CENG 463
Machine Learning

Lecture 15
Neural Networks

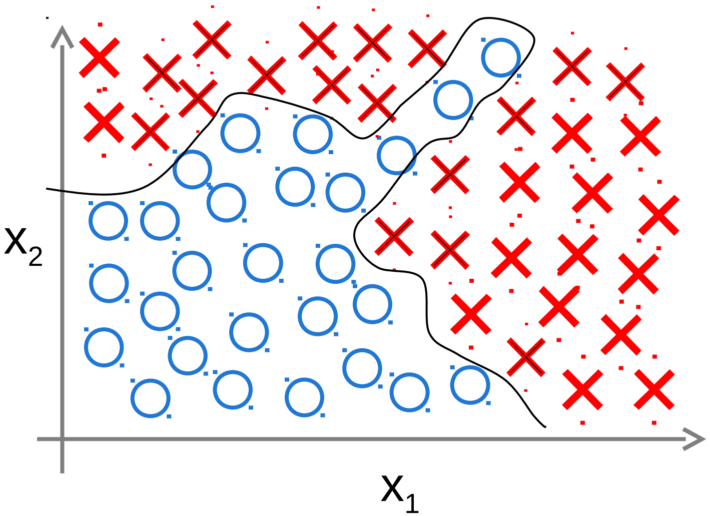
Slides were prepared using the material provided in Stanford's Machine Learning Course given by Andrew Ng

Large Feature Space Examples

When feature space (n) is large, logistic regression is not a good classification algorithm.

Think of a complex classifier for a two-variable case:

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 \\ + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 \\ + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$



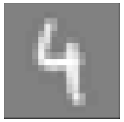
In case of 100 variables, only 2nd order terms (x_1^2, x_1x_2, x_1x_3, .. x_3^2, .. $x_{97}x_{98}$, .. etc.)

constitutes ~5000 features.

Large Feature Space Examples

An example from computer vision:

Hand-written digit recognition

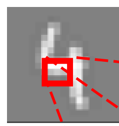
You see this 

Large Feature Space Examples

An example from computer vision:

Hand-written digit recognition

You see this



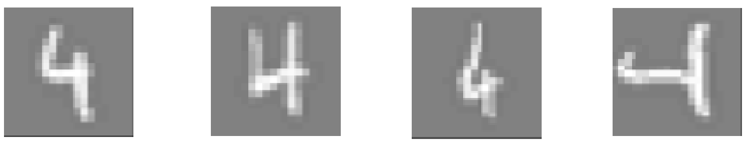
But camera see this

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

Large Feature Space Examples

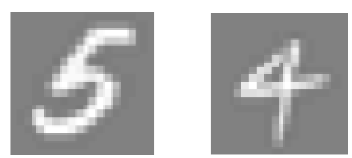
Training set:

4’s



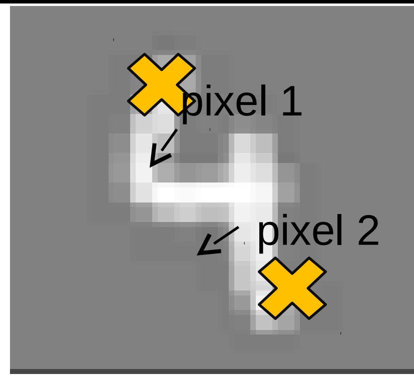
Non-4’s

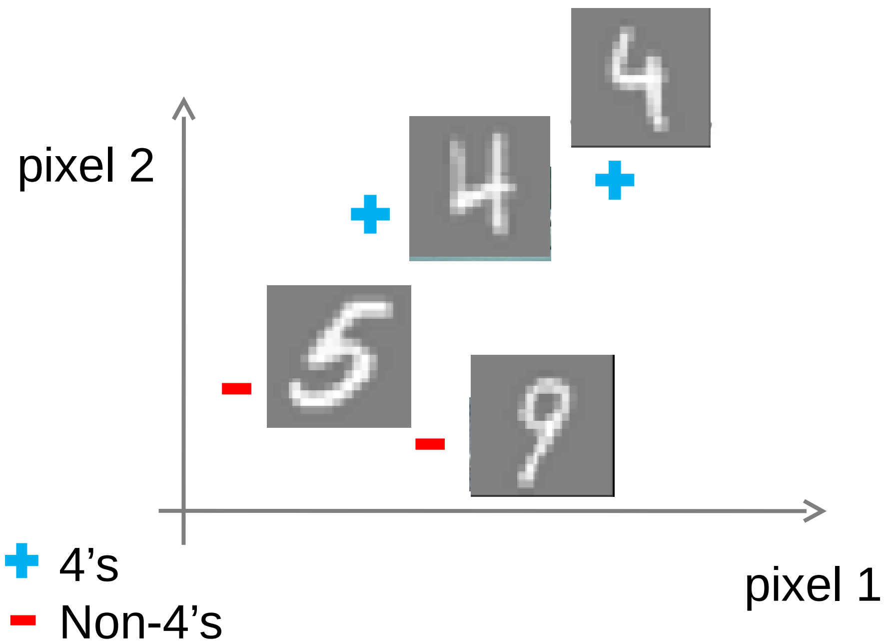
Testing: Are these 4?



Large Feature Space Examples

Training set with two features(pixels):





An image size of 50 x 50 pixels makes 2500 pixels (7500 if RGB)

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

Only quadratic (2nd order) terms (x_1^2, x_1x_2, x_1x_3, .. x_3^2, .. $x_{97}x_{98}$, .. etc.) make ~3000000 features.

Neural Networks

Origin of Neural Networks:
Algorithms that try to mimic the brain.

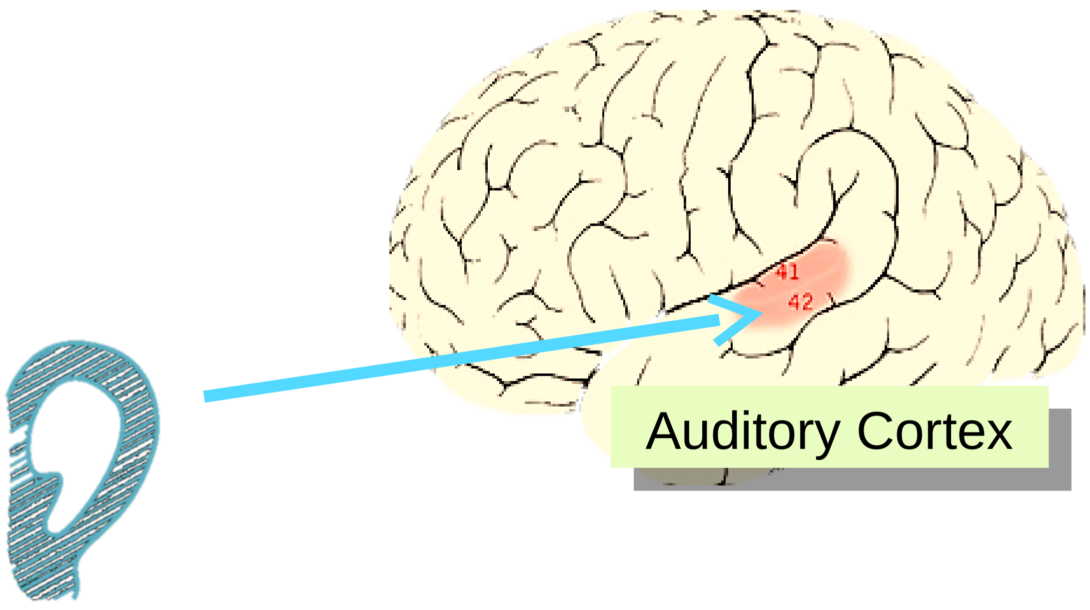
Now, it is a state-of-the-art technique for many machine learning/pattern recognition problems.

Analogy with the Human Brain

The “one learning algorithm” hypothesis:

Brain does not use different algorithms for different tasks, but it trains its tissues to accomplish the tasks.

Eg.

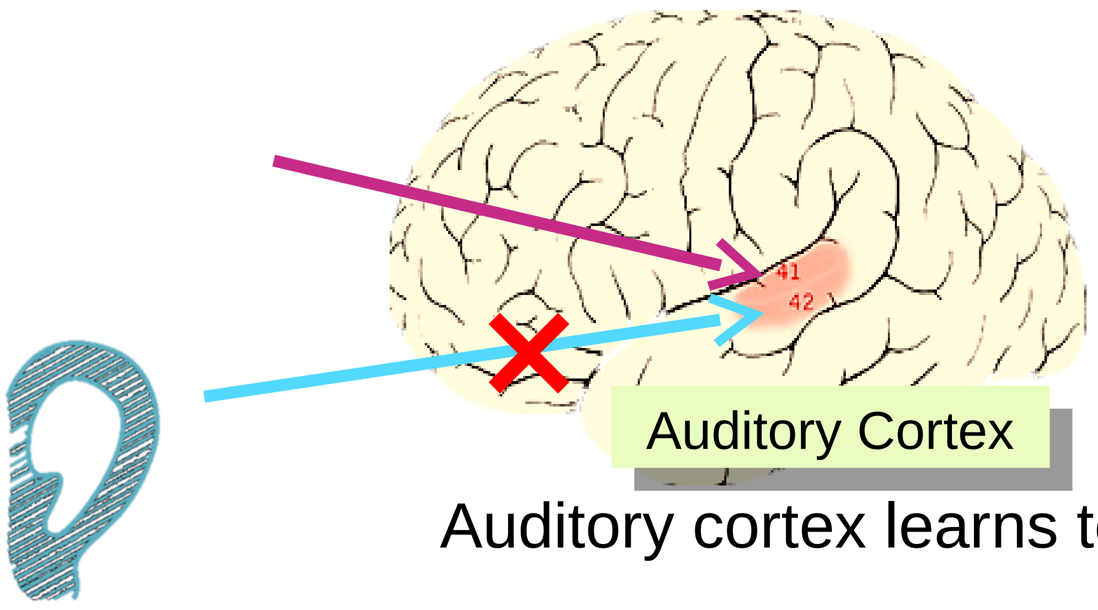


Analogy with the Human Brain

The "one learning algorithm" hypothesis:

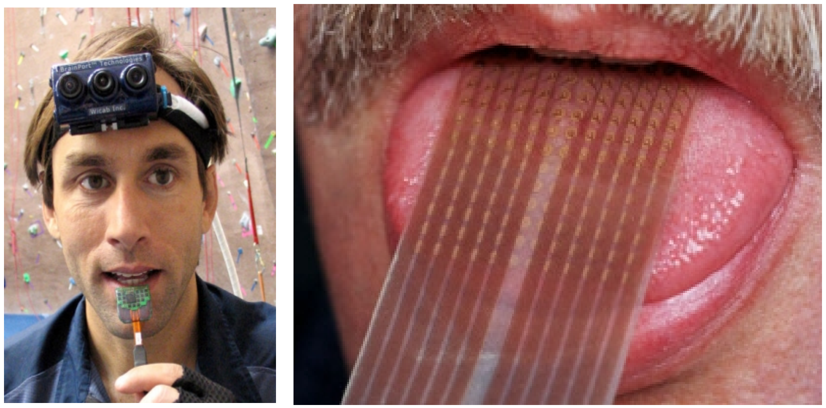
Brain does not use different algorithms for different tasks, but it trains its tissues to accomplish the tasks.

Eg.



Auditory cortex learns to see

Sensor Representations in Brain



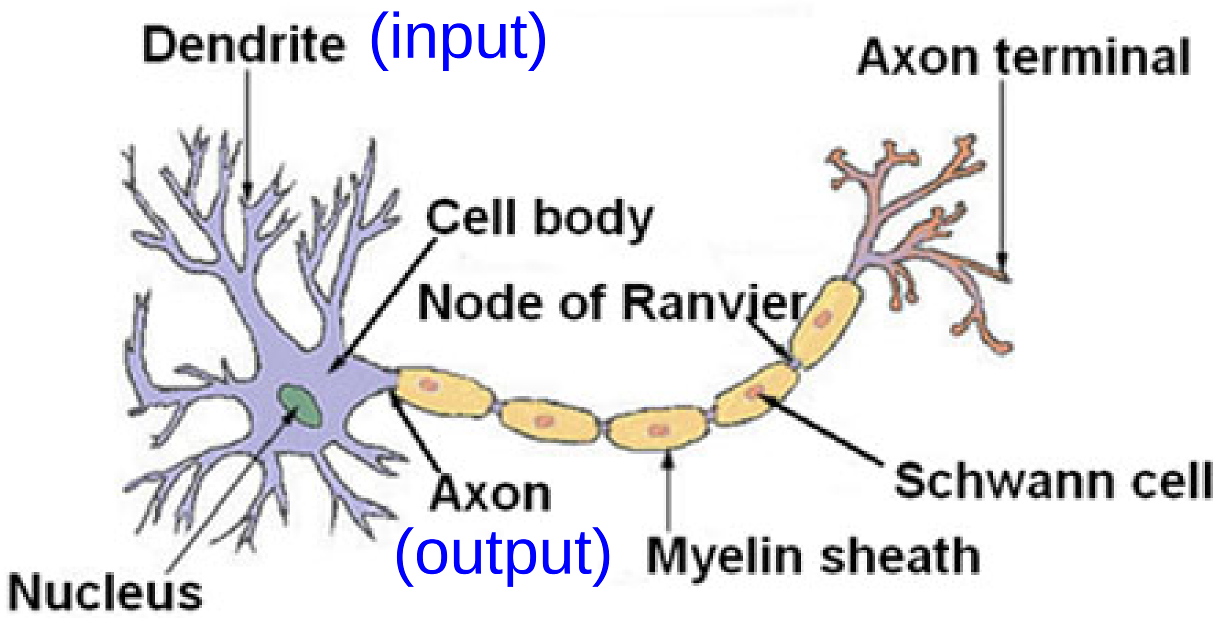
Seeing with your tongue

Human echolocation (sonar)

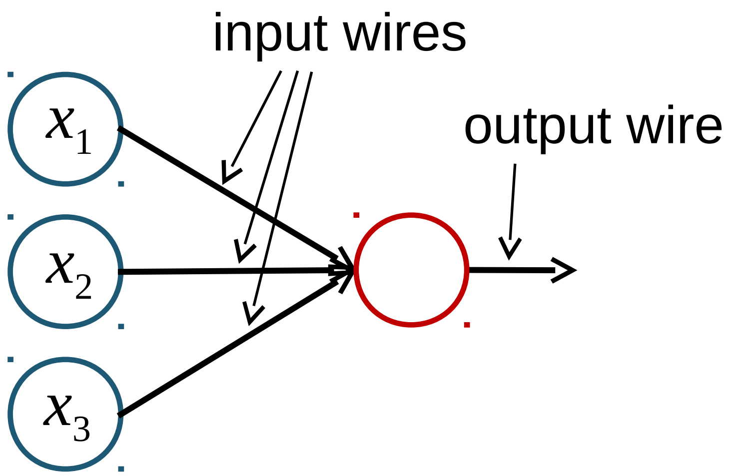
Brain learns how to interpret data coming from a sensor

Implanting a 3rd eye

Brain Neurons

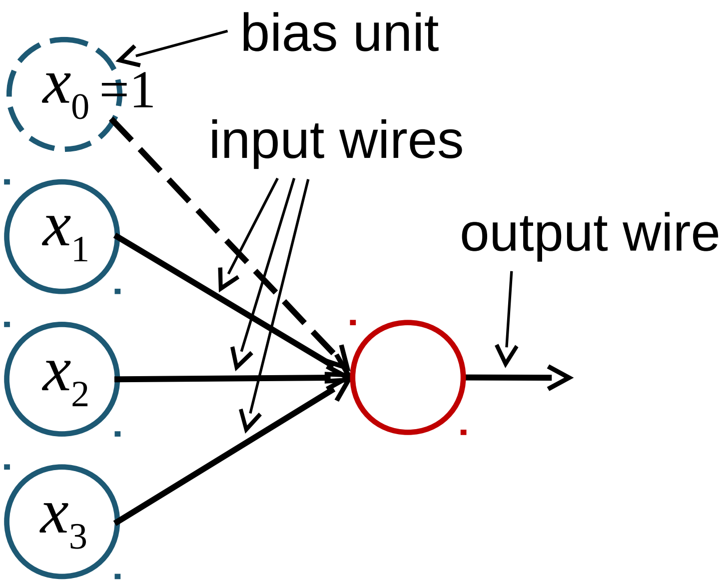


Neuron model: Logistic function



Neuron Model

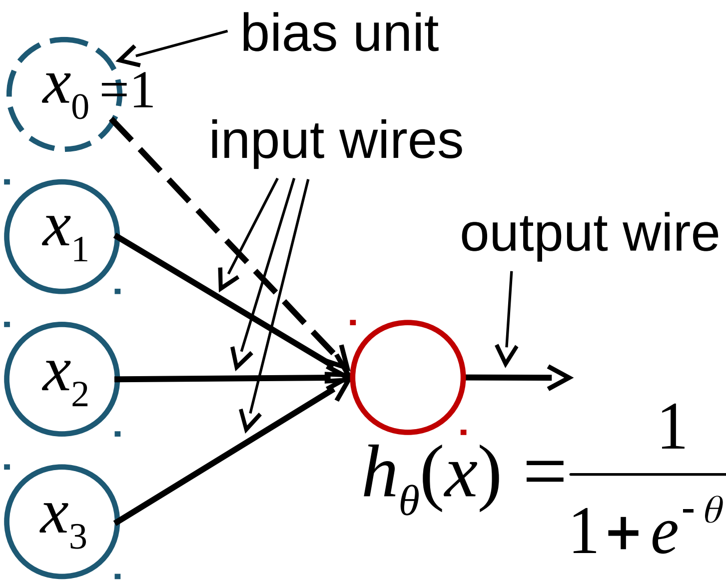
Neuron model: Logistic function



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Neuron Model

Neuron model: Logistic function



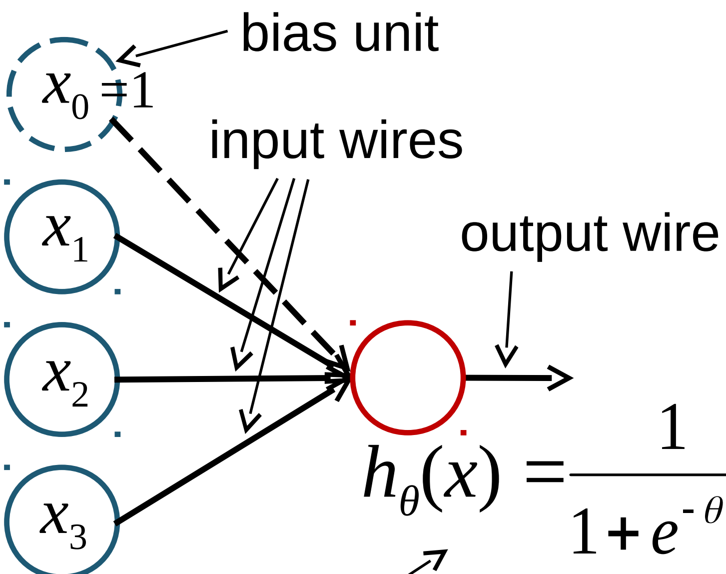
$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

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

The parameters θ are also called ‘weights’.

Neuron Model

Neuron model: Logistic function



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

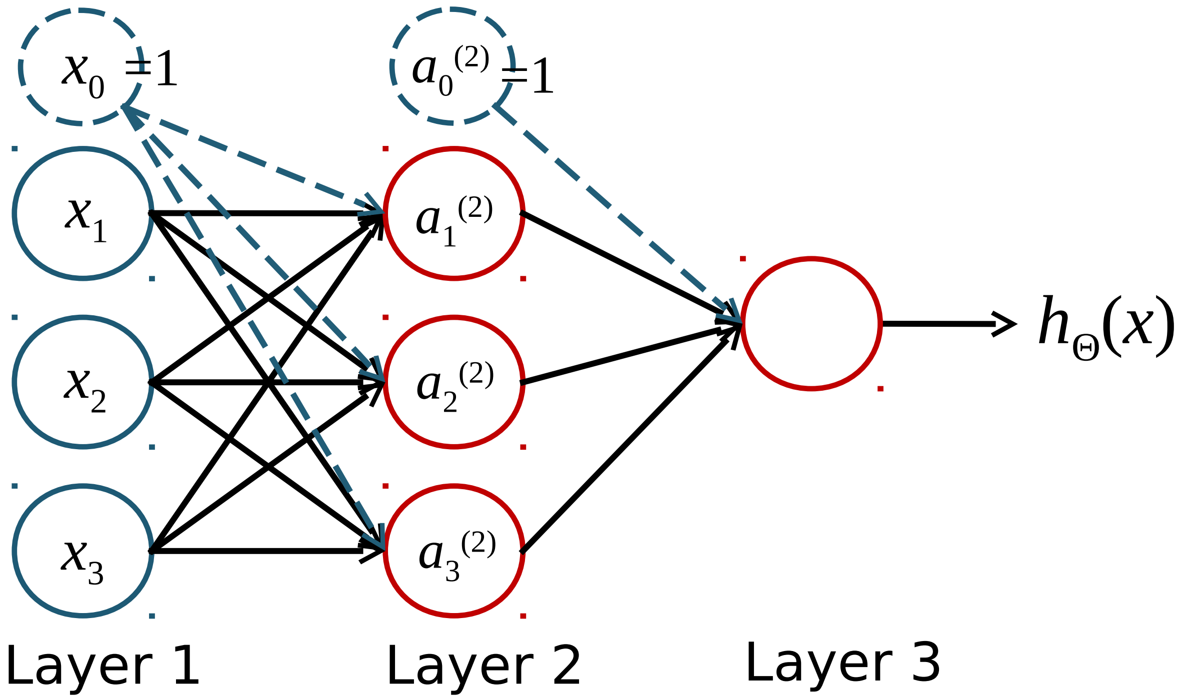
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

The parameters θ are also called ‘weights’.

Artificial neuron does the computation.
This computation is determined by the ***activation*** function.
In this example, activation function is sigmoid (logistic) function.

Neural Network

Neural Network is a bunch of neurons grouped in layers. First layer is the *input layer* and last one (Layer 3 in this example) is the *output layer*.
The layers in between are called *hidden layers*.

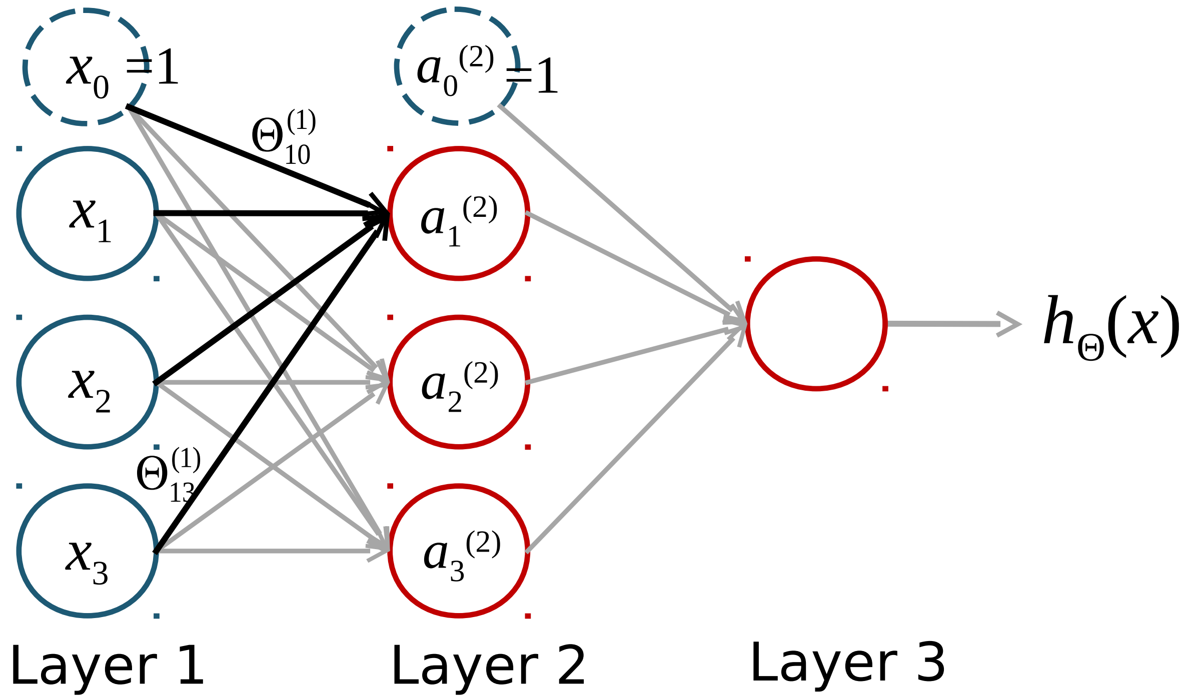


Forward Propagation

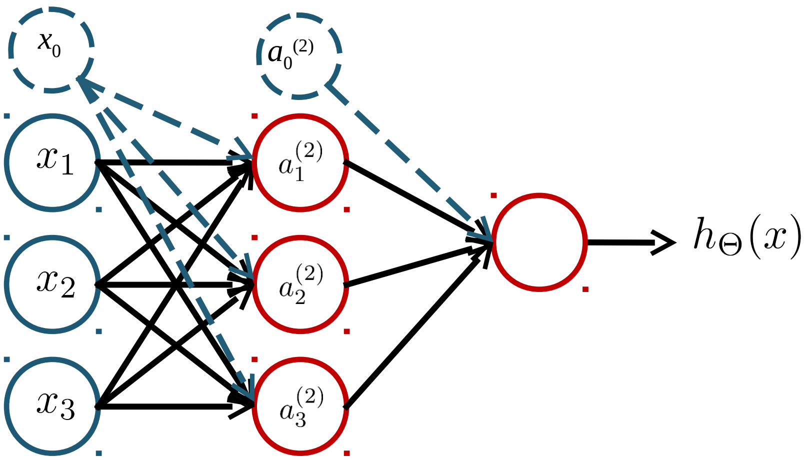
$$z_1^{(2)} = \Theta_{10}^{(1)} + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3$$

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

Remember g is the sigmoid function



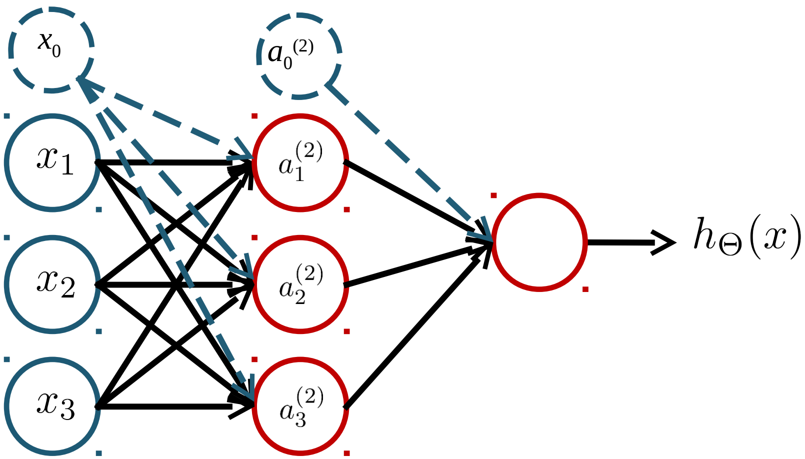
Forward Propagation



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

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

Forward Propagation

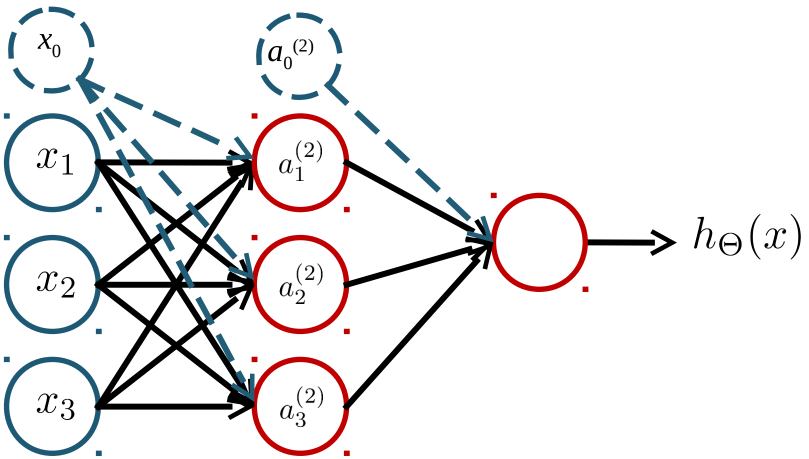


$a_i^{(j)} =$ "activation" of unit i in layer j

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

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

Forward Propagation



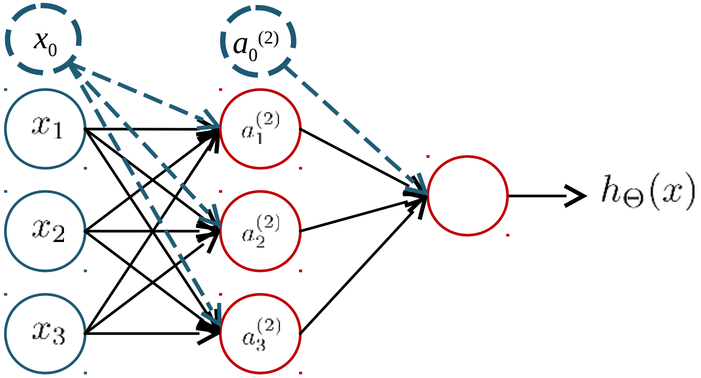
$a_i^{(j)} =$ "activation" of unit i in layer j

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

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

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

Forward Propagation



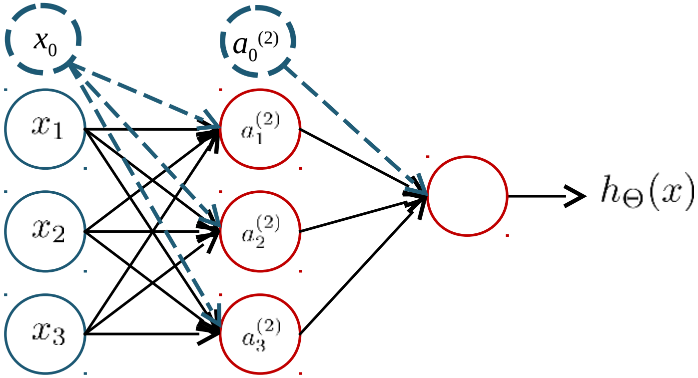
$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

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

Forward Propagation



$$a_1^{(2)} = g(\underbrace{\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3}_{z_1^{(2)}})$$

$$a_2^{(2)} = g(\underbrace{\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3}_{z_2^{(2)}})$$

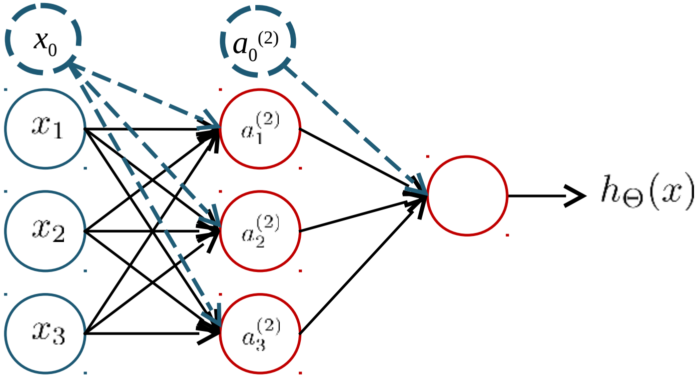
$$a_3^{(2)} = g(\underbrace{\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3}_{z_3^{(2)}})$$

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

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

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

Forward Propagation



$$a_1^{(2)} = g(\underbrace{\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3}_{z_1^{(2)}})$$

$$a_2^{(2)} = g(\underbrace{\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3}_{z_2^{(2)}})$$

$$a_3^{(2)} = g(\underbrace{\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3}_{z_3^{(2)}})$$

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

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

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

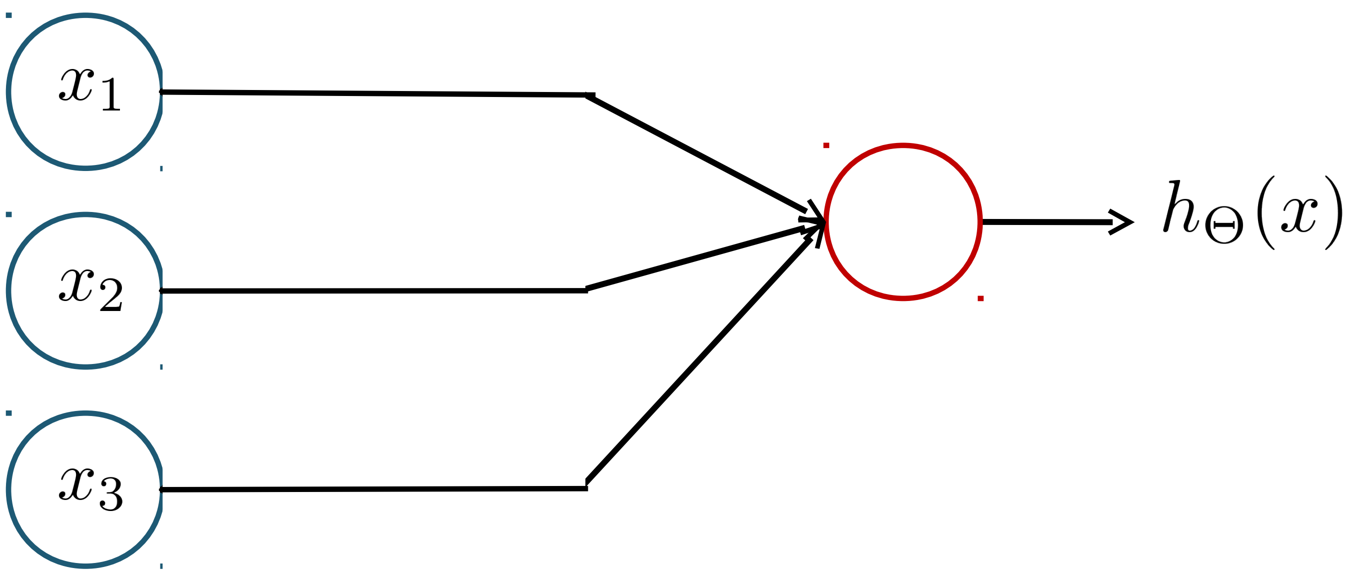
Add $a_0^{(2)} = 1$.

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

This process is called *forward propagation*.

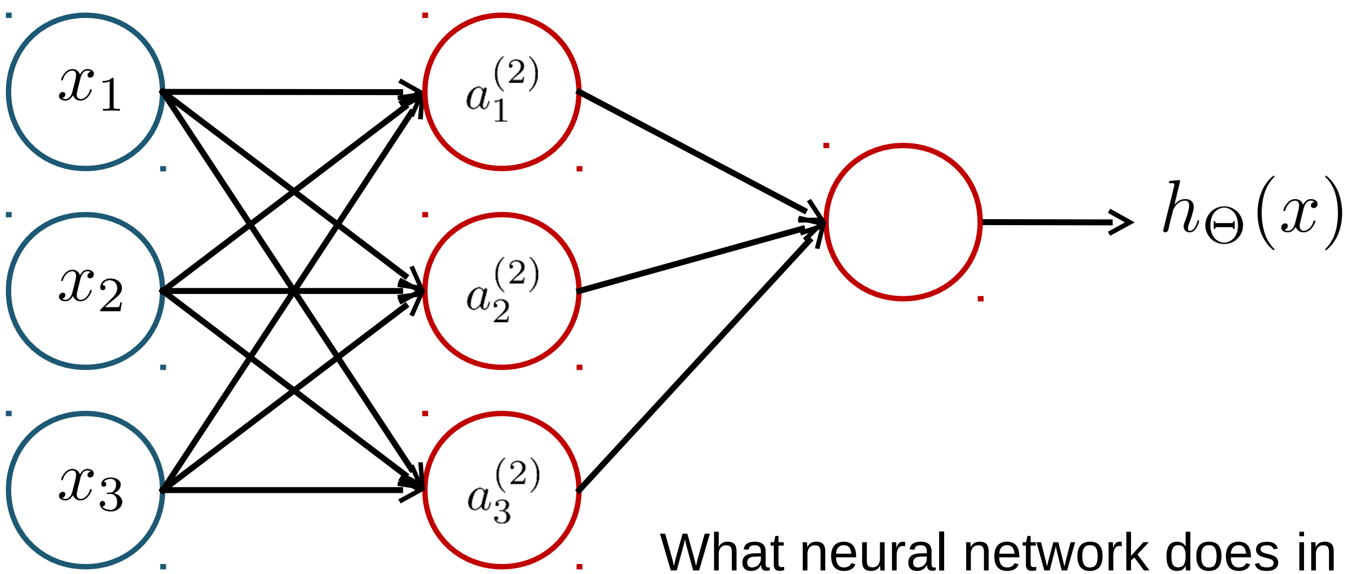
Neural Network Learning

Neural network learns its own features:



Neural Network Learning

Neural network learns its own features:

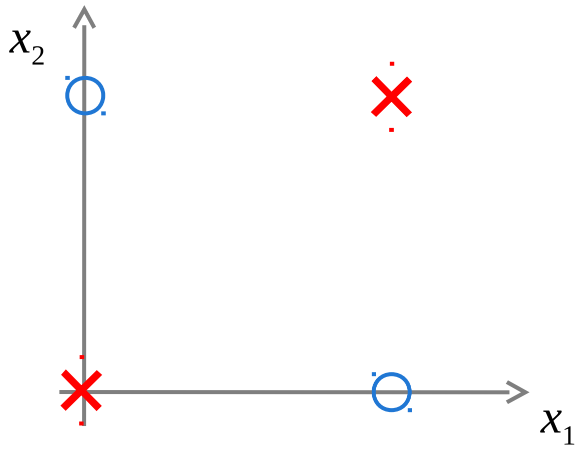


What neural network does in this example is the same with what logistic regression does, but it creates new features (hidden units $a_1^{(2)}$, $a_2^{(2)}$, $a_3^{(2)}$) and learns them.

Examples

Non-linear classification example: XNOR
Let's build a neural network to represent XNOR.

x_1, x_2 are binary (0 or 1).



y=0 if x_1 XOR x_2

y=1 if NOT (x_1 XOR x_2)= x_1 XNOR x_2

(So red crosses are positive class)

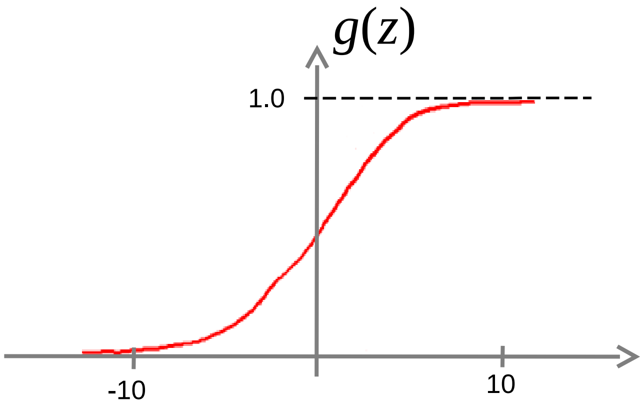
Hint: x_1 XNOR x_2 = (NOT x_1 AND NOT x_2) OR (x_1 AND x_2)

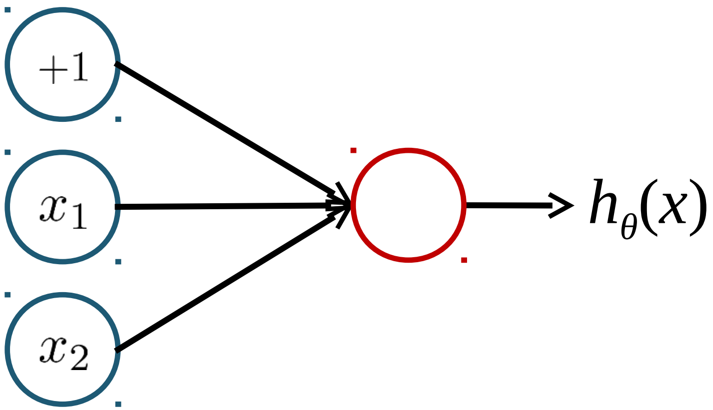
Examples

AND operator:
Let a one-unit network compute AND

x_1, x_2 are binary (0 or 1).

$y = x_1 \text{ AND } x_2$



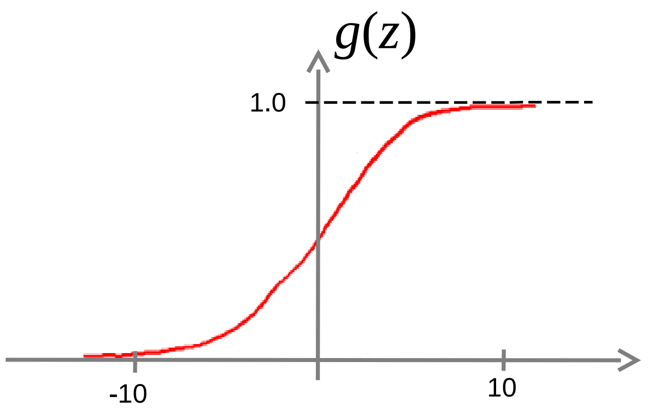


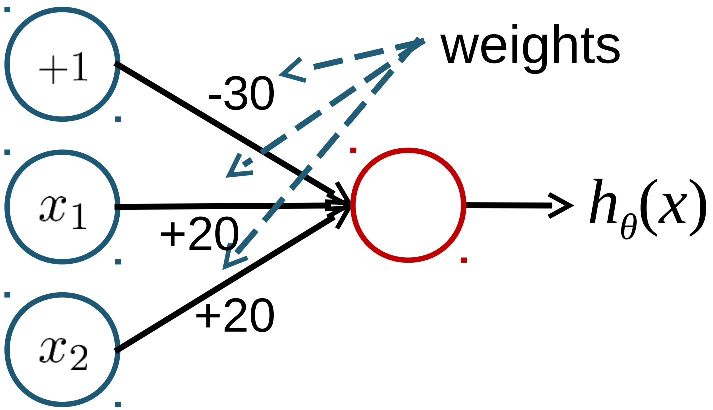
Examples

AND operator:
Let a one-unit network compute AND

x_1, x_2 are binary (0 or 1).

$y = x_1 \text{ AND } x_2$



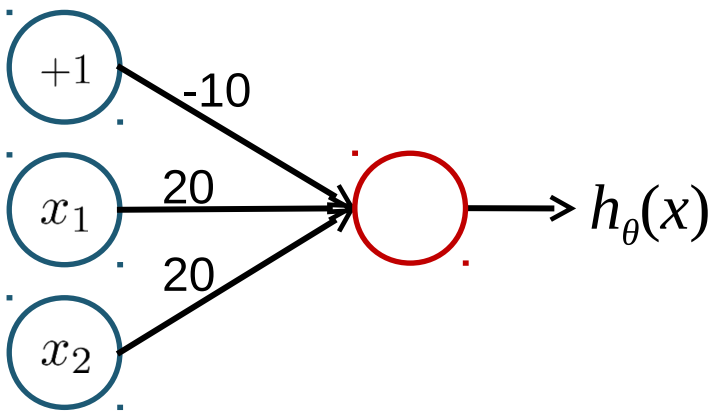


x_1	x_2	$h_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

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

$$h_\theta(x) \approx x_1 \text{ AND } x_2$$

OR operator:

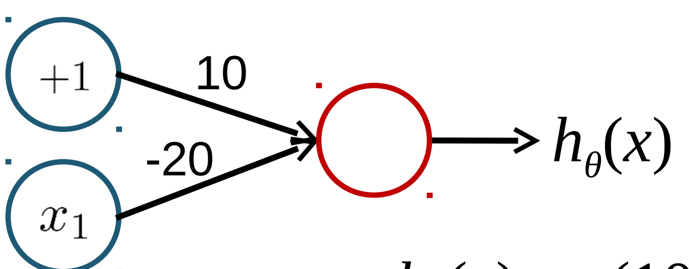


$$h_\theta(x) = g(-10+20x_1+20x_2)$$

x_1	x_2	$h_\theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$

$$h_\theta(x) \approx x_1 \text{ OR } x_2$$

NOT operator:

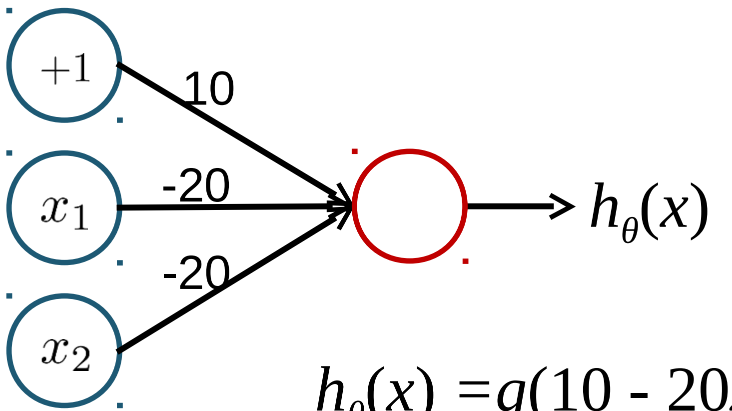


x_1	$h_\theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

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

Examples

(NOT x_1) AND (NOT x_2):

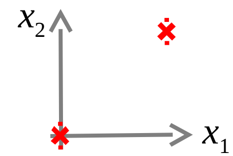


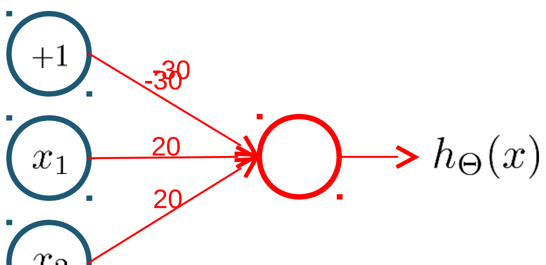
$$h_\theta(x) = g(10 - 20x_1 - 20x_2)$$

x_1	x_2	$h_\theta(x)$
0	0	$g(10) \approx 1$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(-30) \approx 0$

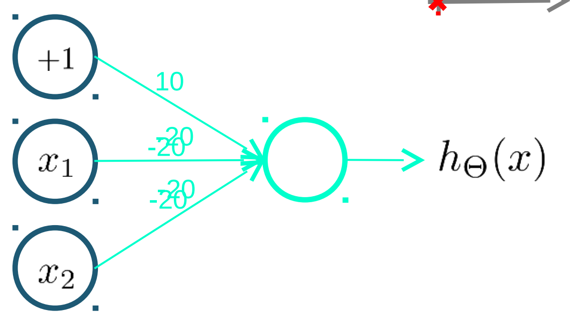
Examples

Putting all together: x_1 XNOR x_2

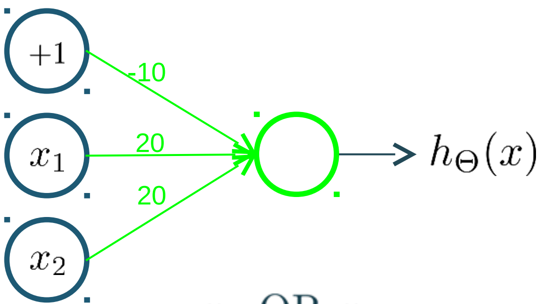




x_1 AND x_2



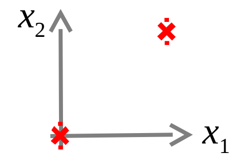
(NOT x_1) AND (NOT x_2)

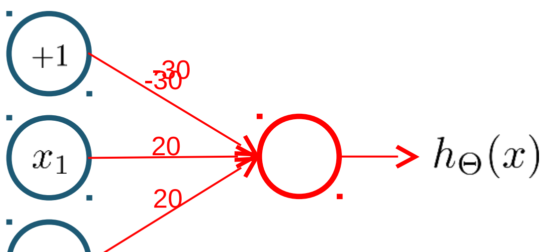


x_1 OR x_2

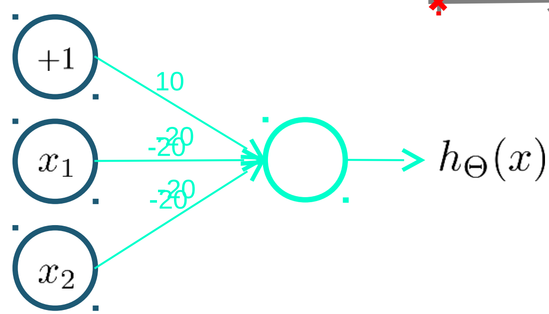
Examples

Putting all together: x_1 XNOR x_2

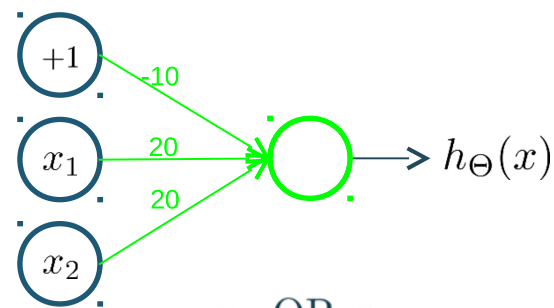




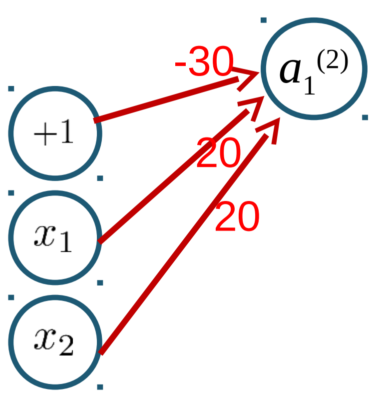
x_1 AND x_2



(NOT x_1) AND (NOT x_2)

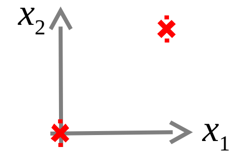


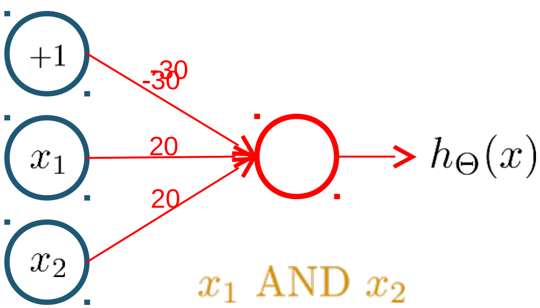
x_1 OR x_2

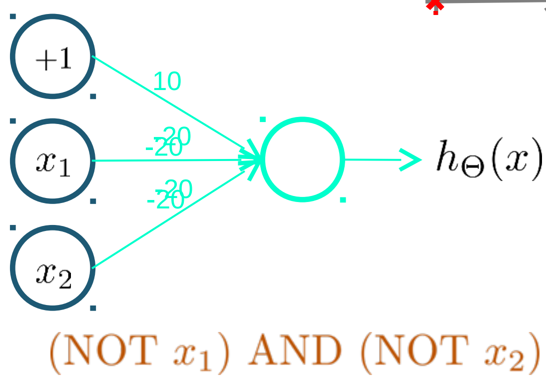


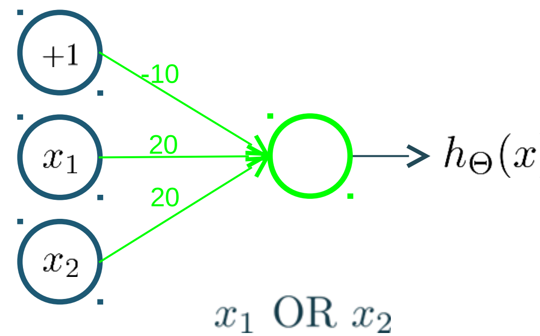
Examples

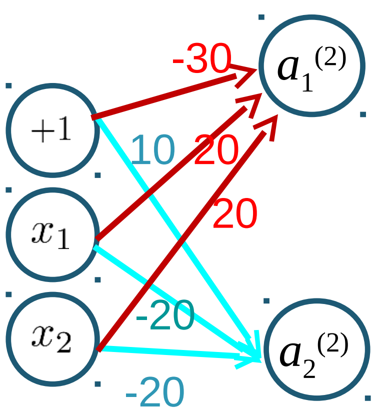
Putting all together: x_1 XNOR x_2





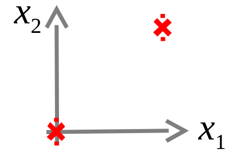


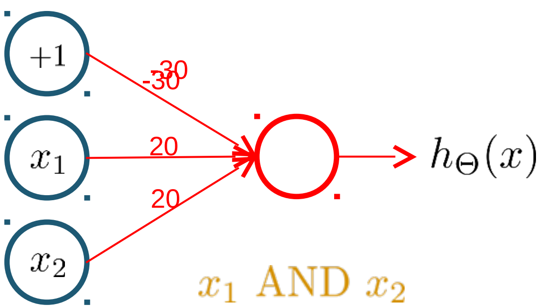


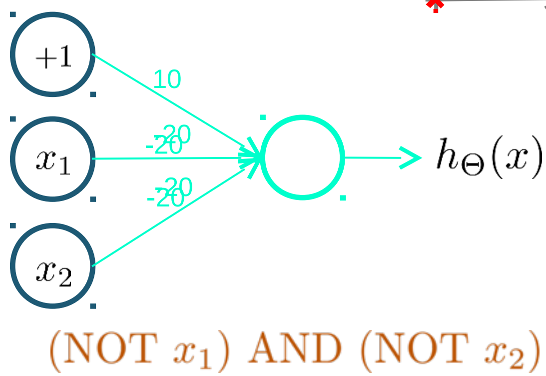


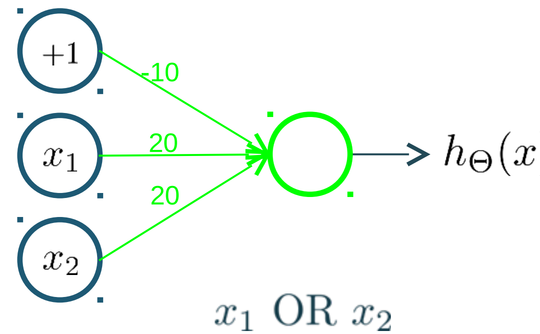
Examples

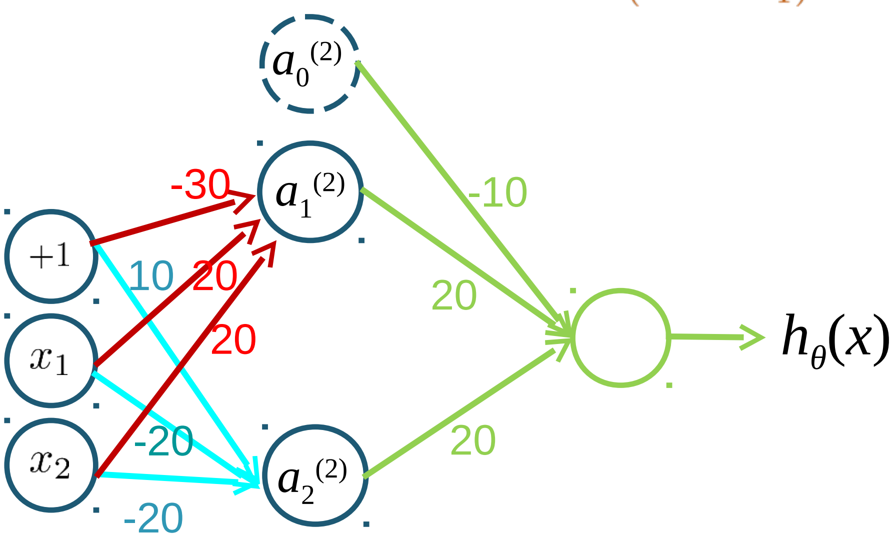
Putting all together: x_1 XNOR x_2







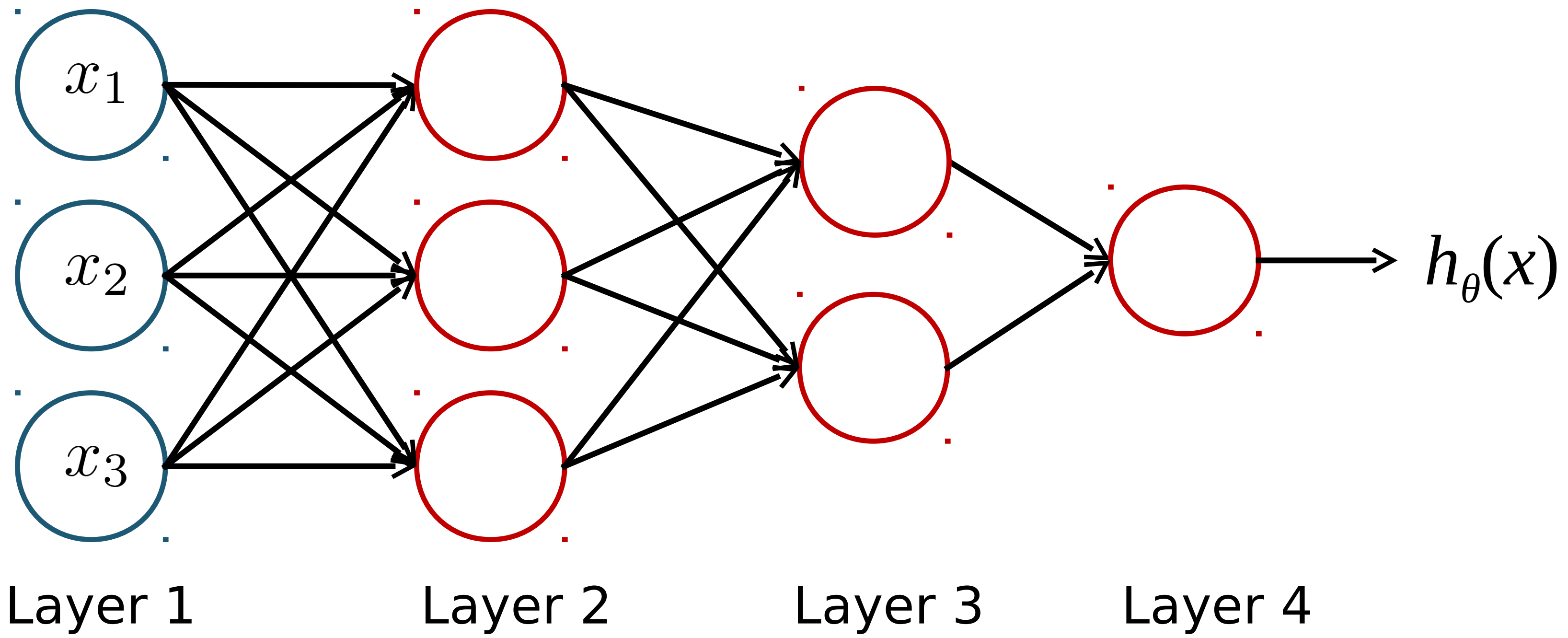




x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Neural Network Intuition

As we go further in layers, more complex functions are modeled and computed.



Multi-class Classification

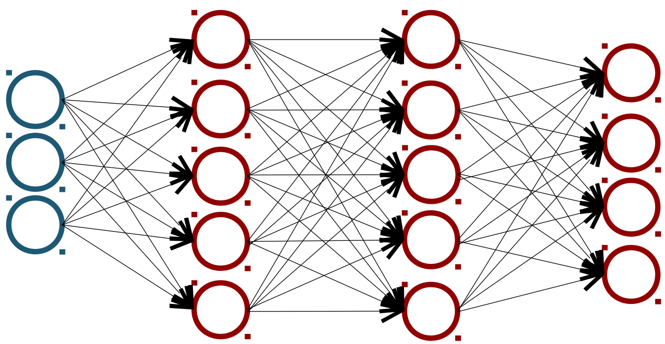
Multiple output units: One-vs-rest

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(x) \in \mathbb{R}^4$$

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$

when pedestrian when car when motorcycle

So far

We have learned about:

- Analogy with the human brain
- Layers of neural networks
- Forward propagation
- Examples to understand forward propagation

But how do neural networks 'learn' actually?

Learning corresponds to determining the 'weights' of units.

These weights are optimized using a **cost function**.

Cost Function

If we consider binary classification, i.e y=0 or 1,

There is one output unit and cost function is:

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

This cost function is the one we used for logistic regression. For each sample:

if y=1, cost becomes $-\log(h_\theta(x))$

if y=0, cost becomes $-\log(1-h_\theta(x))$

Cost Function

If we consider multi-class classification (K classes),

There are K output units.

$$y = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

pedestrian car motorcycle truck

Cost function:

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

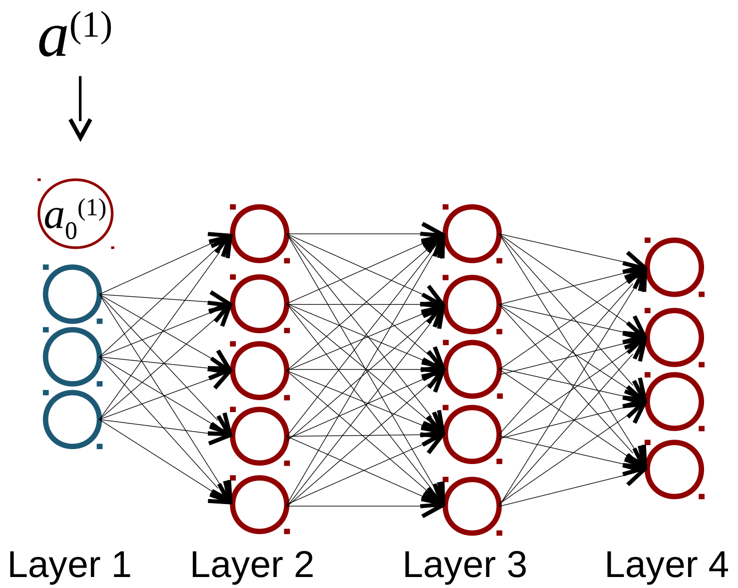
Gradient Computation: Forward Propagation

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

Given one training example (x,y),

Forward propagation:

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



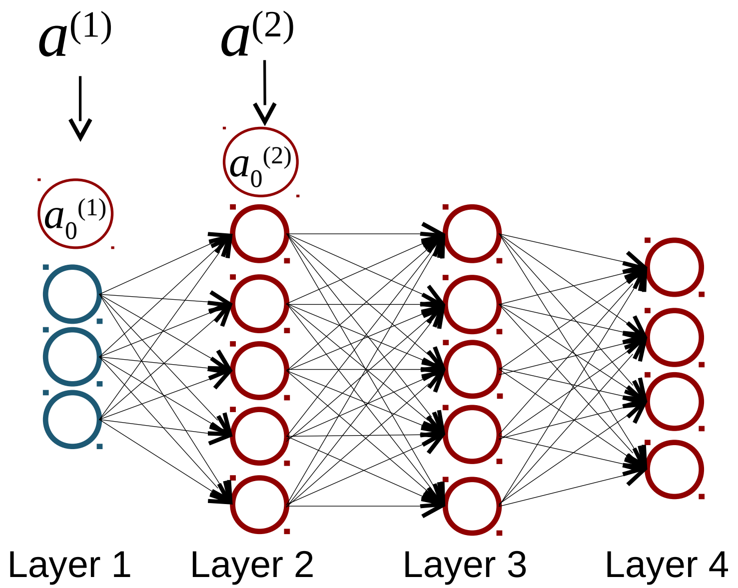
Gradient Computation: Forward Propagation

Need to compute $\dfrac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Given one training example (x,y),

Forward propagation:

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



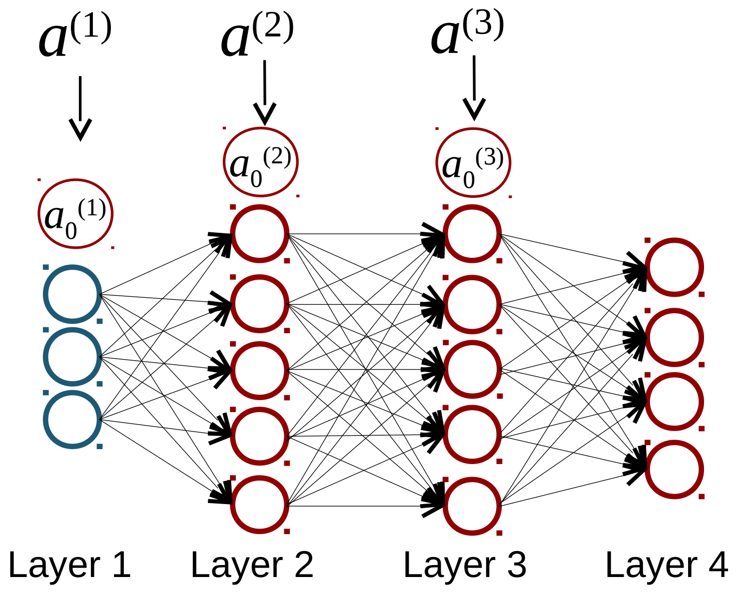
Gradient Computation: Forward Propagation

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

Given one training example (x,y),

Forward propagation:

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



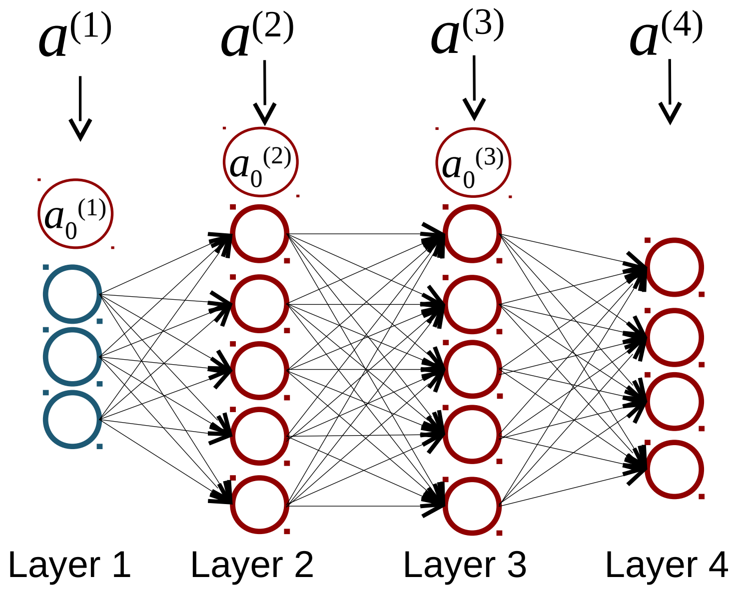
Gradient Computation: Forward Propagation

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

Given one training example (x,y),

Forward propagation:

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

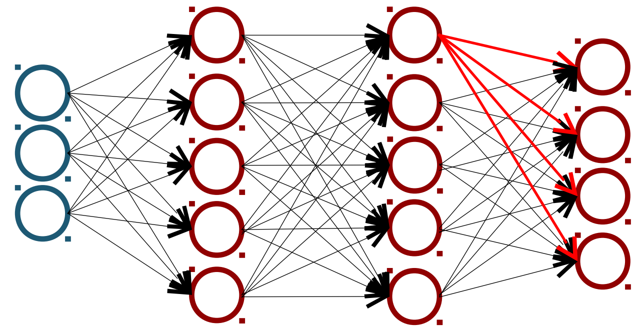


Gradient Computation: Backpropagation

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

For each output unit (layer 4)

$$\delta_j^{(4)} = \underbrace{a_j^{(4)}}_{h_\theta(x)_j} - y_j$$



$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)}$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)}$$

E.g.: $\delta_1^{(3)} = \Theta_{11}^{(3)}\delta_1^{(4)} + \Theta_{21}^{(3)}\delta_2^{(4)} + \Theta_{31}^{(3)}\delta_3^{(4)} + \Theta_{41}^{(3)}\delta_4^{(4)}$

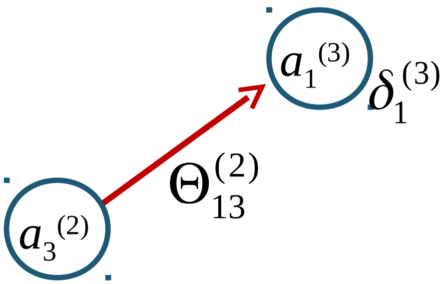
We 'backpropagate' the error.

Gradient Computation: Backpropagation

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

Derivation is not shown here, but $\dfrac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$

E.g. $\dfrac{\partial}{\partial \Theta_{13}^{(2)}} J(\Theta) = a_3^{(2)} \delta_1^{(3)}$



Complete Backpropagation Algorithm

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

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

Complete Backpropagation Algorithm

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

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

For $i = 1 \text{ to } m$

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

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Backpropagation: Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Complete Backpropagation Algorithm

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

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

For $i = 1 \text{ to } m$

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

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Backpropagation: Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

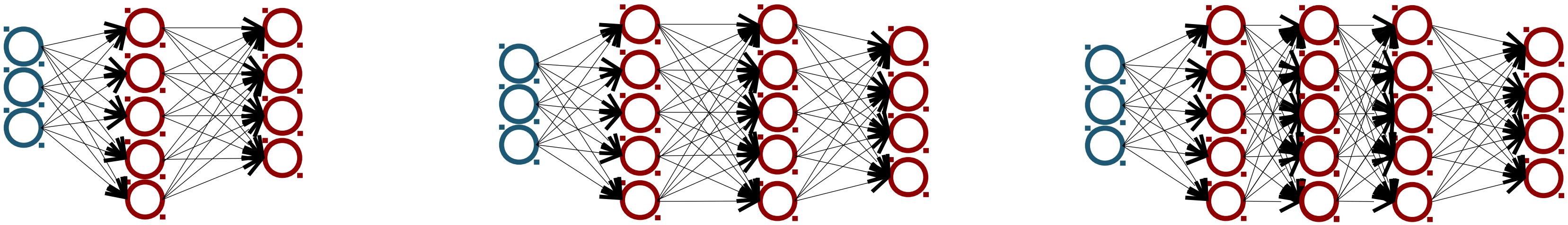
$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)}$

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

Putting It Together

Training a neural network:

Pick a network architecture (connectivity pattern between neurons)



No. of input units: Dimension of features (x_i)

No. output units: Number of classes

One or more hidden layers (usually the more the better):
- If there is more than one hidden layer, they have same number of hidden units in each.

Putting It Together

Training a neural network

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backpropagation to compute partial derivatives:
   $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$
5. Use gradient descent or an advanced optimization method with backpropagation and try to minimize $J(\Theta)$ as a function of parameters Θ