

# *Neural Networks*

*Jaegul Choo* (주재걸)  
Korea University

<https://sites.google.com/site/jaegulchoo/>

Slides made by my student, Yunjey Choi

## *Today's Contents*

### **Neural Networks**

**1. Deep Learning**

**2. Perceptron**

**3. Neural Network**

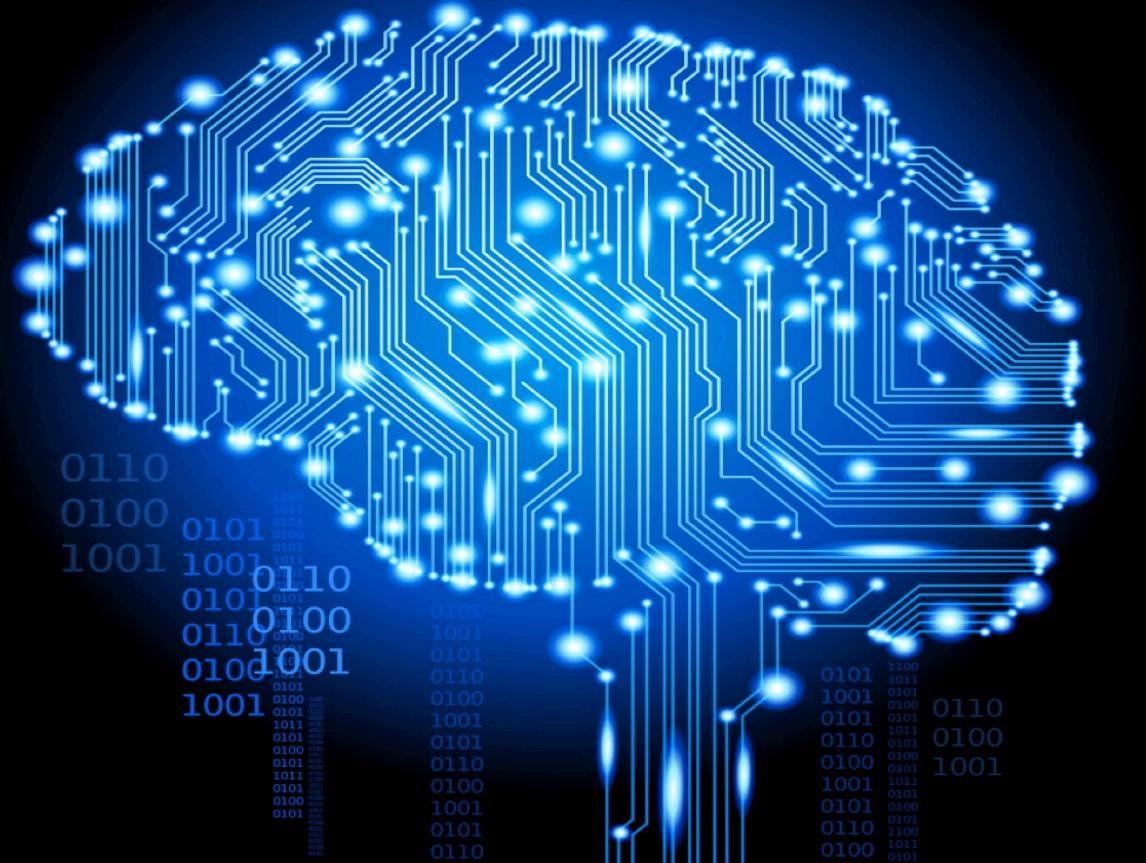
**4. Forward Propagation**

**5. Backpropagation**

# **1. Deep Learning**

# *Deep Learning*

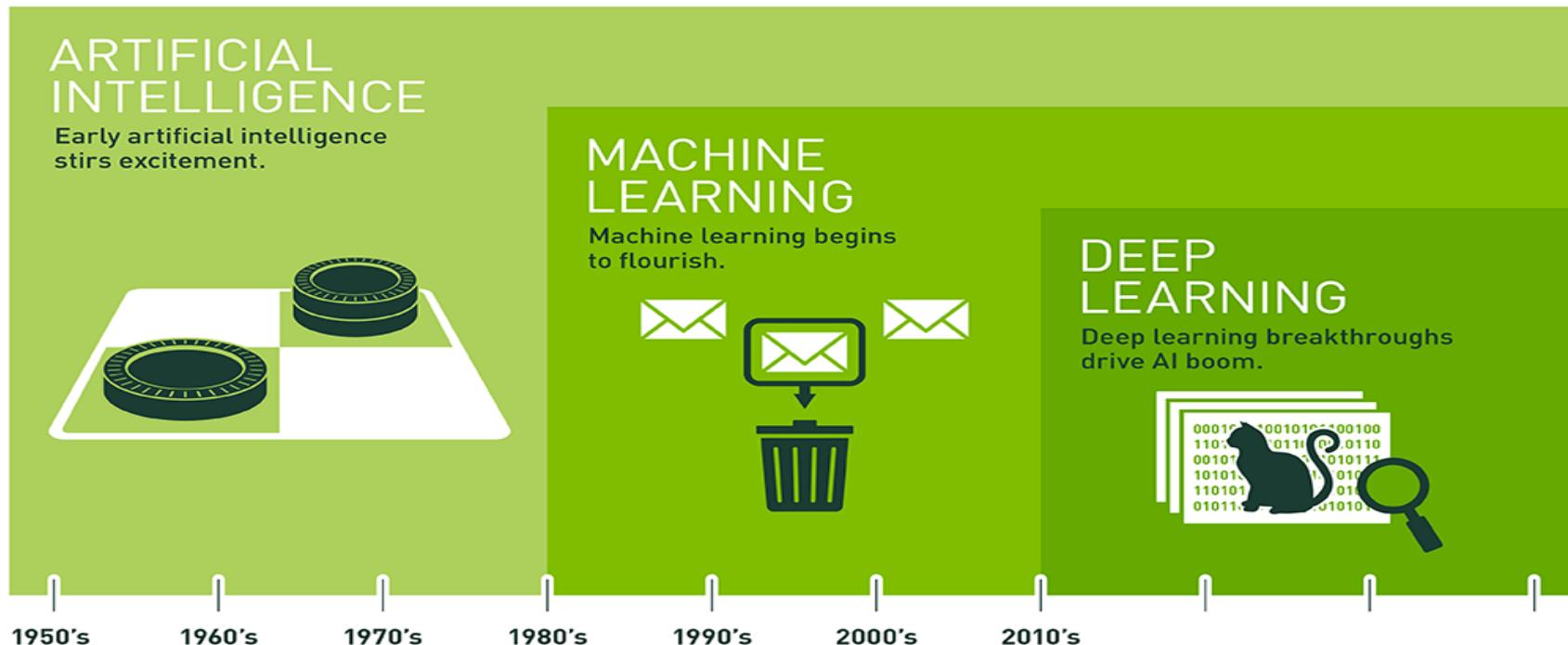
**Deep learning refers to artificial neural networks that are composed of many layers.**



# *Deep Learning*

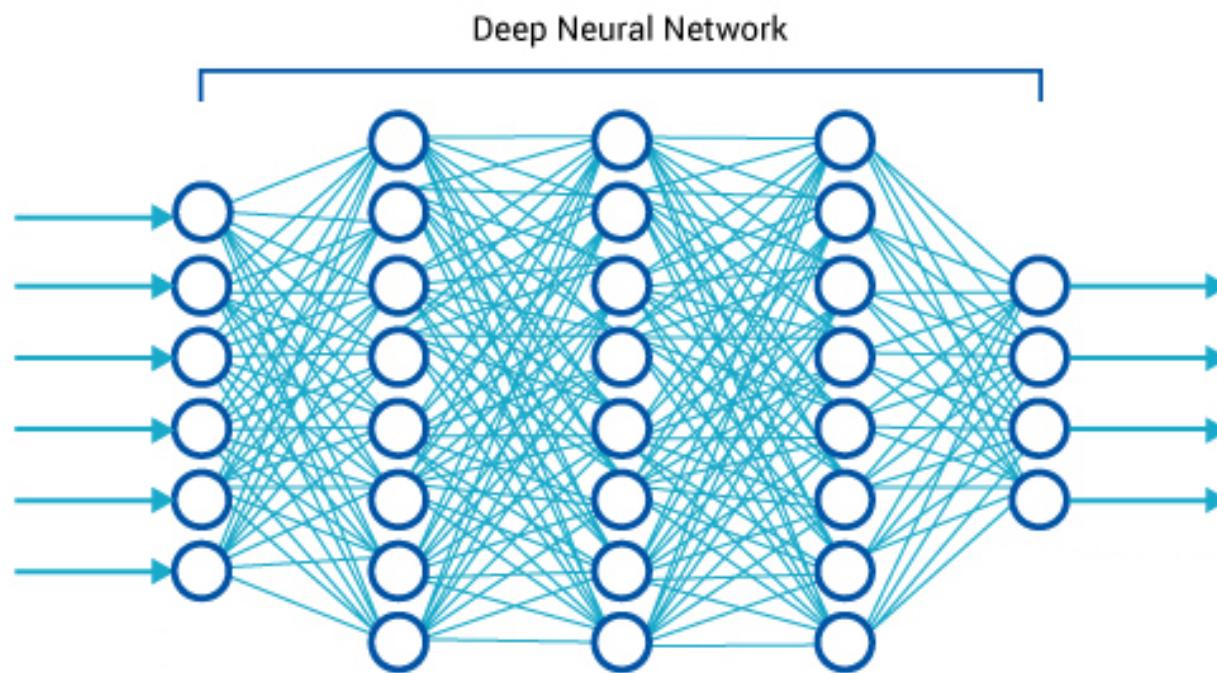


# Deep Learning

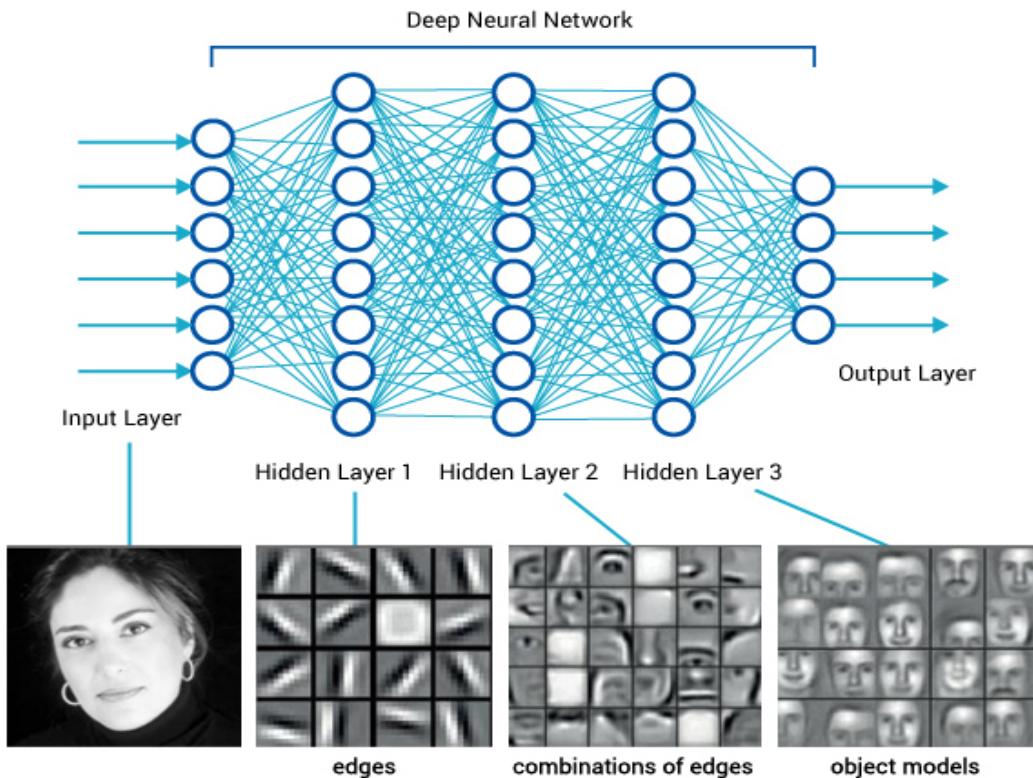


Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

# *Deep Learning*



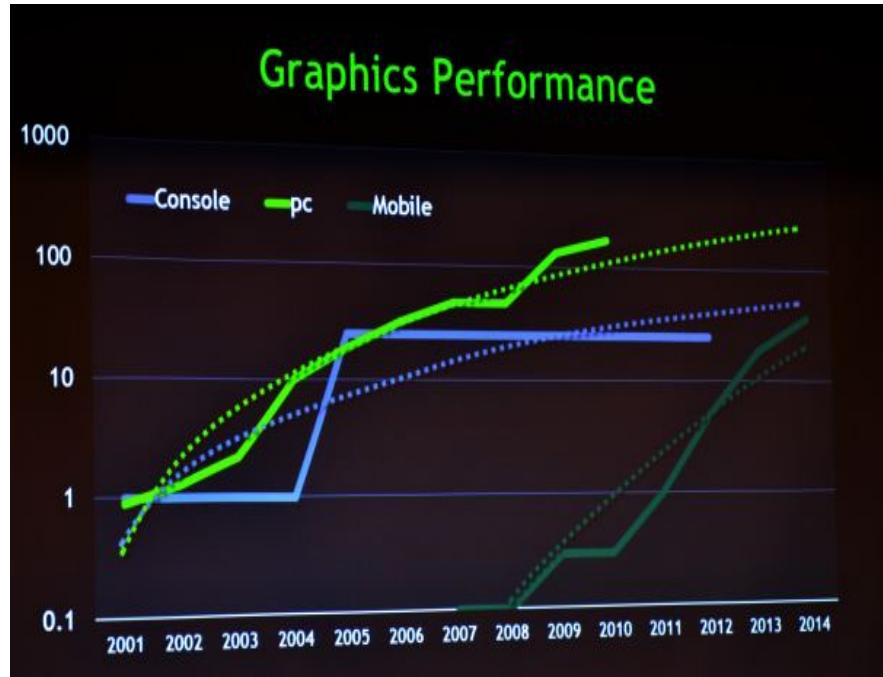
# Deep Learning



**Why is deep learning a growing trend?**

- **few feature engineering**
- **state-of-the-art performance**

# *Deep Learning*



## *Deep Learning Heroes*



[Geoffrey Hinton](#)



[Yann LeCun](#)



[Andrew Ng](#)



[Yoshua Bengio](#)

# Lectures for Neural Network

**coursera**

Catalog Search catalog 

Institutions  ▾

Home ▶ Data Science ▶ Machine Learning

## Neural Networks for Machine Learning

**About this course:** Learn about artificial neural networks and how they're being used for machine learning, as applied to speech and object recognition, image segmentation, modeling language and human motion, etc. We'll emphasize both the basic algorithms and the practical tricks needed to get them to work well.

▼ More

**Created by:** University of Toronto



**Taught by:** Geoffrey Hinton, Professor  
Department of Computer Science

Financial Aid is available for learners who cannot afford the fee. [Learn more and apply.](#)

Go to Course

Already enrolled



[Neural Networks for Machine Learning - University of Toronto](#)

# Lectures for Neural Network

The screenshot shows the Coursera website interface. At the top, there is a navigation bar with the Coursera logo, a 'Catalog' button, a search bar containing 'Search catalog', and a magnifying glass icon. To the right of the search bar are links for 'Institutions' and a blue circular icon with 'YC' and a dropdown arrow. Below the navigation bar, the main content area displays the course page for 'Machine Learning'.

The course page header includes the course title 'Machine Learning' in large white text, followed by a blurred background image of a person working at a computer. On the left side, there is a sidebar with links: 'Overview', 'Syllabus', 'Creators', 'Ratings and Reviews', and a larger section for 'Machine Learning'. This section contains a 'Go to Course' button (which is blue and outlined) and a link 'Already enrolled'. Below this, a note states: 'Financial Aid is available for learners who cannot afford the fee. [Learn more and apply.](#)'

The main content area starts with the heading 'About this course:' followed by a detailed description of machine learning. It includes a 'More' link with a downward arrow. Below this, it says 'Created by: Stanford University' and shows the Stanford University logo (a red rectangle with 'Stanford University' written on it). Further down, it lists 'Taught by: Andrew Ng, Associate Professor, Stanford University; Chief Scientist, Baidu; Chairman and Co-founder, Coursera' and includes a small circular profile picture of Andrew Ng.

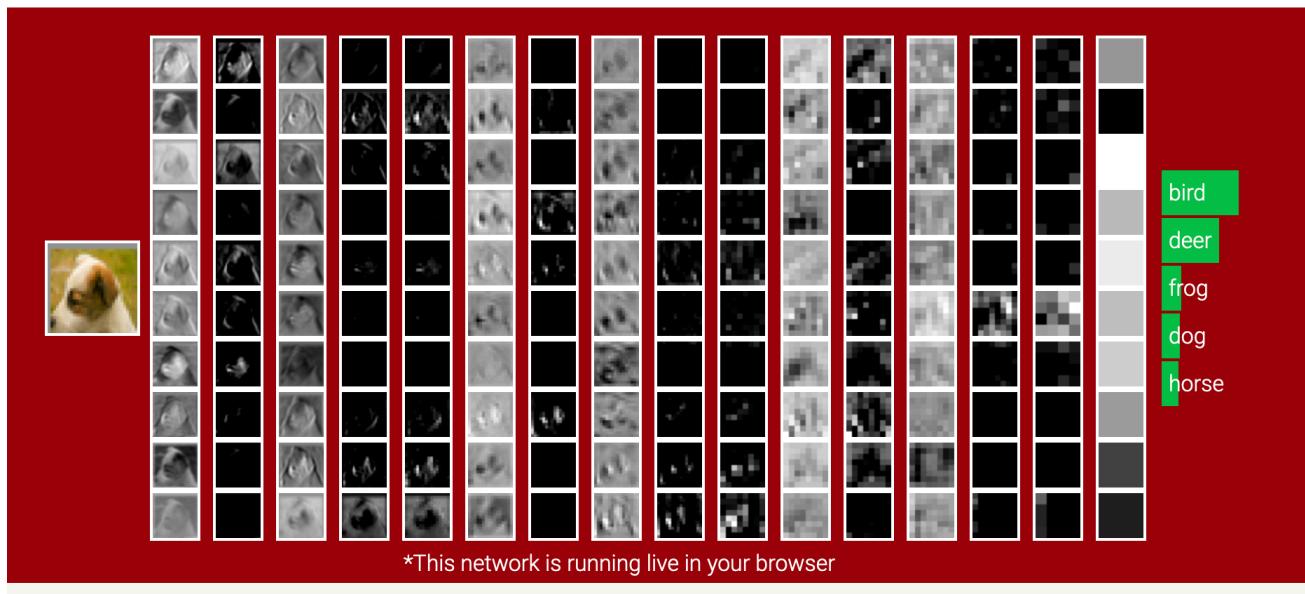
[Machine Learning - Stanford University](#)

# Lectures for Neural Network

CS231n: Convolutional Neural Networks for Visual Recognition

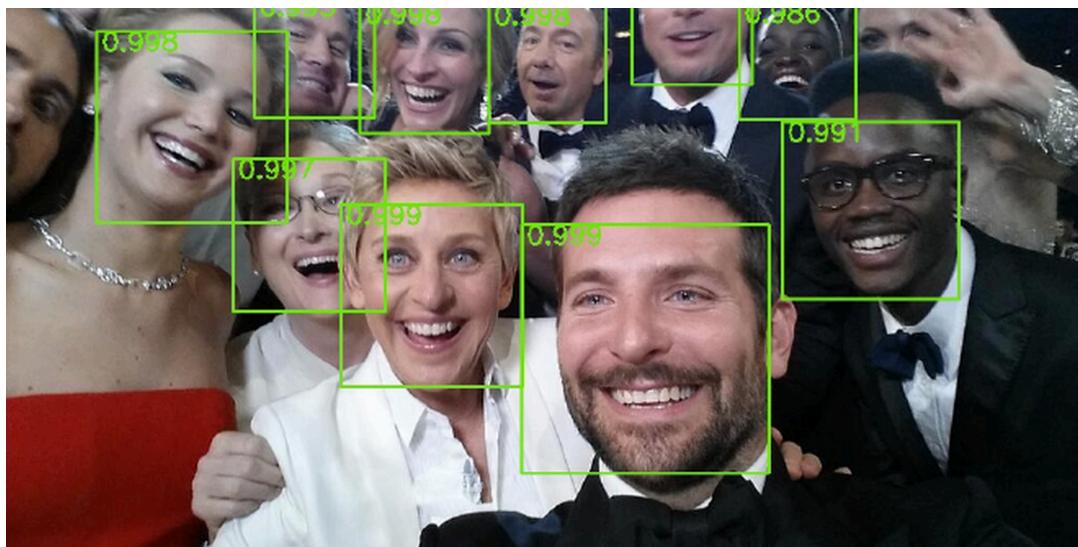
Spring 2019

Previous Years: [\[Winter 2015\]](#) [\[Winter 2016\]](#) [\[Spring 2017\]](#) [\[Spring 2018\]](#)

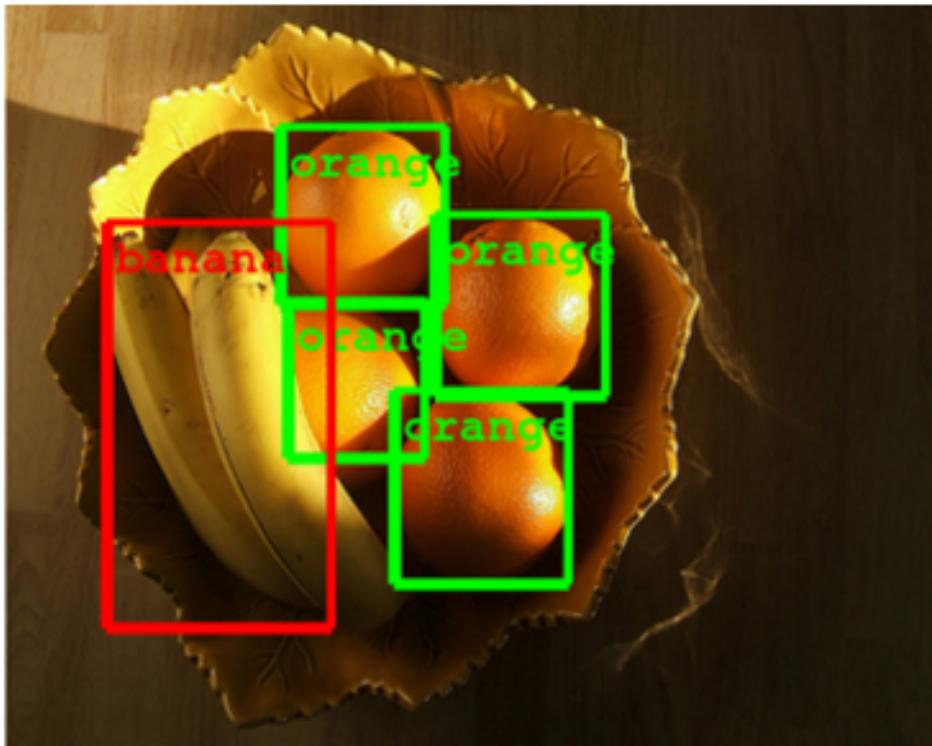
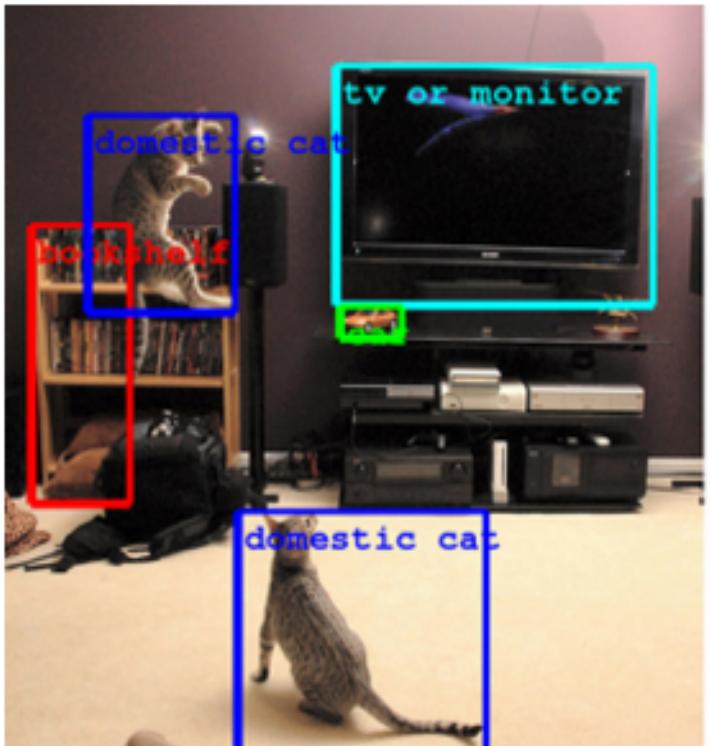


CS231n – Stanford University

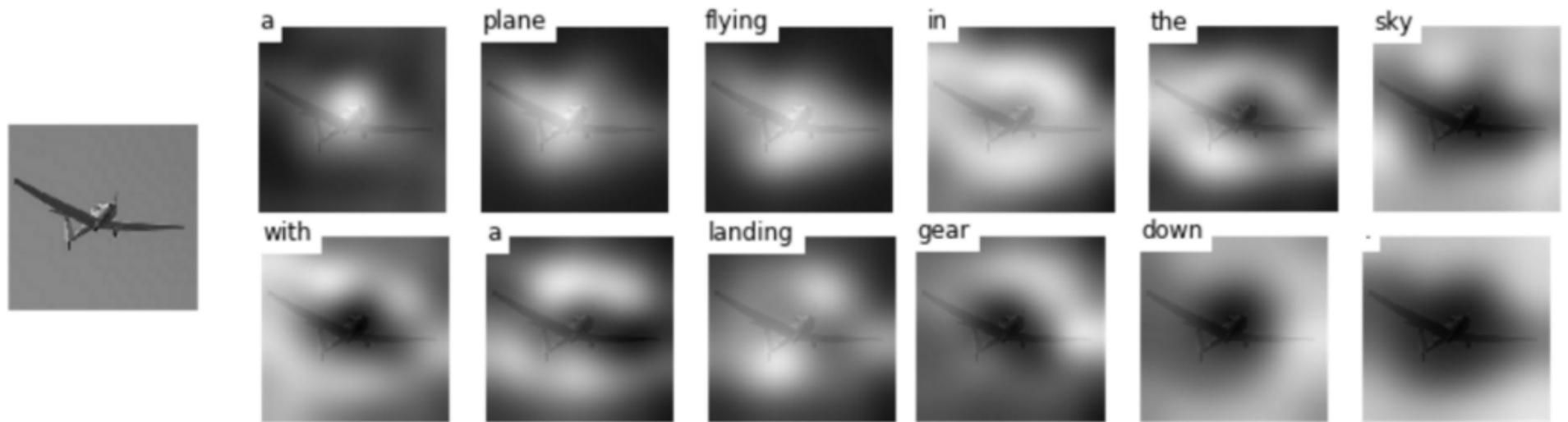
# *Face Detection & Recognition*



# *Object Detection & Recognition*



# *Image Captioning*



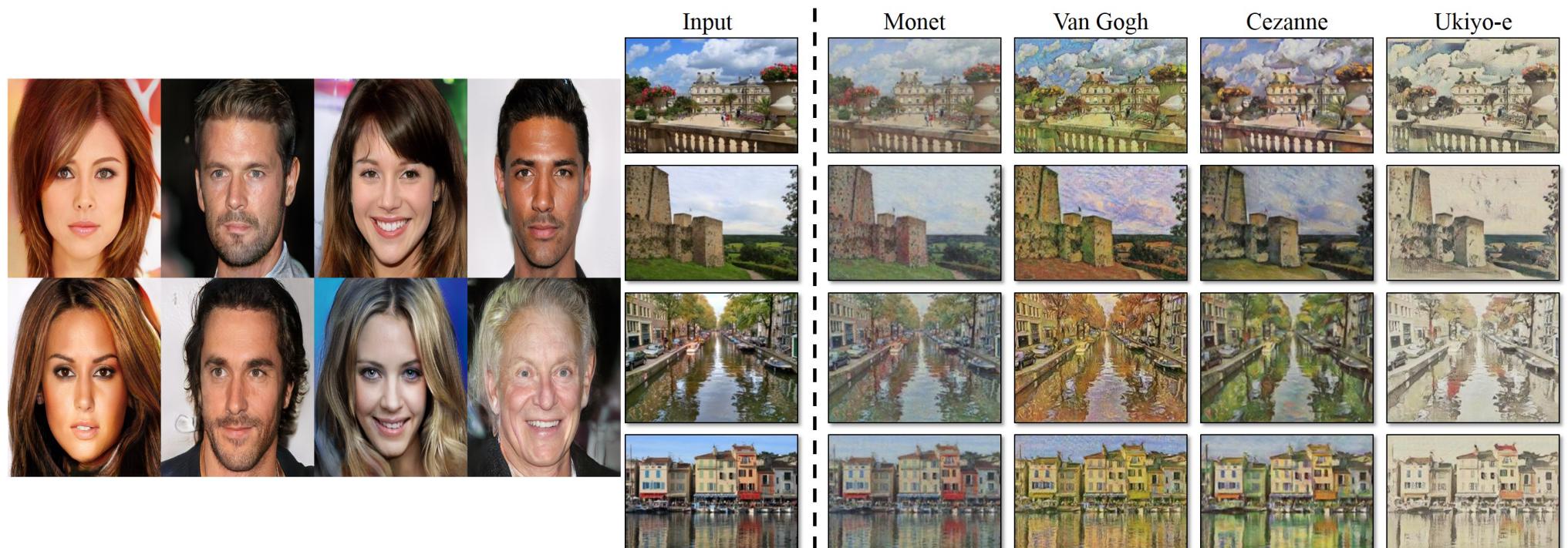
<https://github.com/yunjey/show-attend-and-tell>

# *Image Captioning*



[DenseCap: Fully Convolutional Localization Networks for Dense Captioning](#)

# *Image Generation/Translation*



# *Deep Fake*



## Deepfakes Are Coming. And They're Dangerous.

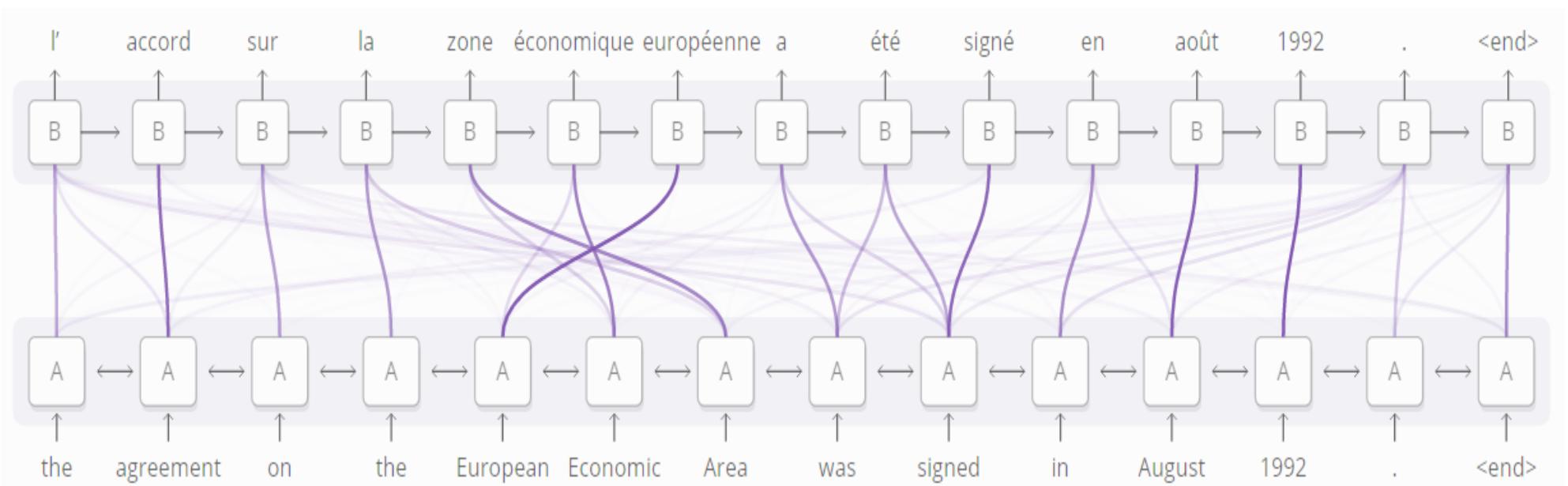
by Jenna Lifhits | July 20, 2018 06:19 AM



Marco Rubio warns that the United States is not ready for the havoc that impersonation technology can wreak.

<https://www.youtube.com/watch?v=tISBgFxY3SE>

# Machine Translation



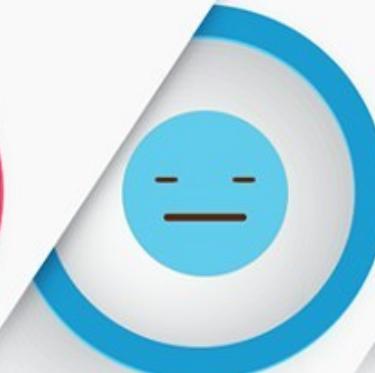
# *Sentiment Analysis*

## SENTIMENT ANALYSIS



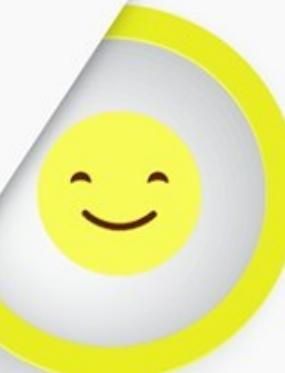
### NEGATIVE

Totally dissatisfied with the service. Worst customer care ever.



### NEUTRAL

Good Job but I will expect a lot more in future.



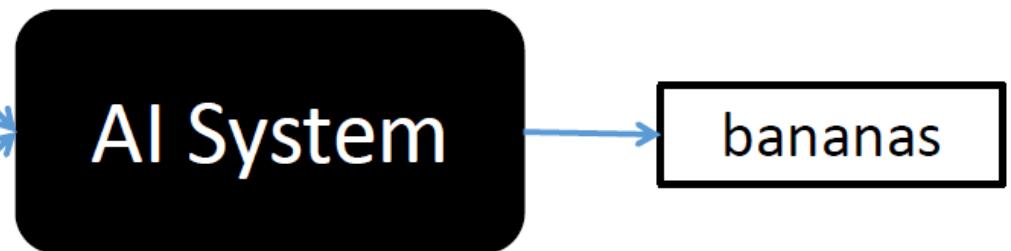
### POSITIVE

Brilliant effort guys! Loved Your Work.

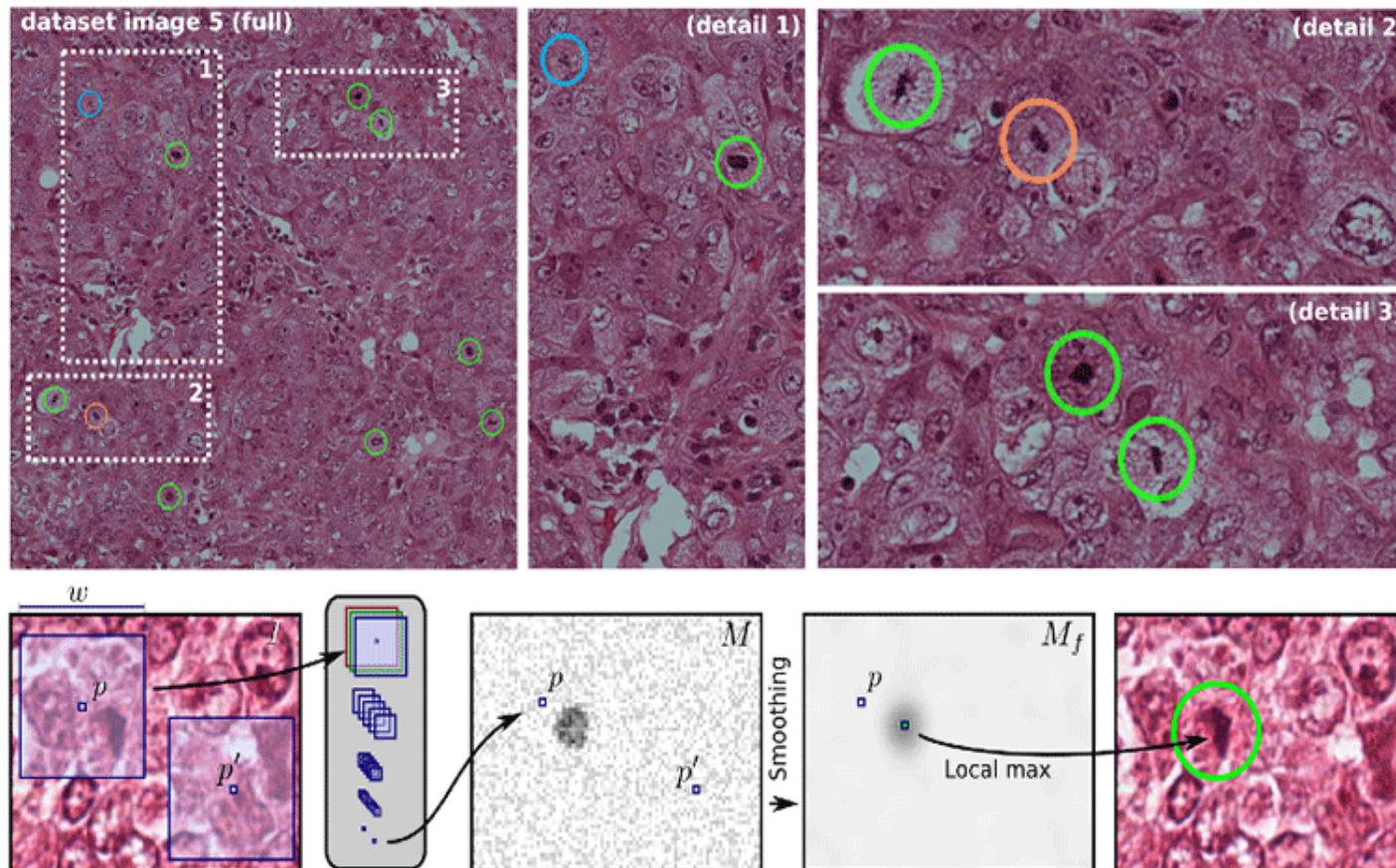
# *Question Answering*



What is the mustache  
made of?



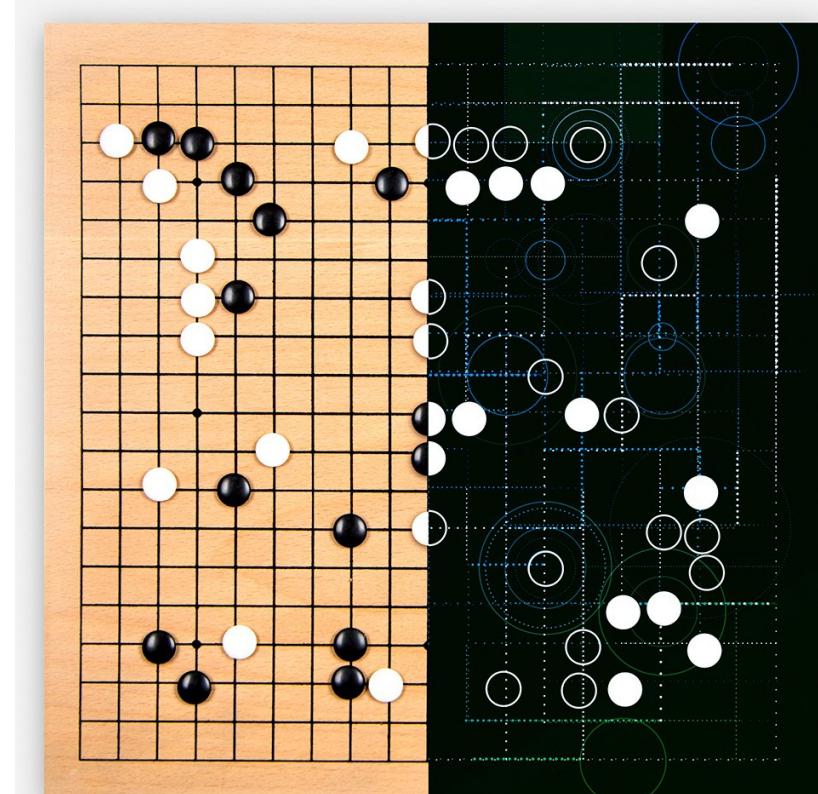
# Medical Analysis



# *Recommender System*

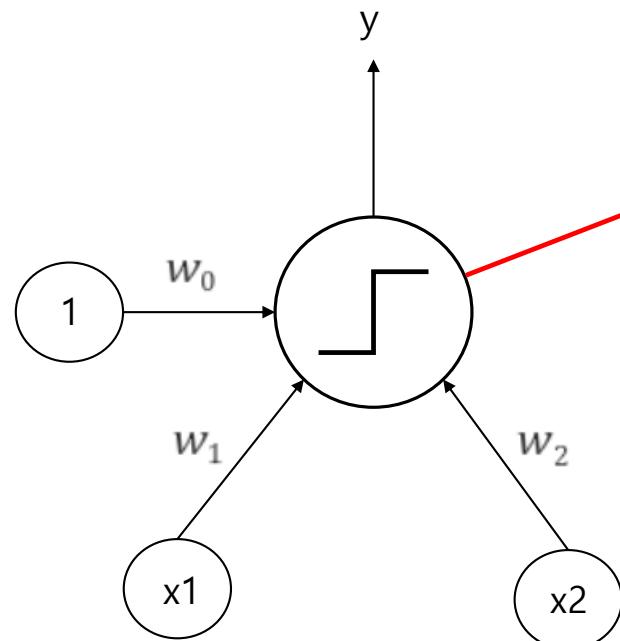


# *Game AI*

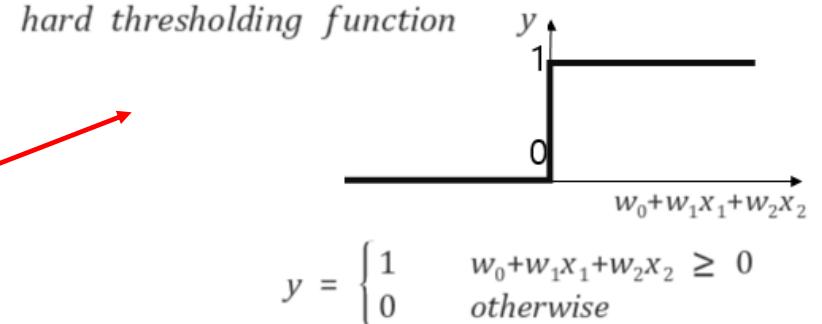


## **2. Perceptron**

# Single Layer Perceptron



*hard thresholding function*



$$y = \begin{cases} 1 & w_0 + w_1 x_1 + w_2 x_2 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

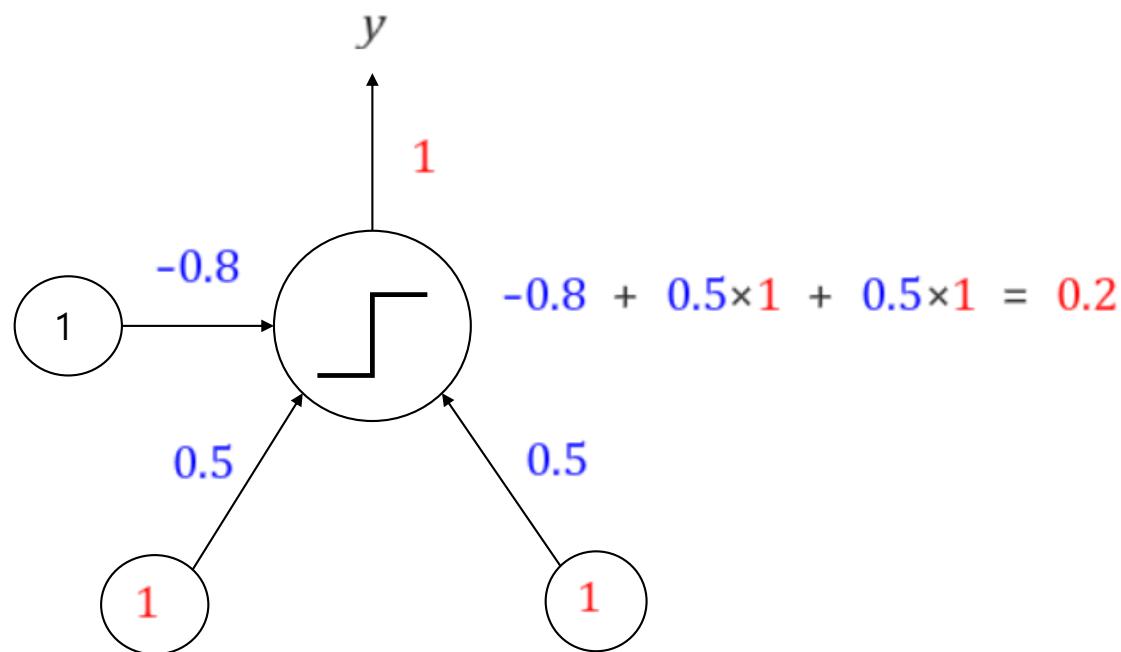
Inputs:  $x_1, x_2$

Output:  $y$

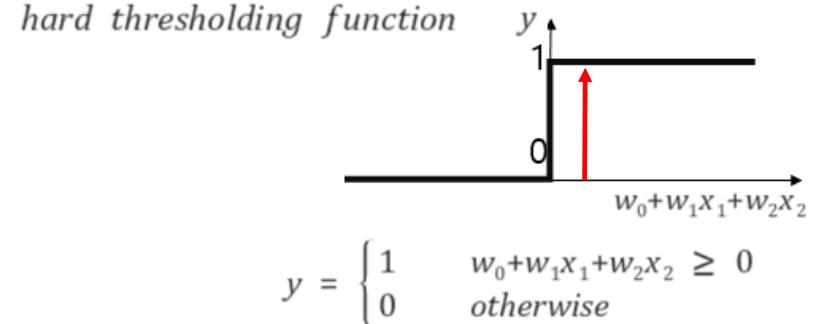
Weights:  $w_0, w_1, w_2$

Bias: 1

# Single Layer Perceptron



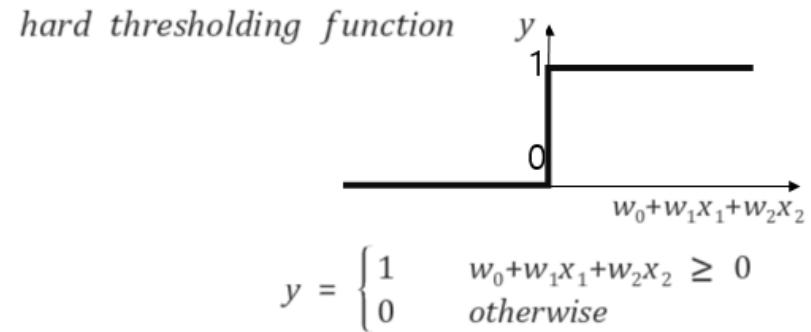
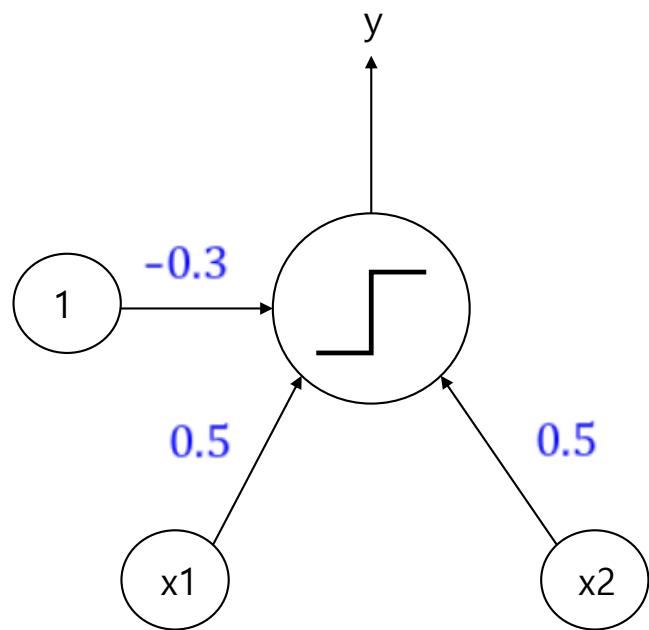
hard thresholding function



AND Gate

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

# Single Layer Perceptron

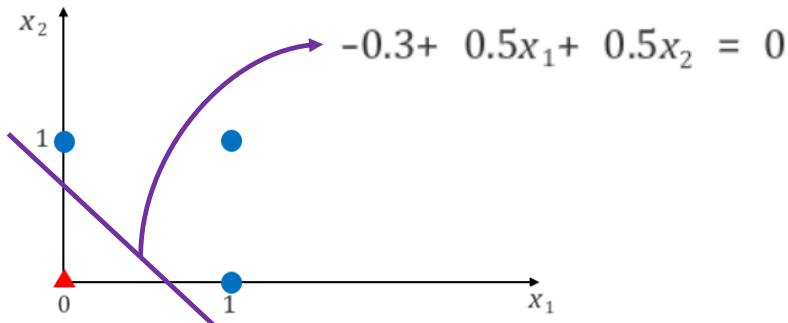
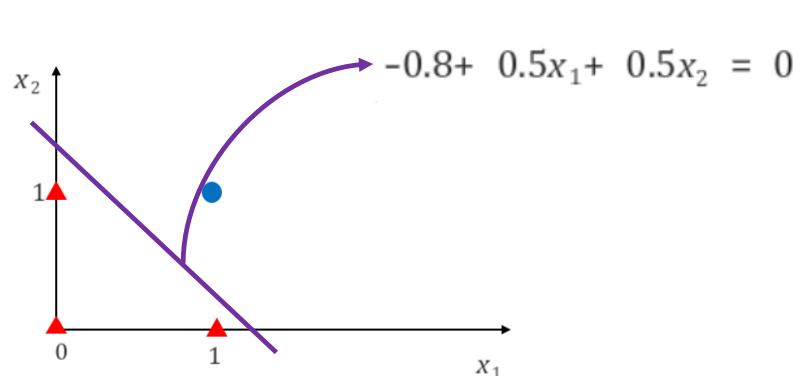


OR Gate

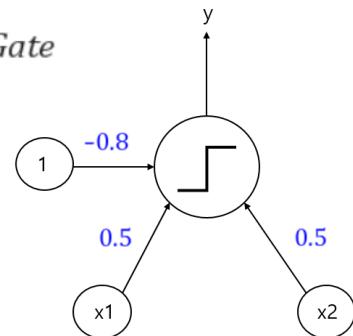
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1

# Single Layer Perceptron

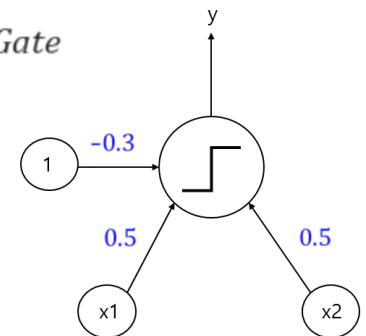
● : 1      ▲ : 0



AND Gate



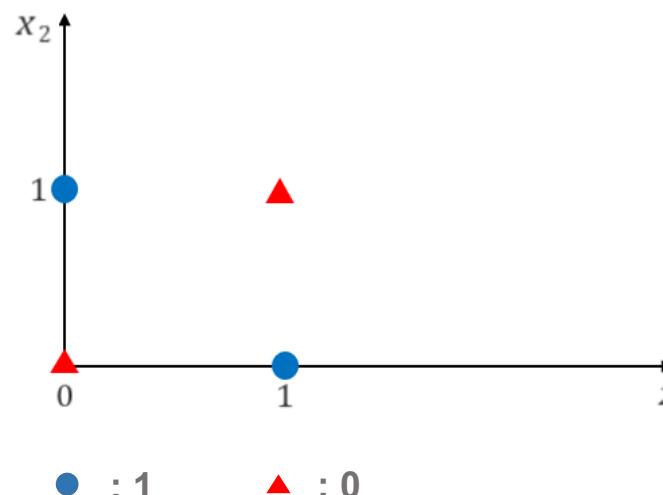
OR Gate



## XOR Gate

**Is it possible to solve XOR problem using a single layer perceptron?**

**No. Single layer perceptron can only solve linear problem. XOR problem is non-linear.**

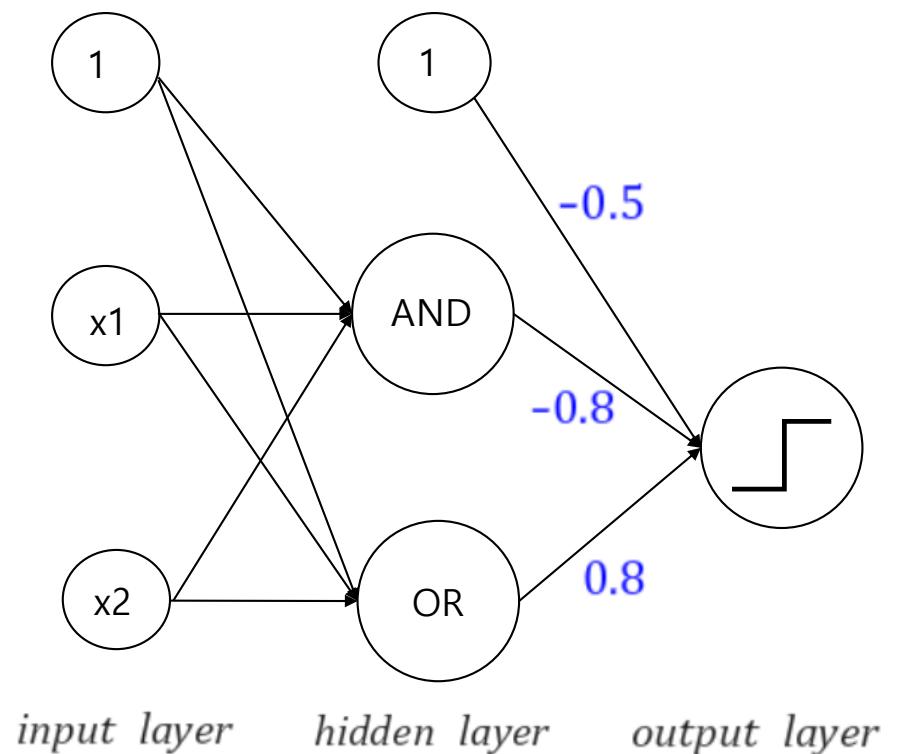
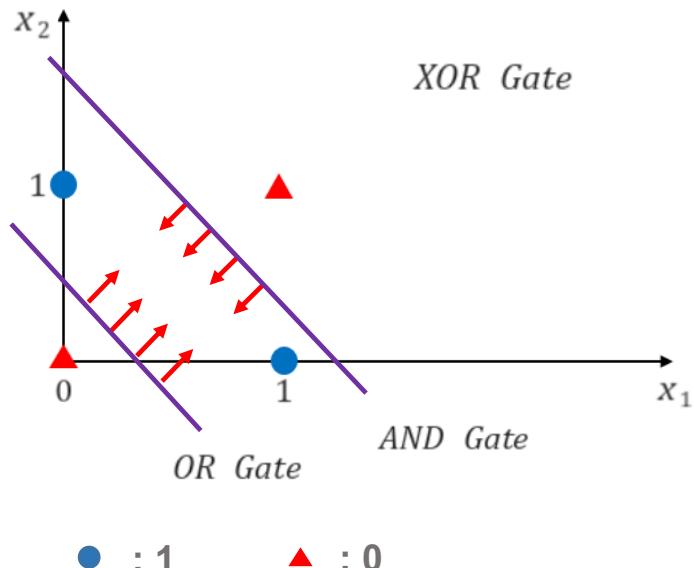


*XOR Gate*

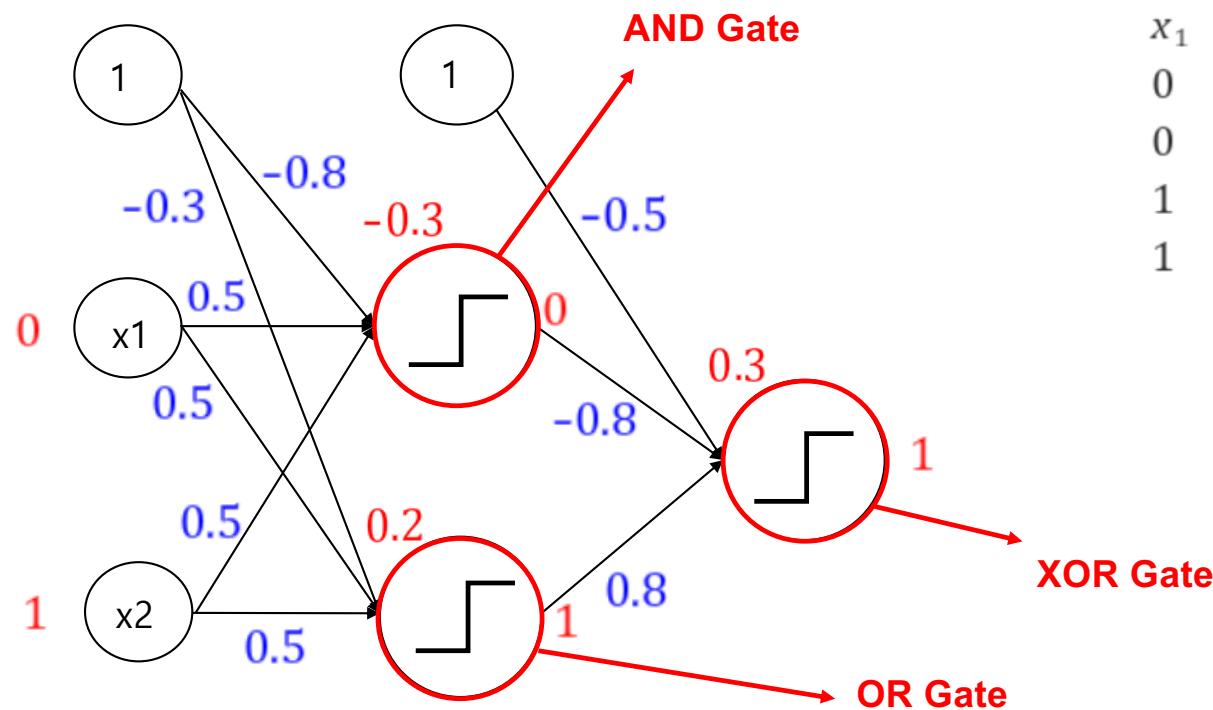
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

# Multi Layer Perceptron

But if we use 2 single layer perceptron, we can solve XOR problem. This model is called multi layer perceptron.



# Multi Layer Perceptron



*OR Gate*

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

*AND Gate*

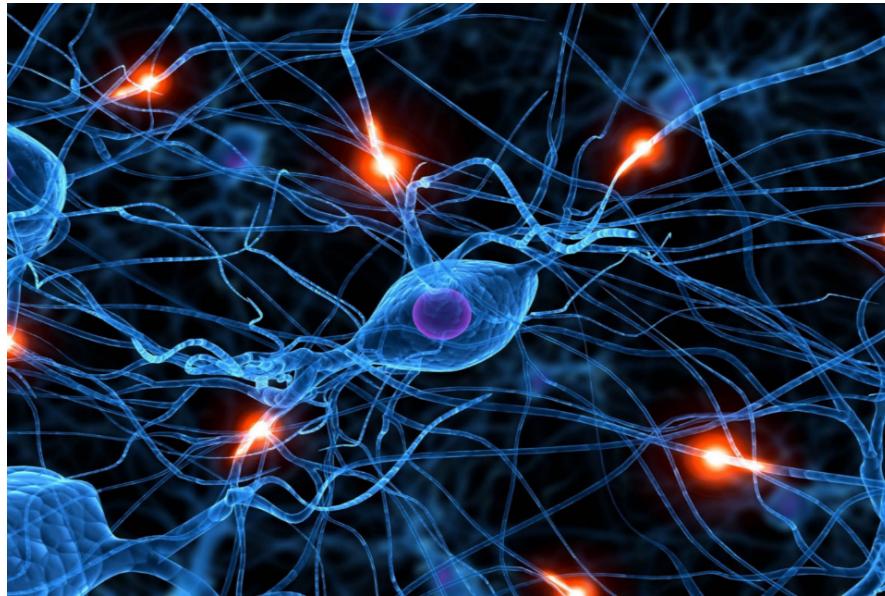
$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1

*XOR Gate*

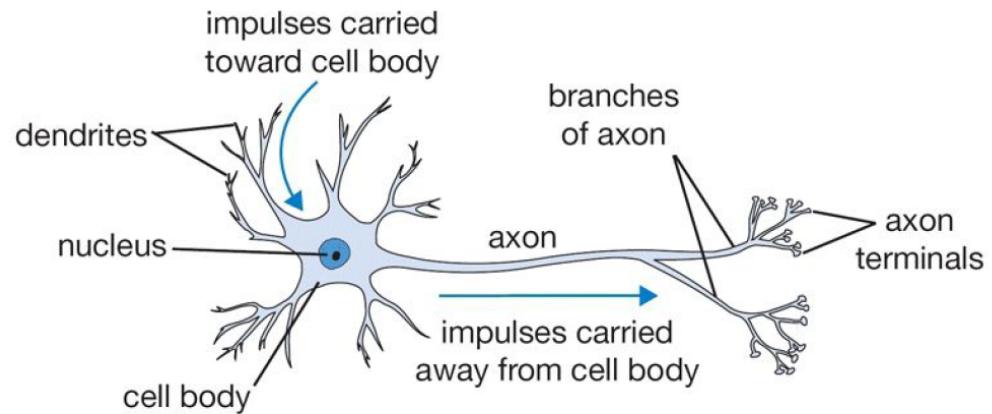
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

# **3. Neural Network**

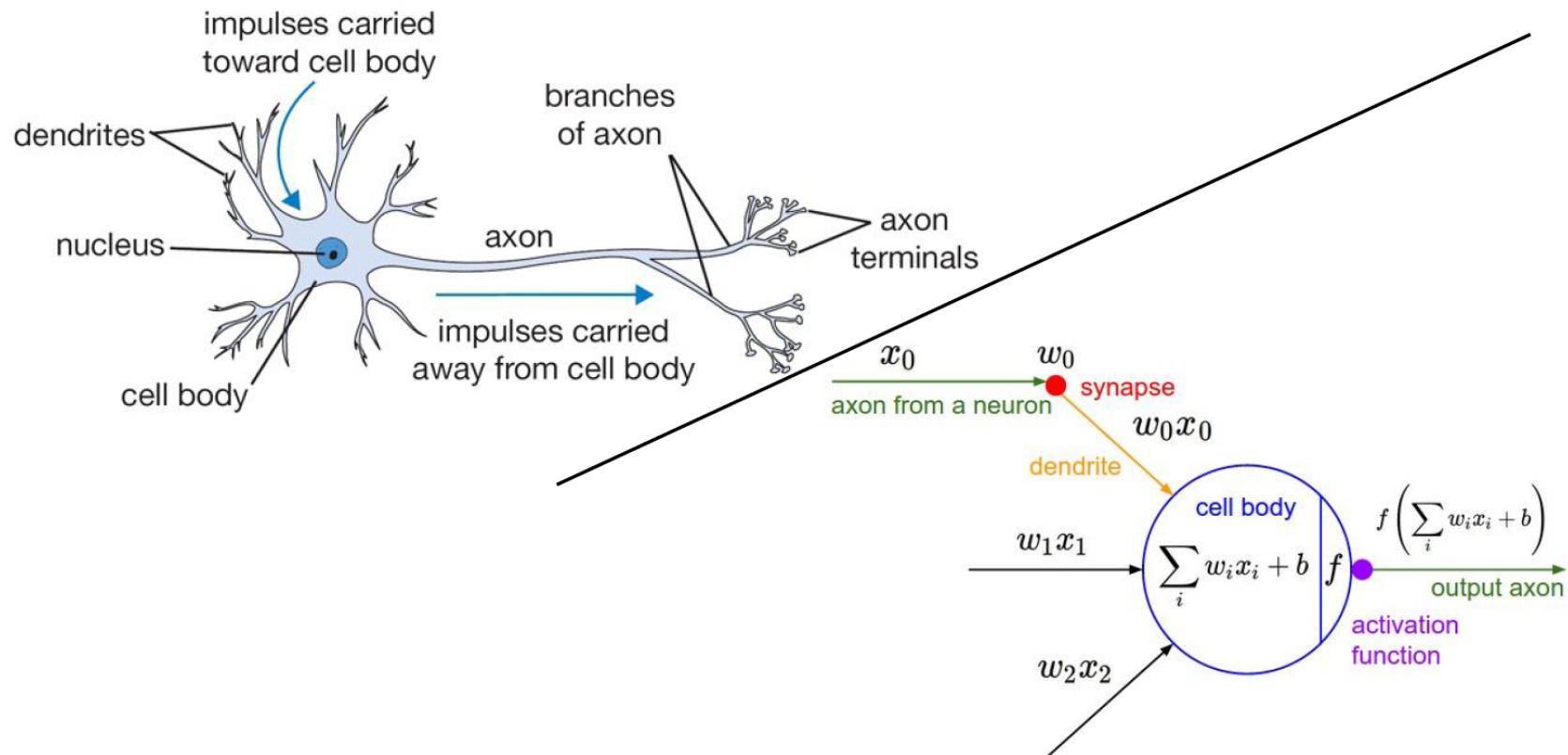
# *Human Brain and Neural Network*



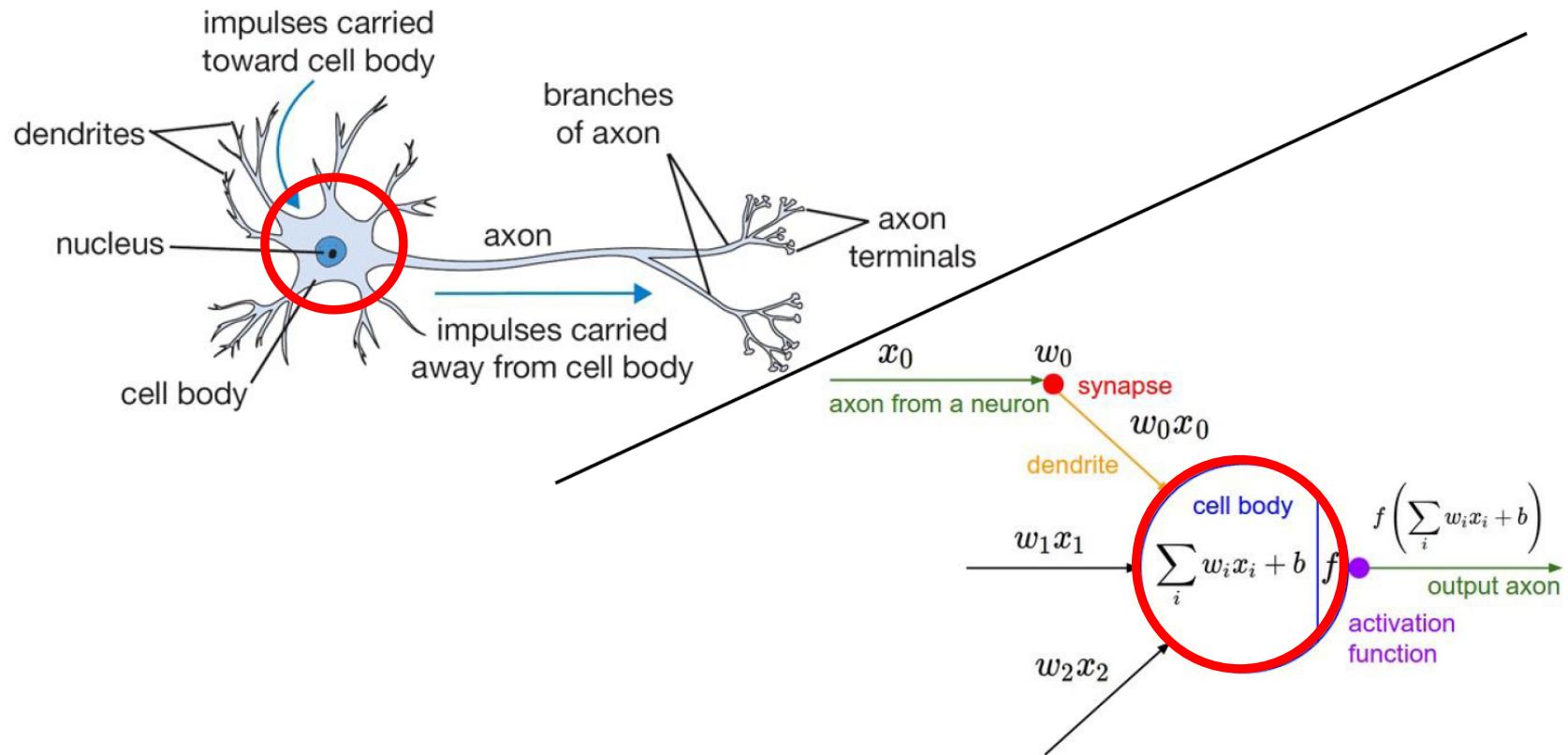
# *Human Brain and Neural Network*



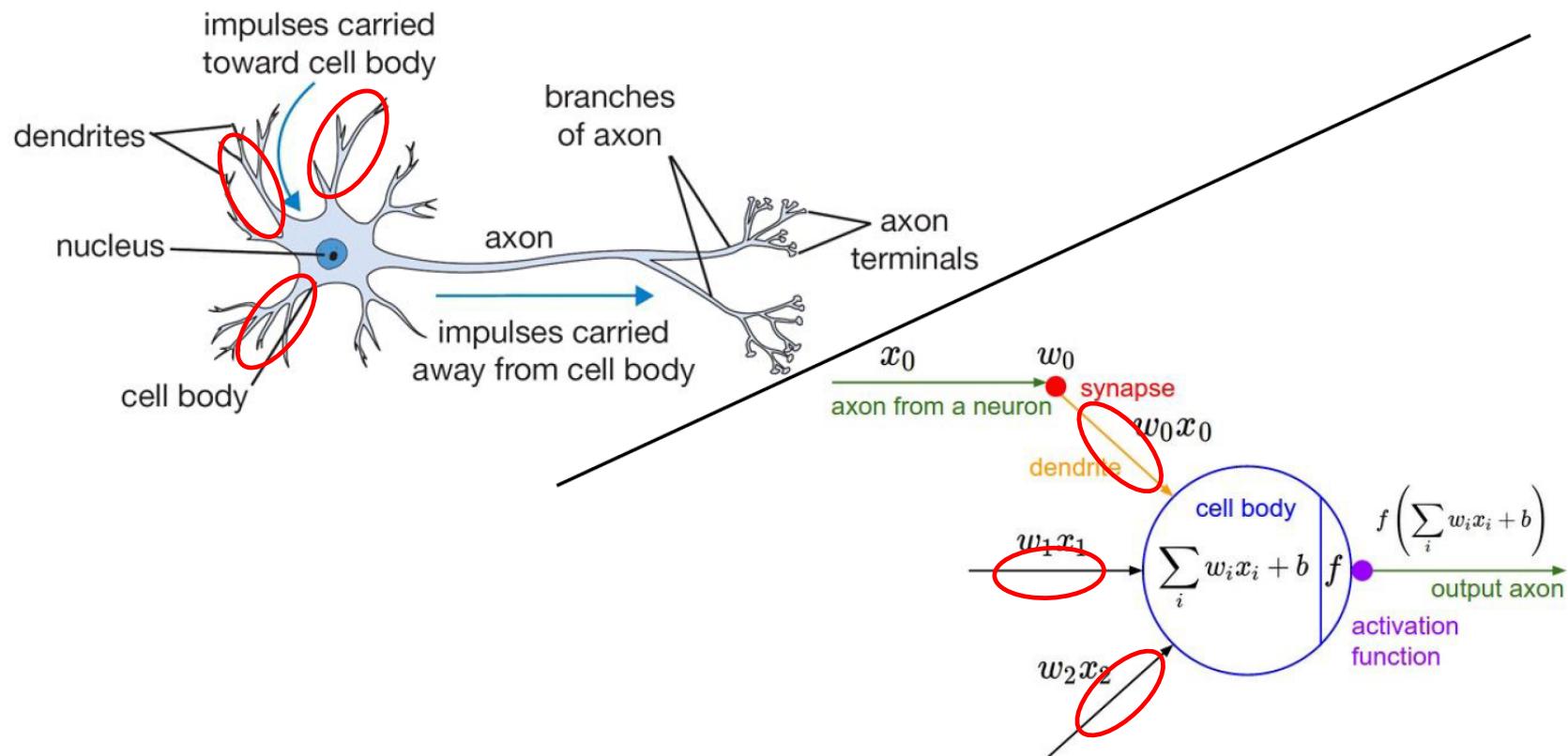
# Human Brain and Neural Network



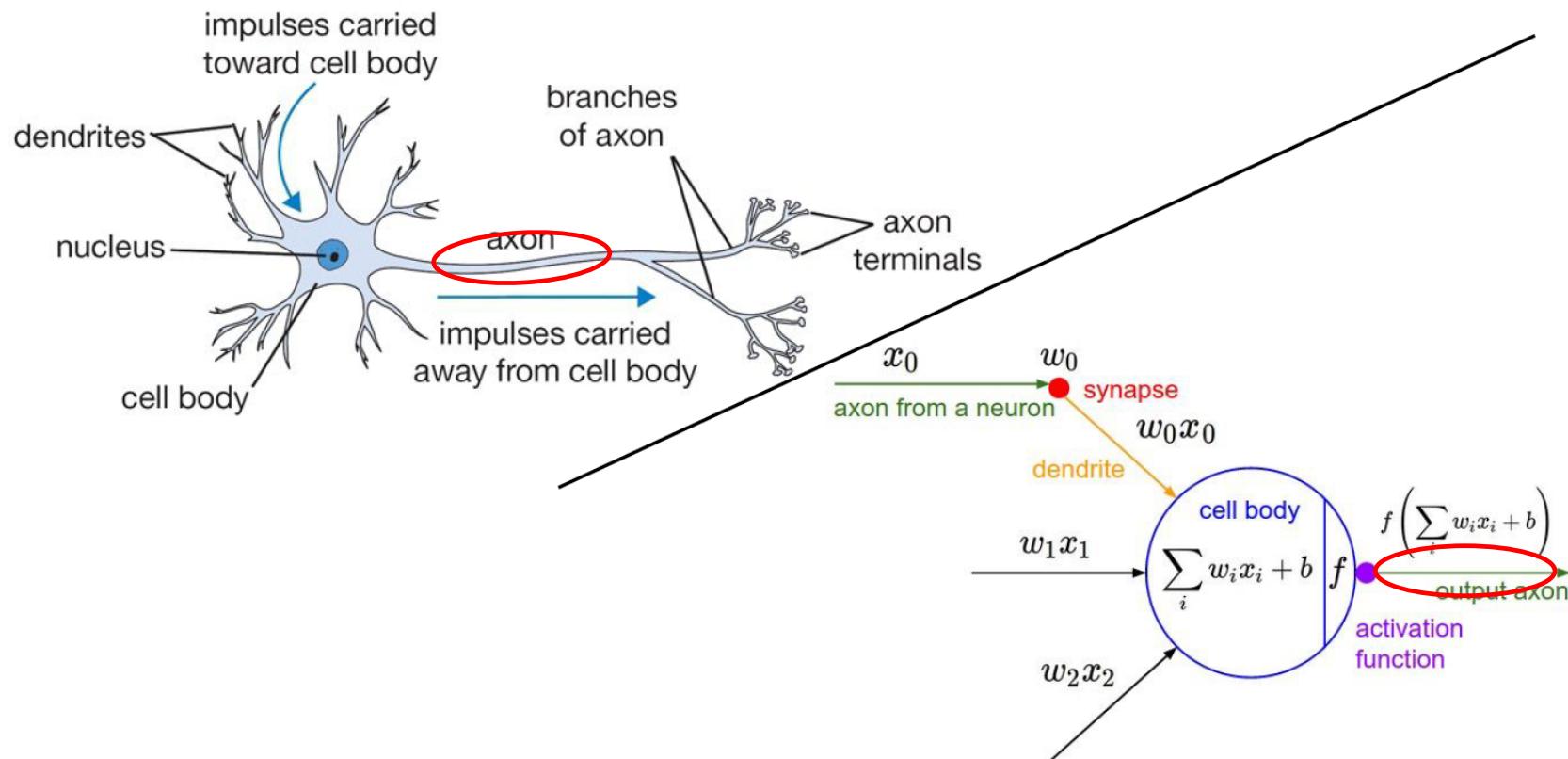
# Human Brain and Neural Network



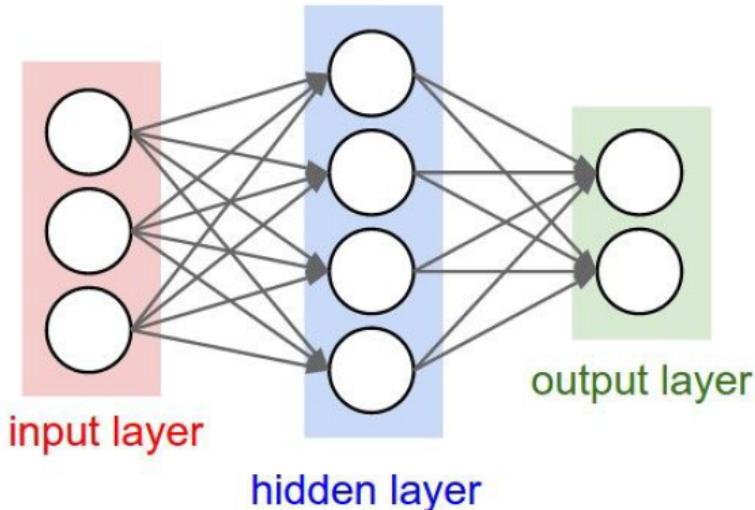
# Human Brain and Neural Network



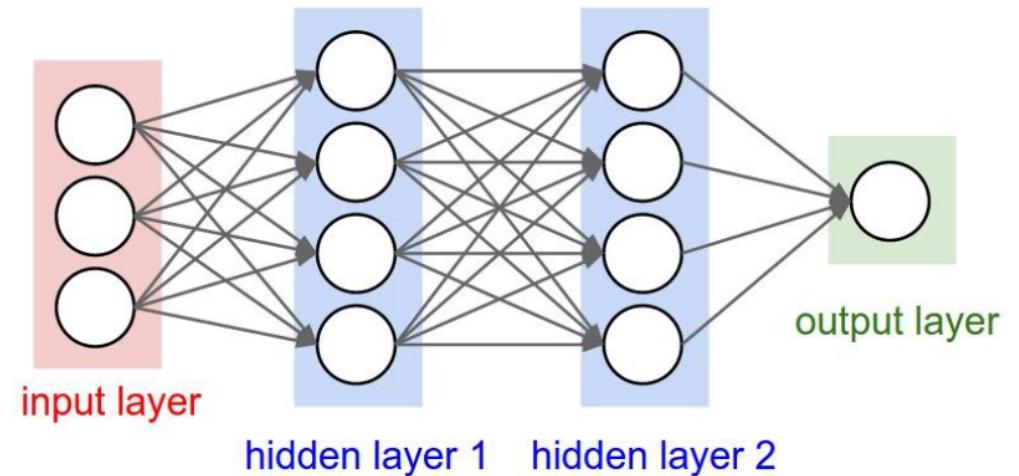
# Human Brain and Neural Network



# **Neural Network Architecture**

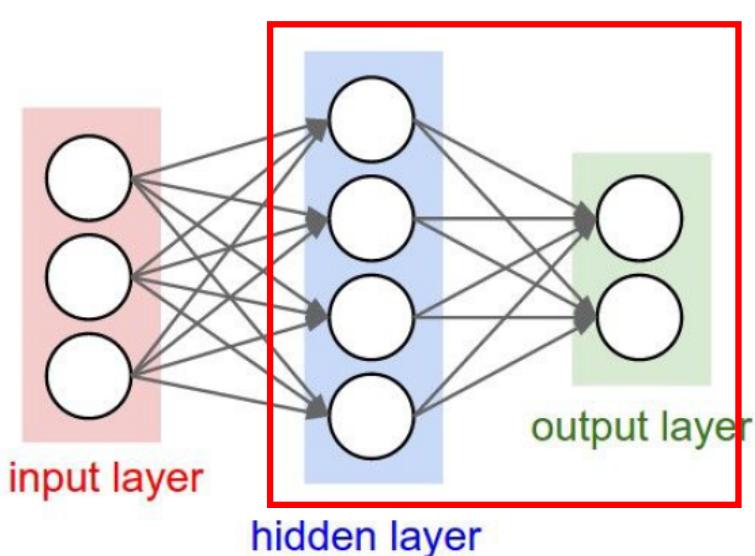


**“2-layer Neural Network”**  
or  
**“1-hidden-layer Neural Network”**

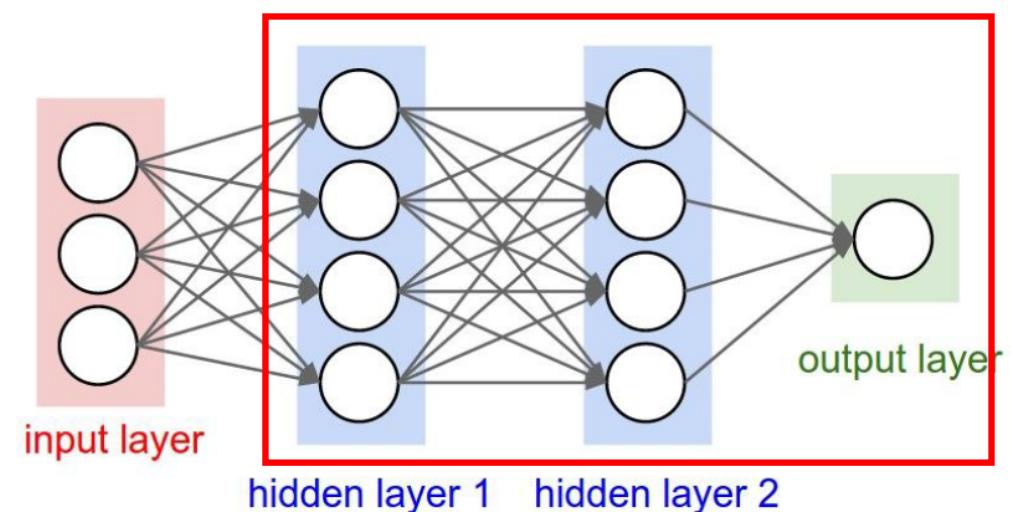


**“3-layer Neural Network”**  
or  
**“2-hidden-layer Neural Network”**

# **Neural Network Architecture**

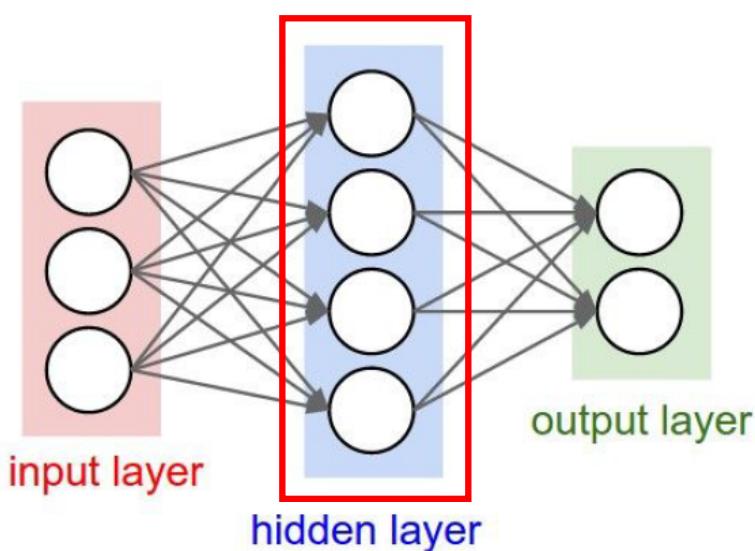


**“2-layer Neural Network”**  
or  
**“1-hidden-layer Neural Network”**



**“3-layer Neural Network”**  
or  
**“2-hidden-layer Neural Network”**

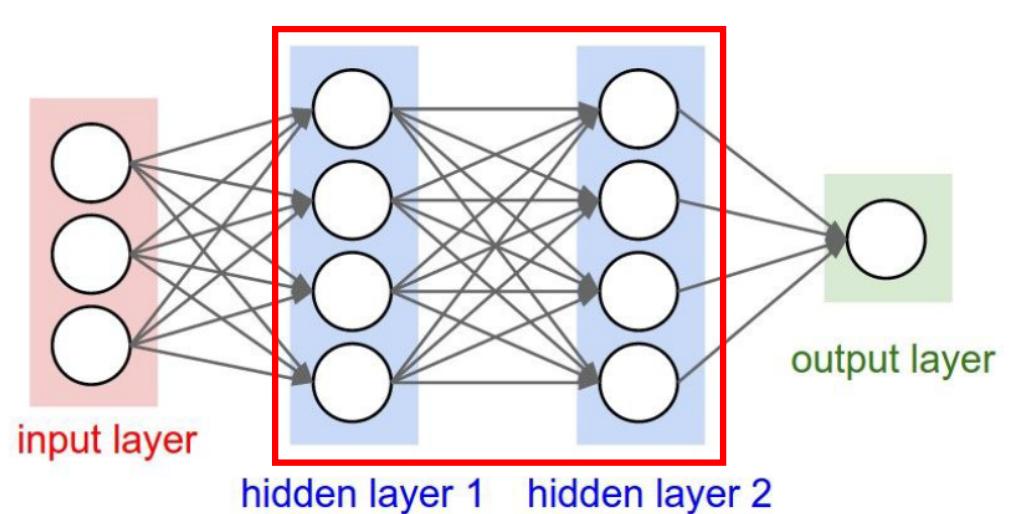
# **Neural Network Architecture**



**“2-layer Neural Network”**

**or**

**“1-hidden-layer Neural Network”**



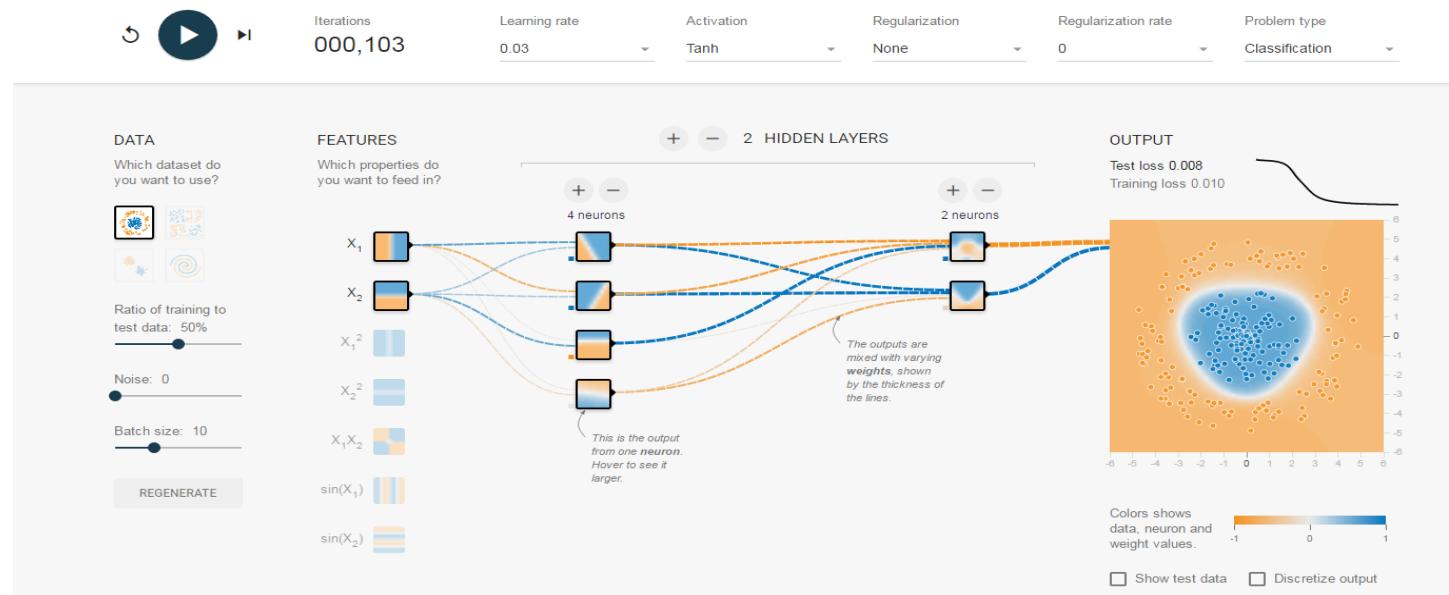
**“3-layer Neural Network”**

**or**

**“2-hidden-layer Neural Network”**

# Neural Network

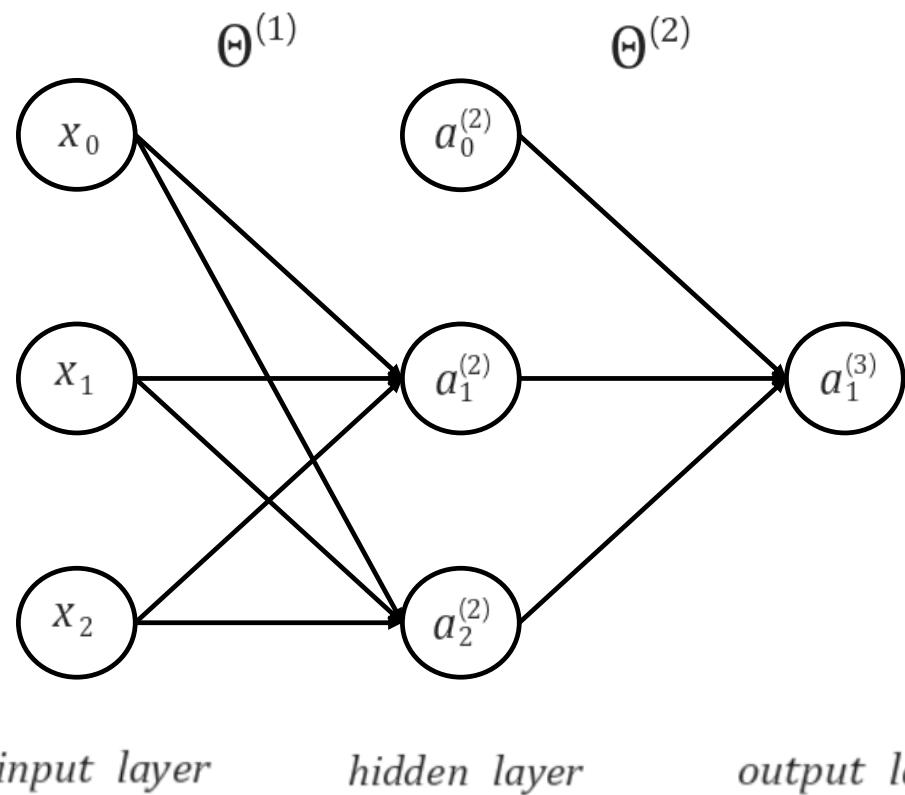
Tinker With a **Neural Network** Right Here in Your Browser.  
Don't Worry, You Can't Break It. We Promise.



A Neural Network Playground

# **4. Forward Propagation**

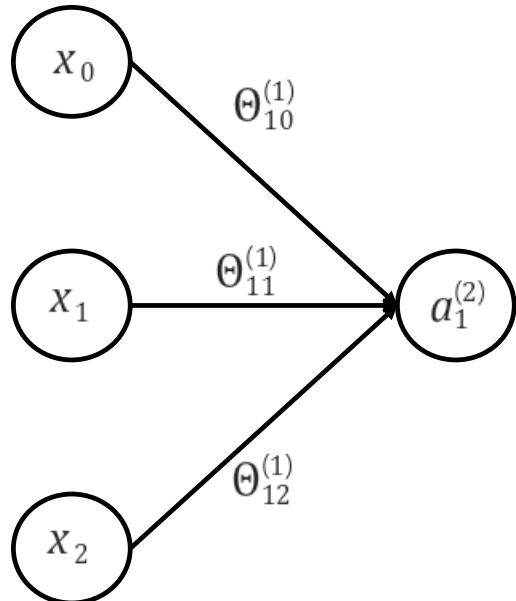
## Forward Propagation



$a_i^j$  : “activation” of i-th unit in j-th layer

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

## Forward Propagation

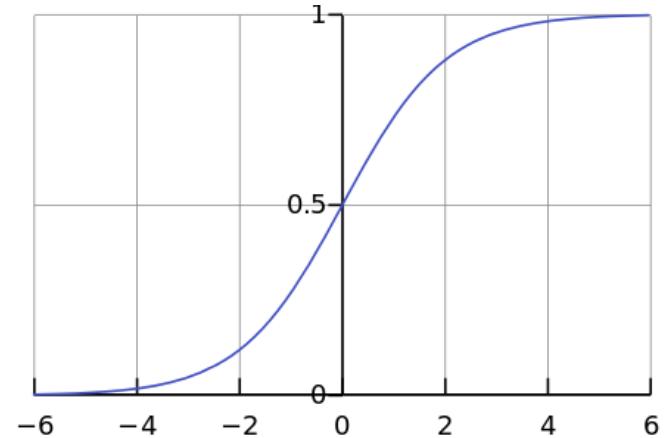


$$z_1^{(2)} = \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2$$

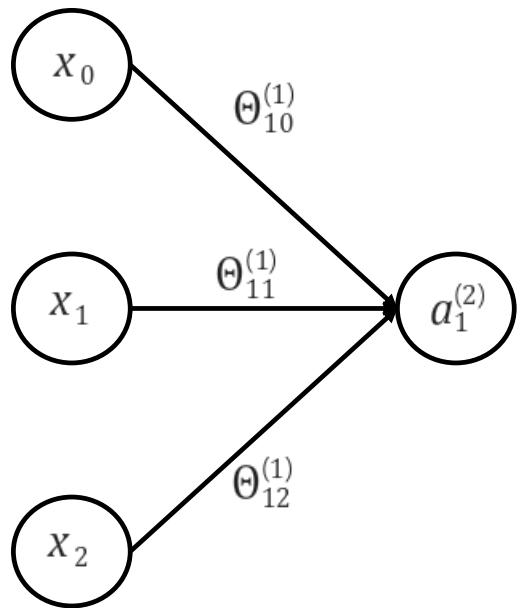
$$a_1^{(2)} = g(z_1^{(2)})$$

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

*logistic sigmoid*



## Forward Propagation



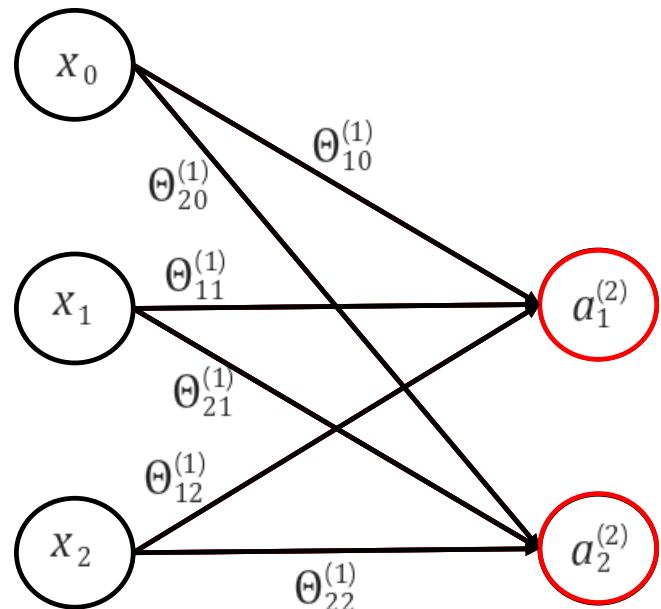
$$Z_1^{(2)} = \Theta_{10}^{(1)} X_0 + \Theta_{11}^{(1)} X_1 + \Theta_{12}^{(1)} X_2$$

$$= \begin{pmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} \end{pmatrix} \begin{pmatrix} X_0 \\ X_1 \\ X_2 \end{pmatrix}$$

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

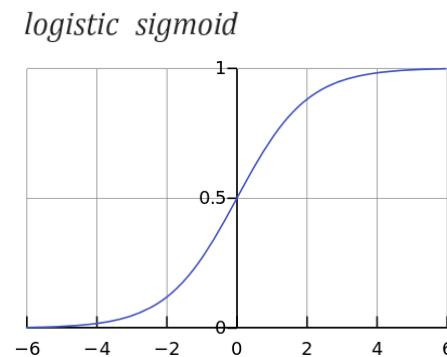
$$g(x) = \frac{1}{1 + e^{-x}}$$

## Forward Propagation

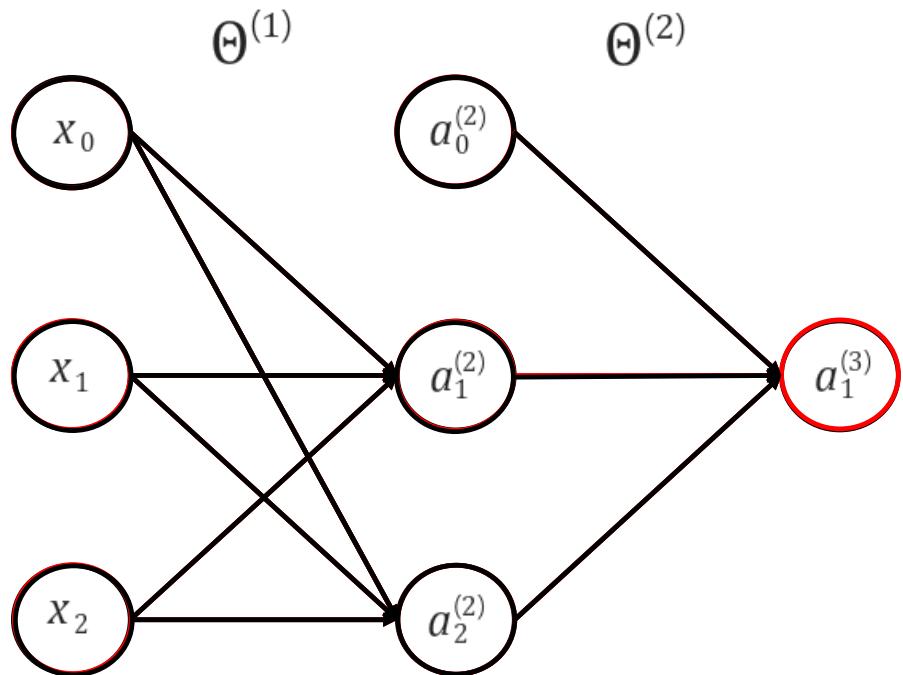


$$\begin{pmatrix} Z_1^{(2)} \\ Z_2^{(2)} \end{pmatrix} = \begin{pmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} \end{pmatrix} \begin{pmatrix} X_0 \\ X_1 \\ X_2 \end{pmatrix}$$

$$\begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \begin{pmatrix} g(Z_1^{(2)}) \\ g(Z_2^{(2)}) \end{pmatrix}$$



## Forward Propagation



$$\begin{pmatrix} Z_1^{(2)} \\ Z_2^{(2)} \end{pmatrix} = \begin{pmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} \end{pmatrix} \begin{pmatrix} X_0 \\ X_1 \\ X_2 \end{pmatrix}$$

$$\begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \begin{pmatrix} g(Z_1^{(2)}) \\ g(Z_2^{(2)}) \end{pmatrix}$$

$$Z_1^{(3)} = \begin{pmatrix} \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} \end{pmatrix} \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \end{pmatrix}$$

$$a_1^{(3)} = g(Z_1^{(3)})$$

## MNIST Example



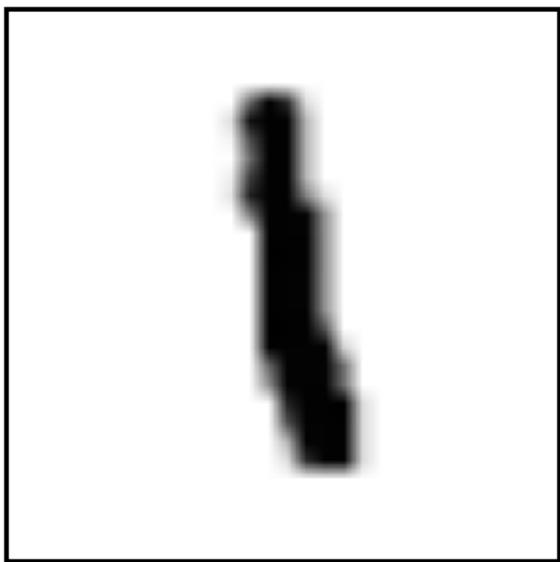
0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9

**MNIST dataset of handwritten digits for 0 to 9.**

**It has 55,000 examples for training and 10,000 examples for testing.**

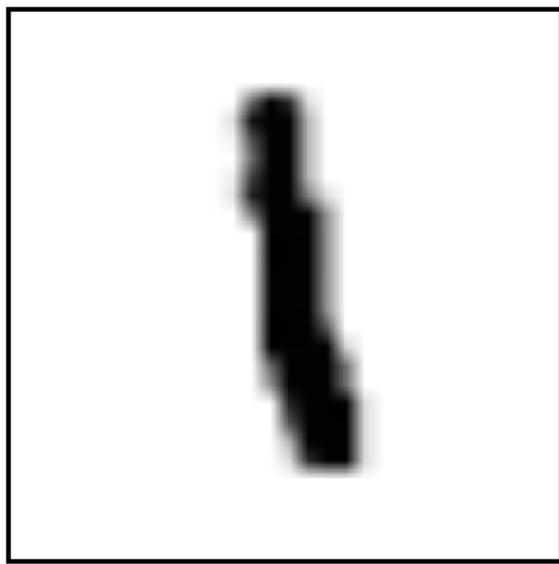
**The digits have been size-normalized and centered in a fixed-size image, so each sample is represented as a 2D matrix of size 28x28 with values from 0 to 1.**

## **MNIST Example**



2

# MNIST Example



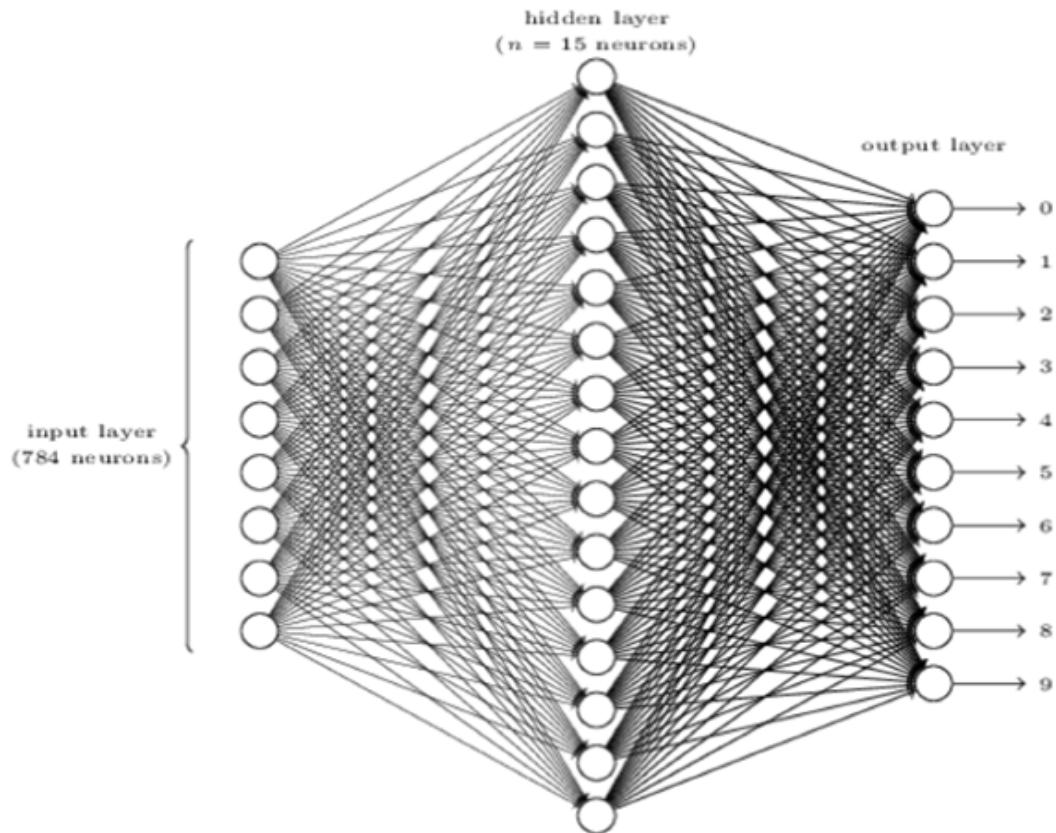
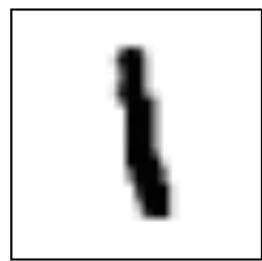
$\approx$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 1 & .1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

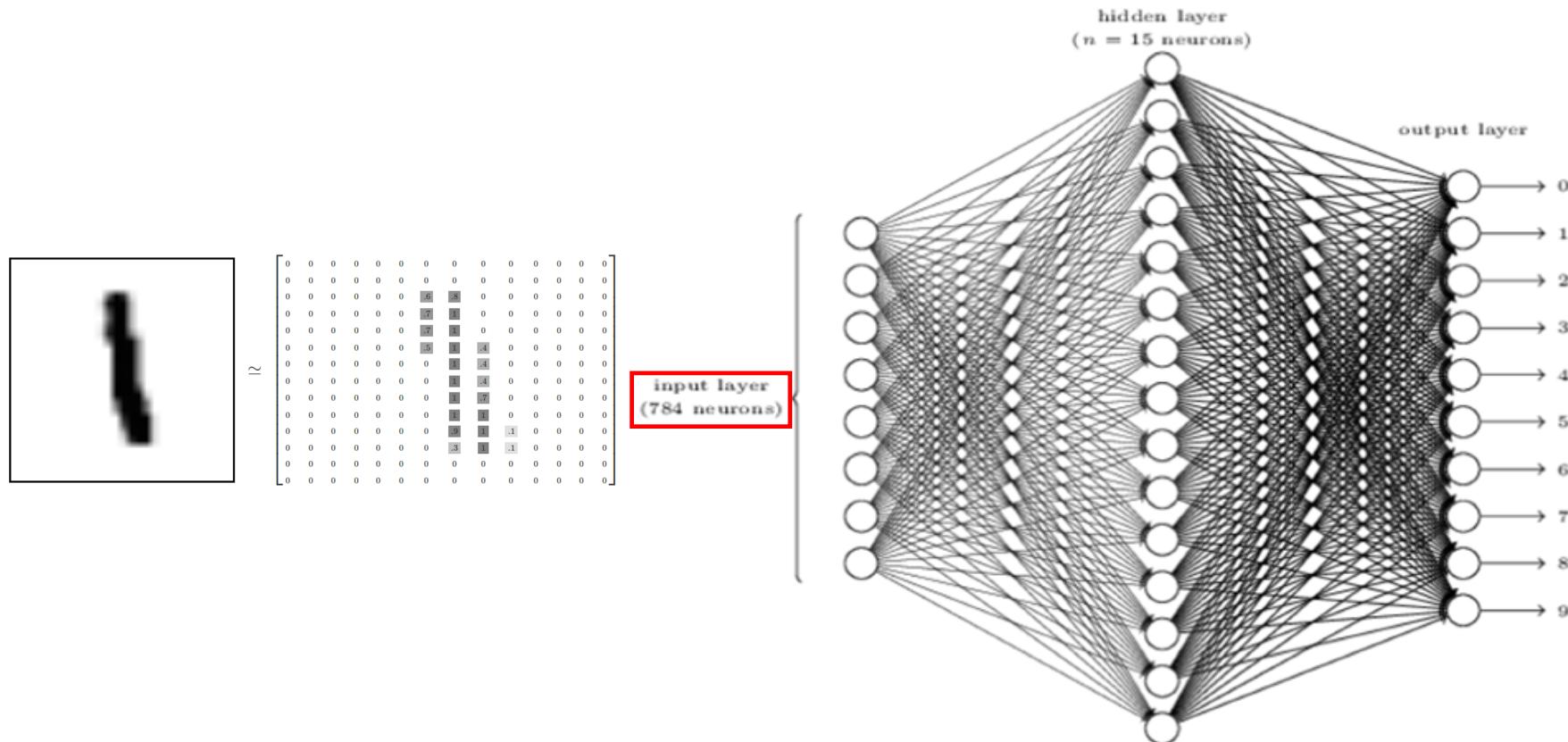
28

28

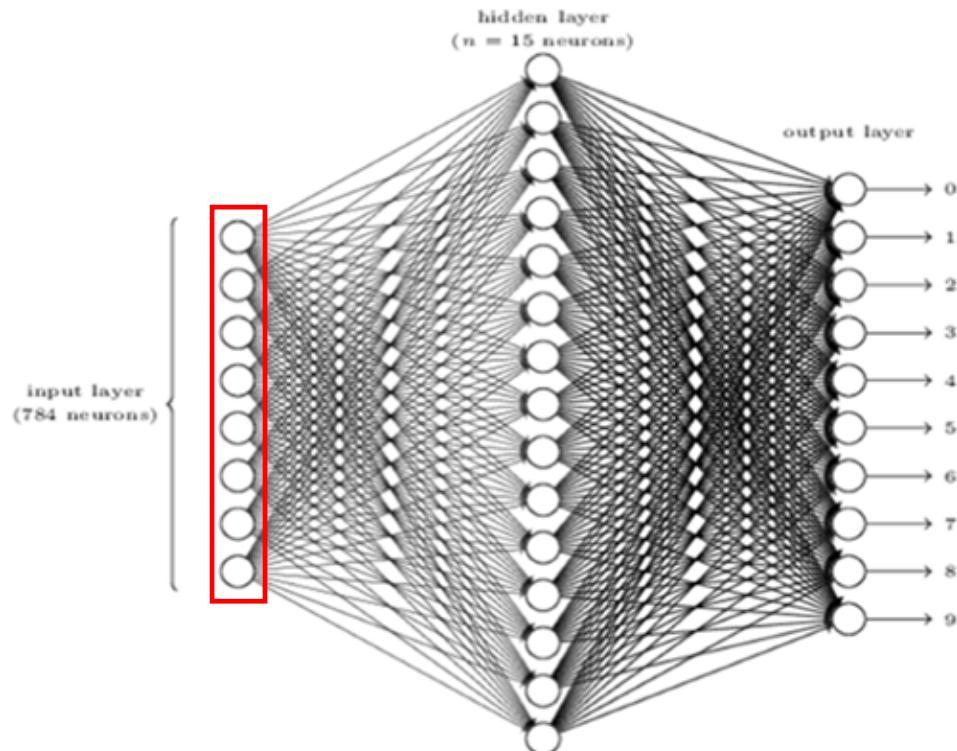
## **MNIST Example**



## **MNIST Example**

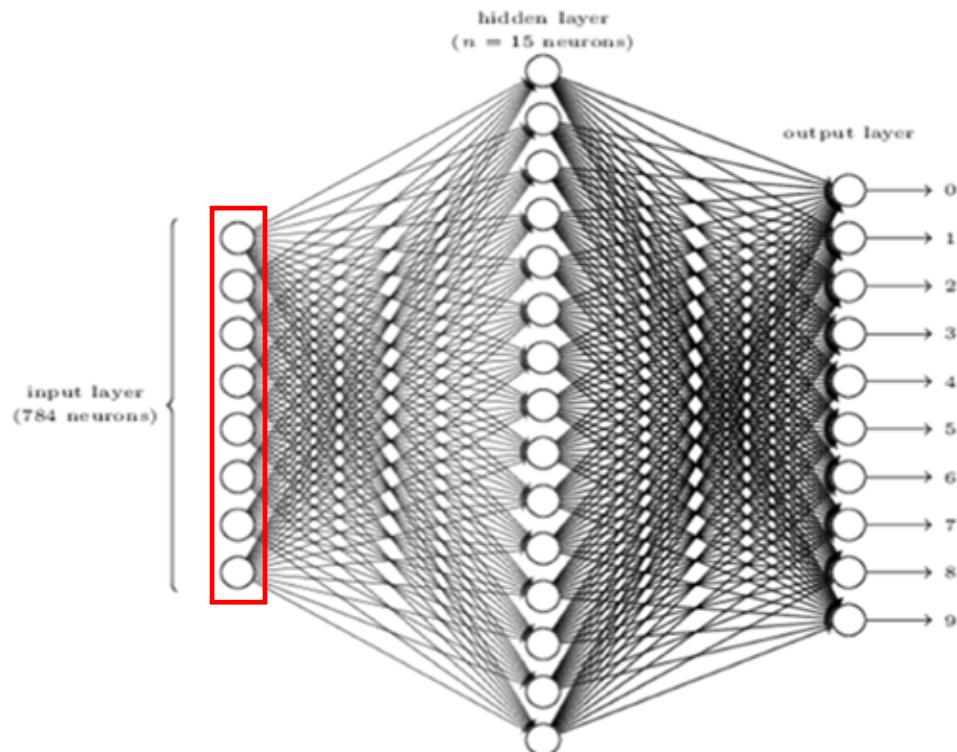


# MNIST Example



```
x: 784 x 1  
w1: 15 x (784 + 1)  
w2: 10 x (15 + 1)  
  
import numpy as np  
  
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
def neural_network(x, w1, w2):  
    a1 = np.concatenate([x, [[1]]], 0)  
    z2 = np.dot(w1, a1)  
    a2 = sigmoid(z2)  
    a2 = np.concatenate([a2, [[1]]], 0)  
    z3 = np.dot(w2, a2)  
    a3 = sigmoid(z3)  
    return a3
```

# MNIST Example



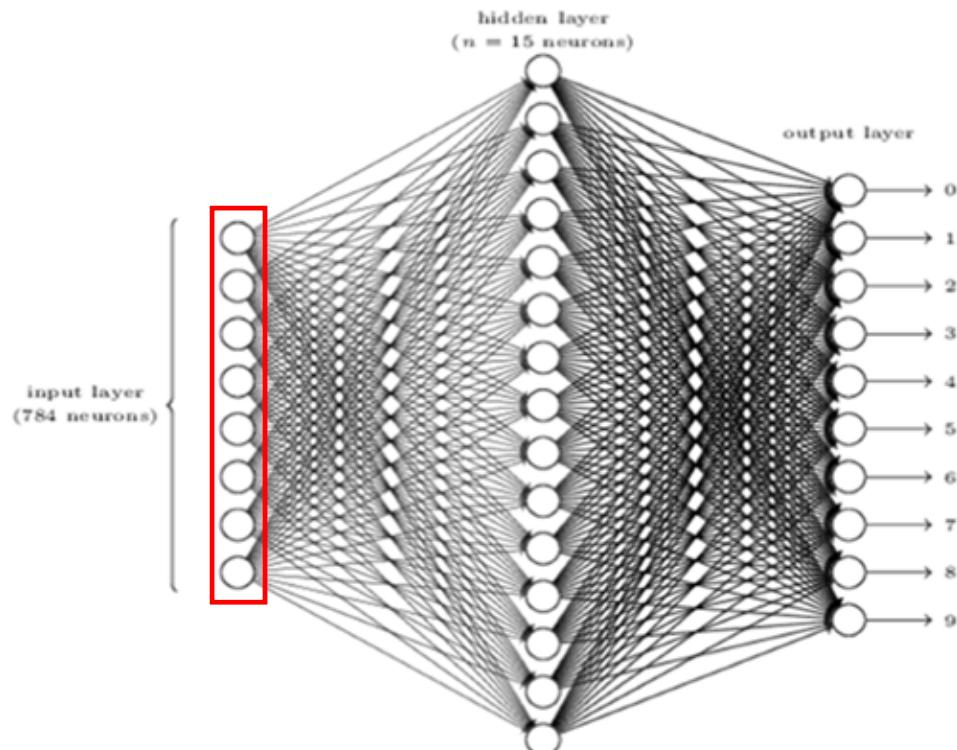
```
x: 784 x 1
w1: 15 x (784 + 1)
w2: 10 x (15 + 1)

import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

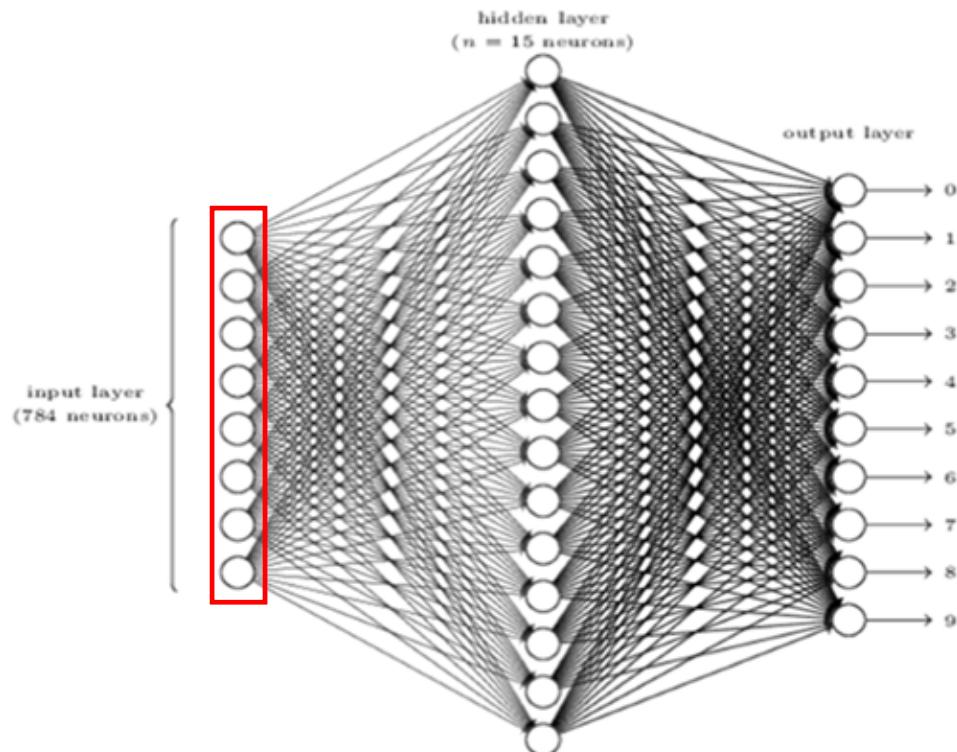
def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

# MNIST Example



```
x: 784 x 1  
w1: 15 x (784 + 1)  
w2: 10 x (15 + 1)  
  
import numpy as np  
  
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
def neural_network(x, w1, w2):  
    a1 = np.concatenate([x, [[1]]], 0)  
    z2 = np.dot(w1, a1)  
    a2 = sigmoid(z2)  
    a2 = np.concatenate([a2, [[1]]], 0)  
    z3 = np.dot(w2, a2)  
    a3 = sigmoid(z3)  
    return a3
```

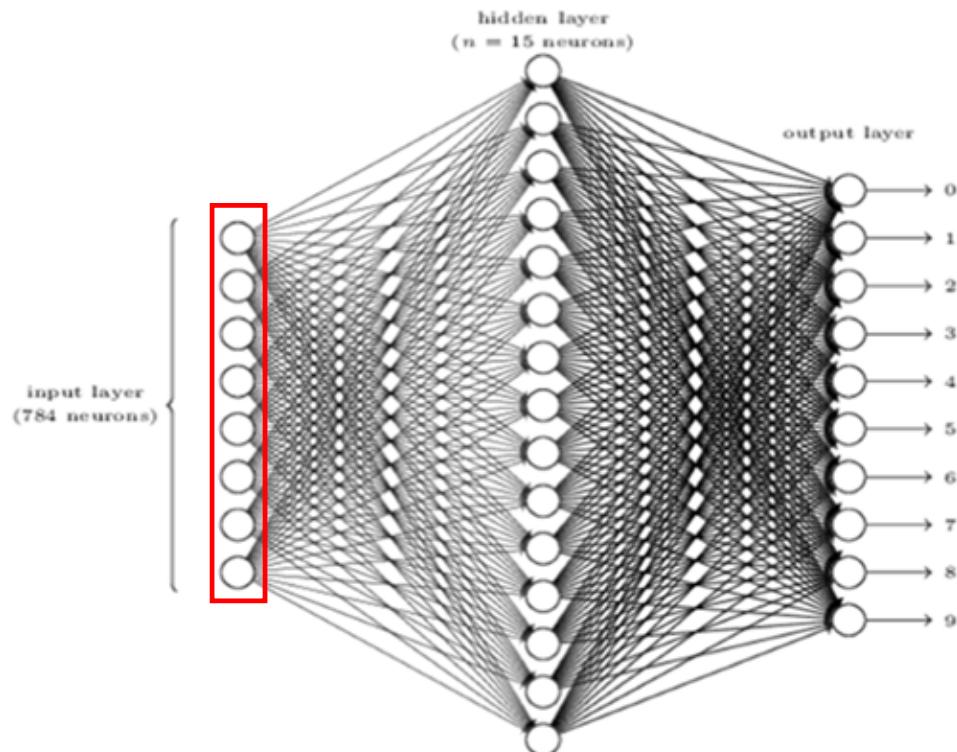
# MNIST Example



```
x: 784 x 1  
w1: 15 x (784 + 1)  
w2: 10 x (15 + 1)
```

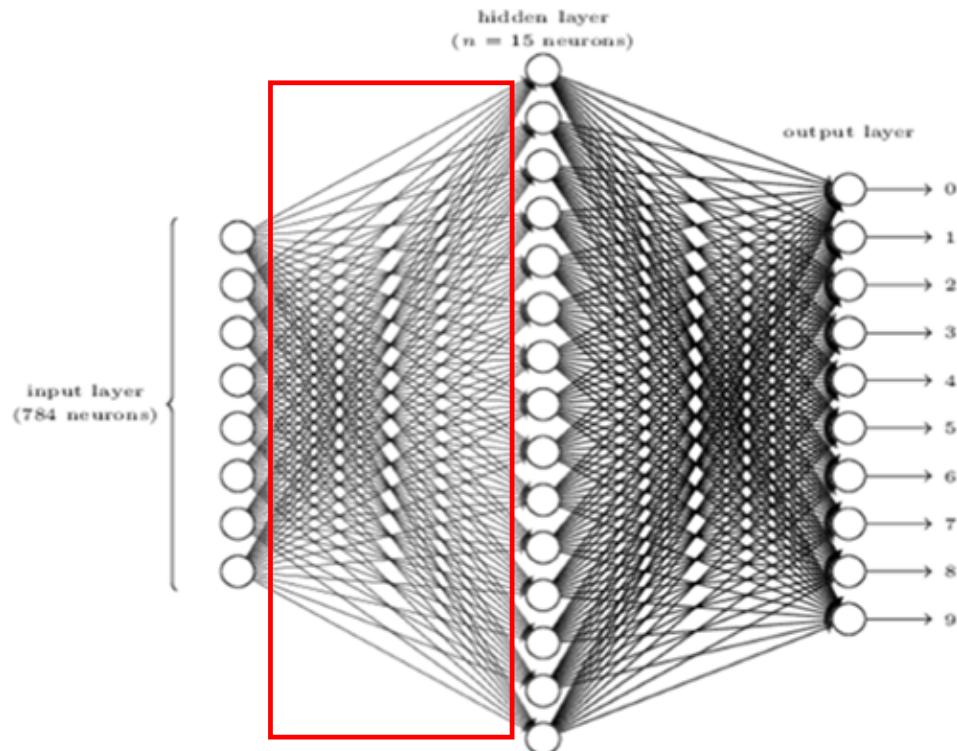
```
import numpy as np  
  
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
def neural_network(x, w1, w2):  
    a1 = np.concatenate([x, [[1]]], 0) 785 x 1  
    z2 = np.dot(w1, a1)  
    a2 = sigmoid(z2)  
    a2 = np.concatenate([a2, [[1]]], 0)  
    z3 = np.dot(w2, a2)  
    a3 = sigmoid(z3)  
    return a3
```

# MNIST Example



```
x: 784 x 1  
w1: 15 x (784 + 1)  
w2: 10 x (15 + 1)  
  
import numpy as np  
  
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
def neural_network(x, w1, w2):  
    a1 = np.concatenate([x, [[1]]], 0)  
    z2 = np.dot(w1, a1)  
    a2 = sigmoid(z2)  
    a2 = np.concatenate([a2, [[1]]], 0)  
    z3 = np.dot(w2, a2)  
    a3 = sigmoid(z3)  
    return a3
```

# MNIST Example



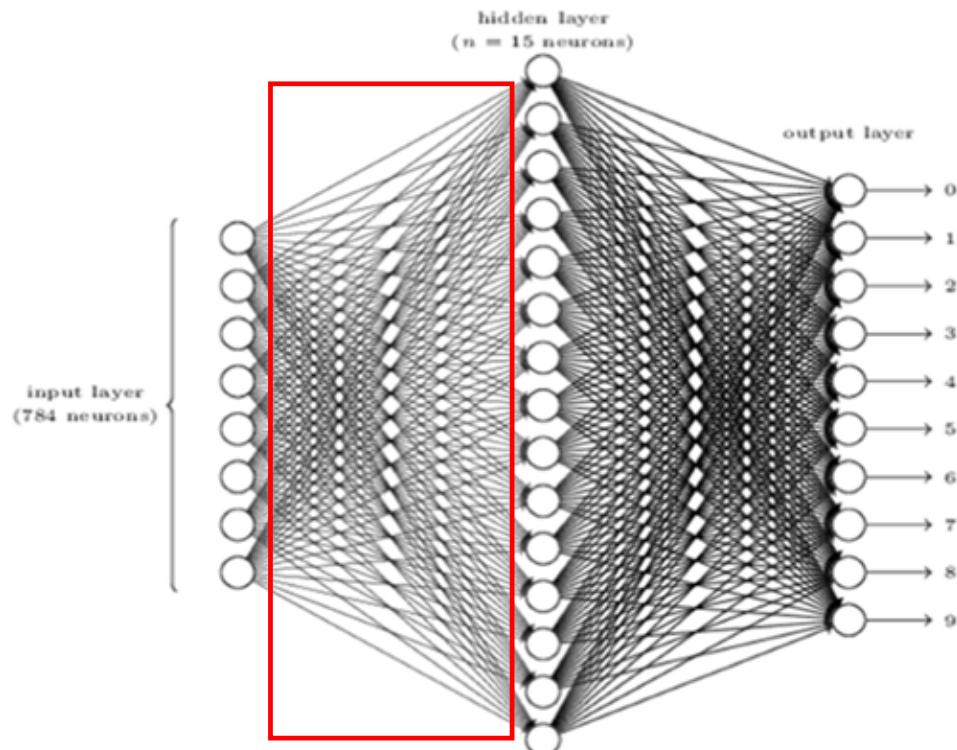
```
x: 784 x 1           a1: 785 x 1
w1: 15 x (784 + 1)
w2: 10 x (15 + 1)

import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

# MNIST Example



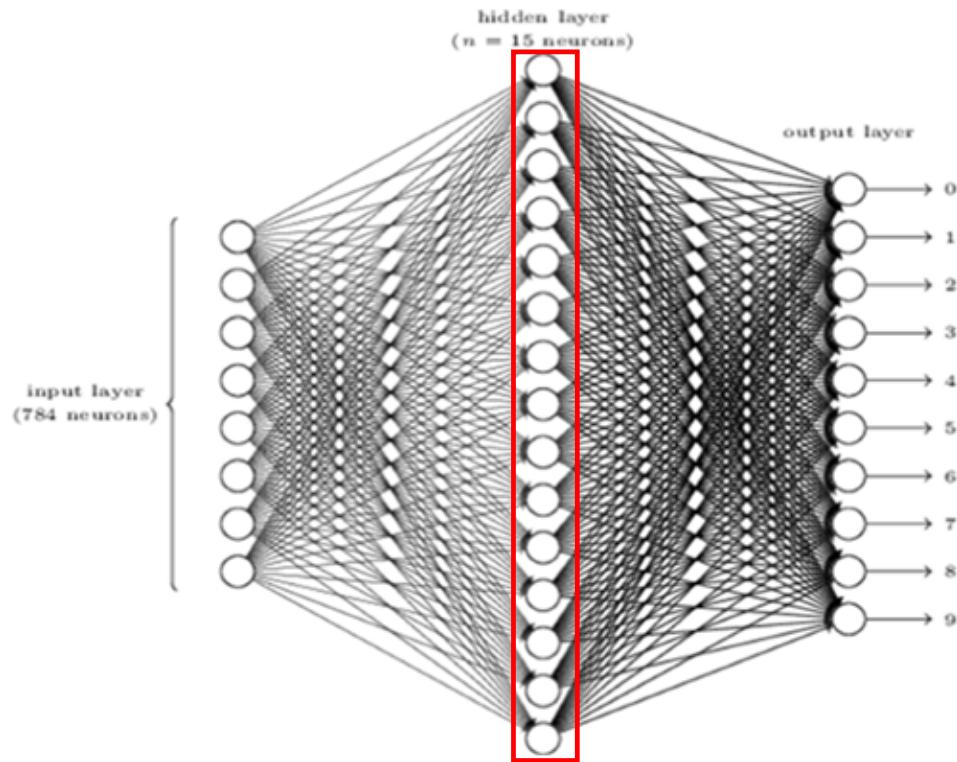
```
x: 784 x 1           a1: 785 x 1
w1: 15 x (784 + 1)  w2: 10 x (15 + 1)

import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

# MNIST Example



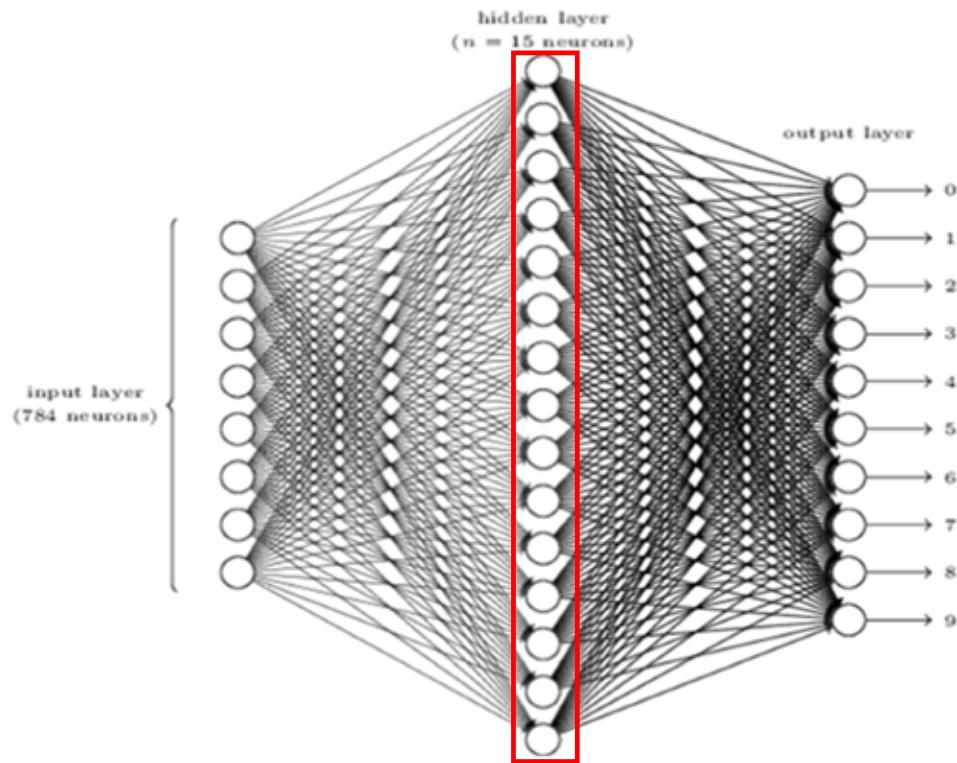
```
x: 784 x 1           a1: 785 x 1
w1: 15 x (784 + 1)
w2: 10 x (15 + 1)

import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

# MNIST Example



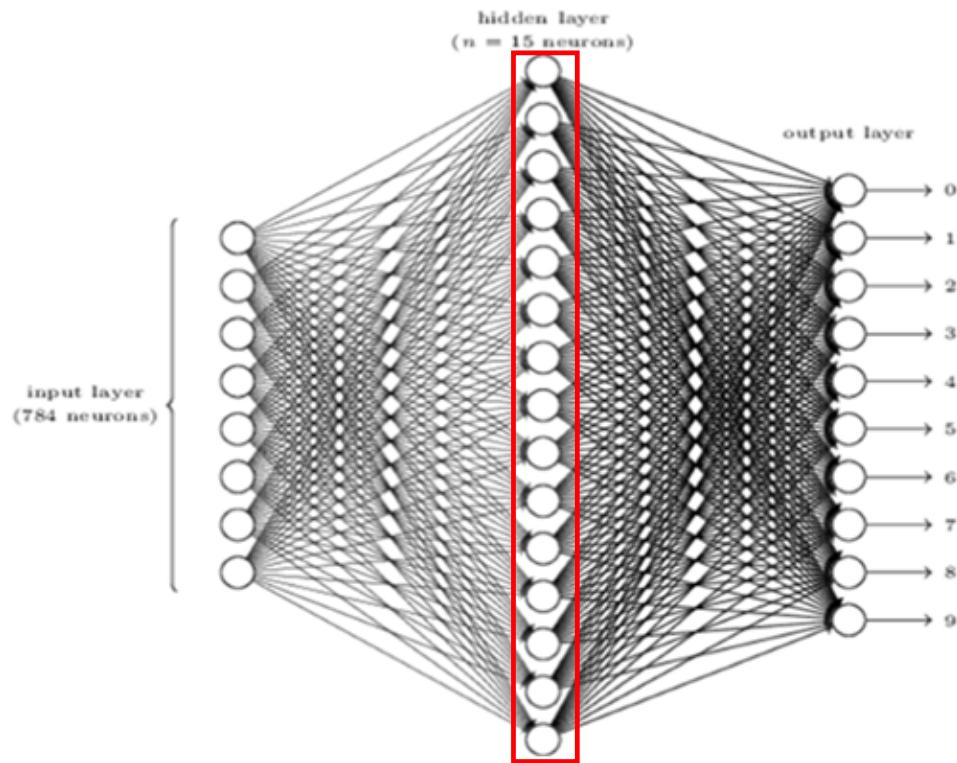
x: 784 x 1  
w1: 15 x (784 + 1)  
w2: 10 x (15 + 1)

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

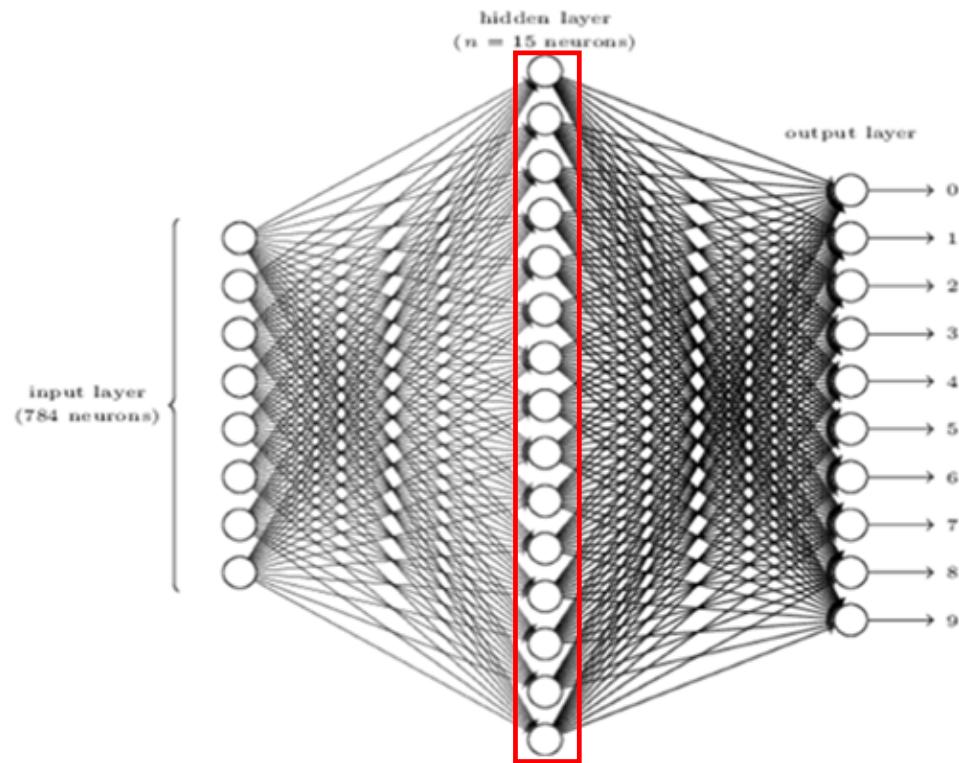
def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

# MNIST Example



```
x: 784 x 1  
w1: 15 x (784 + 1)  
w2: 10 x (15 + 1)  
  
import numpy as np  
  
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
def neural_network(x, w1, w2):  
    a1 = np.concatenate([x, [[1]]], 0)  
    z2 = np.dot(w1, a1) 15 x 1  
    a2 = sigmoid(z2)  
    a2 = np.concatenate([a2, [[1]]], 0)  
    z3 = np.dot(w2, a2)  
    a3 = sigmoid(z3)  
    return a3
```

# MNIST Example



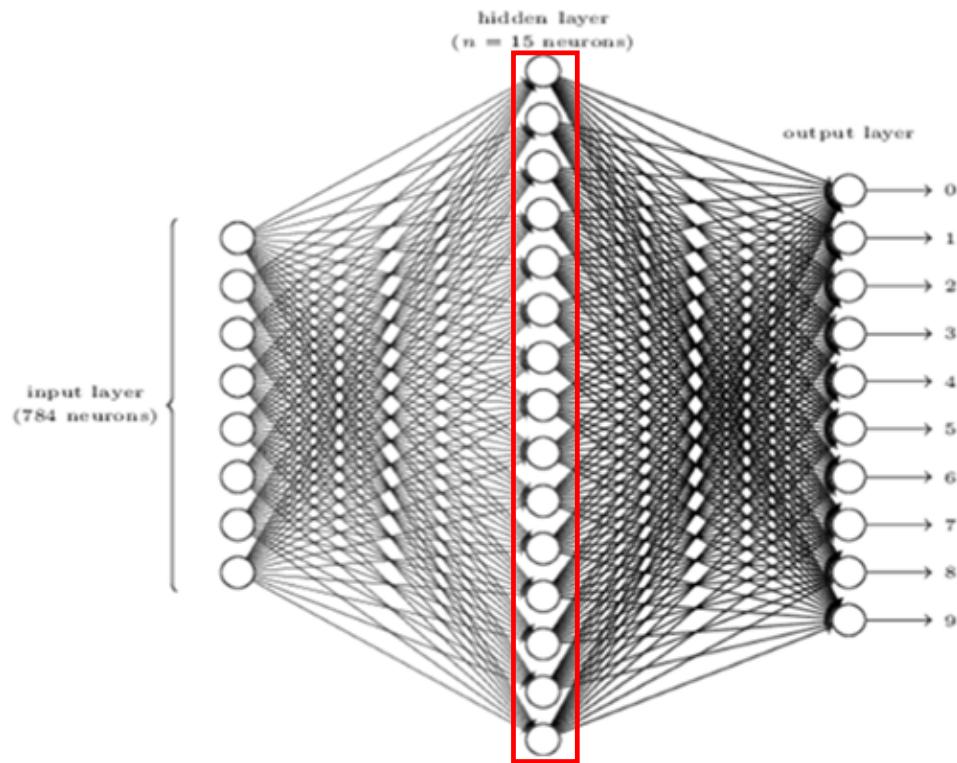
```
x: 784 x 1           a1: 785 x 1
w1: 15 x (784 + 1)
w2: 10 x (15 + 1)

import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

# MNIST Example



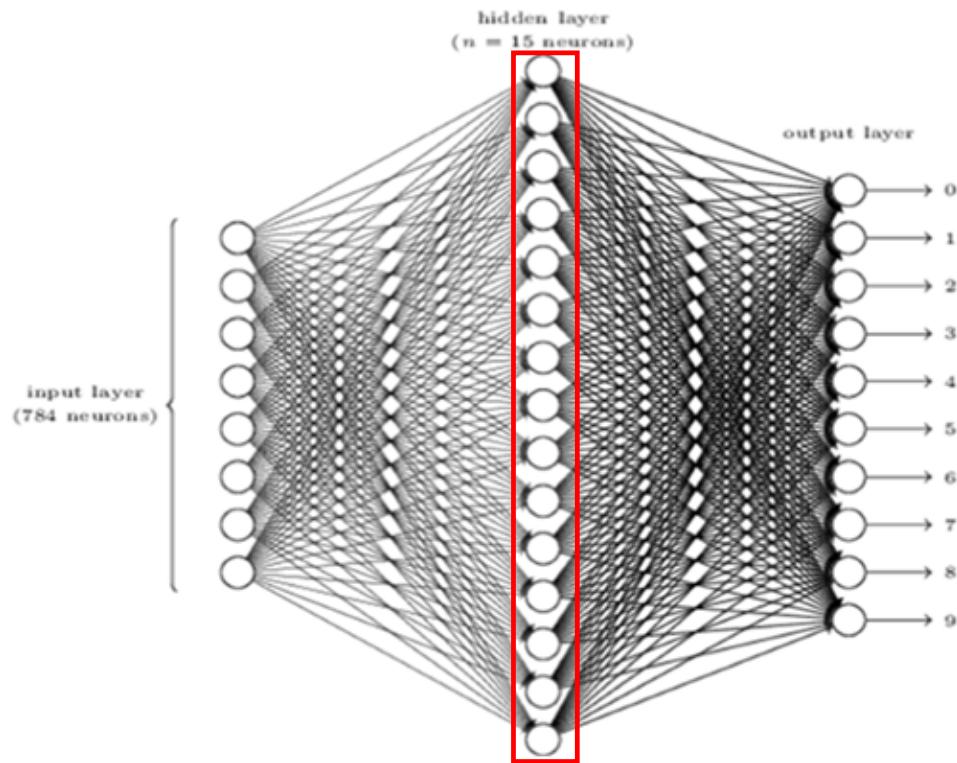
```
x: 784 x 1           a1: 785 x 1
w1: 15 x (784 + 1)
w2: 10 x (15 + 1)

import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2) 15 x 1
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

# MNIST Example



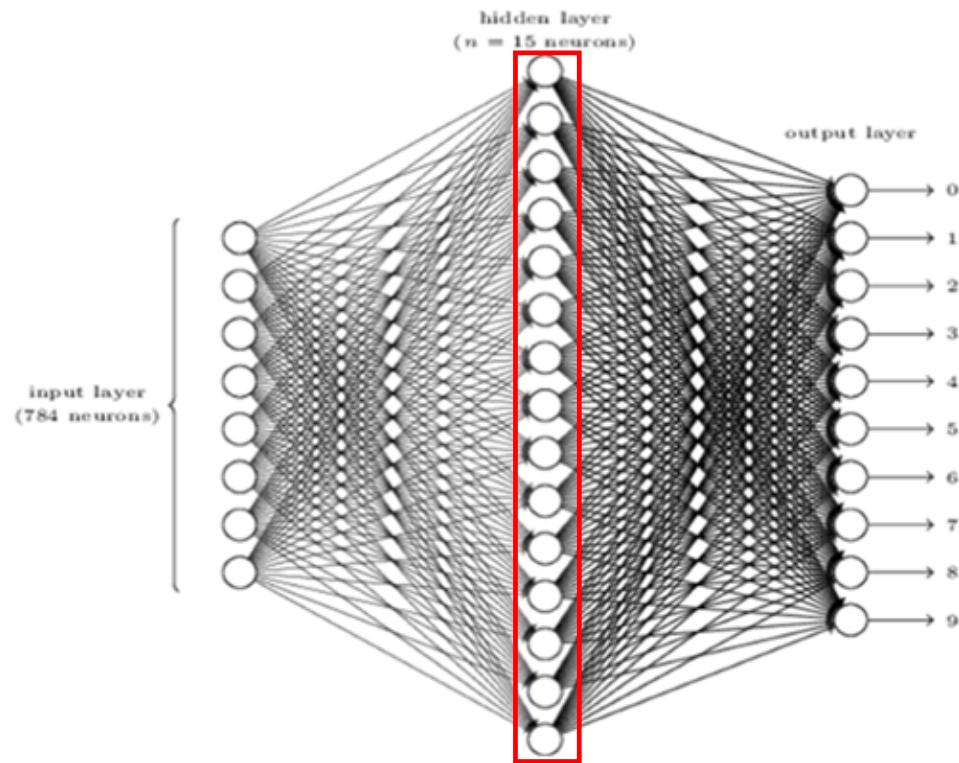
```
x: 784 x 1           a1: 785 x 1
w1: 15 x (784 + 1)
w2: 10 x (15 + 1)

import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

# *MNIST Example*



x: 784 x 1                                a1: 785 x 1  
w1: 15 x (784 + 1)  
w2: 10 x (15 + 1)

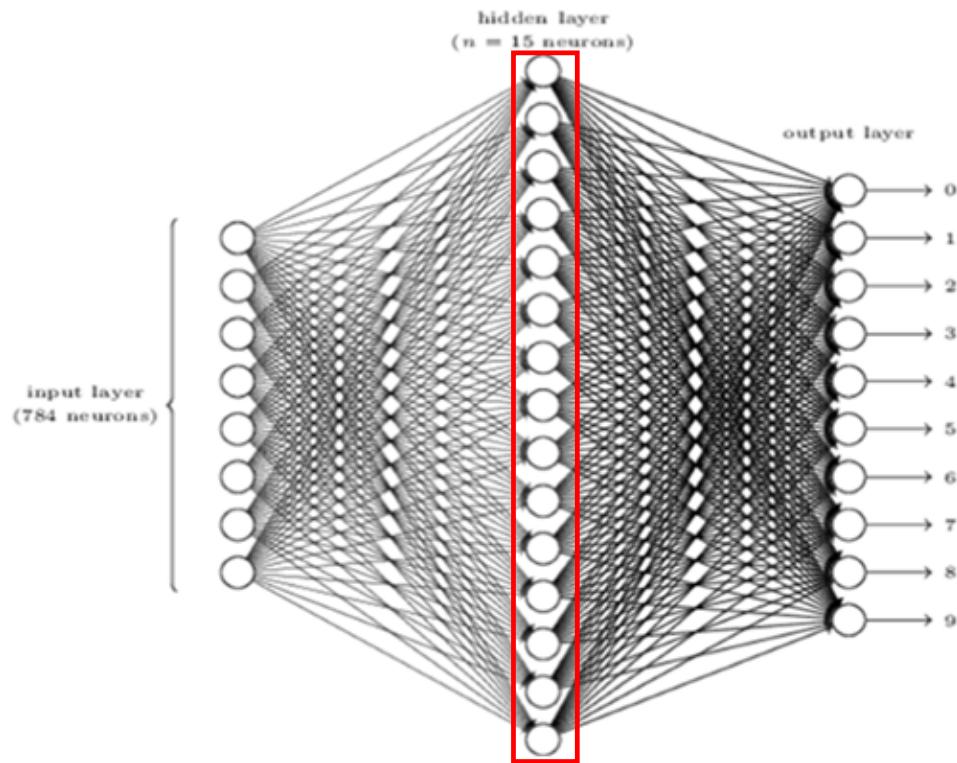
```
import numpy as np
```

```
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))
```

```
def neural_network(x, w1, w2):  
    a1 = np.concatenate([x, [[1]]], 0)  
    z2 = np.dot(w1, a1)  
    a2 = sigmoid(z2)
```

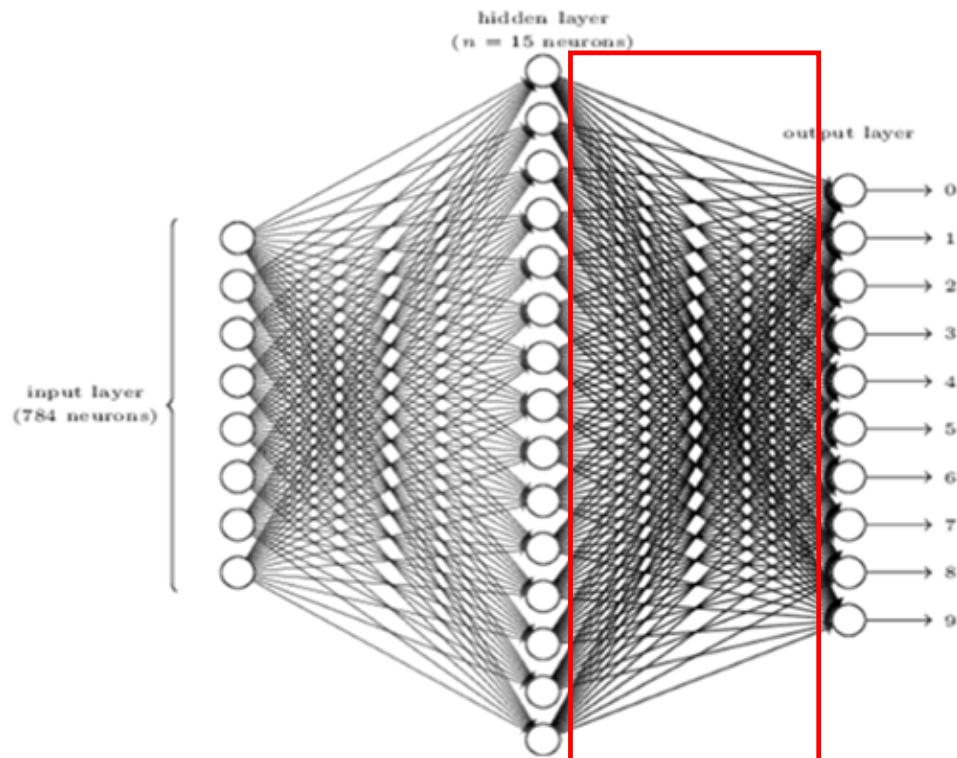
16 x 1

# MNIST Example



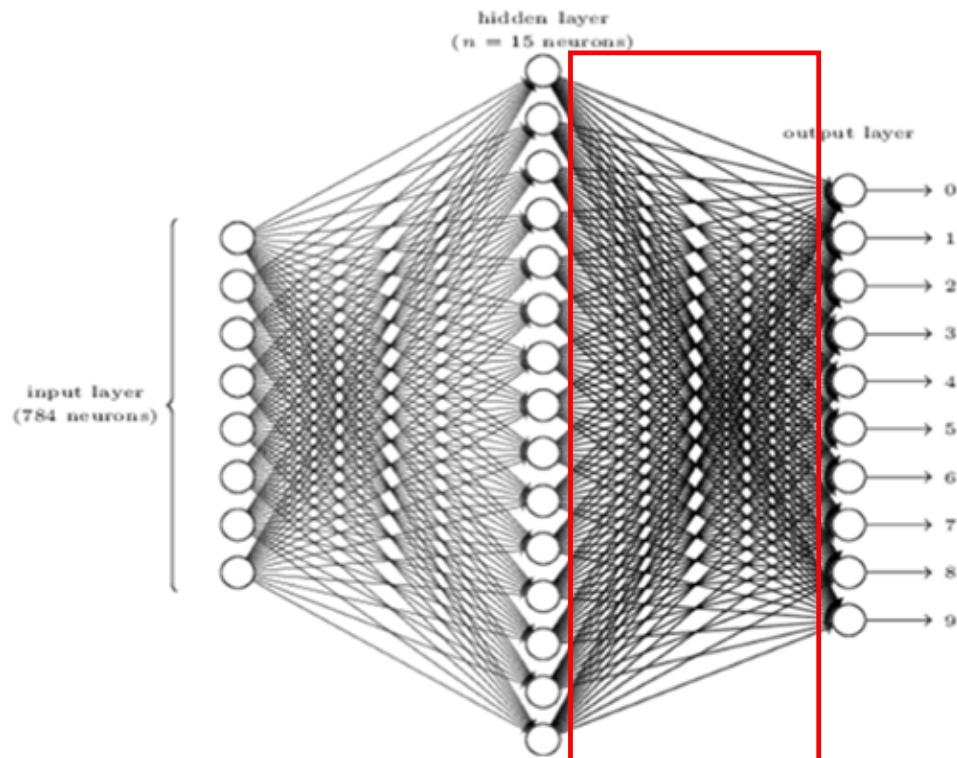
```
x: 784 x 1           a1: 785 x 1  
w1: 15 x (784 + 1)  a2: 16 x 1  
w2: 10 x (15 + 1)  
  
import numpy as np  
  
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
def neural_network(x, w1, w2):  
    a1 = np.concatenate([x, [[1]]], 0)  
    z2 = np.dot(w1, a1)  
    a2 = sigmoid(z2)  
    a2 = np.concatenate([a2, [[1]]], 0)  
    z3 = np.dot(w2, a2)  
    a3 = sigmoid(z3)  
    return a3
```

# MNIST Example



```
x: 784 x 1           a1: 785 x 1  
w1: 15 x (784 + 1)   a2: 16 x 1  
w2: 10 x (15 + 1)  
  
import numpy as np  
  
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
def neural_network(x, w1, w2):  
    a1 = np.concatenate([x, [[1]]], 0)  
    z2 = np.dot(w1, a1)  
    a2 = sigmoid(z2)  
    a2 = np.concatenate([a2, [[1]]], 0)  
    z3 = np.dot(w2, a2)  
    a3 = sigmoid(z3)  
    return a3
```

# MNIST Example



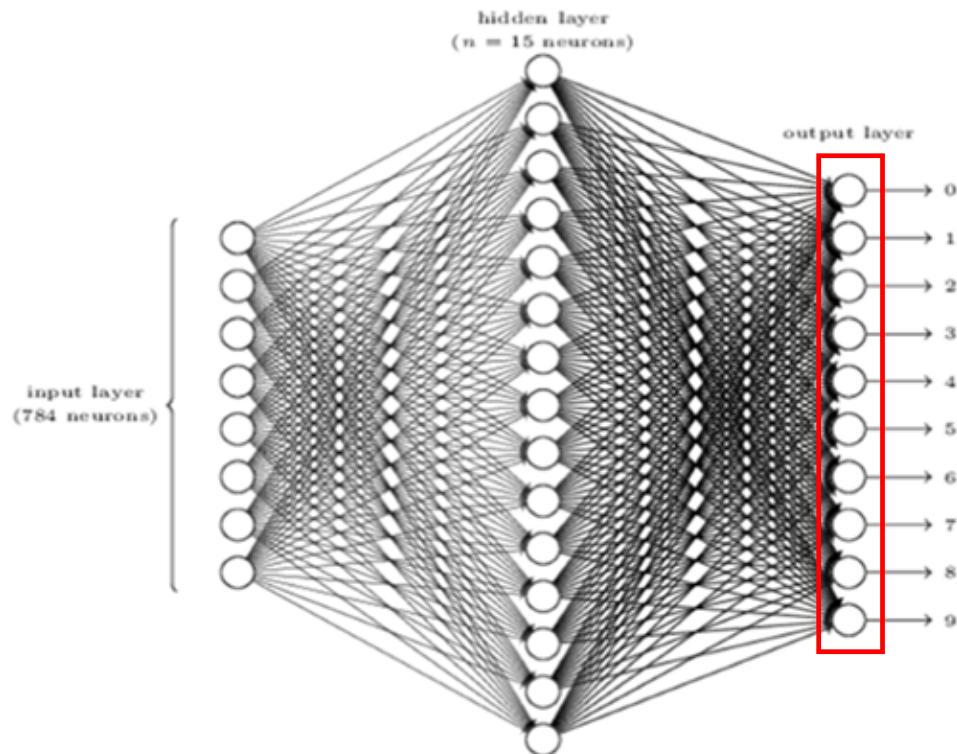
```
x: 784 x 1           a1: 785 x 1  
w1: 15 x (784 + 1)   a2: 16 x 1  
w2: 10 x (15 + 1)
```

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

# MNIST Example



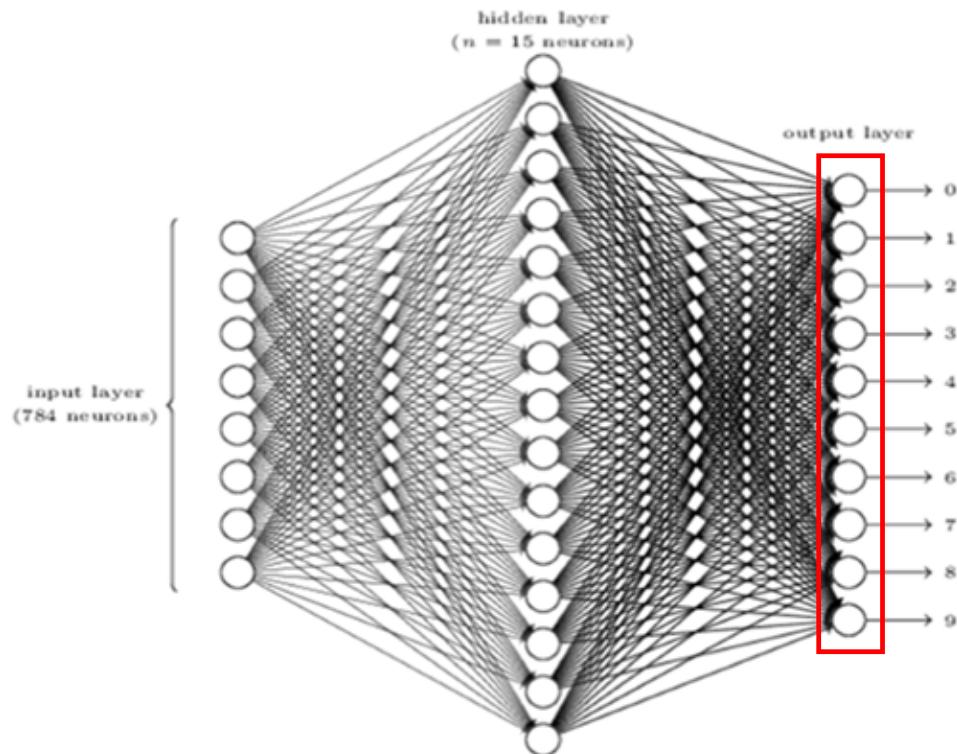
```
x: 784 x 1           a1: 785 x 1
w1: 15 x (784 + 1)   a2: 16 x 1
w2: 10 x (15 + 1)

import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

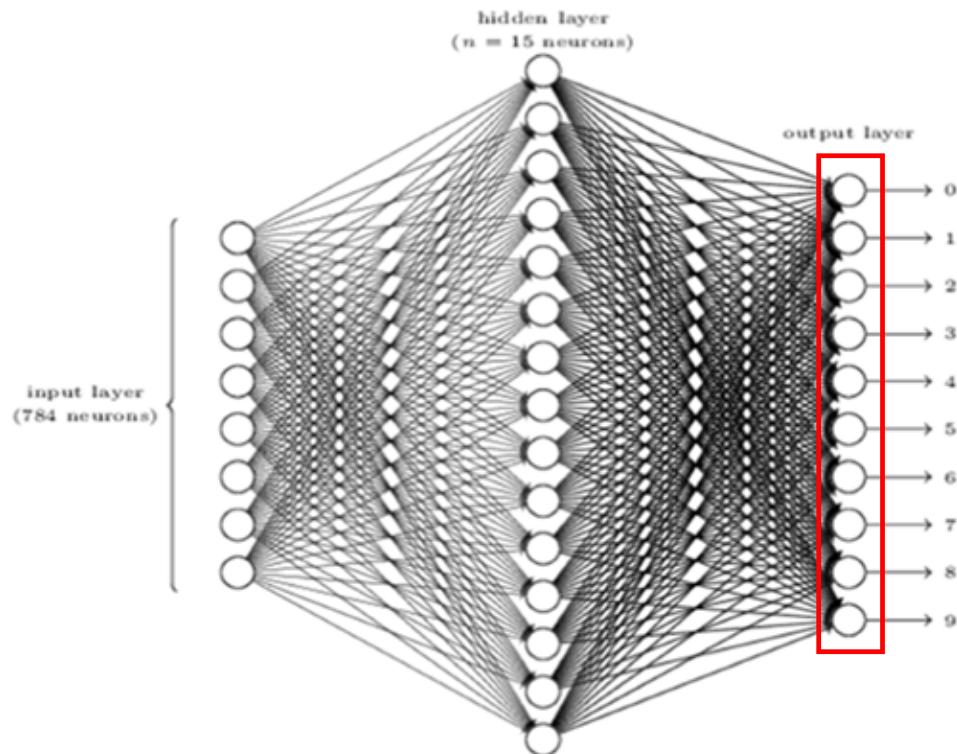
def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

# MNIST Example



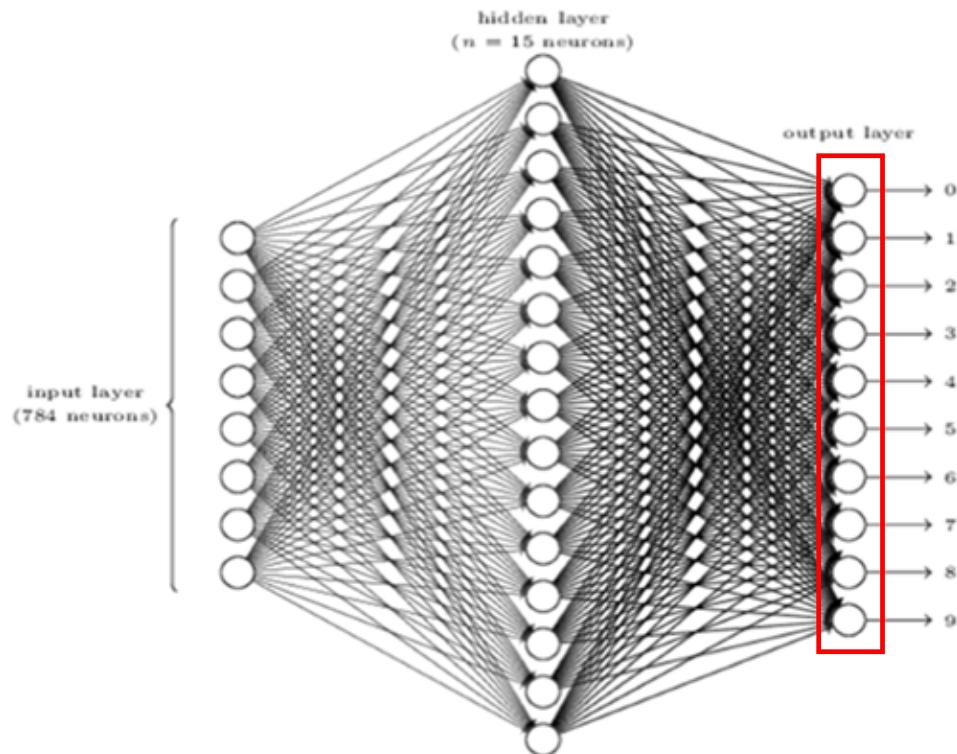
```
x: 784 x 1           a1: 785 x 1  
w1: 15 x (784 + 1)   a2: 16 x 1  
w2: 10 x (15 + 1)  
  
import numpy as np  
  
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
def neural_network(x, w1, w2):  
    a1 = np.concatenate([x, [[1]]], 0)  
    z2 = np.dot(w1, a1)  
    a2 = sigmoid(z2)  
    a2 = np.concatenate([a2, [[1]]], 0)  
    z3 = np.dot(w2, a2)  
    a3 = sigmoid(z3)  
    return a3
```

# MNIST Example



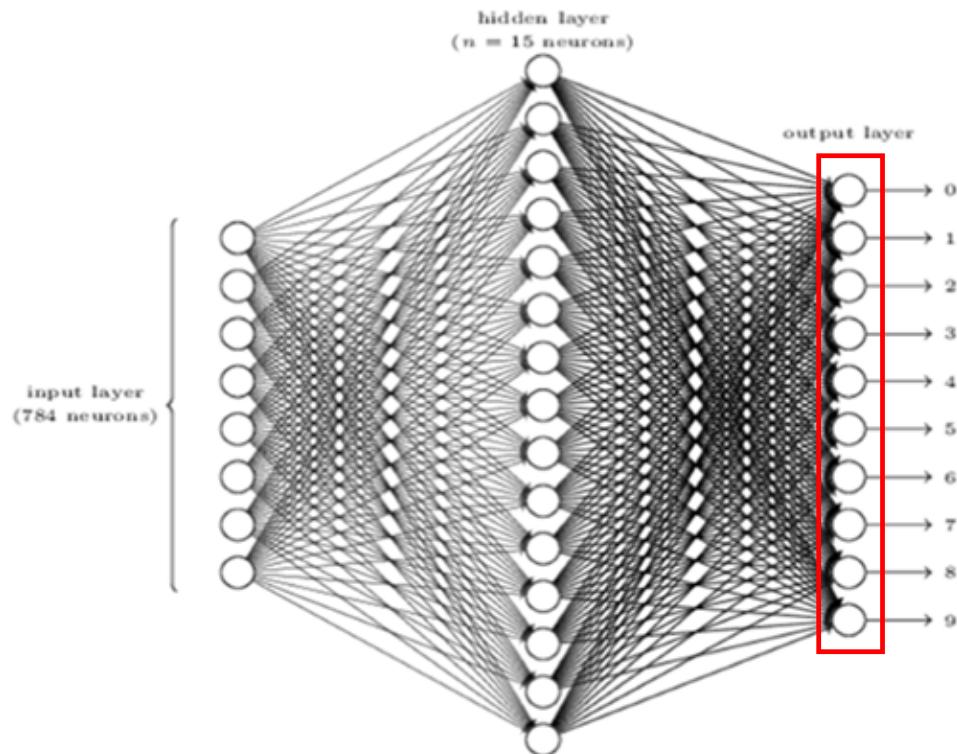
```
x: 784 x 1           a1: 785 x 1  
w1: 15 x (784 + 1)  a2: 16 x 1  
w2: 10 x (15 + 1)  
  
import numpy as np  
  
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
def neural_network(x, w1, w2):  
    a1 = np.concatenate([x, [[1]]], 0)  
    z2 = np.dot(w1, a1)  
    a2 = sigmoid(z2)  
    a2 = np.concatenate([a2, [[1]]], 0)  
    z3 = np.dot(w2, a2)  
    a3 = sigmoid(z3)  
    return a3
```

# MNIST Example



```
x: 784 x 1           a1: 785 x 1  
w1: 15 x (784 + 1)  a2: 16 x 1  
w2: 10 x (15 + 1)  
  
import numpy as np  
  
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
def neural_network(x, w1, w2):  
    a1 = np.concatenate([x, [[1]]], 0)  
    z2 = np.dot(w1, a1)  
    a2 = sigmoid(z2)  
    a2 = np.concatenate([a2, [[1]]], 0)  
    z3 = np.dot(w2, a2) 10 x 1  
    a3 = sigmoid(z3)  
    return a3
```

# MNIST Example



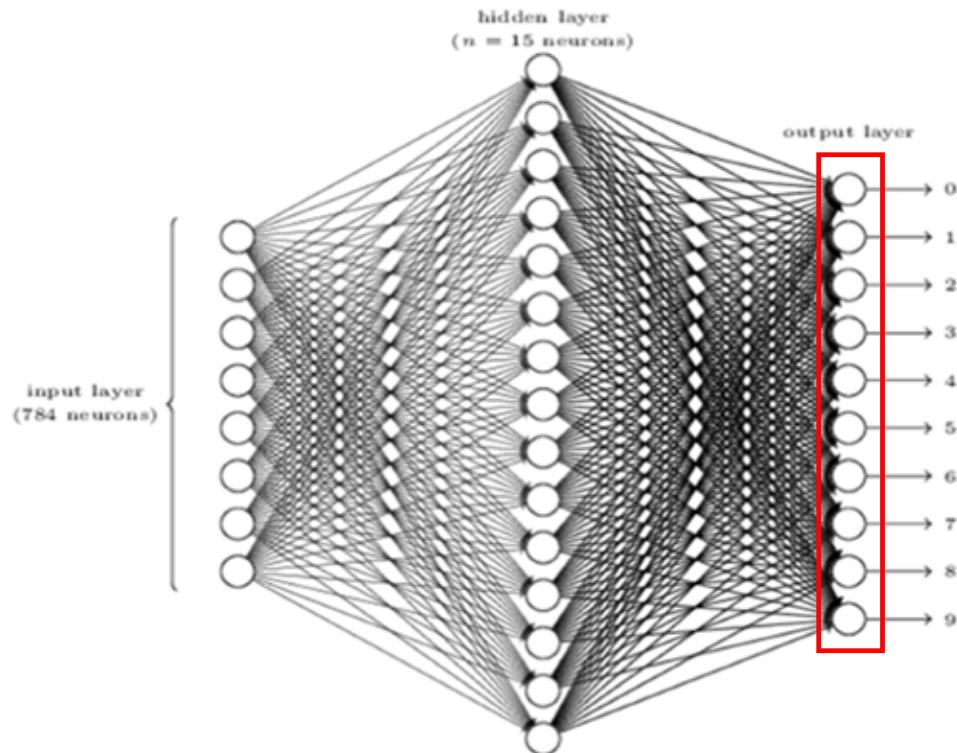
```
x: 784 x 1           a1: 785 x 1
w1: 15 x (784 + 1)   a2: 16 x 1
w2: 10 x (15 + 1)

import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

# MNIST Example



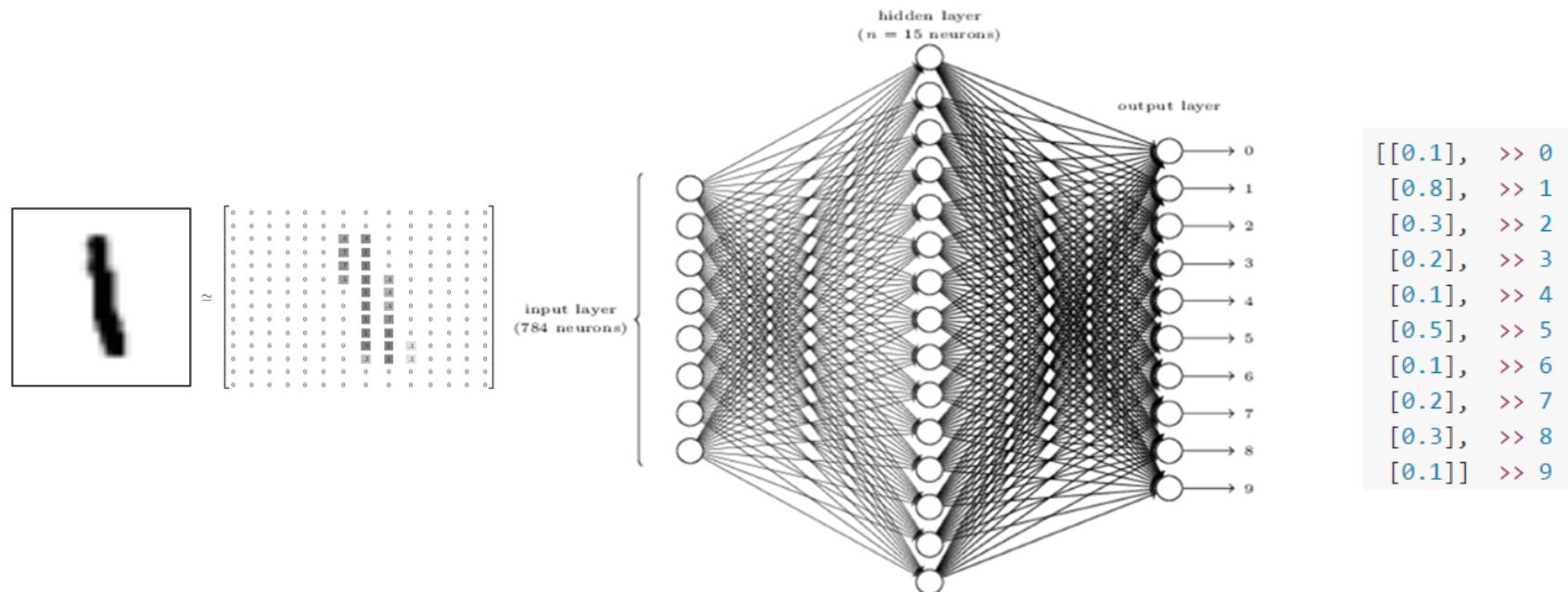
```
x: 784 x 1           a1: 785 x 1
w1: 15 x (784 + 1)   a2: 16 x 1
w2: 10 x (15 + 1)

import numpy as np

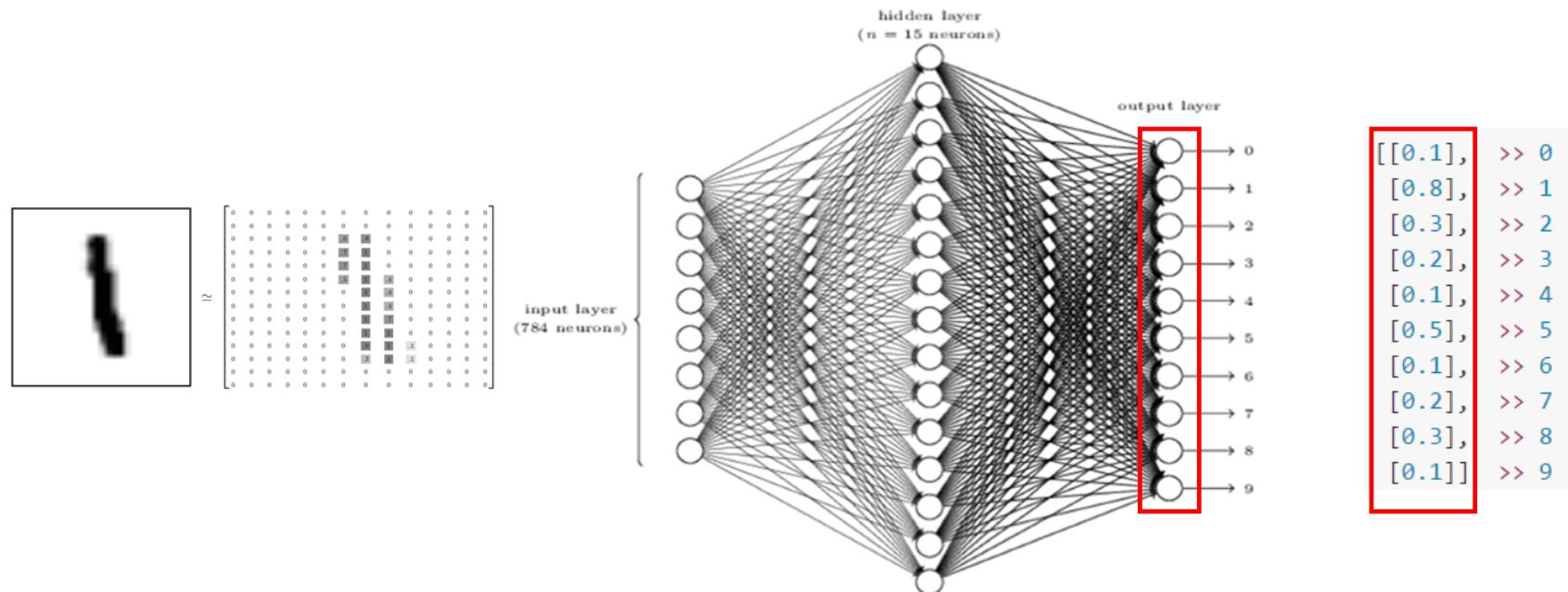
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3) 10 x 1
    return a3
```

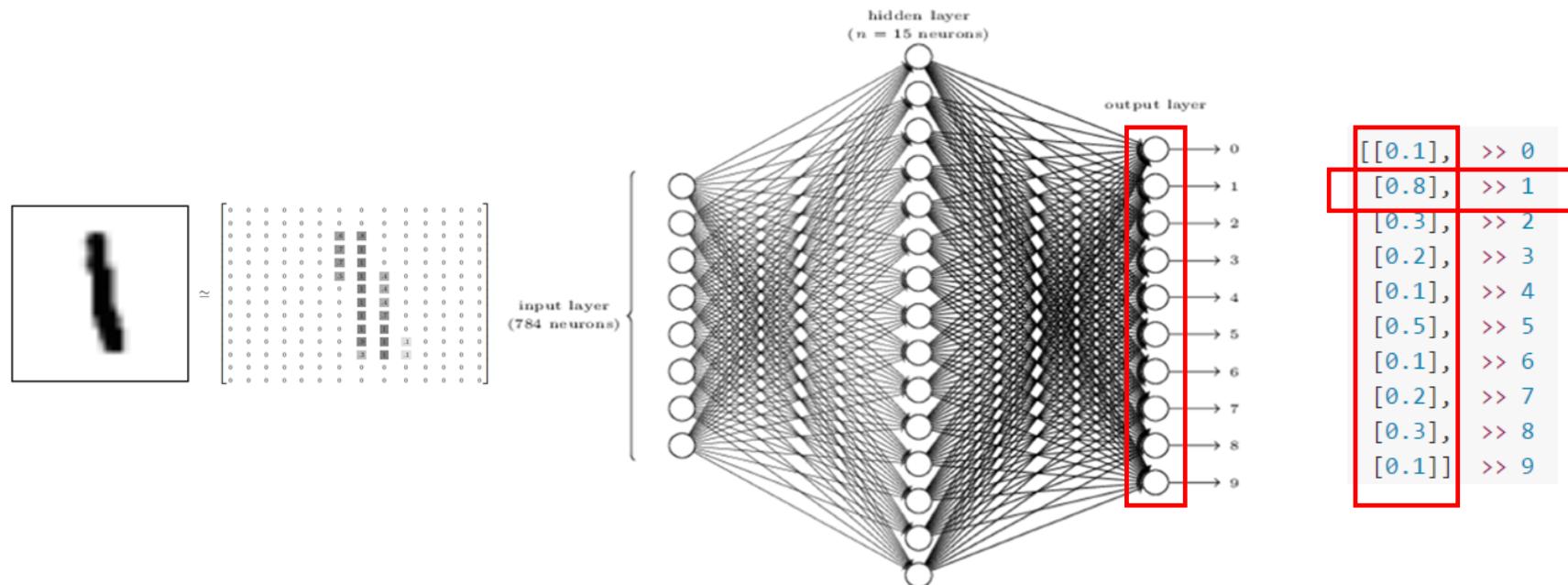
# MNIST Example



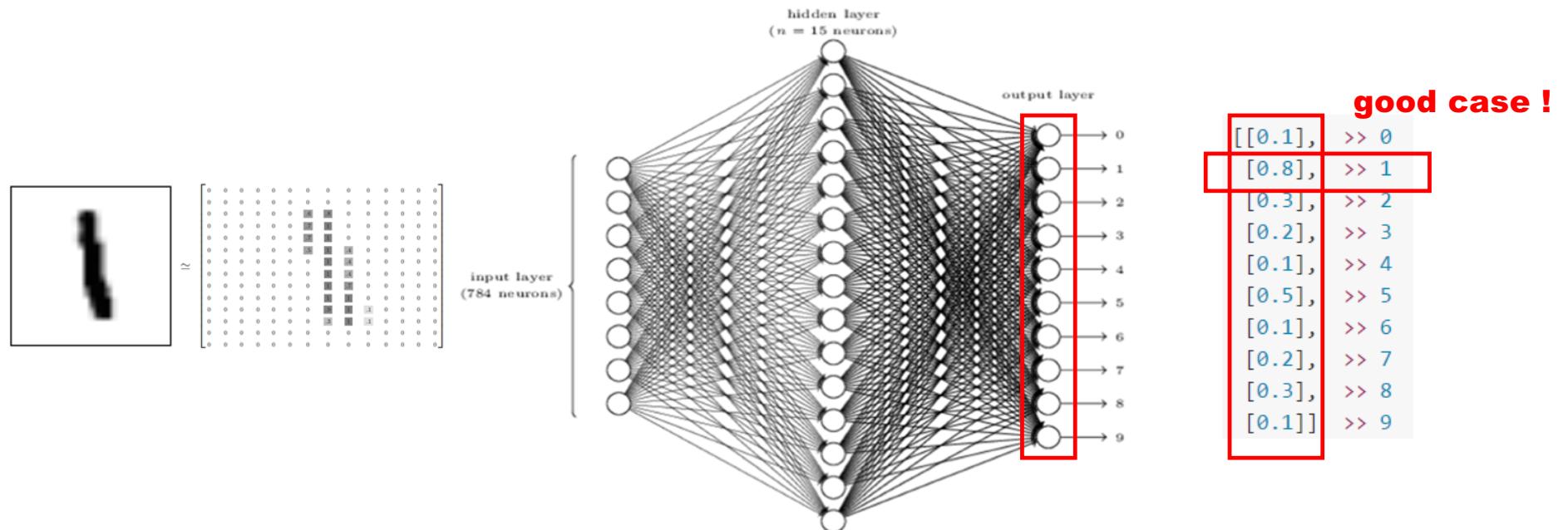
# MNIST Example



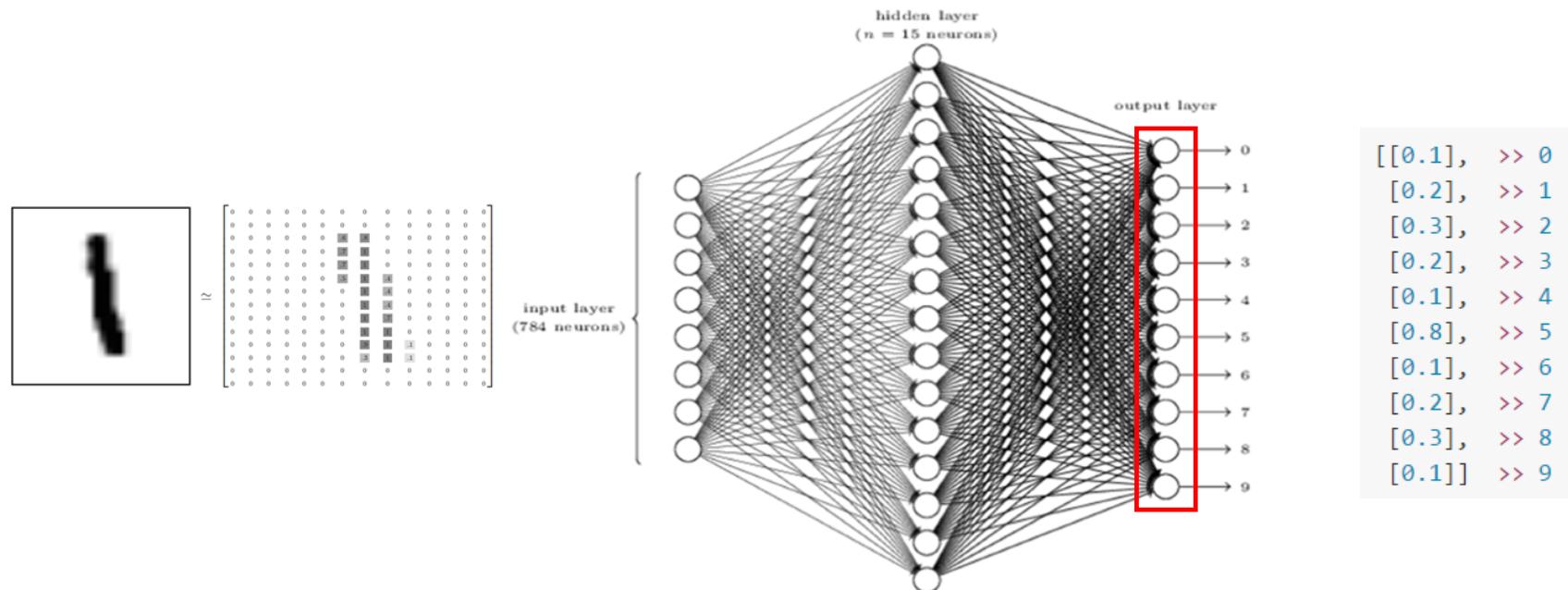
# MNIST Example



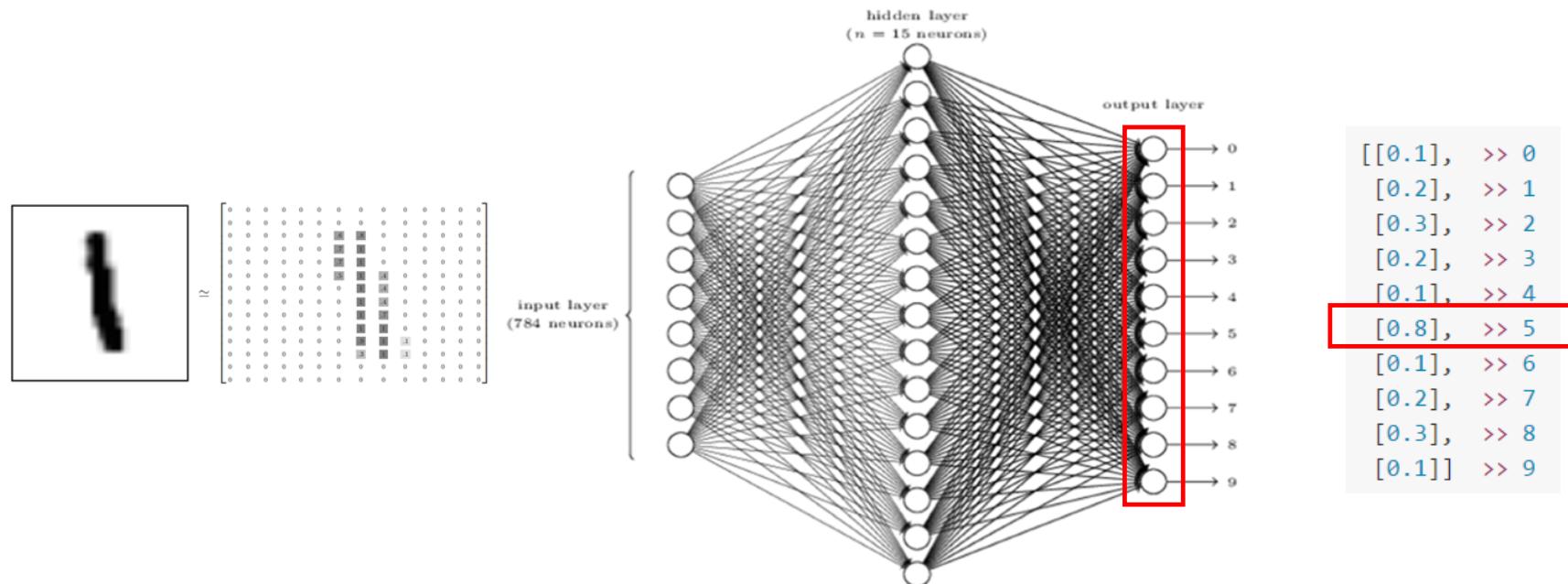
# MNIST Example



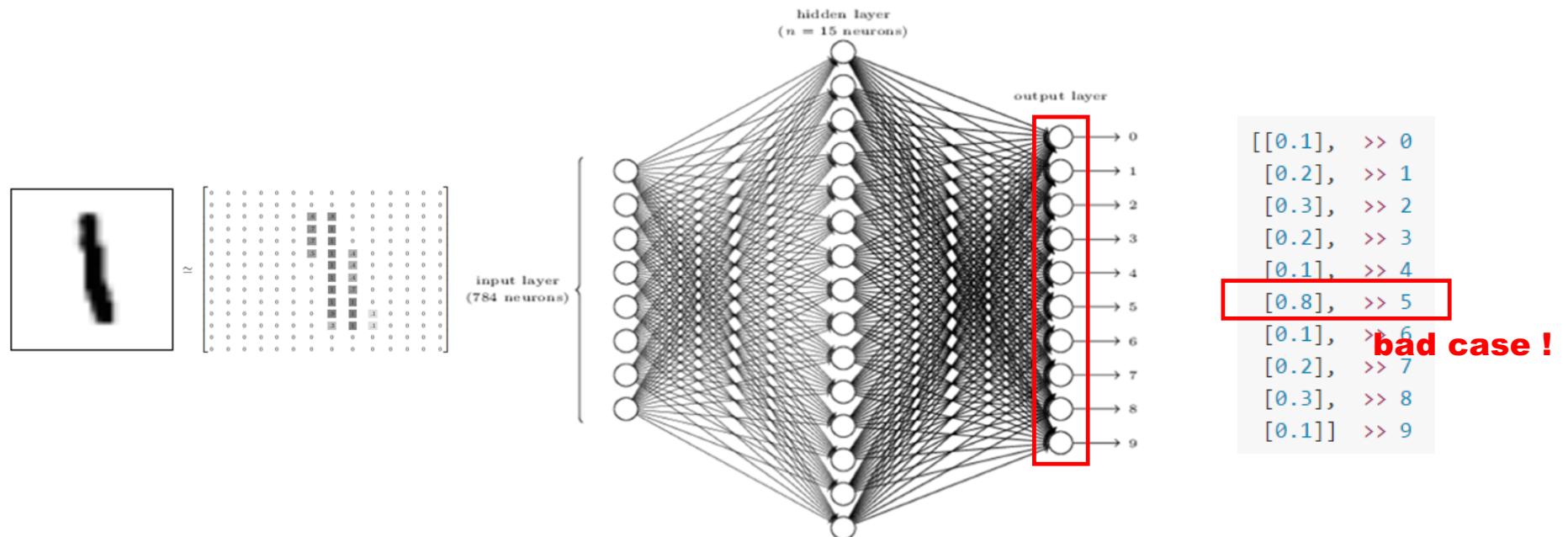
# MNIST Example



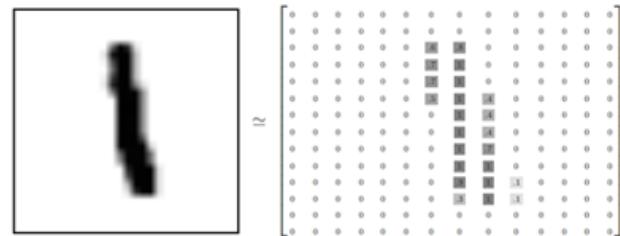
# MNIST Example



# MNIST Example



# MNIST Example



$$\approx \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$$

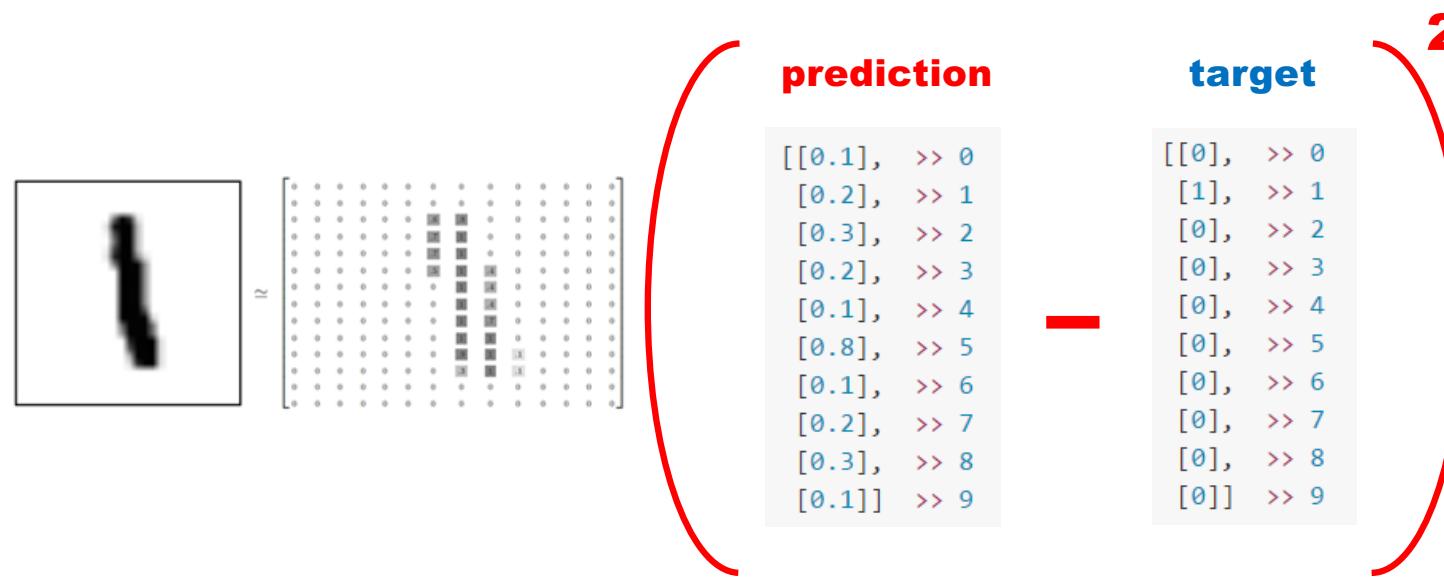
**prediction**

```
[[0.1],  >> 0  
[0.2],  >> 1  
[0.3],  >> 2  
[0.2],  >> 3  
[0.1],  >> 4  
[0.8],  >> 5  
[0.1],  >> 6  
[0.2],  >> 7  
[0.3],  >> 8  
[0.1]] >> 9
```

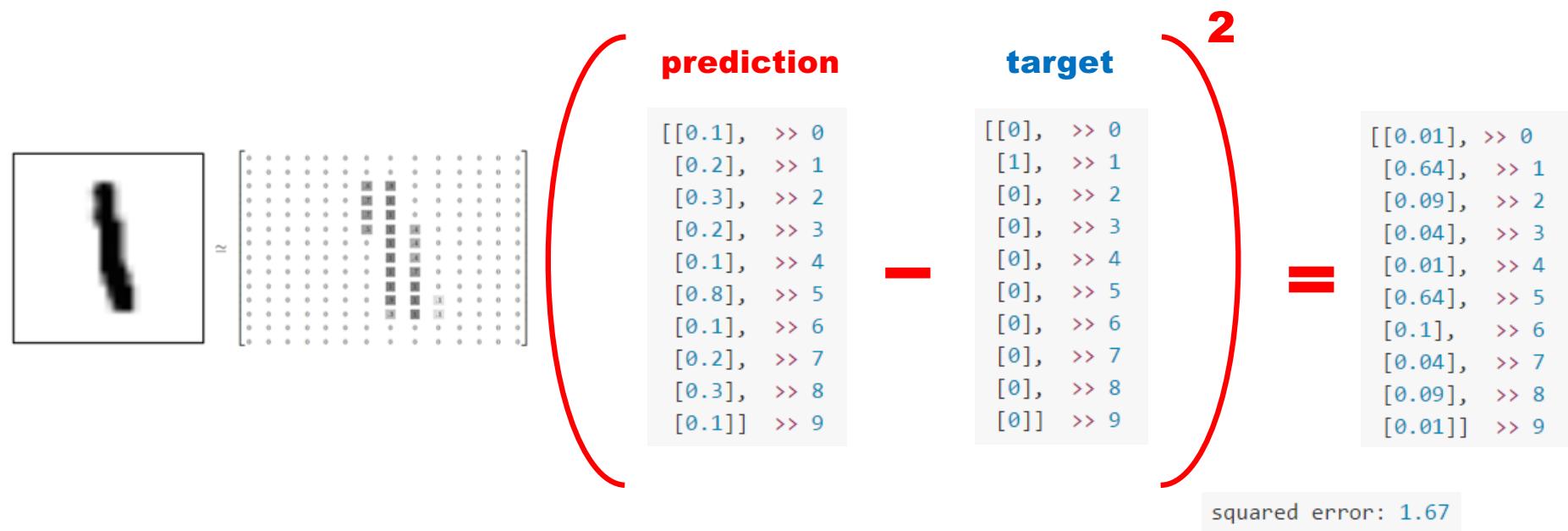
**target**

```
[[0],  >> 0  
[1],  >> 1  
[0],  >> 2  
[0],  >> 3  
[0],  >> 4  
[0],  >> 5  
[0],  >> 6  
[0],  >> 7  
[0],  >> 8  
[0]] >> 9
```

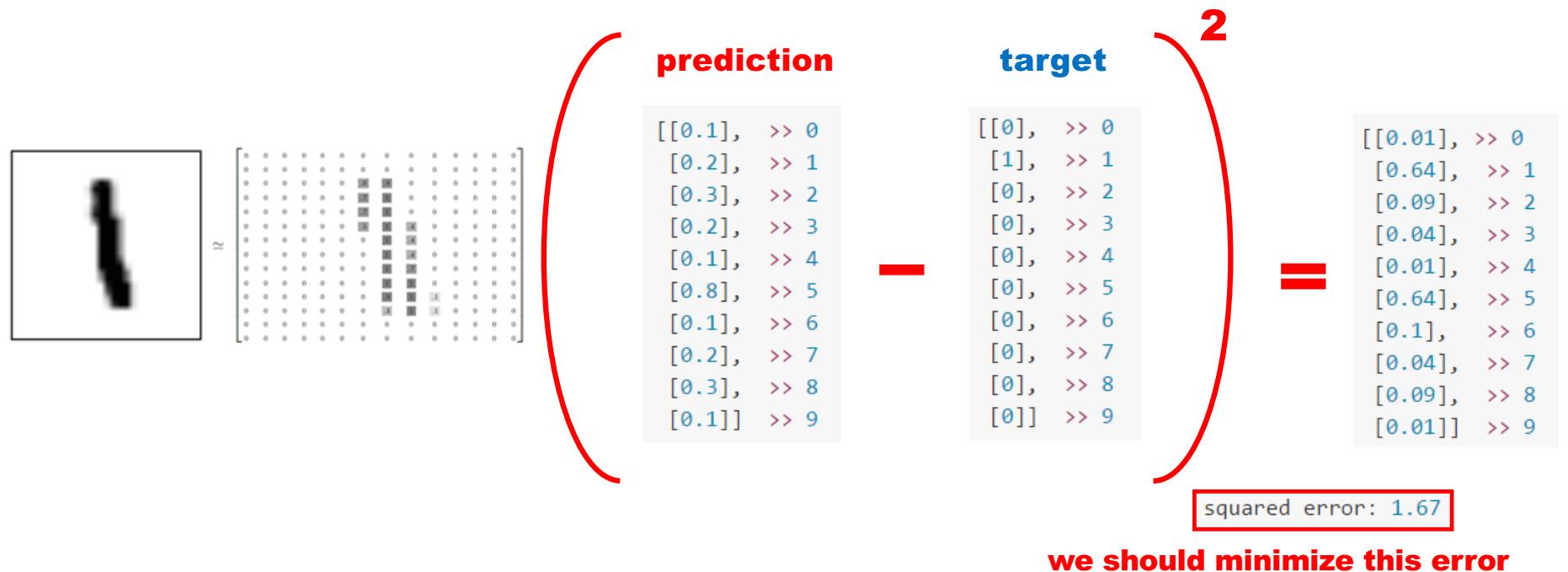
# MNIST Example



# MNIST Example



# MNIST Example



# Cost Function

Assume we have 1 data and dimension of output vector is n.

## Regression

Mean Squared Error

$$\sum_{i=1}^n (\hat{y}_i - y_i)^2$$

```
[[0.1], >> 0  
[0.2], >> 1  
[0.3], >> 2  
[0.2], >> 3  
[0.1], >> 4  
[0.8], >> 5  
[0.1], >> 6  
[0.2], >> 7  
[0.3], >> 8  
[0.1]] >> 9
```

```
[[0], >> 0  
[1], >> 1  
[0], >> 2  
[0], >> 3  
[0], >> 4  
[0], >> 5  
[0], >> 6  
[0], >> 7  
[0], >> 8  
[0]] >> 9
```

## Classification

Sigmoid Cross Entropy Loss

$$\sum_{i=1}^n -y_i \log \hat{y}_i - (1-y_i) \log(1-\hat{y}_i)$$

Softmax Cross Entropy Loss

$$\sum_{i=1}^n -y_i \log \hat{y}_i$$

# Cost Function

Assume we have 1 data and dimension of output vector is n.

## Regression

Mean Squared Error

$$\sum_{i=1}^n (\hat{y}_i - y_i)^2$$

## Classification

Sigmoid Cross Entropy Loss

$$\sum_{i=1}^n -y_i \log \hat{y}_i - (1-y_i) \log(1-\hat{y}_i)$$

when using sigmoid  
function in last layer

Softmax Cross Entropy Loss

$$\sum_{i=1}^n -y_i \log \hat{y}_i$$

# Cost Function

Assume we have 1 data and dimension of output vector is n.

**Regression**

Mean Squared Error

$$\sum_{i=1}^n (\hat{y}_i - y_i)^2$$

**Classification**

Sigmoid Cross Entropy Loss

$$\sum_{i=1}^n -y_i \log \hat{y}_i - (1-y_i) \log(1-\hat{y}_i)$$

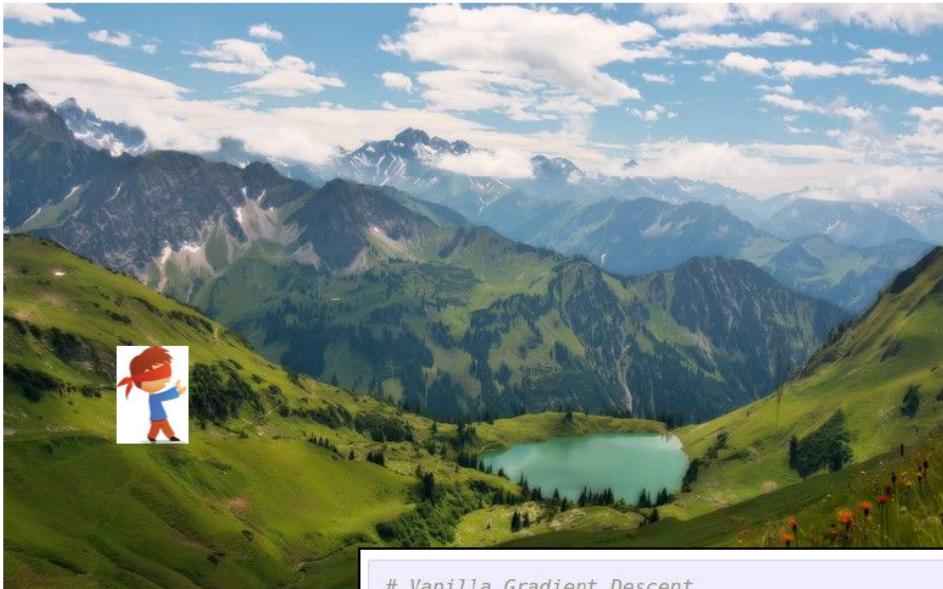
Softmax Cross Entropy Loss

**when using softmax  
function in last layer**

$$\sum_{i=1}^n -y_i \log \hat{y}_i$$

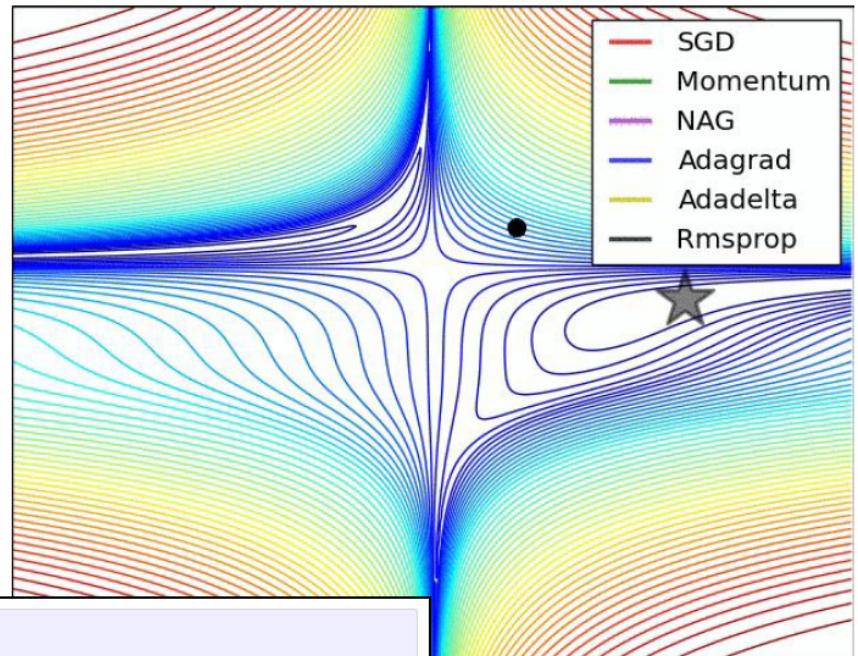
# **5. Backpropagation**

# Gradient Descent

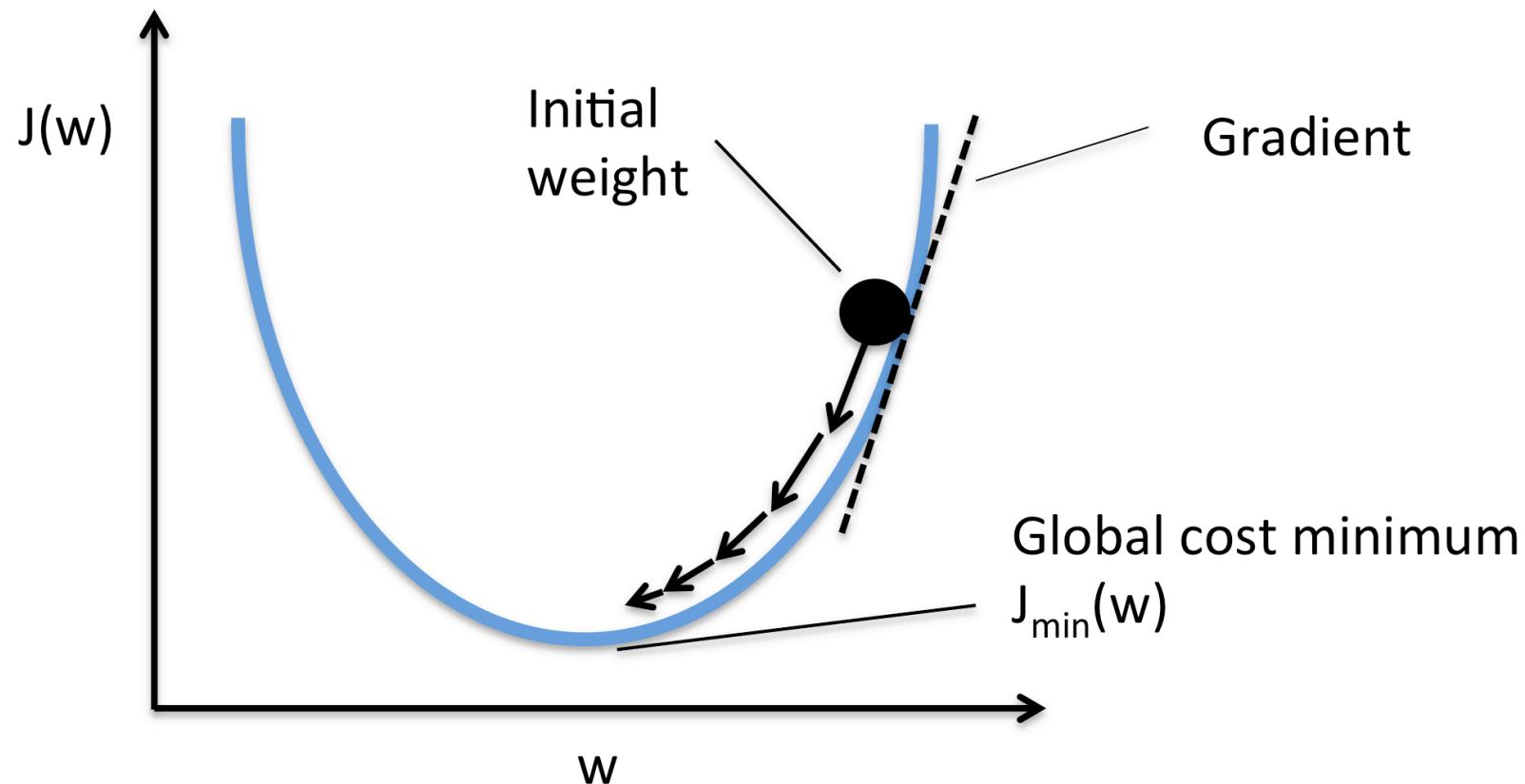


```
# Vanilla Gradient Descent

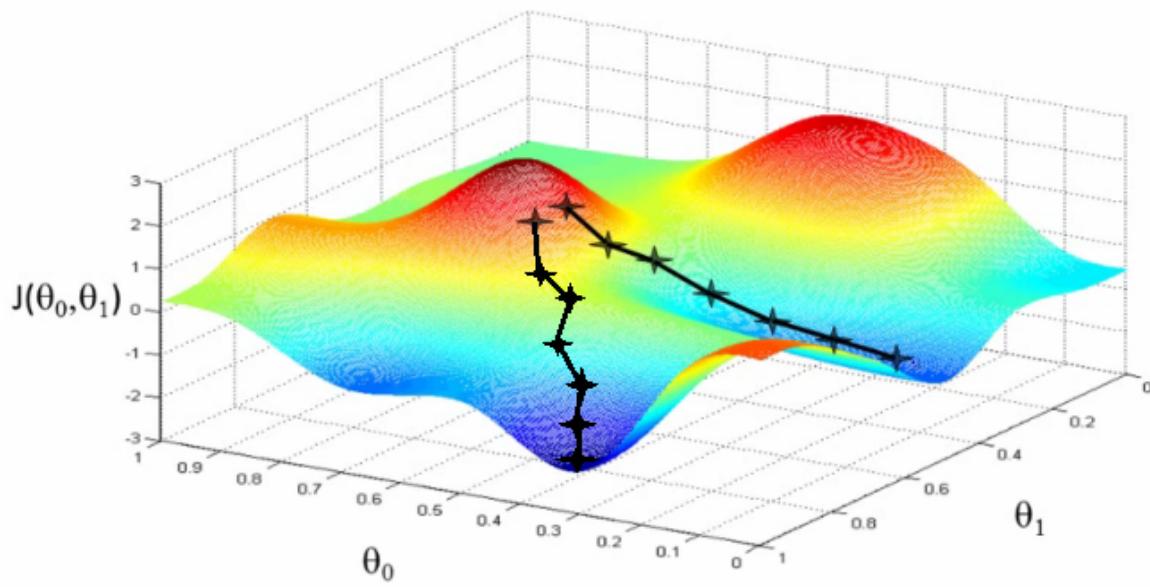
while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```



## *Gradient Descent*



## Gradient Descent



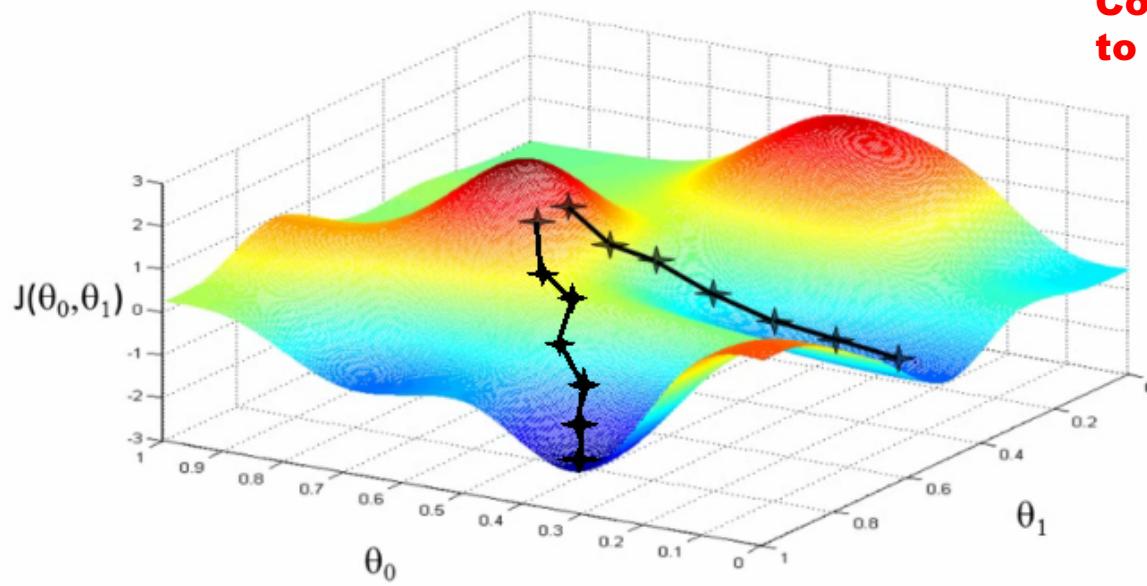
compute  $\frac{\partial J}{\partial \theta_0}$  and  $\frac{\partial J}{\partial \theta_1}$

update weights with

$$\theta_0 := \theta_0 - \alpha \cdot \frac{\partial J}{\partial \theta_0}$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial J}{\partial \theta_1}$$

# Gradient Descent



**Compute gradient of loss with respect to every weights (parameters)**

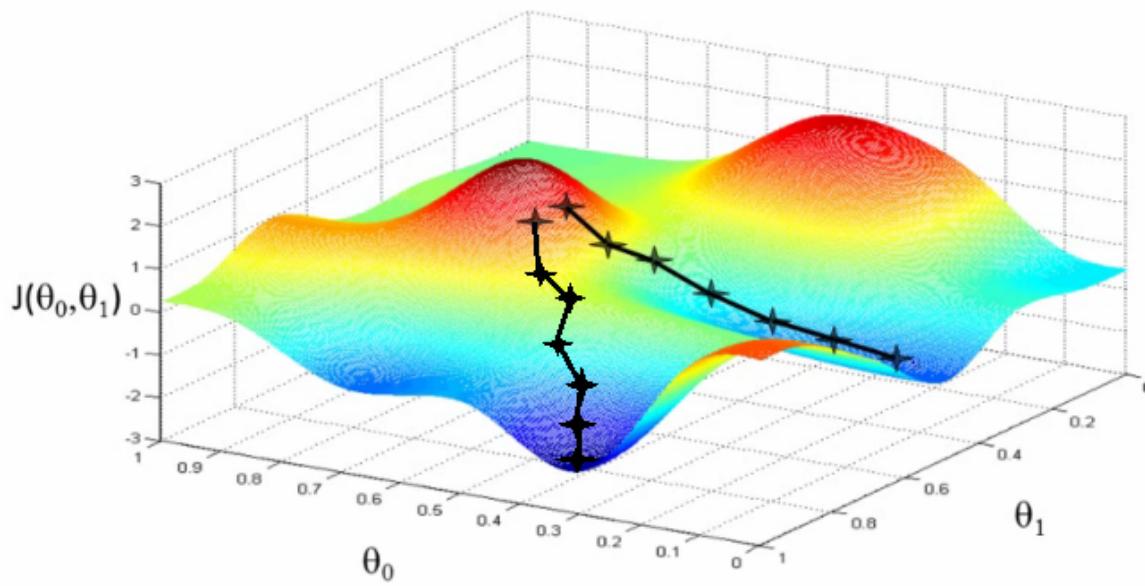
compute  $\frac{\partial J}{\partial \theta_0}$  and  $\frac{\partial J}{\partial \theta_1}$

update weights with

$$\theta_0 := \theta_0 - \alpha \cdot \frac{\partial J}{\partial \theta_0}$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial J}{\partial \theta_1}$$

# Gradient Descent



compute  $\frac{\partial J}{\partial \theta_0}$  and  $\frac{\partial J}{\partial \theta_1}$

update weights with

$$\theta_0 := \theta_0 - \alpha \cdot \frac{\partial J}{\partial \theta_0}$$

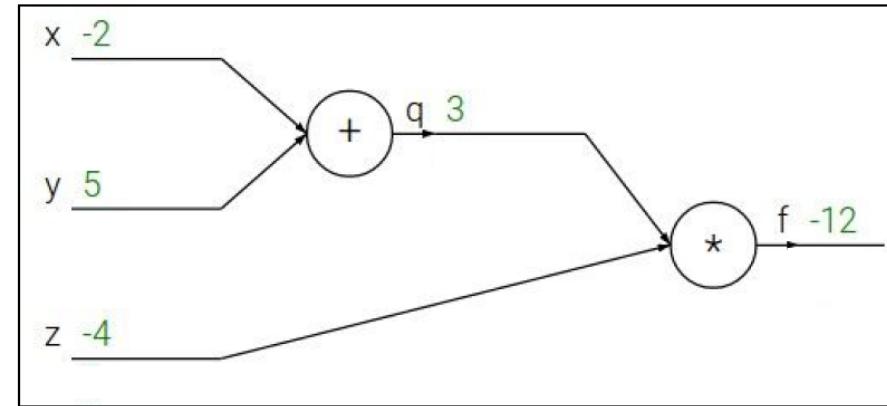
$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial J}{\partial \theta_1}$$

**Gradient “descent” optimization with learning rate ‘alpha’ (e.g. 0.01)**

## *Computational Graph*

$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$



## Computational Graph

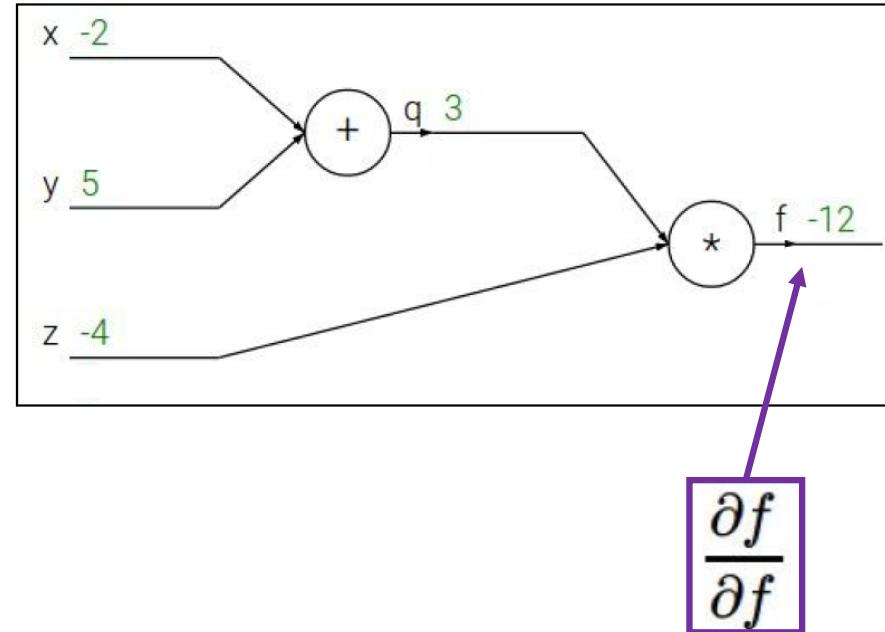
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



## Computational Graph

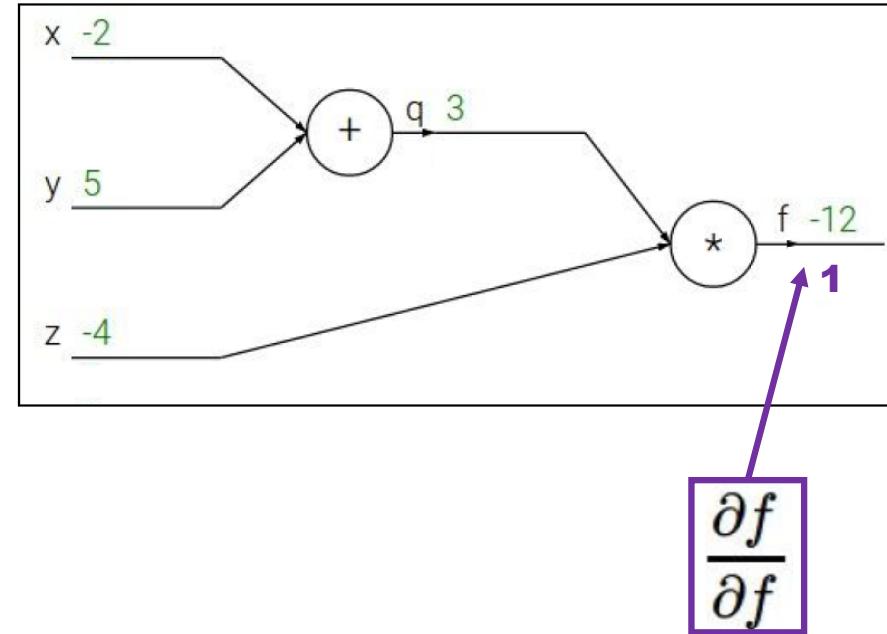
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



## Backpropagation using Chain Rule

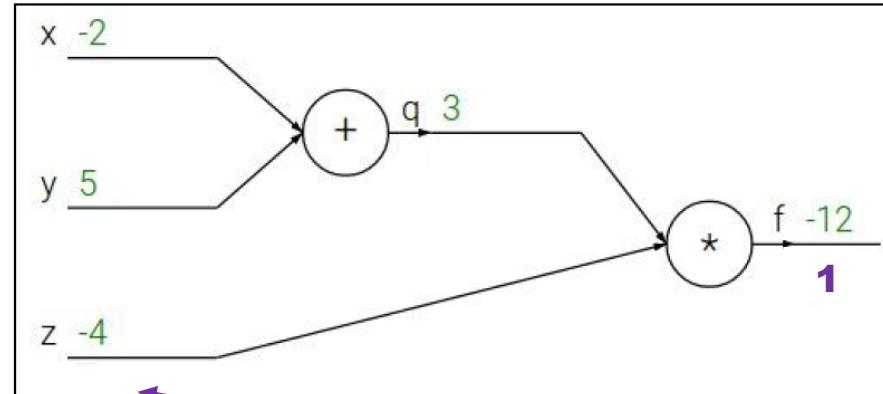
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z}$$

## Backpropagation using Chain Rule

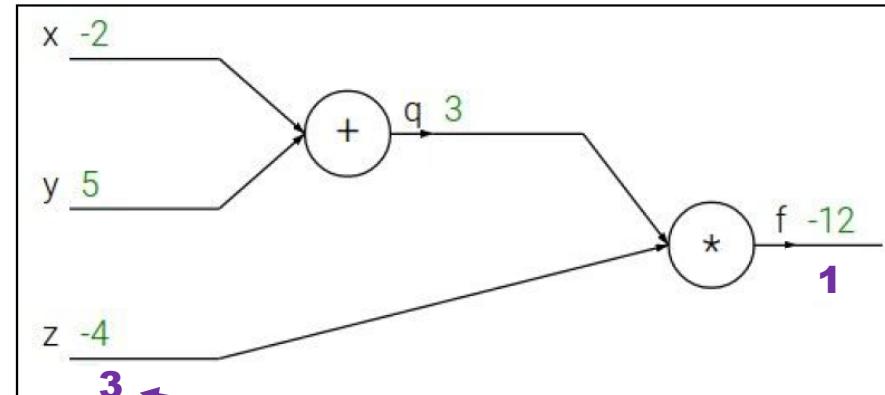
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z}$$

## Computational Graph

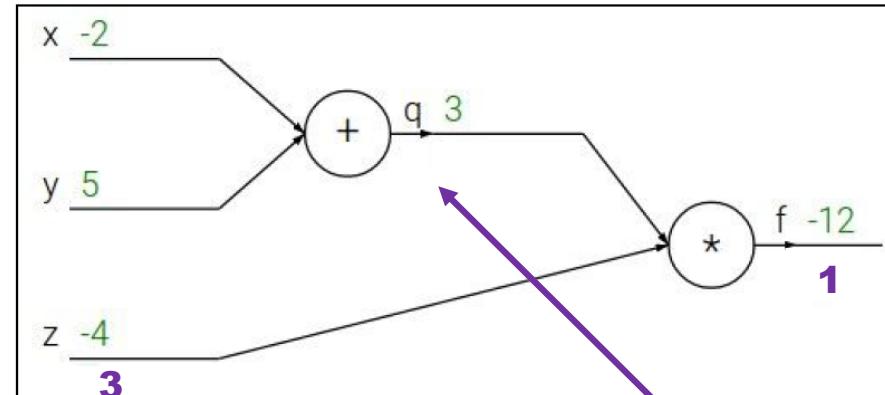
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

## Backpropagation using Chain Rule

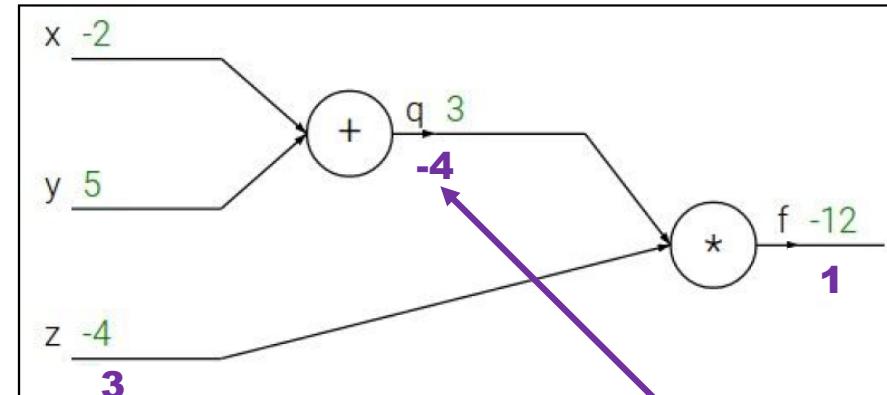
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

## Backpropagation using Chain Rule

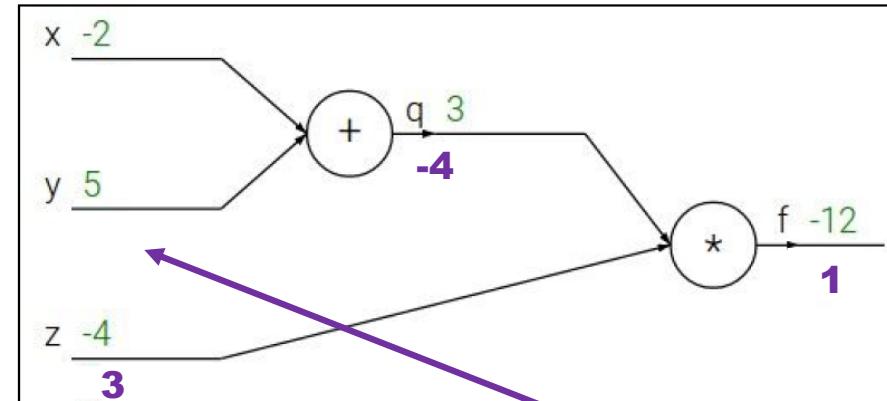
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

## Backpropagation using Chain Rule

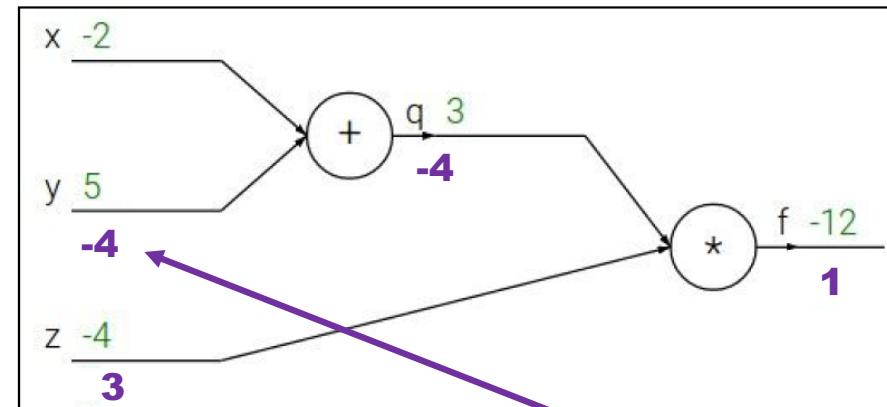
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

$$\frac{\partial f}{\partial y}$$

## Backpropagation using Chain Rule

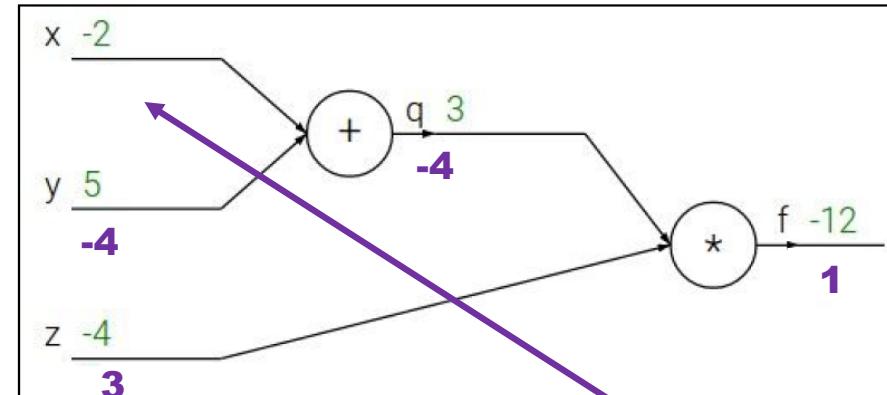
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x}$$

## Backpropagation using Chain Rule

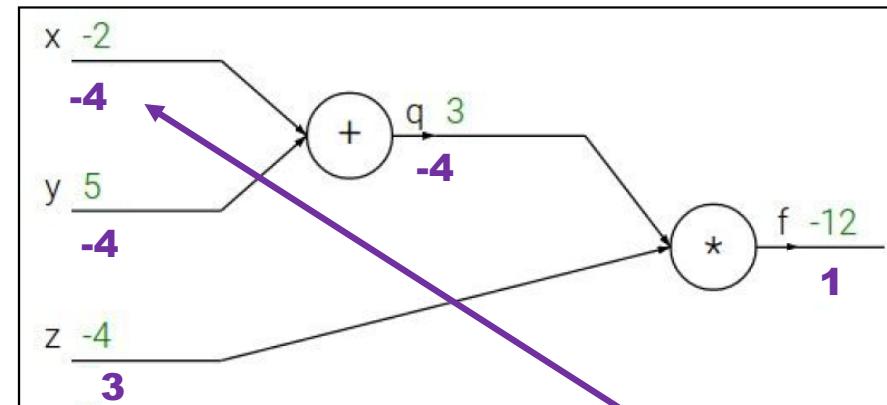
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

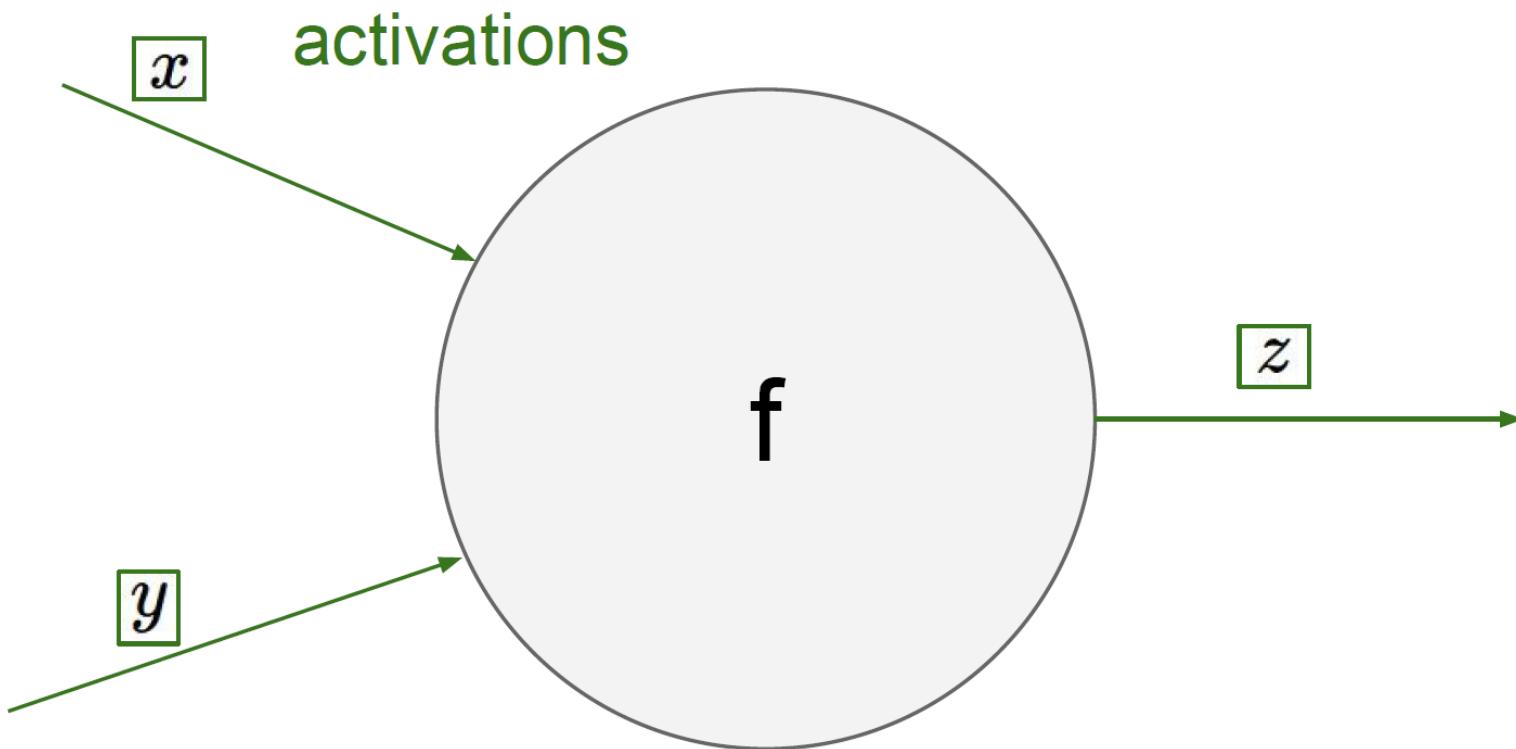


Chain rule:

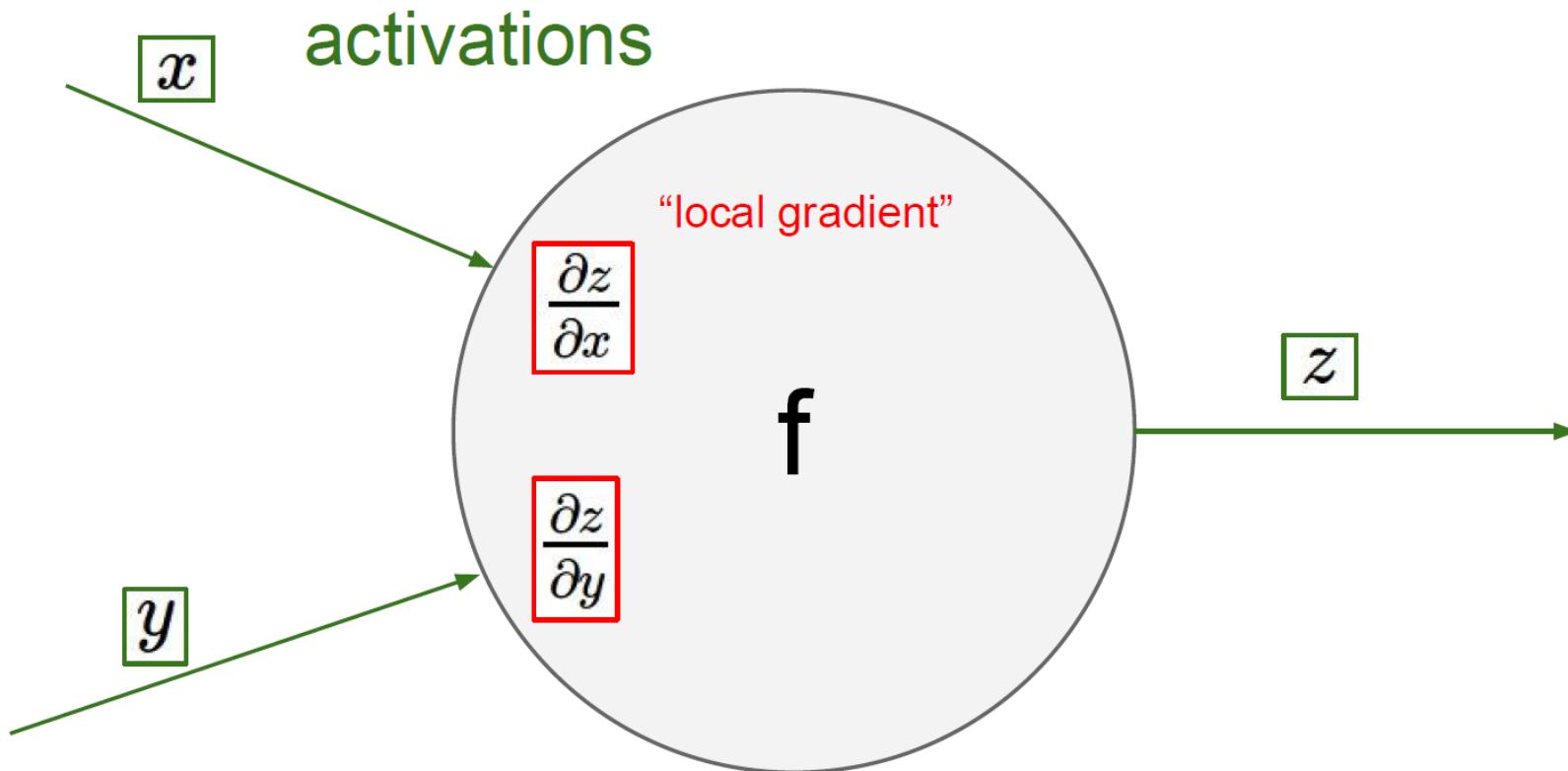
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

$$\frac{\partial f}{\partial x}$$

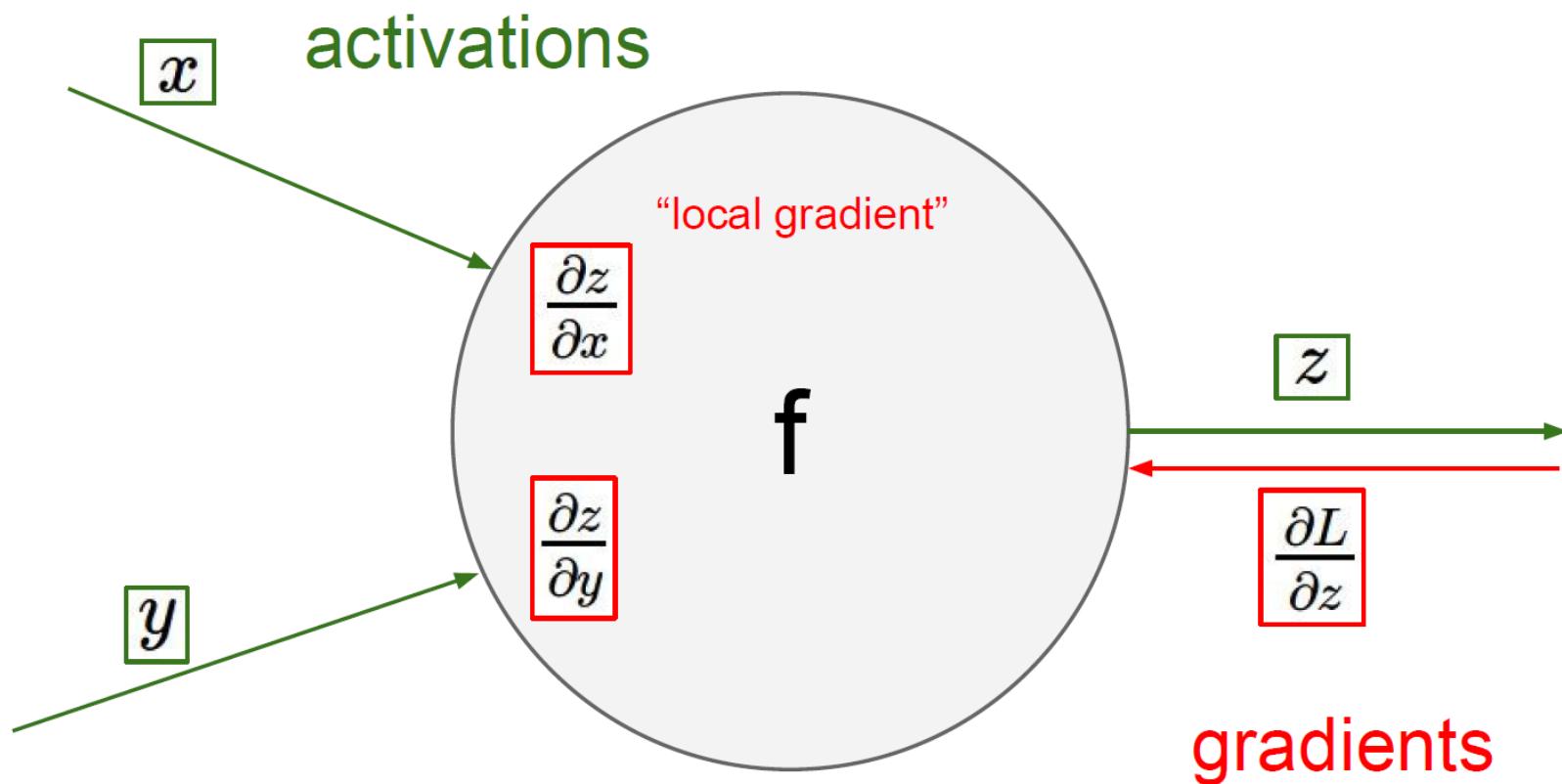
## *Backpropagation using Chain Rule*



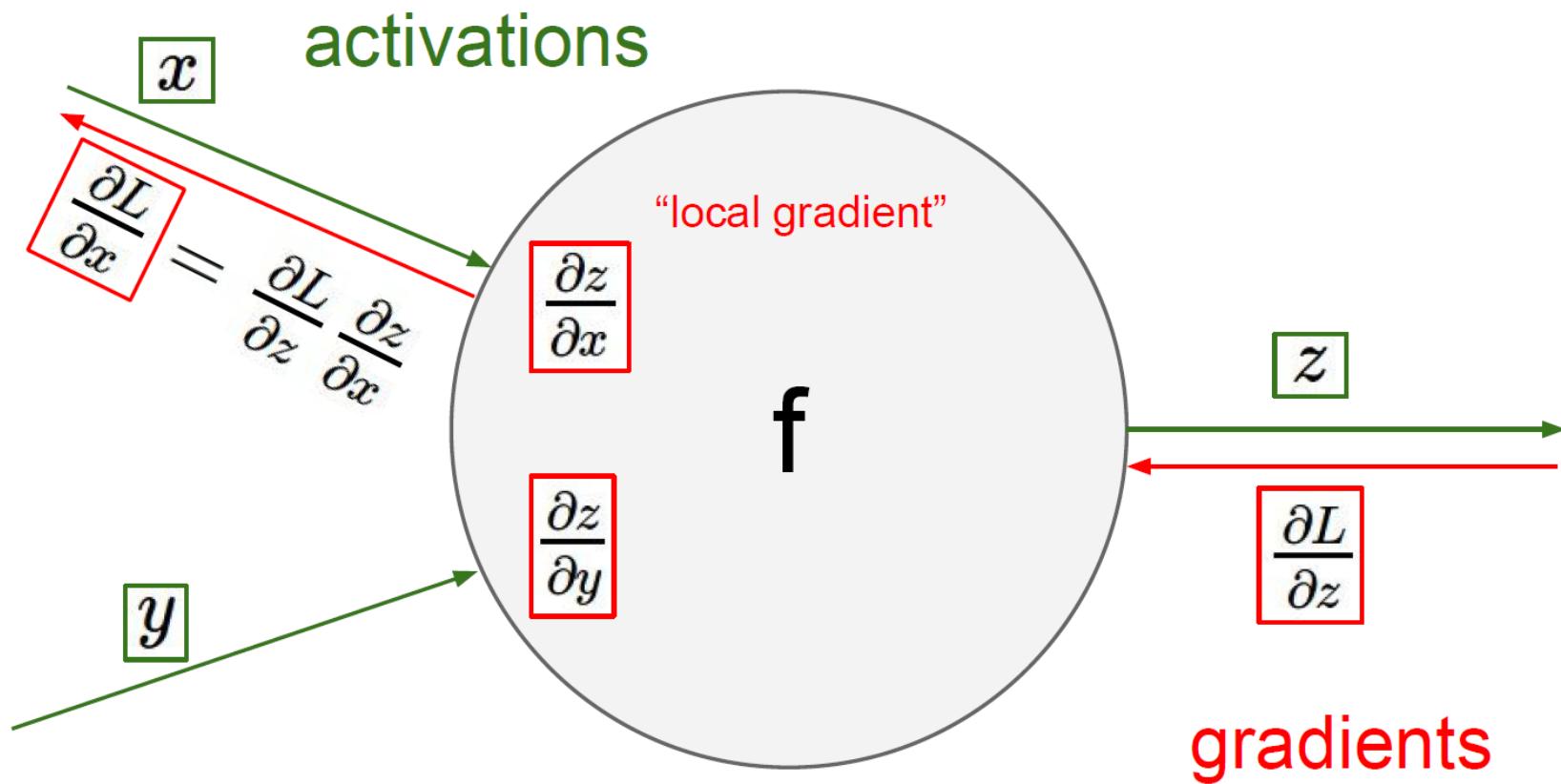
## *Backpropagation using Chain Rule*



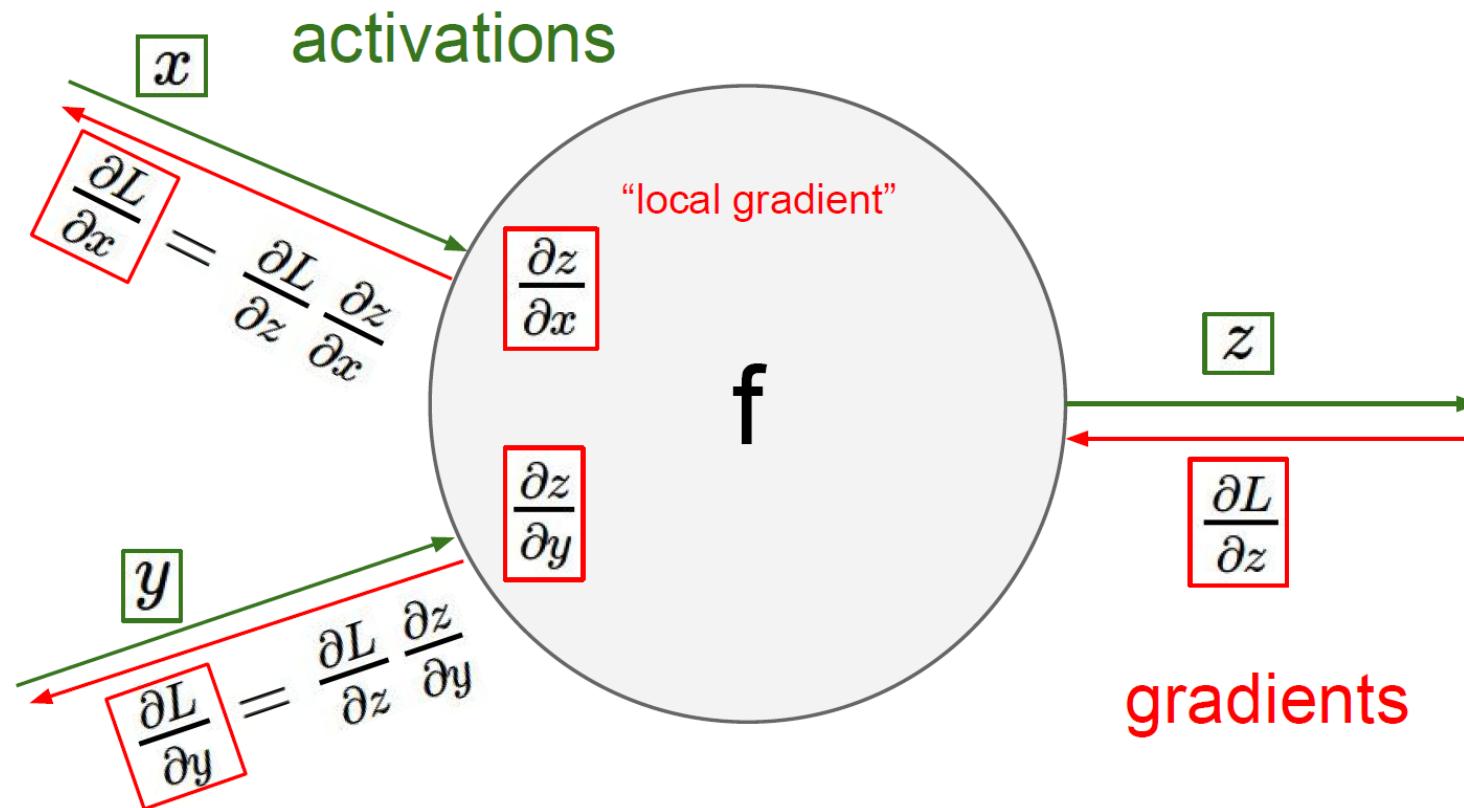
## *Backpropagation using Chain Rule*



## *Backpropagation using Chain Rule*

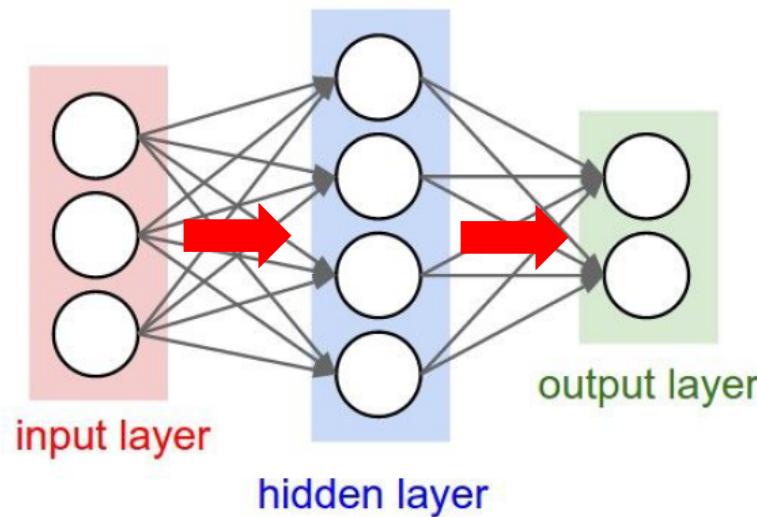


## Backpropagation using Chain Rule



# Training Neural Network

**Forward propagate activations for given input data**



x: 784 x 1  
w1: 15 x (784 + 1)  
w2: 10 x (15 + 1)

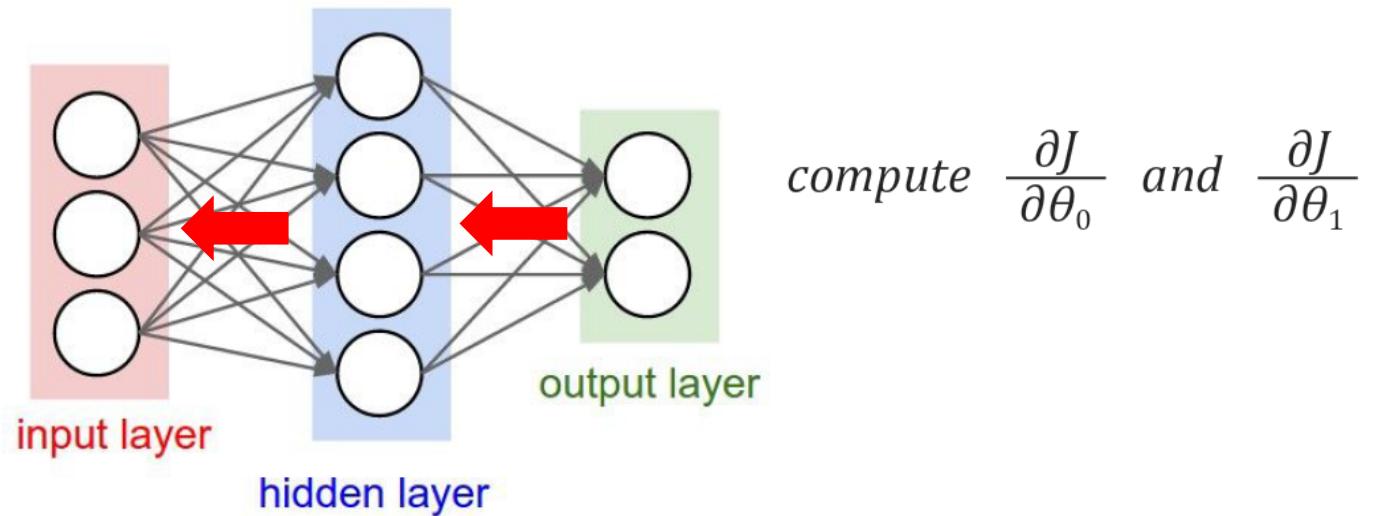
```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def neural_network(x, w1, w2):
    a1 = np.concatenate([x, [[1]]], 0)
    z2 = np.dot(w1, a1)
    a2 = sigmoid(z2)
    a2 = np.concatenate([a2, [[1]]], 0)
    z3 = np.dot(w2, a2)
    a3 = sigmoid(z3)
    return a3
```

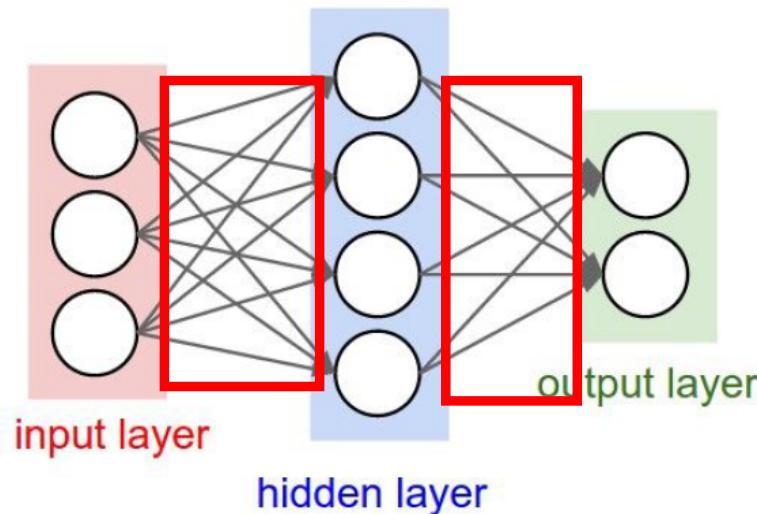
# Training Neural Network

**Backpropagate using chain rule and compute gradient of loss with respect to all weights**



# *Training Neural Network*

## **Update weights with gradient descent optimization**



*update weights with*

$$\theta_0 := \theta_0 - \alpha \cdot \frac{\partial J}{\partial \theta_0}$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial J}{\partial \theta_1}$$

## *Deep Learning Framework*



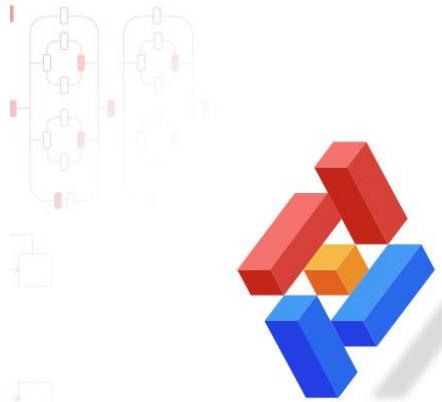
TensorFlow

K Keras

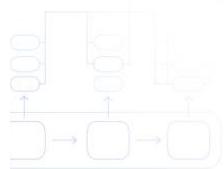
P Y T Ö R C H

m x net

# *Deep Learning Community*



TensorFlow  
Korea



Vi  
Intelligence  
Computer  
Networks

# **References**

[Convolutional Neural Networks for Visual Recognition](#)

[Machine Learning - Stanford University | Coursera](#)

[Attention and Augmented Recurrent Neural Networks — Distill](#)

[Alec Radford's animations for optimization](#)

[A Neural Network Playground](#)

[DenseCap: Fully Convolutional Localization Networks for Dense Captioning](#)