Intro to Deep Learning

(without PyTorch 😂)

DECATHLON

UNIVERSITÉ Concordia UNIVERSITY

ERICSSON

Deep Learning

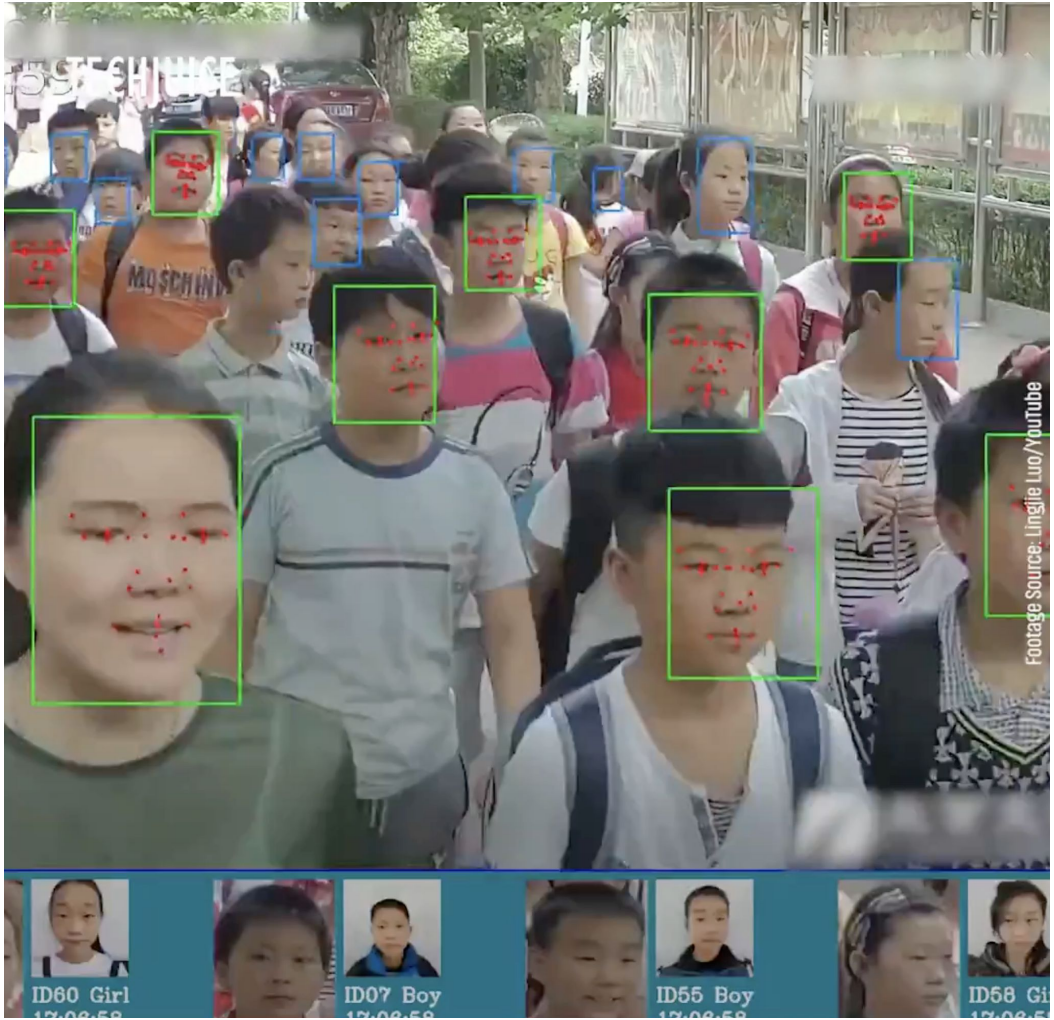
Deep learning (DL) is a *machine learning technique* that teaches computers to do what comes naturally to humans: **learn by example**. - MathWorks

[Deep] Neural Networks

The core of DL is built on **neural networks** which are developed as a *machine learning approach* to artificial intelligence (AI). - Wiki

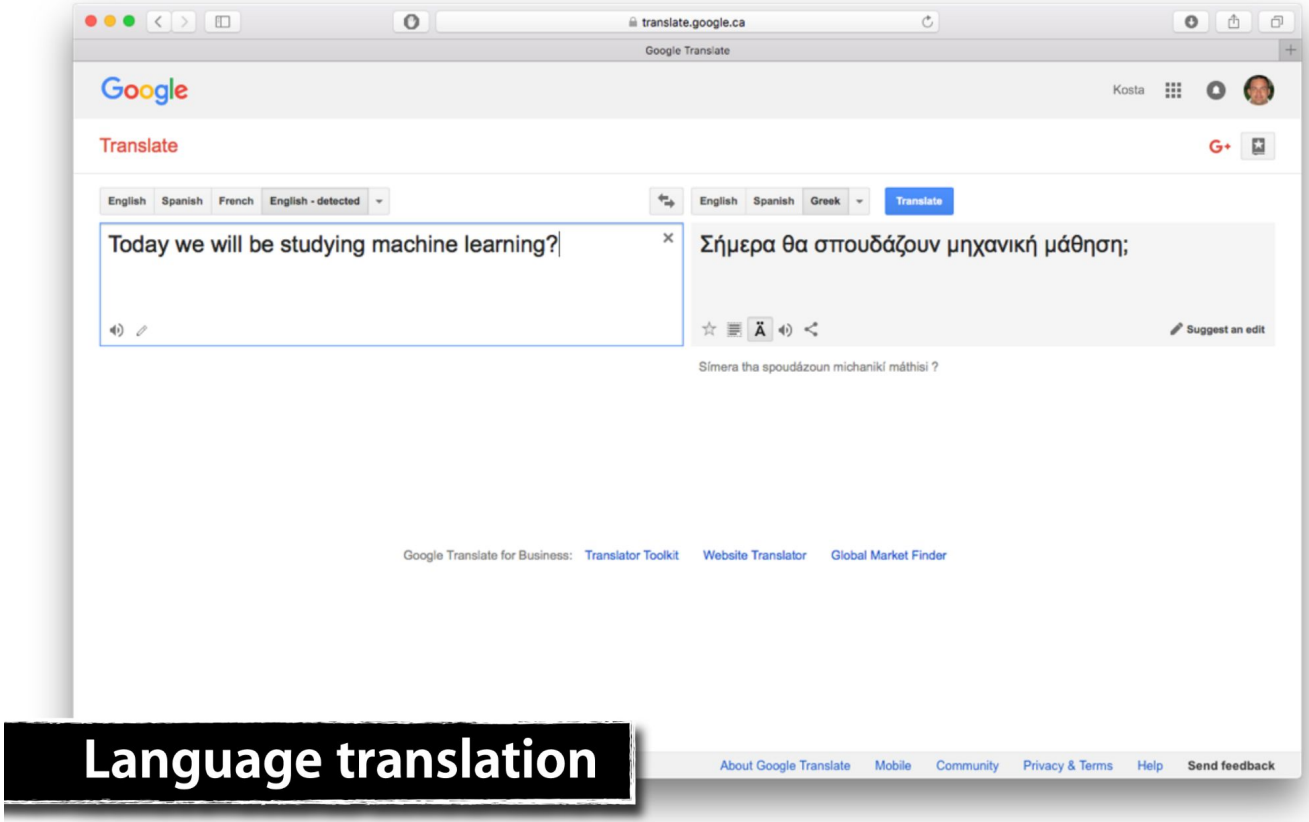
Interview: https://youtu.be/FTrQNmI3G0s

Some real world applications


ID60 Girl
ID07 Boy
ID55 Boy
Footage Source: Linglie Luo/YouTube

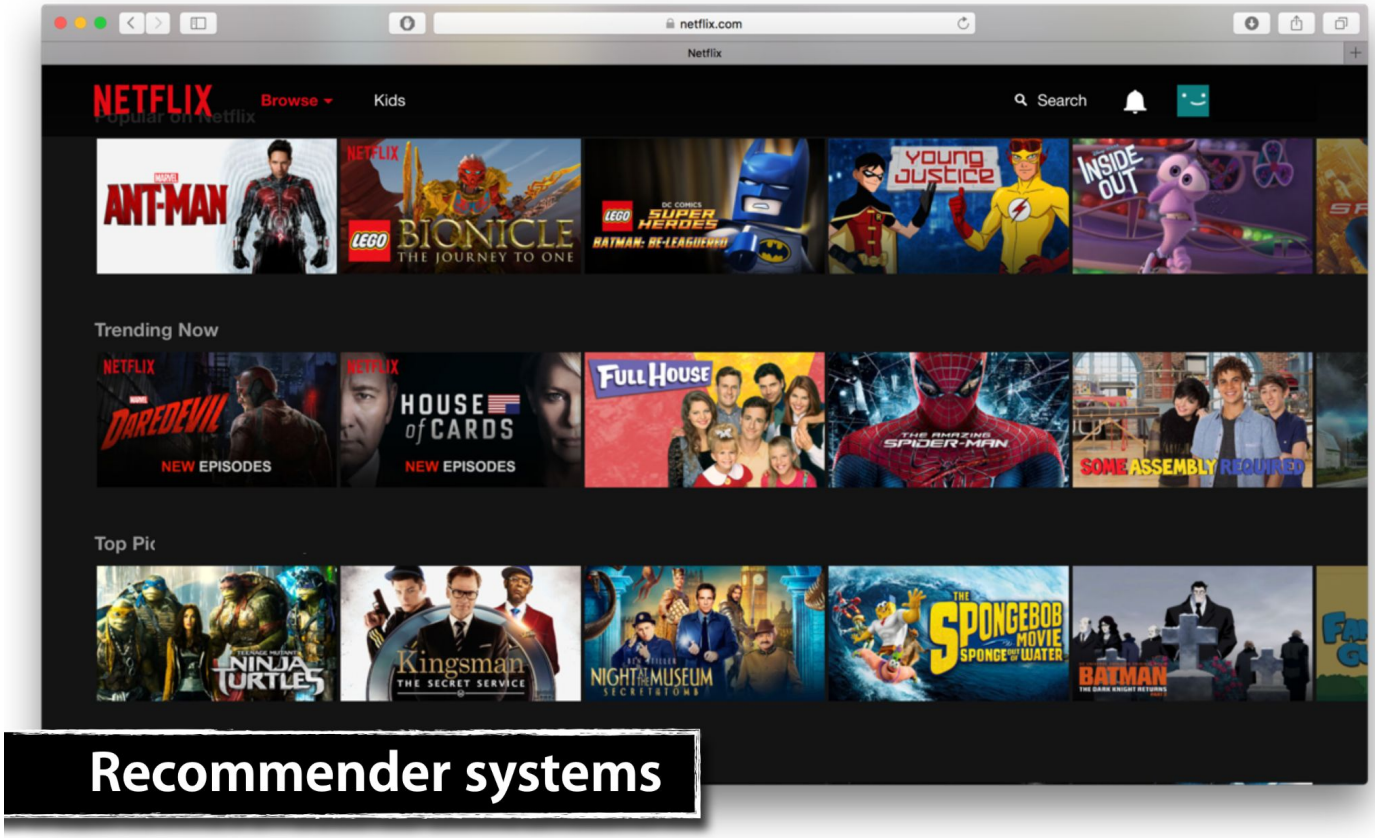

oculus
Vision-based tracking
Facebook Reality Labs and Oculus


KBA
Korea Baduk Association
Google DeepMind
Challenge Match
8 - 15 March 2016
AlphaGo defeats World Go Champion 4-1
ALPHAGO
01:58:37
AlphaGo
Lee Sedol
Google DeepMind
Challenge Match
8 - 15 March 2016



Language translation

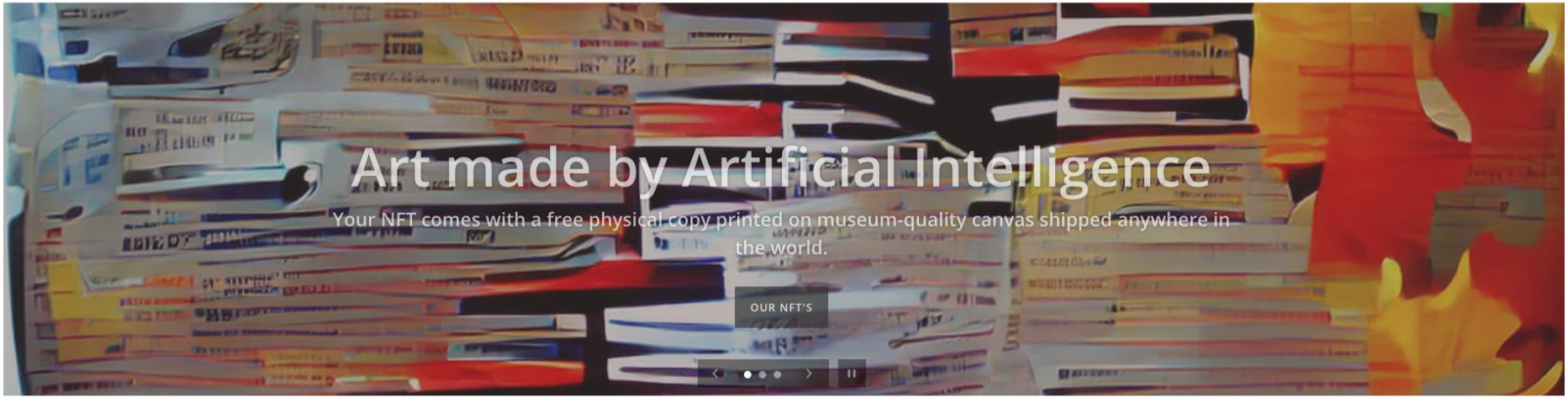
Speech recognition



Recommender systems

AIMade.Art
NFT ART ON THE BLOCKCHAIN

Home Gallery FAQ Blog Contact us

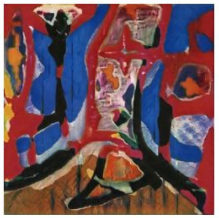



"Cubist Colors of Fall"
0.07 ETH ~~0.1 ETH~~
SALE


"de los Cuérragos"
0.1 ETH


"Ali the Duchess of Norbury"
0.1 ETH


"Sligoville"
0.07 ETH ~~0.1 ETH~~
SALE


"Muhimbasa"
0.07 ETH ~~0.1 ETH~~
SALE


"Big Blue Boy"
0.1 ETH


"The Golden Giraffes"
0.1 ETH


"Wake up Daddy, it's a Starry Safari"
0.1 ETH

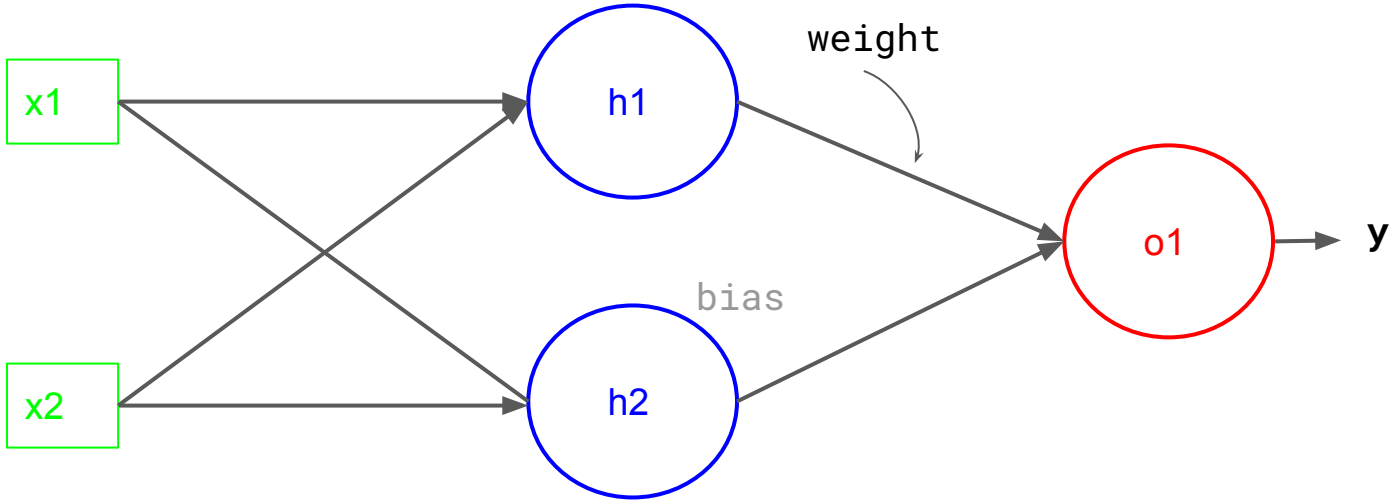
Most of these technologies (if not all!) are powered by **neural networks**

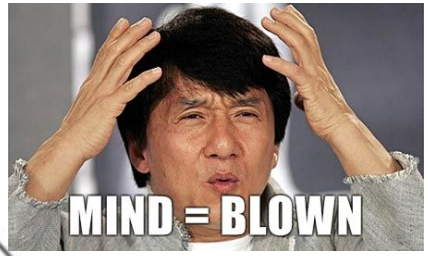
https://aiderm.herokuapp.com/
https://hasibzunair.github.io/resm3dvton/
https://github.com/hasibzunair/boss-detector

How to make them?

Anatomy of neural networks

This is a “2-2-1” sigmoid net with logistic output





x1
x2
x3
x4
x5
x6

Input Layer

6 neurons

100 neurons

500 neurons

200 neurons

50 neurons

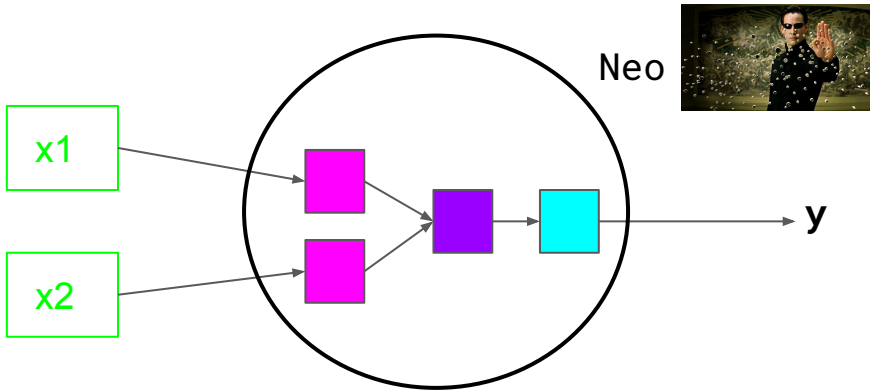
Hidden Layers

y1
y2

Output Layer

A Single Neuron

Takes **input**, does some math and produces an **output**.



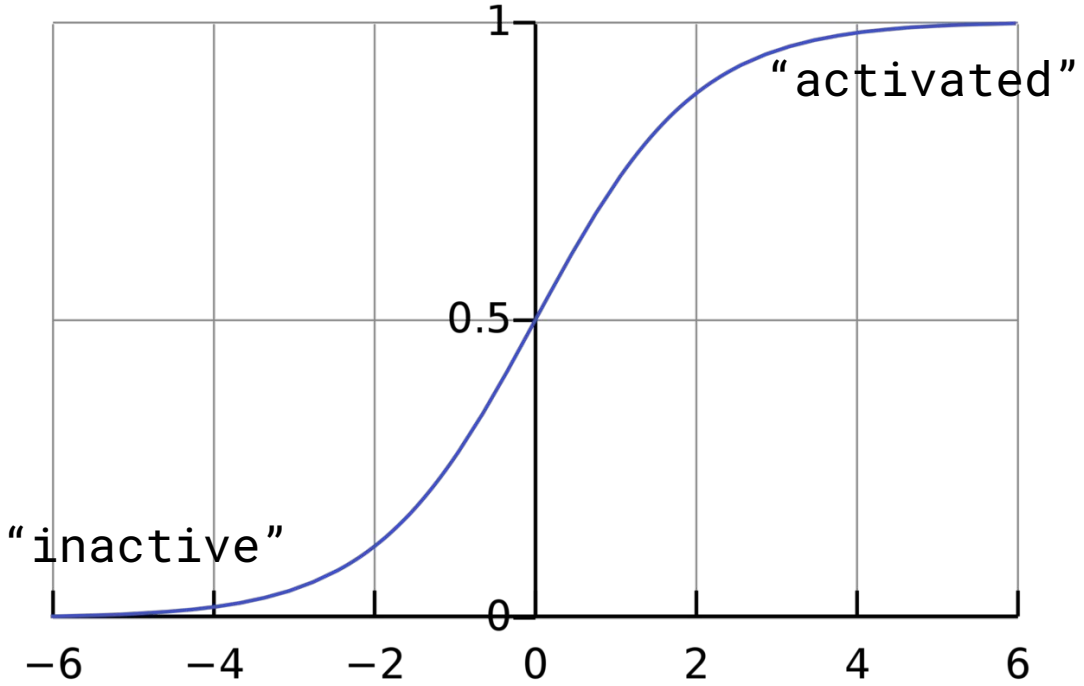
Neo can be h1, h2 or o1

“multiply”

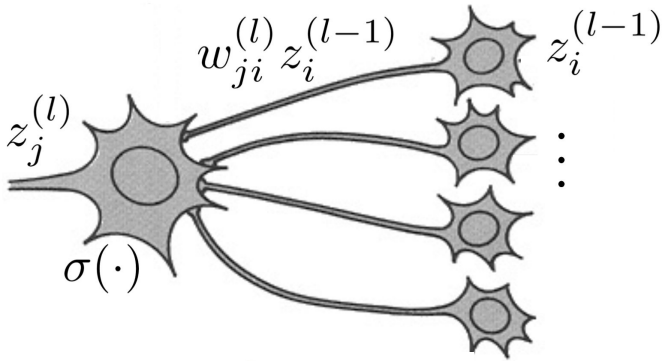
“add” bias

“activate”

Activation Function (How is Neo a neuron?)



This is a sigmoid function.



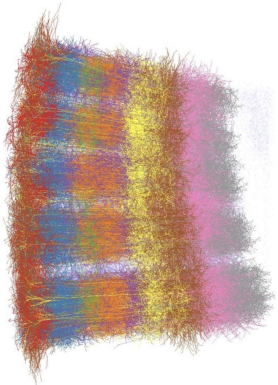


Image credit: Marcel Oberlaender

An example with one neuron (math)

$$w = [0, 1] \quad b = 4 \quad x = [2, 3]$$

"multiply weights" "add bias"

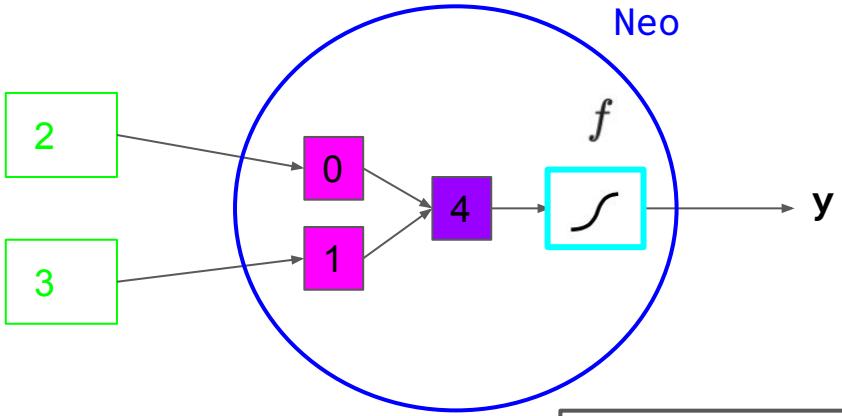
$$\begin{aligned}(w \cdot x) + b &= \boxed{((w_1 * x_1) + (w_2 * x_2))} + \boxed{b} \\ &= 0 * 2 + 1 * 3 + 4 \\ &= 7\end{aligned}$$

"activate!"

$$y = f(w \cdot x + b) = \boxed{f(7)} = \boxed{0.999}$$

This is known as the **forward pass** in a neural network.

An example with one neuron (visual)



"multiply weights" "add bias"

$$(w \cdot x) + b = \boxed{((w_1 * x_1) + (w_2 * x_2))} + \boxed{b}$$
$$= 0 * 2 + 1 * 3 + 4$$
$$= 7$$

"activate!"

$$y = f(w \cdot x + b) = \boxed{f(7)} = \boxed{0.999}$$

An example with one neuron (Code)

1

```
def sigmoid(x):
    """
    Sigmoid activation function.
    f(x) = 1 / (1 + e^(-x))
    """
    return 1 / (1 + np.exp(-x))

class BabyNeuron:
    """
    Implementation of one neuron.
    """
    def __init__(self, weights, bias):
        self.weights = weights
        self.bias = bias

    def forward(self, inputs):
        """
        Multiply inputs with "weights", then "activate".
        y = W * x + b
        """
        total = np.dot(self.weights, inputs) + self.bias
        return sigmoid(total)
```

"multiply weights"

"add bias"

"activate!"

2

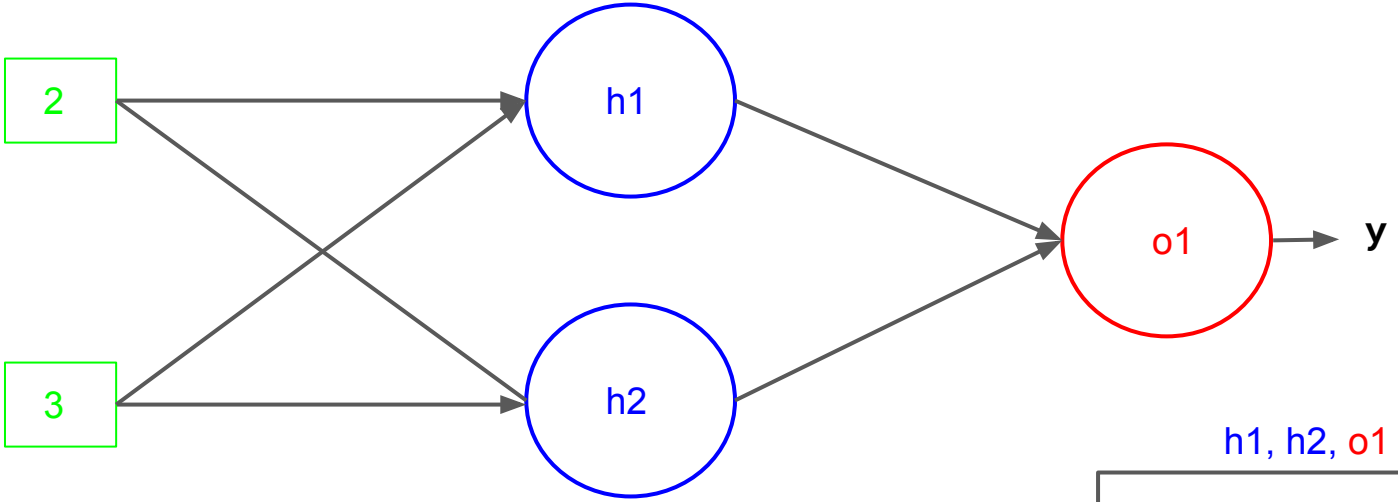
```
# Setup weights and bias (W and b)
weights = np.array([0, 1]) # w1 = 0, w2 = 1
bias = 4                   # b = 4

# Create our baby neuron
babynet = BabyNeuron(weights, bias)
```

3

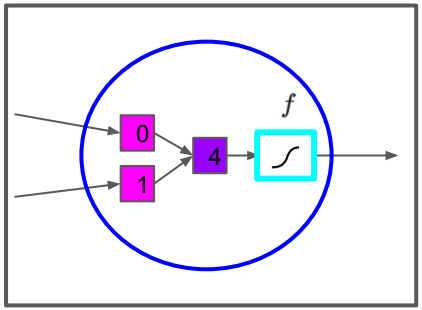
```
# Set inputs, x1 = 2 and x2 = 3
x = np.array([2, 3])

# Pass input x to babynet
print("Output: {:.5f}".format(babynet.forward(x)))
# Should get 0.99908
```

```
Output: 0.99909
```

A small neural net (visual)



h1, h2, o1



A small neural net (math)

$$w = [0, 1] \quad b = 0 \quad x = [2, 3]$$

3 neurons (h1,h2,o1)
with same weight

$$\begin{aligned} h_1 = h_2 &= f(w \cdot x + b) \\ &= f(\boxed{(0 * 2) + (1 * 3)} + \boxed{0}) \\ &= \boxed{f(3)} \\ &= 0.9526 \end{aligned}$$

"multiply weights" "add bias"

"activate!"

$$\begin{aligned} o_1 &= f(w \cdot [h_1, h_2] + b) \\ &= f(\boxed{(0 * h_1) + (1 * h_2)} + \boxed{0}) \\ &= \boxed{f(0.9526)} \\ &= \boxed{0.7216} \end{aligned}$$

A small neural net (code)

1

```
class BigBaby:
    """
    A "network" with:
     - 2 inputs
     - a hidden layer with 2 neurons (h1, h2)
     - an output layer with 1 neuron (o1)

    Each neuron (i.e. BabyNeuron) has the same weight and bias:
     - w = [0, 1]
     - b = 0
    """
    def __init__(self):
        # We initialize using same weights and bias for each neuron
        weights = np.array([0, 1])
        bias = 0
        # BabyNeuron class from previous section
        self.h1 = BabyNeuron(weights, bias)
        self.h2 = BabyNeuron(weights, bias)
        self.o1 = BabyNeuron(weights, bias)

    def forward(self, x):
        out_h1 = self.h1.forward(x)
        out_h2 = self.h2.forward(x)
        # The inputs for o1 are the outputs from h1 and h2
        out_o1 = self.o1.forward(np.array([out_h1, out_h2]))
        return out_o1
```

2

```
# Create our BigBaby network
network = BigBaby()

# Set inputs, x1 = 2 and x2 = 3
x = np.array([2, 3])

# Pass input x to network and make predictions
print("Output: {:.5f}".format(network.forward(x)))
# Should get 0.72163
```

```
Output: 0.72163
```

For **each** neuron: “multiply weights” “add bias” “activate!”

Training a neural network

- Neural networks are trained by **gradient descent**
- Gradients are computed by a special algorithm called “backpropagation”
- “Backpropagation” is an *efficient algorithm* to *compute derivatives*

https://en.wikipedia.org/wiki/Backpropagation

Stochastic Gradient Descent (SGD)

lower loss = better predictions

- Update **parameters** of neurons such that it **minimizes** a cost/error/loss/objective function.

- Loss is used to quantify how “good” network is

- Requires:
  - Current weights (and biases)
  - Loss
  - Gradient with respect to loss (backprop)

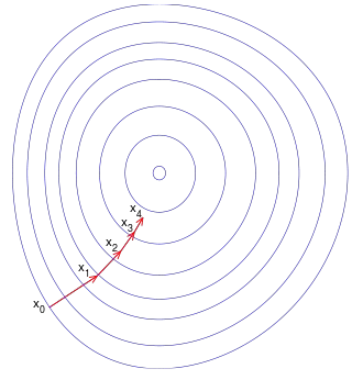


Image credit: Wiki

https://en.wikipedia.org/wiki/Stochastic_gradient_descent

Stochastic Gradient Descent

Gradient Descent

Move towards negative of the gradient in order to “minimize” the loss

$$w_1 \leftarrow w_1 - \boxed{\eta} \frac{\partial L}{\partial w_1}$$

learning rate

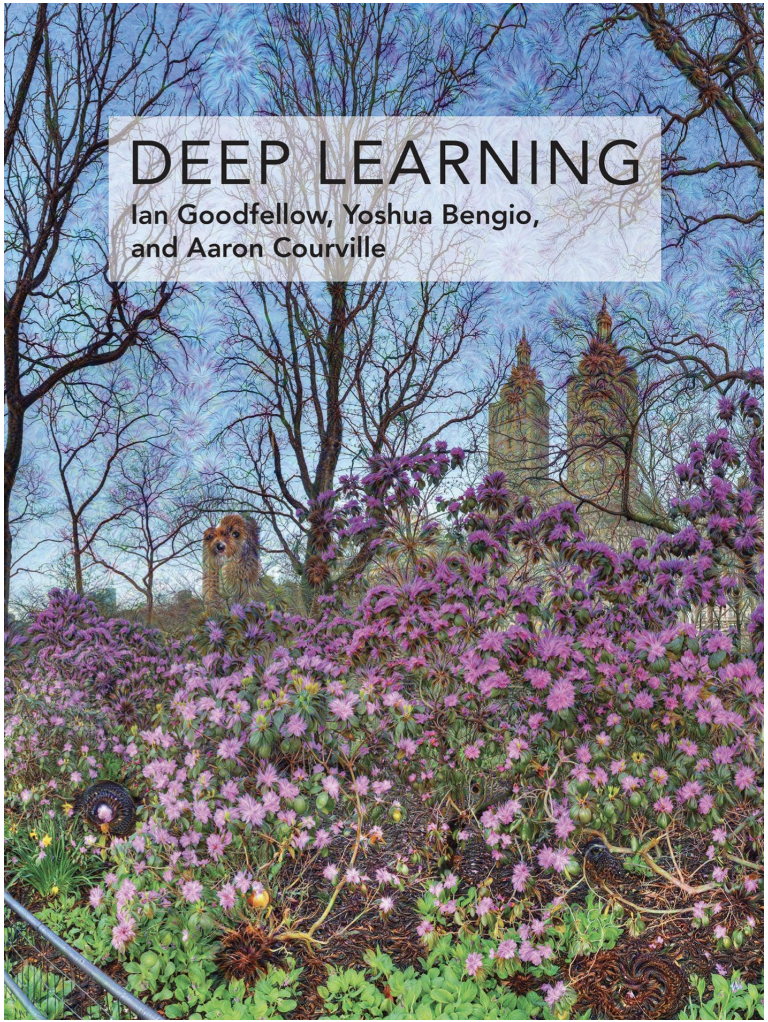
How a neural network learns?

- Neural net makes a “guess” (**forward pass / make predictions**)
- Compute how good the *guess* (prediction) is against the *known correct answer* (ground truth) (**loss**)

lower loss = better predictions

- Update weights so that **guess** is better by minimizing the loss (**gradient descent**)
- Do until certain number of steps or loss is minimal

Let’s see an example!


DEEP LEARNING
Ian Goodfellow, Yoshua Bengio,
and Aaron Courville


WERE
FINISHED