#### Reference: 3Blue1Brown video

# **AVRIL CUI**

# print("Neural Network Structure")

**Content**:

- 1. Definition of a neuron
- 2. Neural network layers analogy
- 3. Weights
- 4. Activation functions
- 5. Gradient Descent

## Pause! What Do You See?



How do you "KNOW" it is seven? In other words, what is your proof?

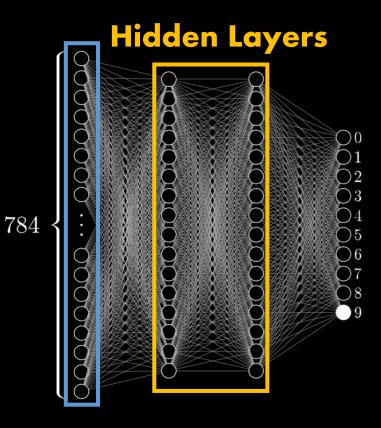
But how can the computer tell? Difficulty upgrade!





## A Neuron = placeholder for a number

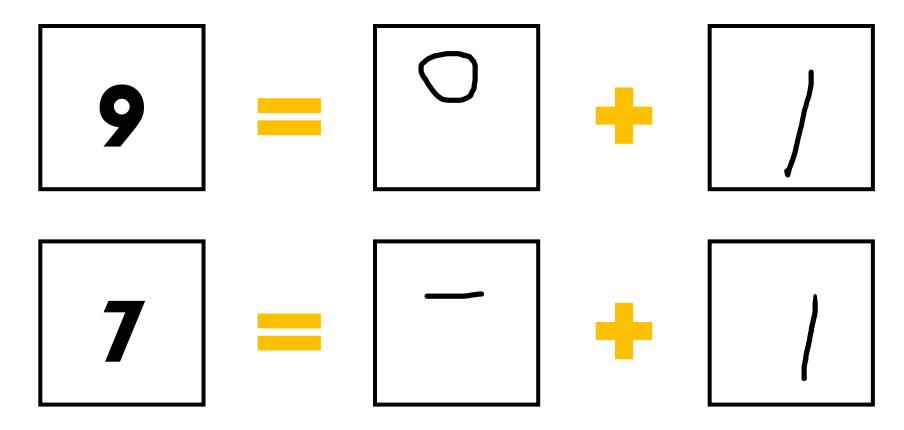
0.0 (	0.0 (	00 (i	0.0	0.0	6.0	0.0	00	0.0	0.0	0.0	0.0	00	0.0	0.0	0.0	0.0	0.0	6.0	0.0	0.0	0.0	0.0	0.0	@0	0.0	0.0
0.0	0.0	0.0 6	0.0	0.0	0.0	0.0	00	<u>()</u>	00	0.0	(n)	(n)	0.0	0.0	0.0	0.0	0.0	<u>(</u> )	00	<u>()</u>	<u>(</u> )	<b>(</b> )	(ii)	<u>()</u>	0.0	<u>.</u>
0.0		00 6	0.0	0.0	0.0	(a)	<u>(0)</u>	00	<u>(</u> )	0.0	00	00	0.0	0.0	0.0	6.0	0.0	(iii)	<u>(0)</u>	<b>(</b> )	<u>(</u> )	<b>(</b> 0)	(ii)	60	0.0	<u>.</u>
0.0	) (III	) (i	00	0.0	0.0	ũ	ŏ	ũ	ŏ	ŏ	(III)	(III)	(0.0)	0.0	0.0	<u>(0.0)</u>	<u>(0.0)</u>	ŏ	ŏ	(III)	ũ	(II)	ũ	ŏ	(0.0)	<u>a</u> o
0.0	) (III	0.	00	$\widetilde{0}$	0.0	(iii)	ŏ	õ	ŏ	ŏ	(III)	ě	(0.0)	(0,0)	0.0	$\tilde{(0)}$	(0,0)	ŏ	ŏ	$\widetilde{(1)}$	ŏ	(III)	(III)	ŏ	(0.0)	0.0
0.0	) (III	0.0	00	0.0	0.0	(a)	ŏ	(III)	(III)	(iii)	(III)	(1)	(0.0)	0.0	0.0	(0,0)	(0,0)	ŏ	ŏ	$\widetilde{(1)}$	õ	(i i)	(III)	ě	(0.0)	<u>00</u>
0.0	) (III	0.) (i	00	0.0	0.0	(iii)	ě	(III)	(III)	(II)	(II)	(II)	(0.0)	(0,0)	0.0	$\tilde{(0)}$	0.0	ŏ	ŏ	(III)	õ	(iii)	(III)	ŏ	(iii)	0.0
0.0)	) (	0.0 6	000	0.0	6.0	0.0	ŏ	() ()	(0.0)	0.0	(0.2)		0.8	1.0	1.0	ŏ	()	ŏ	$\tilde{(2)}$	0.0	(0.0)	0.0	(0.0)	ŏ	0.0	0.0
(0.0)	.) (	0.0 6	0.0	0.0	6.0	()		$\simeq$	$\simeq$	-	_	1.0	1.0	6	0.8	1.0	1.0	1.0	ŏ	(0.0)	(0.0)	0.0	(U.U)	ŏ	(0.0)	<u>()</u>
0.0	0.0) (	0.0 6	0.0	0.0	6.0	()		() ()	(2)	0.9	1.0		(0.3)	(0.0)	(0.0)		1.0	1.0	(15)	(0.0)	(0.0)	(0.0)	(0.0)	(iii)	0.0	<u>()</u>
0.0		0.0 6	000	0.0	6.0	(.) (.)													-	(iii)	$\tilde{(0)}$	(0.0)	(0.0)	(iii)	(iii)	6.0
0.0		0.0 6.	0.0	0.0		0.0	-		_	_							1.0		~	$\simeq$	×	(0.0)	(iii)	ŏ	(iii)	6.0
0.0	) (	0.0 6	0.0	0.0	0.0	ŏ	1.0	1.0	0	(0.0)	(in)							-	ŏ	<u>()</u>	(0.0)	(0.0)	(0.0)	ŏ	().()	<u>60</u>
0.0	) (0.0	006	00	0.0	(I)	0.9	1.0	0	(0.0)	(i)	õ	ŏ	ŏ	1.0	1.0	0	0.0	ŏ	ŏ	õ	$\check{0}$	$\check{0}$	ŏ	ŏ	() ()	õ
0.0	) (10	00 6	00	0.0	(i)	1.0	1.0	0.1	0.1	č	ŏ	0.9	1.0	1.0	1.0	63	0.0	ŏ	<u>(0)</u>	(i)	õ	0.0	(ii)	ŏ	0.0	õ
0.0	) (III	00) (0	00	0.0	01	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.9	1.0	0.9	0.0	(0,0)	ŏ	ŏ	(III)	õ	(iii)	(1)	ŏ	(iii)	<u>0</u> 0
0.0	) (III	0.0	00	0.0	(0,0)	(0.2)	0.9	1.0	0.5		(0.2)	(0.1)		1.0	66		<u>(0,0)</u>	ŏ	ŏ	(III)	õ	(III)	(1)	ŏ	(0.0)	õ
0.0	) (III	a.) (a.	00	0.0	0.0	(iii)	(11)	(11)	(0.0)	(1.1)	(11)	(0.2)	1.0	1.0	0.1	$\widetilde{(0,0)}$	0.0	ŏ	õ	(III)	õ	(iii)	(III)	ě	(iii)	<u>a</u> o
0.0	0.0)	0.0 6.	0.0	0.0	0.0	0.0	(iii)	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.1	0.0	0.0	$\widetilde{0}$	0.0	<u>(0.0)</u>	(0.0)	0.0	0.0	(0)	0.0	0.0
0.0	0.0)	0.0 6.	00	0.0	0.0	0.0	õ	0.0	0.0	0.0	0.0	(1)	1.0	1.0	$\tilde{(0,1)}$	0.0	0.0	<u>(</u> )	0.0	0.0	(iii)	0.0	(0.0)	00	0.0	0.0
0.0	0.0	0.0 6.	00	0.0	0.0	0.0	õ	(iii)	0.0	0.0	0.0	0.3	1.0	1.0	(). ().	6.0	0.0	(iii)	0.0	0.0	0.0	0.0	(0.0)	õ	0.0	0.0
0.0	0.0	0.0 6.	00	0.0	6.0	0.0		$\tilde{(0)}$	$\tilde{(0)}$	0.0	0.0		1.0	1.0	())	6.0	0.0		0.0	0.0	0.0	0.0	0.0	ŏ	0.0	0.0
0.0	.) (0.0	0.0 6.	00	0.0	0.0	0.0	$\widetilde{(0)}$	(iii)	0.0	0.0	0.0	(0.2)	1.0	1.0	(1)	0.0	0.0		ũ	(a.0)	(0,0)	0.0	(0.0)	ŏ	().)	$\tilde{0.9}$
0.0	õ.)	0.0 6.	0.0	0.0	0.0	(i.i)	ŏ	$\widetilde{(0)}$	$\check{0}$	$\check{0.0}$	0.0	0.0	0.9	1.0	$\check{(2)}$	ŏ	0.0	ŏ	ŏ	(0.0)	$\widetilde{0.0}$	0.0	$\widetilde{(0,0)}$	ŏ	(i.i)	õ
0.0	.) (0.0)	0.0 6	00	0.0	0.0	(i.i)	ŏ	$_{0}$	$\check{0}$	$\check{0}$			0.1			$\check{(2)}$	0.0	ŏ	ŏ	$\widetilde{(0)}$	$\check{0}$	(0.0)	$\widetilde{(0)}$	ŏ	().()	$\tilde{0.0}$
0.0	) (0.0	0.0 6	000	0.0	<u>(0.0</u>	$\simeq$	$\simeq$	$\simeq$	$\simeq$	$\simeq$	$\simeq$		(0.1)	_	-	~	0.0	$\simeq$	$\simeq$	$\widetilde{(0)}$	$\widetilde{(0)}$	<u>(00)</u>		ŏ	(i)	<u>60</u>
(0.0)	0.0) (	00 6	00		í)	) (iii)	ŏ	õ	(0)	$\simeq$	$\simeq$	$\simeq$	$\simeq$	(0.1)	_	-	$\simeq$	$\simeq$	$\simeq$	õ	$\widetilde{(0)}$	(0)	õ	ŏ	60	<u>(</u> )
0.0	$\simeq$			$\simeq$	$\simeq$	) (II)	$\simeq$	) (ii)	$\simeq$	$\simeq$	$\simeq$	$\simeq$	()	$\simeq$	_	_	0	$\simeq$	$\simeq$	$\simeq$	$\simeq$	$\simeq$	$\simeq$	ĩ	$\simeq$	) (1)
0																										



# Network Layers

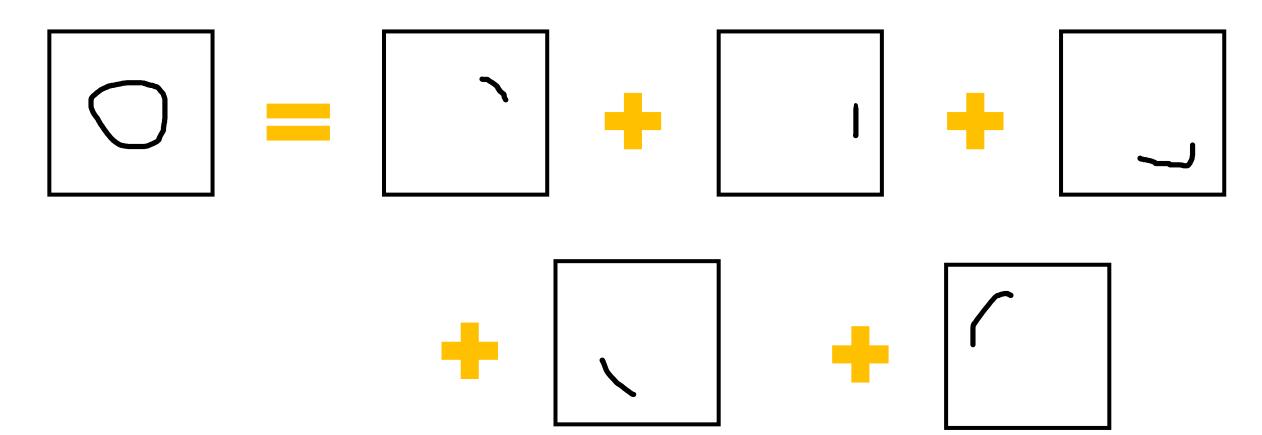
## **Before Going Forward...**

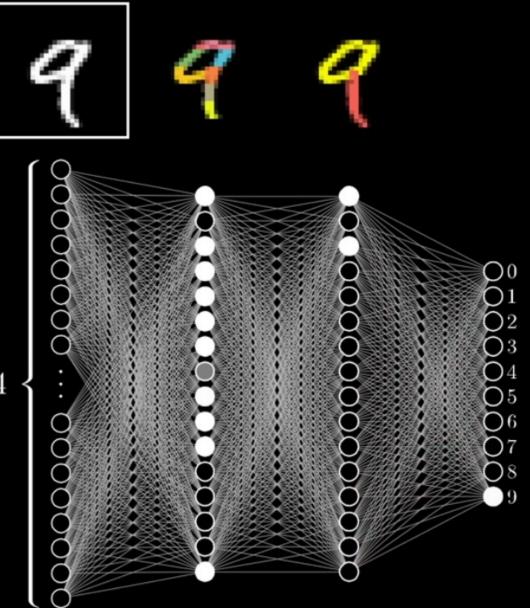
#### What does these layers do???



# And how to recognize patterns?

#### Well, smaller edges

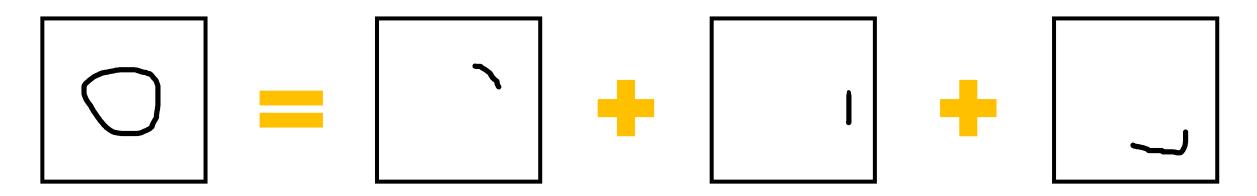




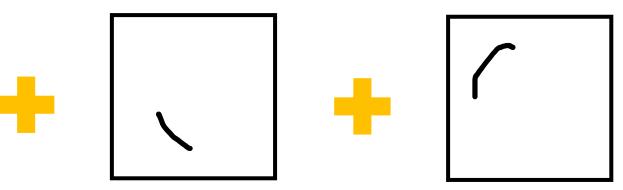


# And how to recognize patterns?

#### Well, smaller edges

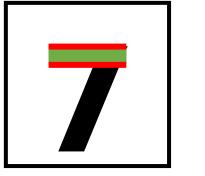


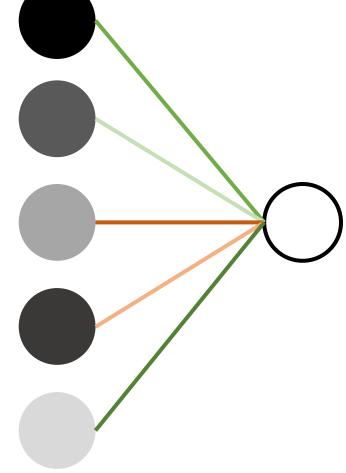
\*Although this is not exactly how neural networks "learn," this is an intuitive (or human method) way to visualize the layers.



# Weights & Parameters

## Weights and Parameters





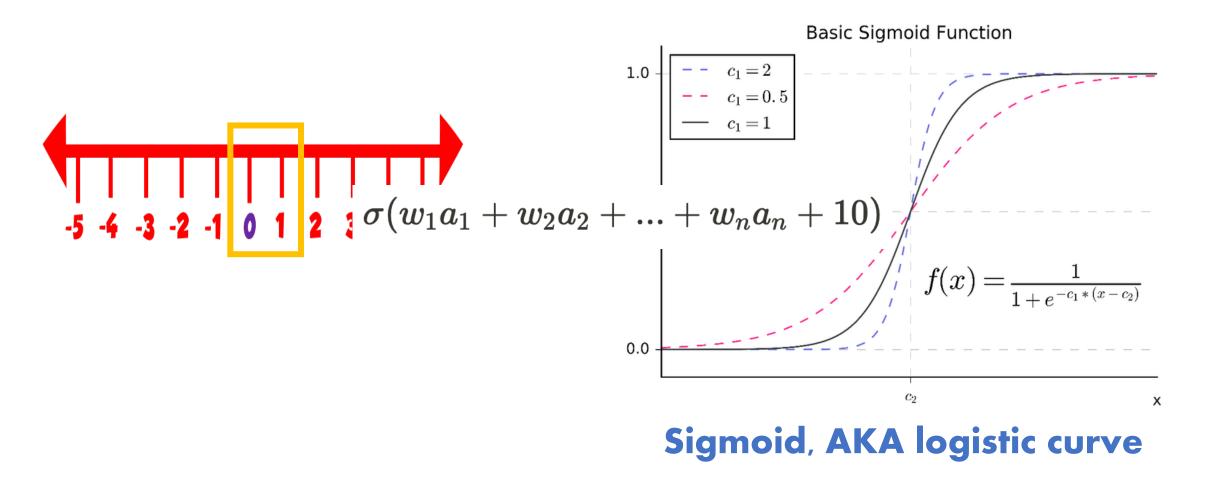
 $w_1a_1 + w_2a_2 + \ldots + w_na_n$ 

Ideally, we want the pixels (neurons) at the region to be highly positive and all the rest to be zero!

Even better, if we want an edge around the region, then we want those respective neurons to be negative.

# Activation Functions

## **Standardized Output**



Bias



# Hi neuron, Please light up only if the weighted sum is greater than ten!

# $\sigma(w_1a_1+w_2a_2+...+w_na_n+10)$ "Bias"

And... This is just one neuron! All 784 neurons in our example have weights and biases. This resulted in 13,002 parameters!

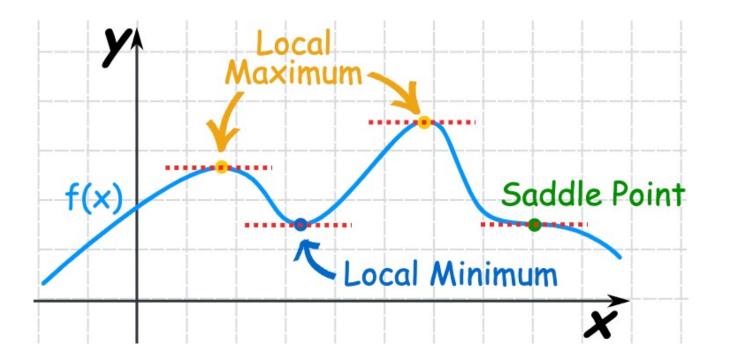


#### Find the right (most optimal) weights and biases.

# Gradient Descent

# **Everything is CALCULUS**!

#### Finding maximum/minimum



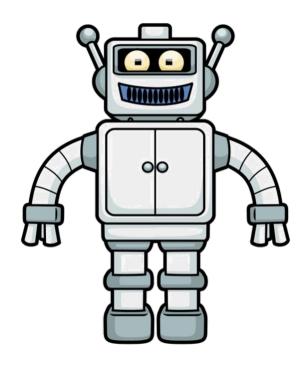
### **Cost Function: the Error**

What we tried to minimize

$$rac{\sum_{i=0}^n (x_{p,i}-x_{a,i})^2}{n}$$

# How exactly does this work?

- 1. Prepare a training set (large!) with labels (supervised)
- 2. Initialize weights and biases randomly
- 3. Calculate the cost
- 4. Use gradient descent to start minimizing (decreasing the cost

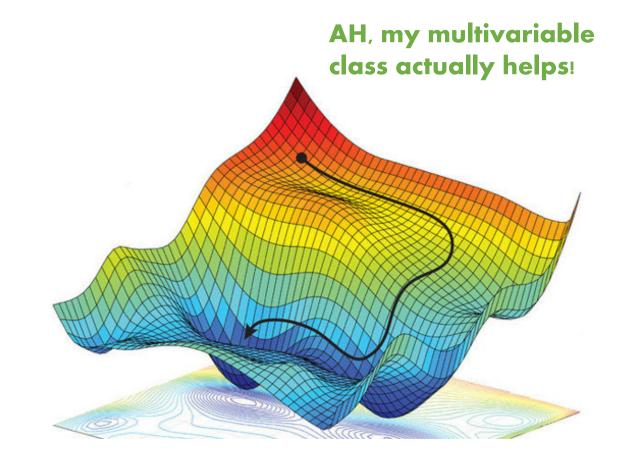


Well, it's easy to find the minimum point of a two-dimensional function. But what do we do if it's 13002 dimensions?

## **Gradient Descent**

Core idea: the function decreases the fastest at the direction of the negative gradient!

$$abla f(x,y,z) = < f_x, f_y, f_z >$$

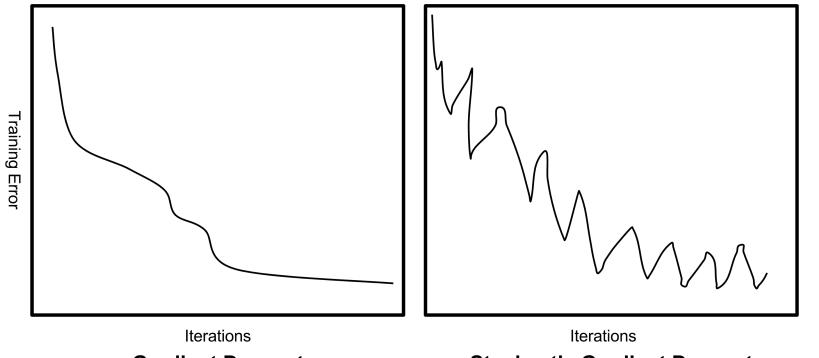


# Simple Algorithm

```
def simple_gd(gradient, start, learn_rate, n_iter=50, tolerance=1e-06):
    .....
    .....
    vector = start
    vector_lst = [start]
    for _ in range(n_iter):
        vector -= learn_rate * gradient(vector)
        vector_lst.append(vector)
        if abs(learn_rate * gradient(vector)) <= tolerance:</pre>
            break
    return vector, vector_lst
```

## **Stochastic Gradient Descent**

#### More efficient!



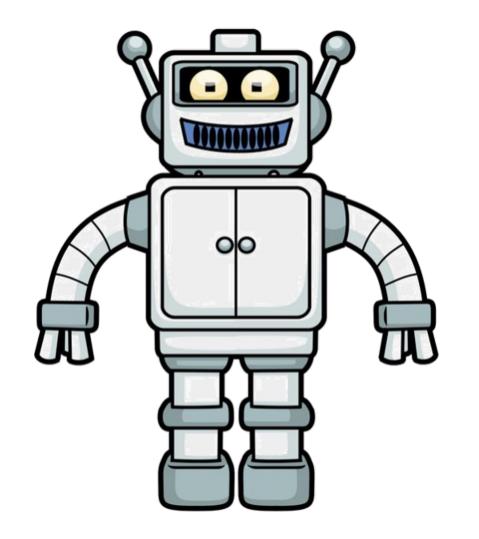
A drunk man walking down a hill... But faster

**Gradient Descent** 

**Stochastic Gradient Descent** 

# **Next Steps**

- 1. Calculate gradient (partial derivatives): Backpropagation
- 2. Understanding what actually happened in the learning process
- 3. Understanding stochastic gradient descent and learning rate's role



# Thank you!