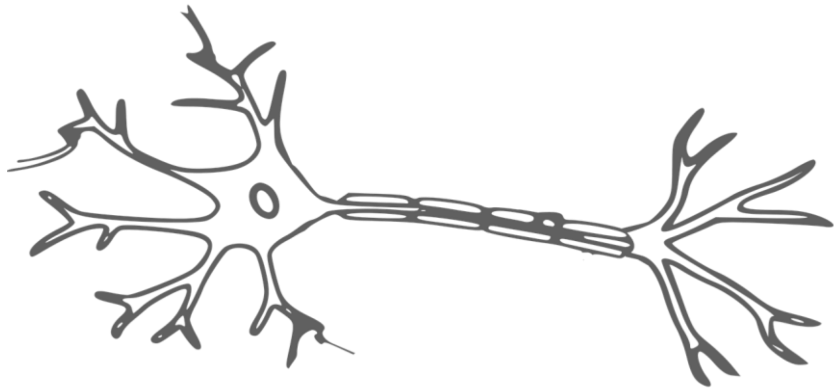
Neural Networks

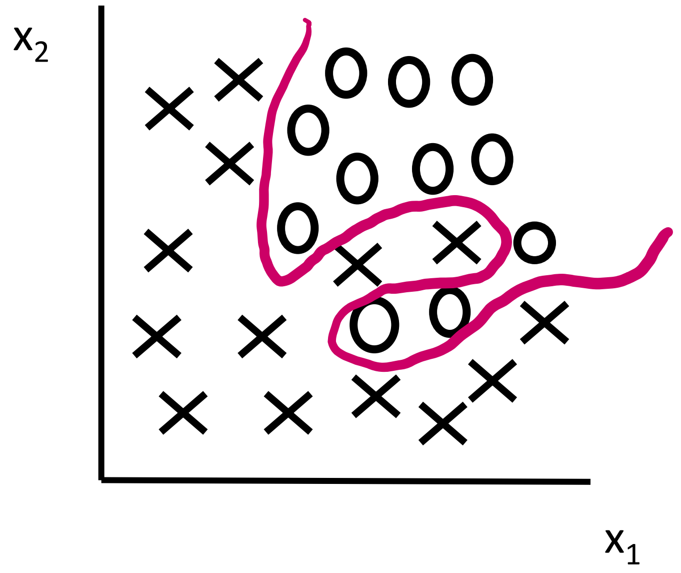
Prof. Dr. Christina Bauer

christina.bauer@th-deg.de

Faculty of Computer Science



Non-linear classification

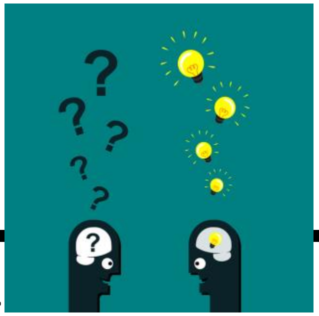


X_1 = size of the house
X_2 = no. of bedrooms
X_3 = no. of floors
X_4 = age of the house
….
X_{100}

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \ldots)$$

- Including „only“ the quadratic features → $O(n^2)$

QUESTION



Suppose you are learning to recognize cars from 100 x 100 pixel images (grayscale, not RGB). Let the features be pixel intensity values. If you train logistic regression including all the quadratic terms ($x_i x_j$) as features, about how many features will you have?

Hint: # of features: $m \times n + m \times n + C(m \times n, 2)$ (c = binomial coefficient)

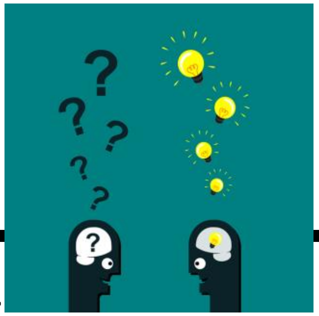
A: 5,000

B: 100,000

C: 50 million (5×10^7)

D: 5 billion (5×10^9)

QUESTION



Suppose you are learning to recognize cars from 100 x 100 pixel images (grayscale, not RGB). Let the features be pixel intensity values. If you train logistic regression including all the quadratic terms ($x_i x_j$) as features, about how many features will you have?

A: 5,000

B: 100,000

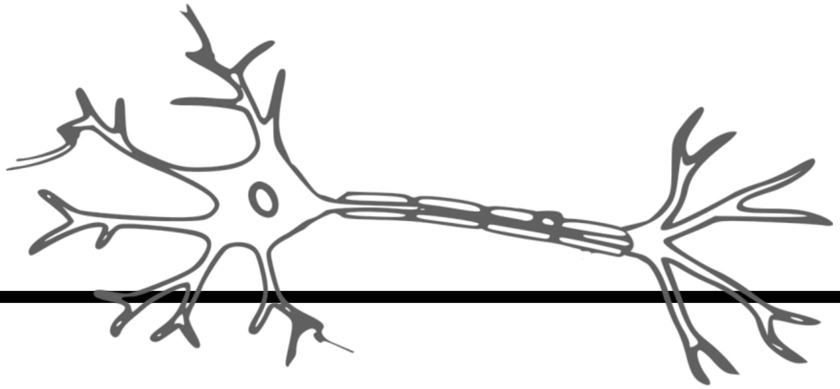
[✓ C]: 50 million (5×10^7) $\rightarrow$ $m \times n + m \times n + C(m \times n, 2)$ (c = binomial coefficient ~ 49,995,000)

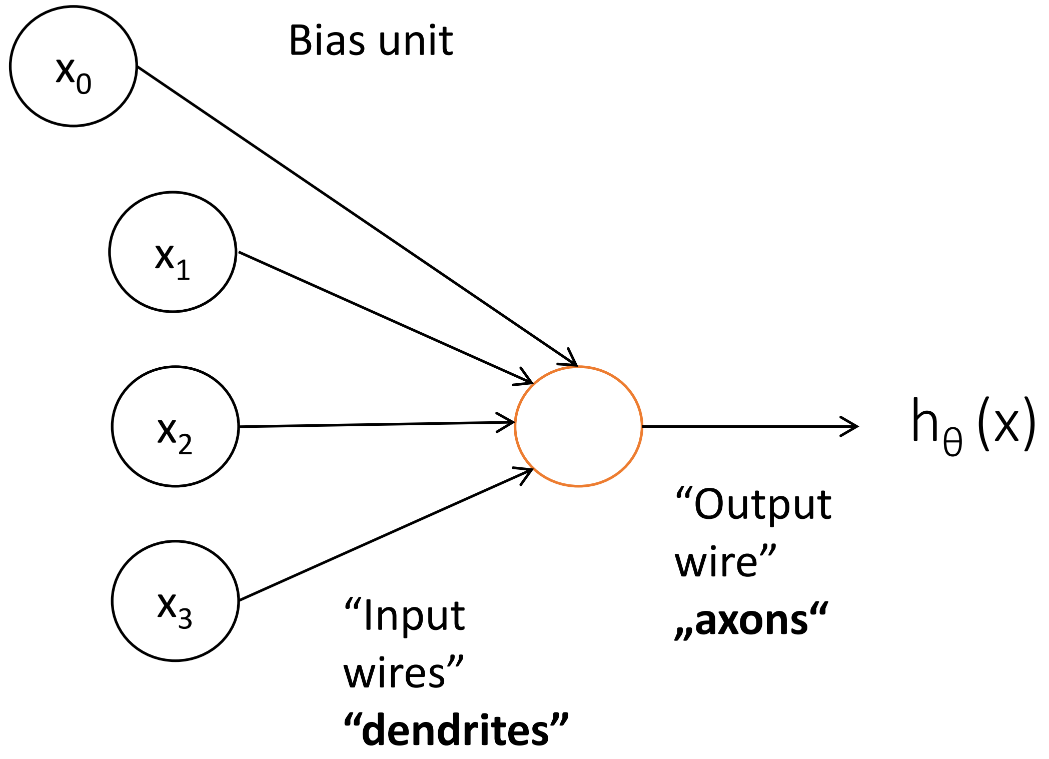
D: 5 billion (5×10^9)

Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- + Deep learning

Model representation – Single neuron





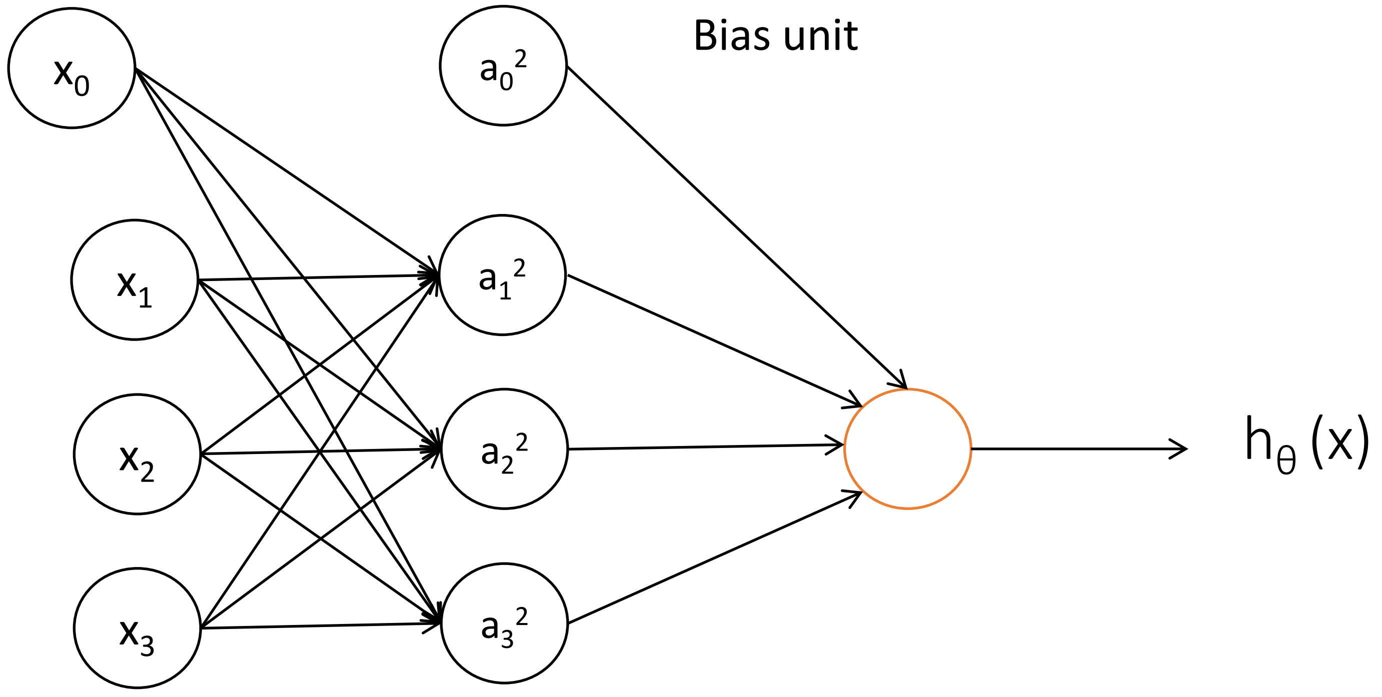
$$x = \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{bmatrix}$$

Parameters are often called **weights**

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

Neuron with sigmoid (logistic) **activation function**

NEURAL NETWORK

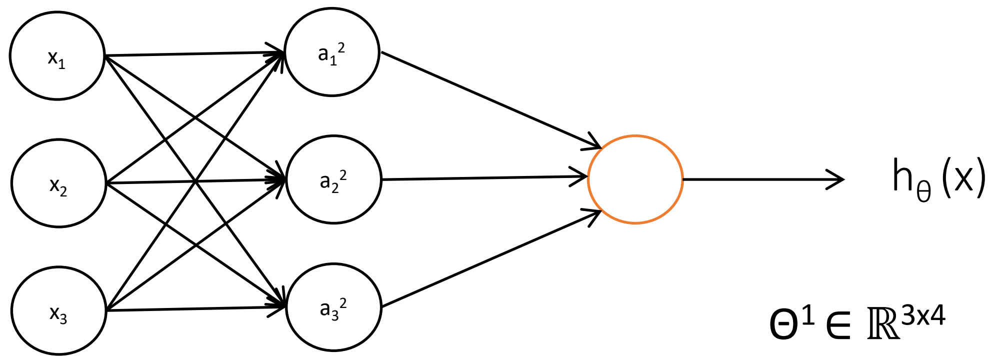


Layer 1
Input Layer

Layer 2
Hidden Layer

Layer 3
Output layer

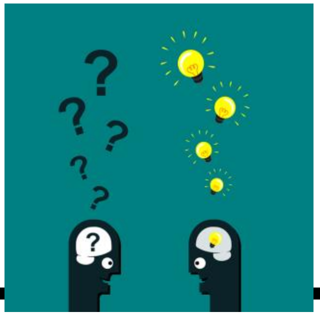
NEURAL NETWORK



- a_i^j = activation of unit i in layer j
- θ^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 the network has s_j units in layer j, s_{j+1} units in layer j+1, then Θ^j will be of dimension $s_{j+1} x (s_j + 1)$

QUESTION



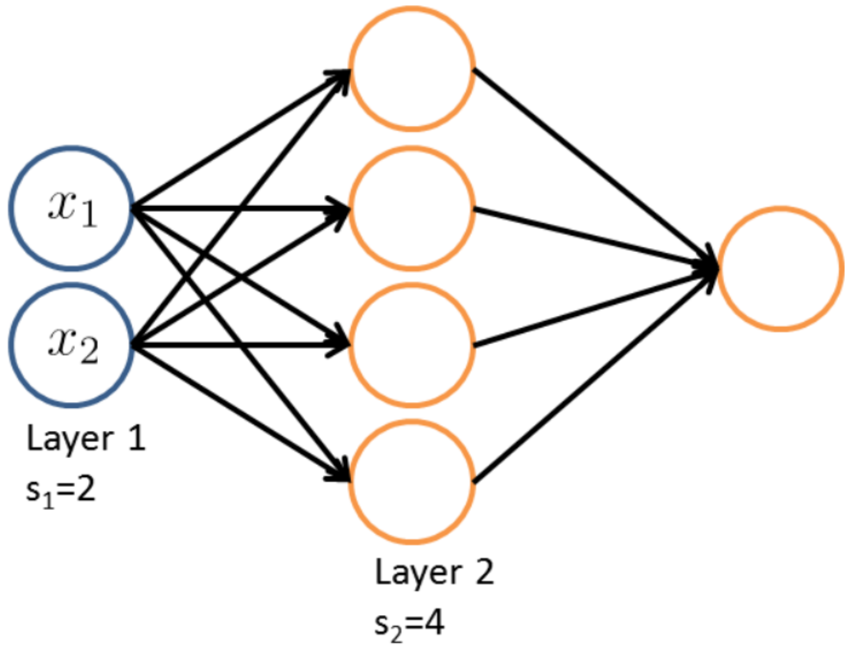
Consider the following neural network: What is the dimension of $\Theta^{(1)}$?

A: 2×4

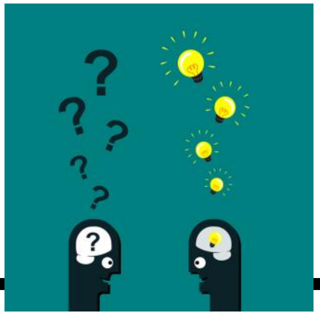
B: 4×2

C: 3×4

D: 4×3



QUESTION



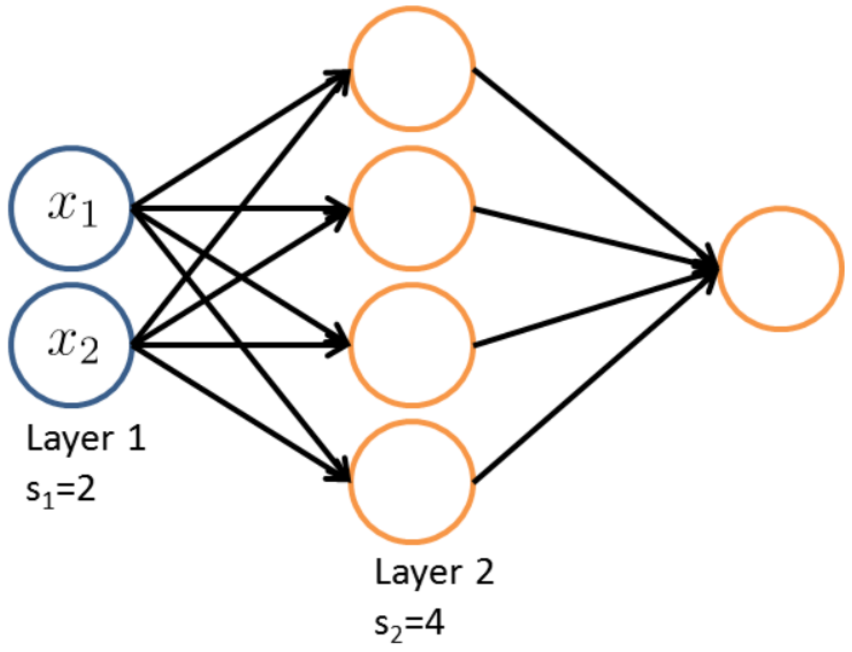
Consider the following neural network: What is the dimension of $\Theta^{(1)}$?

A: 2×4

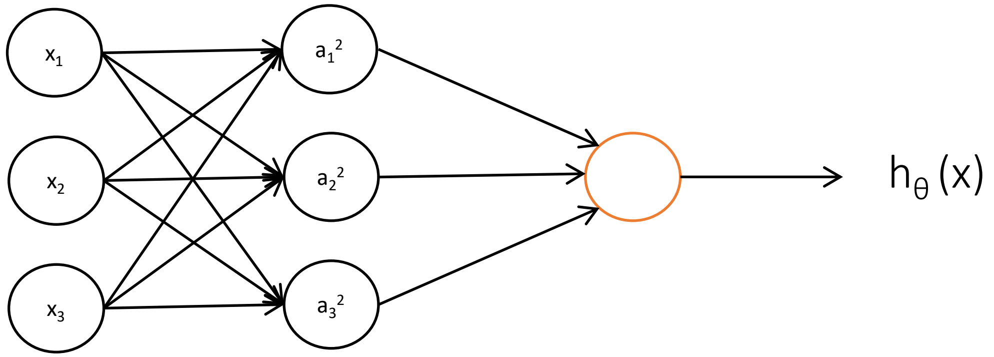
B: 4 x 2

C: 3 x 4

[✓ D: 4 x 3]



Forward propagation: Vectorized implementation



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

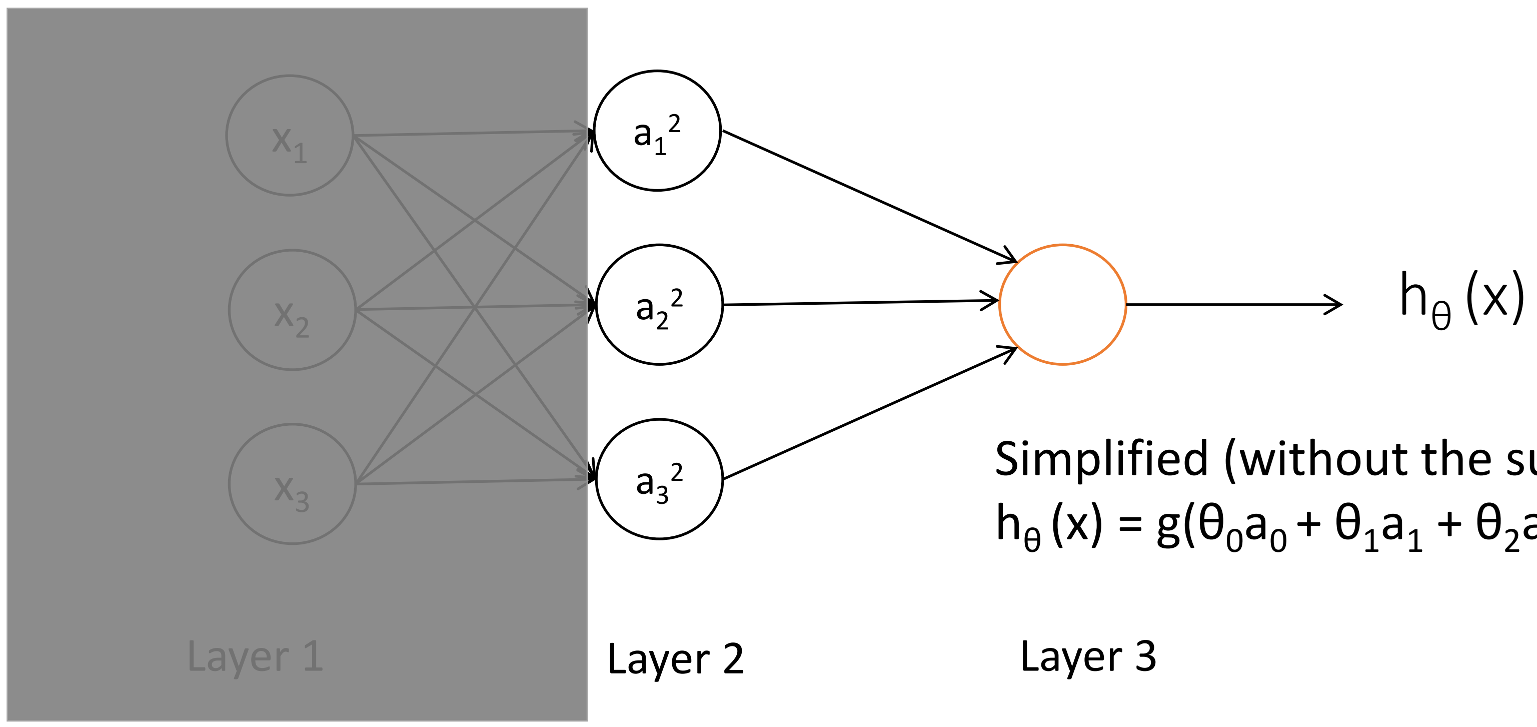
$\Theta^j x$

- $a_1^2 = g(\boxed{\theta_{10}^1 x_0 + \theta_{11}^1 x_1 + \theta_{12}^1 x_2 + \theta_{13}^1 x_3}) = g(z_1^2)$
- $a_2^2 = g(\boxed{\theta_{20}^1 x_0 + \theta_{21}^1 x_1 + \theta_{22}^1 x_2 + \theta_{23}^1 x_3}) = g(z_2^2)$
- $a_3^2 = g(\boxed{\theta_{30}^1 x_0 + \theta_{31}^1 x_1 + \theta_{32}^1 x_2 + \theta_{33}^1 x_3}) = g(z_2^2)$
- $h_\theta(x) = a_1^3 = g(\underline{\theta_{10}^2 a_0^2 + \theta_{11}^2 a_1^2 + \theta_{12}^2 a_2^2 + \theta_{13}^2 a_3^2})$
-

$z^2 = \Theta^1 x \rightarrow$ if x is definded as $a^1 \rightarrow z^2 = \Theta^1 a^1$
$a^2 = g(z^2) \rightarrow a^2, z^2 \in \mathbb{R}^3$

Bias unit: Add $a_0^2 = 1 \rightarrow a^2 \in \mathbb{R}^4$
Output $z^3 = \Theta^2 a^2$
$h_\theta(x) = a^3 = g(z^3)$

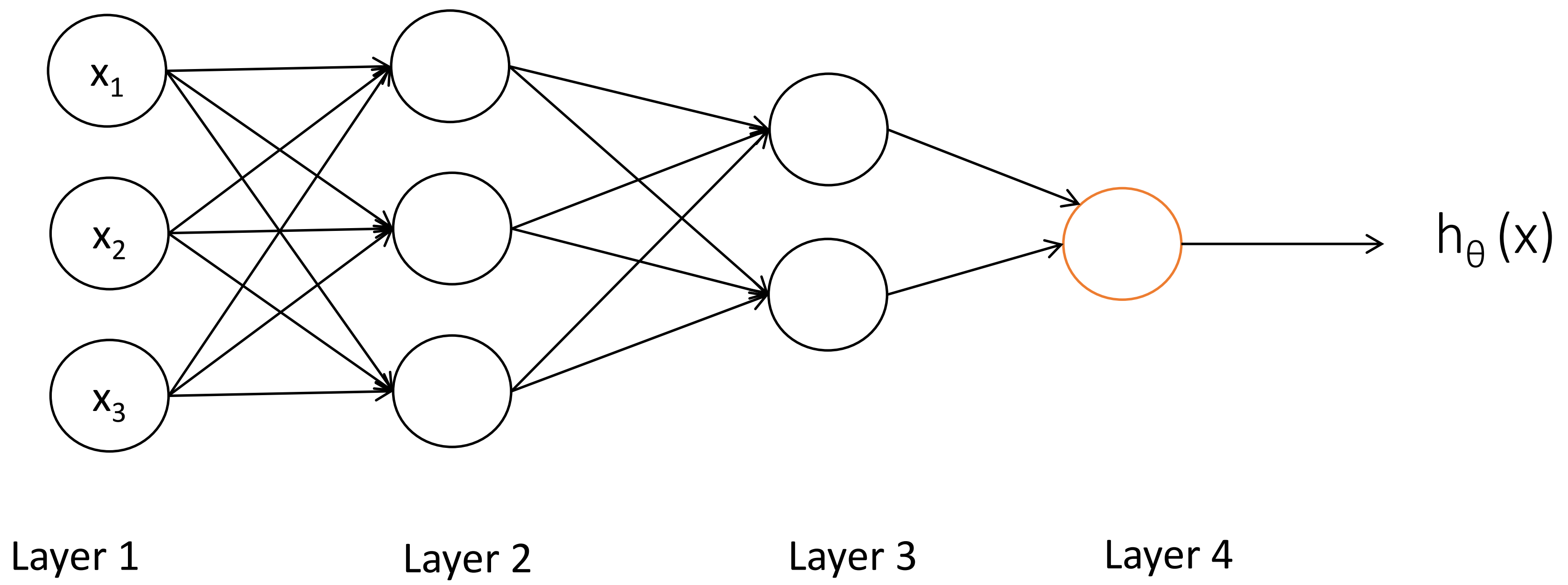
Neural Network learning its own features



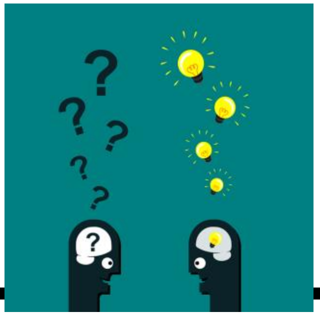
Simplified (without the superscript):

$h_\theta(x) = g(\theta_0 a_0 + \theta_1 a_1 + \theta_2 a_2 + \theta_3 a_3) \rightarrow$ Logistic Regression

Network Architectures



QUESTION



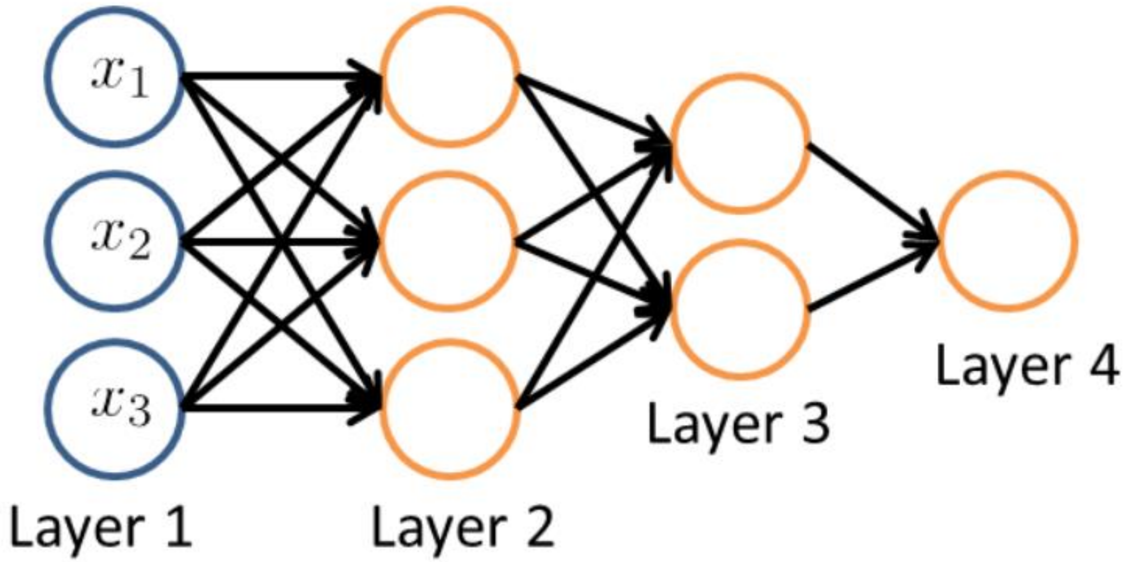
Consider the following neural network: Let $a^{(1)} = x \in R^{n+1}$ denote the input (with $a_0^{(1)} = 1$). How would you compute a^2?

A: $a^2 = \Theta^{(1)} a^{(1)}$

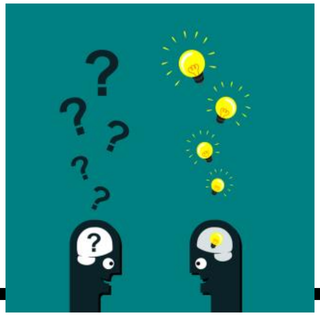
B: $z^2 = \Theta^{(2)} a^{(1)}$; $a^{(2)} = g(z^{(2)})$

C: $z^2 = \Theta^{(1)} a^{(1)}$; $a^{(2)} = g(z^{(2)})$

D: $z^2 = \Theta^{(2)} g(a^{(1)})$; $a^{(2)} = g(z^{(2)})$



QUESTION



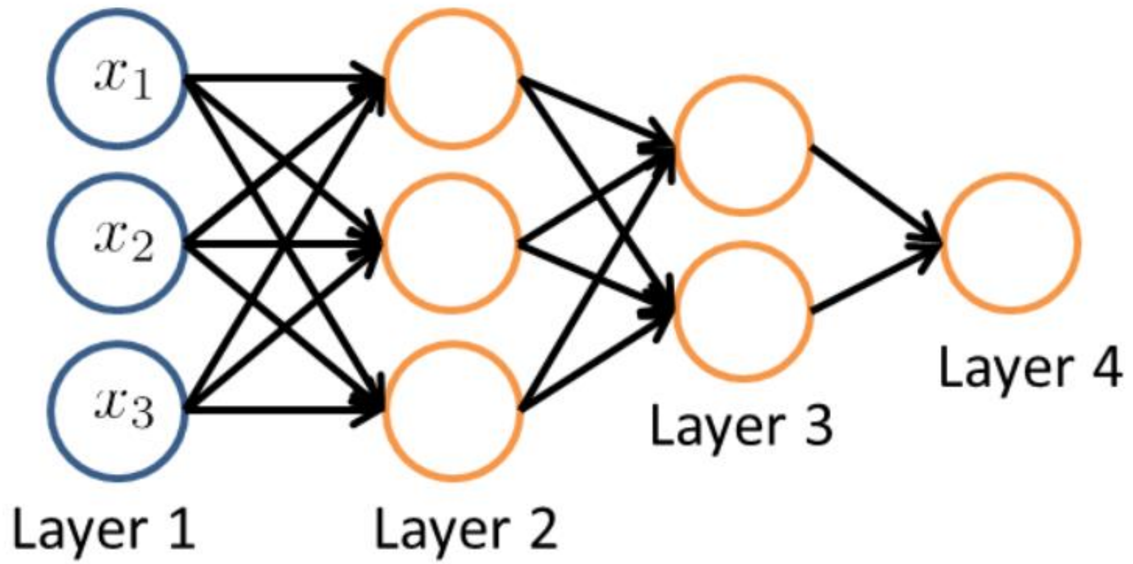
Consider the following neural network: Let $a^{(1)} = x \in R^{n+1}$ denote the input (with $a_0^{(1)} = 1$). How would you compute a^2?

A: $a^2 = \Theta^{(1)} a^{(1)}$

B: $z^2 = \Theta^{(2)} a^{(1)}$; $a^{(2)} = g(z^{(2)})$

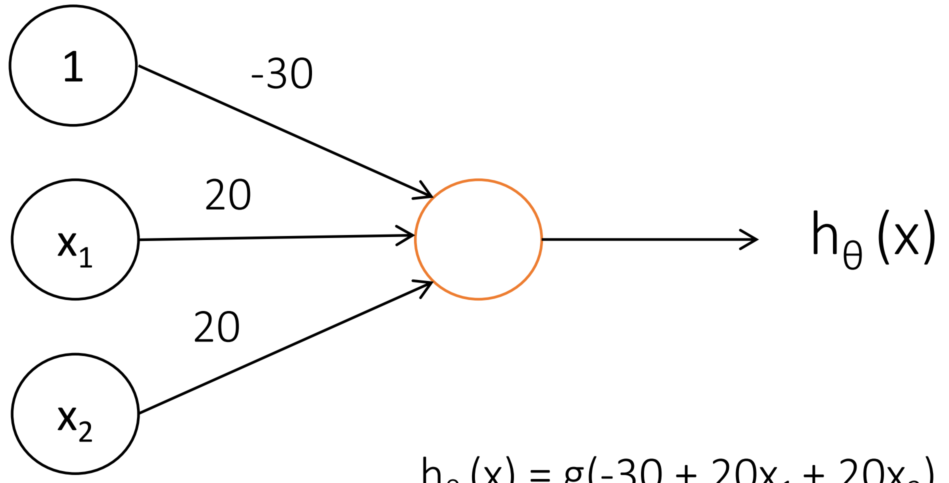
~~C~~: $z^2 = \Theta^{(1)} a^{(1)}$; $a^{(2)} = g(z^{(2)})$

D: $z^2 = \Theta^{(2)} g(a^{(1)})$; $a^{(2)} = g(z^{(2)})$



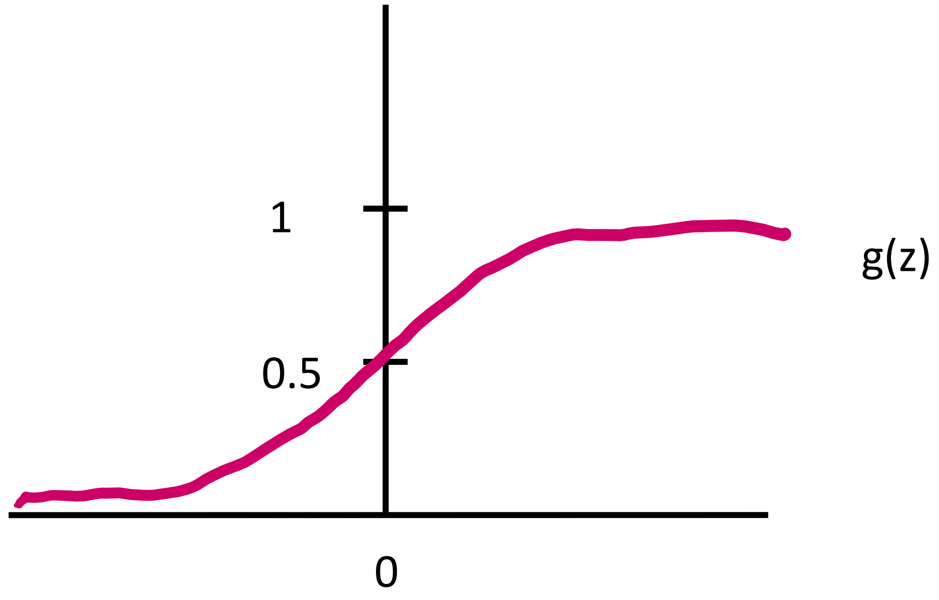
Example: And

- $x_1, x_2 \in \{1,0\}$
- $Y = x_1 \text{ AND } x_2$



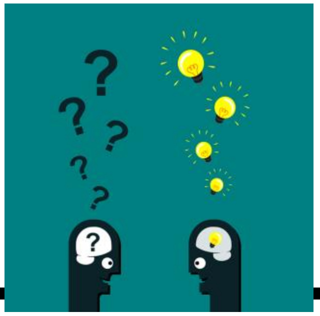
$h_\theta(x) = g(-30 + 20x_1 + 20x_2)$

$\Theta^{(1)}_{10}\ \Theta^{(1)}_{11}\ \Theta^{(1)}_{12}$



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

QUESTION



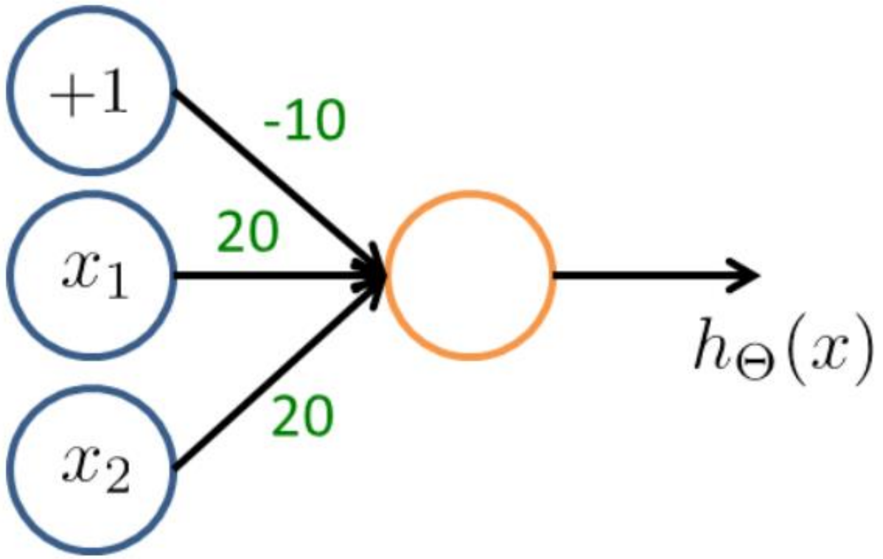
Suppose x_1 and x_2 are binary valued (0 or 1). What boolean function does the network shown below (approximately) compute?

A: x_1 AND x_2

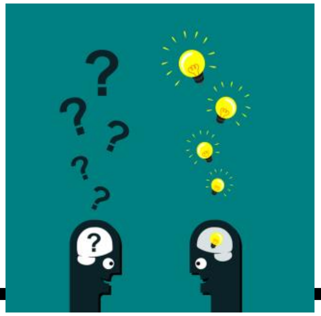
B: (NOT x_1) OR (NOT x_2)

C: x_1 OR x_2

D: (NOT x_1) AND (NOT x_2)



QUESTION



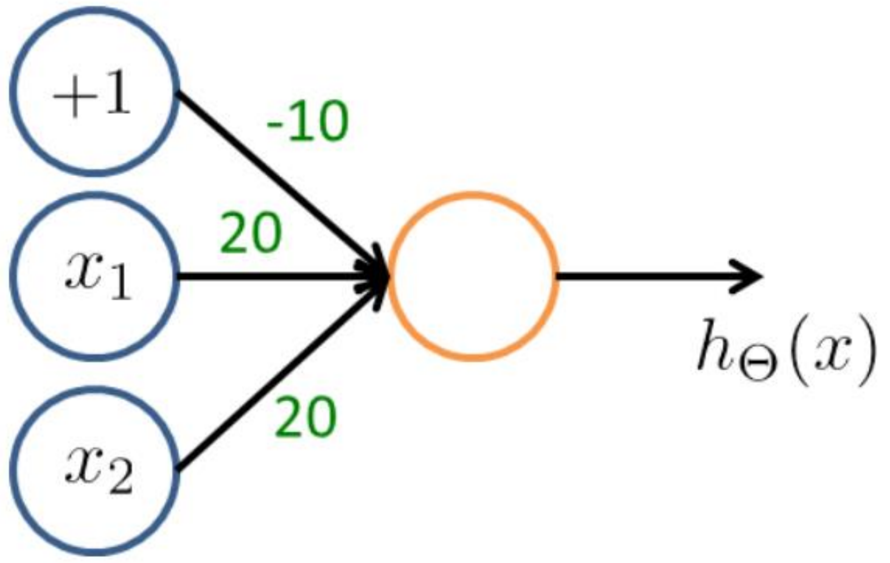
Suppose x_1 and x_2 are binary valued (0 or 1). What boolean function does the network shown below (approximately) compute?

A: x_1 AND x_2

B: (NOT x_1) OR (NOT x_2)

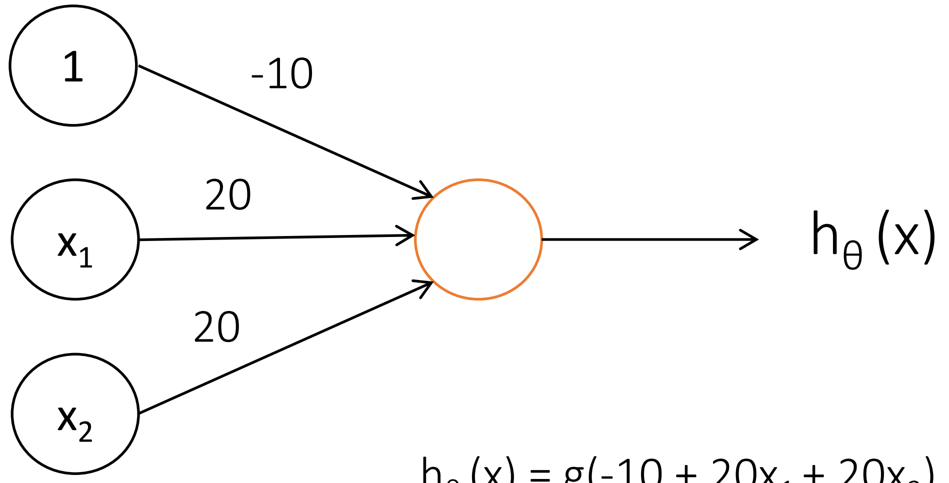
~~C~~: x_1 OR x_2

D: (NOT x_1) AND (NOT x_2)



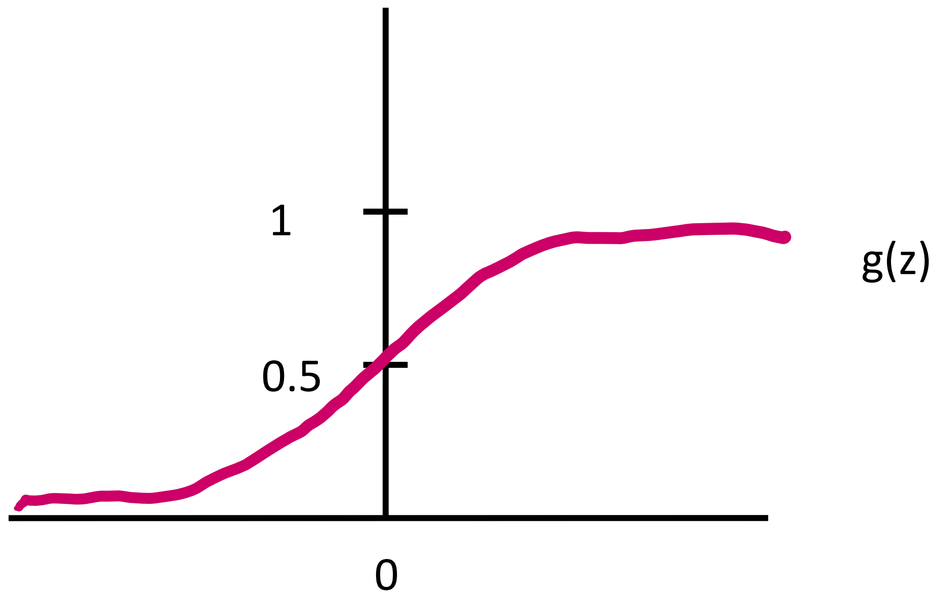
EXAMPLE: OR

- $x_1, x_2 \in \{1,0\}$
- $Y = x_1 \text{ OR } x_2$



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

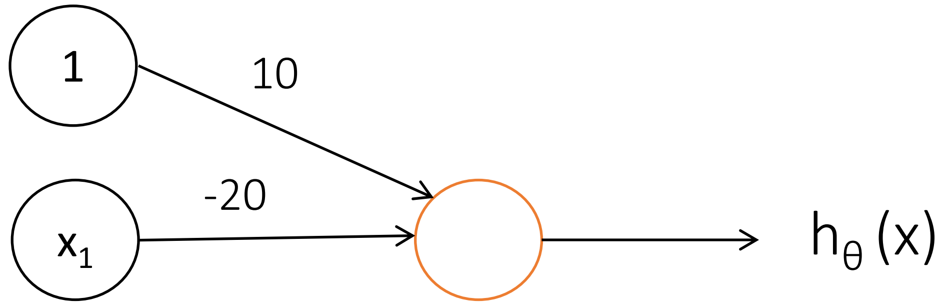
$\Theta^{(1)}_{10}\ \Theta^{(1)}_{11}\ \Theta^{(1)}_{12}$



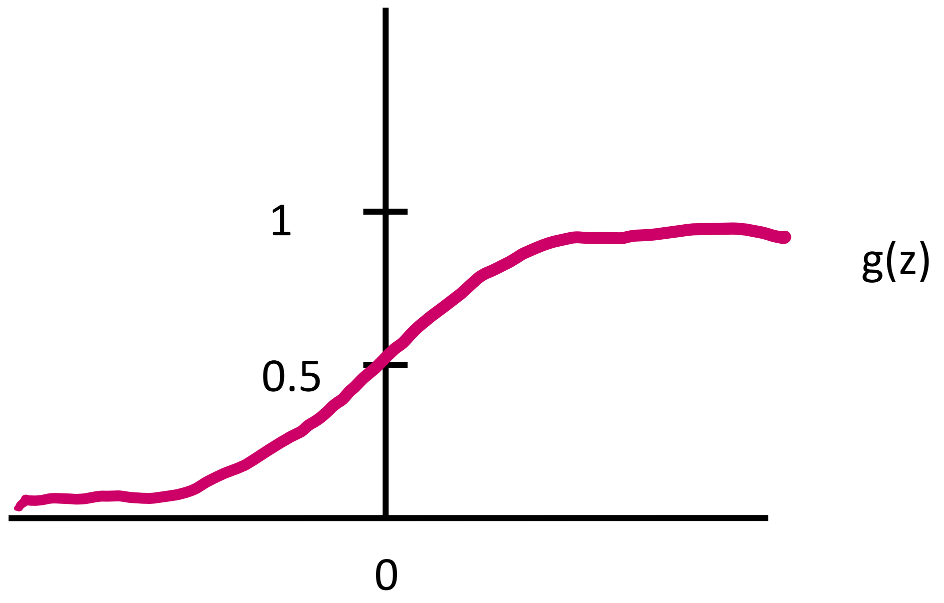
x_1	x_2	$h_\theta(x)$
0	0	g(-10) ~ 0
0	1	g(10) ~ 1
1	0	g(10) ~ 1
1	1	g(30) ~ 1

Example: Negation

- $x_1, x_2 \in \{1,0\}$
- $Y = \text{NOT } x_1$

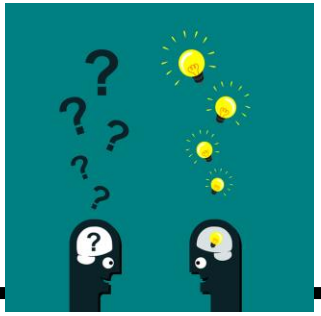


$h_\theta(x) = g(10 - 20x_1)$
$\Theta^{(1)}_{10}\ \Theta^{(1)}_{11}$



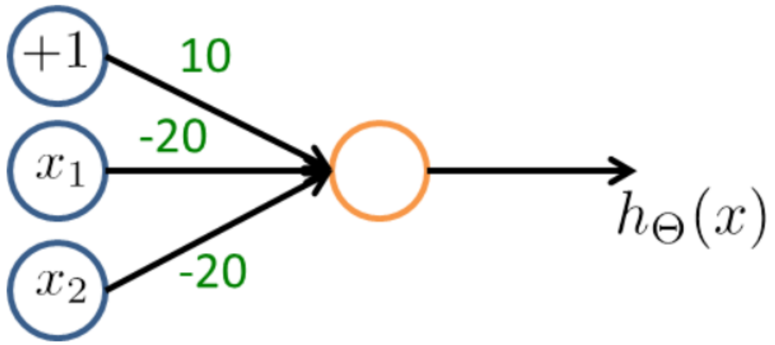
x_1	$h_\theta(x)$
0	g(10) ~ 1
1	g(-10) ~ 0

QUESTION

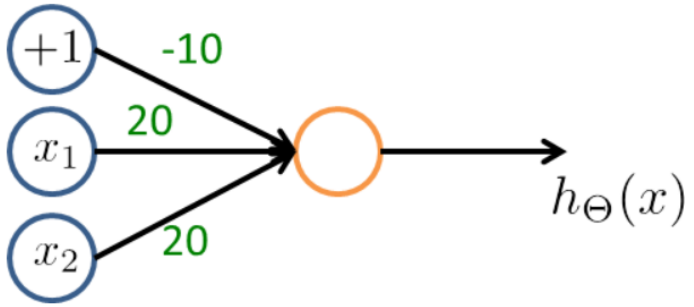


Suppose x_1 and x_2 are binary valued (0 or 1). Which of the following networks (approximately) computes the boolean function (NOT x_1) AND (NOT x_2)?

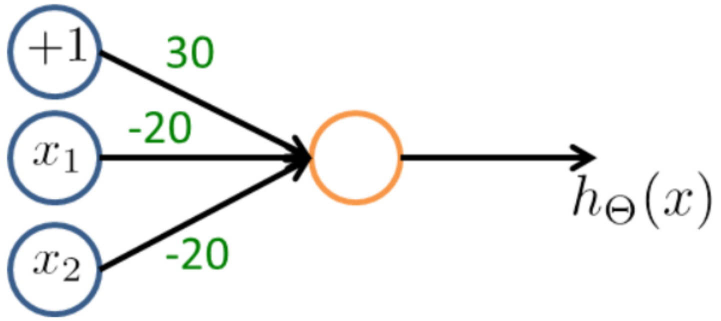
A



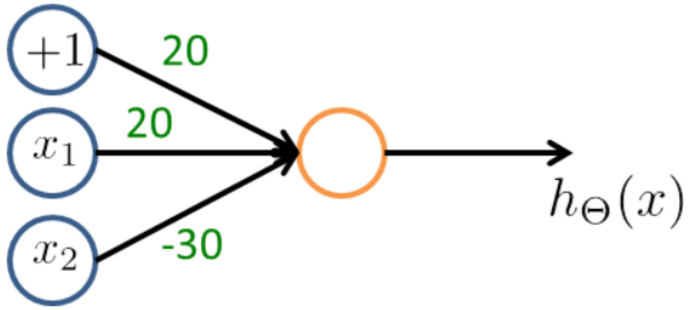
B



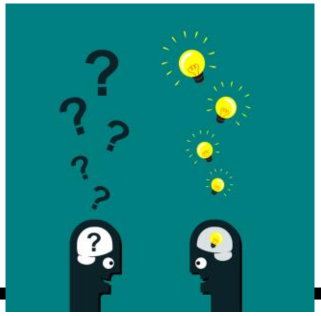
C



D

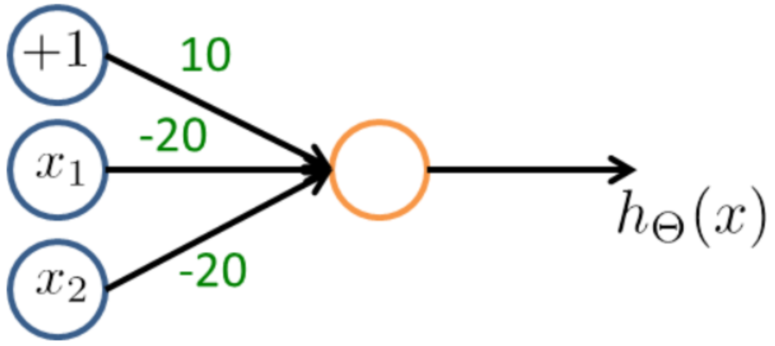


QUESTION

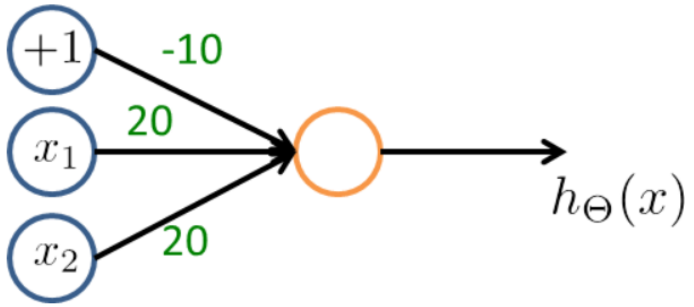


Suppose x_1 and x_2 are binary valued (0 or 1). Which of the following networks (approximately) computes the boolean function (NOT x_1) AND (NOT x_2)?

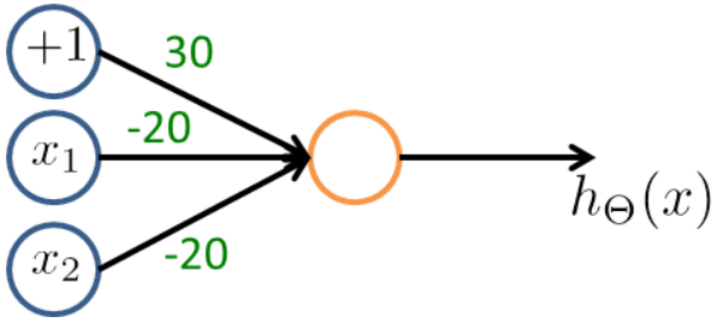
A



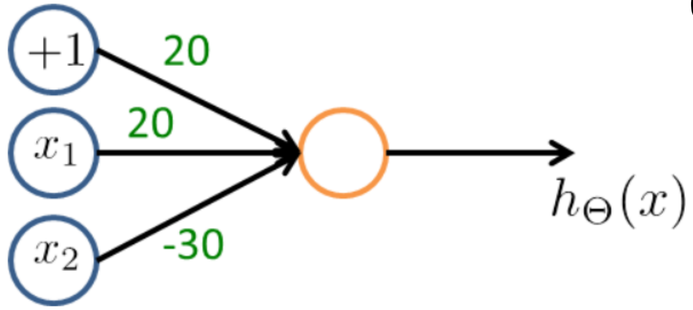
B



C

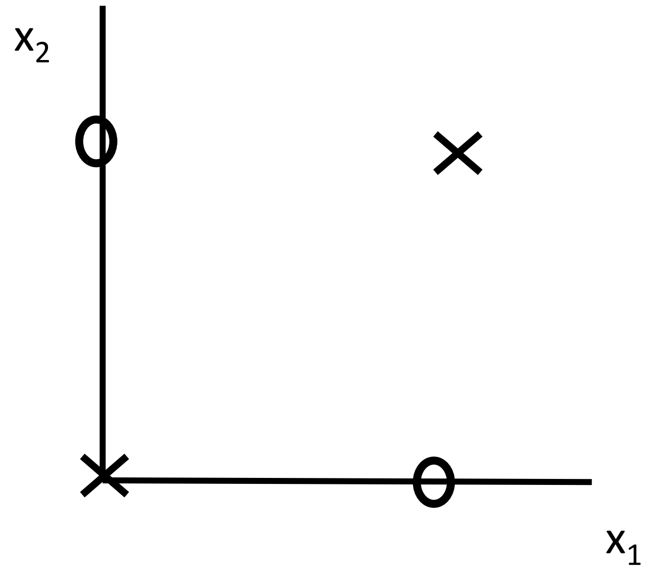


D



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

EXAMPLE: x_1 XNOR x_2

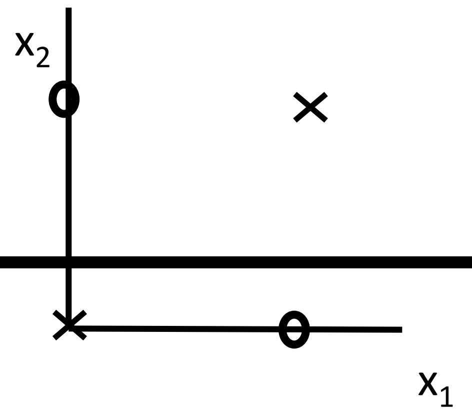


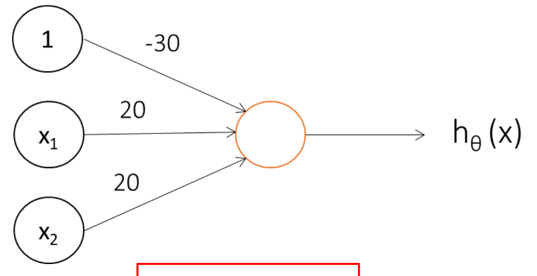
- $x_1, x_2 \in \{1,0\}$

x_1	x_2	x_1 XNOR x_2
0	0	1
0	1	0
1	0	0
1	1	1

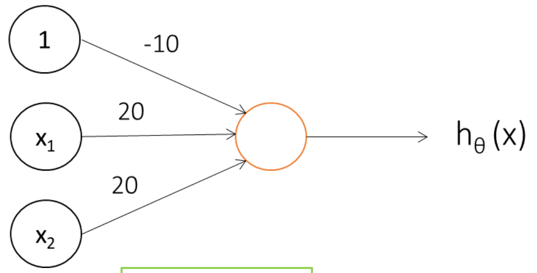
→ We need a non-linear decision boundary

Example: x_1 XNOR x_2

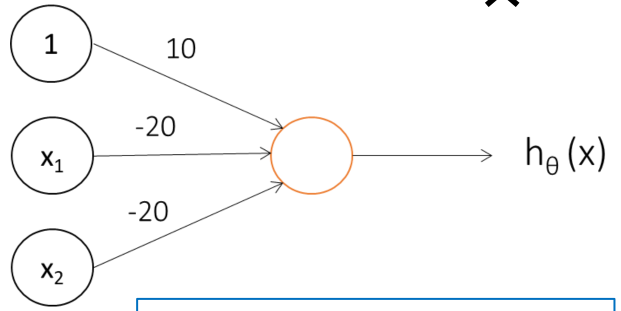




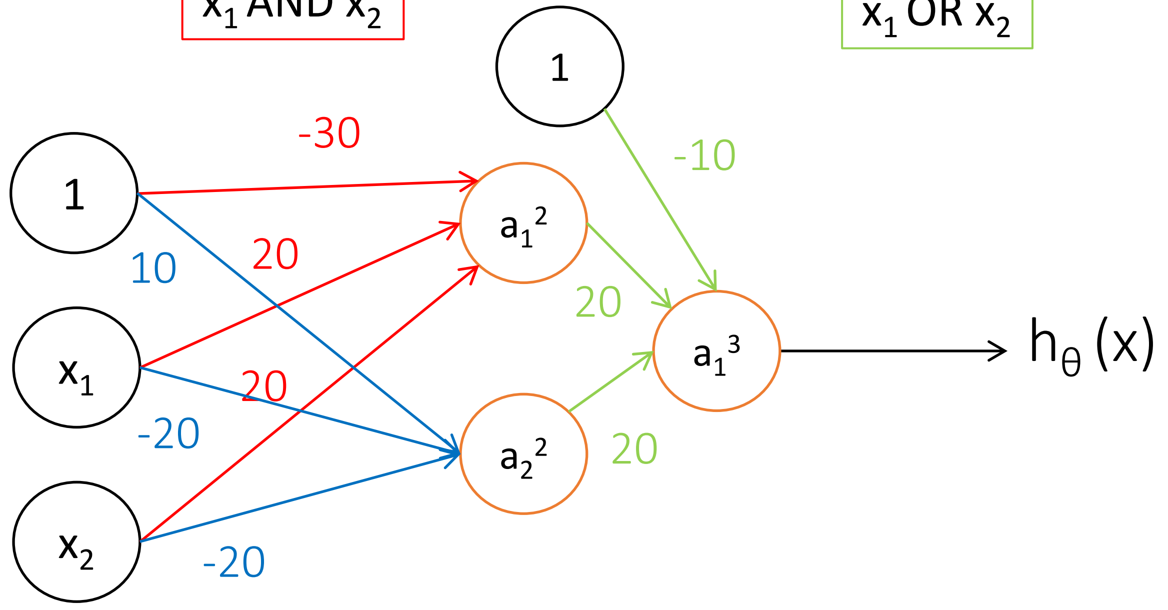
x_1 AND x_2



x_1 OR x_2



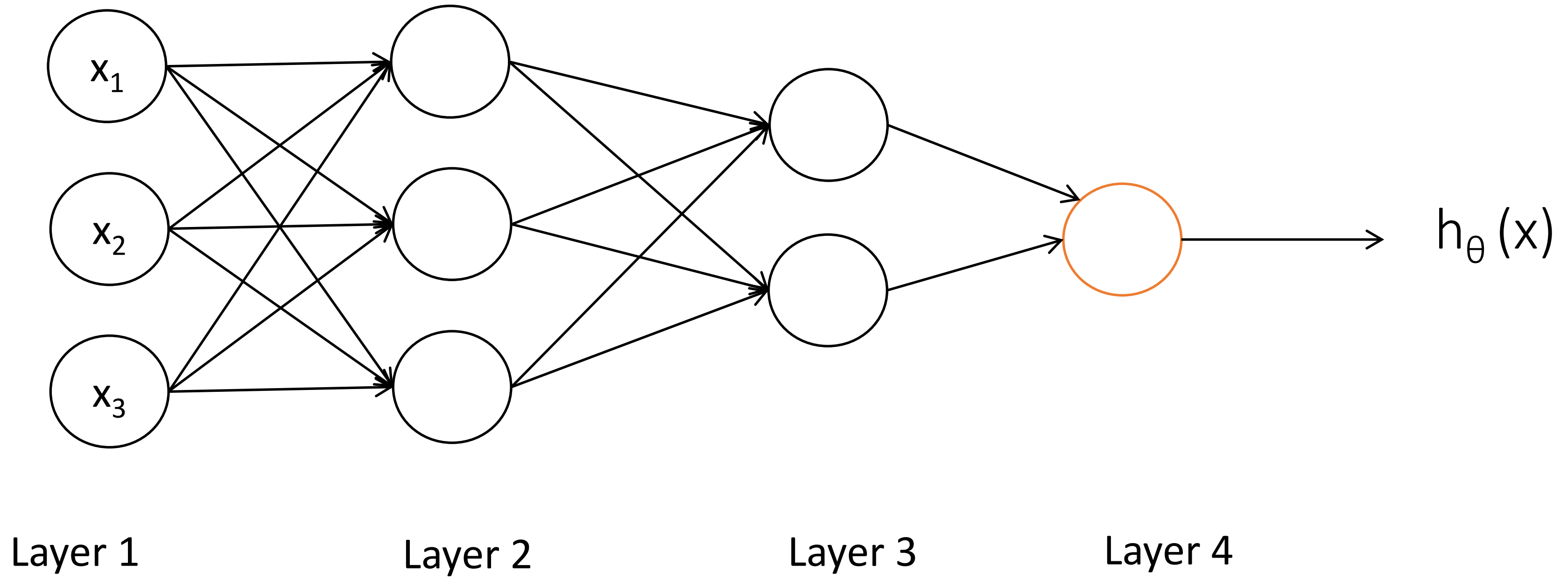
(NOT x_1) AND (NOT x_2)



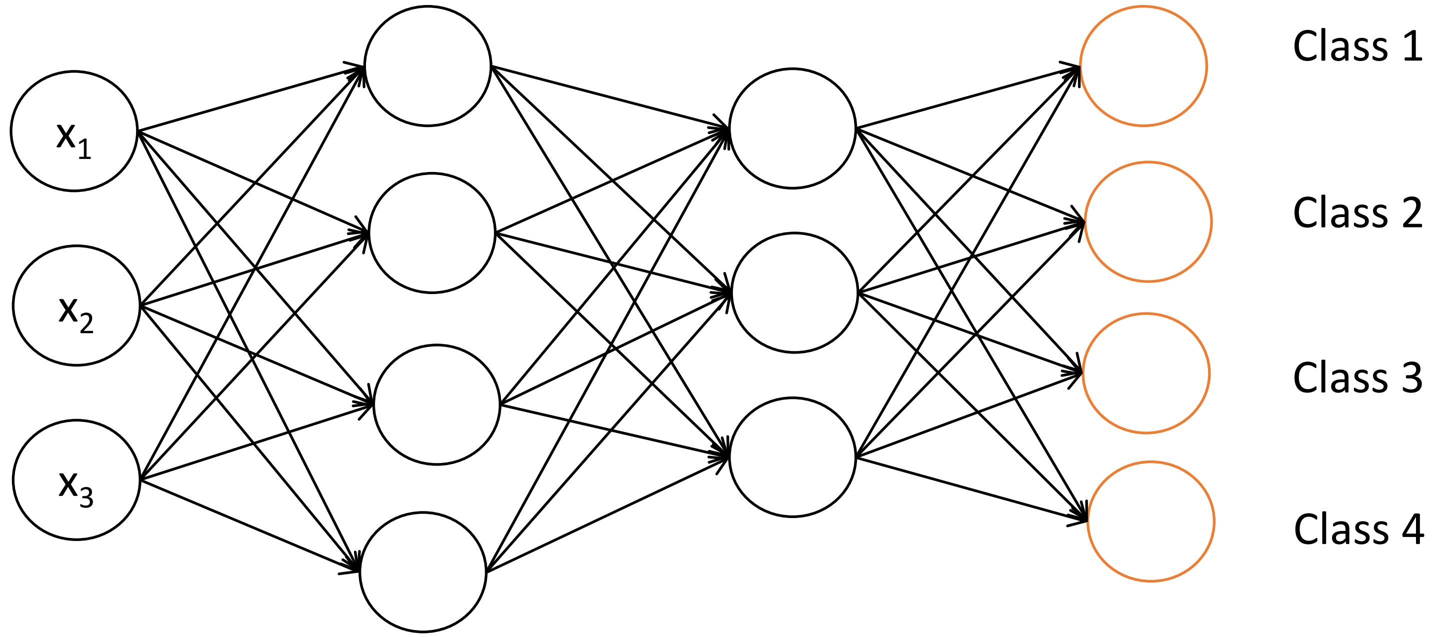
x_1	x_2	a_1^2	a_2^2	$h_\theta(x)$	x_1 XNOR x_2
0	0	0	1	1	1
0	1	0	0	0	0
1	0	0	0	0	0
1	1	1	0	1	1

NEURAL NETWORK

Each (hidden) layer can compute even more complex functions



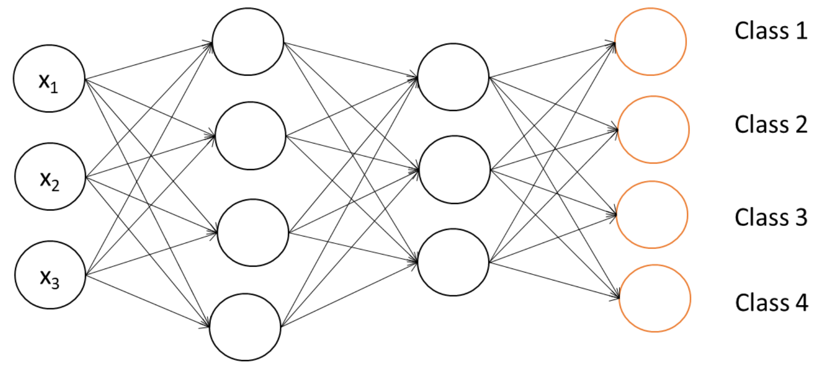
MULTI-CLASS CLASSIFICATION



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

Goal: $h_\theta(x) \sim \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ if class 1 $\quad h_\theta(x) \sim \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ if class 2, etc.

MULTIPLE OUTPUT UNITS: ONE VS- ALL



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

Goal: $h_\theta(x) \sim \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ if class 1 $\quad h_\theta(x) \sim \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ if class 2, etc.

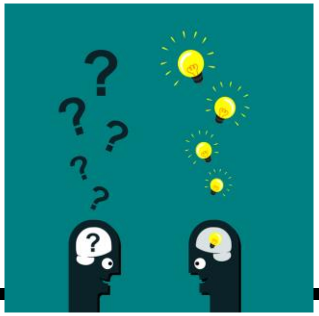
Rearrange the training set: $(x^1, y^1),(x^2, y^2),\dots,(x^m, y^m)$

y^i is one if $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

Instead of {1,2,3,4}

$h_\theta(x), y^i \in \mathbb{R}^4$

QUESTION



Suppose you have a multi-class classification problem with 10 classes. Your neural network has 3 layers, and the hidden layer (layer 2) has 5 units. Using the one-vs-all method, how many elements does $\Theta^{(2)}$ have?

A: 50

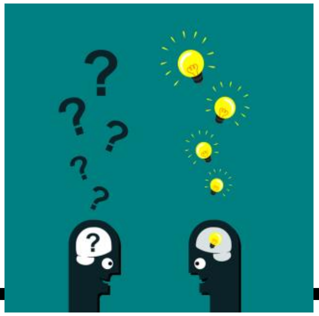
B: 55

C: 60

D: 66

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

QUESTION



Suppose you have a multi-class classification problem with 10 classes. Your neural network has 3 layers, and the hidden layer (layer 2) has 5 units. Using the one-vs-all method, how many elements does $\Theta^{(2)}$ have?

A: 50

B: 55

[✓ C: 60]

D: 66

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

Layer j = 1: ?
Layer j = 2: $s_2 = 5$
Layer j = 3: $s_3 = 10$

$\rightarrow \Theta^{(2)} = s_{j+1} \times (s_j + 1) = 10 \times (5+1) = 60$

WRAP-UP

Neural Networks Model Representation

- Let's examine how we will represent a hypothesis function using neural networks. At a very simple level, neurons are basically computational units that take inputs (**dendrites**) as electrical inputs that are channeled to outputs (**axons**). In our model, our dendrites are like the input features $x_1 \dots x_n$ and the output is the result of our hypothesis function. In this model our x_0 input node may be called the "bias unit." It is always equal to 1. In neural networks, we use the same logistic function as in classification, yet it is calles a sigmoid (logistic) **activation** function. In this situation, our "theta" parameters are called "weights".
- Our input nodes (layer 1), also known as the "input layer", go into another node (layer 2), which finally outputs the hypothesis function, known as the "output layer". We can have intermediate layers of nodes between the input and output layers called the "hidden layers."
- In this example, we label these intermediate or "hidden" layer nodes $a_0^2 \dots a_n^2$ and call them "activation units."
- a_i^j = activation of unit i in layer j
- θ^j = matrix of weights controlling function mapping from layer j to layer j +1

Wrap-Up

- The values for each of the "activation" nodes is obtained as follows:
- $a_1^2 = g(\theta_{10}^1 x_0 + \theta_{11}^1 x_1 + \theta_{12}^1 x_2 + \theta_{13}^1 x_3)$
- $a_2^2 = g(\theta_{20}^1 x_0 + \theta_{21}^1 x_1 + \theta_{22}^1 x_2 + \theta_{23}^1 x_3)$
- $a_3^2 = g(\theta_{30}^1 x_0 + \theta_{31}^1 x_1 + \theta_{32}^1 x_2 + \theta_{33}^1 x_3)$
- $h_\theta (x) = a_1^3 = g(\theta_{10}^2 a_0^2 + \theta_{11}^2 a_1^2 + \theta_{12}^2 a_2^2 + \theta_{13}^2 a_3^2)$
- This is saying that we compute our activation nodes by using a 3×4 matrix of parameters. We apply each row of the parameters to our inputs to obtain the value for one activation node. Our hypothesis output is the logistic function applied to the sum of the values of our activation nodes, which have been multiplied by yet another parameter matrix $\Theta^{(2)}$ containing the weights for our second layer of nodes.
- Each layer gets its own matrix of weights, $\Theta^{(j)}$.
- The dimensions of these matrices of weights is determined as follows:
- If the network has s_j units in layer j, s_{j+1} units in layer j+1, then Θ^j will be of dimension s_{j+1} x $(s_j + 1)$. The +1 comes from the addition in Θ^j of the "bias nodes," x_0 and $\Theta_0^{(j)}$. In other words the output nodes will not include the bias nodes while the inputs will.
- Example: If layer 1 has 2 input nodes and layer 2 has 4 activation nodes. Dimension of Θ^1 is going to be 4 × 3 where $s_j = 2$ $s_{j+1} = 4$, so $s_{j+1} \times (s_j+1) = 4 \times 3$.

WRAP-UP

- We can write:
- $a_1^2 = g(\theta_{10}^1\, x_0 + \theta_{11}^1\, x_1 + \theta_{12}^1\, x_2 + \theta_{13}^1\, x_3) = g(z_1^2)$
- $a_2^2 = g(\theta_{20}^1\, x_0 + \theta_{21}^1\, x_1 + \theta_{22}^1\, x_2 + \theta_{23}^1\, x_3) = g(z_2^2)$
- $a_3^2 = g(\theta_{30}^1\, x_0 + \theta_{31}^1\, x_1 + \theta_{32}^1\, x_2 + \theta_{33}^1\, x_3) = g(z_2^2)$
- In other words, for layer j=2 and node k, the variable z will be:
- $z_k^2 = \theta_{k,0}^1\, x_0 + \theta_{k,1}^1\, x_1 + \theta_{k,2}^1\, x_2 + \ldots + \theta_{k,3}^1\, x_n$
- Setting $x=a^{(1)}$, we can rewrite the equation as: $z^j = \theta^{j-1}a^{j-1}$
- We are multiplying our matrix $\theta^{j-1}$with dimensions $s_j \times (n+1)$ (where s_j is the number of our activation nodes) by our vector a^{j-1} with height (n+1). This gives us our vector z^j with height s_j. Now we can get a vector of our activation nodes for layer j as follows:
- $a^{(j)}=g(z^{(j)})$ Where our function g can be applied element-wise to our vector $z^{(j)}$.

Wrap-Up

- We can then add a bias unit (equal to 1) to layer j after we have computed $a^{(j)}$. This will be element $a_0^{(j)}$ and will be equal to 1. To compute our final hypothesis, let's first compute another z vector:
- $z^{(j+1)}=\Theta^{(j)}a^{(j)}$
- We get this final z vector by multiplying the next theta matrix after $\Theta^{(j-1)}$ with the values of all the activation nodes we just got. This last theta matrix $\Theta^{(j)}$will have only **one row** which is multiplied by one column $a^{(j)}$ so that our result is a single number. We then get our final result with:
- $h_\Theta(x)=a^{(j+1)}=g(z^{(j+1)})$
- Notice that in this **last step**, between layer j and layer j+1, we are doing **exactly the same thing** as we did in logistic regression. Adding all these intermediate layers in neural networks allows us to produce interesting and more complex non-linear hypotheses.

QUIZ - QUESTION 1

Which of the following statements are true? Check all that apply.

A: Any logical function over binary-valued (0 or 1) inputs x_1 and x_2 can be (approximately) represented using some neural network.

B: The activation values of the hidden units in a neural network, with the sigmoid activation function applied at every layer, are always in the range (0, 1).

C: A two layer (one input layer, one output layer; no hidden layer) neural network can represent the XOR function.

D: Suppose you have a multi-class classification problem with three classes, trained with a 3 layer network. Let $a_1^{(3)} = (h_\Theta(x))_1$ be the activation of the first output unit, and similarly $a_2^{(3)} = (h_\Theta(x))_2$ and $a_1^{(3)} = (h_\Theta(x))_3$.

Then for any input x, it must be the case that $a_1^{(3)} + a_2^{(3)} + a_3^{(3)} = 1$.

QUIZ - QUESTION 1

Which of the following statements are true? Check all that apply.

[✓ A]: Any logical function over binary-valued (0 or 1) inputs x_1 and x_2 can be (approximately) represented using some neural network.

[✓ B]: The activation values of the hidden units in a neural network, with the sigmoid activation function applied at every layer, are always in the range (0, 1).

C: A two layer (one input layer, one output layer; no hidden layer) neural network can represent the XOR function.

D: Suppose you have a multi-class classification problem with three classes, trained with a 3 layer network. Let $a_1^{(3)} = (h_\Theta(x))_1$ be the activation of the first output unit, and similarly $a_2^{(3)} = (h_\Theta(x))_2$ and $a_1^{(3)} = (h_\Theta(x))_3$.
Then for any input x, it must be the case that $a_1^{(3)} + a_2^{(3)} + a_3^{(3)} = 1$.

QUIZ - QUESTION 2

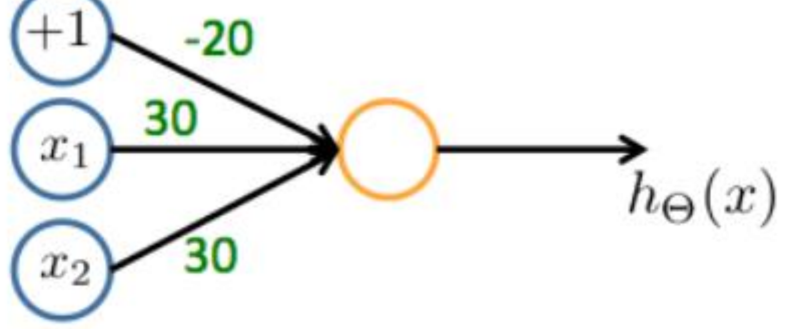
Consider the following neural network which takes two binary-valued inputs x1, x2 ∈ {0,1} and outputs $h_\Theta(x)$. Which of the following logical functions does it (approximately) compute?

A: OR

B: AND

C: NAND (meaning "NOT AND")

D: XOR (exclusive OR)



Quiz - Question 2

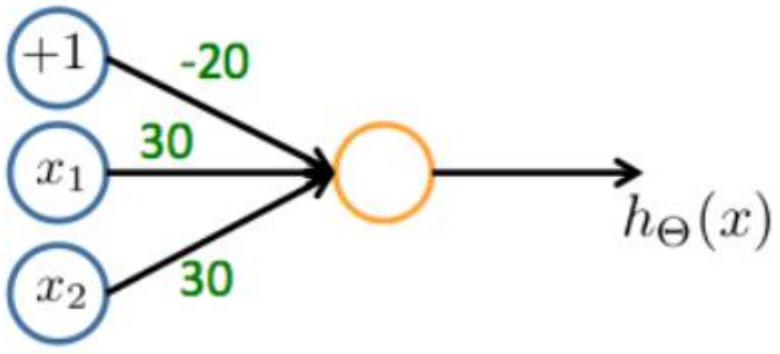
Consider the following neural network which takes two binary-valued inputs x1, x2 ∈ {0,1} and outputs $h_\Theta(x)$. Which of the following logical functions does it (approximately) compute?

✗ A: OR

B: AND

C: NAND (meaning "NOT AND")

D: XOR (exclusive OR)



x_1	x_2	$h_\theta (x)$
0	0	g(-20) ~ 0
0	1	g(10) ~ 1
1	0	g(10) ~ 1
1	1	g(60) ~ 1

QUIZ - QUESTION 3

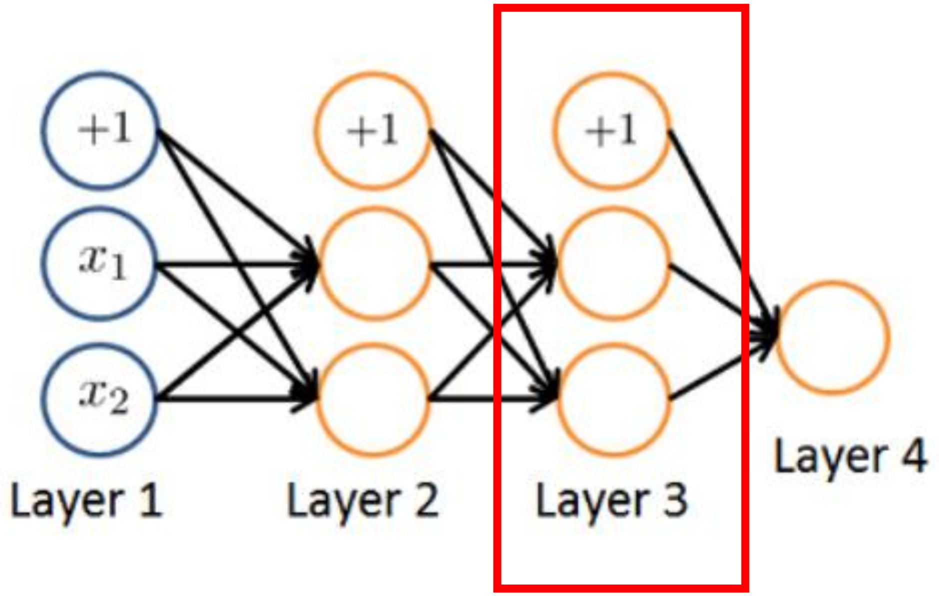
Consider the neural network given below. Which of the following equations correctly computes the activation $a_1^{(3)}$? Note: g(z) is the sigmoid activation function.

A: $a_1^{(3)} = g(\theta_{10}^{2}\, a_0^{(2)} + \theta_{11}^{2}\, a_1^{(2)} + \theta_{12}^{2}\, a_2^{(2)})$

B: $a_1^{(3)} = g(\theta_{10}^{2}\, a_0^{(1)} + \theta_{11}^{2}\, a_1^{(1)} + \theta_{12}^{2}\, a_2^{(1)})$

C: $a_1^{(3)} = g(\theta_{10}^{1}\, a_0^{(2)} + \theta_{11}^{1}\, a_1^{(2)} + \theta_{12}^{1}\, a_2^{(2)})$

D: $a_1^{(3)} = g(\theta_{10}^{1}\, a_0^{(1)} + \theta_{11}^{1}\, a_1^{(1)} + \theta_{12}^{1}\, a_2^{(1)})$



Quiz - Question 3

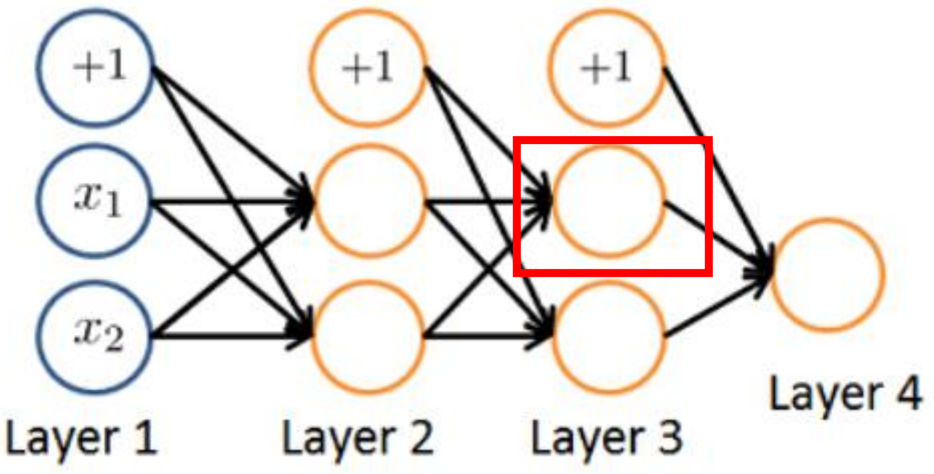
Consider the neural network given below. Which of the following equations correctly computes the activation $a_1^{(3)}$? Note: g(z) is the sigmoid activation function.

~~A~~: $a_1^{(3)} = g(\theta_{10}^{2} a_0^{(2)} + \theta_{11}^{2} a_1^{(2)} + \theta_{12}^{2} a_2^{(2)})$

B: $a_1^{(3)} = g(\theta_{10}^{2} a_0^{(1)} + \theta_{11}^{2} a_1^{(1)} + \theta_{12}^{2} a_2^{(1)})$

C: $a_1^{(3)} = g(\theta_{10}^{1} a_0^{(2)} + \theta_{11}^{1} a_1^{(2)} + \theta_{12}^{1} a_2^{(2)})$

D: $a_1^{(3)} = g(\theta_{10}^{1} a_0^{(1)} + \theta_{11}^{1} a_1^{(1)} + \theta_{12}^{1} a_2^{(1)})$

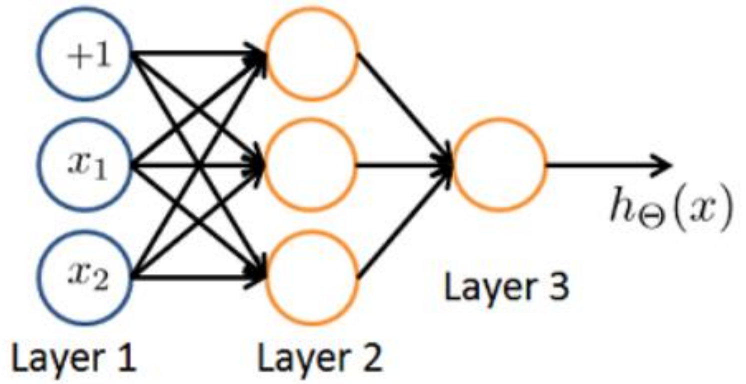


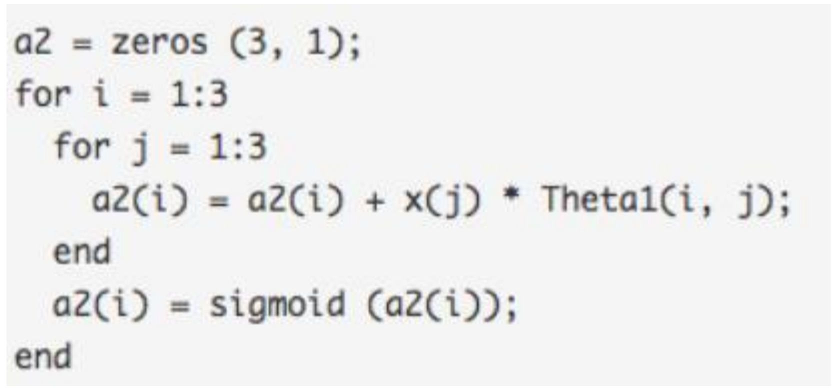
QUIZ - QUESTION 4

You have the neural network shown in the picture.

You'd like to compute the activations of the hidden layer $a^{(2)} \in R^{3.}$ One way to do so is the code shown richt.

You want to have a vectorized implementation of this (i.e., one that does not use for loops). Which of the following implementations correctly compute $a^{(2)}$? Check all that apply.





```
a2 = zeros (3, 1);
for i = 1:3
  for j = 1:3
    a2(i) = a2(i) + x(j) * Theta1(i, j);
  end
  a2(i) = sigmoid (a2(i));
end
```

A: z = Theta1 * x; a2 = sigmoid (z);

B: a2 = sigmoid (x * Theta1);

C: a2 = sigmoid (Theta2 * x);

D: z = sigmoid(x); a2 = sigmoid (Theta1 * z);

QUIZ - QUESTION 4

You have the neural network shown in the picture.

You'd like to compute the activations of the hidden layer $a^{(2)} \in R^{3.}$ One way to do so is the code shown richt.

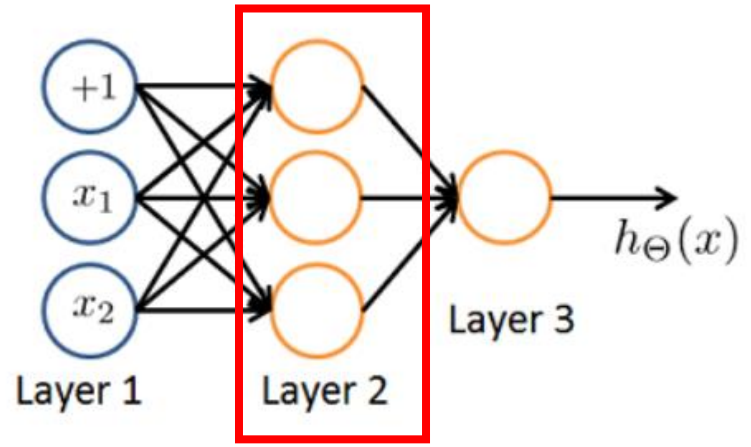
You want to have a vectorized implementation of this (i.e., one that does not use for loops). Which of the following implementations correctly compute $a^{(2)}$? Check all that apply.

$\cancel{\text{A}}$: z = Theta1 * x; a2 = sigmoid (z);

B: a2 = sigmoid (x * Theta1);

C: a2 = sigmoid (Theta2 * x);

D: z = sigmoid(x); a2 = sigmoid (Theta1 * z);



```
a2 = zeros (3, 1);
for i = 1:3
  for j = 1:3
    a2(i) = a2(i) + x(j) * Theta1(i, j);
  end
  a2(i) = sigmoid (a2(i));
end
```

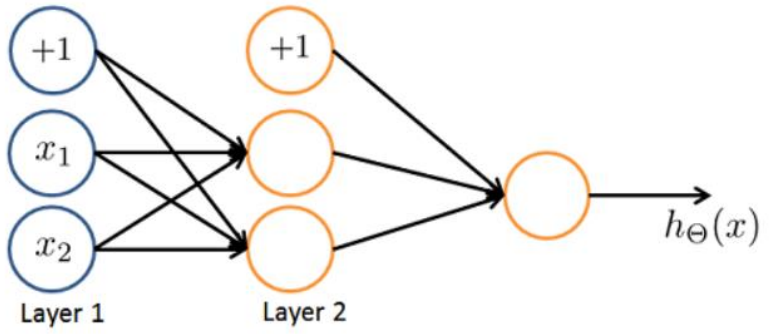
QUIZ - QUESTION 5

You are using the neural network shown in the picture and have learned the parameters

Θ^1 = [(1,0.5,1.9),(1,1.2,2.7)] (used to compute $a^{(2)}$)

and Θ^2 =[1,-0.2,-1.7] (used to compute $a^{(3)}$ as a function of $a^{(2)}$).

Suppose you swap the parameters for the first hidden layer between its two units so Θ^1 = [(1,1.2,2.7), (1,0.5,1.9)] and also swap the output layer so Θ^2 =[1,-1.7,-0.2].How will this change the value of the output $h_\Theta(x)$?



A: It will stay the same.

B: It will increase.

C: It will decrease

D: Insufficient information to tell: it may increase or decrease.

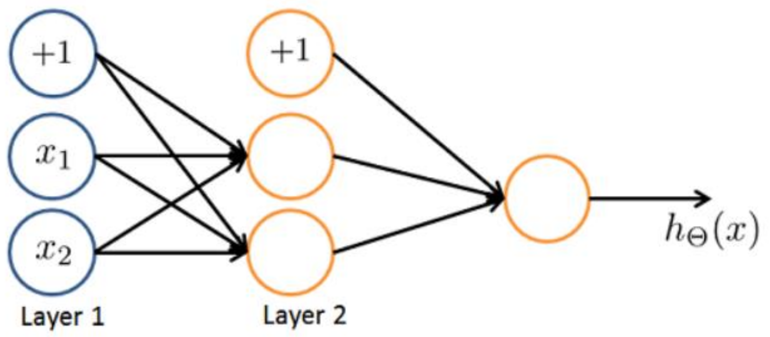
Quiz - Question 5

You are using the neural network shown in the picture and have learned the parameters

Θ^1 = [(1,0.5,1.9),(1,1.2,2.7)] (used to compute $a^{(2)}$)

and Θ^2 =[1,-0.2,-1.7] (used to compute $a^{(3)}$ as a function of $a^{(2)}$).

Suppose you swap the parameters for the first hidden layer between its two units so Θ^1 = [(1,1.2,2.7), (1,0.5,1.9)] and also swap the output layer so Θ^2 =[1,-1.7,-0.2].How will this change the value of the output $h_\Theta(x)$?



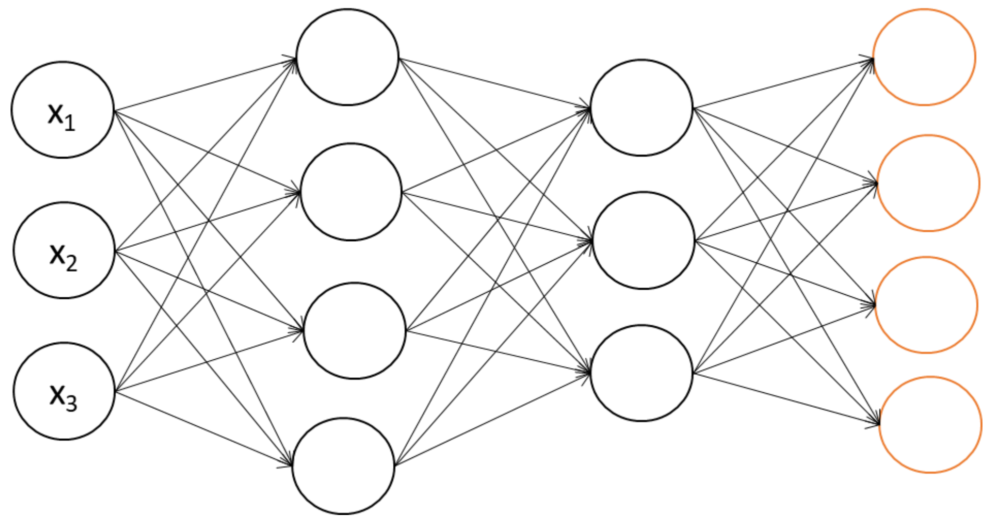
~~A:~~ It will stay the same.

B: It will increase.

C: It will decrease

D: Insufficient information to tell: it may increase or decrease.

NEURAL NETWORK



$\{(x^1, y^1),(x^2, y^2),\ldots,(x^m, y^m)\}$
L= total number of layers in network
s_l = number of units (not counting bias unit) in layer l

Binary classification
y = 0 or 1

1 output unit
S_L= 1
$h_\theta(x) \in \mathbb{R}$

Multi-class classification (k classes)
$y \in \mathbb{R}^K$

e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

k output unit
$h_\theta(x) \in \mathbb{R}^k$
S_L = K (k >=3)

COST FUNCTION

Logistic regression:

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

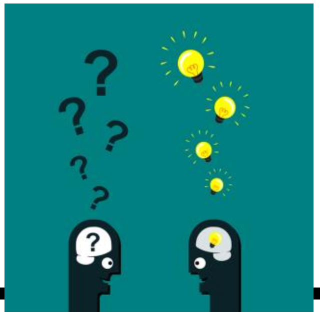
Neural network:

$h_\theta(x) \in \mathbb{R}^K$ $(h_\theta(x))_i$ = ith output

summing the cost function over each of my output units

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

QUESTION



Suppose we want to try to minimize J(Θ) as a function of Θ, using one of the advanced optimization methods (fminunc, conjugate gradient, BFGS, L-BFGS, etc.). What do we need to supply code to compute (as a function of Θ)?

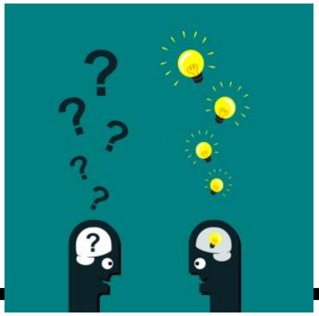
A: Θ

B: J(Θ)

C: The (partial) derivative terms $\frac{\partial}{\partial \theta_{ji}^{(l)}}$ for every i,j,l.

D: J(Θ) and the (partial) derivative terms $\frac{\partial}{\partial \theta_{ji}^{(l)}}$ for every i,j,l.

QUESTION



Suppose we want to try to minimize J(Θ) as a function of Θ, using one of the advanced optimization methods (fminunc, conjugate gradient, BFGS, L-BFGS, etc.). What do we need to supply code to compute (as a function of Θ)?

A: Θ

B: J(Θ)

C: The (partial) derivative terms $\frac{\partial}{\partial \theta_{ji}^{(l)}}$ for every i,j,l.

~~D~~: J(Θ) and the (partial) derivative terms $\frac{\partial}{\partial \theta_{ji}^{(l)}}$ for every i,j,l.

GRADIENT COMPUTATION

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

- $\text{Min}_\theta\ J(\theta)$
- Need code to compute
- $J(\theta)$
- $\frac{\partial}{\partial \theta_{ji}{}^{(l)}} J(\theta) \rightarrow \theta_{ji}{}^{(l)} \in \mathbb{R}$

Gradient computation

- Given one training example (x,y):
- Forward propagation to compute output:

$a^{(1)} = x$

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

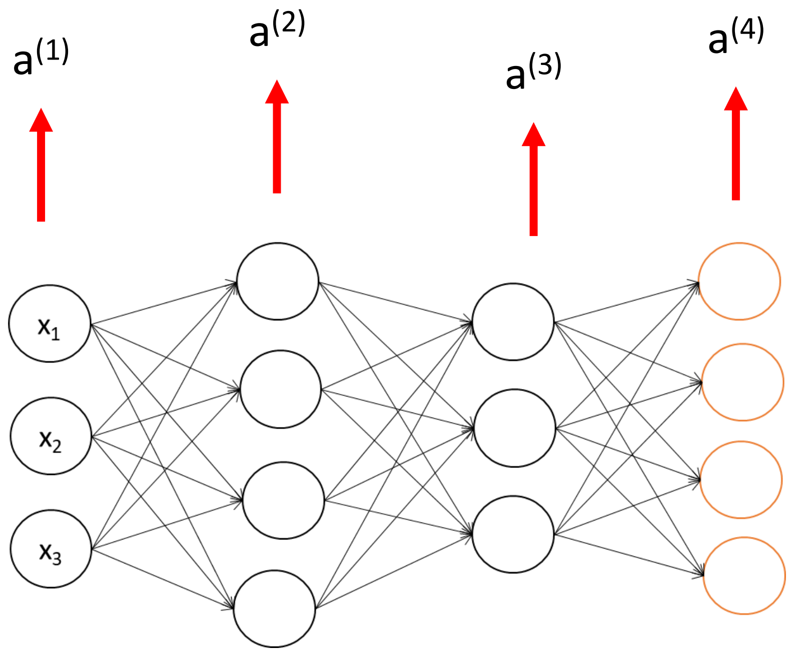
$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

$z^{(3)} = \theta^{(2)} a^{(2)}$

$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)

$z^{(4)} = \theta^{(3)} a^{(3)}$

$a^{(4)} = h_\theta(x) = g(z^{(4)})$



GRADIENT COMPUTATION: BACKPROPAGATION ALGORITHM

Intuition: δ_j^l = "error" of node j in layer l

Example: For each output unit (L = 4)

$\delta_j^4 = a_j^4 - yj = h_\theta(x)_j - yj$

→Vectorized: $\delta^4 = a^4 - y$

→Dimension is equal to no. output units in network

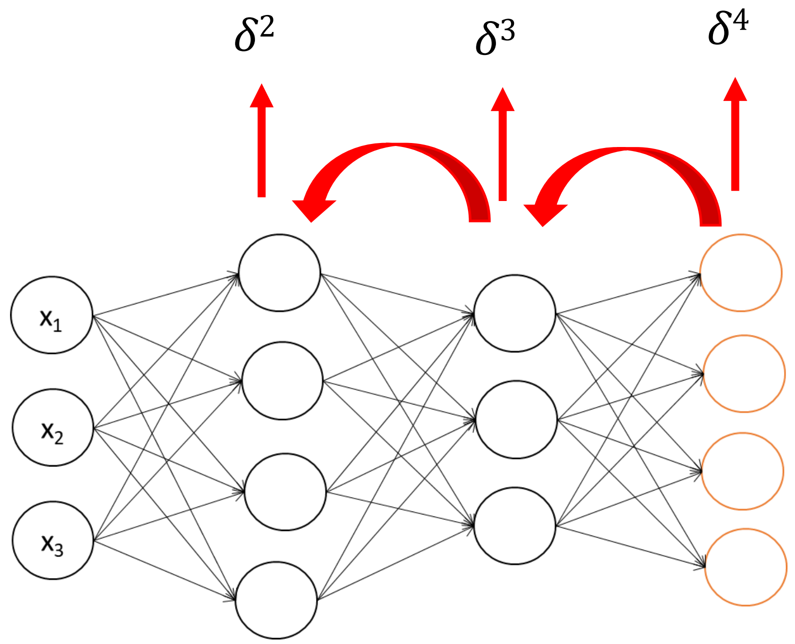
$\delta_j^3 = \theta^{3T} \delta^4 .* g'(z^3)$ → 2 vectors; $g'(z^3) = a^3 .* (1-a^3)$

$\delta_j^2 = \theta^{2T} \delta^3 .* g'(z^2)$ → $g'(z^2) = a^2 .* (1-a^2)$

→No δ^1 term → input layer → no error

→$\boxed{\frac{\partial}{\partial \theta_{ji}^{(l)}} J(\theta) = a_j^l \delta_i^{l+1}}$ ignoring λ , i.e. regularization

Backpropagation



BACKPROPAGATION ALGORITHM

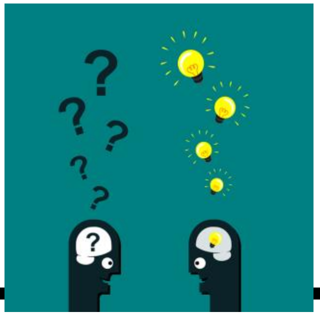
- Training set $\{(x^1, y^1),(x^2, y^2),\ldots,(x^m, y^m)\}$

```
Set $\Delta_{ji}^{l} = 0$ (for all i,j,l)  → used to compute $\frac{\partial}{\partial \theta_{ji}^{(l)}} J(\theta)$

For i = 1 to m
        Set $a^1 = x^1$
        Perform forward propagation to compute $a^L$ for l= 2,3,..L
        using $y^i$, compute $\delta^L = a^L - y^i$
        compute $\delta^{L-1}, \delta^{L-2}, \ldots, \delta^2$
        $\Delta_{ji}^{l} := \Delta_{ji}^{l} + a_j^l \delta_i^{l+1}$
$D_{ji}^{l} := \frac{1}{m} \Delta_{ji}^{l} + \lambda \theta_{ji}^{(l)}$   if $j \neq 0$
$D_{ji}^{l} := \frac{1}{m} \Delta_{ji}^{l}$   if $j = 0$
```

$$\frac{\partial}{\partial \theta_{ji}^{(l)}} J(\theta) = D_{ji}^{l}$$

QUESTION



Suppose you have two training examples (x^1, y^1) and (x^2, y^2). Which of the following is a correct sequence of operations for computing the gradient? (Below, FP = forward propagation, BP = back propagation).

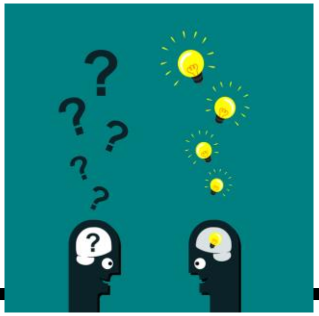
A: FP using x^1 followed by FP using x^2. Then BP using y^1 followed by BP using y^2.

B: FP using x^1 followed by BP using y^2. Then FP using x^2 followed by BP using y^1.

C: BP using y^1 followed by FP using x^1. Then BP using y^2 followed by FP using x^2.

D: FP using x^1 followed by BP using y^1. Then FP using x^2 followed by BP using y^2.

QUESTION



Suppose you have two training examples (x^1, y^1) and (x^2, y^2). Which of the following is a correct sequence of operations for computing the gradient? (Below, FP = forward propagation, BP = back propagation).

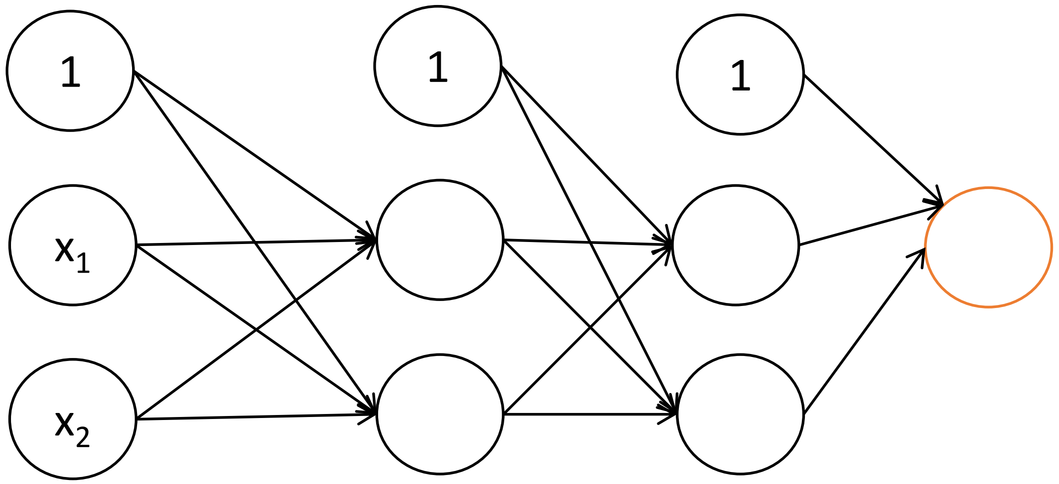
A: FP using x^1 followed by FP using x^2. Then BP using y^1 followed by BP using y^2.

B: FP using x^1 followed by BP using y^2. Then FP using x^2 followed by BP using y^1.

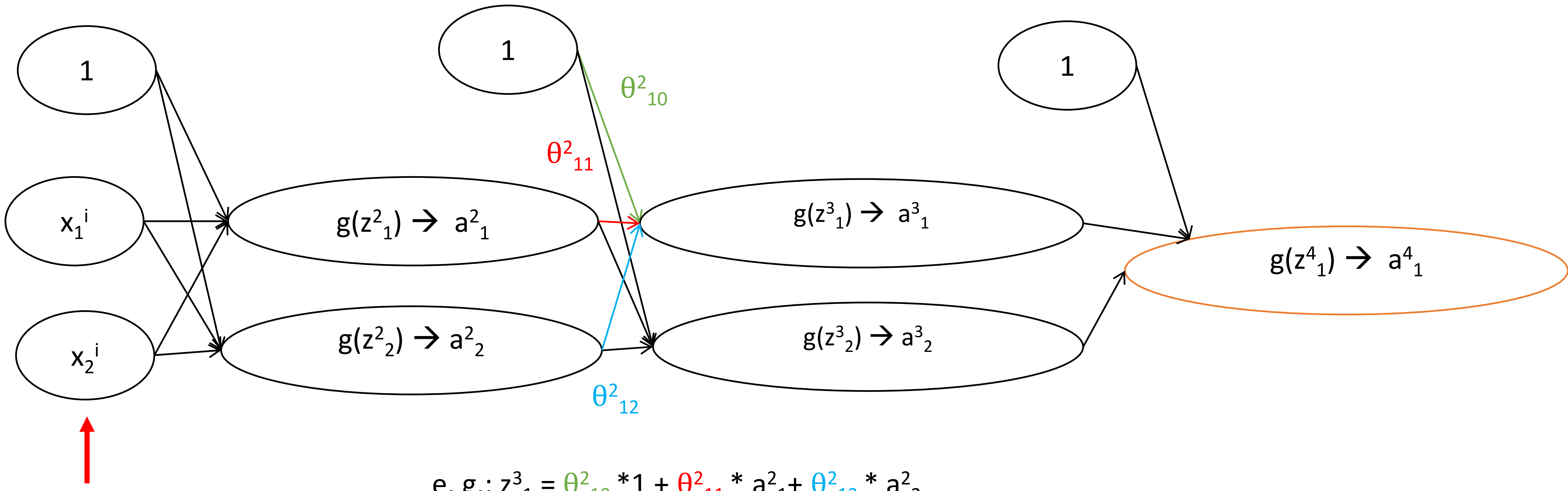
C: BP using y^1 followed by FP using x^1. Then BP using y^2 followed by FP using x^2.

[✓ D]: FP using x^1 followed by BP using y^1. Then FP using x^2 followed by BP using y^2.

EXAMPLE: FORWARD PROPAGATION



EXAMPLE: FORWARD PROPAGATION



1
x_1^i
x_2^i
$g(z^2_1) \rightarrow a^2_1$
$g(z^2_2) \rightarrow a^2_2$
1
θ^2_{10}
θ^2_{11}
θ^2_{12}
$g(z^3_1) \rightarrow a^3_1$
$g(z^3_2) \rightarrow a^3_2$
1
$g(z^4_1) \rightarrow a^4_1$

Examples (x^i, y^i)

e. g.: $z^3_1 = \theta^2_{10} * 1 + \theta^2_{11} * a^2_1 + \theta^2_{12} * a^2_2$

What is Backpropagation Doing?

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

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

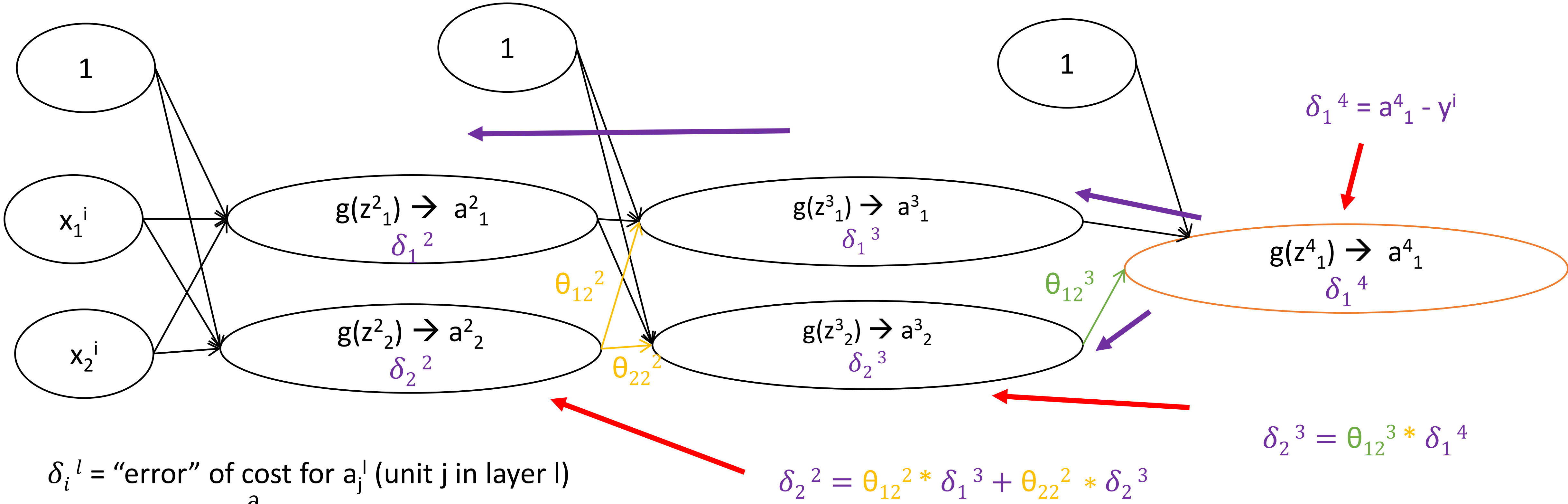
Focusing on a single example x^i, y^i, the case of 1 output unit and ignoring regularization (λ = 0),

cost(i) = $y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))$

(hint: think of cost ~ $(h_\theta(x^{(i)}) - y^{(i)})^2$ if this is easier to understand)

i. e. how well is the network doing on example i?

Example: Backpropagation

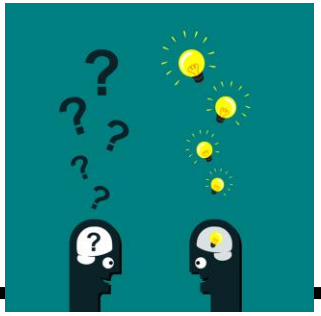

1
x_1^i
x_2^i
1
$g(z^2_1) \rightarrow a^2_1$
$\delta_1{}^2$
$g(z^2_2) \rightarrow a^2_2$
$\delta_2{}^2$
1
$g(z^3_1) \rightarrow a^3_1$
$\delta_1{}^3$
$g(z^3_2) \rightarrow a^3_2$
$\delta_2{}^3$
$\theta_{12}{}^2$
$\theta_{22}{}^2$
$\theta_{12}{}^3$
$g(z^4_1) \rightarrow a^4_1$
$\delta_1{}^4$
$\delta_1{}^4 = a^4_1 - y^i$
$\delta_2{}^3 = \theta_{12}{}^3 * \delta_1{}^4$
$\delta_2{}^2 = \theta_{12}{}^2 * \delta_1{}^3 + \theta_{22}{}^2 * \delta_2{}^3$

$\delta_i{}^l$ = "error" of cost for a_j^l (unit j in layer l)

Formally, $\delta_i{}^l = \frac{\partial}{\partial z_j{}^l}$cost(i) (for j>=0), where

Cost(i) = $y^{(i)}\log(h_\theta(x^{(i)}))+ (1-y^{(i)}{}_k)\log(1-h_\theta(x^{(i)}))$

QUESTION



Consider the given neural network:

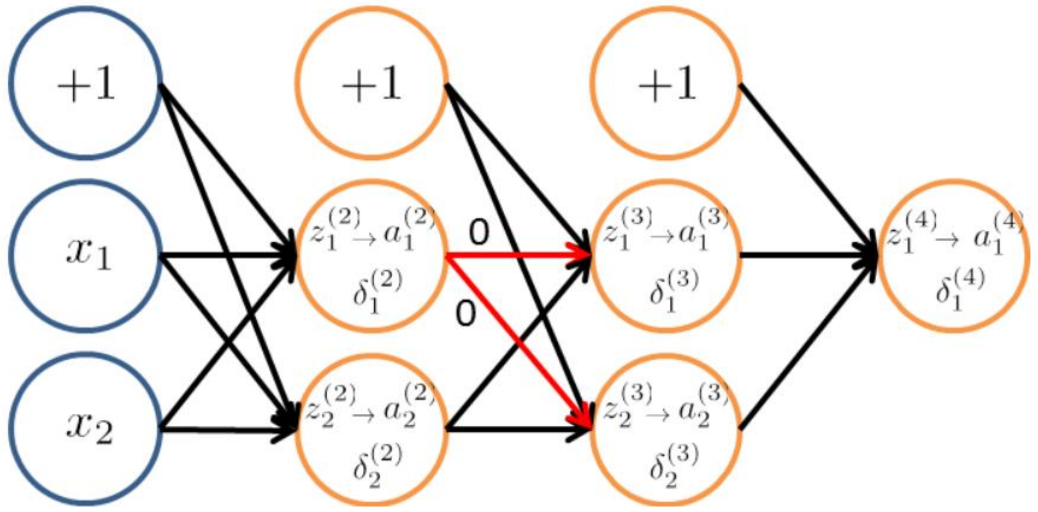
Suppose both of the weights shown in red ($\theta_{11}^{2}$and θ_{21}^{2}) are equal to 0. After running backpropagation, what can we say about the value of $\delta_1{}^{3}$?

A: $\delta_1{}^{3} > 0$

B: $\delta_1{}^{3}$= 0, only if $\delta_1{}^{2} = \delta_2{}^{2} = 0$, but not necessarily otherwise

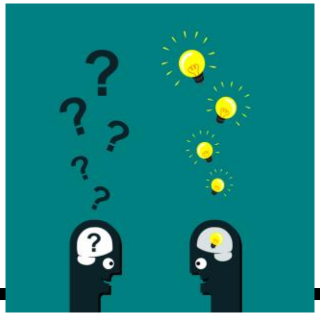
C: $\delta_1{}^{3} <= 0$ regardless the values of $\delta_1{}^{2}$ and $\delta_2{}^{2}$

B: There is insufficient information to tell



+1
+1
+1
x_1
x_2
$z_1^{(2)} \to a_1^{(2)}$
$\delta_1^{(2)}$
$z_2^{(2)} \to a_2^{(2)}$
$\delta_2^{(2)}$
0
0
$z_1^{(3)} \to a_1^{(3)}$
$\delta_1^{(3)}$
$z_2^{(3)} \to a_2^{(3)}$
$\delta_2^{(3)}$
$z_1^{(4)} \to a_1^{(4)}$
$\delta_1^{(4)}$

QUESTION



Consider the given neural network:

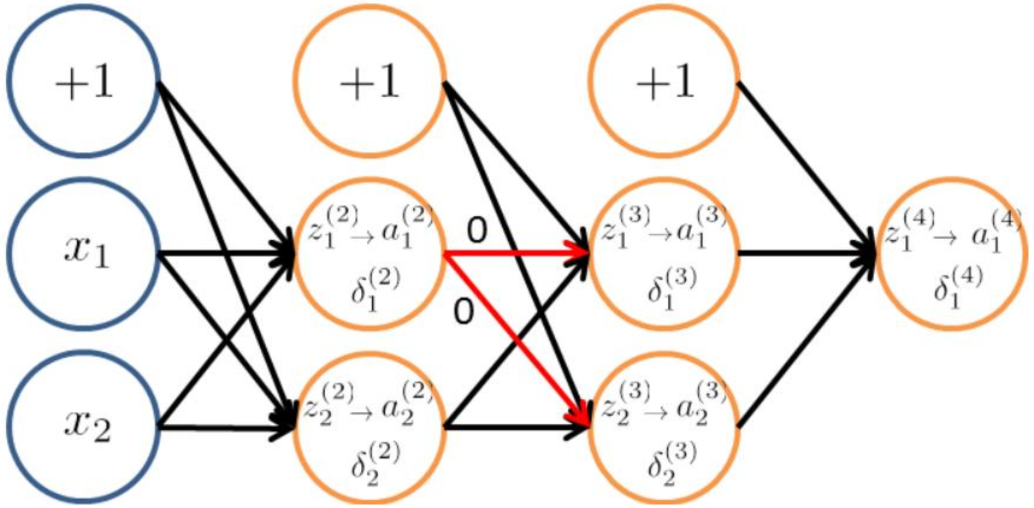
Suppose both of the weights shown in red ($\theta_{11}{}^2$and $\theta_{21}{}^2$) are equal to 0. After running backpropagation, what can we say about the value of $\delta_1{}^3$?

A: $\delta_1{}^3 > 0$

B: $\delta_1{}^3 = 0$, only if $\delta_1{}^2 = \delta_2{}^2 = 0$, but not necessarily otherwise

C: $\delta_1{}^3 <= 0$ regardless the values of $\delta_1{}^2$ and $\delta_2{}^2$

~~D~~: There is insufficient information to tell



$+1$
$+1$
$+1$
x_1
x_2
$z_1^{(2)} \to a_1^{(2)}$
$\delta_1^{(2)}$
$z_2^{(2)} \to a_2^{(2)}$
$\delta_2^{(2)}$
0
0
$z_1^{(3)} \to a_1^{(3)}$
$\delta_1^{(3)}$
$z_2^{(3)} \to a_2^{(3)}$
$\delta_2^{(3)}$
$z_1^{(4)} \to a_1^{(4)}$
$\delta_1^{(4)}$

ADVANCED OPTIMIZATION

```
function[jVal, gradient] = costFunction(theta)
```

$\in \mathbb{R}^{n+1} \rightarrow$ vector

$\in \mathbb{R}^{n+1} \rightarrow$ vectors

```
optTheta = fminunc(@costfucntion, initialTheta,
options)
```

Neural Network (L =4)

$\Theta^1, \Theta^2, \Theta^3$ – matrices (`Theta1, Theta2, Theta3`)

D^1, D^2, D^3 – matrices (`D1, D2, D3`)

"Unroll" into vectors

Example

s_1 = 10 (input layer) ; s_2 = 10 (hidden layer); s_3 = 1 (output layer)

$\Theta^1 \in \mathbb{R}^{10x11}$, $\Theta^2 \in \mathbb{R}^{10x11}$, $\Theta^3 \in \mathbb{R}^{1x11}$
$D^1 \in \mathbb{R}^{10x11}$, $D^2 \in \mathbb{R}^{10x11}$, $D^3 \in \mathbb{R}^{1x11}$

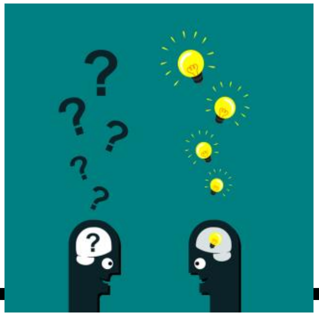
Octave:

```
thetaVec = [Theta1(:); Theta2(:); Theta3(:)];
Dvec = [D1(:); D2(:); D3(:)];
```

→unrolls the matrices and puts all the elements into a big vector)

```
Theta1 = reshape (thetaVec(1:110), 10, 11);
Theta2 = reshape (thetaVec(111:220), 10, 11);
Theta3 = reshape (thetaVec(221:230), 1, 11);
```

QUESTION



Suppose D1 is a 10x6 matrix and D2 is a 1x11 matrix. You set:

```
DVec = [D1(:); D2(:)];
```

Which of the following would get D2 back from DVec?

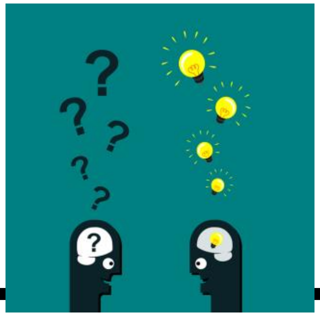
A: reshape(DVec(60:71), 1, 11)

B: reshape(DVec(61:72), 1, 11)

C: reshape(DVec(61:71), 1, 11)

D: reshape(DVec(60:70), 11, 1)

QUESTION



Suppose D1 is a 10x6 matrix and D2 is a 1x11 matrix. You set:

DVec = [D1(:); D2(:)];

Which of the following would get D2 back from DVec?

A: reshape(DVec(60:71), 1, 11)

B: reshape(DVec(61:72), 1, 11)

~~C~~: reshape(DVec(61:71), 1, 11)

D: reshape(DVec(60:70), 11, 1)

Process – Learning Algorithm

Have initial parameters $\Theta^1, \Theta^2, \Theta^3$

Unroll to get `initialTheta` to pass to

```
fminunc(@costfunction, initialTheta, options)
```

```
function[jVal, gradientVec] = costFunction(thetaVec)
```

From `thetaVec`, get $\Theta^1, \Theta^2, \Theta^3 \rightarrow$ reshape

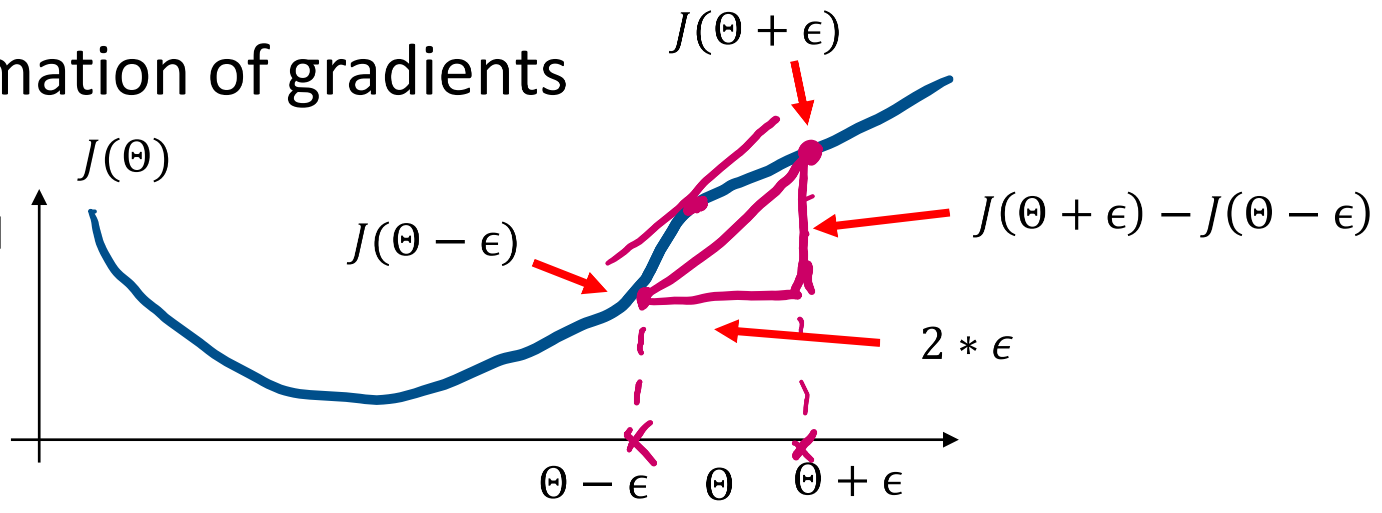
Use forward prop/back prop to compute D^1, D^2, D^3 and $J(\Theta)$

Unroll D^1, D^2, D^3 to get `gradientVec`.

Gradient Checking

Numerical estimation of gradients

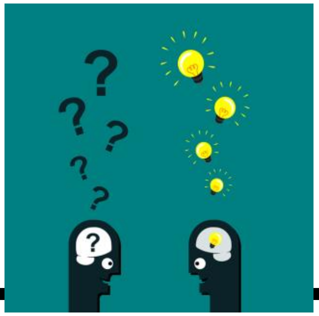
Example: $\Theta \in \mathbb{R} \rightarrow$ real number



$$\frac{d}{d\Theta} J(\Theta) \sim \frac{J(\Theta+\epsilon) - J(\Theta-\epsilon)}{2 * \epsilon} \qquad \rightarrow \epsilon \sim 10^{-4}$$

Implement: `gradApprox = (J(theta + EPSILON)- J(theta  - EPSILON))/(2*EPSILON)`

QUESTION



Let J(θ) = θ^3. Furthermore, let θ = 1 and ϵ = 0.01. You use the formula:

$$\frac{J(\Theta + \epsilon) - J(\Theta - \epsilon)}{2 * \epsilon}$$

to approximate the derivative. What value do you get using this approximation? (When θ = 1 the true, exact derivative $\frac{d}{d\Theta}J(\Theta)$ is 3).

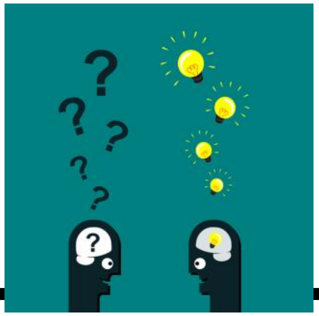
A: 3.0000

B: 3.0001

C: 3.0301

D: 6.0002

QUESTION



Let J(θ) = θ^3. Furthermore, let θ = 1 and ϵ = 0.01. You use the formula:

$$\frac{J(\Theta+\epsilon)-J(\Theta-\epsilon)}{2*\epsilon}$$

to approximate the derivative. What value do you get using this approximation? (Hint: For θ = 1 the true, exact derivative $\frac{d}{d\Theta}J(\Theta)$ is 3).

A: 3.0000

~~B~~: 3.0001

C: 3.0301

D: 6.0002

$$\frac{(1+0.01)^3-(1-0.01)^3}{2*0.01}$$

PARAMETER VECTOR Θ

- $\Theta \in \mathbb{R}^n$ → (e.g. Θ is "unrolled" version of Θ^1, Θ^2, Θ^3)
- $\Theta = \Theta_1, \Theta_2, \Theta_3, \ldots, \Theta_n$, i.e. a vector
- Same idea:
- $\frac{\partial}{\partial \theta_1} J(\Theta) \sim \frac{J(\Theta_1+\epsilon, \Theta_2, \ldots, \Theta_n) - J(\Theta_1-\epsilon, \Theta_2, \ldots, \Theta_n)}{2*\epsilon}$
- $\frac{\partial}{\partial \theta_2} J(\Theta) \sim \frac{J(\Theta_1, \Theta_2+\epsilon, \ldots, \Theta_n) - J(\Theta_1, \Theta_2-\epsilon, \ldots, \Theta_n)}{2*\epsilon}$
- …
- $\frac{\partial}{\partial \theta_n} J(\Theta) \sim \frac{J(\Theta_1, \Theta_2, \ldots, \Theta_n+\epsilon) - J(\Theta_1, \Theta_2, \ldots, \Theta_n-\epsilon)}{2*\epsilon}$

EXAMPLE

```
epsilon = 10e-4;
for i = 1:n,
  thetaPlus = theta;
  thetaPlus(i) += epsilon;
  thetaMinus = theta;
  thetaMinus(i) -= epsilon;
  gradApprox(i) = (J(thetaPlus) -
J(thetaMinus))/(2*epsilon)
End;
```

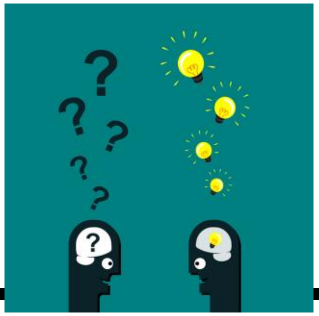
Check that `gradApprox ~ DVec`

IMPLEMENTATION ADVICE

- Implement backprop to compute `DVec` (unrolled D^1, D^2, D^3).
- Implement numerical gradient check to compute `gradApprox`.
- Make sure they give similar values.
- Turn off gradient checking. Using backprop code for learning `Dvec`, δ^i

- Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of `costFunction()`) your code will be very slow.

Question



What is the main reason that we use the backpropagation algorithm rather than the numerical gradient computation method during learning?

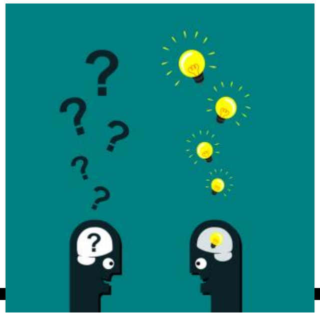
A: The numerical gradient computation method is much harder to implement.

B: The numerical gradient algorithm is very slow.

C: Backpropagation does not require setting the parameter EPSILON.

D: None of the above.

QUESTION



What is the main reason that we use the backpropagation algorithm rather than the numerical gradient computation method during learning?

A: The numerical gradient computation method is much harder to implement.

[✓ B]: The numerical gradient algorithm is very slow.

C: Backpropagation does not require setting the parameter EPSILON.

D: None of the above.

Initial Value of Θ

For gradient descent and the advanced optimization method, need initial value for Θ.

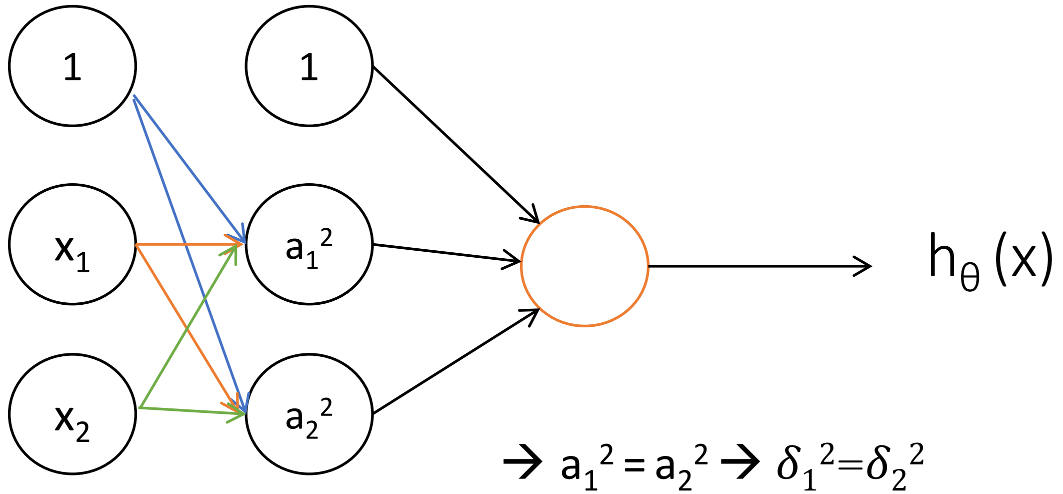
```
optTheta = fminunc(@costFunction, initialTheta,
options)
```

Consider gradient descent

Set `initialTheta = zeros(n,1)?`

→ Does not work for neural networks

ZERO INITIALIZATION



$\theta_{ji}^{(l)} = 0$ for all i,j,l

→ $a_1^2 = a_2^2$ → $\delta_1^2 = \delta_2^2$

→ $\frac{\partial}{\partial \theta_{01}^{1}} J(\Theta) = \frac{\partial}{\partial \theta_{02}^{1}} J(\Theta)$ → After an gradient descent update: $\theta_{01}^{1} = \theta_{02}^{1}$

After each update, parameters corresponding to inputs going into each of two hidden units are identical
→ After updating still: $a_1^2 = a_2^2$ → all hidden units are computing the exact same feature.

RANDOM INITIALIZATION: SYMMETRY BREACKING

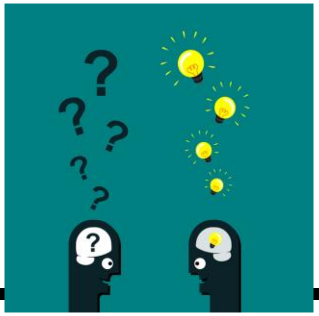
Initialize each $\theta_{ji}^{(l)}$ to a random value [- ϵ, ϵ]

→- ϵ <= $\theta_{ji}^{(l)}$ <= ϵ

E. g.:

```
Theta1 = rand(10,11) * (2 * INIT_EPSILON) -
INIT_EPSILON;
```

QUESTION



Consider this procedure for initializing the parameters of a neural network:

Pick a random number r = rand(1,1) * (2 * INIT_EPSILON) - INIT_EPSILON;
Set $\theta_{ji}^{(l)}$ = r for all i,j,l.

Does this work?

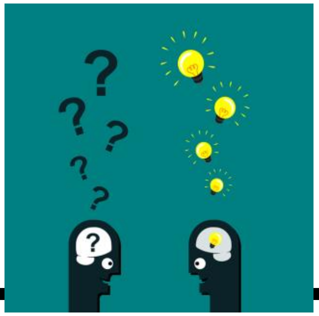
A: Yes, because the parameters are chosen randomly.

B: Yes, unless we are unlucky and get r = 0 (up to numerical precision).

C: Maybe, depending on the training set inputs $x^{(i)}$.

D: No, because this fails to break symmetry.

QUESTION



Consider this procedure for initializing the parameters of a neural network:

Pick a random number r = rand(1,1) * (2 * INIT_EPSILON) - INIT_EPSILON;
Set $\theta_{ji}^{(l)}$ = r for all i,j,l.

Does this work?

A: Yes, because the parameters are chosen randomly.

B: Yes, unless we are unlucky and get r = 0 (up to numerical precision).

C: Maybe, depending on the training set inputs $x^{(i)}$.

[✓ D: No, because this fails to break symmetry.]

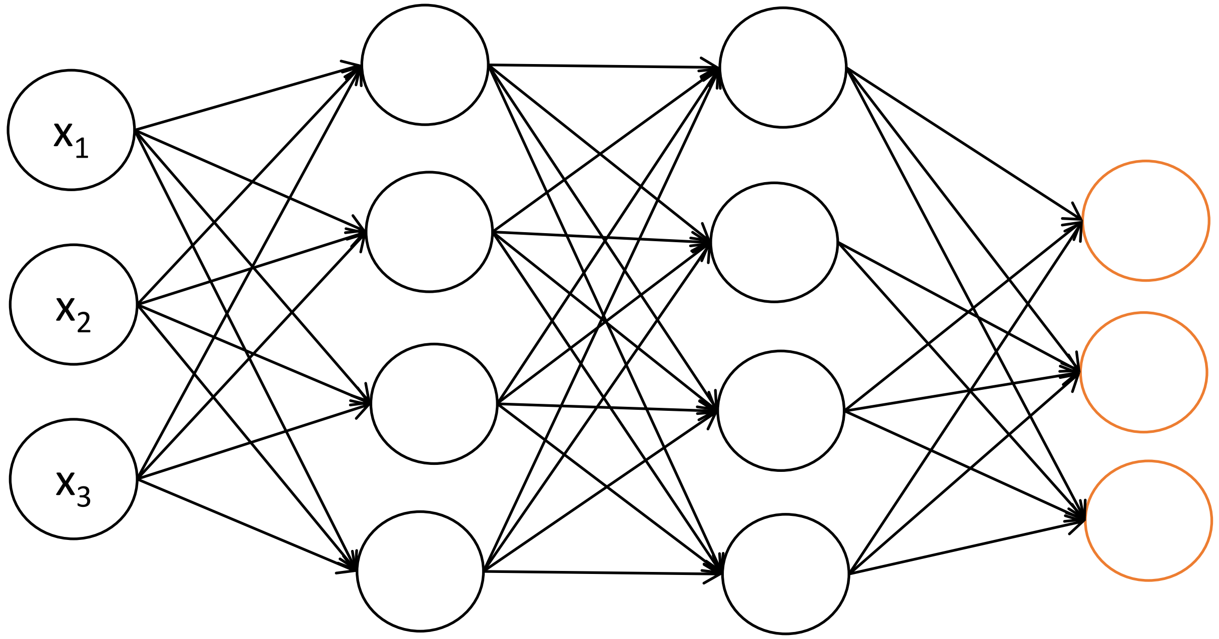
Training a neural Network

Pick a network architecture

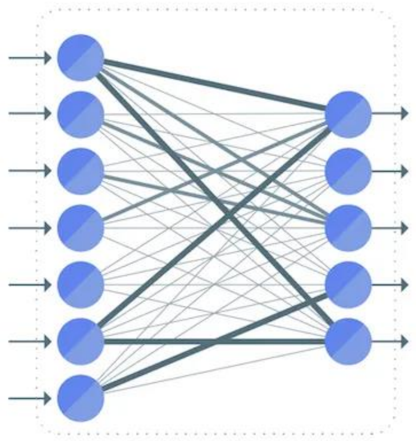
→ number of input units: Dimensions of x^i

→ number of output units: Number of classes

→ Reasonable default: 1 hidden layer, or if > 1 hidden layer, have the same umber of hidden units in every layer (usually the more the better)

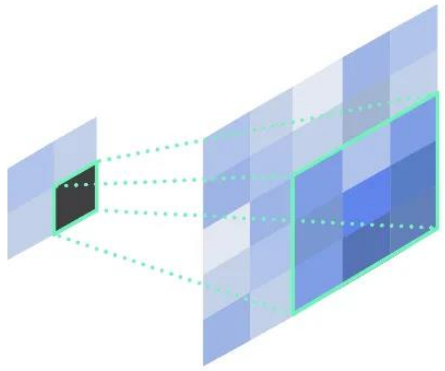


Layer Types



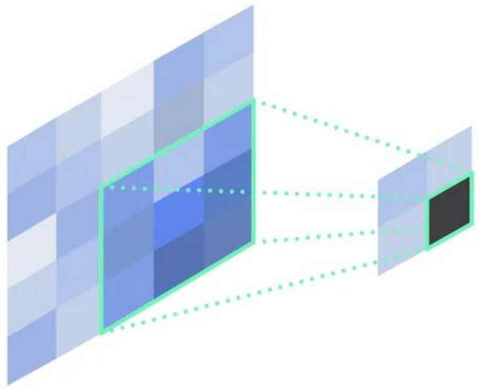
Fully connected layer

every neuron in one layer is connected to every neuron in the next layer



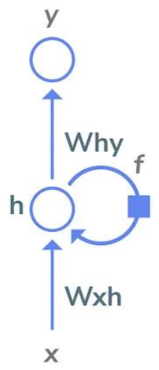
Deconvolution Layer

Upsampling data



Convolutional Layer

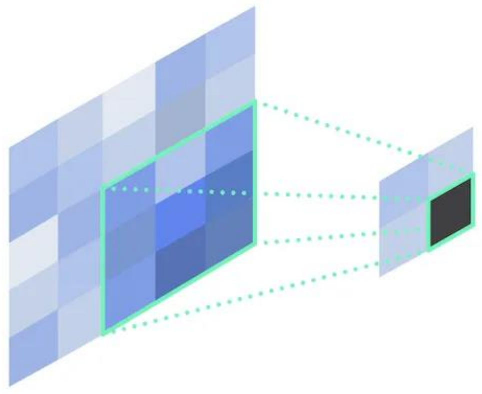
"Filtering" the input

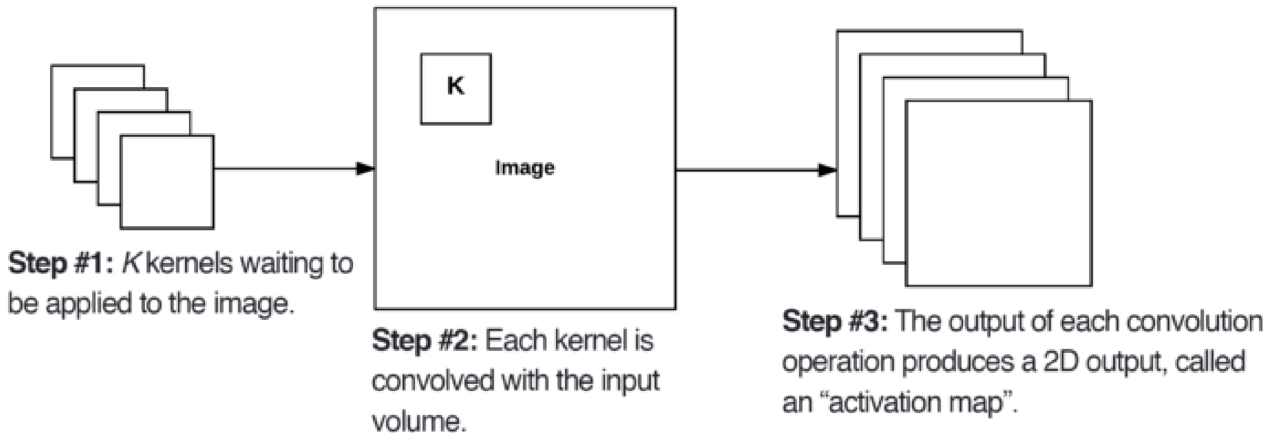


Recurrent Layer

output from nodes affects subsequent input to the nodes

LAYER TYPES: CONVOLUTIONAL LAYER





Step #1: *K* kernels waiting to be applied to the image.

Step #2: Each kernel is convolved with the input volume.

Step #3: The output of each convolution operation produces a 2D output, called an "activation map".

We have to decide:

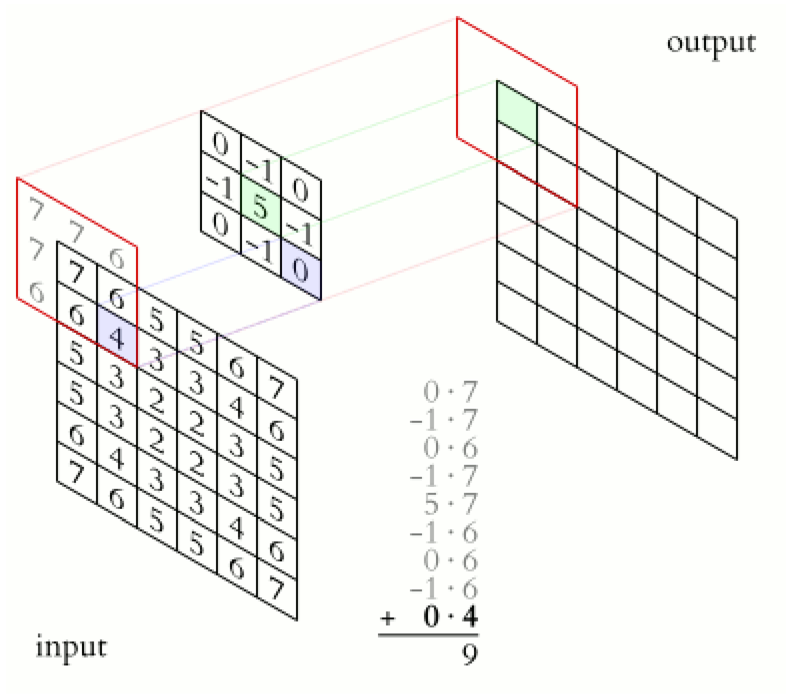
Depth: number of filters

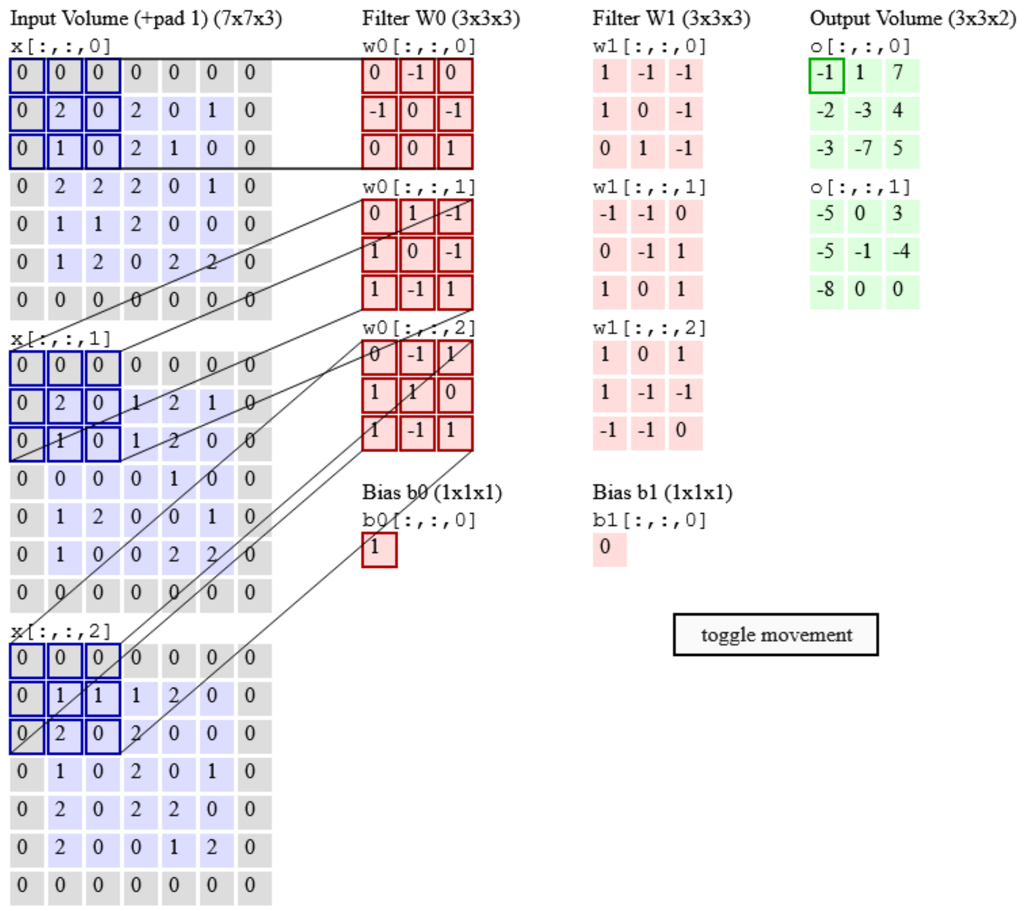
Stride: slide of the filter

→When the stride is 1 then we move the filters one pixel at a time

Padding: zeros around the border

Layer Types: Convolutional Layer





Input Volume (+pad 1) (7x7x3)

`x[:,:,0]`

0	0	0	0	0	0	0
0	2	0	2	0	1	0
0	1	0	2	1	0	0
0	2	2	2	0	1	0
0	1	1	2	0	0	0
0	1	2	0	2	2	0
0	0	0	0	0	0	0

`x[:,:,1]`

0	0	0	0	0	0	0
0	2	0	1	2	1	0
0	1	0	1	2	0	0
0	0	0	0	1	0	0
0	1	2	0	0	1	0
0	1	0	0	2	2	0
0	0	0	0	0	0	0

`x[:,:,2]`

0	0	0	0	0	0	0
0	1	1	1	2	0	0
0	2	0	2	0	0	0
0	1	0	2	0	1	0
0	2	0	2	2	0	0
0	2	0	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

`w0[:,:,0]`

0	-1	0
-1	0	-1
0	0	1

`w0[:,:,1]`

0	1	-1
1	0	-1
1	-1	1

`w0[:,:,2]`

0	-1	1
1	1	0
1	-1	1

Bias b0 (1x1x1)

`b0[:,:,0]`

1

Filter W1 (3x3x3)

`w1[:,:,0]`

1	-1	-1
1	0	-1
0	1	-1

`w1[:,:,1]`

-1	-1	0
0	-1	1
1	0	1

`w1[:,:,2]`

1	0	1
1	-1	-1
-1	-1	0

Bias b1 (1x1x1)

`b1[:,:,0]`

0

Output Volume (3x3x2)

`o[:,:,0]`

-1	1	7
-2	-3	4
-3	-7	5

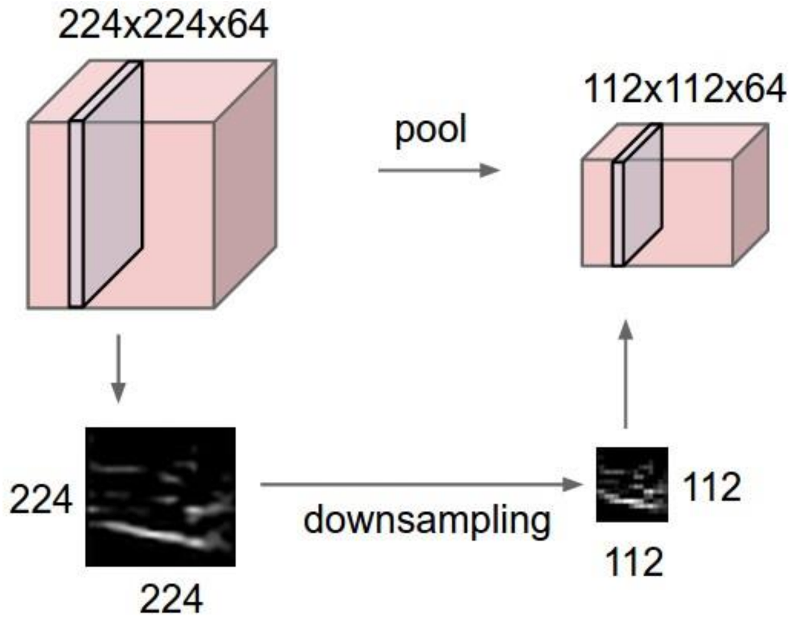
`o[:,:,1]`

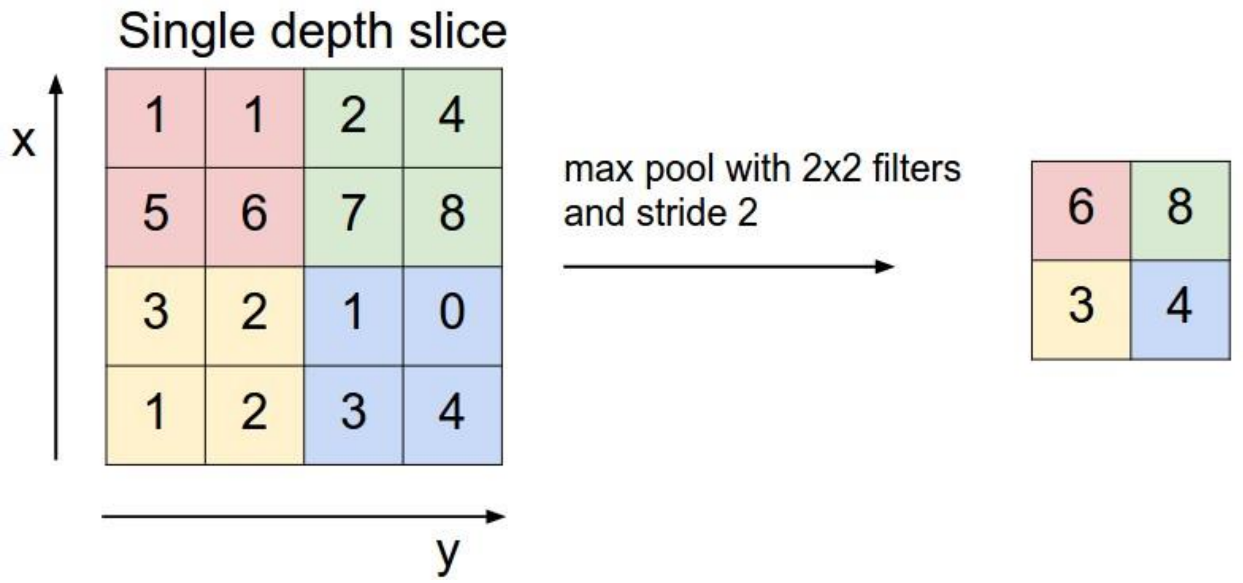
-5	0	3
-5	-1	-4
-8	0	0

Layer Types: Convolutional Layer

Pooling layer

Commonly maxpooling





1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

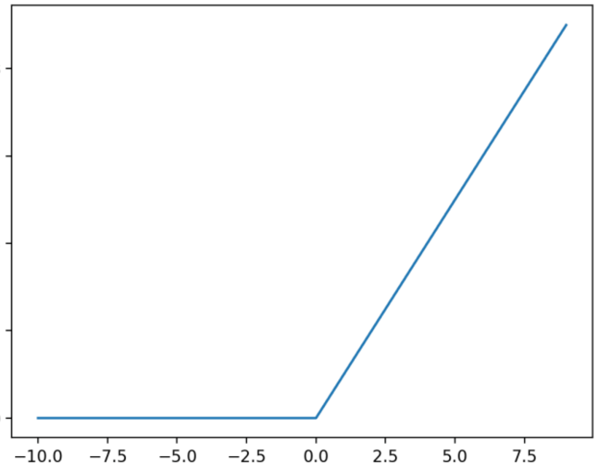
https://cs231n.github.io/convolutional-networks/

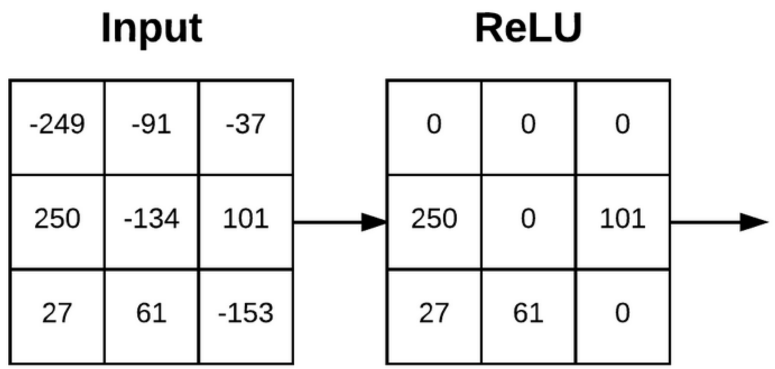
Layer Types: Convolutional Layer

Activation functions

RELU: g(z) = max{0, z}

→ better convergence performance than the sigmoid function





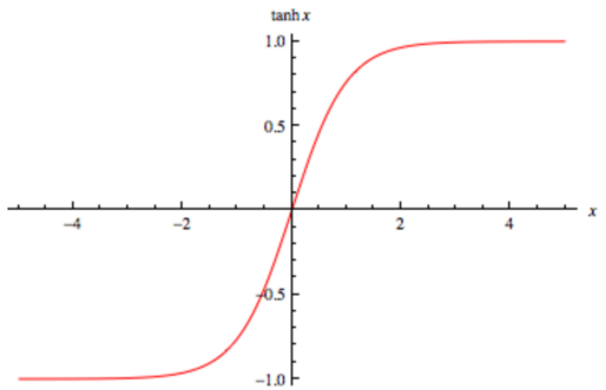
-249	-91	-37
250	-134	101
27	61	-153

0	0	0
250	0	101
27	61	0

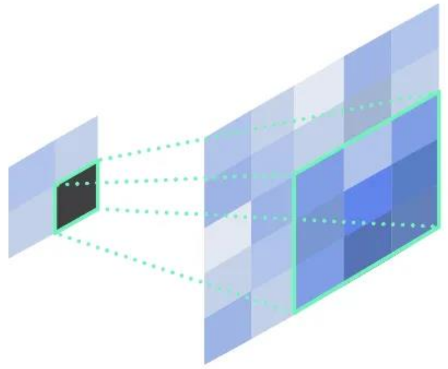
Hyperbolic Tangent Function:

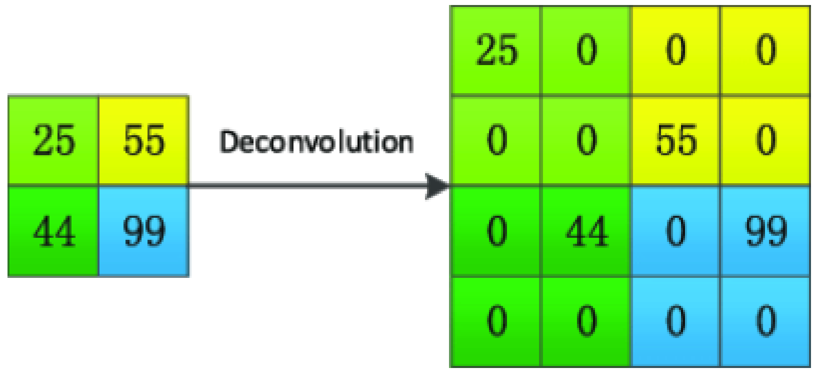
tanh(x) = $(e^x - e^{-x}) / (e^x + e^{-x})$

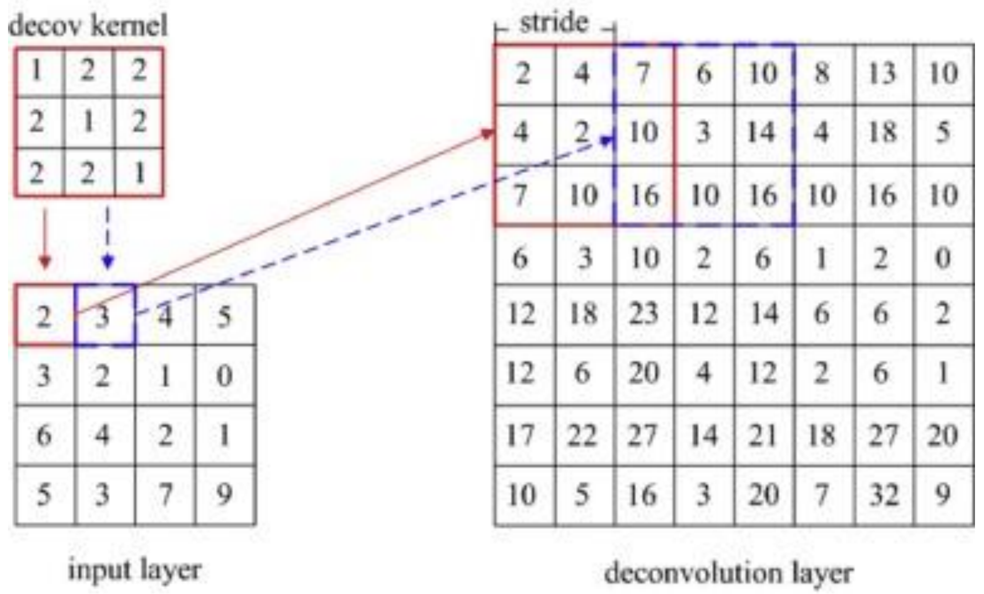
→ Output [-1, +1]



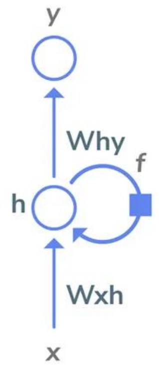
Layer Types: Deconvolution Layer

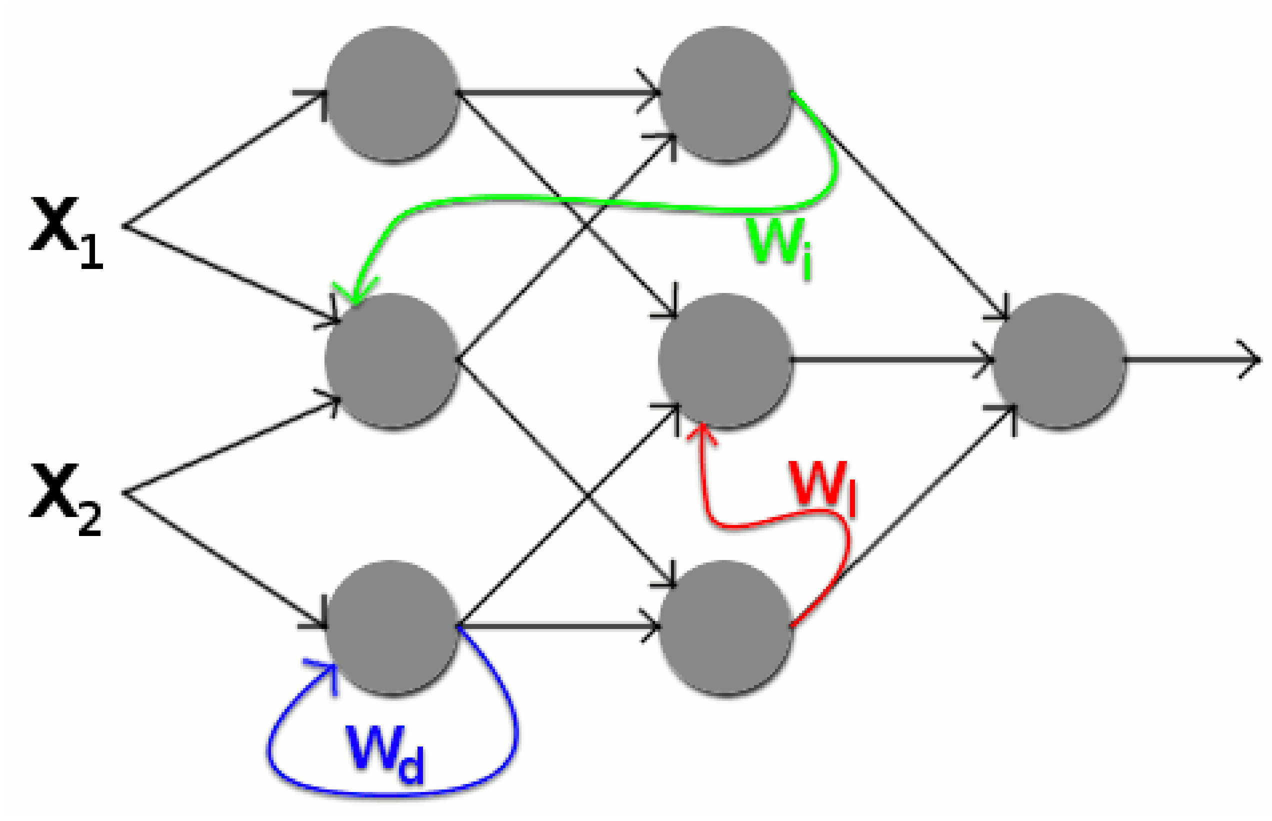






Layer Types: Recurrent Layer

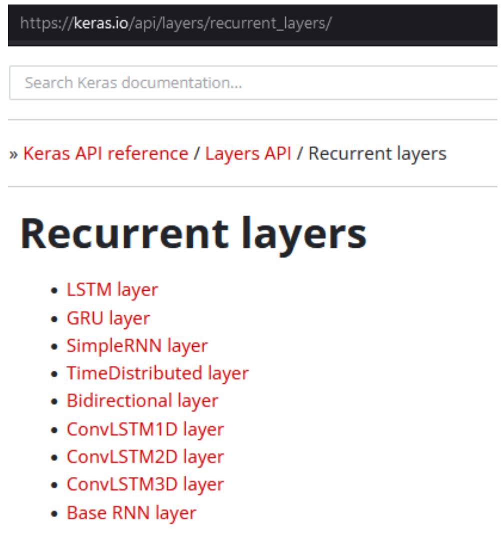




Blue: **direct feedback** (same neuron)

Green: **indirect feedback** (connection to layer – 1)

Red: **lateral feedback** (same layer)


https://keras.io/api/layers/recurrent_layers/

Search Keras documentation...

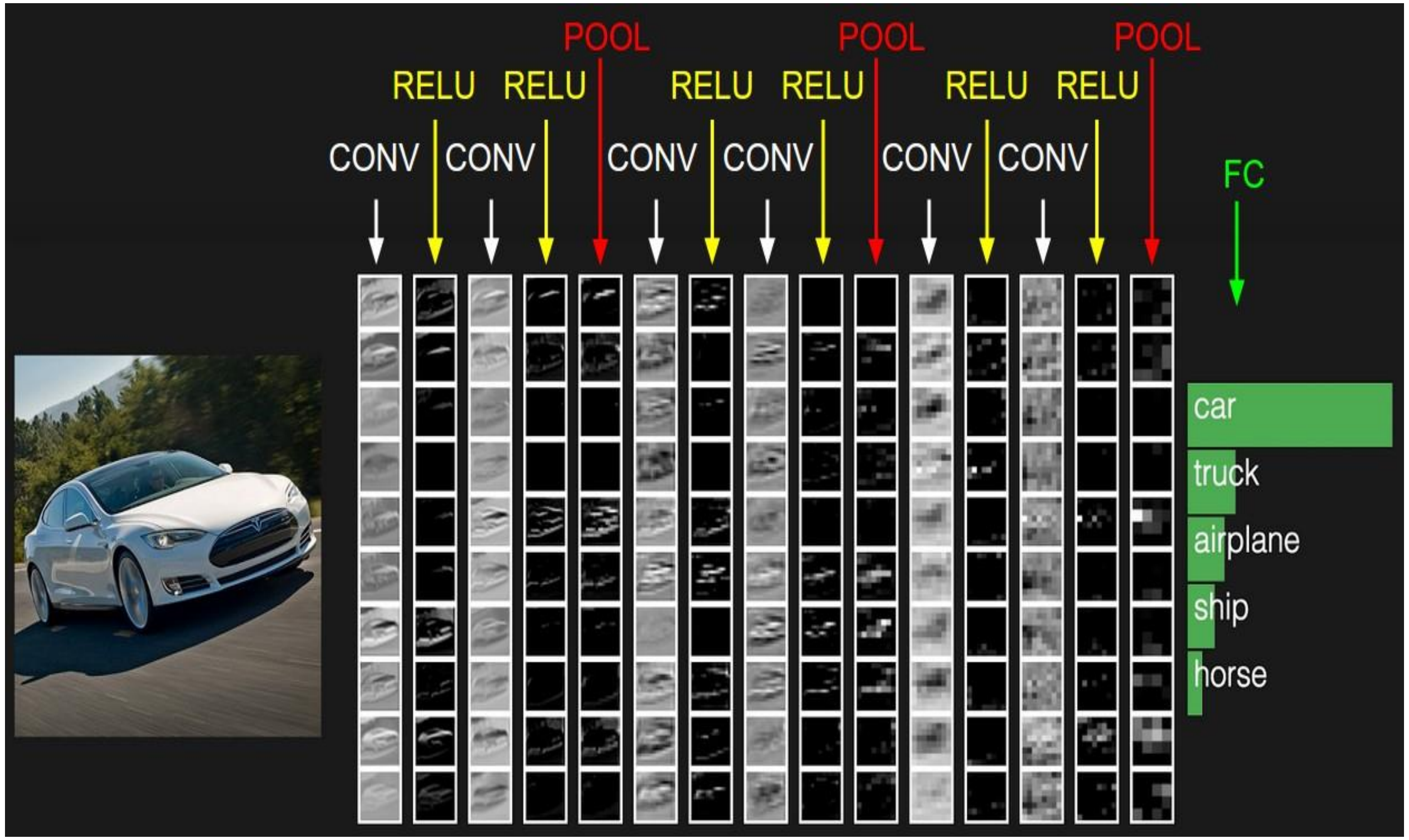
» Keras API reference / Layers API / Recurrent layers

Recurrent layers

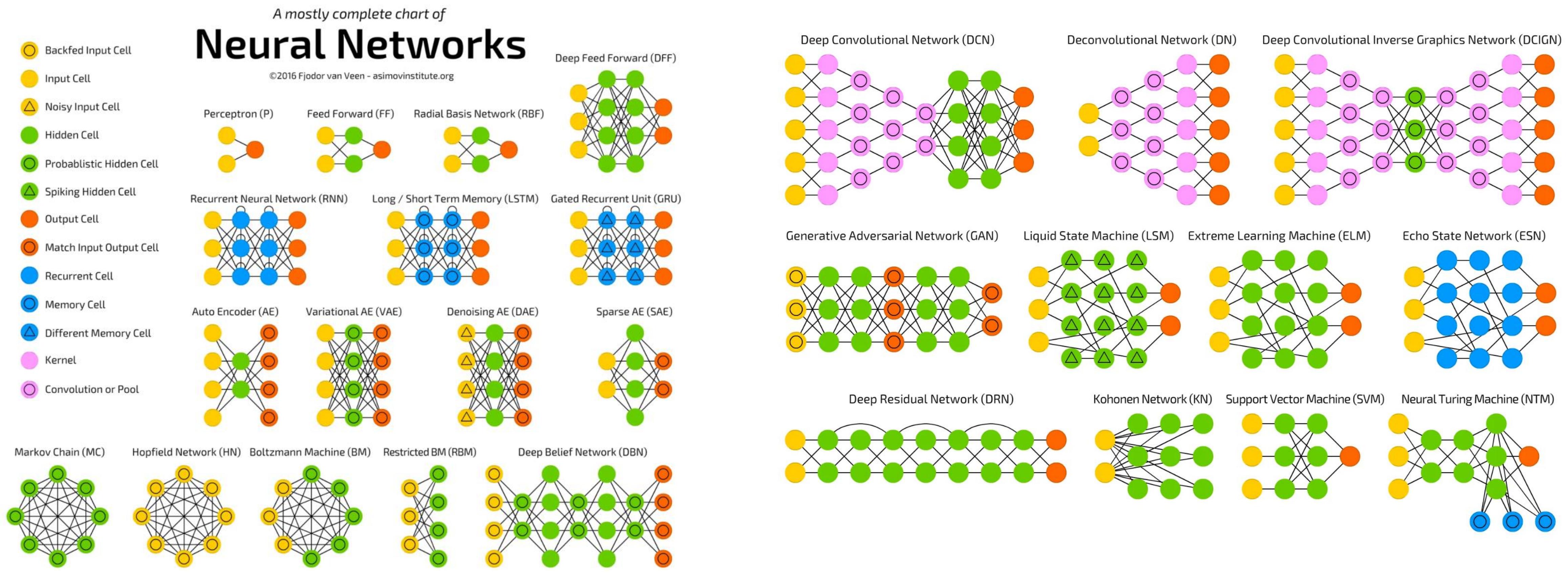
- LSTM layer
- GRU layer
- SimpleRNN layer
- TimeDistributed layer
- Bidirectional layer
- ConvLSTM1D layer
- ConvLSTM2D layer
- ConvLSTM3D layer
- Base RNN layer

Network Architecture

Example



NETWORK ARCHITECTURE OVERVIEW


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

TRAINING A NEURAL NETWORK

1. Randomly initiate weights
2. Implement forward propagation to get $h_\theta(x)$ for any x^i
3. Implement code to compute the cost function $J(\theta)$
4. Implement backpropagation to compute partial derivatives $\frac{\partial}{\partial \theta_{ji}^{(l)}} J(\theta)$

For i = 1:m

Perform forward propagation and backpropagation using the example (x^i, y^i)

→ Get activations a^l, delta terms δ^l for l = 2,…,L, $\Delta_{ji}^l := \Delta_{ji}^l + a_j^l \, \delta_i^{l+1}$

Training a neural Network

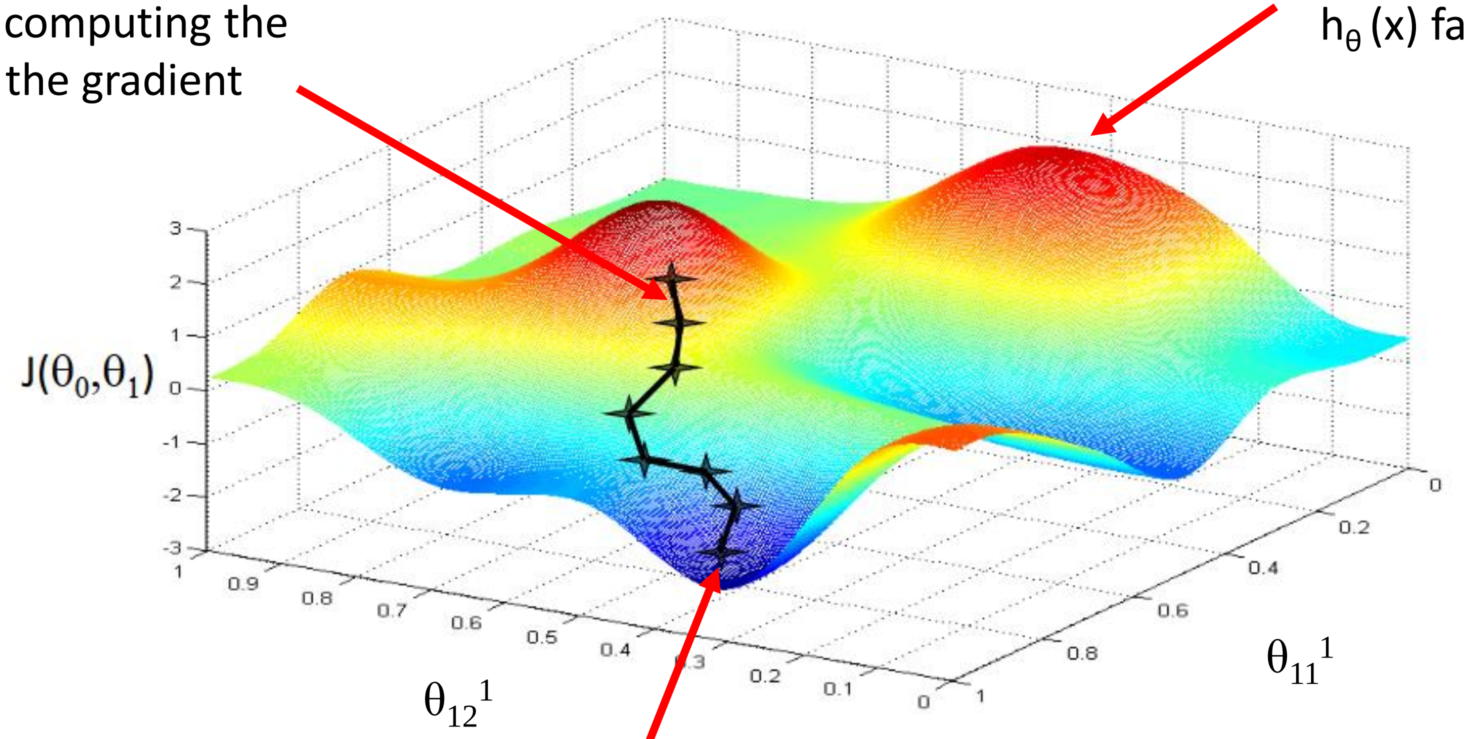
5. Use gradient checking to compare $\frac{\partial}{\partial \theta_{ji}^{(l)}} J(\theta)$ computed using backpropagation vs. using the numerical estimation of the gradient $J(\theta)$. Then disable gradient checking.

6. Use gradient descent or an advanced optimization method with backpropagation to try to minimize $J(\theta)$ as a function of the parameters θ.

Reminder: What are we Doing?

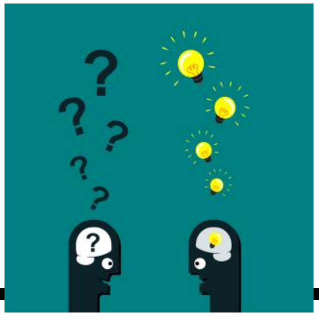
Backprop is computing the direction of the gradient

h_θ (x) far fom y^i



h_θ (x) ~ y^i

QUESTION



Suppose you are using gradient descent together with backpropagation to try to minimize J(Θ) as a function of Θ. Which of the following would be a useful step for verifying that the learning algorithm is running correctly?

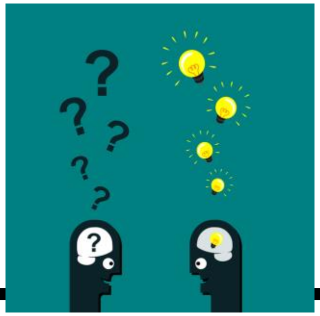
A: Plot J(Θ) as a function of Θ, to make sure gradient descent is going downhill.

B: Plot J(Θ) as a function of the number of iterations and make sure it is increasing (or at least non-decreasing) with every iteration.

C: Plot J(Θ) as a function of the number of iterations and make sure it is decreasing (or at least non-increasing) with every iteration.

D: Plot J(Θ) as a function of the number of iterations to make sure the parameter values are improving in classification accuracy.

QUESTION



Suppose you are using gradient descent together with backpropagation to try to minimize J(Θ) as a function of Θ. Which of the following would be a useful step for verifying that the learning algorithm is running correctly?

A: Plot J(Θ) as a function of Θ, to make sure gradient descent is going downhill.

B: Plot J(Θ) as a function of the number of iterations and make sure it is increasing (or at least non-decreasing) with every iteration.

[✓ C:] Plot J(Θ) as a function of the number of iterations and make sure it is decreasing (or at least non-increasing) with every iteration.

D: Plot J(Θ) as a function of the number of iterations to make sure the parameter values are improving in classification accuracy.

Wrap-Up

Cost Function

Let's first define a few variables that we will need to use:

- L = total number of layers in the network
- s_l = number of units (not counting bias unit) in layer l
- K = number of output units/classes

Recall that in neural networks, we may have many output nodes. We denote $(h_\theta(x))_k$ as being a hypothesis that results in the kth output. Our cost function for neural networks is going to be a generalization of the one we used for logistic regression.

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

WRAP-UP

Cost Function

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

We have added a few nested summations to account for our multiple output nodes. In the first part of the equation we have an additional nested summation that loops through the number of output nodes.

In the regularization part we must account for multiple theta matrices. The number of columns in our current theta matrix is equal to the number of nodes in our current layer (including the bias unit). The number of rows in our current theta matrix is equal to the number of nodes in the next layer (excluding the bias unit). As before with logistic regression, we square every term.

Note:

- the double sum adds up the logistic regression costs calculated for each cell in the output layer
- the triple sum adds up the squares of all the individual Θs in the entire network.
- the i in the triple sum does not refer to training example i

WRAP-UP

Backpropagation

"Backpropagation" is neural-network terminology for minimizing our cost function, just like what we were doing with gradient descent in logistic and linear regression. Our goal is to compute:

$\text{Min}_\theta\ J(\theta)$ and $\frac{\partial}{\partial \theta_{ji}^{(l)}} J(\theta)$

That is, we want to minimize our cost function J using an optimal set of parameters in theta. We have to look at the equations we use to compute the partial derivative of J(Θ):

- $\frac{\partial}{\partial \theta_{ji}^{(l)}} J(\theta)$

To do so, we use the following algorithm:

```
Set $\Delta_{ji}^{l} = 0$ (for all i,j,l)
For i = 1 to m
        Set $a^1 = x^1$
        Perform forward propagation to compute $a^1$ for l= 2,3,..L
        using $y^i$, compute $\delta^L = a^L - y^i$
        compute $\delta^{L-1}, \delta^{L-2}, \ldots, \delta^2$
        $\Delta_{ji}^{l} := \Delta_{ji}^{l} + a_j^l \delta_i^{l+1}$ (vectorized: $\Delta^l := \Delta^l + \delta^{l+1} a^{lT}$)
$D_{ji}^{l} := \frac{1}{m} \Delta_{ji}^{l} + \lambda \theta_{ji}^{(l)}$    if j ≠ 0
$D_{ji}^{l} := \frac{1}{m} \Delta_{ji}^{l} +$    if j ≠ 0
```

Wrap-up

In Detail: **Back propagation Algorithm**

Given training set $\{(x^1, y^1),(x^2, y^2),\ldots,(x^m, y^m)\}$

Set $\Delta_{ji}^l = 0$ (for all i,j,l) (hence you end up having a matrix full of zeros)

For training example t =1 to m:

1. Set $a^1 := x^t$

Wrap-up

In Detail: **Back propagation Algorithm**

1. Perform forward propagation to compute $a^{(l)}$ for l=2,3,…,L

➔ Forward propagation example:

- $a^{(1)} = x$
- $z^{(2)} = \theta^{(1)}a^{(1)}$
- $a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)
- $z^{(3)} = \theta^{(2)}a^{(2)}$
- $a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)
- $z^{(4)} = \theta^{(3)}a^{(3)}$
- $a^{(4)} = h_\theta(x) = g(z^{(4)})$

WRAP-UP

In Detail: **Back propagation Algorithm**

3. Using y^i, compute $\delta^L = a^L - y^i$

Where L is our total number of layers and a^L is the vector of outputs of the activation units for the last layer. So our "error values" for the last layer are simply the differences of our actual results in the last layer and the correct outputs in y. To get the delta values of the layers before the last layer, we can use an equation that steps us back “from right to left”:

4. compute $\delta^{L-1}, \delta^{L-2}, \ldots, \delta^2$ using $\delta^l = (\theta^l)^T \delta^{l+1} .* a^l .*(1-a^l)$

The delta values of layer l are calculated by multiplying the delta values in the next layer with the theta matrix of layer l. We then element-wise multiply that with a function called g', or g-prime, which is the derivative of the activation function g evaluated with the input values given by $z^{(l)}$. The g-prime derivative terms can also be written out as: $g'(z^l) = a^l .*(1-a^l)$

WRAP-UP

In Detail: **Back propagation Algorithm**

5. $\Delta_{ji}^{l} := \Delta_{ji}^{l} + a_j^{l}\,\delta_i^{l+1}$ (vectorized: $\Delta^{l} := \Delta^{l} + \delta^{\,l+1} a^{lT}$)

Hence we update our new Δ matrix.

$D_{ji}^{l} := \frac{1}{m}\Delta_{ji}^{l} + \lambda\,\theta_{ji}^{(l)}$ if $j \neq 0$

$D_{ji}^{l} := \frac{1}{m}\Delta_{ji}^{l}$ if $j \neq 0$

The capital-delta matrix D is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative. Thus we get $\frac{\partial}{\partial\theta_{ji}^{(l)}} J(\theta) = D_{ji}^{l}$

WRAP-UP

Intuition Back propagation

- The cost function for a neural network is:

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

- If we consider simple non-multiclass classification (k = 1) and disregard regularization, the cost is computed with:

$cost(i) = y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)}_k)\log(1 - h_\theta(x^{(i)}))$

- Intuitively, δ_i^l is the "error" for $a_j^{(l)}$ (unit j in layer l). More formally, the delta values are actually the derivative of the cost function:

$\delta_i{}^l = \frac{\partial}{\partial z_j{}^l}$

- Recall that our derivative is the slope of a line tangent to the cost function, so the steeper the slope the more incorrect we are.

Wrap-up

Gradient Checking

- Gradient checking will assure that our backpropagation works as intended. We can approximate the derivative **with respect to** Θ_j as follows: $\frac{\partial}{\partial \theta_j} J(\Theta) \sim \frac{J(\Theta_1, \Theta_j + \epsilon, \ldots, \Theta_n) - J(\Theta_1, \Theta_j - \epsilon, \ldots, \Theta_n)}{2 * \epsilon}$ (choose a small value for Epsilon)
- After implementation we can check whether the approximation is similar to our delta vectors. Once you have verified once that your backpropagation algorithm is correct, you don not need to compute the approximation again. The code to compute the approximation can be very slow.

WRAP-UP

Random Initialization

- Initializing all theta weights to zero does not work with neural networks. When we backpropagate, all nodes will update to the same value repeatedly. Instead we must randomly initialize our weights for our Θ matrices. We initialize each $\Theta_{ij}^{(l)}$ to a random value between$[-\epsilon,\epsilon]$.

→ $-\epsilon <= \theta_{ji}^{(l)} <= \epsilon$

WRAP-UP

We can use different layer types, such as:

- **Fully connected layer:** Here every neuron in one layer is connected to every neuron in the next layer.
- **Deconvolution Layer:** For “upsampling” data.
- **Convolutional Layer:** In order to “filter” the input.
    - **We have to decide on:**
    - **Depth:** number of filters
    - **Stride:** slide of the filter
    - →When the stride is 1 then we move the filters one pixel at a time
    - **Padding:** Are there zeros around the border?
    - We can user different Kernels and different activations functions (e.g. RELU).
- **Recurrent Layer:** The output from nodes affects subsequent input to the nodes.

Wrap-Up

Training the network

First, pick a network architecture; choose the layout of your neural network, including how many hidden units in each layer and how many layers in total you want to have.

- Number of input units = dimension of features $x^{(i)}$
- Number of output units = number of classes
- Number of hidden units per layer = usually more the better (must balance with cost of computation as it increases with more hidden units)
- Defaults: 1 hidden layer. If you have more than 1 hidden layer, then it is recommended that you have the same number of units in every hidden layer.

Then:

1. Randomly initialize the weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement the cost function
4. Implement backpropagation to compute partial derivatives
5. Use gradient checking to confirm that your backpropagation works. Then disable gradient checking.
6. Use gradient descent or a built-in optimization function to minimize the cost function with the weights in theta.

QUIZ - QUESTION 1

You are training a three layer neural network and would like to use backpropagation to compute the gradient of the cost function. In the backpropagation algorithm, one of the steps is to update

$\Delta_{ij}^{(2)} := \Delta_{ij}^{(2)} + \delta_i^{(3)} * (a^{(2)})_j$

for every i,j. Which of the following is a correct vectorization of this step?

A: $\Delta^{(2)} := \Delta^{(2)} + (a^{(2)})^T * \delta^{(3)}$

B: $\Delta^{(2)} := \Delta^{(2)} + \delta^{(3)} * (a^{(2)})^T$

C: $\Delta^{(2)} := \Delta^{(2)} + \delta^{(3)} * (a^{(3)})^T$

D: $\Delta^{(2)} := \Delta^{(2)} + (a^{(3)})^T * \delta^{(3)}$

QUIZ - QUESTION 1

$\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$

You are training a three layer neural network and would like to use backpropagation to compute the gradient of the cost function. In the backpropagation algorithm, one of the steps is to update

$\Delta_{ij}^{(2)} := \Delta_{ij}^{(2)} + \delta_i^{(3)} * (a^{(2)})_j$

for every i,j. Which of the following is a correct vectorization of this step?

A: $\Delta^{(2)} := \Delta^{(2)} + (a^{(2)})^T * \delta^{(3)}$

$\cancel{B}$: $\Delta^{(2)} := \Delta^{(2)} + \delta^{(3)} * (a^{(2)})^T$

C: $\Delta^{(2)} := \Delta^{(2)} + \delta^{(3)} * (a^{(3)})^T$

D: $\Delta^{(2)} := \Delta^{(2)} + (a^{(3)})^T * \delta^{(3)}$

Quiz - Question 2

Suppose Theta1 is a 5x3 matrix, and Theta2 is a 4x6 matrix. You set thetaVec=[Theta1(:);Theta2(:)]. Which of the following correctly recovers Theta2?

A: reshape(thetaVec(16:39),4,6)

B: reshape(thetaVec(15:38),4,6)

C: reshape(thetaVec(16:24),4,6)

D: reshape(thetaVec(15:39),4,6)

E: reshape(thetaVec(16:39),6,4)

QUIZ - QUESTION 2

Suppose Theta1 is a 5x3 matrix, and Theta2 is a 4x6 matrix. You set thetaVec=[Theta1(:);Theta2(:)]. Which of the following correctly recovers Theta2?

[✓ A: reshape(thetaVec(16:39),4,6)]

B: reshape(thetaVec(15:38),4,6)

C: reshape(thetaVec(16:24),4,6)

D: reshape(thetaVec(15:39),4,6)

E: reshape(thetaVec(16:39),6,4)

Quiz - Question 3

Let J(θ) = 2θ^4 + 2. Let θ = 1 and ϵ = 0.01. Use the formula

$$\frac{J(\Theta + \epsilon) - J(\Theta - \epsilon)}{2 * \epsilon}$$

to numerically compute an approximation to the derivative at θ = 1. What value do you get?

A: 8.0008

B: 7.9992

C: 10

D: 8

QUIZ - QUESTION 3

Let $J(\theta) = 2\theta^4 + 2$. Let $\theta = 1$ and $\epsilon = 0.01$. Use the formula

$$\frac{J(\Theta + \epsilon) - J(\Theta - \epsilon)}{2 * \epsilon}$$

to numerically compute an approximation to the derivative at $\theta = 1$. What value do you get?

$$\frac{(2(1 + 0.01)^4 + 2) - (2(1 - 0.01)^4 + 2)}{2 * 0.01}$$

$\cancel{\text{A}}$: 8.0008

B: 7.9992

C: 10

D: 8

QUIZ - QUESTION 4

Which of the following statements are true? Check all that apply.

A: Using a large value of λ cannot hurt the performance of your neural network; the only reason we do not set λ to be too large is to avoid numerical problems.

B: Using gradient checking can help verify if one's implementation of backpropagation is bug-free.

C: If our neural network overfits the training set, one reasonable step to take is to increase the regularization parameter λ.

D: Gradient checking is useful if we are using gradient descent as our optimization algorithm. However, it serves little purpose if we are using one of the advanced optimization methods.

Quiz - Question 4

Which of the following statements are true? Check all that apply.

A: Using a large value of λ cannot hurt the performance of your neural network; the only reason we do not set λ to be too large is to avoid numerical problems.

[✓ B]: Using gradient checking can help verify if one's implementation of backpropagation is bug-free.

[✓ C]: If our neural network overfits the training set, one reasonable step to take is to increase the regularization parameter λ.

D: Gradient checking is useful if we are using gradient descent as our optimization algorithm. However, it serves little purpose if we are using one of the advanced optimization methods.

QUIZ - QUESTION 5

Which of the following statements are true? Check all that apply.

A: Suppose you are training a neural network using gradient descent. Depending on your random initialization, your algorithm may converge to different local optima (i.e., if you run the algorithm twice with different random initializations, gradient descent may converge to two different solutions).

B: If we initialize all the parameters of a neural network to ones instead of zeros, this will suffice for the purpose of "symmetry breaking" because the parameters are no longer symmetrically equal to zero.

C: If we are training a neural network using gradient descent, one reasonable "debugging" step to make sure it is working is to plot $J(\Theta)$ as a function of the number of iterations, and make sure it is decreasing (or at least non-increasing) after each iteration.

D: Suppose you have a three layer network with parameters $\Theta^{(1)}$(controlling the function mapping from the inputs to the hidden units) and $\Theta^{(2)}$ (controlling the mapping from the hidden units to the outputs). If we set all the elements of $\Theta^{(1)}$ to be 0, and all the elements of $\Theta^{(2)}$ to be 1, then this suffices for symmetry breaking, since the neurons are no longer all computing the same function of the input.

QUIZ - QUESTION 5

Which of the following statements are true? Check all that apply.

[✓ A]: Suppose you are training a neural network using gradient descent. Depending on your random initialization, your algorithm may converge to different local optima (i.e., if you run the algorithm twice with different random initializations, gradient descent may converge to two different solutions).

B: If we initialize all the parameters of a neural network to ones instead of zeros, this will suffice for the purpose of "symmetry breaking" because the parameters are no longer symmetrically equal to zero.

[✓ C]: If we are training a neural network using gradient descent, one reasonable "debugging" step to make sure it is working is to plot $J(\Theta)$ as a function of the number of iterations, and make sure it is decreasing (or at least non-increasing) after each iteration.

D: Suppose you have a three layer network with parameters $\Theta^{(1)}$(controlling the function mapping from the inputs to the hidden units) and $\Theta^{(2)}$ (controlling the mapping from the hidden units to the outputs). If we set all the elements of $\Theta^{(1)}$ to be 0, and all the elements of $\Theta^{(2)}$ to be 1, then this suffices for symmetry breaking, since the neurons are no longer all computing the same function of the input.