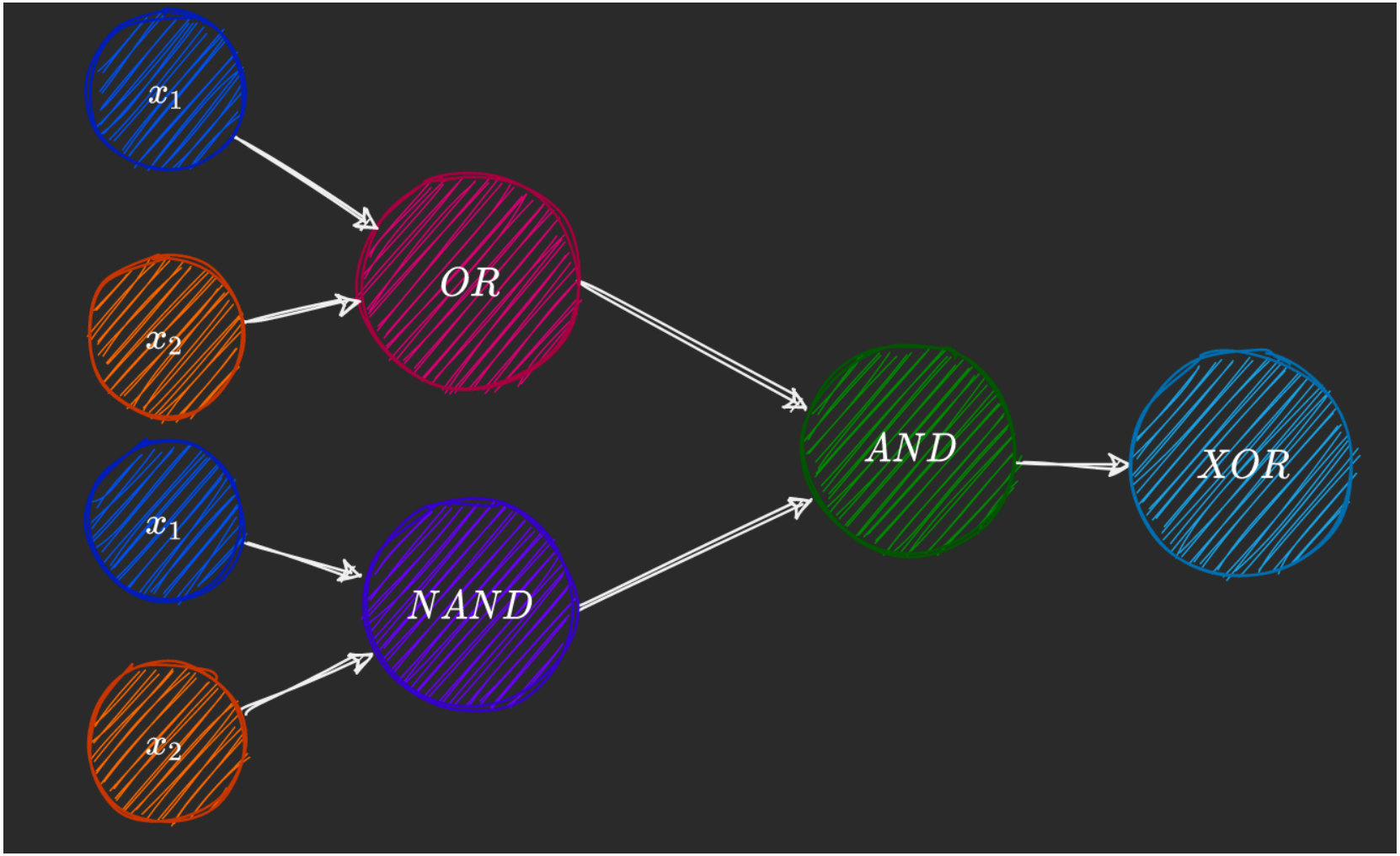


$$a_1 = g(-10 + 20x_1 + 20x_2)$$

$$a_2 = g(30^{-}20x_1^{-}20x_2)$$

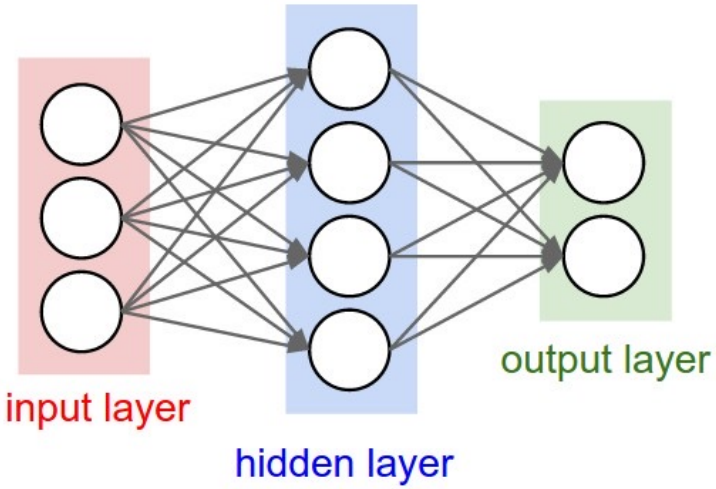
$$h(x) = g(-30 + 20a_1 + 20a_2) = g(-30 + 20g(-10 + 20x_1 + 20x_2) + 20g(30^{-}20x_1^{-}20x_2))$$

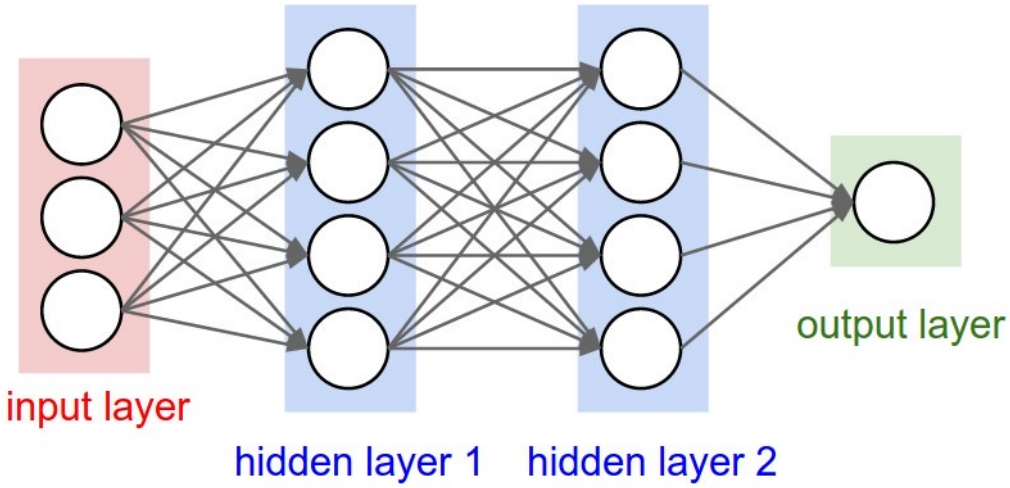
Intuition



https://towardsdatascience.com/how-neural-networks-solve-the-xor-problem-59763136bdd7

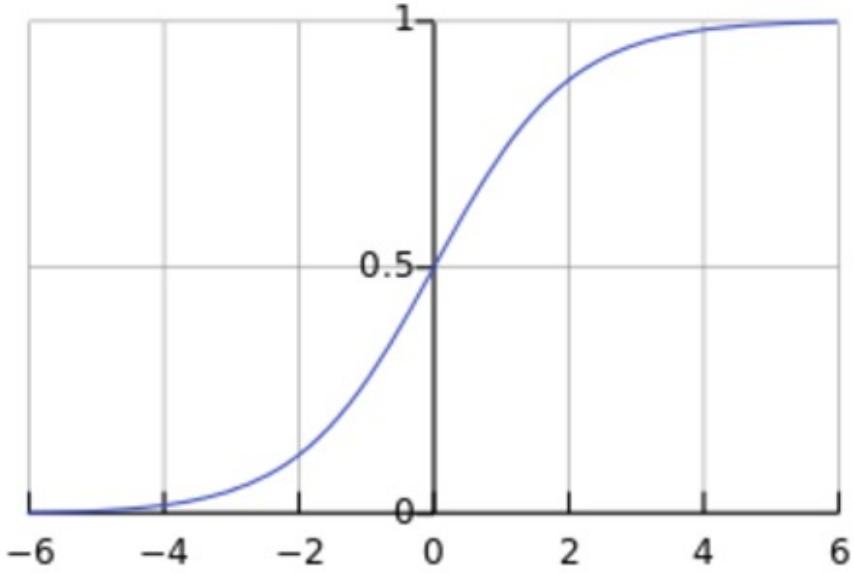
Neural Network





Recap: Logistic Regression Classifier

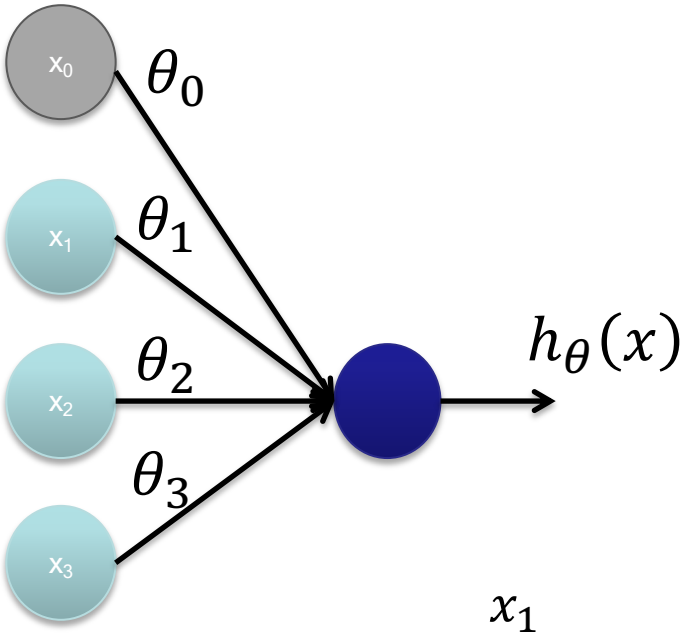
- $h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}} \in (0,1)$
- Predict $y = 1$ when $h_\theta(x) \geq 0.5$, i.e., $\theta^T x \geq 0$
- Predict $y = 0$ when $\theta^T x < 0$

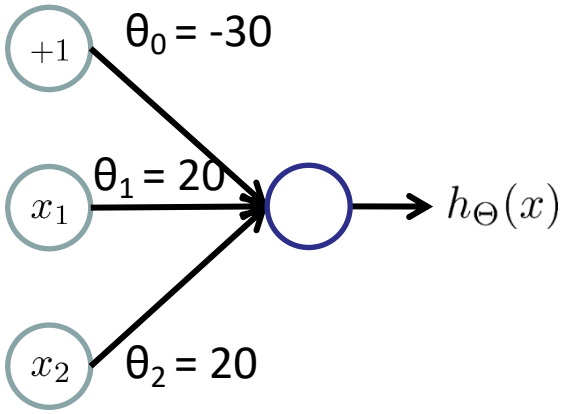


Recap: One-Layer Neuron

- Logistic: $h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$ $\quad y = \begin{cases} 1 & h_\theta(x) \geq 0.5 \\ 0 & h_\theta(x) < 0.5 \end{cases}$

Activation function

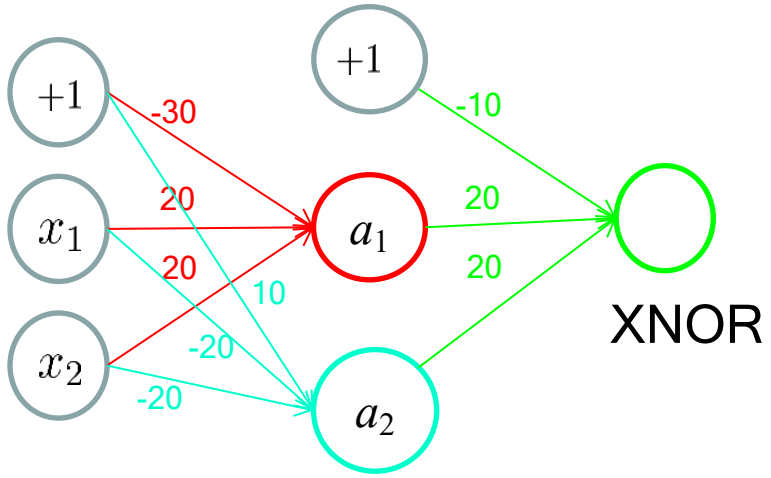


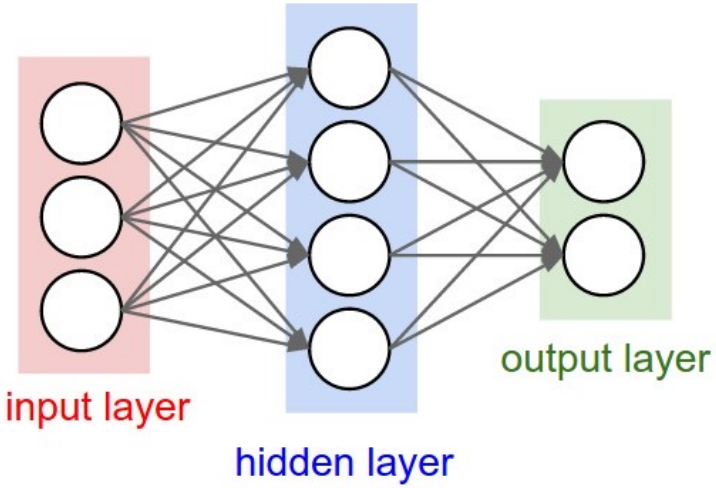


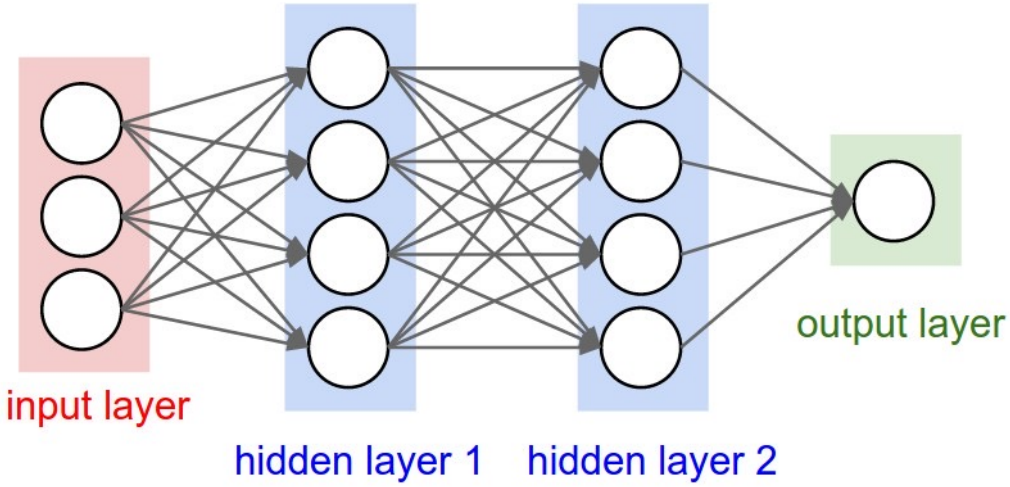
x_1	x_2	$g(\theta_0+\theta_1 x_1+\theta_2 x_2)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Neural Network

x_1	x_2	a_1	a_2	h
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

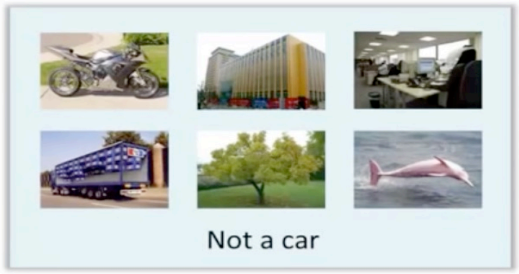






Example: Car Detection

Computer Vision: Car detection



Testing:

What is this?

What is this?

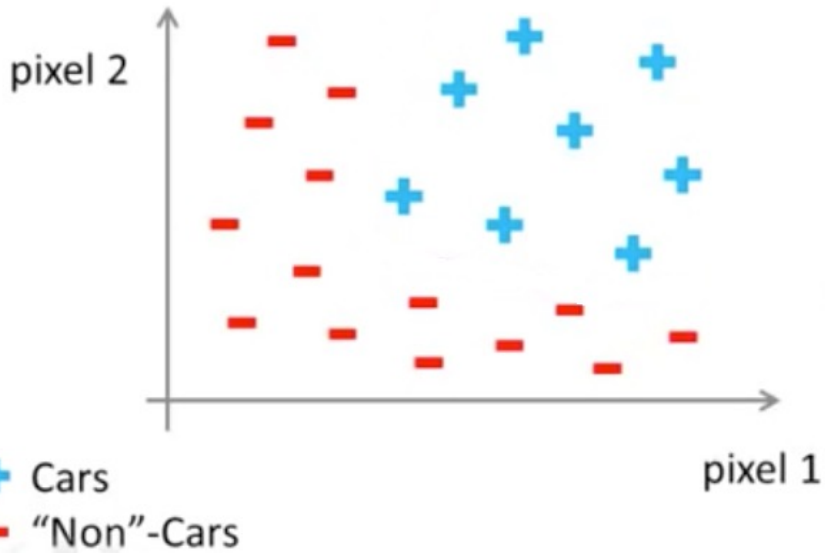
You see this:

But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Pixels as features

Learning Algorithm



50 x 50 pixel images→ 2500 pixels

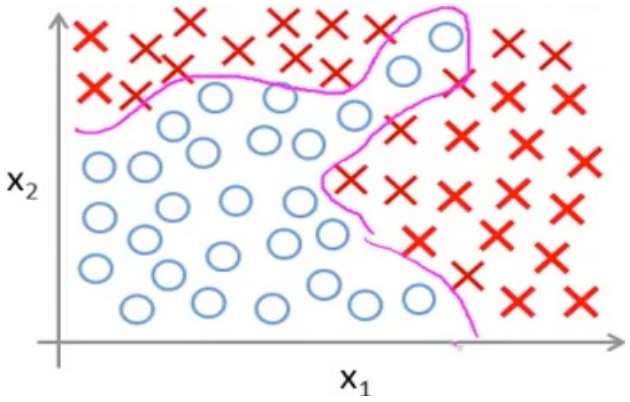
$n = 2500$ (7500 if RGB)

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

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

High-order polynomials vs neural network

- Need non-linear decision boundary
- Need complicated hypothesis with different combinations of features

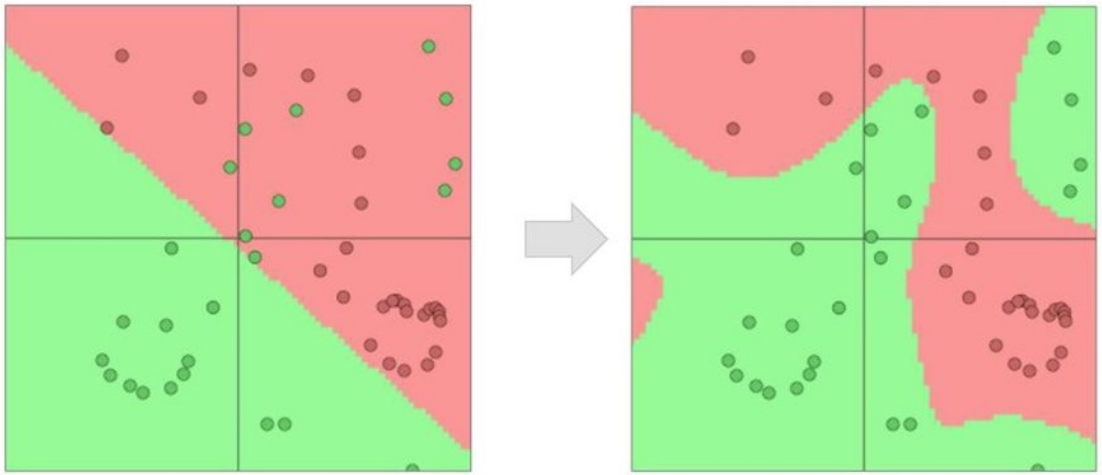


$$g(\theta^T X) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_k x_1 x_2 + \theta_{k+1} x_1 x_3 + \ldots + \theta_n x_1^6 + \theta_{n+1} x_2^6 + \ldots)$$

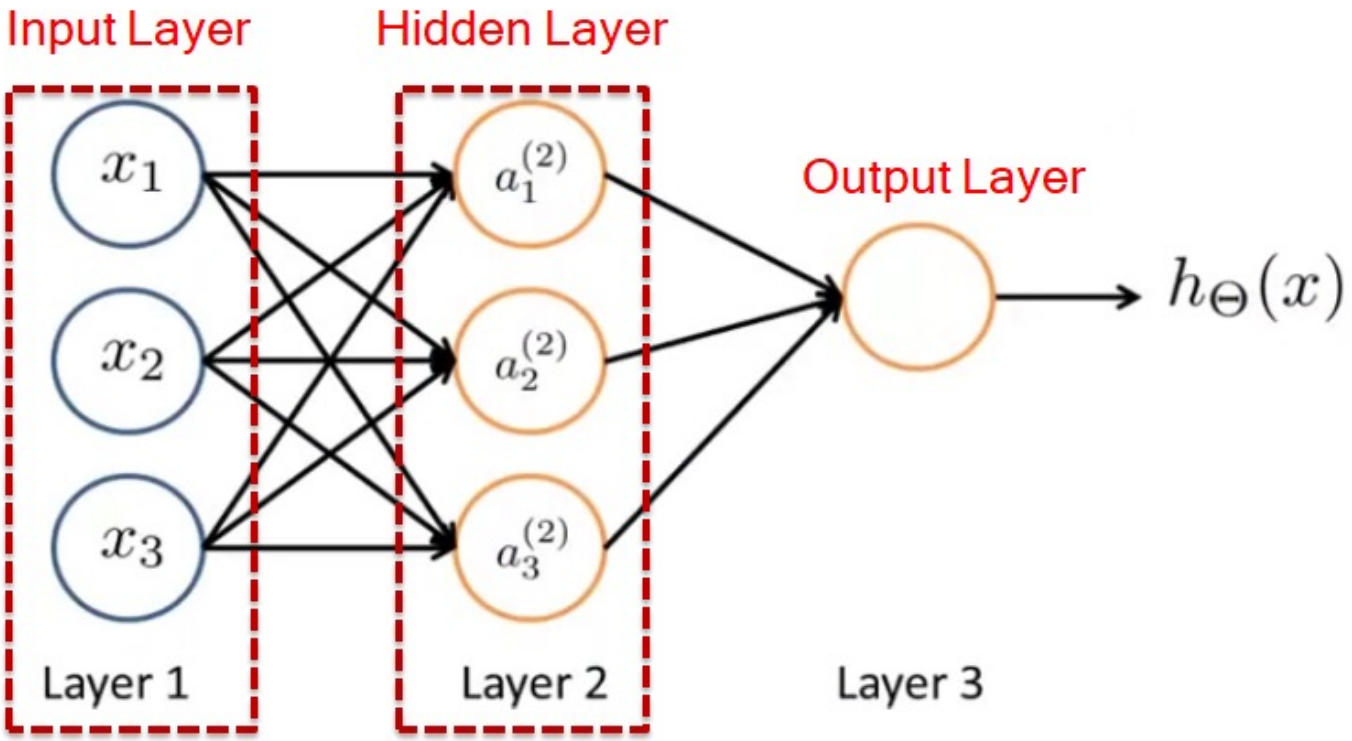
- The computation complexity is extremely huge, hence we have to take another method, called **Neural Network**

Feature Learning

- A network function associated with a neural network characterizes the relationship between input and output layers, which is parameterized by the weights
- With appropriately defined network functions, various learning tasks can be performed by minimizing a cost function over the network function (weights)



Notations in neural networks



$a_i^{(j)}$: activation of unit i at layer j

$\Theta_{ki}^{(j)}$: weight on link from $a_i^{(j)}$ to $a_k^{(j+1)}$; $a_i^{(1)} = x_i$

Model Representation

$a_i^{(j)}$: activation of unit i at layer j

$\Theta_{ki}^{(j)}$: weight on link from $a_i^{(j)}$ to $a_k^{(j+1)}$; $a_i^{(1)} = x_i$

- Connections between layer 1 and layer 2

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

- Connections between layer 2 and layer 3

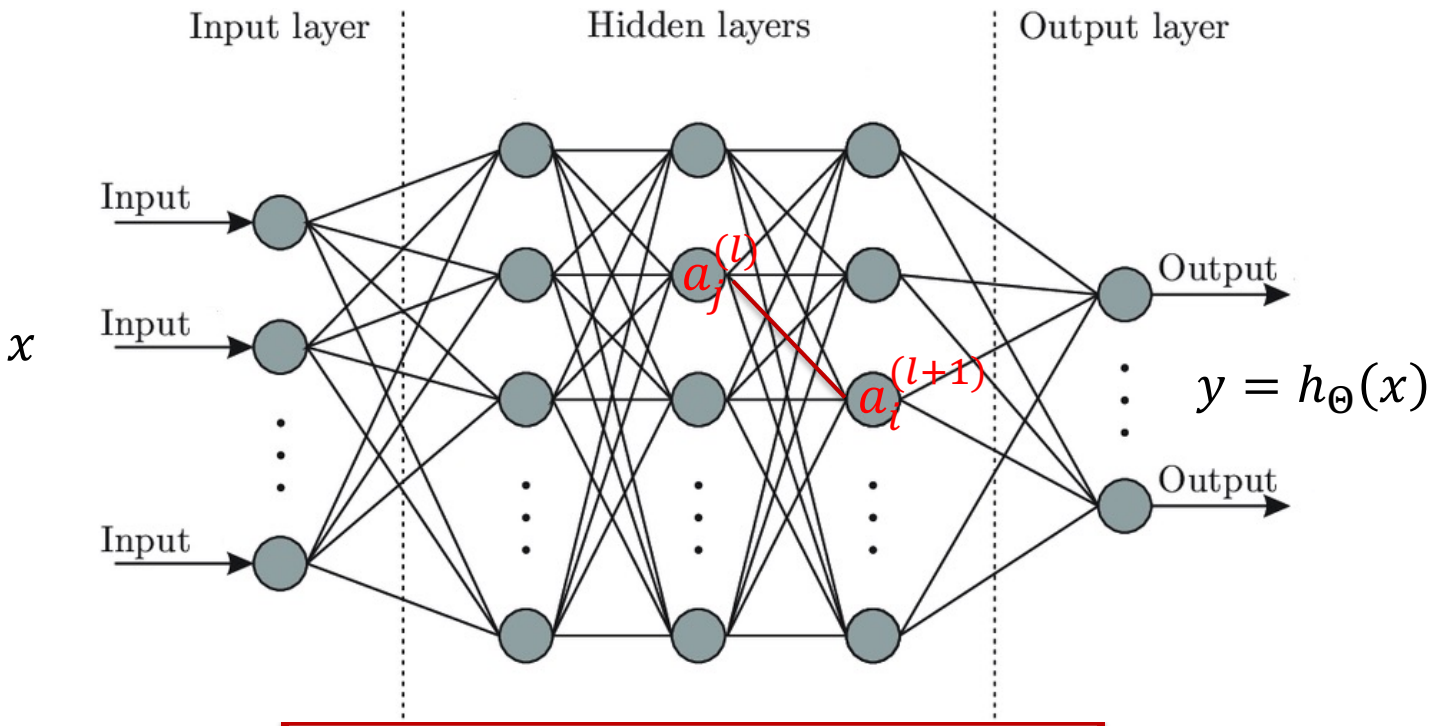
$$h_\theta(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right)$$

Exercise

- Assume that we have a neuron that takes 3 inputs. Its weight vector corresponding to the 3 inputs is [−0.1, 0.4, 0.6]. The weight of the bias is $w_0 = 0.7$. If the input vector is [0.3, −0.4, 0.5], then what is the output of this neuron? (Assume that the activation function is g(z), and you only need to compute the z.)

- $g(0.7 + (-0.1) * 0.3 + 0.4 * (-0.4) + 0.6 * 0.5) = g(0.81)$
- 0.81

Gradient Descent Framework



Repeat until convergence {

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

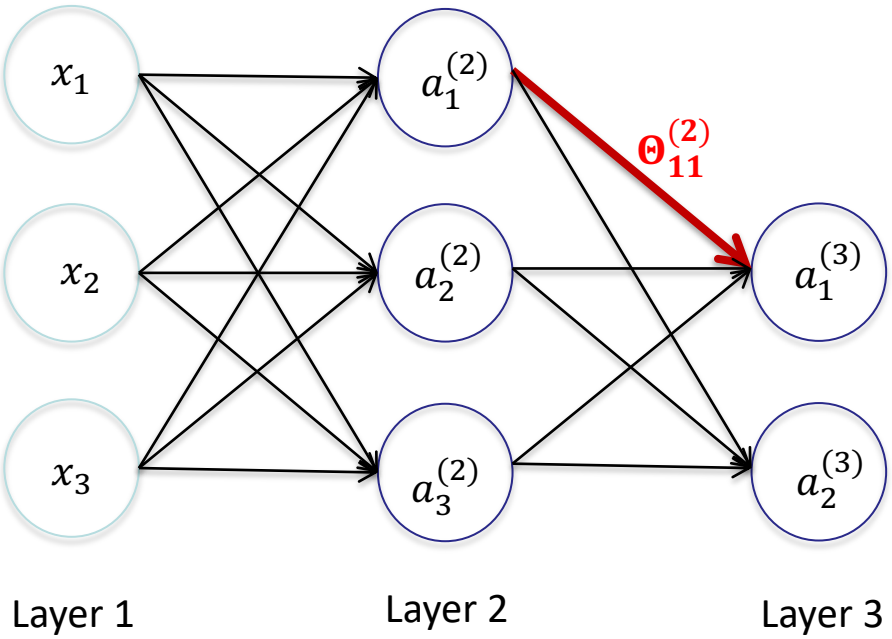
}

Gradient Computation Preparation

- Simplification
  - Single sample $m = 1$, penalized items ignored $\lambda = 0$
  - Linear perceptron
  - $J(\theta) = \frac{1}{2}\sum_{k=1}^{K}(h_\Theta(x)_k - y_k)^2$

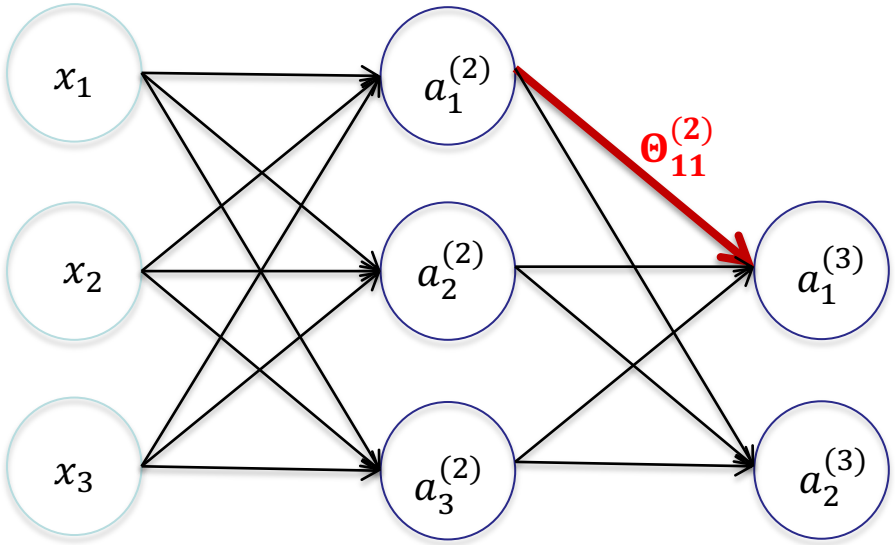
- Chain Rule
  - If $z = f(y)$ and $y = g(x)$, then $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y) \cdot g'(x)$
  - Example: for $z(x) = \frac{1}{2}(x-5)^2$ ($z = \frac{1}{2}y^2; y = x - 5$), we have $z'(x) = f'(y) \cdot g'(x) = y \cdot 1 = x - 5$

Output Layer



Gradient Derivation

$$J(\Theta) = \frac{1}{2}\left(\left(h_\Theta(x)\right)_1 - y_1\right)^2 + \frac{1}{2}\left(\left(h_\Theta(x)\right)_2 - y_2\right)^2$$



$$\frac{\partial J(\Theta)}{\partial \Theta_{11}^{(2)}} = \frac{\partial \frac{1}{2}\left(\left(h_\Theta(x)\right)_1 - y_1\right)^2}{\partial \Theta_{11}^{(2)}} + 0$$

$$= \frac{\partial \frac{1}{2}\left(a_1^{(3)} - y_1\right)^2}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial \Theta_{11}^{(2)}} = \left(a_1^{(3)} - y_1\right) \cdot \frac{\partial a_1^{(3)}}{\partial \Theta_{11}^{(2)}}$$

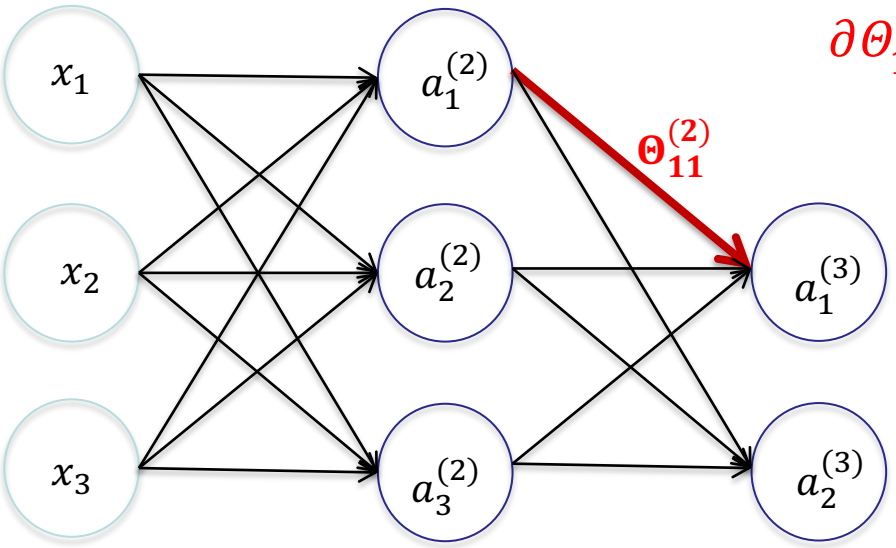
$$= \left(a_1^{(3)} - y_1\right) \cdot \frac{\partial \left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)}{\partial \Theta_{11}^{(2)}}$$

$$= \left(a_1^{(3)} - y_1\right) a_1^{(2)} = \delta_1^{(3)} a_1^{(2)}$$

(δ : delta)

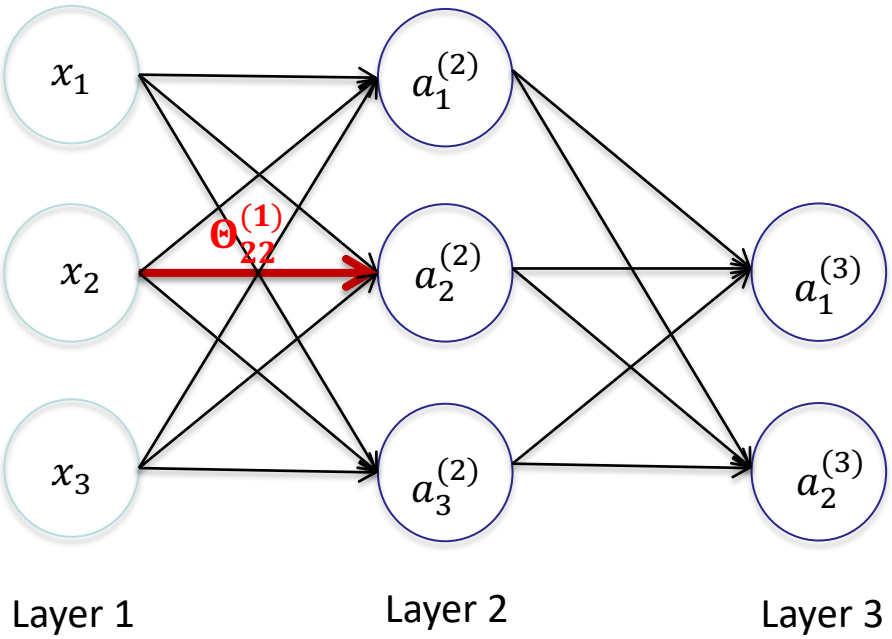
Gradient Computation

$$\frac{\partial J(\Theta)}{\partial \Theta_{11}^{(2)}} = \delta_1^{(3)} a_1^{(2)}$$



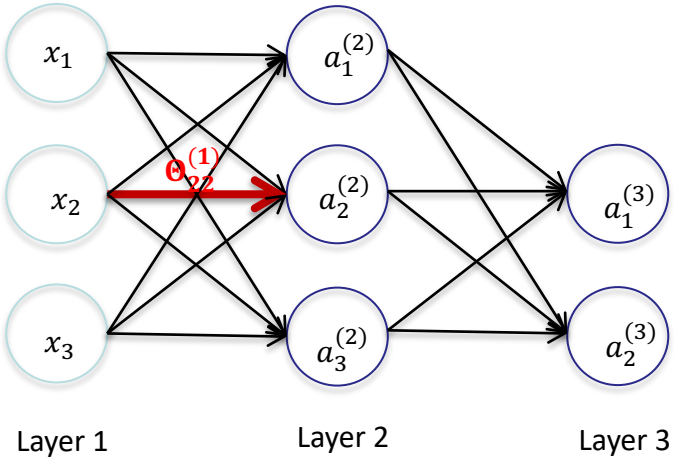
$$\delta_1^{(3)} = a_1^{(3)} - y_1$$

Hidden Layer



Derivation

$$J(\Theta) = \frac{1}{2}\left(\left(h_\Theta(x)\right)_1 - y_1\right)^2 + \frac{1}{2}\left(\left(h_\Theta(x)\right)_2 - y_2\right)^2$$



$$\frac{\partial J(\Theta)}{\partial \Theta_{22}^{(1)}} = \frac{\partial \frac{1}{2}\left(a_1^{(3)} - y_1\right)^2}{\partial \Theta_{22}^{(1)}} + \frac{\partial \frac{1}{2}\left(a_2^{(3)} - y_2\right)^2}{\partial \Theta_{22}^{(1)}}$$

$$= \frac{\partial \frac{1}{2}\left(a_1^{(3)} - y_1\right)^2}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial \Theta_{22}^{(1)}} + \frac{\partial \frac{1}{2}\left(a_2^{(3)} - y_2\right)^2}{\partial a_2^{(3)}} \frac{\partial a_2^{(3)}}{\partial \Theta_{22}^{(1)}}$$

$$= \left(a_1^{(3)} - y_1\right) \frac{\partial a_1^{(3)}}{\partial \Theta_{22}^{(1)}} + \left(a_2^{(3)} - y_2\right) \frac{\partial a_2^{(3)}}{\partial \Theta_{22}^{(1)}}$$

$$= \left(a_1^{(3)} - y_1\right) \frac{\partial \left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)}{\partial \Theta_{22}^{(1)}}$$

$$+ \left(a_2^{(3)} - y_2\right) \frac{\partial \left(\Theta_{20}^{(2)} a_0^{(2)} + \Theta_{21}^{(2)} a_1^{(2)} + \Theta_{22}^{(2)} a_2^{(2)} + \Theta_{23}^{(2)} a_3^{(2)}\right)}{\partial \Theta_{22}^{(1)}}$$

Derivation

$$= \left(a_1^{(3)} - y_1\right) \frac{\partial \left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial \Theta_{22}^{(1)}}$$

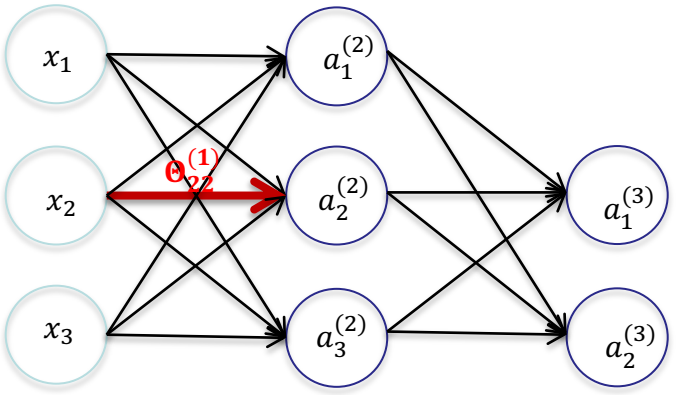
$$+ \left(a_2^{(3)} - y_2\right) \frac{\partial \left(\Theta_{20}^{(2)} a_0^{(2)} + \Theta_{21}^{(2)} a_1^{(2)} + \Theta_{22}^{(2)} a_2^{(2)} + \Theta_{23}^{(2)} a_3^{(2)}\right)}{\partial a_2^2} \cdot \frac{\partial a_2^{(2)}}{\partial \Theta_{22}^{(1)}}$$

$$= \left(\left(a_1^{(3)} - y_1\right)\Theta_{12}^{(2)} + \left(a_2^{(3)} - y_2\right)\Theta_{22}^{(2)}\right) \cdot \frac{\partial a_2^{(2)}}{\partial \Theta_{22}^{(1)}}$$

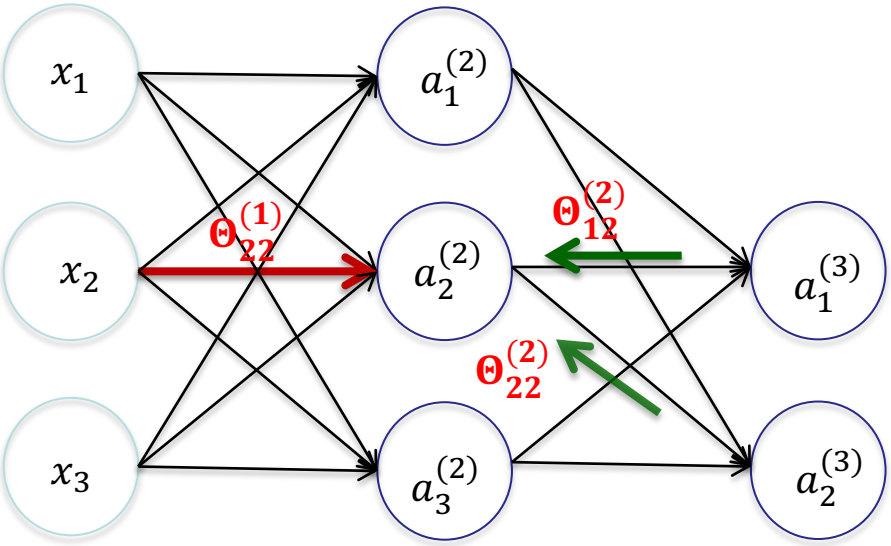
$$= \left(\left(a_1^{(3)} - y_1\right)\Theta_{12}^{(2)} + \left(a_2^{(3)} - y_2\right)\Theta_{22}^{(2)}\right) \cdot \frac{\partial \left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right)}{\partial \Theta_{22}^{(1)}}$$

$$= \left(\left(a_1^{(3)} - y_1\right)\Theta_{12}^{(2)} + \left(a_2^{(3)} - y_2\right)\Theta_{22}^{(2)}\right) x_2$$

$$= \left(\delta_1^{(3)}\Theta_{12}^{(2)} + \delta_2^{(3)}\Theta_{22}^{(2)}\right) x_2$$



Gradient Computation



$$\frac{\partial J(\Theta)}{\partial \Theta_{11}^{(2)}} = \delta_1^{(3)} a_1^{(2)}$$

$$\delta_1^{(3)} = a_1^{(3)} - y_1$$

$$\delta_2^{(3)} = a_2^{(3)} - y_2$$

$$\frac{\partial J(\Theta)}{\partial \Theta_{22}^{(1)}} = \left(\left(a_1^{(3)} - y_1 \right) \Theta_{12}^{(2)} + \left(a_2^{(3)} - y_2 \right) \Theta_{22}^{(2)} \right) x_2$$

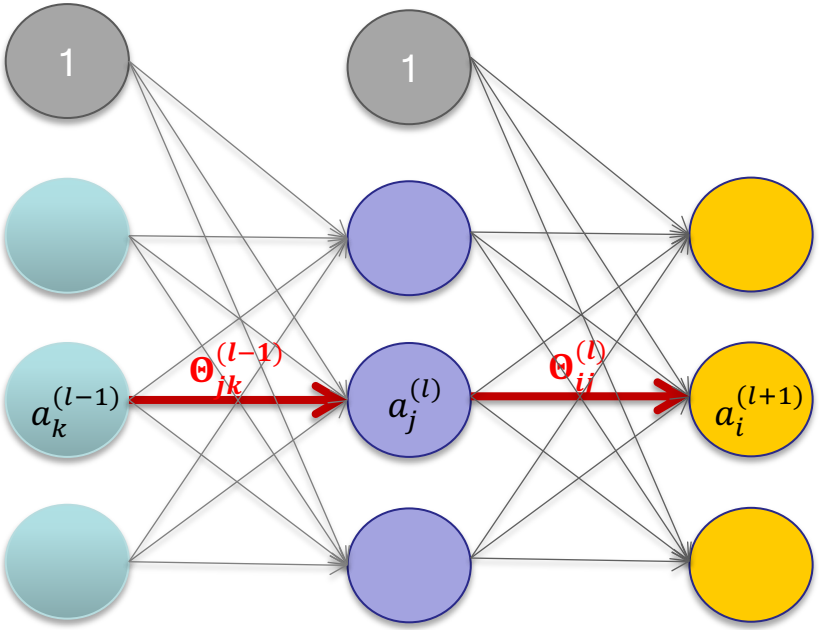
$$= \left(\delta_1^{(3)} \Theta_{12}^{(2)} + \delta_2^{(3)} \Theta_{22}^{(2)} \right) x_2 \quad = \delta_2^{(2)} x_2$$

Derivation – Output Layer

$$\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = \frac{\partial \frac{1}{2}\left(a_i^{(L)} - y_i\right)^2}{\partial \Theta_{ij}^{(l)}} = \frac{\partial \frac{1}{2}\left(a_i^{(L)} - y_i\right)^2}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial \Theta_{ij}^{(l)}} = \left(a_i^{(L)} - y_i\right) \cdot \frac{\partial a_i^{(L)}}{\partial \Theta_{ij}^{(l)}}$$

$$= \left(a_i^{(L)} - y_i\right) \cdot \frac{\partial \left(\sum_{j=0}^{S_l} \Theta_{ij}^{(l)} a_j^{(l)}\right)}{\partial \Theta_{ij}^{(l)}} = \left(a_i^{(L)} - y_i\right) a_j^{(l)} = \boxed{\delta_i^{(L)}} a_j^{(l)}$$

Error generated in the output layer



Derivation – Hidden Layer 1

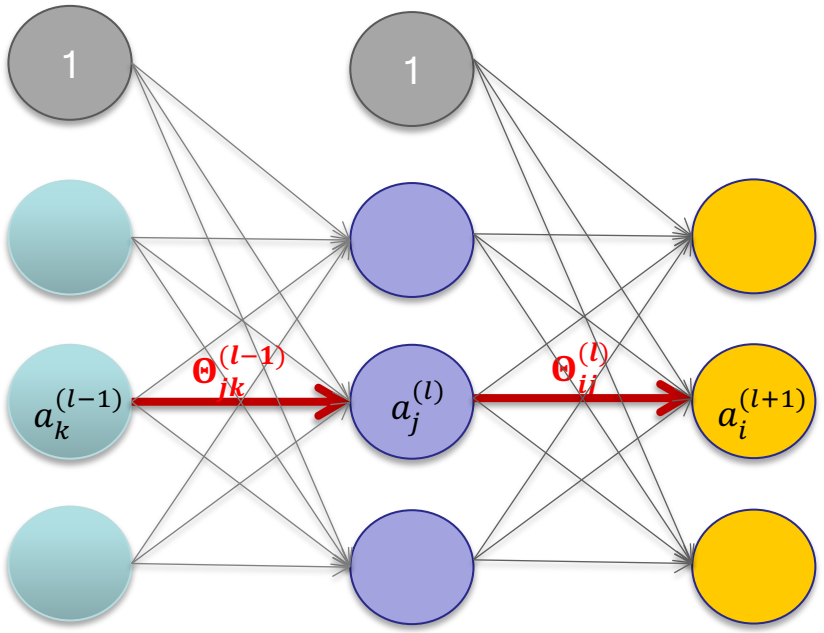
$$\frac{\partial J(\Theta)}{\partial \Theta_{jk}^{(l-1)}} = \sum_{i=1}^{S_L} \frac{\partial J_i(\Theta)}{\partial \Theta_{jk}^{(l-1)}} = \sum_{i=1}^{S_L} \left(\frac{\partial \frac{1}{2}\left(a_i^{(L)} - y_i\right)^2}{\partial a_i^{(L)}} \cdot \frac{\partial a_i^{(L)}}{\partial \Theta_{jk}^{(l-1)}} \right)$$

$$= \sum_{i=1}^{S_L} \left(\left(a_i^{(L)} - y_i\right) \cdot \frac{\partial \left(\sum_{p=0}^{S_l} \Theta_{ip}^{(l)} a_p^{(l)} \right)}{\partial \Theta_{jk}^{(l-1)}} \right) = \sum_{i=1}^{S_L} \left(\delta_i^{(L)} \cdot \frac{\partial \left(\sum_{p=0}^{S_l} \Theta_{ip}^{(l)} a_p^{(l)} \right)}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial \Theta_{jk}^{(l-1)}} \right)$$

$$= \sum_{i=1}^{S_{l+1}} \left(\delta_i^{(l+1)} \cdot \Theta_{ij}^{(l)} \cdot \frac{\partial \left(\sum_{k=0}^{S_{l-1}} \Theta_{jk}^{(l-1)} a_k^{(l-1)} \right)}{\partial \Theta_{jk}^{(l-1)}} \right)$$

$$= \left(\sum_{i=1}^{S_{l+1}} \left(\delta_i^{(l+1)} \Theta_{ij}^{(l)} \right) \right) a_k^{(l-1)} = \delta_j^{(l)} a_k^{(l-1)}$$

Error generated in the hidden layer



$$\frac{\partial J(\Theta)}{\partial \Theta_{22}^{(1)}} = \left(\delta_1^{(3)} \Theta_{12}^{(2)} + \delta_2^{(3)} \Theta_{22}^{(2)} \right) x_2 = \delta_2^{(2)} x_2$$

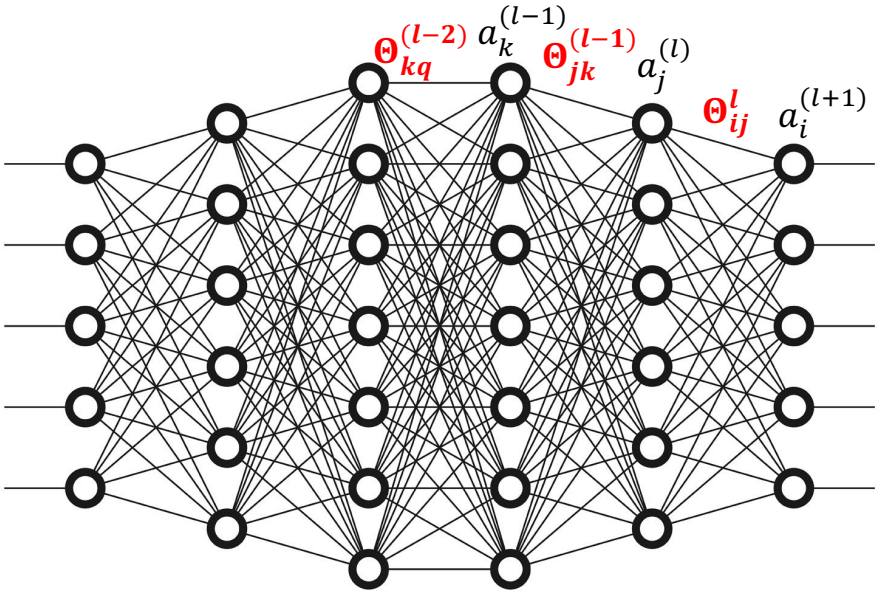
Directly Applying Gradient Descent is Expensive

$$\frac{\partial J(\Theta)}{\partial \Theta_{kq}^{(l-2)}} = \sum_{i=1}^{S_L} \frac{\partial J_i(\Theta)}{\partial \Theta_{kq}^{(l-2)}} = \sum_{i=1}^{S_L} \left(\frac{\partial \frac{1}{2}\left(a_i^{(L)} - y_i\right)^2}{\partial a_i^{(L)}} \cdot \frac{\partial a_i^{(L)}}{\partial \Theta_{kq}^{(l-2)}} \right)$$

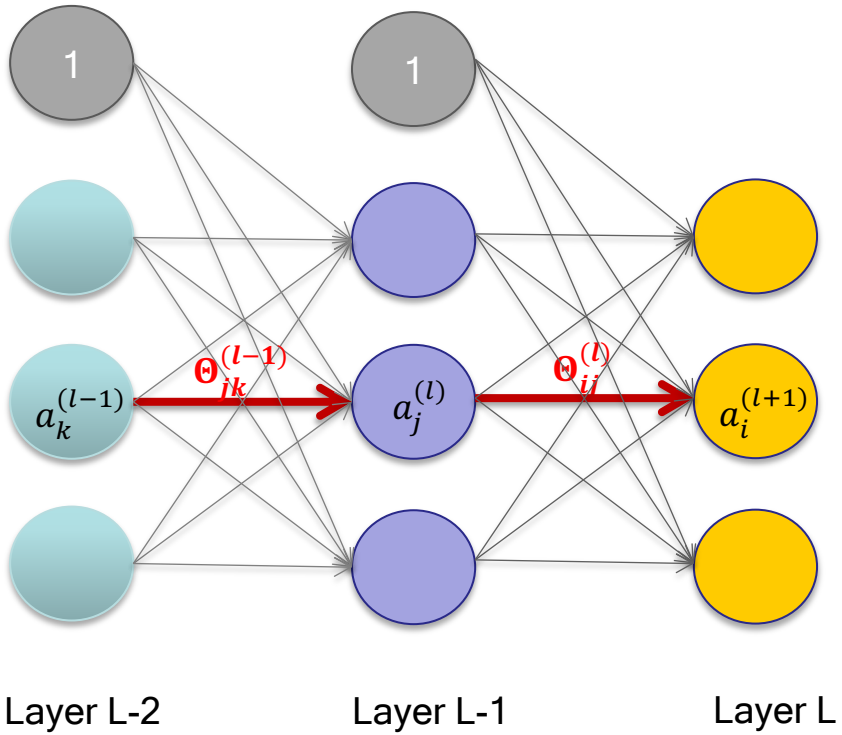
$$= \sum_{i=1}^{S_L} \left(\left(a_i^{(L)} - y_i\right) \cdot \frac{\partial \left(\sum_{p=0}^{S_l} \Theta_{ip}^{(l)} a_p^{(l)} \right)}{\partial \Theta_{kq}^{(l-2)}} \right)$$

$$= \sum_{i=1}^{S_L} \left(\delta_i^{(L)} \cdot \sum_{p=0}^{S_l} \left[\frac{\partial \Theta_{ip}^{(l)} a_p^{(l)}}{\partial a_p^{(l)}} \cdot \frac{\partial a_p^{(l)}}{\partial \Theta_{kq}^{(l-2)}} \right] \right)$$

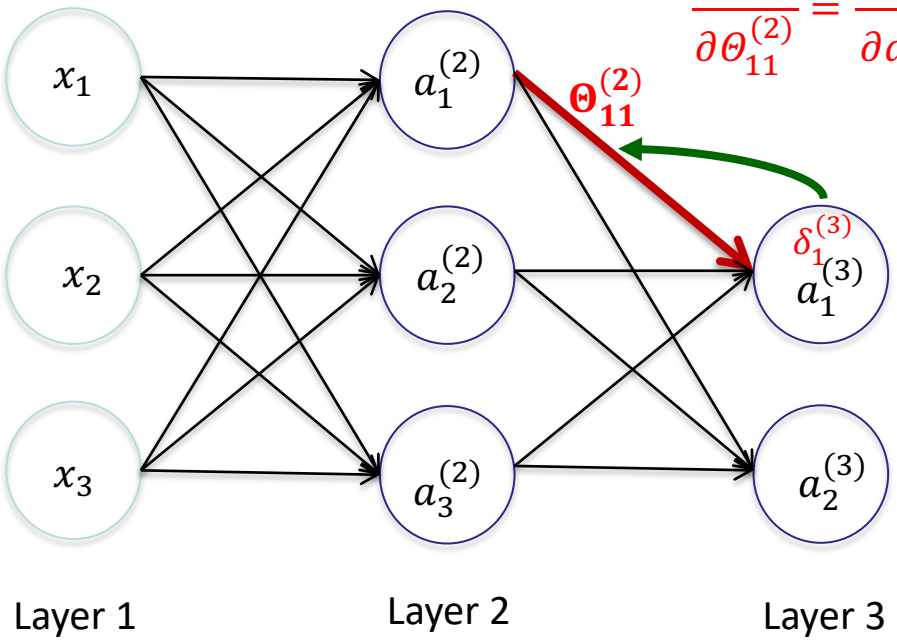
$$= \sum_{i=1}^{S_L} \left(\delta_i^{(L)} \cdot \sum_{p=0}^{S_l} \left[\Theta_{ip}^{(l)} \cdot \frac{\partial \left(\sum_{c=0}^{S_{l-1}} \Theta_{pc}^{(l-1)} a_c^{(l-1)} \right)}{\partial \Theta_{kq}^{(l-2)}} \right] \right)$$



General Network



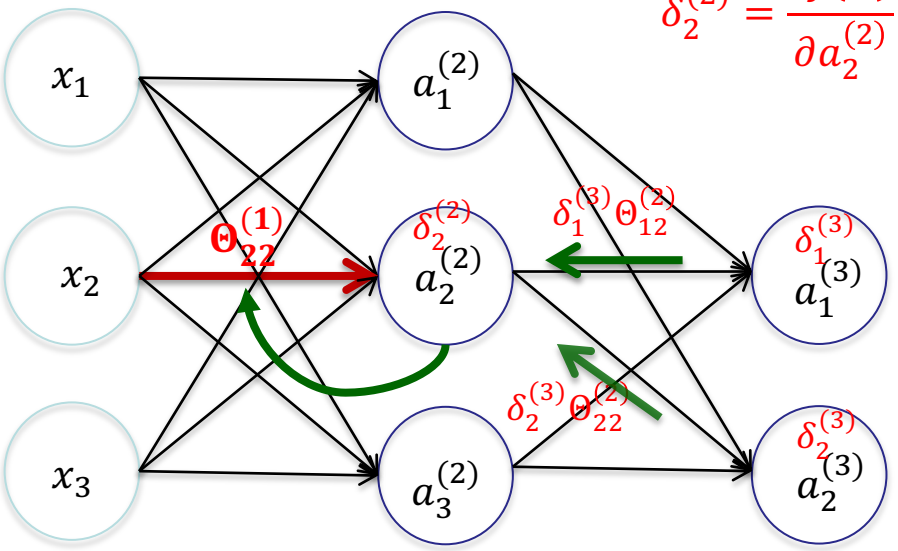
Gradient Propagation



$$\frac{\partial J(\Theta)}{\partial \Theta_{11}^{(2)}} = \frac{\partial J(\Theta)}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial \Theta_{11}^{(2)}} = \delta_1^{(3)} a_1^{(2)}$$

$$\delta_1^{(3)} = a_1^{(3)} - y_1 = \frac{\partial J(\Theta)}{\partial a_1^{(3)}}$$

Gradient Propagation



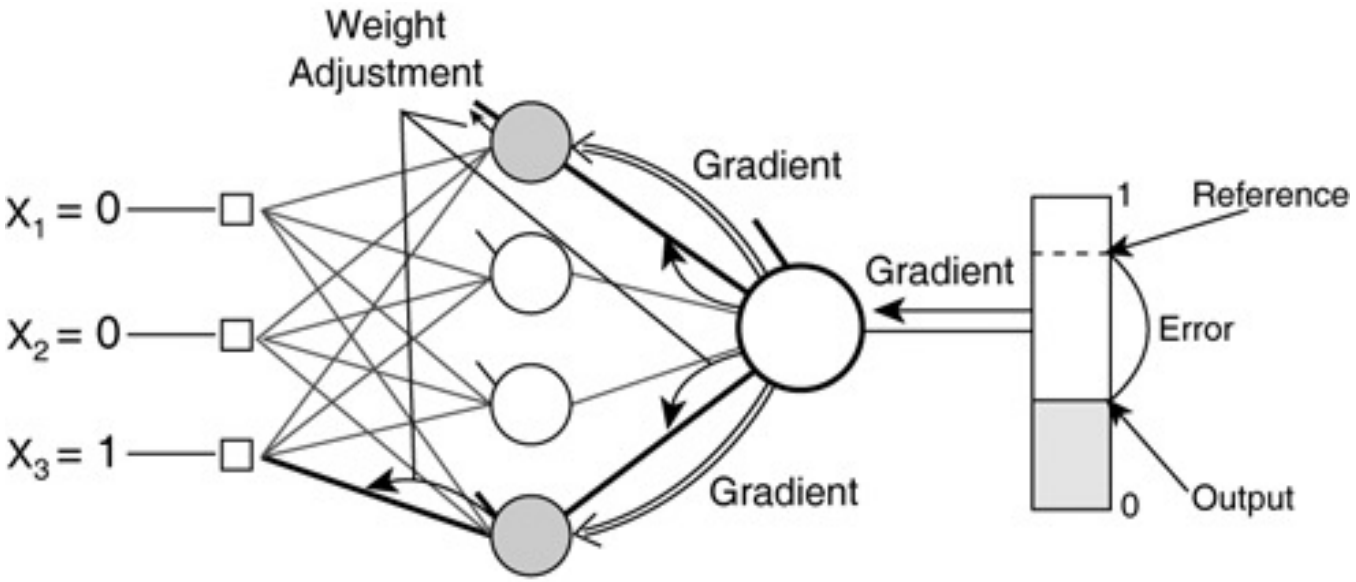
$$\delta_2^{(2)} = \frac{\partial J(\Theta)}{\partial a_2^{(2)}} = \delta_1^{(3)}\Theta_{12}^{(2)} + \delta_2^{(3)}\Theta_{22}^{(2)}$$

$$\delta_1^{(3)} = a_1^{(3)} - y_1 = \frac{\partial J(\Theta)}{\partial a_1^{(3)}}$$

$$\delta_2^{(3)} = a_2^{(3)} - y_2 = \frac{\partial J(\Theta)}{\partial a_2^{(3)}}$$

$$\frac{\partial J(\Theta)}{\partial \Theta_{22}^{(1)}} = \delta_2^{(2)} x_2 = \left(\delta_1^{(3)}\Theta_{12}^{(2)} + \delta_2^{(3)}\Theta_{22}^{(2)}\right) x_2$$

Back Propagation

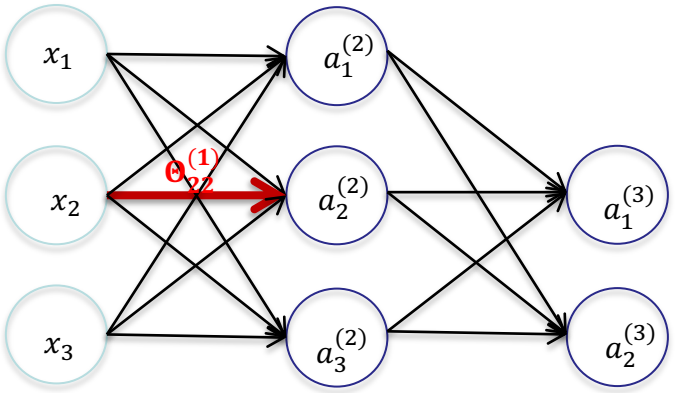


$$\delta_i^{(L)} = a_i^{(L)} - y_i$$

$$\delta_j^{(l)} = \sum_{i=1}^{S_{l+1}} \left(\delta_i^{(l+1)} \Theta_{ij}^{(l)} \right)$$

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

$$\frac{\partial J(\Theta)}{\partial \Theta_{jk}^{(l-1)}} = \delta_j^{(l)} a_k^{(l-1)}$$



Back Propagation Algorithm

$$\delta_i^{(L)} = a_i^{(L)} - y_i$$

$$\delta_j^{(l)} = \sum_{i=1}^{S_{l+1}} \left(\delta_i^{(l+1)} \Theta_{ij}^{(l)} \right)$$

Training Set $\{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$, for all (i, j, l)

For $s = 1$ to m

Set $a^{(1)} = x^{(s)}$

Performance Forward Propagation to compute $a^{(2)} \cdots, a^{(L)}$

Set $\delta^{(L)} = a^{(L)} - y^{(s)}$

Performance Backward Propagation to compute $\delta^{(L-1)} \cdots, \delta^{(2)}$

Set $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + \delta_i^{(l+1)} a_j^{(l)}$, for all (i, j, l)

Gradient Descent Algorithm

$$J(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}\sum_{k=1}^{S_L}\left(h_\Theta\left(x^{(i)}\right)_k - y_i\right)^2 + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{j=1}^{S_l}\sum_{i=1}^{S_{l+1}}\left(\Theta_{ij}^{(l)}\right)^2$$

Repeat until convergence {

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

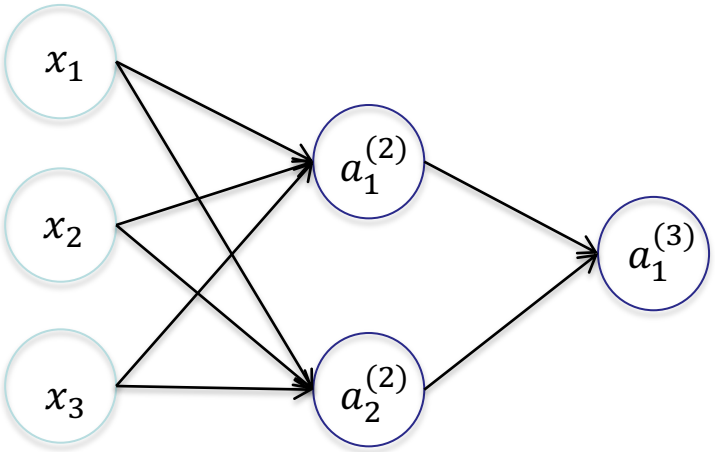
}

$$\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = \begin{cases} \frac{1}{m}\Delta_{ij}^{(l)} + \frac{\lambda}{m}\Theta_{ij}^{(l)} & l \neq L \\ \frac{1}{m}\Delta_{ij}^{(l)} & l = L \end{cases}$$

Obtained by the Backward Propagation Algorithm

Implementation Detail

- Important to randomize initial weights Θ in the network
- Can't have uniform initial weights, otherwise all updates will be identical, and the network won't learn anything.



Zero Initialization Example:

If all weights are initialized to 0, $a_1^{(2)} = a_2^{(2)}$,

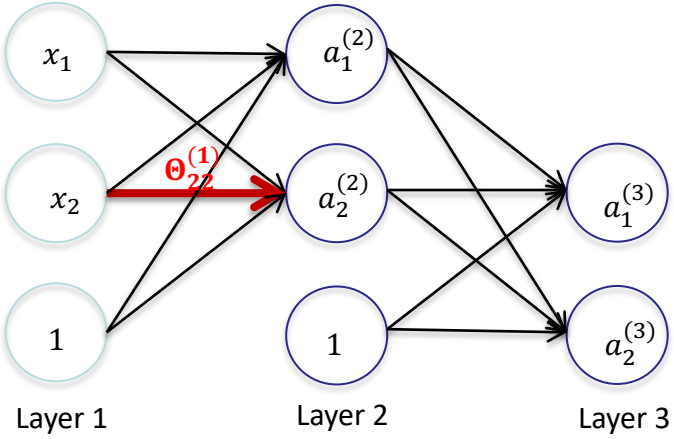
also $\delta_1^{(2)} = \delta_2^{(2)}$.

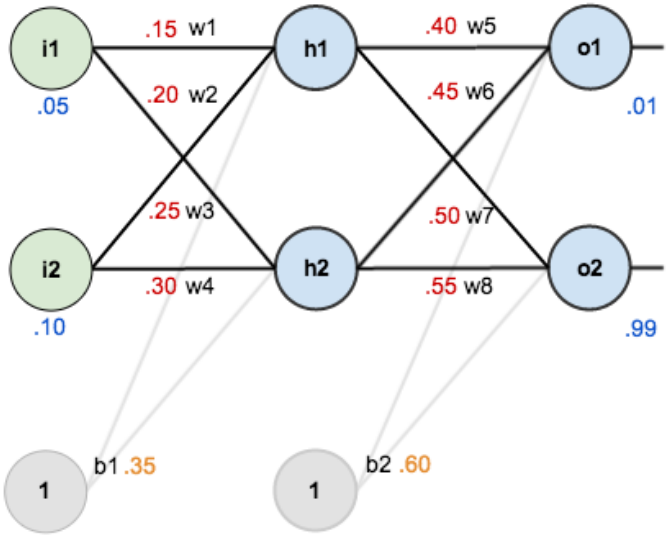
$\frac{\partial J(\Theta)}{\partial a_1^{(2)}} = \frac{\partial J(\Theta)}{\partial a_2^{(2)}}$, thus $\Theta_{01}^{(1)} = \Theta_{02}^{(1)} \ldots$, $a_1^{(2)} = a_2^{(2)}$.

After each update, two hidden units in Layer 2 are identical.

Example

The neural network given below adopts sigmoid function as its activation function $\sigma(x)$, e.g., $a_1^{(3)} = \sigma\left(\Theta_{10}^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)}\right)$. The cost function is defined as $J(\Theta) = \frac{1}{2}\left(a_1^{(3)} - y_1\right)^2 + \frac{1}{2}\left(a_2^{(3)} - y_2\right)^2$.





We consider a single data sample $x = \begin{pmatrix} 0.05 \\ 0.1 \end{pmatrix}$ and the corresponding label $y = \begin{pmatrix} 0.01 \\ 0.99 \end{pmatrix}$. Use the training data to develop the neural network model and solve it by using gradient descent algorithm with initialized setting:

$\Theta^{(1)}[0] = \begin{pmatrix} 0.15 & 0.25 & 0.35 \\ 0.20 & 0.30 & 0.35 \end{pmatrix}$, $\Theta^{(2)}[0] = \begin{pmatrix} 0.40 & 0.50 & 0.6 \\ 0.45 & 0.55 & 0.6 \end{pmatrix}$, and $\alpha = 0.5$.

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

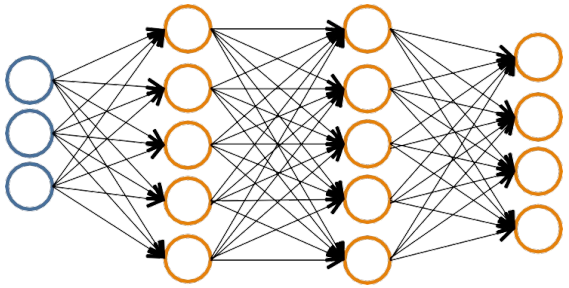
Application: Multi-class Classification

Pedestrian

Car

Motorcycle

Truck



$h_\theta(\boldsymbol{x}) \in R^K$

K: The number of classes

We want:

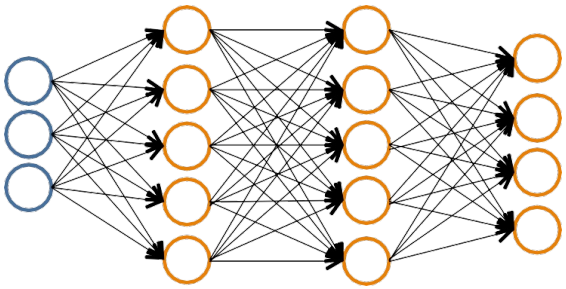
$$h_\theta(\boldsymbol{x}) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$ when pedestrian

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

$$h_\theta(\boldsymbol{x}) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$ when motorcycle

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

Multi-class Classification



$h_\theta(\boldsymbol{x}) \in R^K$

K: The number of classes

We want:

$$h_\theta(\boldsymbol{x}) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad h_\theta(\boldsymbol{x}) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad h_\theta(\boldsymbol{x}) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad h_\theta(\boldsymbol{x}) \approx \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

when pedestrian | when car | when motorcycle | when truck

- Given $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$
- Must convert labels to 1-of-*K* representation

e.g., $y_i = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ when pedestrian, $y_i = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$ when motorcycle.

Cost Function

- Recall that cost function of Logistic Regression:

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

- Cost function of K-class Neural Network

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

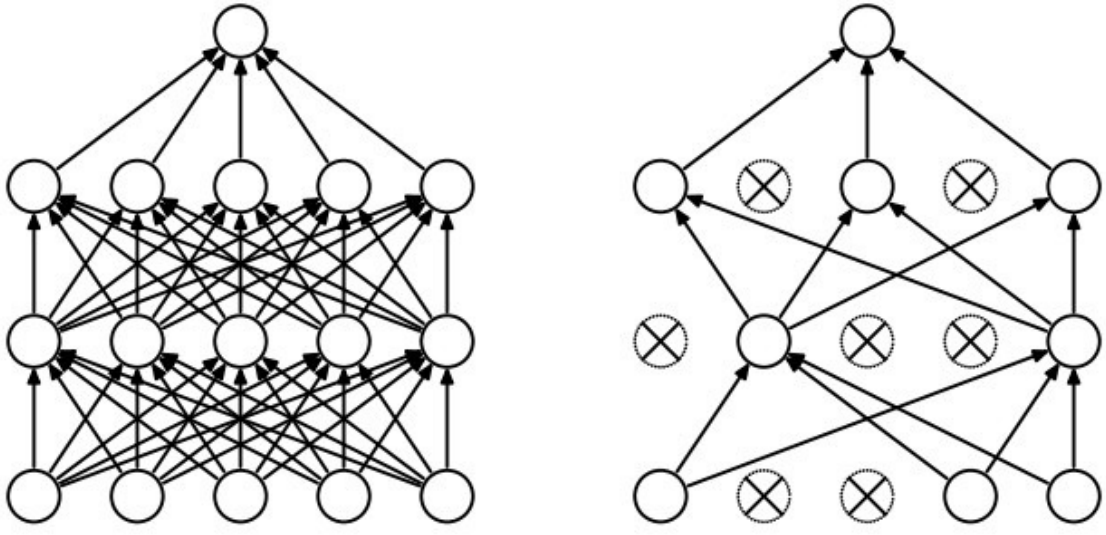
- The i-th output: $\left(h_\Theta(x)\right)_i$

k^{th} class: true, predicted
Not k^{th} class: true, predicted

Stochastic Gradient Descent

- Gradient descent, follows the gradient of an entire training set downhill
- Stochastic gradient descent, follows the gradient of randomly selected minbatches downhill
  - minibatches: The gradients are calculated and the variables are updated iteratively with subsets of all observations
  - randomly divides the set of observations into minibatches
  - For each minibatch, the gradient is computed and the vector is moved
  - Once all minibatches are used, you say that the iteration, or epoch, is finished and start the next one

The dropout regularization



(a) Standard Neural Net (b) After applying dropout.

- Randomly shutdown a subset of units in training
- It is a sparse representation
- It is a different net each time, but all nets share the parameters
- A net with n units can be seen as a collection of 2^n possible thinned nets, all of which share weights
- At test time, it is a single net with averaging*(1-p), where p is the dropout rate
- Avoid overfitting

model.eval() and torch.no_grad()

- model.eval() will notify all your layers that you are in eval mode, that way, batchnorm or dropout layers will work in eval mode instead of training mode.
- torch.no_grad() impacts the autograd engine and deactivate it. It will reduce memory usage and speed up computations but you won't be able to backprop (which you don't want in an eval script).
- model.train() tells your model that you are training the model. This helps inform layers such as Dropout and BatchNorm, which are designed to behave differently during training and evaluation.