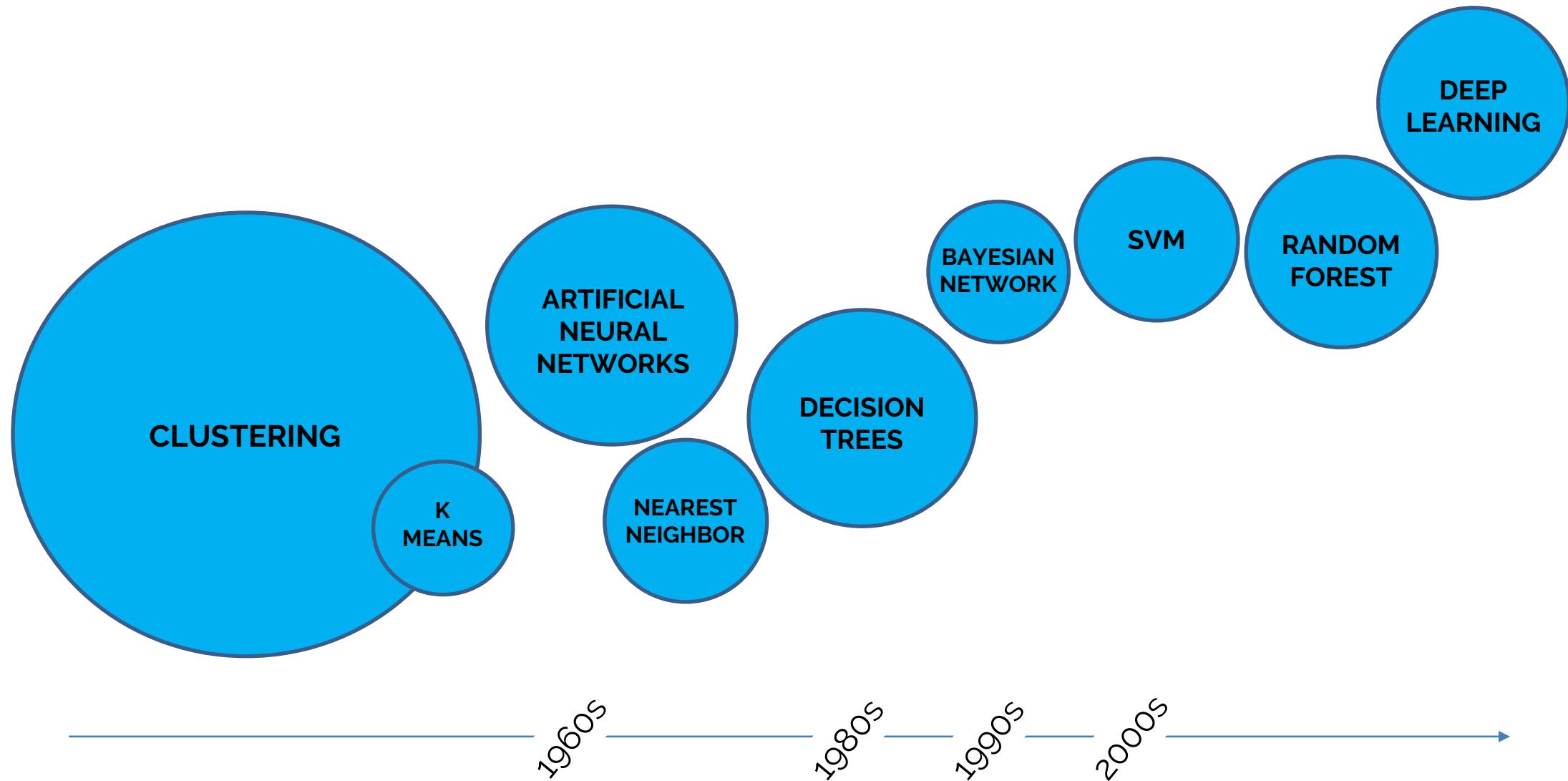


Machine Learning: Basi e Sue Applicazioni

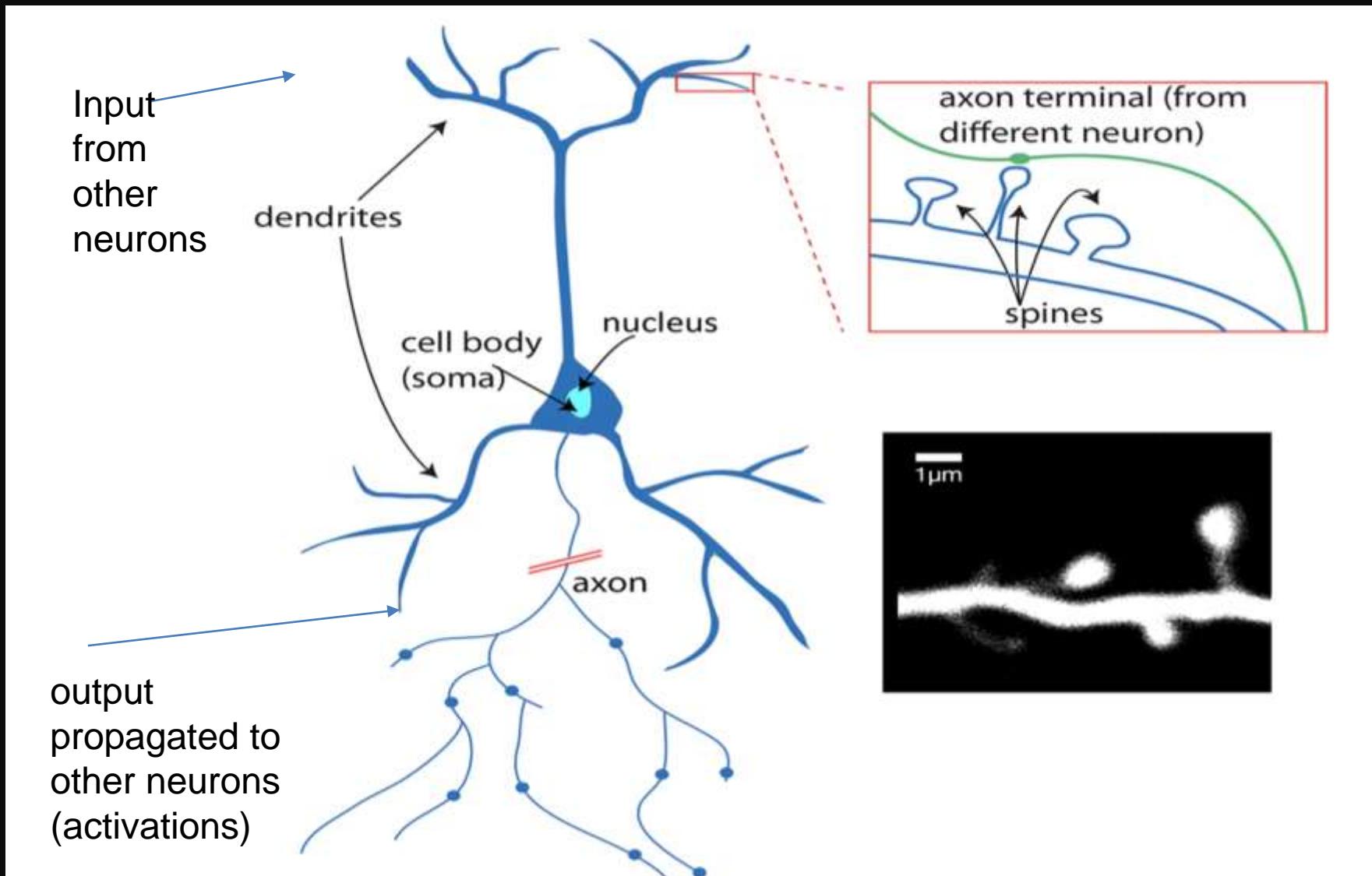
Christian Salvatore
Scuola Universitaria Superiore IUSS Pavia

Machine learning



NEURAL NETWORKs

A Biological Neuron



A Human-Vision Task

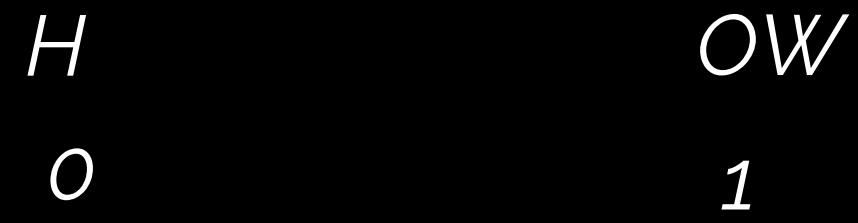
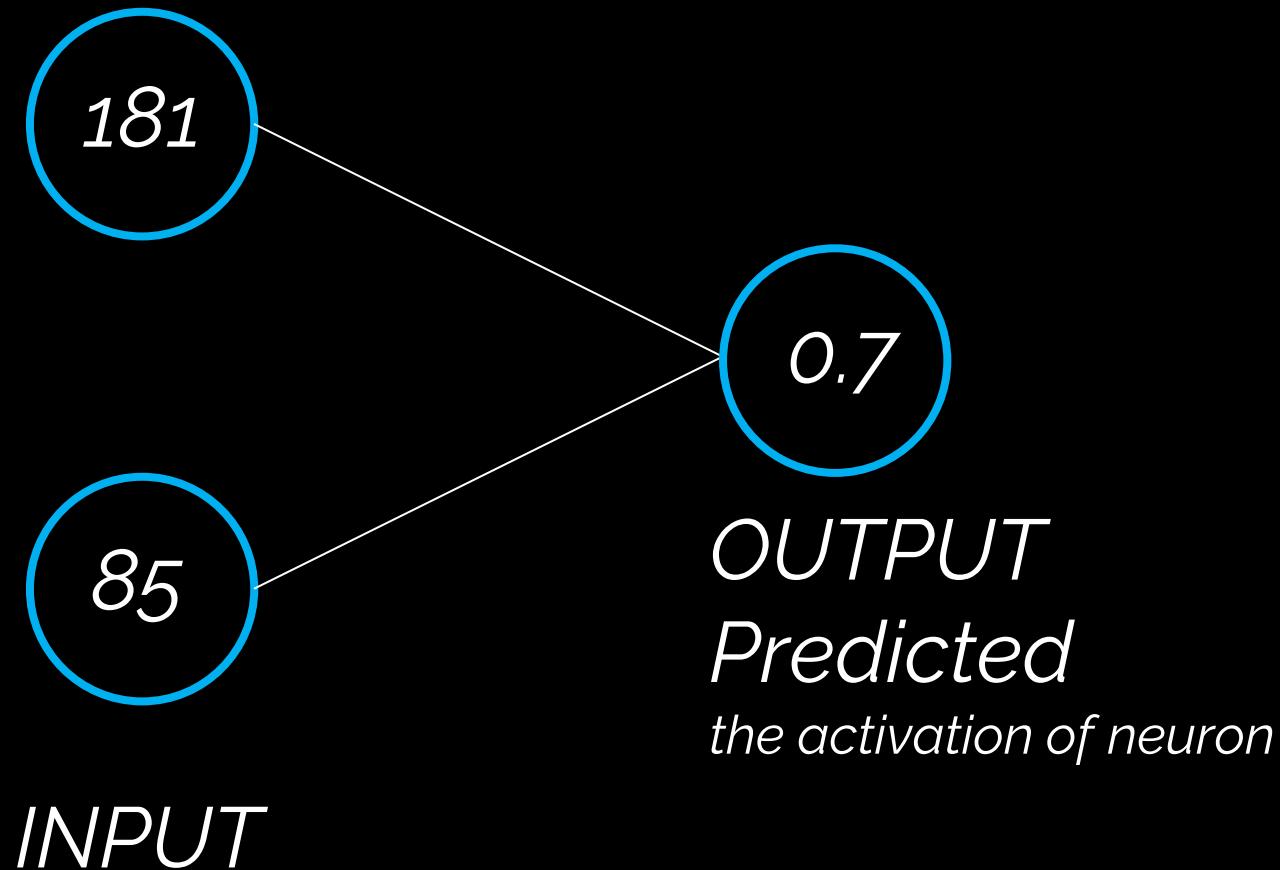
'healthy' vs overweighting men

| H (cm) | 181 | 184 | 172 | 160 | 170 | 187 | 184 | 176 | 190 |
|---------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| AC (cm) | 85 | 94 | 102 | 80 | 98 | 110 | 116 | 77 | 84 |

H
O
—
OW
1

Neural Network

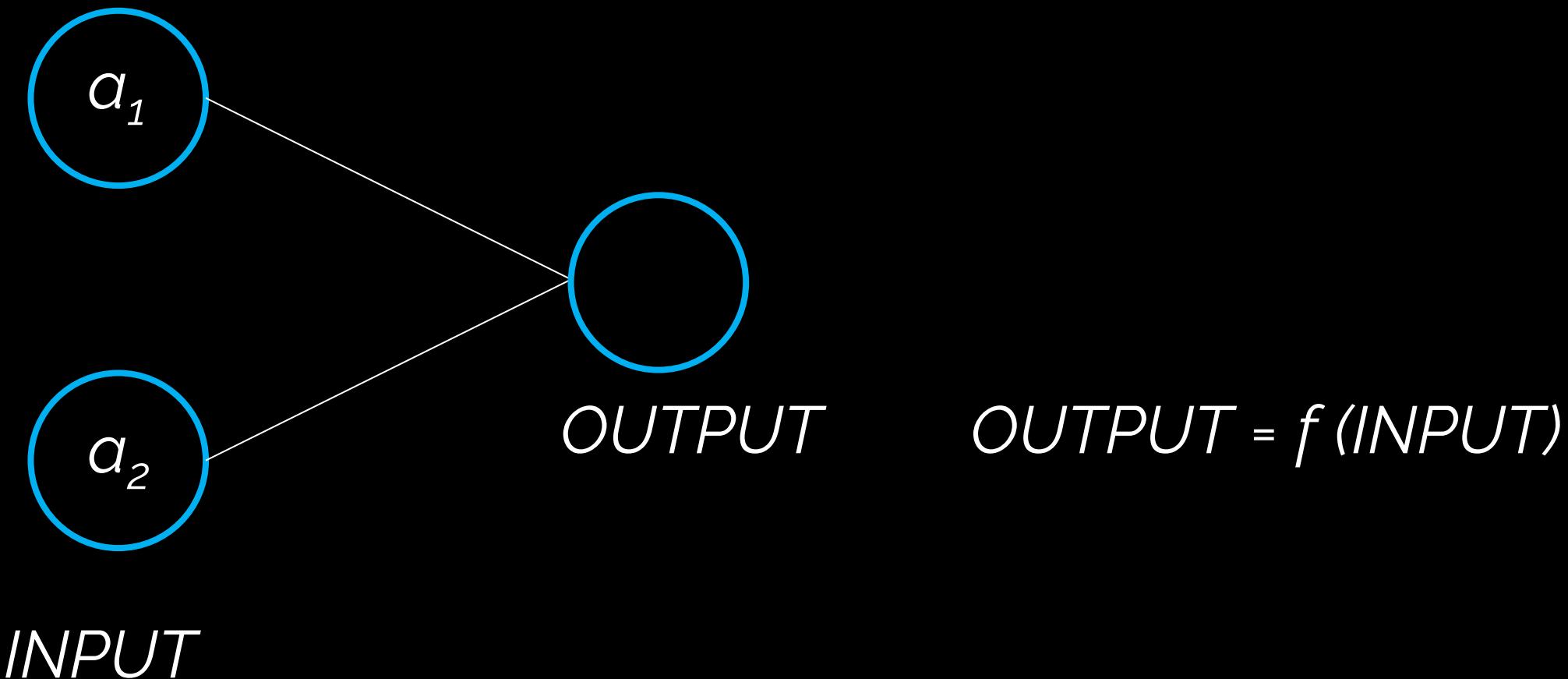
$$H = a_1 \quad AC = a_2$$



Neural Network

$$H=a_1$$

$$AC=a_2$$

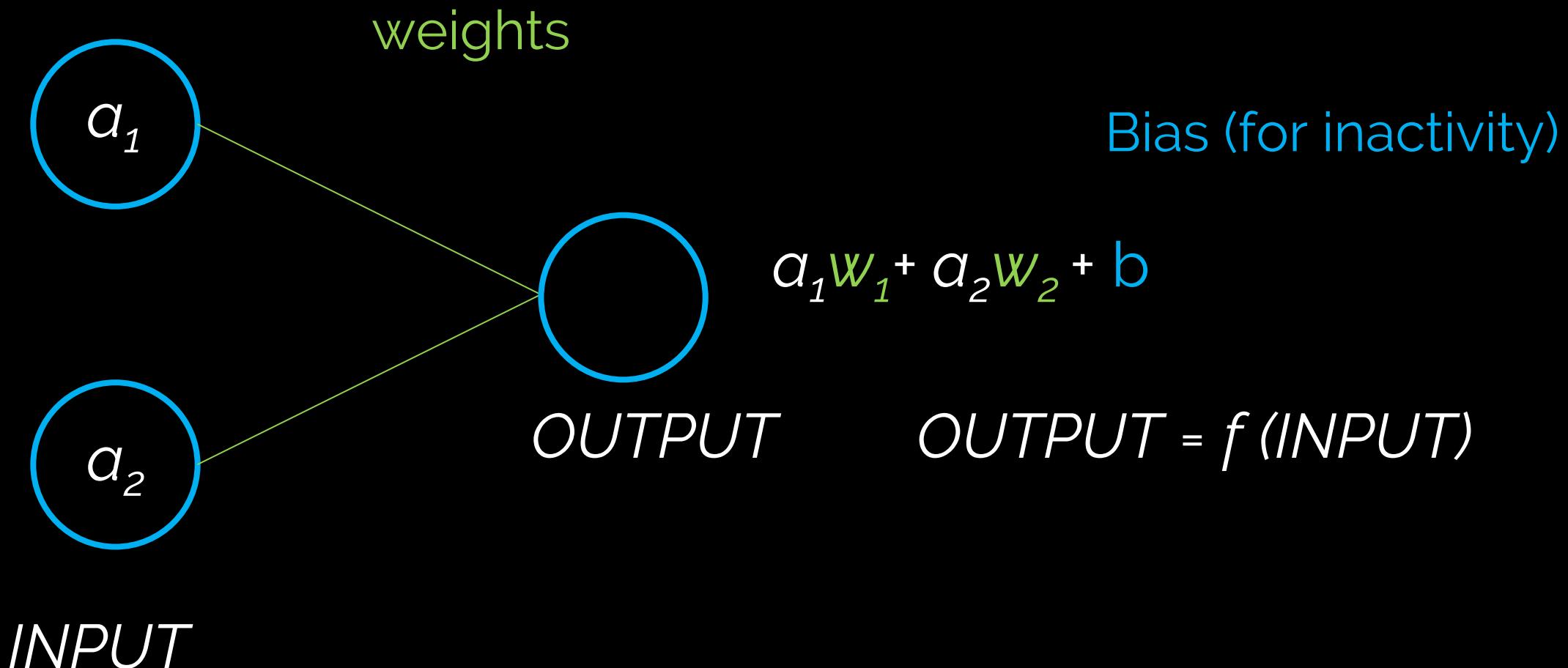


Weights and Bias

Neural Network | Weights and Bias

$$H = a_1$$

$$AC = a_2$$

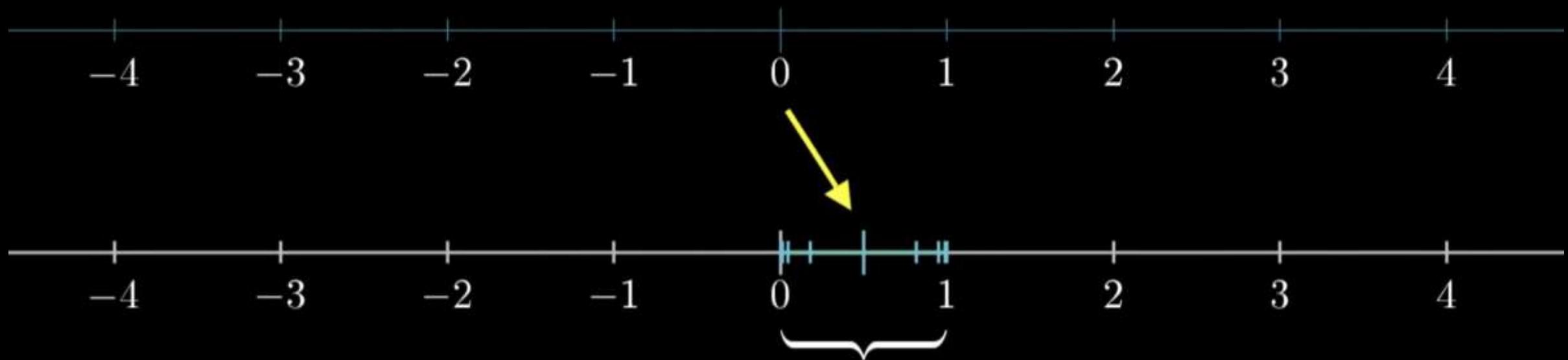


The Activation Function

Neural Network

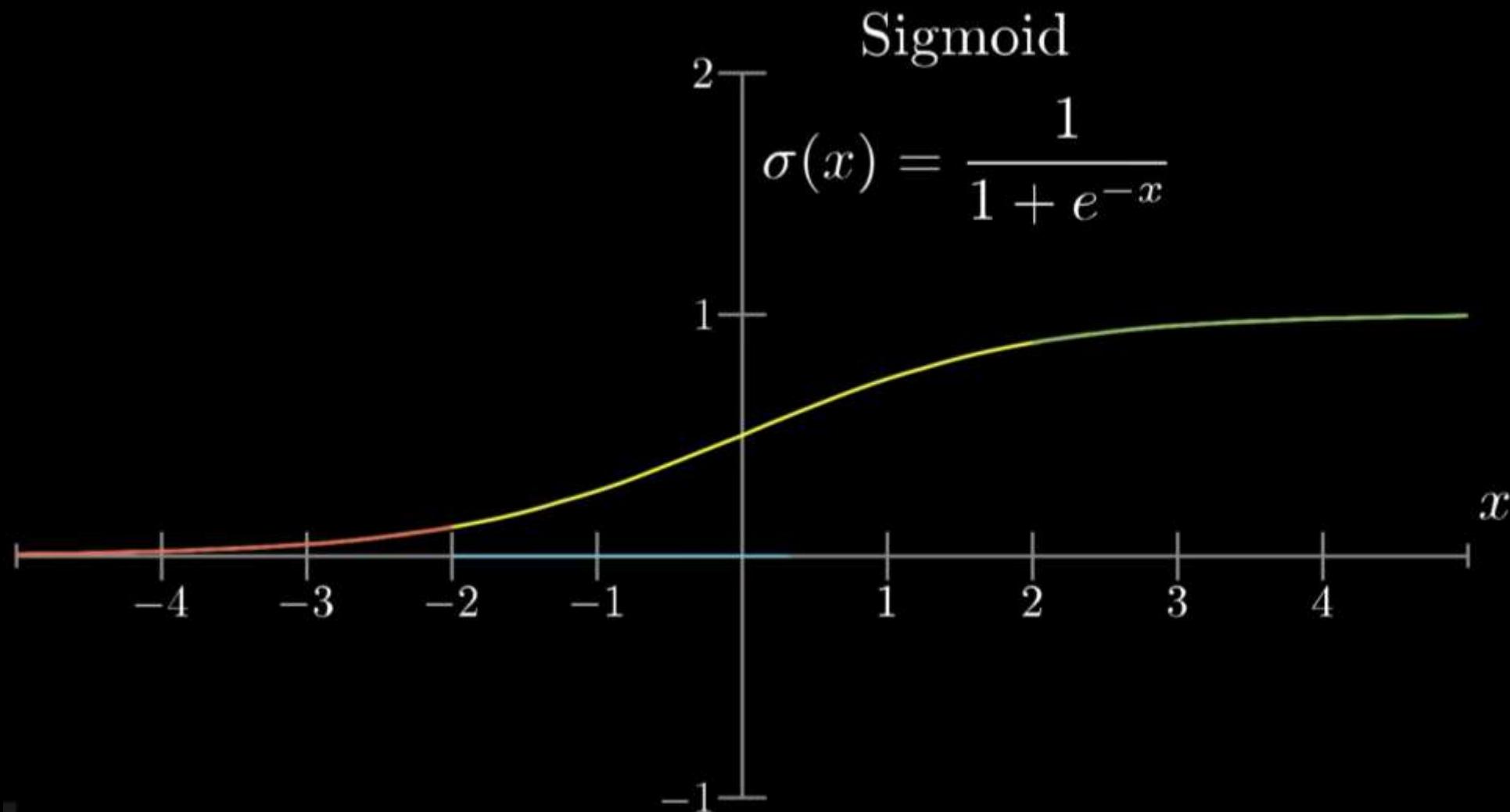
i

$$w_1a_1 + w_2a_2$$



Activations should be in this range

Neural Network

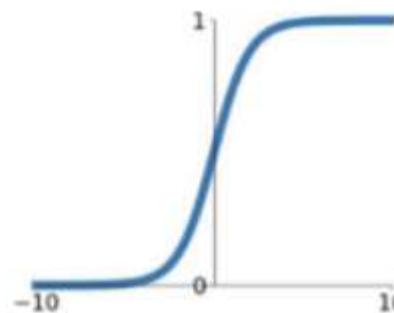


Neural Network

Other functions that progressively change from 0 to 1 with no discontinuity

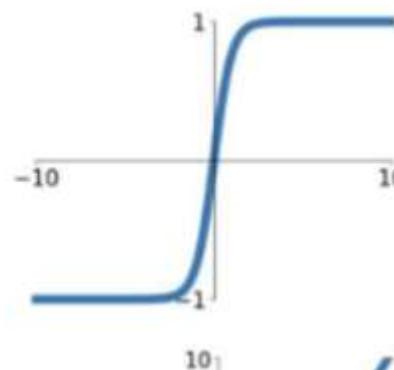
Sigmoid

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



tanh

$$\tanh(x)$$



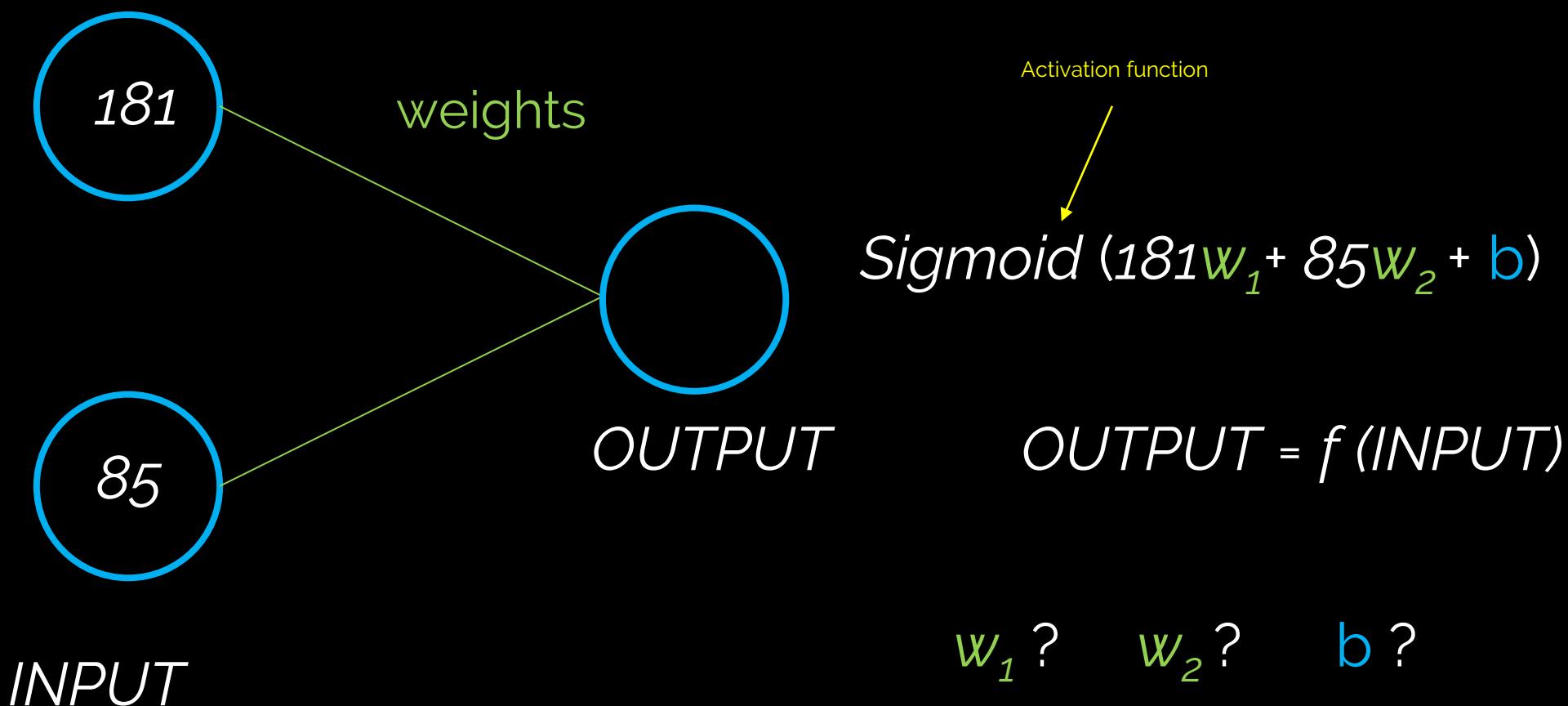
Hyperbolic
tangent
function

Neural Network

$$H=a_1$$

$$AC=a_2$$

Forward propagation



INPUT

$a_1 \quad a_2$

Cost Function?

Sigmoid ($a_1 w_1 + a_2 w_2 + b$)

OUTPUT

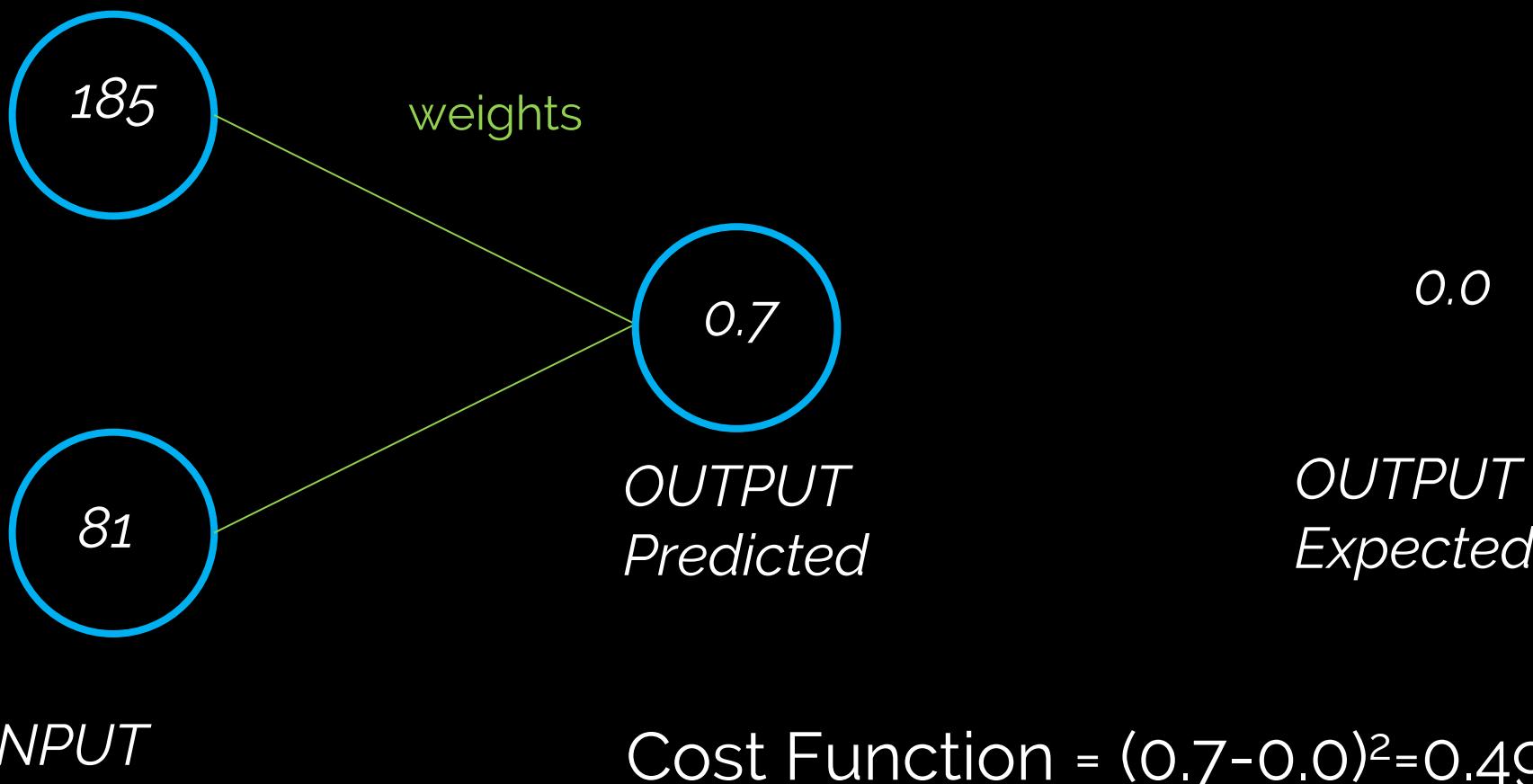
Cost Function = (Predicted-Expected)²

Squared error cost

Neural Network

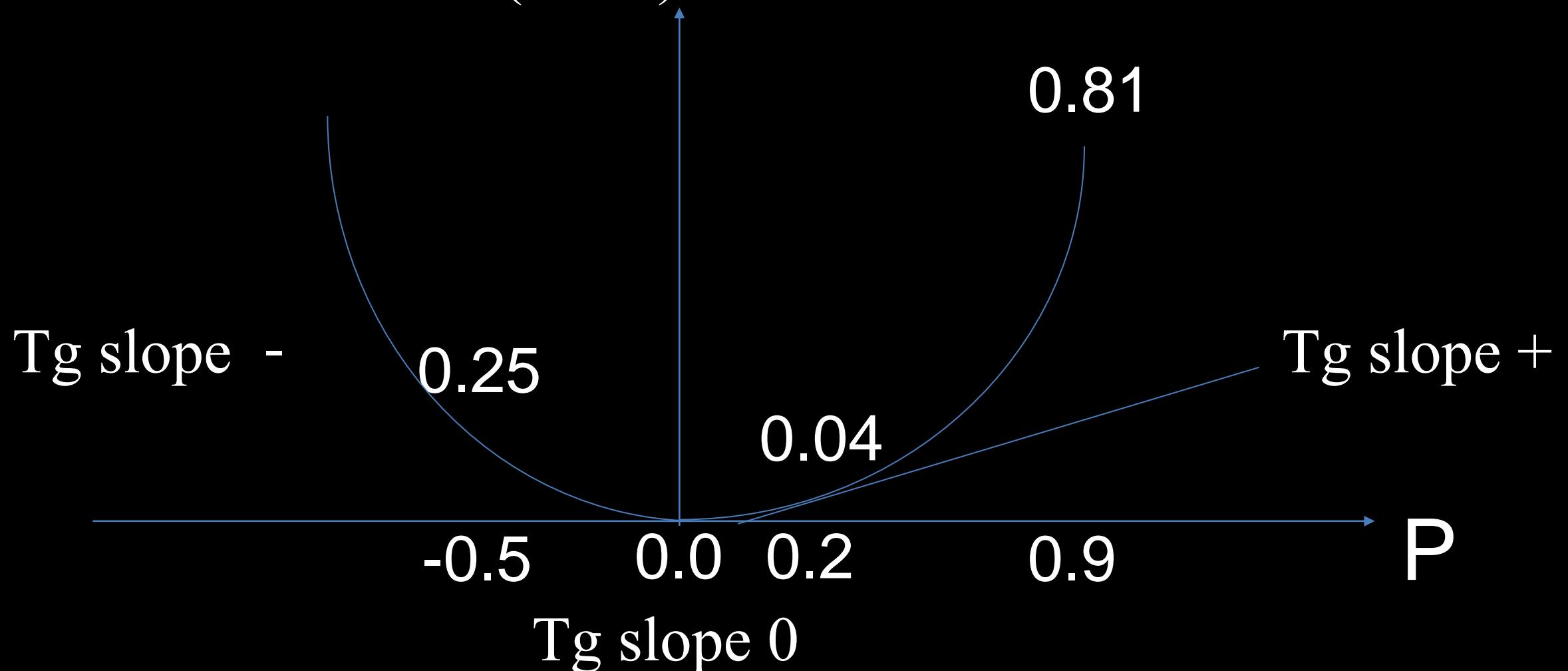
$$H=a_1$$

$$AC=a_2$$



Neural Network

The Cost Function = $(P-0.0)^2$



Neural Network

If the slope is + we must decrease P of a fraction of the slope

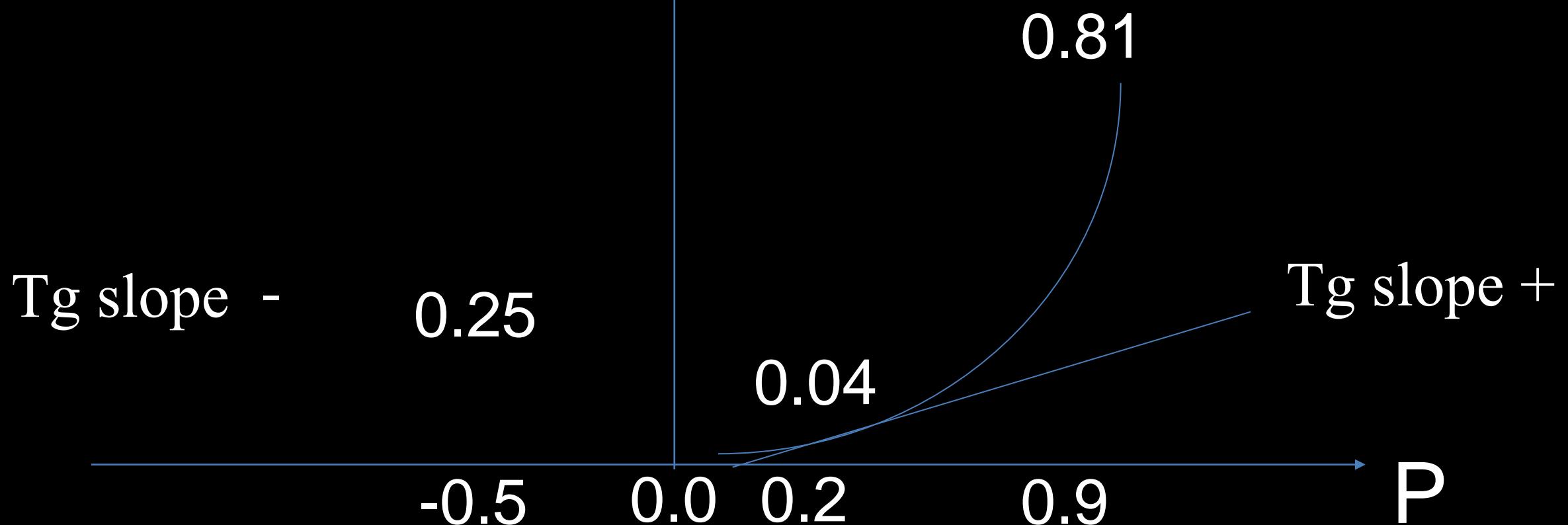
If the slope is - we must increase P of a fraction of the slope

If the slope is 0 we have the solution

The Learning Rate

Neural Network

The Cost Function = $(P-0.0)^2$



Next $P = P - LR * (\text{Tg slope})P$ LR^* =Learning Rate

Neural Network

$$\text{Next } P = P - LR^* \text{ (slope Tg)}P$$

Slope Tg= derivative of the Cost Function vs P= $2(P-E)$

In our case

LR determines how much weights are changed every time
Too high → output wanders around the expected solutions
Too low → output fails to converge to acceptable solution

The Training

Different training methods

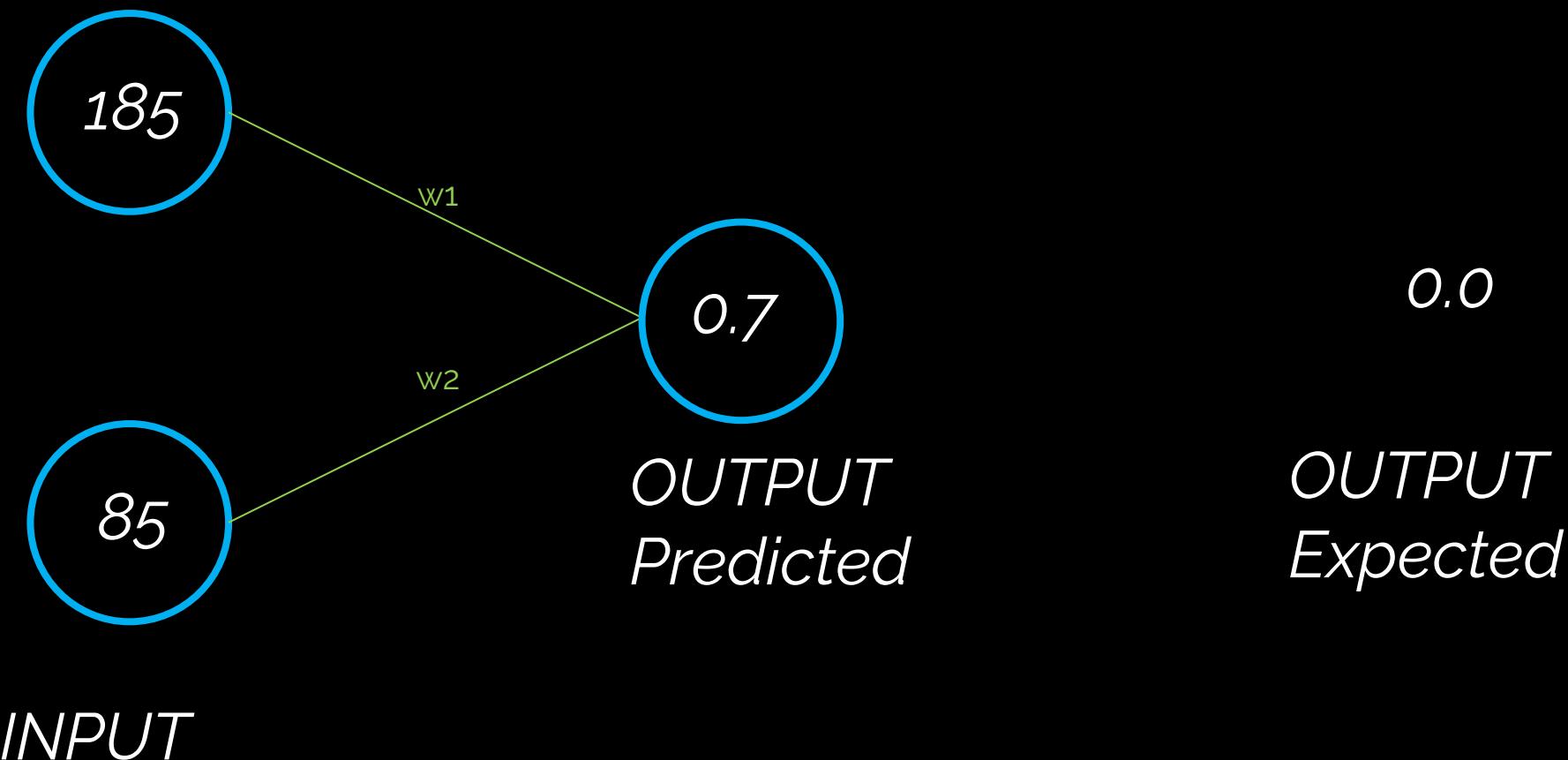


Supervised

learning rule that trains the neural network on
already known correct output

Neural Network

$$H=a_1 \quad CA=a_2$$



Neural Network

w1 = 0.0006

w2 = 0.0002

b=1

$$0.7 = \text{Sigmoid} (185 \times 0.0006 + 85 \times 0.0002 + 1) = \\ \text{Sigmoid} (0.111 + 0.281 + 1) = \text{Sigmoid} (1.281)$$

the back propagation

Computation of the error (STEP 2)

The Cost Function = $(0.7 - 0.0)^2 = 0.49$

$$= (\text{Sigmoid}(185 \times 0.0006 + 85 \times 0.0002 + 1) - 0.0)^2$$

Weights and Bias Adjustment: the Back Propagation

Neural Network

Adjust weights and b to reduce the error (STEP 3)

$$0.0005 = 0.0006 - 0.0001$$

$$0.0001 = 0.0002 - 0.0001$$

$$0.0008 = 1 - 0.0002$$

$w_1 = w_1 - LR^* \text{slope} = w_1 - LR^* \text{derivative}_{w1} \text{ of the Cost Function}$

$w_2 = w_2 - LR^* \text{slope} = w_2 - LR^* \text{derivative}_{w2} \text{ of the Cost Function}$

$b = b - LR^* \text{slope} = b - LR^* \text{derivative}_b \text{ of the Cost Function}$

Neural Network

$$w_1 = w_1 - LR^* \frac{\partial \text{costo}}{\partial w_1}$$

$$w_2 = w_2 - LR^* \frac{\partial \text{costo}}{\partial w_2}$$

$$b = b - LR^* \frac{\partial \text{costo}}{\partial b}$$

Esci

Neural Network

$$\frac{\partial \text{costo}}{\partial w_1} = \frac{\partial \text{costo}}{\partial p} \times \frac{\partial p}{\partial t}$$

$$\frac{\partial}{\partial t} \text{sigmoide}(t) = \text{sigmoide}(t)(1 - \text{sigmoide}(t))$$

$$\frac{\partial \text{costo}}{\partial w_1} = \frac{\partial \text{costo}}{\partial p} \times \frac{\partial p}{\partial t} \times \frac{\partial t}{\partial w_1}$$

$$\frac{\partial \text{costo}}{\partial w_1} = 2(\text{sigmoide}(2w_1 + 5w_2 + b) - 1) \times \text{sigmoide}(2w_1 + 5w_2 + b)(1 - \text{sigmoide}(2w_1 + 5w_2 + b)) \times 2$$

Neural Network

$$\frac{\partial \text{costo}}{\partial w_1} = 2(\text{sigmoide}(2w_1+5w_2+b) - 1) \times \text{sigmoide}(2w_1+5w_2+b)(1-\text{sigmoide}(2w_1+5w_2+b)) \times 2$$

$$\frac{\partial \text{costo}}{\partial w_2} = 2(\text{sigmoide}(2w_1+5w_2+b) - 1) \times \text{sigmoide}(2w_1+5w_2+b)(1-\text{sigmoide}(2w_1+5w_2+b)) \times 5$$

$$\frac{\partial \text{costo}}{\partial b} = 2(\text{sigmoide}(2w_1+5w_2+b) - 1) \times \text{sigmoide}(2w_1+5w_2+b)(1-\text{sigmoide}(2w_1+5w_2+b)) \times 1$$

Neural Network

$$\frac{\partial \text{costo}}{\partial w_1} =$$

$$\frac{\partial \text{costo}}{\partial w_1} = \frac{\partial \text{costo}}{\partial p} \times \frac{\partial p}{\partial t} \times \frac{\partial t}{\partial w_1}$$

$$= 2(\text{sigmoide}(2w_1+5w_2+b) - 1) \times \text{sigmoide}(2w_1+5w_2+b)(1-\text{sigmoide}(2w_1+5w_2+b)) \times 2$$

$$\frac{\partial \text{costo}}{\partial w_2} =$$

$$\frac{\partial \text{costo}}{\partial w_2} = \frac{\partial \text{costo}}{\partial p} \times \frac{\partial p}{\partial t} \times \frac{\partial t}{\partial w_2}$$

$$= 2(\text{sigmoide}(2w_1+5w_2+b) - 1) \times \text{sigmoide}(2w_1+5w_2+b)(1-\text{sigmoide}(2w_1+5w_2+b)) \times 5$$

$$\frac{\partial \text{costo}}{\partial b} =$$

$$\frac{\partial \text{costo}}{\partial b} = \frac{\partial \text{costo}}{\partial p} \times \frac{\partial p}{\partial t} \times \frac{\partial t}{\partial b}$$

$$= 2(\text{sigmoide}(2w_1+5w_2+b) - 1) \times \text{sigmoide}(2w_1+5w_2+b)(1-\text{sigmoide}(2w_1+5w_2+b)) \times 1$$

Neural Network

$$w_1 = 0.0006 \quad w_2 = 0.0002 \quad b = +1$$

The Cost Function ($w_1 w_2 b$) = (Sigmoid-0.0)²=0.49

Back propagation

Weights and bias adjustment

$$w_1 = 0.0005 \quad w_2 = 0.0001 \quad b = 0.0008$$

The Cost Function ($w_1 w_2 b$) = (Sigmoid-0.0)²=0.43

$$\begin{aligned} \text{Sigmoid } (185 \times 0.0005 + 85 \times 0.0001 + 1) &= \text{Sigmoid} \\ (0.0925 + 0.085 + 0.008) &= \text{Sigmoid } (1.178) = 0.65 \end{aligned}$$

the back propagation

Neural Network

$$H = a_1$$

$$CA = a_2$$

Forward propagation

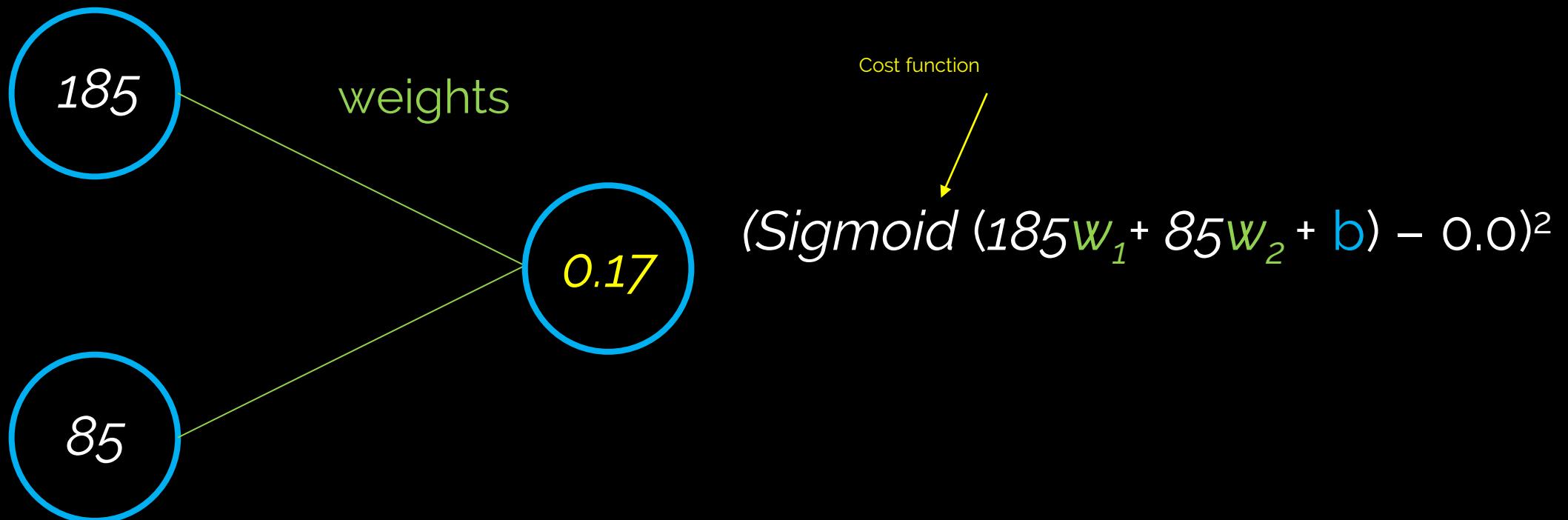


Neural Network

$$H = a_1$$

$$CA = a_2$$

Back propagation



Neural Network

Training set: ‘healthy’ vs overweighting men

Neural Network

Calculate the error (STEP 2)

Adjust weights and b to reduce the error (STEP 3)

Repeat step 2 and step 3 for all training data until the error is within acceptable level

These steps are similar to supervised machine learning (model adjusting vs learning rules adjusting)
(model adjusting vs weights and bias adjusting)

EPOCH (1 Training iteration)

Neural Network

Sigmoid ($185 \times 0.0006 + 85 \times 0.0002 + 1$) = 0.7

Sigmoid ($178 \times 0.0004 + 94 \times 0.0003 + 0.008$) = 0.65

Sigmoid ($184 \times 0.0003 + 102 \times 0.0001 + 0.007$) = 0.64

Sigmoid ($100 \times 0.0005 + 80 \times 0.0002 + 0.008$) = 0.55

$$(\text{Sigmoid} - 0.0)^2 = 0.49$$

$$(\text{Sigmoid} - 0.0)^2 = 0.43$$

$$(\text{Sigmoid} - 0.0)^2 = 0.41$$

$$(\text{Sigmoid} - 0.0)^2 = 0.30$$

Generalized delta rule

SGD (Stochastic Gradient Descent) method (error is calculated for each training data, weights updated immediately)

Batch method (error is calculated for all training data, each of the weight update are calculated but the average of all weight updates are used only once in each epoch)

Mini-Batch method (mix) →

- Small values give a learning process that converges quickly at the cost of noise in the training process.
- Large values give a learning process that converges slowly with accurate estimates of the error gradient.

Mini-Batch method

Training data 1

Training data 2

Training data 3

Training data N

Training using batch method

Part of the training data is selected

Mini-Batch method

1-10

11-20

21-40

41-60

61-80

Batch method applied

5 weight update will be performed to complete the training process

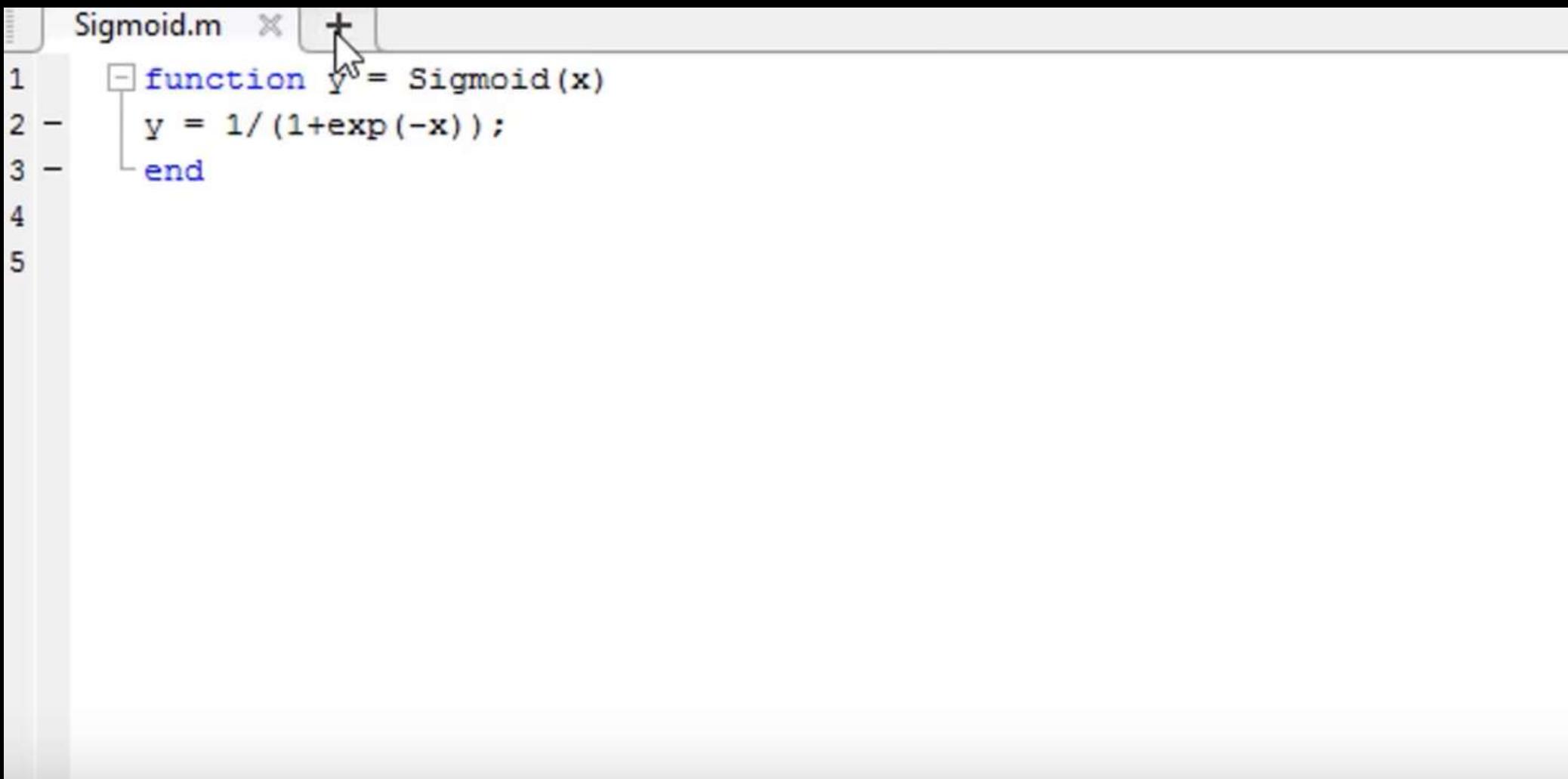
Robustness of SGD
Efficiency of batch

SOFTWARE CODES

Matlab: essential functions and scripts

Matlab: simple examples

Neural Network



The image shows a screenshot of a MATLAB code editor window. The window title is "Sigmoid.m". The code inside the editor is as follows:

```
1 function y = Sigmoid(x)
2 - y = 1/(1+exp(-x));
3 - end
4
5
```

The cursor is positioned at the start of the first line of code. The MATLAB interface includes a toolbar with various icons and a menu bar.

Neural Network

```
Sigmoid.m  SGD_method.m  +
1 function Weight = SGD_method(Weight, input, correct_Output)
2 alpha = 0.9;
3
4 N = 4;
5 for k = 1:N
6     transposed_Input = input(k, :)';
7     d = correct_Output(k);
8     weighted_Sum = Weight*transposed_Input;
9     output = Sigmoid(weighted_Sum);
10
11 error = d - output;
12 delta = output*(1-output)*error;
13
14 dWeight = alpha*delta*transposed_Input;
15
16 Weight(1) = Weight(1) + dWeight(1);
17 Weight(2) = Weight(2) + dWeight(2);
```

Neural Network

The screenshot shows a MATLAB interface with two open files in the workspace:

- Sigmoid.m**: A function file containing the definition of the sigmoid function.
- SGD_method.m***: An MEX file generated by MATLAB Coder, which contains the main logic for a neural network using the Stochastic Gradient Descent (SGD) method.

The code in **SGD_method.m*** is as follows:

```
3 N = 4;
4 for k = 1:N
5     transposed_Input = input(k, :)';
6     d = correct_Output(k);
7     weighted_Sum = Weight*transposed_Input;
8     output = Sigmoid(weighted_Sum);
9
10    error      = d - output;
11    delta = output*(1-output)*error;
12
13
14    dWeight = alpha*delta*transposed_Input;
15
16    Weight(1) = Weight(1) + dWeight(1);
17    Weight(2) = Weight(2) + dWeight(2);
18    Weight(3) = Weight(3) + dWeight(3);
19
20 end
21 end
```

Neural Network

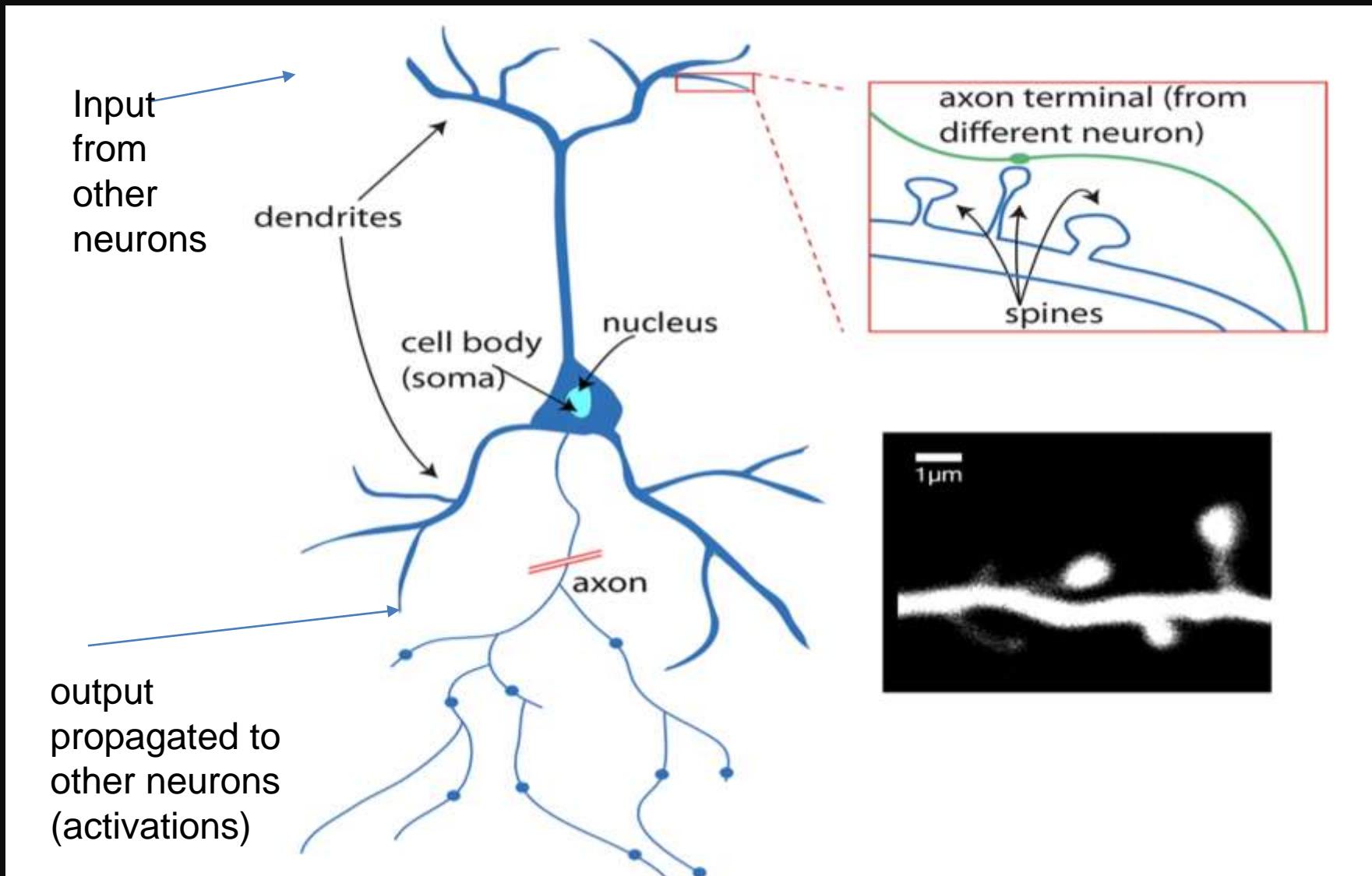
```
Sigmoid.m    SGD_method.m    Training.m    +  
1 -     input = [ 0 0 1;  
2                0 1 1;  
3                1 0 1;  
4                1 1 1;  
5                ];  
6 -     correct_Output = [0  
7                0  
8                1  
9                1  
10              ];  
11 -    Weight = 2*rand(1, 3) - 1;  
12 -    for epoch = 1:10000  
13 -        Weight = SGD_method(Weight, input, correct_Output);  
14 -    end  
15  
16 -    save('Trained_Network.mat')  
17
```

Neural Network

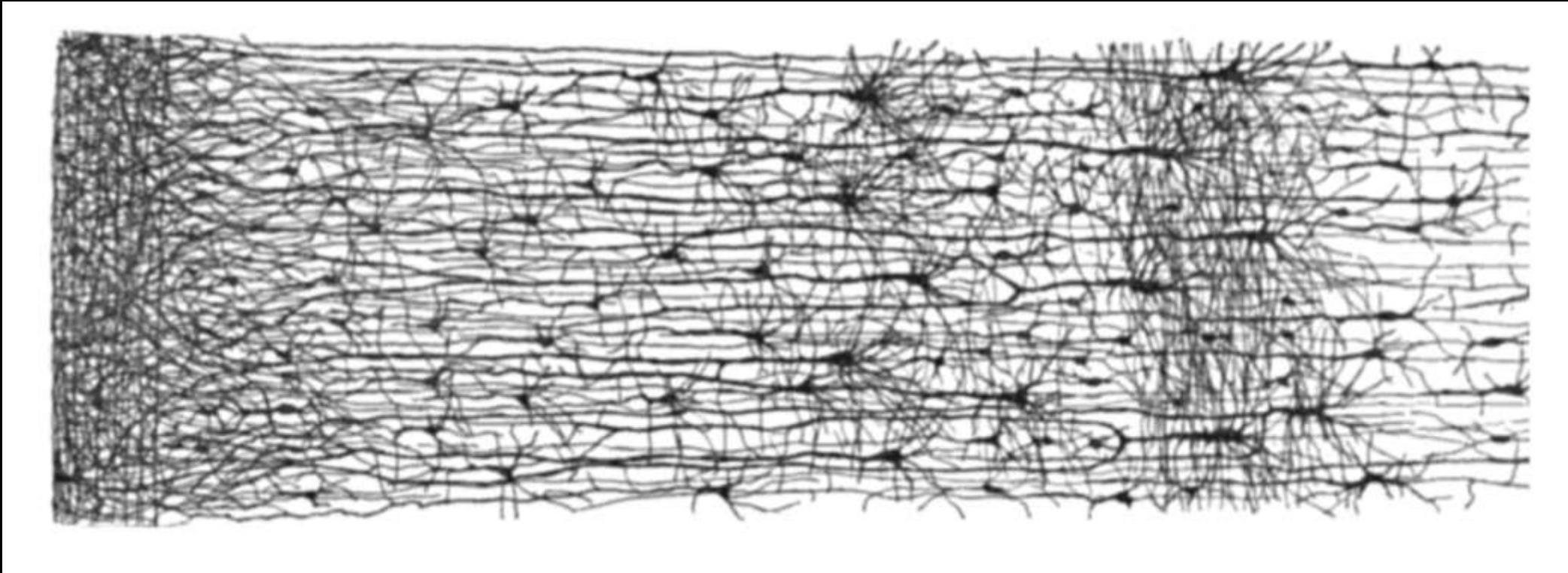
```
Sigmoid.m  SGD_method.m  Training.m  testing.m*  +
1 -     load('Trained_Network.mat');
2 -     input = [ 0 0 1;
3 -                 0 1 1;
4 -                 1 0 1;
5 -                 1 1 1;
6 -                 ];
7 -     N = 4;
8 -     for k = 1:N
9 -         transposed_Input = input(k, :)';
10 -        weighted_Sum = Weight*transposed_Input;
11 -        output = Sigmoid(weighted_Sum)
12 -    end
```

DEEP LEARNING

A Biological Neuron



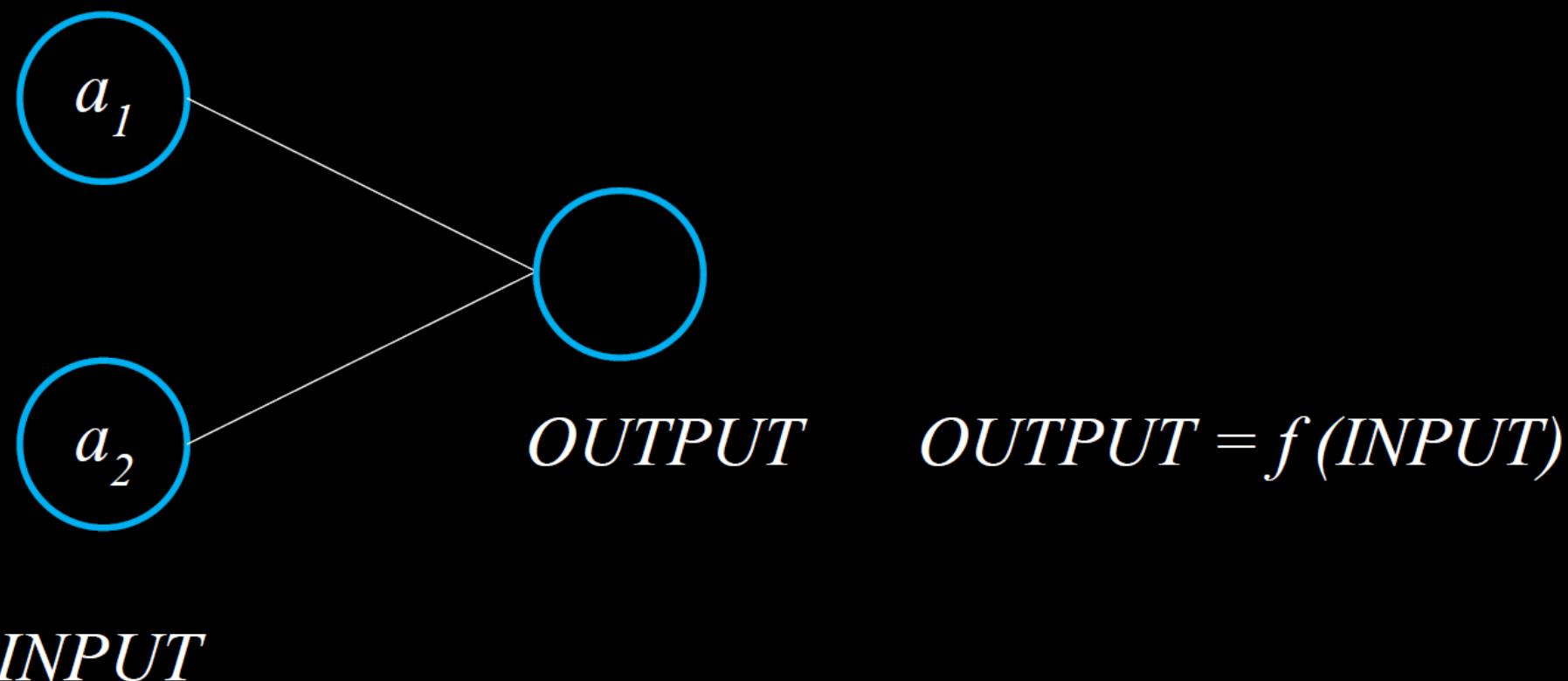
A Biological Neuron Network



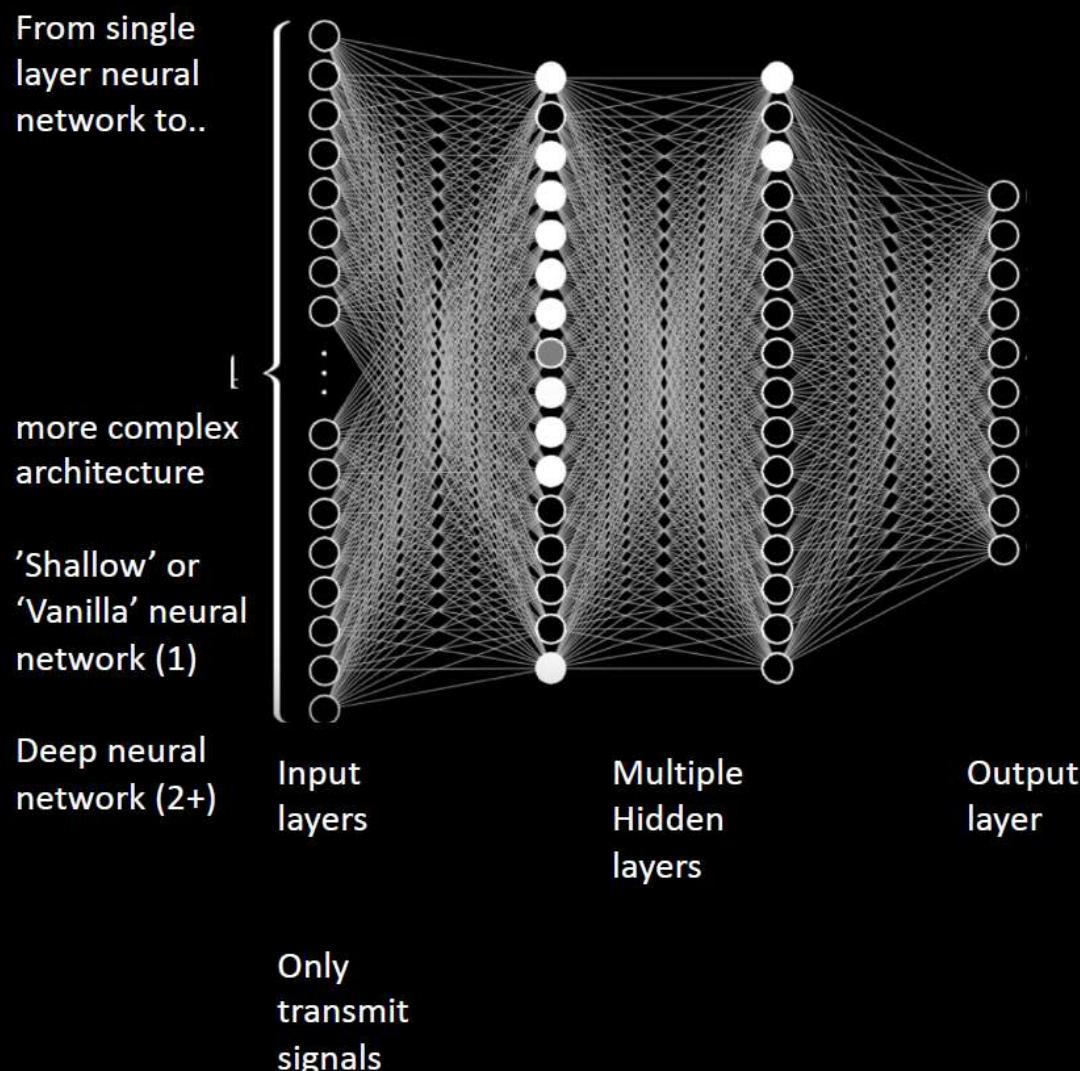
100 B neurons

Deep Learning

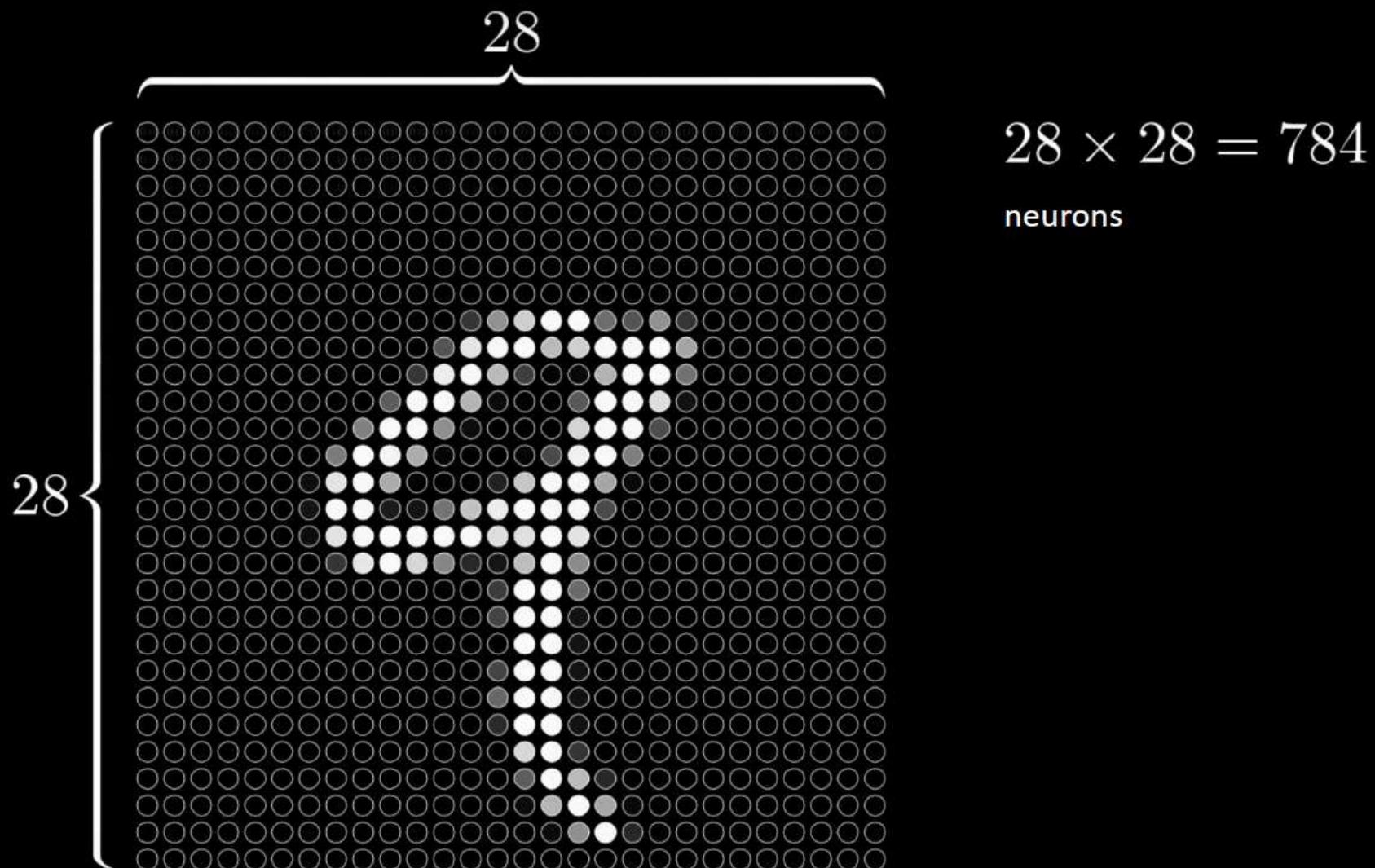
$$H=a_1 \quad AC=a_2$$



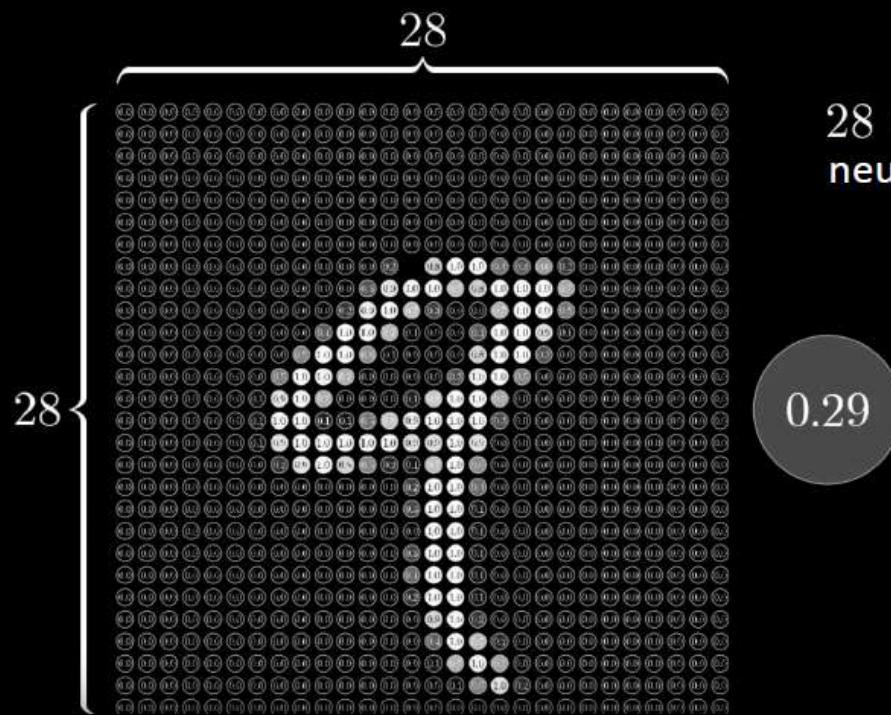
Deep Learning



Deep Learning



Deep Learning



$28 \times 28 = 784$
neurons

→ They make the 1° layer of the
network

«Activation» of the neuron (input)

Grey scale value of each image pixel

Deep Learning

○○○●●●○○○○○○○

784

○○○●●●●○○○●●●○○○○○○○○

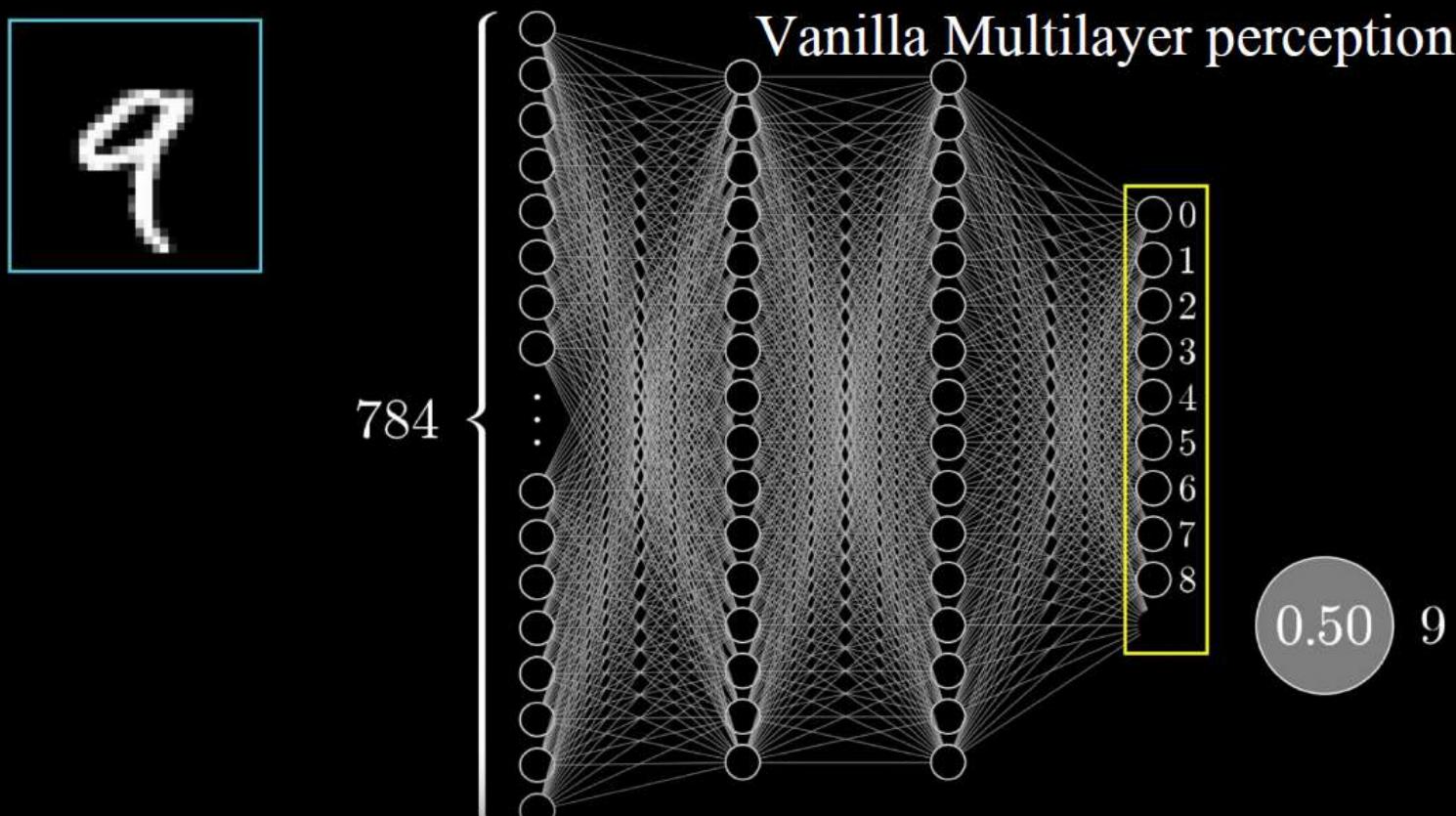
○○○○○○●●●○○○●●●○○○○○○○○○○

○○○○○○●●●●●●●●●○○○○○○○○○○○○

○○○○○○●●●●●●●●●○○○○○○○○○○○○○○○○

784

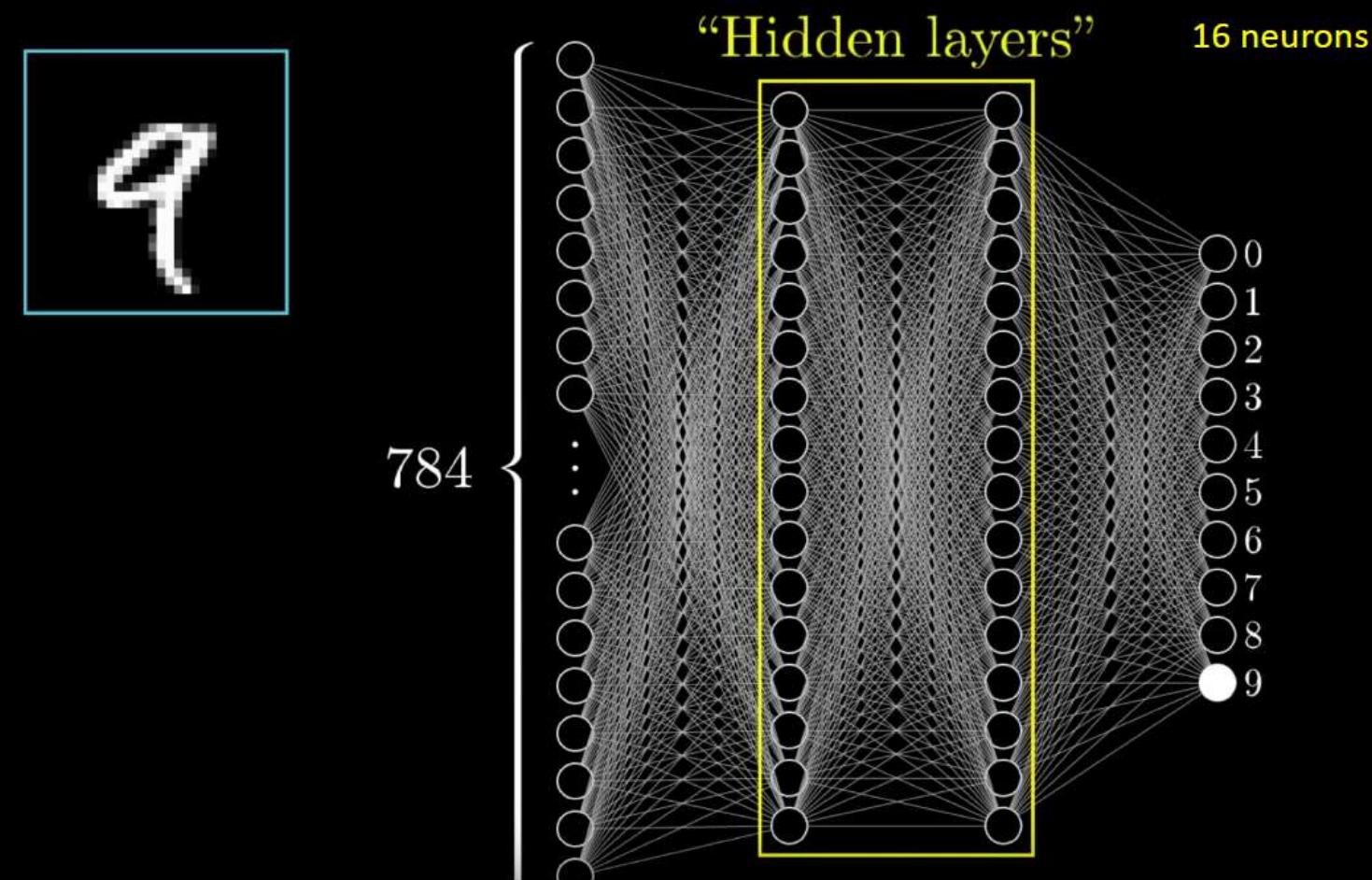
○○○○○○○ ... ○○○○○○○



«Activation» of the neuron (output)

How the system think the given image
corresponds to the given digit

Deep Learning



ACTIVATION of a layer determines activation in the next layers (cascade influence)

Deep Learning

$$\begin{aligned} q &= \text{ } \text{ } \text{ } \\ g &= \text{ } \text{ } \text{ } \\ y &= \text{ } \text{ } \text{ } \end{aligned}$$

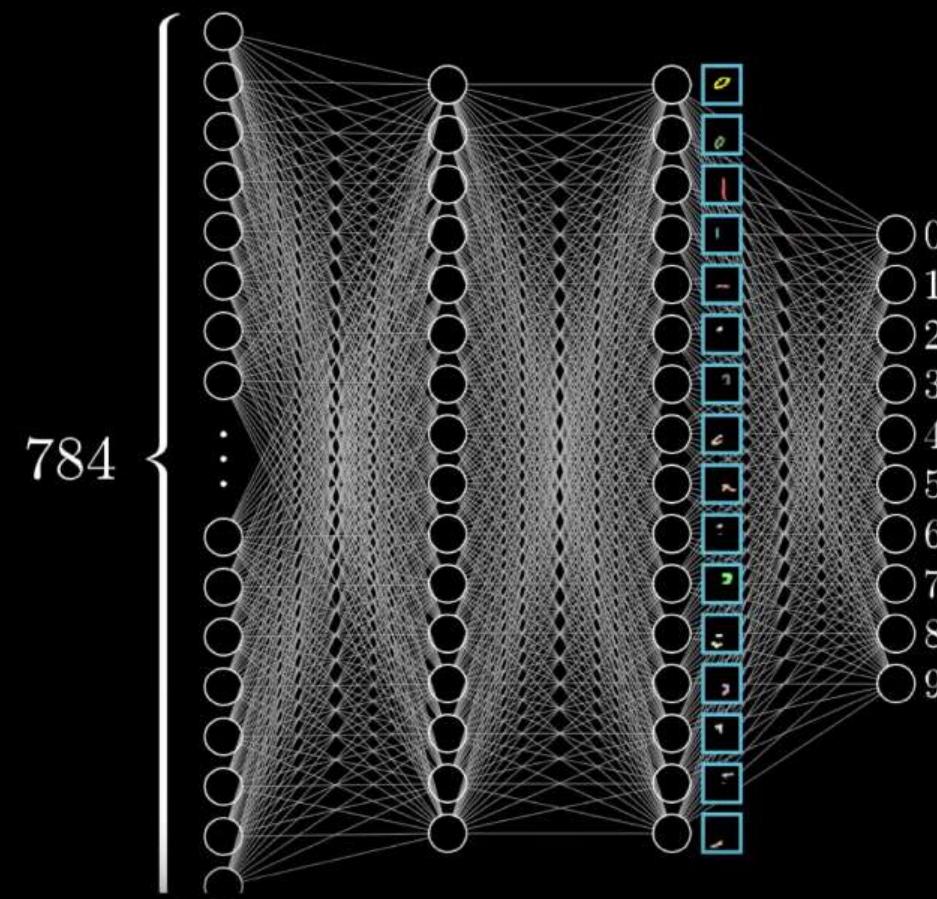
The diagram illustrates the decomposition of handwritten digits into their constituent features. It shows three rows of equations:

- The first row shows the digit 'q' decomposed into a yellow circle and a red vertical stroke.
- The second row shows the digit 'g' decomposed into a yellow circle and a green oval.
- The third row shows the digit 'y' decomposed into a red vertical stroke, a blue vertical stroke, and a small pink horizontal stroke.

Each term in the equations is enclosed in a light blue square frame.

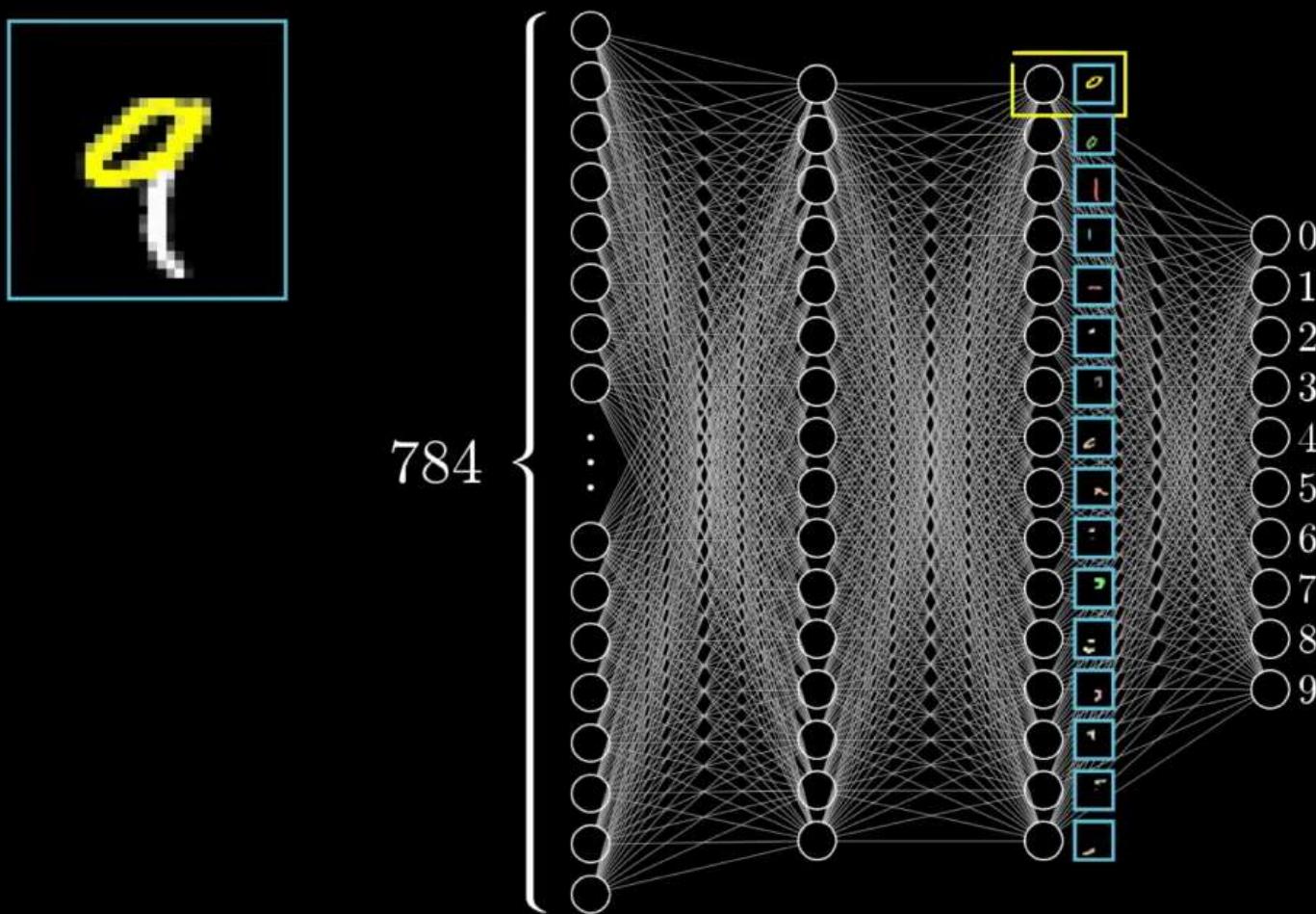
Possible subcomponents recognized by our visual-cortex neurons

Deep Learning



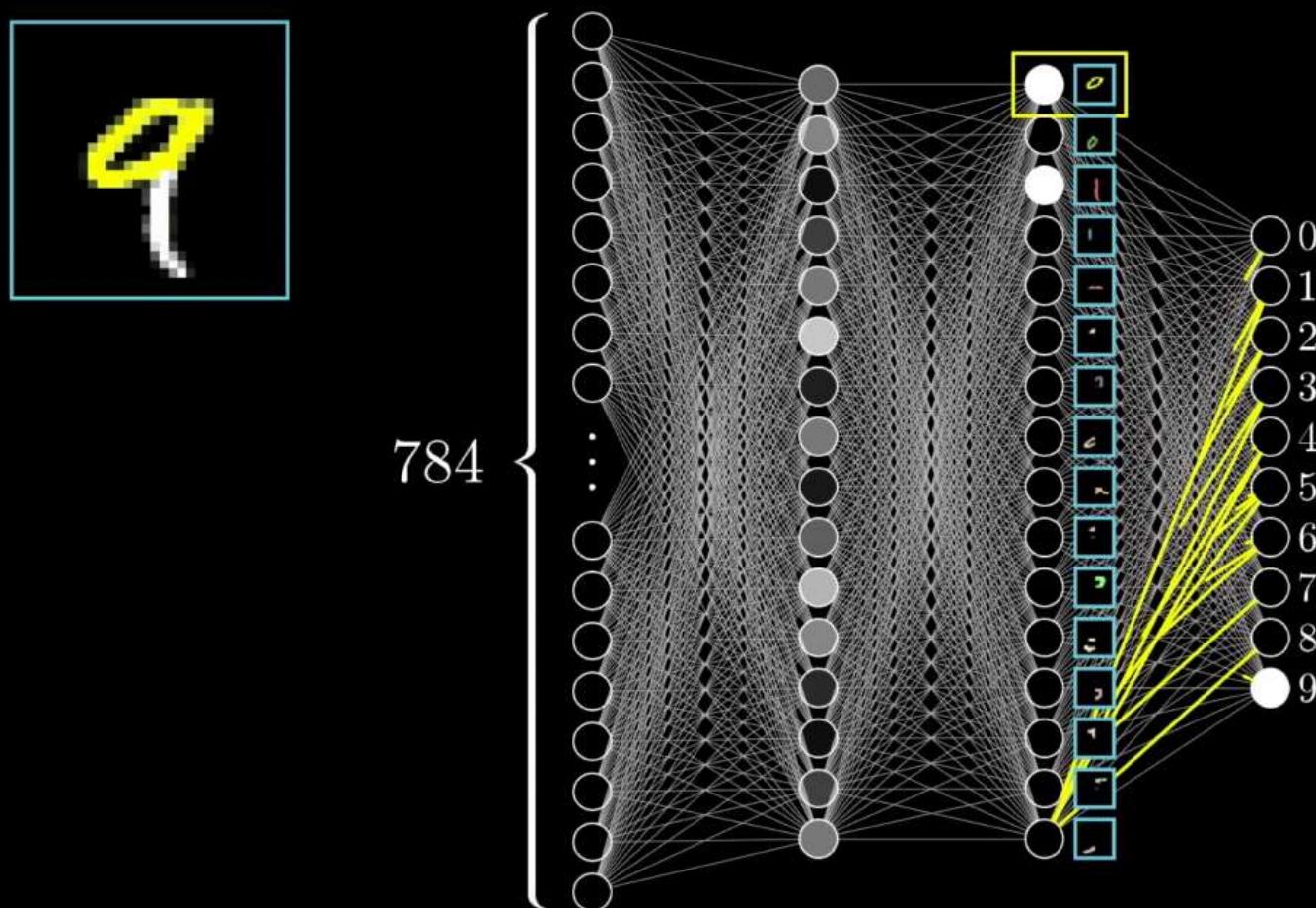
In the optimal situations each neurons of the middle layer correpond to each of the possible sub-components

Deep Learning



Anytime we feed an image with a loop (9, 8) on top there is a specific neuron in the third layer whose activation is near to 1

Deep Learning



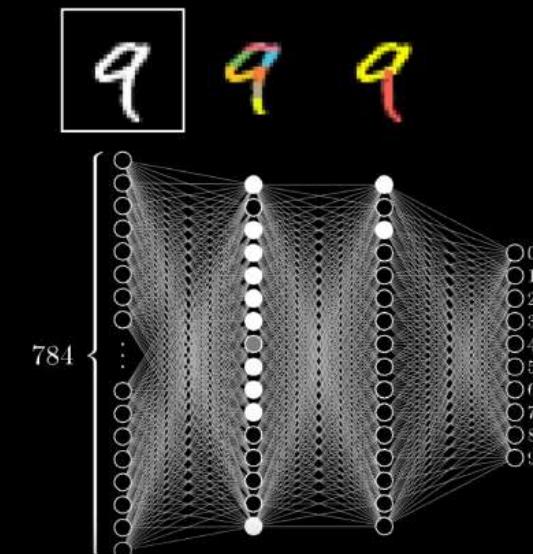
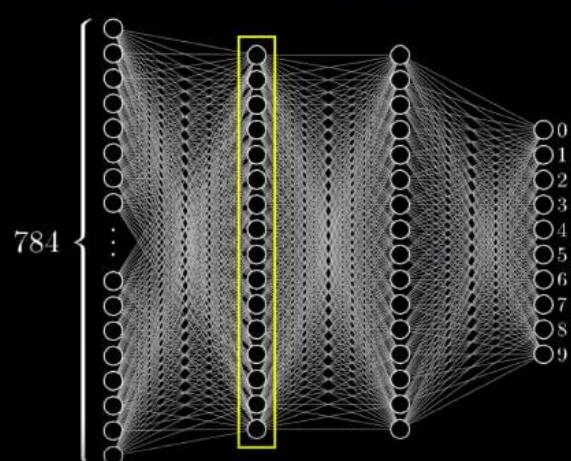
Going from the third layer to the last one just require learning which combination of sub-components corresponds to the given digits

Deep Learning

Break down the task into sub-problems
(e.g., recognize different edges from pixels)



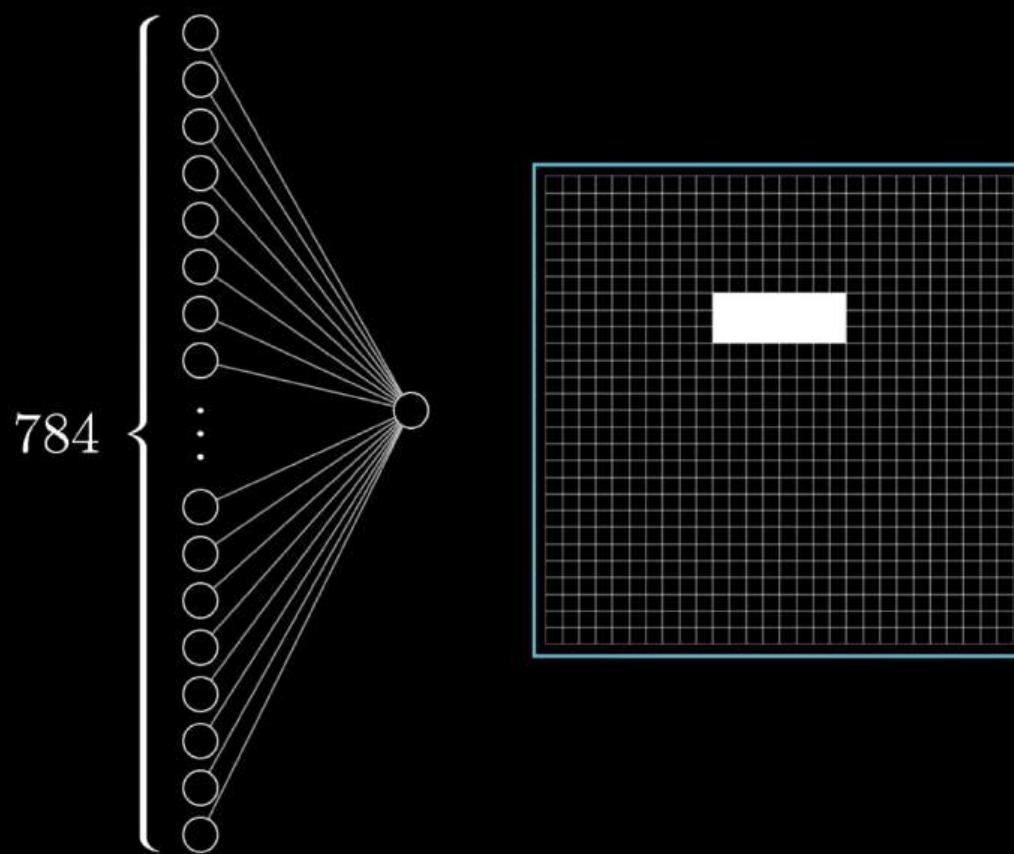
“Little edge” layer?



(recognize different
patterns from edges)

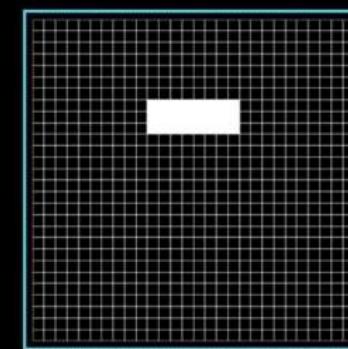
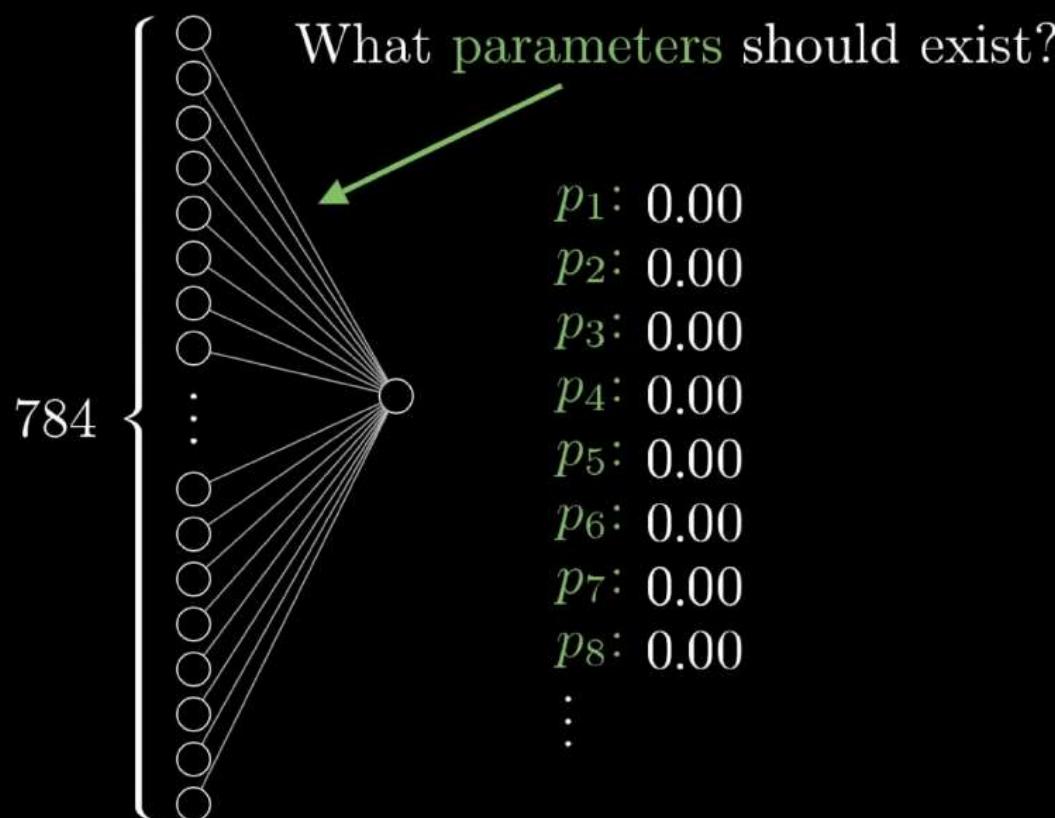
Deep Learning

Consider one specific neuron in the second layer, to “identify” whether an image has an edge in a given region or not



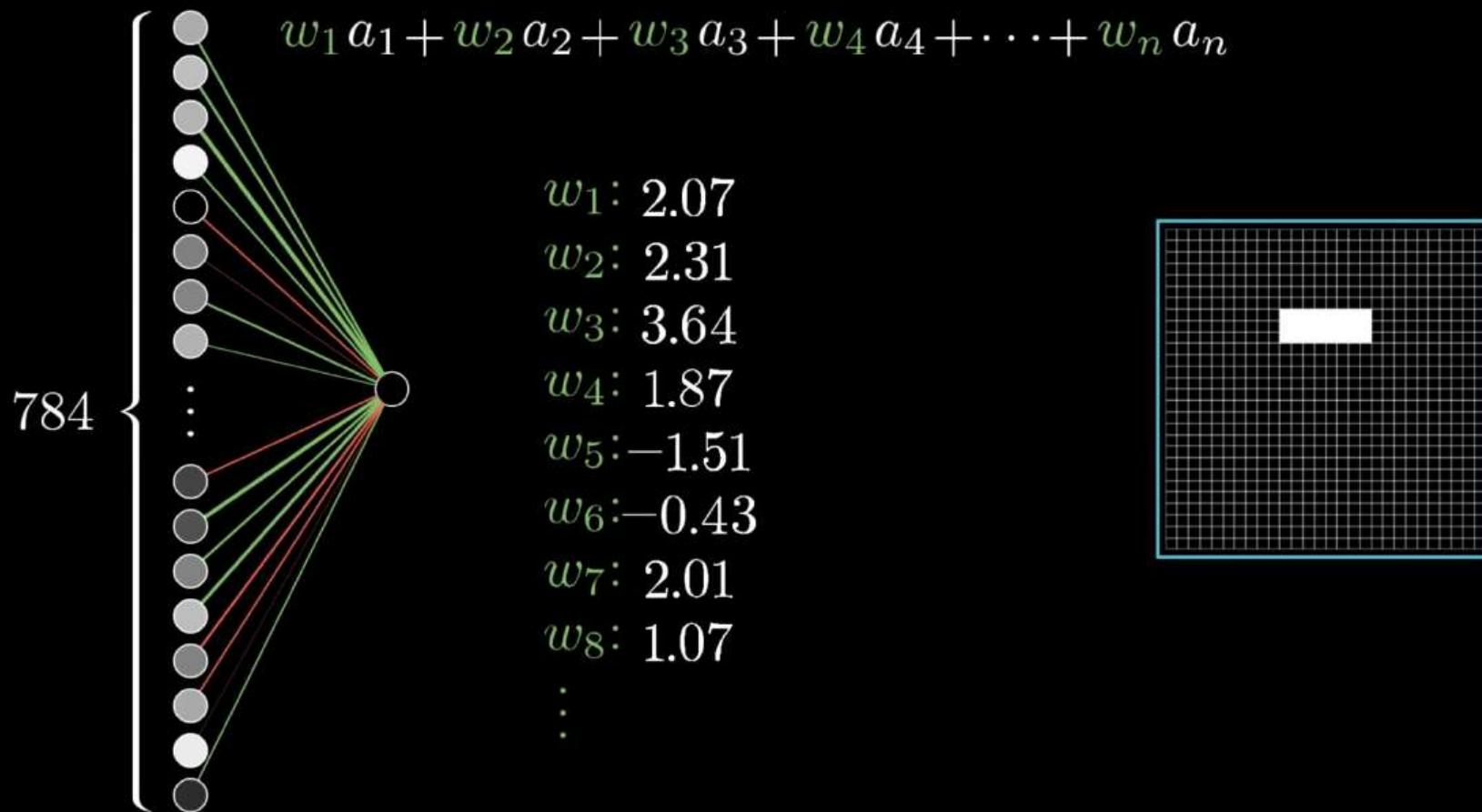
Deep Learning

Which are the best network parameters to capture this pattern?



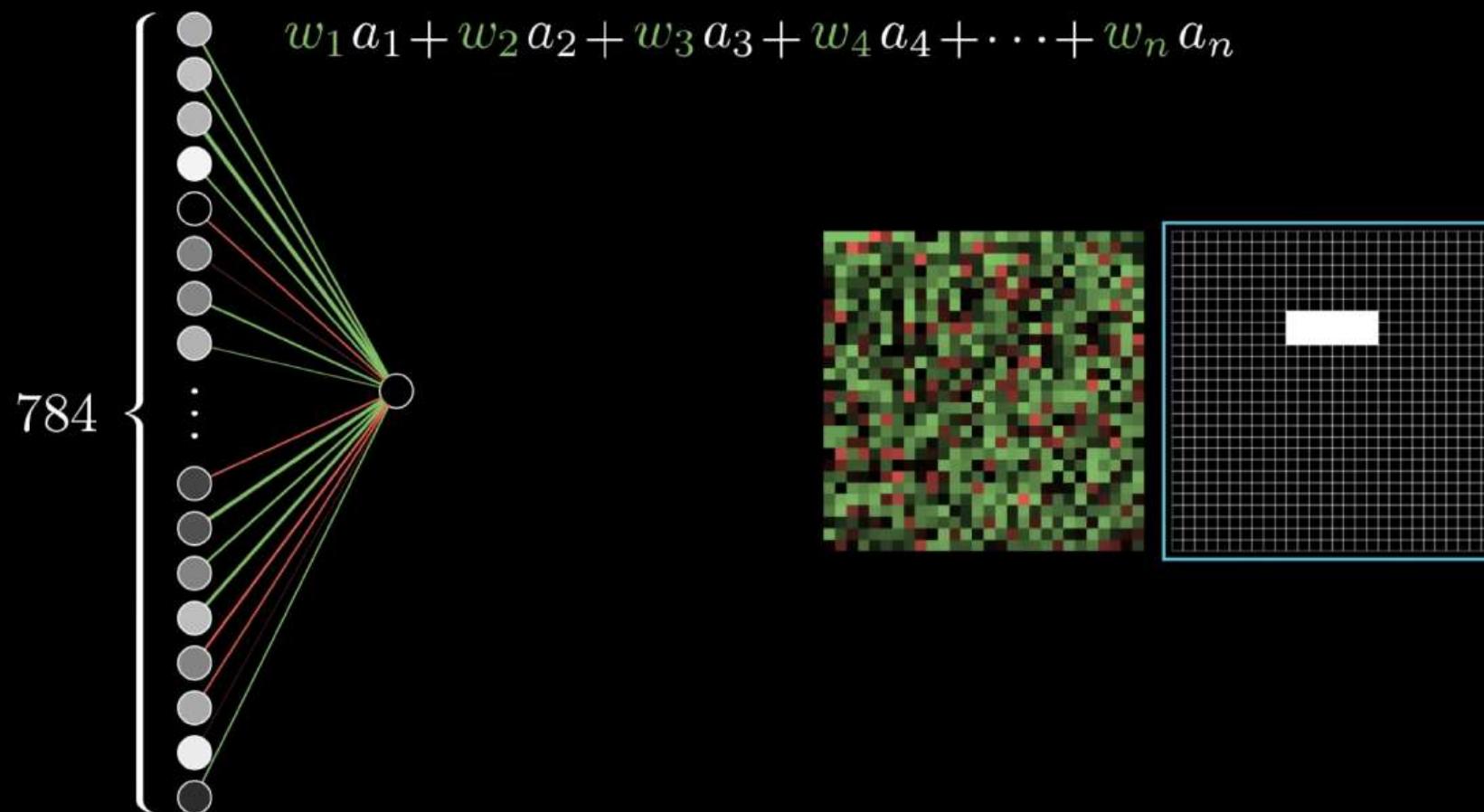
Deep Learning

Algorithm: assign (learn) a weight to each connection between our neuron and all the neuron in the previous (first) layer

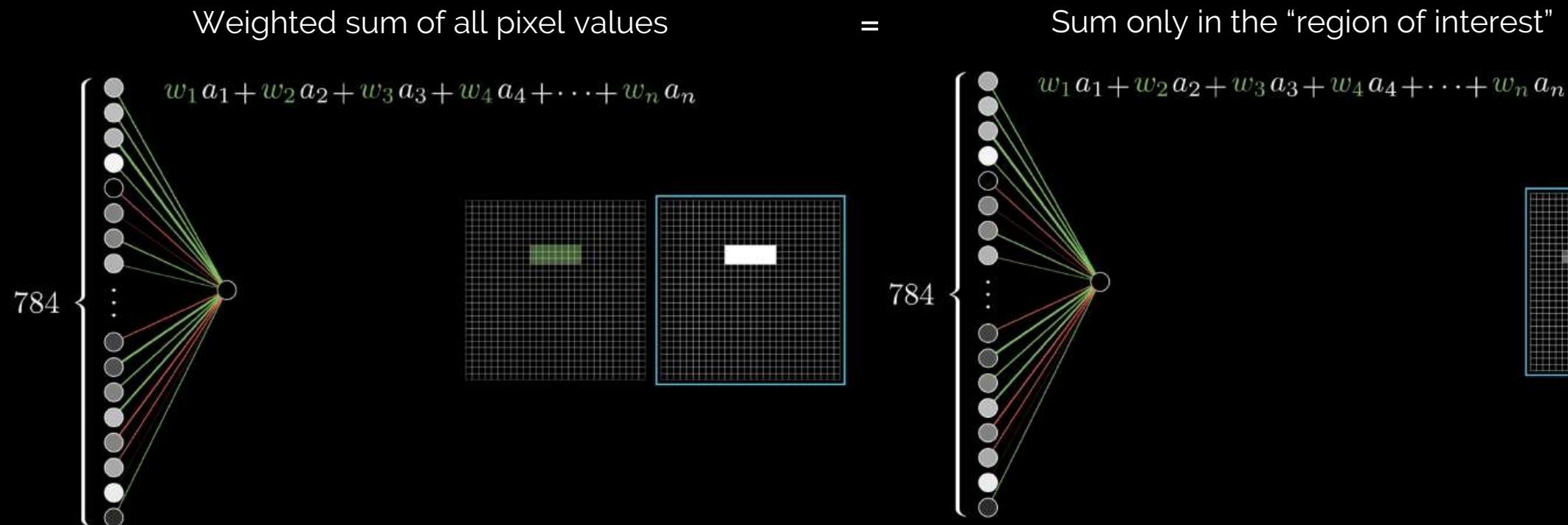


Deep Learning

Organize a grid of weights (green +, red -)

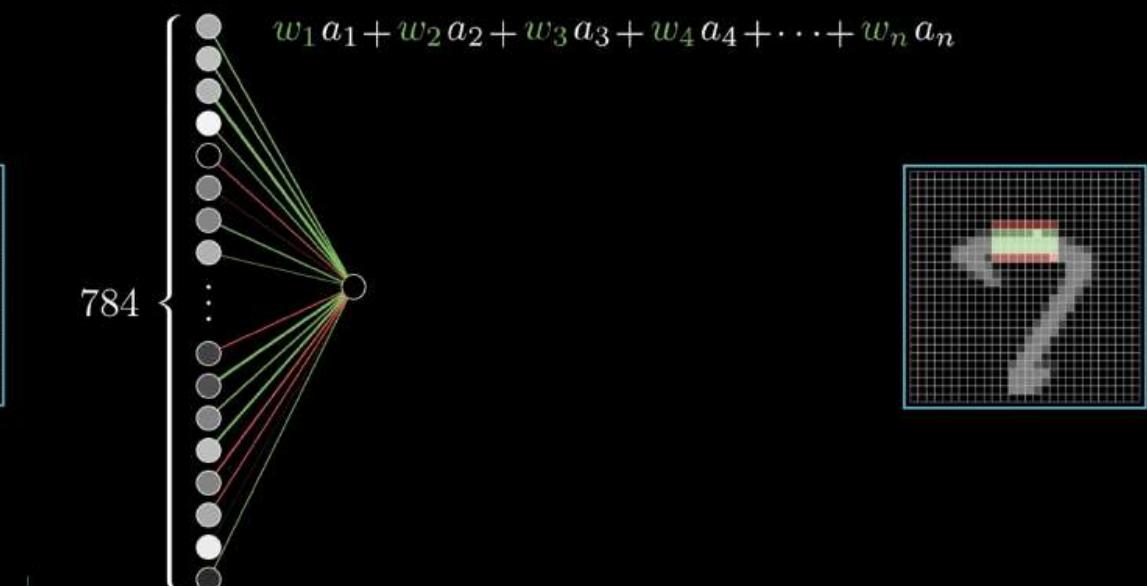
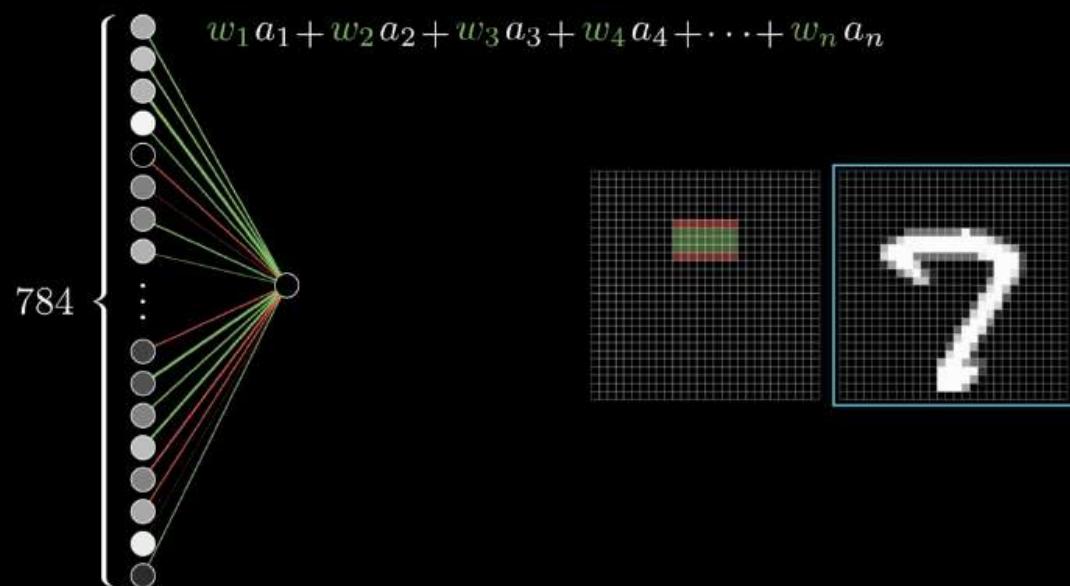


Deep Learning



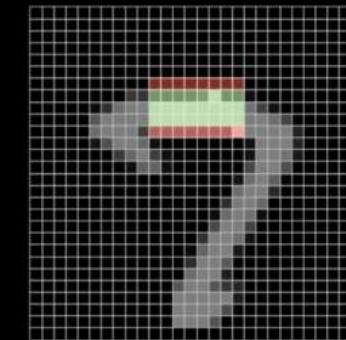
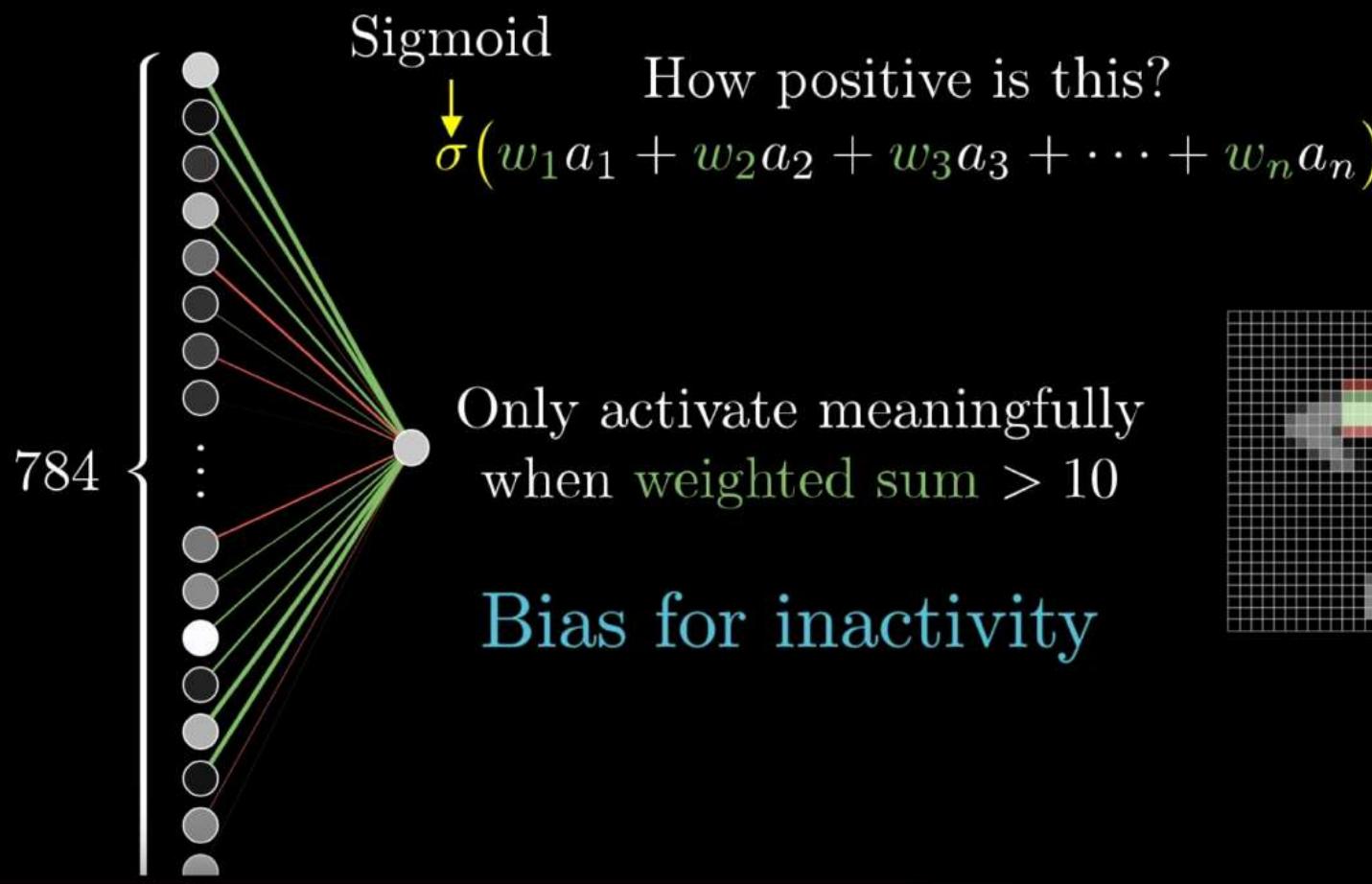
Deep Learning

Are there any negative weights associated with the “surrounding” pixels?

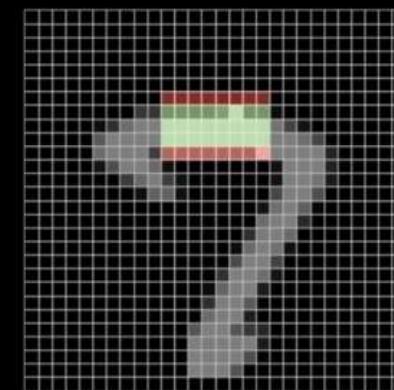
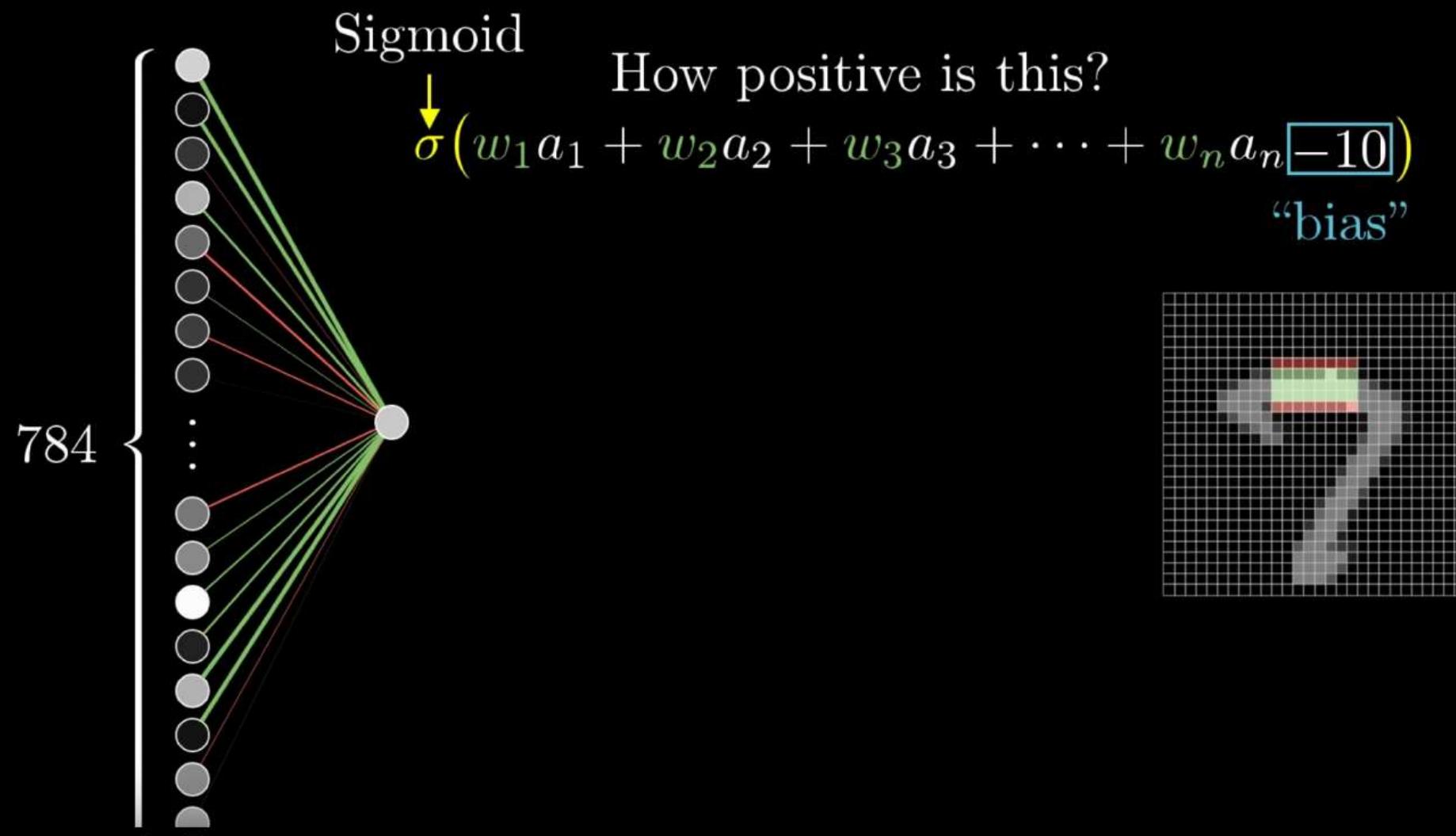


Deep Learning

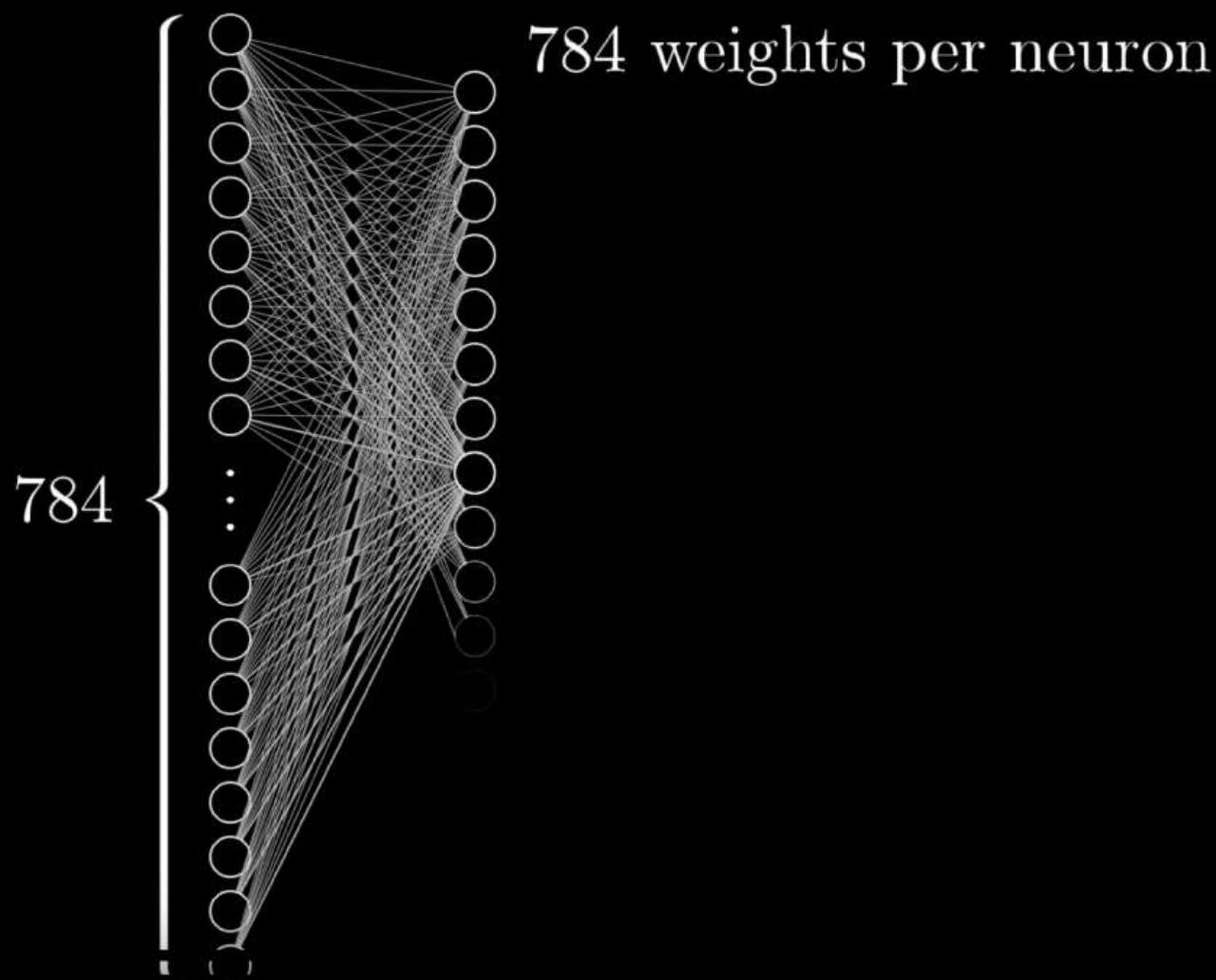
Activation of a neuron is a measure of "how positive" the relevant weighted sum is



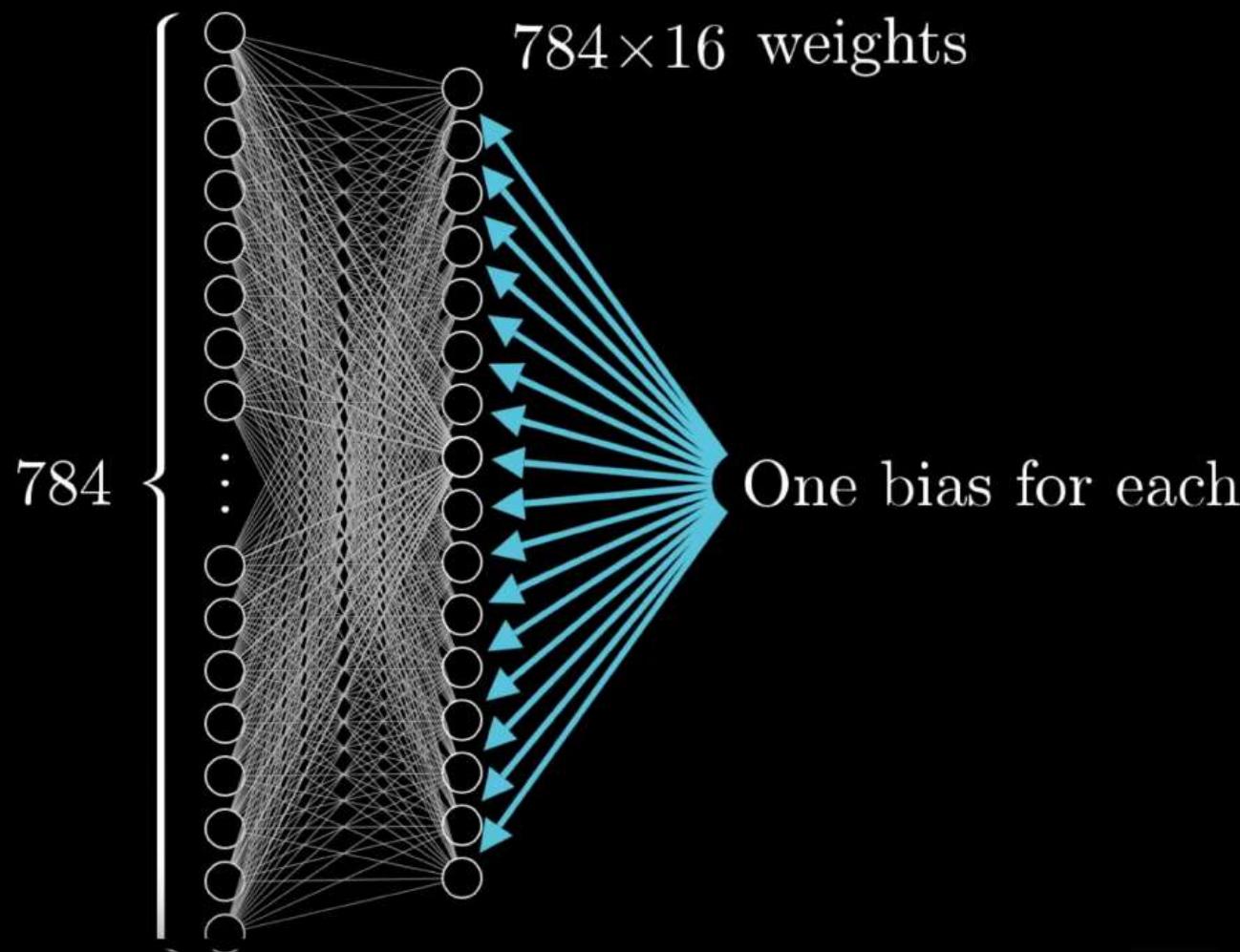
Deep Learning



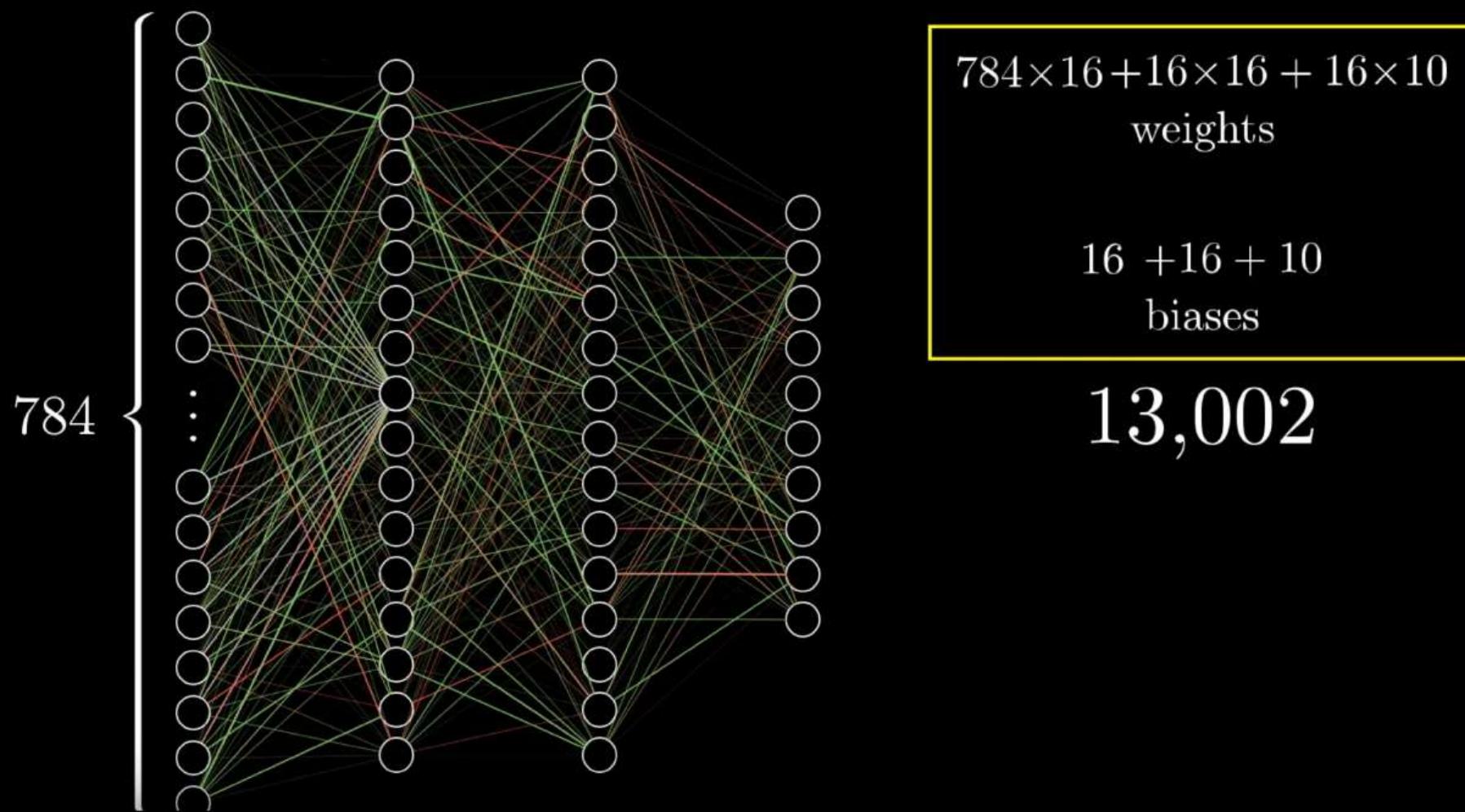
Deep Learning



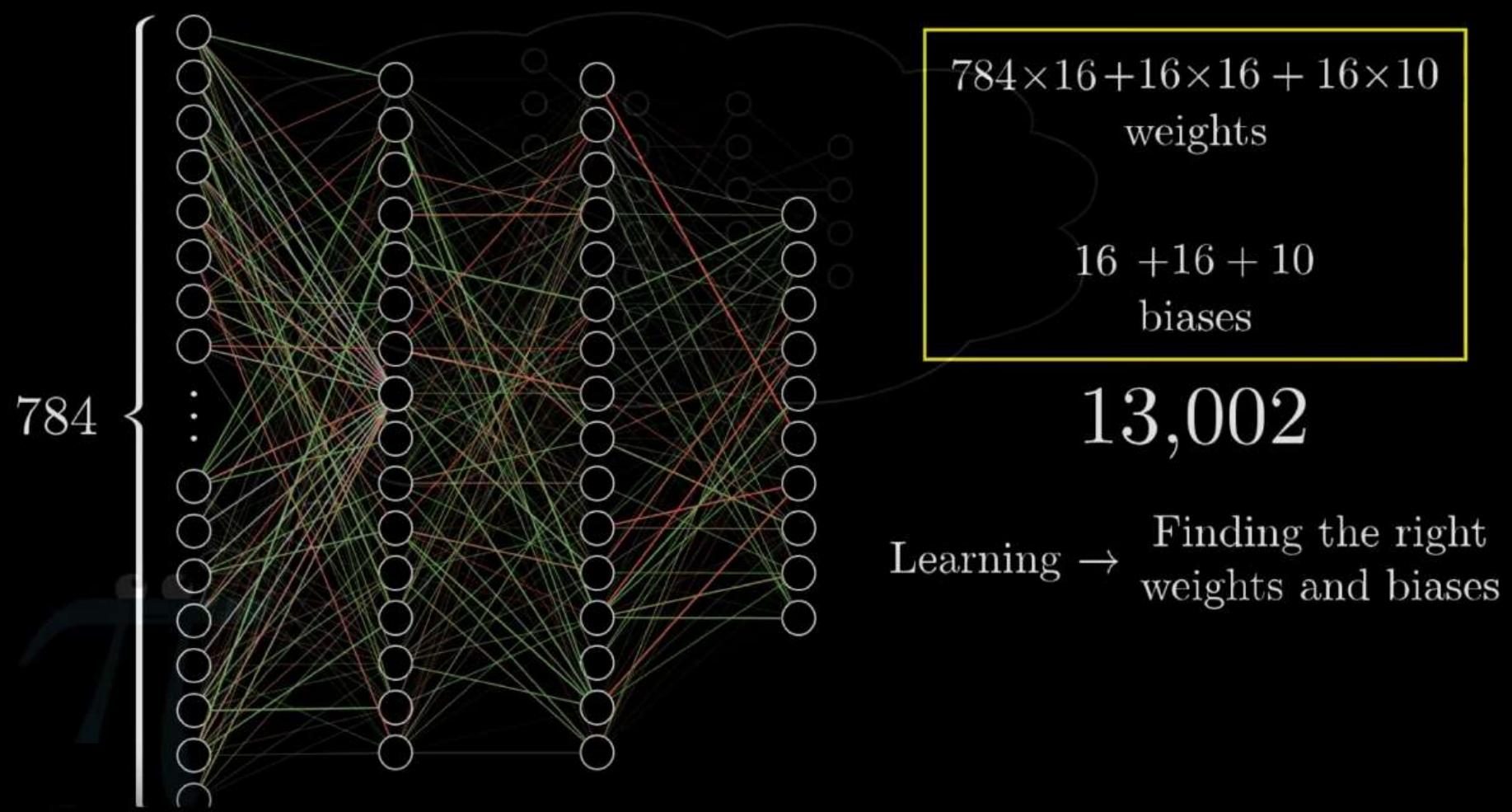
Deep Learning



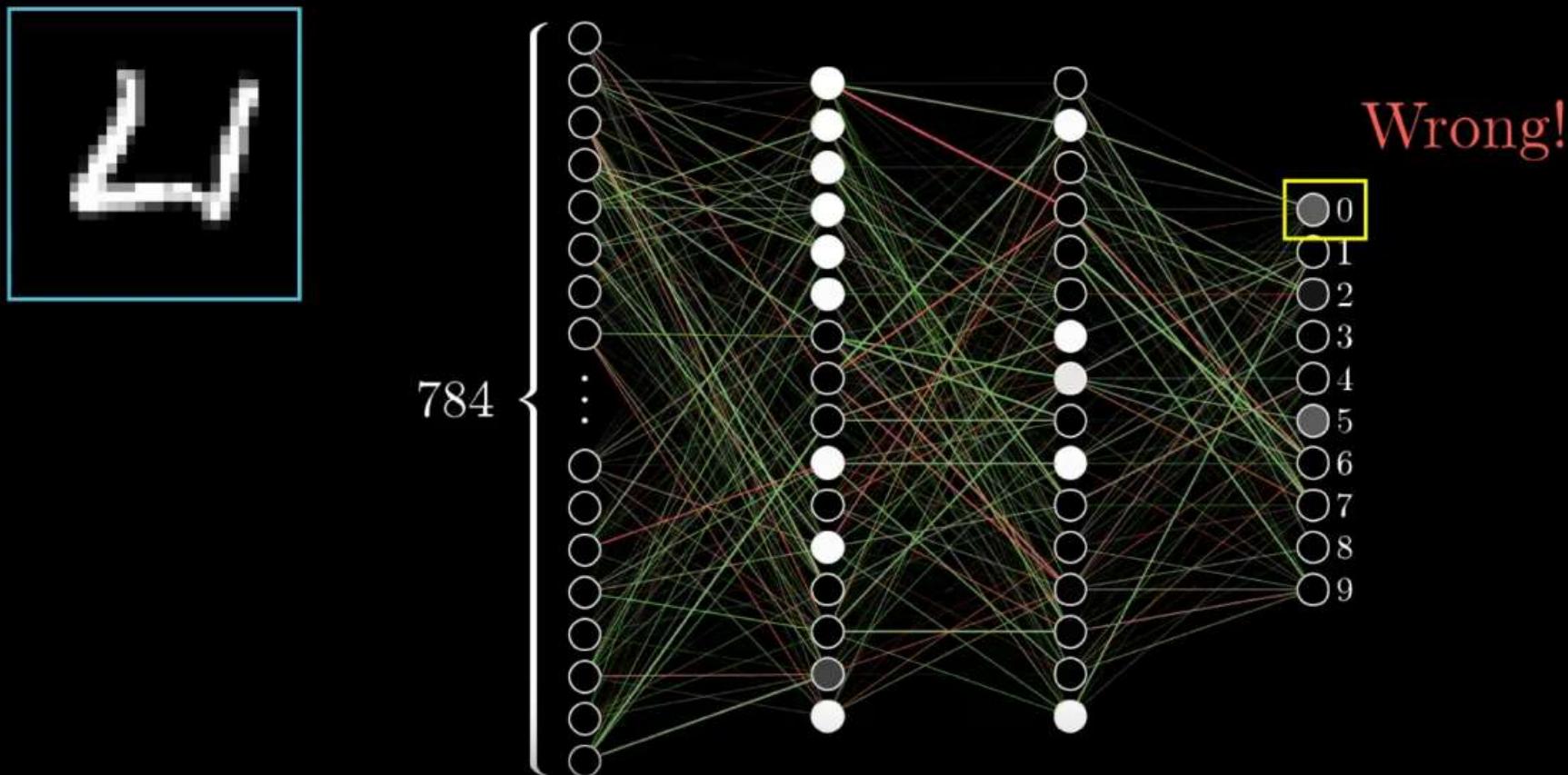
Deep Learning



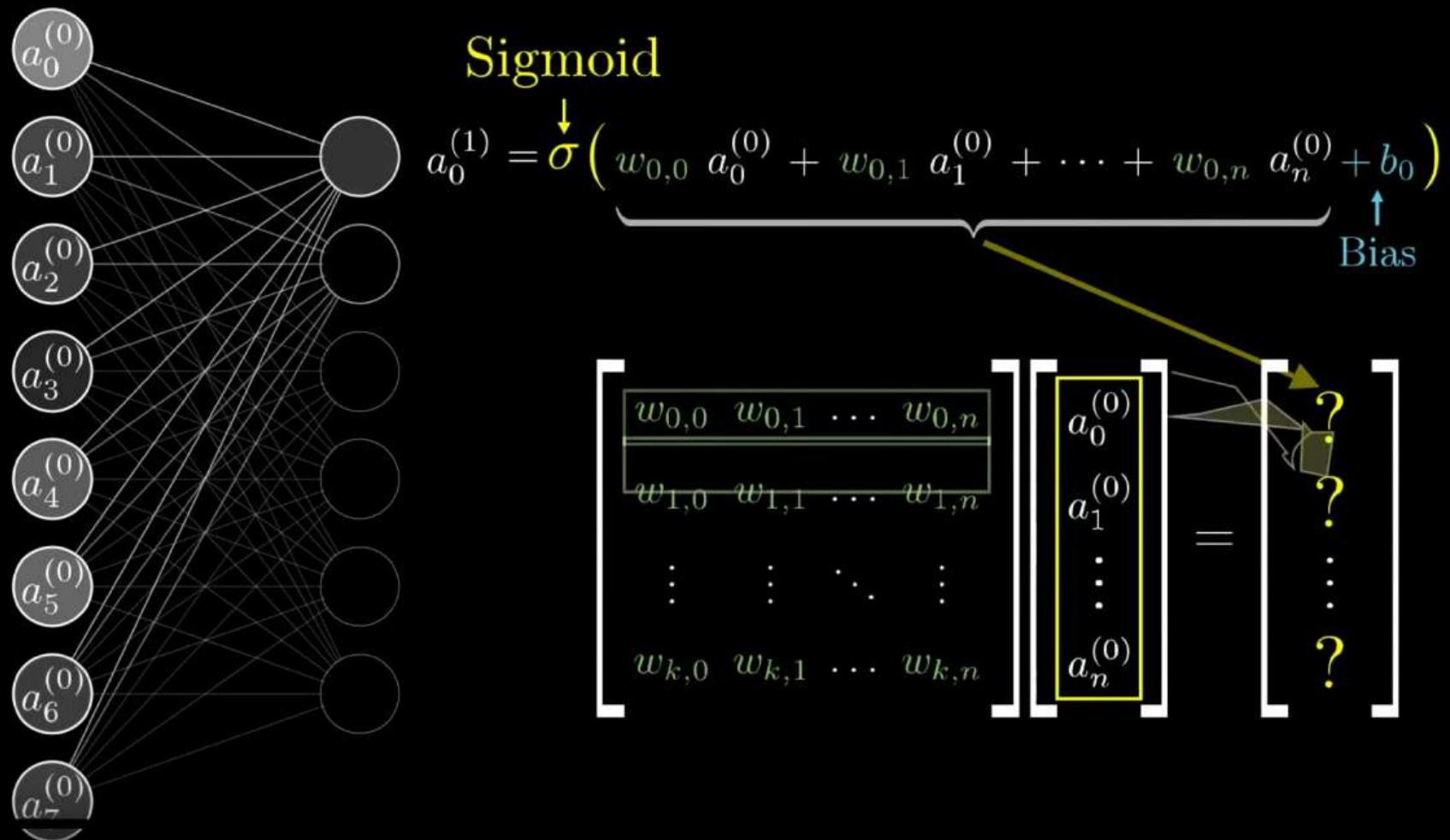
Deep Learning



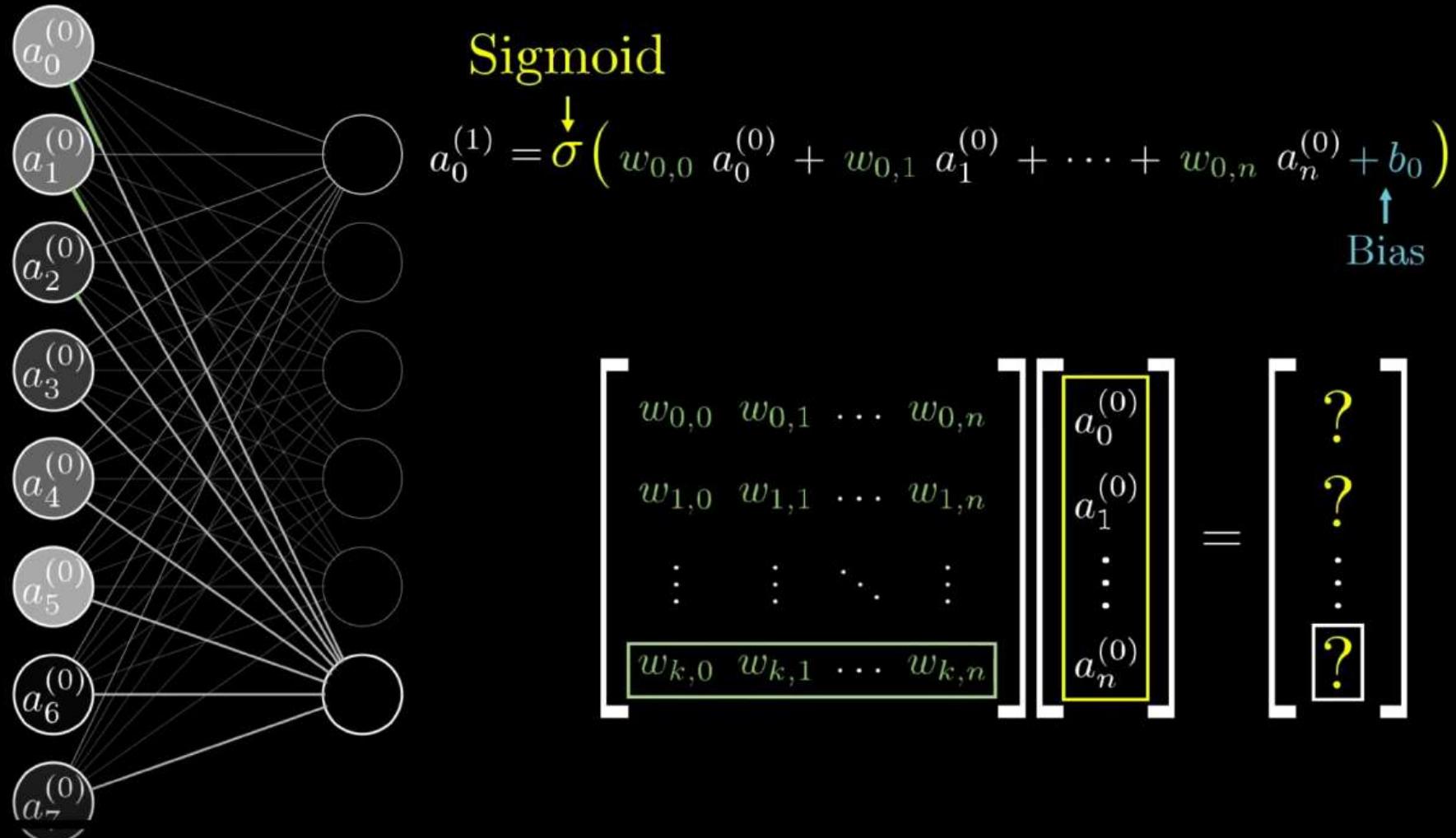
What weights are used here?
What are they doing?



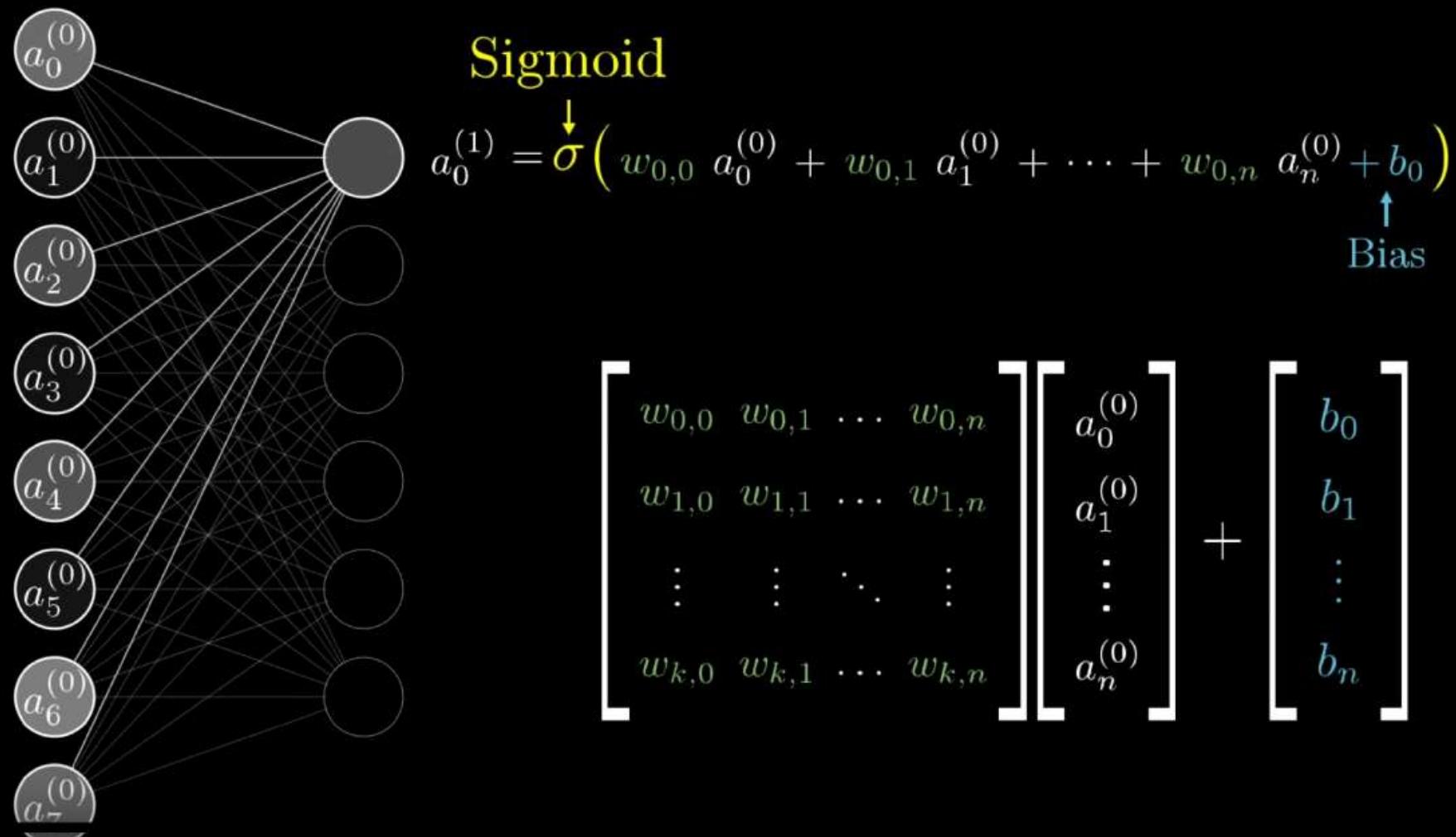
Deep Learning



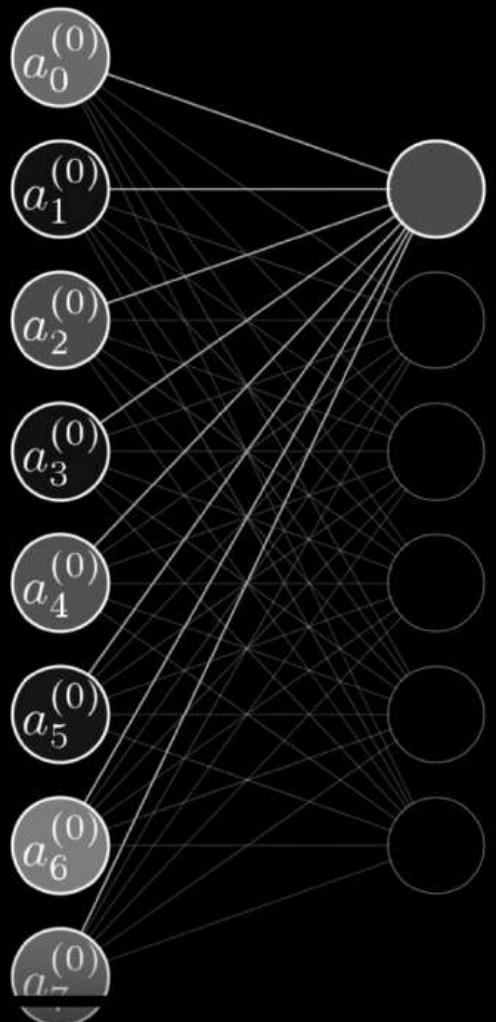
Deep Learning



Deep Learning



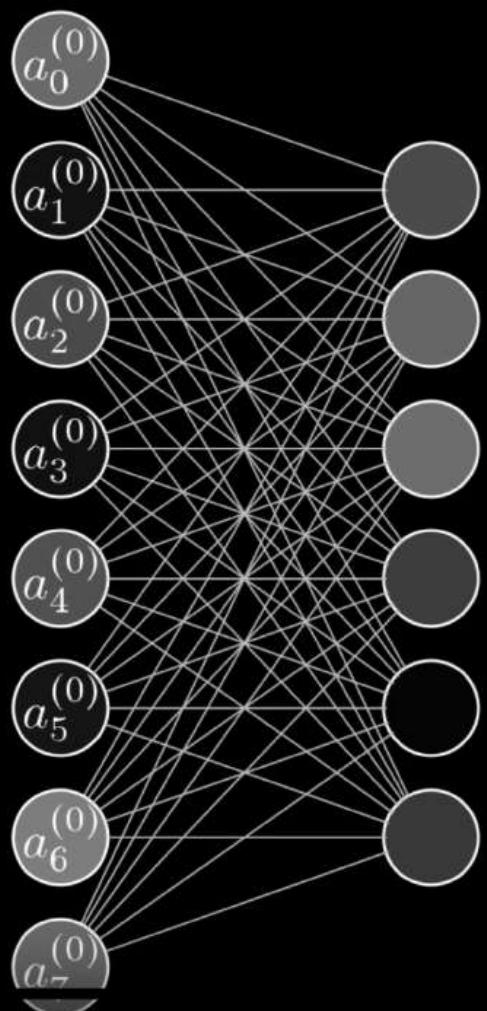
Deep Learning



$$\sigma \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} \sigma(x) \\ \sigma(y) \\ \sigma(z) \end{bmatrix}$$

$$\sigma \left(\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \right)$$

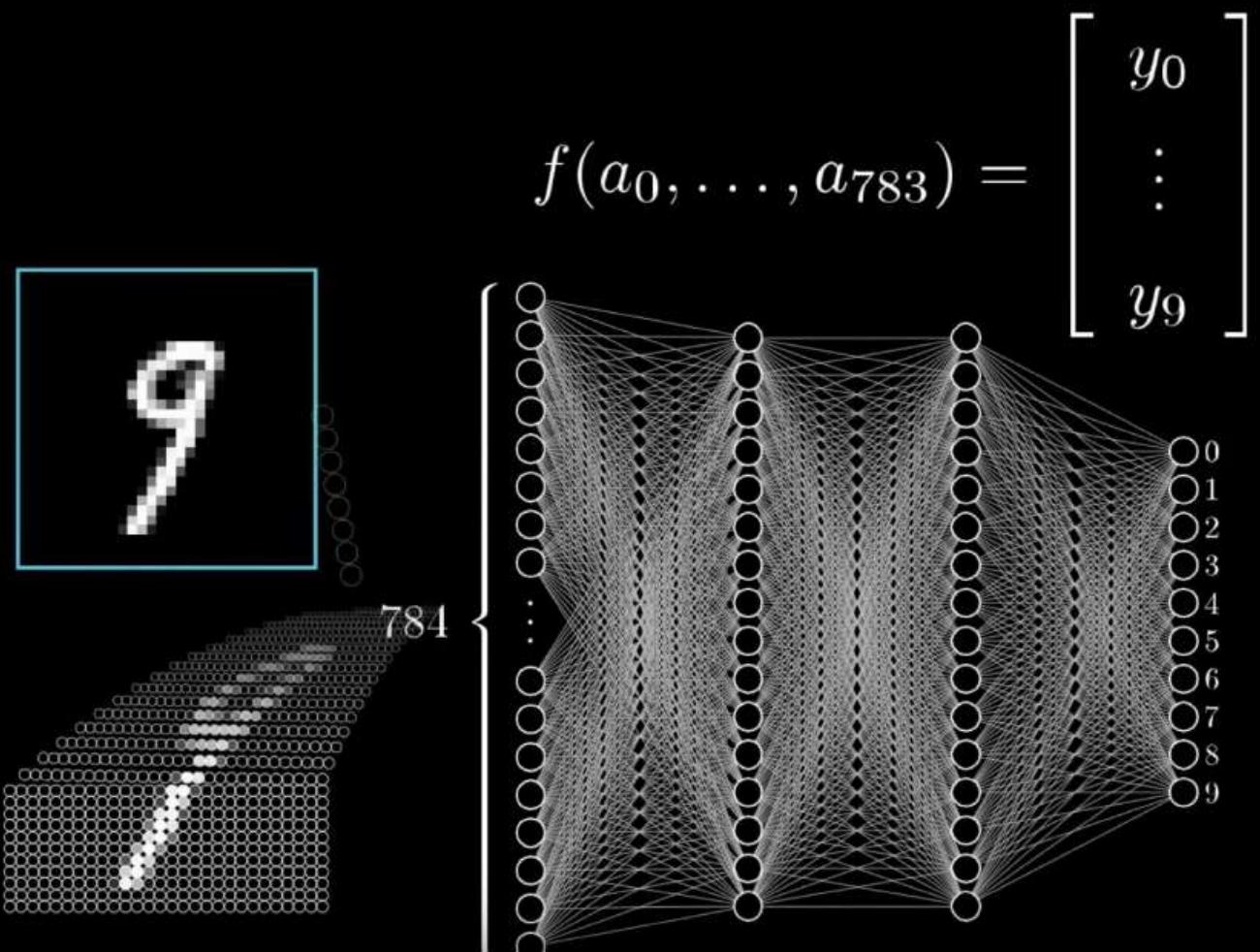
Deep Learning



$$\mathbf{a}^{(1)} = \sigma(\mathbf{W}\mathbf{a}^{(0)} + \mathbf{b})$$

$$\sigma \left(\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \right)$$

Deep Learning



Training multiple hidden layers by back propagation
algorithm

30 years to implement Learning Rules

20 years to implement training performance

Problems in training performance

- Vanishing Gradient
- Overfitting
- Computational load

Vanishing gradients

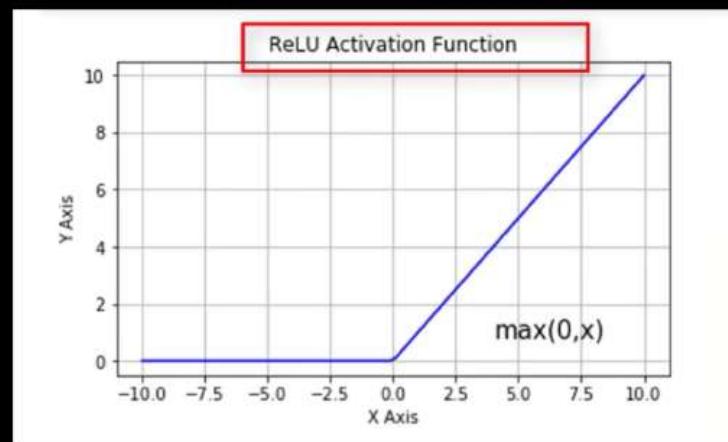
During training each weight receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training.

In some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training.

As one example of the problem cause, traditional activation functions such as the hyperbolic tangent function have gradients in the range $(-1, 1)$, and backpropagation computes gradients by the chain rule. This has the effect of multiplying n of these small numbers to compute gradients of the "front" layers in an n -layer network, meaning that the gradient (error signal) decreases exponentially with n while the front layers train very slowly.

Vanishing gradients

Can be solved using Rectified Linear Unit function
(ReLU) and its derivative as activation function



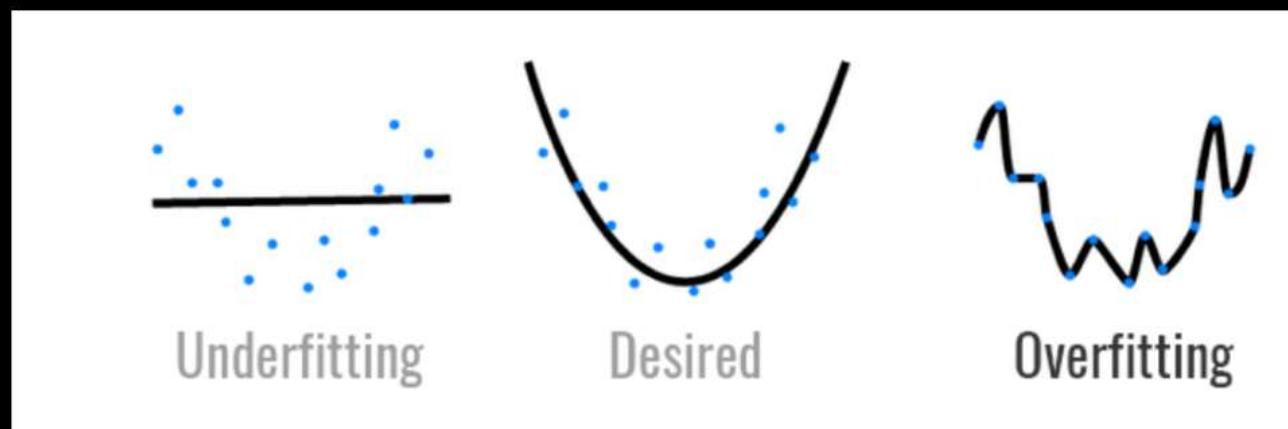
$$\varphi(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} = \max(0, x)$$

↓

$$\varphi'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

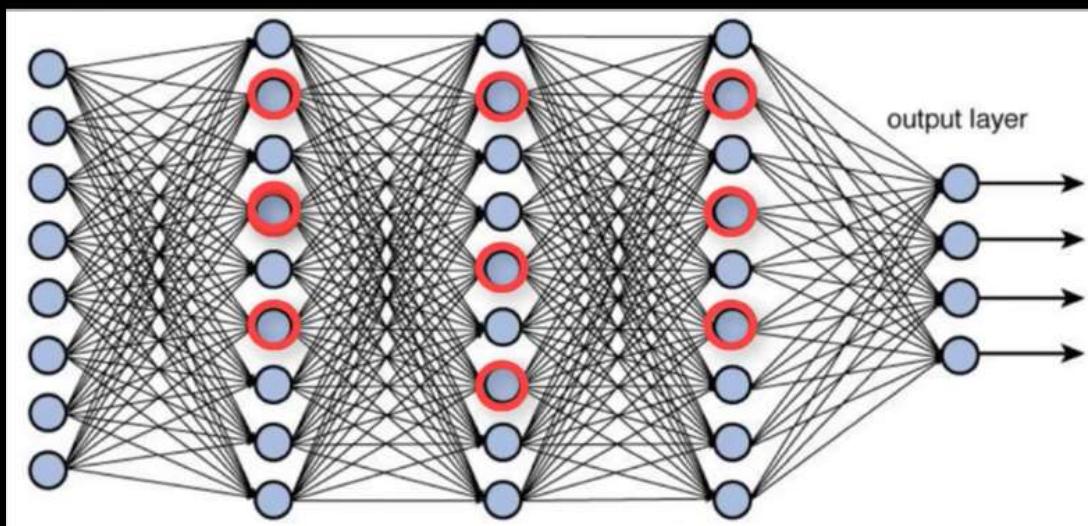
Overfitting

It refers to a model that models the training data too well. Instead of learning the general **distribution** of the data, the model learns the *expected output* for every data point.

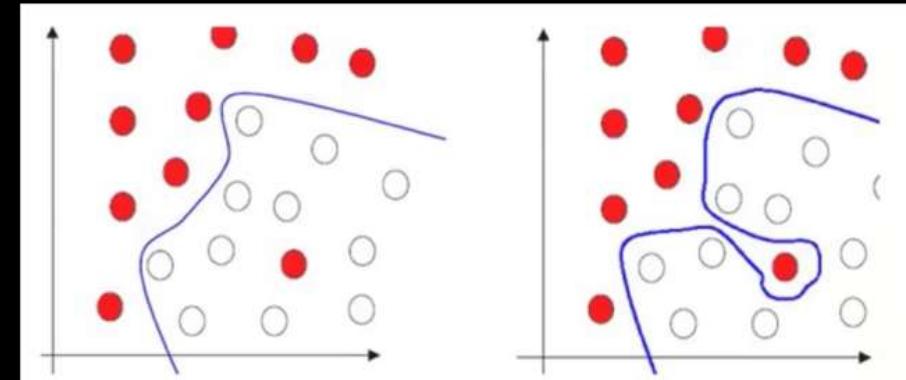


Overfitting

Can be solved using Dropout or Regularization



Dropout



Regularization

Computational load

Can be solved by GPU, Batch Normalization method

SOFTWARE CODES

Matlab: essential functions and scripts

Matlab: simple examples

Deep Learning

```
DeepLearning.m*  Untitled2*  +
1 function [w1, w2, w3, w4] = DeepLearning(w1, w2, w3, w4, input_Image, correct_Output)
2 alpha = 0.01; %to control the learning rate
3
4 N = 5;
5 for k = 1:N
6     reshaped_input_Image = reshape(input_Image(:,:,k), 25, 1);
7
8     input_of_hidden_layer1 = w1*reshaped_input_Image;
9     output_of_hidden_layer1 = ReLU(input_of_hidden_layer1);
10
11    input_of_hidden_layer2 = w2* output_of_hidden_layer1;
12    output_of_hidden_layer2 = ReLU(input_of_hidden_layer2);
13
14    input_of_hidden_layer3 = w3* output_of_hidden_layer2;
15    output_of_hidden_layer3 = ReLU(input_of_hidden_layer3);    ]
16
17    input_of_output_node = w4* output_of_hidden_layer3;
18    final_output = Softmax(input_of_output_node)
```

Deep Learning

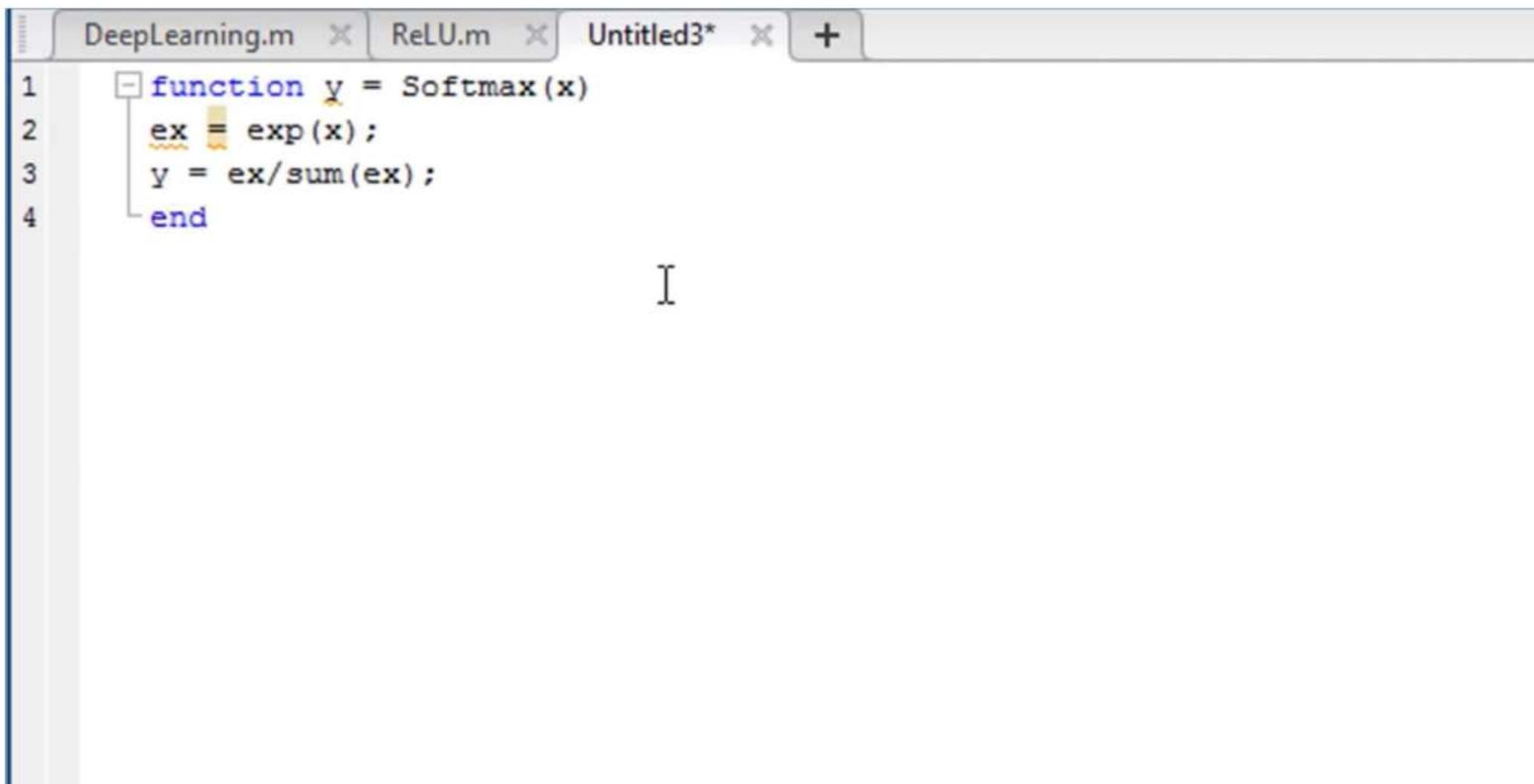


The image shows a screenshot of a MATLAB code editor. The window title bar has two tabs: "DeepLearning.m" and "Untitled2*". Below the tabs, there is a plus sign icon. The main workspace contains the following MATLAB code:

```
1 function y = ReLU(x)
2     y = max(0,x);
3 end
```

A cursor is visible at the end of the third line, indicating the code is ready for further input.

Deep Learning



The image shows a screenshot of a MATLAB code editor. The window title bar contains three tabs: "DeepLearning.m" (selected), "ReLU.m", and "Untitled3*". Below the tabs, there is a plus sign icon. The main workspace displays the following MATLAB code:

```
1 function y = Softmax(x)
2     ex = exp(x);
3     y = ex/sum(ex);
4 end
```

A cursor is visible at the end of the third line of code.

Deep Learning

The screenshot shows a MATLAB code editor window with the tab bar at the top containing four tabs: DeepLearning.m, ReLU.m, Softmax.m, and TrainingNetwork.m*. The TrainingNetwork.m* tab is currently active. The main workspace displays the following MATLAB code:

```
1 - input_Image = zeros(5,5,5);
2
3 - input_Image(:,:,:,1) = [1 0 0 1 1;
4 -                         1 1 0 1 1;
5 -                         1 1 0 1 1;
6 -                         1 1 0 1 1;
7 -                         1 0 0 0 1;
8 -                         ];
9 - input_Image(:,:,:,2) = [0 0 0 0 1;
10 -                         1 1 1 1 0;
11 -                         1 0 0 0 1;
12 -                         0 1 1 1 1;
13 -                         0 0 0 0 0;
14 -                         ];
15 input_Image(:,:,:,3) = []
```

Deep Learning

The screenshot shows a MATLAB interface with several tabs at the top: DeepLearning.m, ReLU.m, Softmax.m, and TrainingNetwork.m*. The TrainingNetwork.m* tab is active. The code in the editor is as follows:

```
30      1 1 1 1 0;
31      0 0 0 0 1;
32    ];
33 -  correct_Output = [1 0 0 0 0;
34      0 1 0 0 0;
35      0 0 1 0 0;
36      0 0 0 1 0;
37      0 0 0 0 1;
38    ];
39 -  w1 = 2*rand(20,25)-1;
40 -  w2 = 2*rand(20,20)-1;
41 -  w3 = 2*rand(20,20)-1;
42 -  w4 = 2*rand(5,20)-1;
43
44 - for epoch = 1:10000
45 -     [w1, w2, w3, w4] = DeepLearning(w1, w2, w3, w4, input_Image, correct_Output);
46 - end
47 save('DeepNeuralNetwork.mat')
```

Deep Learning

The screenshot shows the MATLAB interface with several open tabs at the top: DeepLearning.m, ReLU.m, Softmax.m, TrainingNetwork.m, and TestDeepLearning.m. The TestDeepLearning.m tab is active. Below the tabs is a code editor containing the following MATLAB script:

```
1 - load('DeepNeuralNetwork.mat');
2
3 - input_Image = [1 0 0 1 1;
4 -                 1 1 0 1 1;
5 -                 1 1 0 1 1;
6 -                 1 1 0 1 1;
7 -                 1 0 0 0 1;
8 -                 ];
9
10 - input_Image = reshape(input_Image, 25, 1);
11
12 - input_of_hidden_layer1 = w1*input_Image;
13 - output_of_hidden_layer1 = ReLU(input_of_hidden_layer1);
```

Below the code editor is the Command Window. It displays a welcome message for new users and a command prompt. The user has entered the command `>> TestDeepLearning`, and the output is:

```
final_output =
1.0000
0.0000
0.0000
0.0000
0.0000
```

The final output value, `1.0000`, is highlighted in blue.

Deep Learning

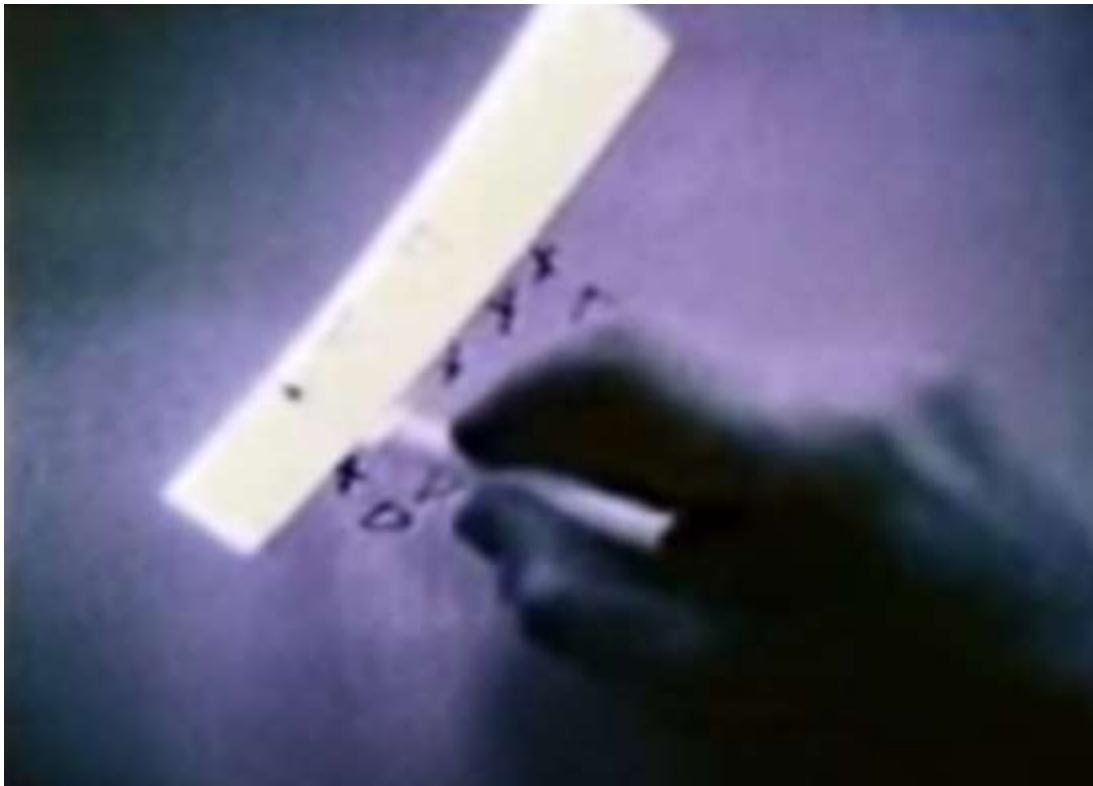
The screenshot shows the MATLAB IDE interface. The top menu bar has tabs for 'DeepLearning.m', 'ReLU.m', 'Softmax.m', 'TrainingNetwork.m', 'TestDeepLearning.m*', and a '+' button. The main workspace shows the following MATLAB code:

```
8
9
10 -    input_Image = reshape(input_Image, 25, 1);
11
12 -    input_of_hidden_layer1 = w1*input_Image;
13 -    output_of_hidden_layer1 = ReLU(input_of_hidden_layer);
14
15 -    input_of_hidden_layer2 = w2*output_of_hidden_layer1;
16 -    output_of_hidden_layer2 = ReLU(input_of_hidden_layer2);
17
18 -    input_of_hidden_layer3 = w3*output_of_hidden_layer2;
19 -    output_of_hidden_layer3 = ReLU(input_of_hidden_layer3);
20
21     input_of_output_node = w4*output_of_h
```

The line 'input_of_output_node' is highlighted with a blue underline. Below the workspace, the 'Command Window' tab is active, showing the message: 'New to MATLAB? See resources for [Getting Started.](#)' and the command '>> TestDeepLearning'. A yellow bar at the bottom also contains the 'Getting Started.' link.

CONVOLUTIONAL NEURAL NETWORKs

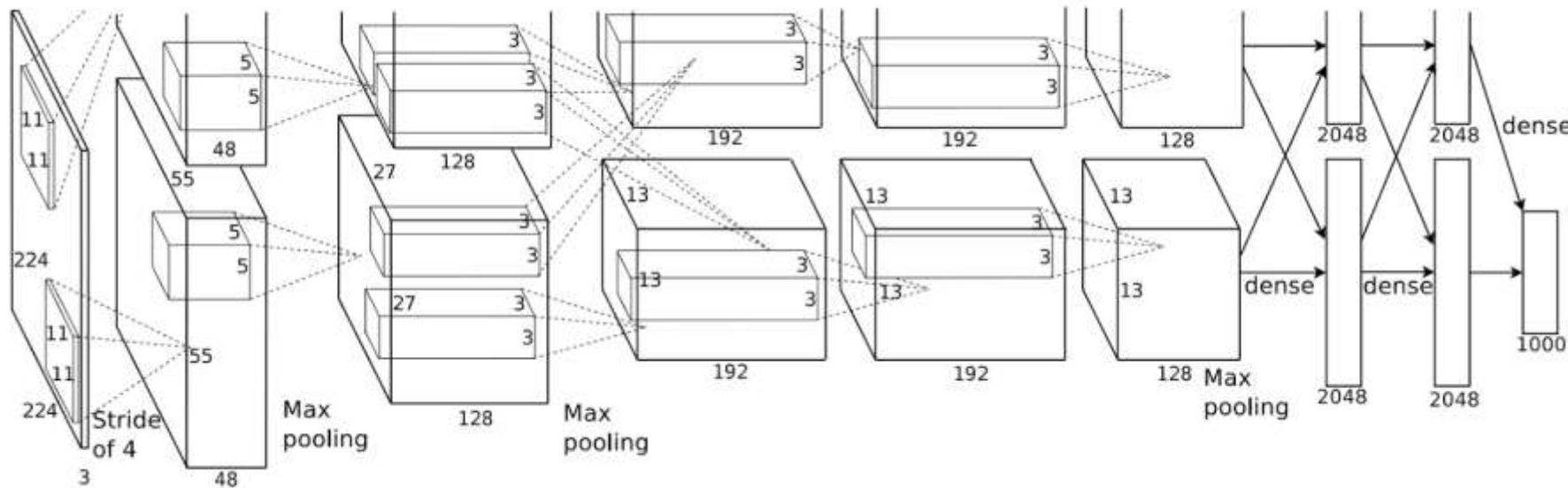
Convolutional Neural Networks



Hubel and Wiesel

<https://www.youtube.com/watch?v=Cw5PKV9Rj3o>

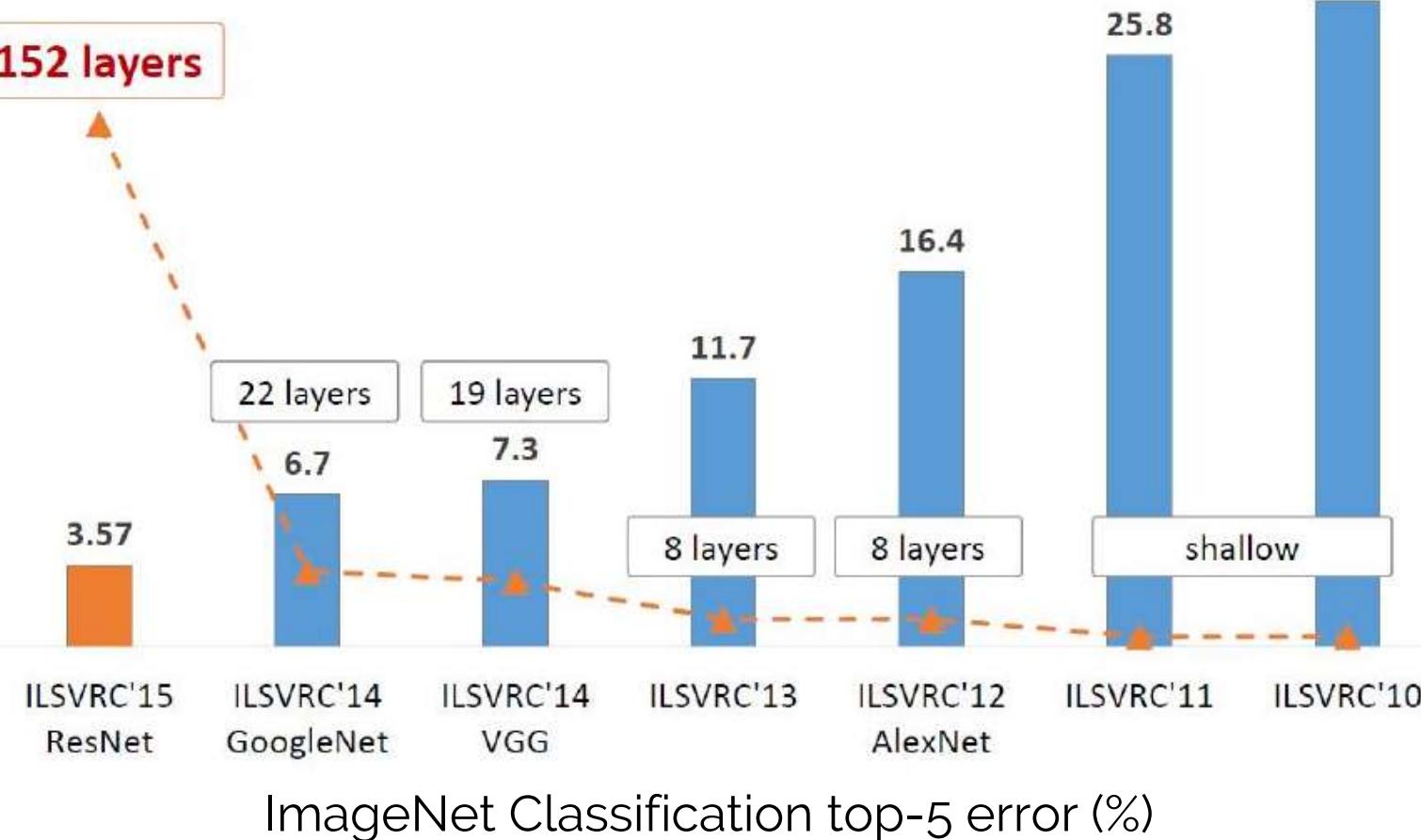
Convolutional Neural Networks



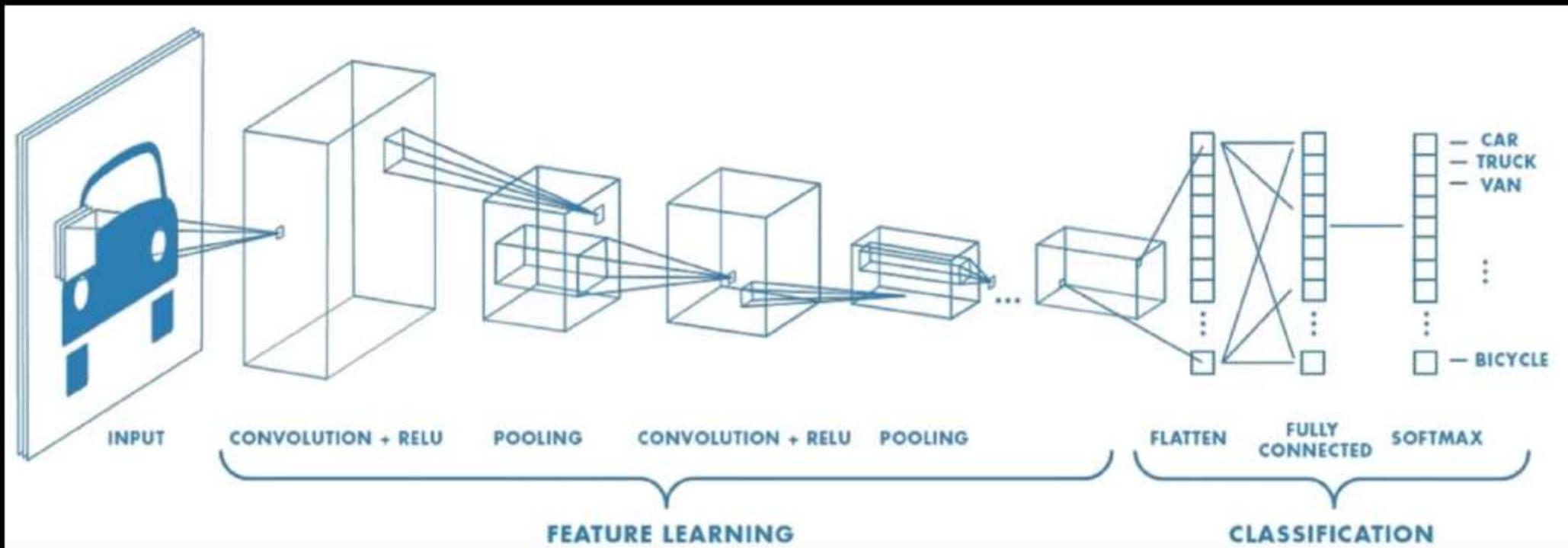
Krizhevsky, Sutskever, and Hinton, 2012

Convolutional Neural Networks

Revolution of Depth

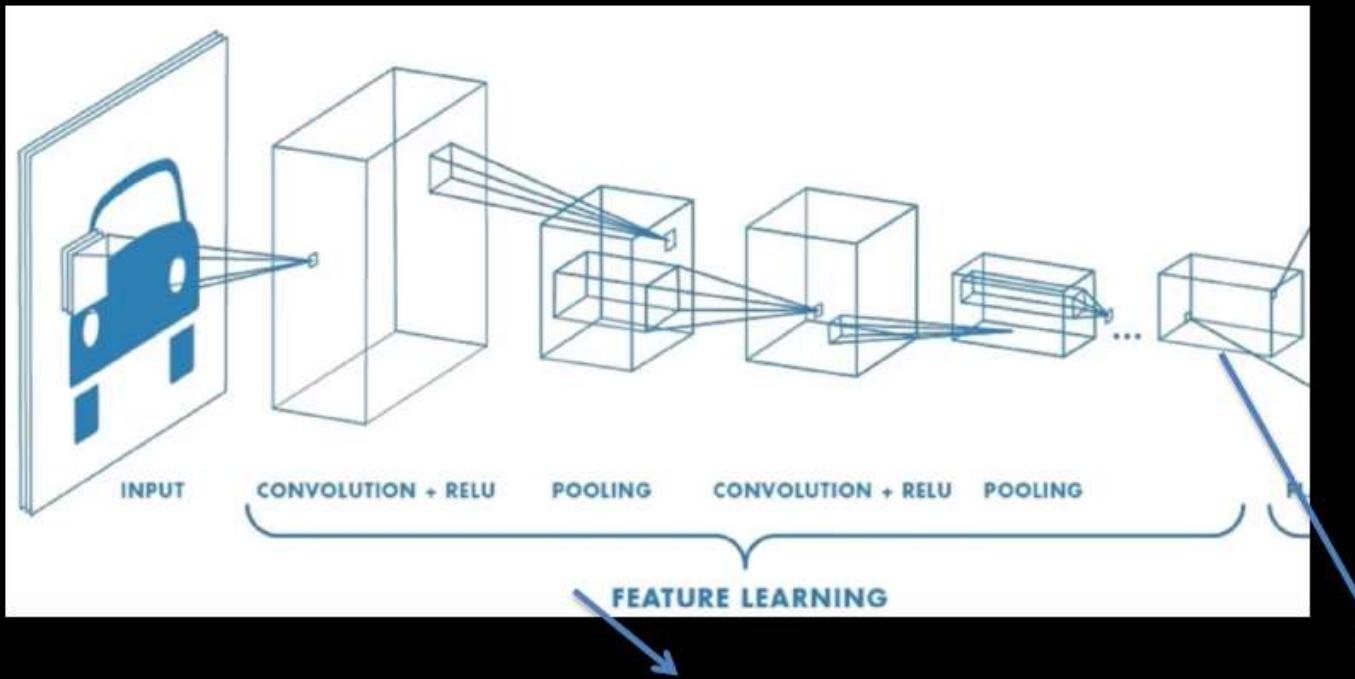


Convolutional Neural Networks



Convolutional Neural Networks

Feature Learning



Feature extraction network

Convolutional layers

Paired pooling layers

Extracted features

Convolution operation: a collection of digital filters
(2D)

Pooling operation combines adjacent pixels into 1 pixel (2D) (reduce the size of the image)

Convolution layers converts images into features map

Different filters (convolutional filters) convert images into different feature maps

N convolution layers x M convolutional filters → NxM feature maps

Convolutional Neural Networks

Convolution filter

A diagram illustrating a convolution operation. On the left is a 4x4 input matrix:

| | | | |
|----|---|---|---|
| 1 | 1 | 1 | 3 |
| 4 | 6 | 4 | 8 |
| 30 | 0 | 1 | 5 |
| 0 | 2 | 2 | 4 |

Next to it is a convolution filter:

$$* \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The result of the multiplication is shown in a red box:

$$1 \times 1 + 1 \times 0 + 4 \times 0 + 6 \times 1 = 7$$

Diagram illustrating a sequence of convolution operations:

Step 1: Input $\begin{bmatrix} 1 & 1 & 1 & 3 \\ 4 & 6 & 4 & 8 \\ 30 & 0 & 1 & 5 \\ 0 & 2 & 2 & 4 \end{bmatrix}$ multiplied by filter $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ results in a 2x2 intermediate output:

| | |
|---|---|
| 7 | 5 |
| | |

Step 2: Intermediate output $\begin{bmatrix} 7 & 5 \\ \text{ } & \text{ } \end{bmatrix}$ multiplied by filter $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ results in a 1x1 final output:

| |
|---|
| 9 |
|---|

Step 3: Input $\begin{bmatrix} 1 & 1 & 1 & 3 \\ 4 & 6 & 4 & 8 \\ 30 & 0 & 1 & 5 \\ 0 & 2 & 2 & 4 \end{bmatrix}$ multiplied by filter $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ results in a 2x2 intermediate output:

| | |
|---|---|
| 7 | 5 |
| 4 | |

Dashed arrow points from the second step's intermediate output to the third step's input.

Final result: $\begin{bmatrix} 32 & 2 & 5 \end{bmatrix}$

Convolutional Neural Networks

Convolution filter: first example

The diagram shows a 4x4 input matrix and a 2x2 filter matrix. The input matrix has values: 1, 1, 1, 3; 4, 6, 4, 8; 30, 0, 1, 5; 0, 2, 2, 4. A red circle highlights the bottom-left 2x2 patch [30, 0; 0, 2]. The filter matrix has values: 1, 0; 0, 1. The result of the multiplication is shown as a 3x3 output matrix: 7, 5, 9; 4, 7, 9; 32, 2, 5. The value 32 is highlighted in gray.

$$\begin{bmatrix} 1 & 1 & 1 & 3 \\ 4 & 6 & 4 & 8 \\ 30 & 0 & 1 & 5 \\ 0 & 2 & 2 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 7 & 5 & 9 \\ 4 & 7 & 9 \\ 32 & 2 & 5 \end{bmatrix}$$

High value when filter mach image patch

The diagram shows a 4x4 input matrix and a 2x2 filter matrix. The input matrix has values: 1, 1, 1, 3; 4, 6, 4, 8; 30, 0, 1, 5; 0, 2, 2, 4. A red box highlights the top-left 2x2 patch [1, 1; 4, 6]. The filter matrix has values: 1, 0; 0, 1. The result of the multiplication is shown as a 3x3 output matrix: 7, 5, 9; 4, 7, 9; 32, 2, 5. The value 4 is highlighted in red.

$$\begin{bmatrix} 1 & 1 & 1 & 3 \\ 4 & 6 & 4 & 8 \\ 30 & 0 & 1 & 5 \\ 0 & 2 & 2 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 7 & 5 & 9 \\ 4 & 7 & 9 \\ 32 & 2 & 5 \end{bmatrix}$$

Low value when filter does not mach image patch

Convolutional Neural Networks

Convolution filter: second example

The diagram illustrates a convolution operation. On the left is a 4x4 input image grid with values:

| | | | |
|----|---|---|---|
| 1 | 1 | 1 | 3 |
| 4 | 6 | 4 | 8 |
| 30 | 0 | 1 | 5 |
| 0 | 2 | 2 | 4 |

A red oval highlights the value 30 in the second row, third column. In the center is a 2x2 filter matrix:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

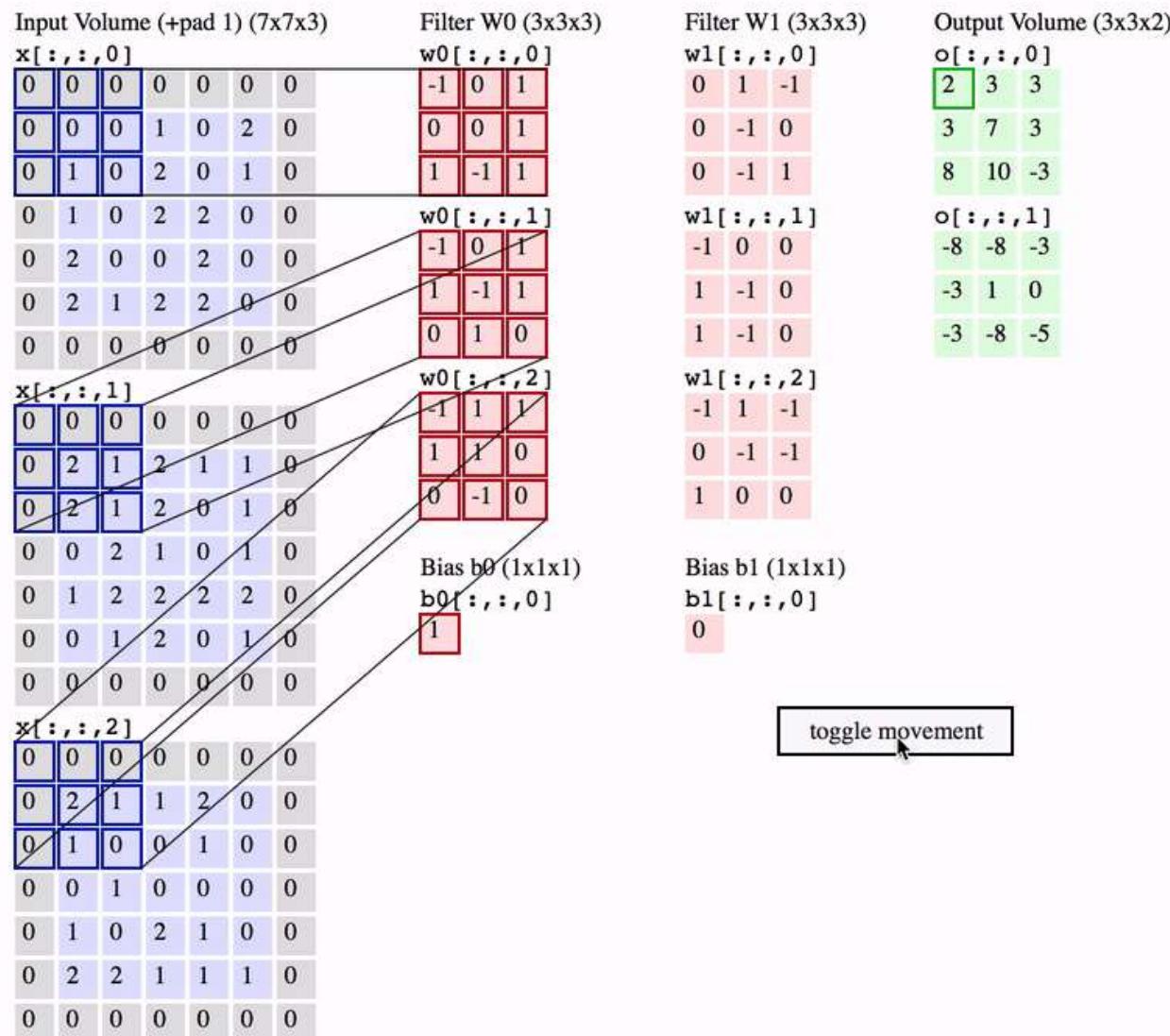
The filter is also highlighted with a red oval. To the right of the filter is an equals sign followed by a 3x3 output image grid with values:

| | | |
|----|---|---|
| 5 | 7 | 7 |
| 36 | 4 | 9 |
| 0 | 3 | 7 |

High value when filter matches image patch

Low value when filter does not match image patch

Convolutional Neural Networks



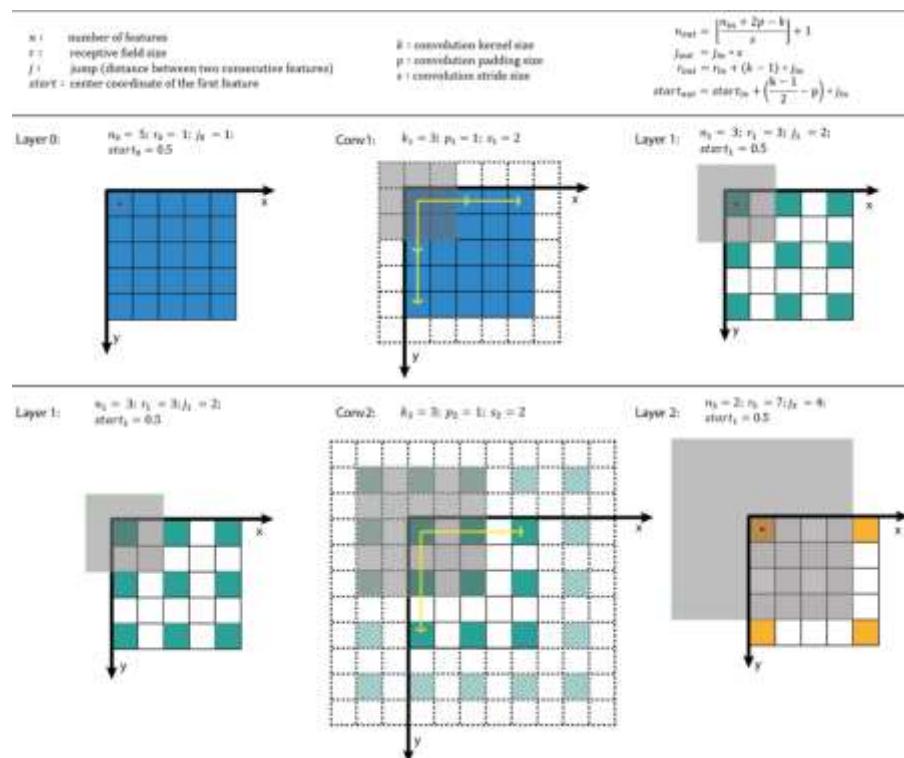
Convolutional Neural Networks

CONVOLUTION HYPERPARAMETERS

DEPTH

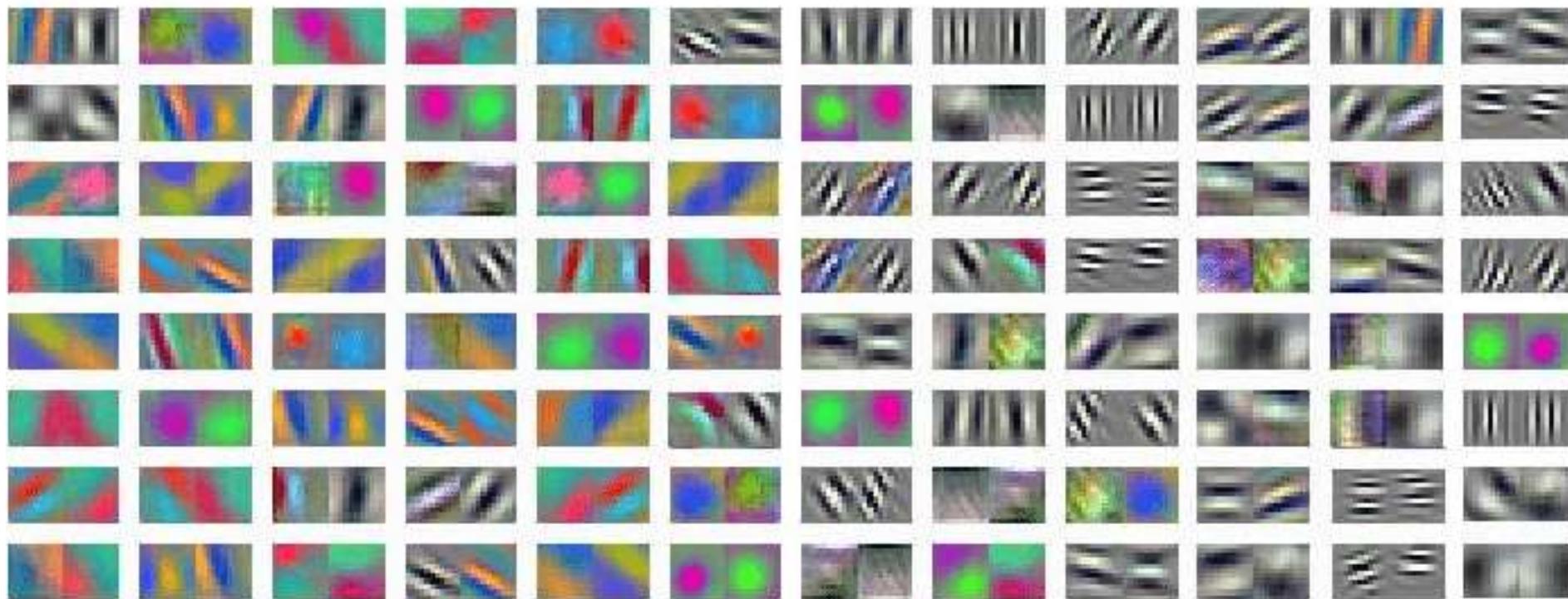
STRIDE

ZERO-PADDING

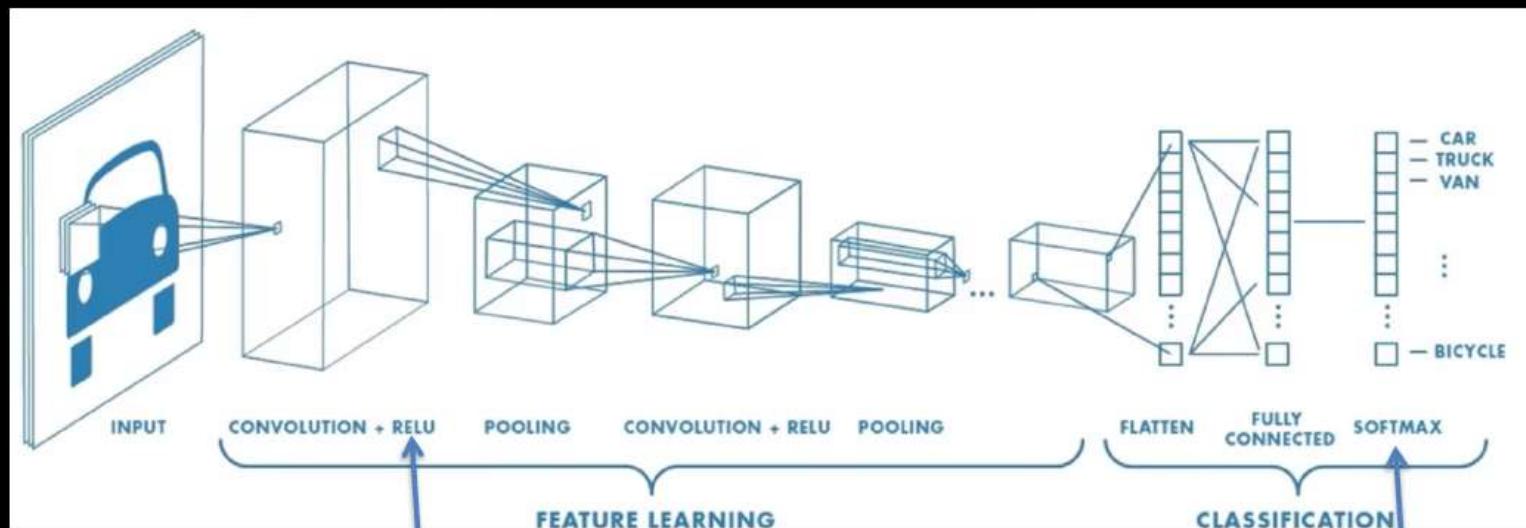


Convolutional Neural Networks

ALEXNET CONV1 LEARNED FILTERS



Convolutional Neural Networks



Activation function

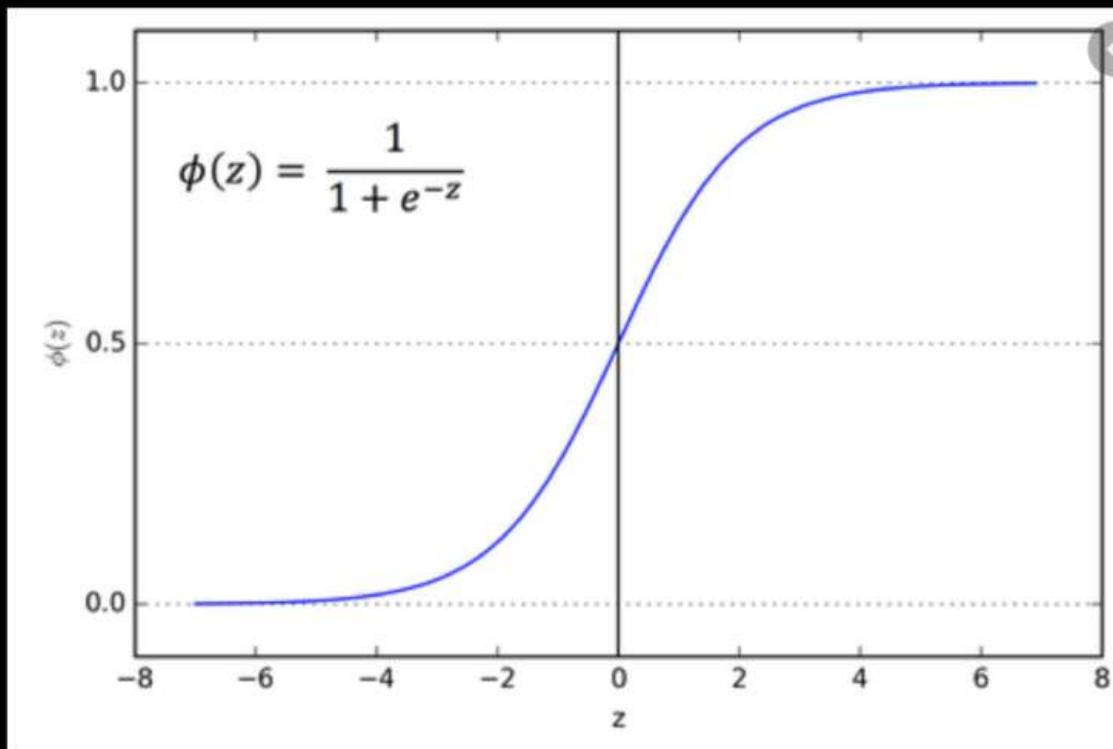
RELU
SIGMOID
TANH

Activation function

SOFTMAX

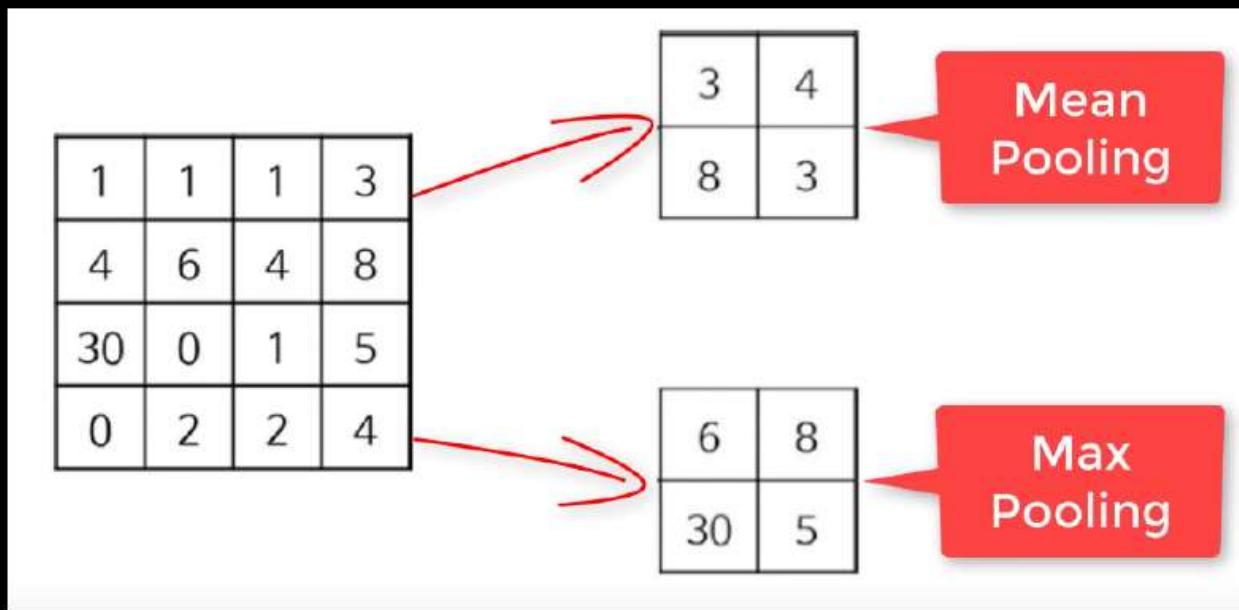
Convolutional Neural Networks

SOFTMAX



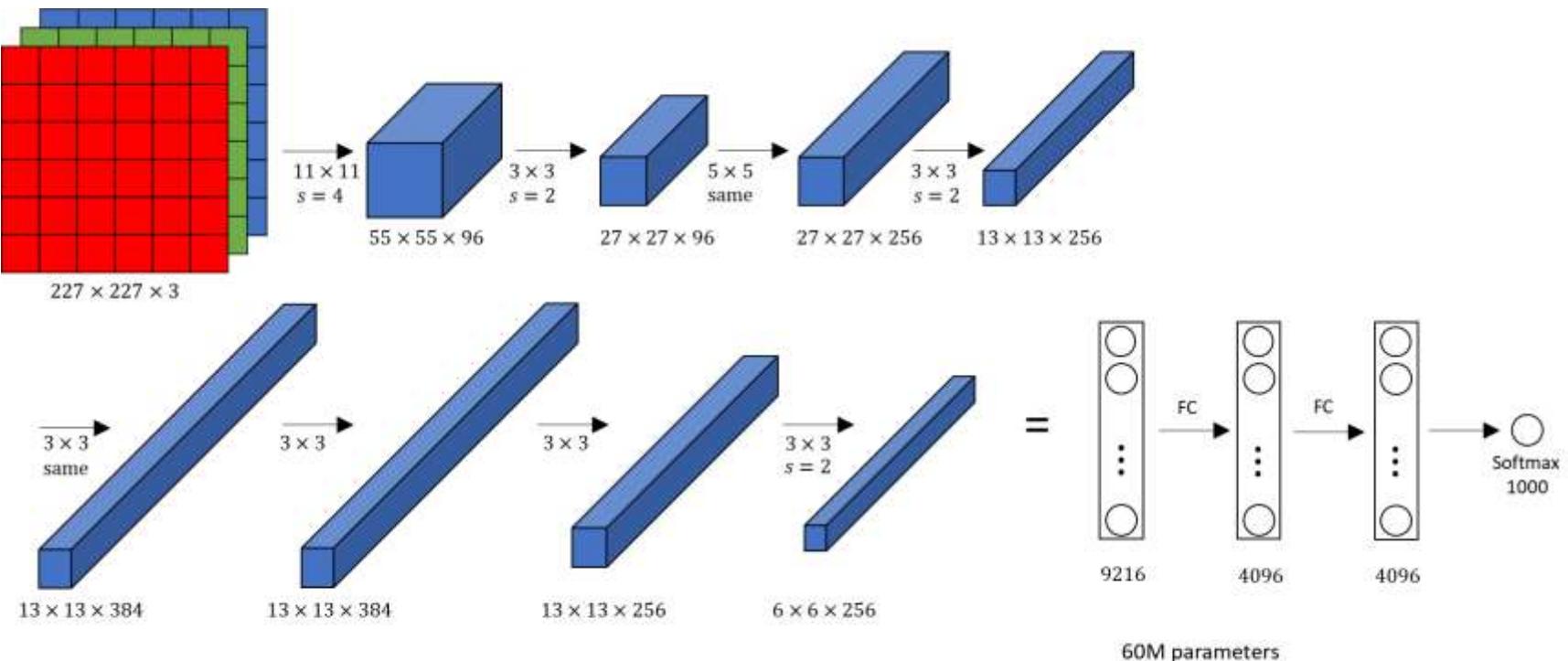
It converts the output of the last layer into a probability distribution

Pooling layer



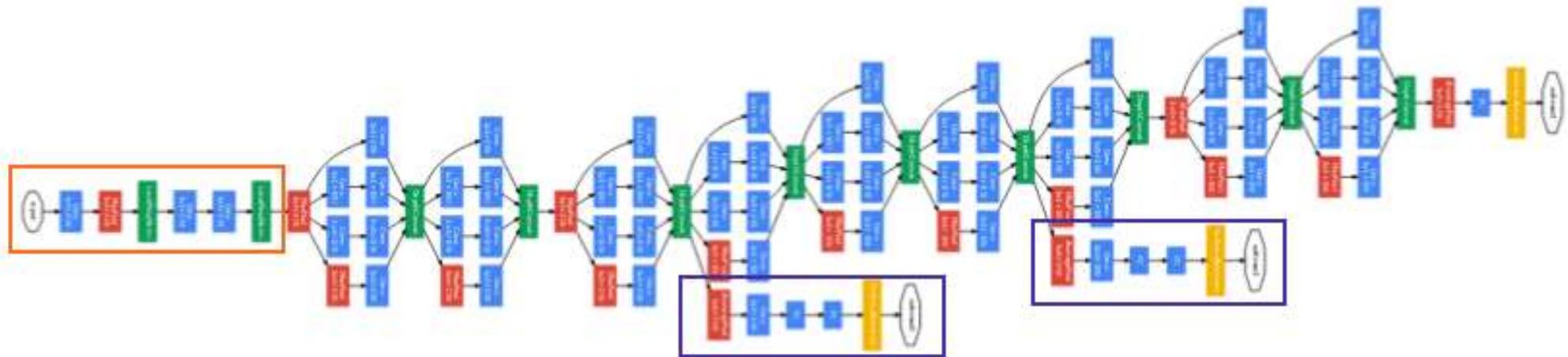
Convolutional Neural Networks | From LeNet to ...

AlexNet [winner of the ImageNet ILSVRS challenge in 2012; (17)] is composed of both stacked and connected layers and includes five convolutional layers followed by three fully-connected layers, with max-pooling layers in between. A rectified linear unit nonlinearity is applied to each convolutional layer along with a fully-connected layer to enable faster training.



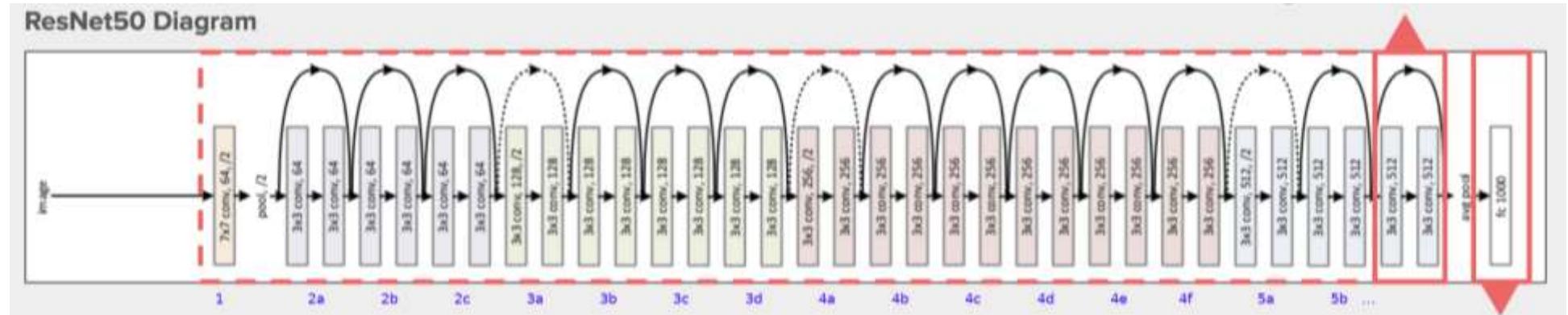
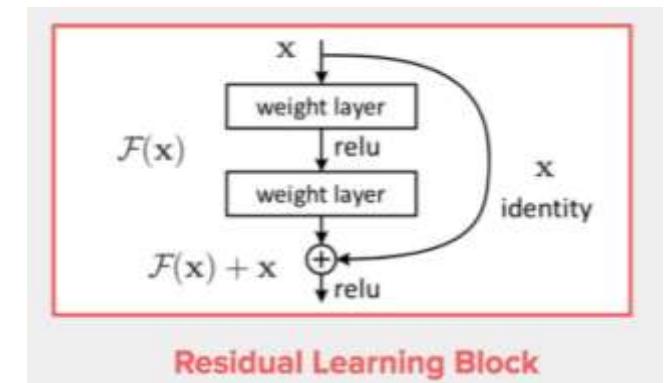
Convolutional Neural Networks | From LeNet to ...

GoogleNet [winner of the ImageNet ILSVRS challenge in 2014; (41)] is the deep-learning algorithm whose design introduced the so-called Inception module, a subnetwork consisting of parallel convolutional filters whose outputs are concatenated. Inception greatly reduces the number of required parameters. GoogleNet is composed by 22 layers that require training (for a total of 27 layers when including the pooling layers).



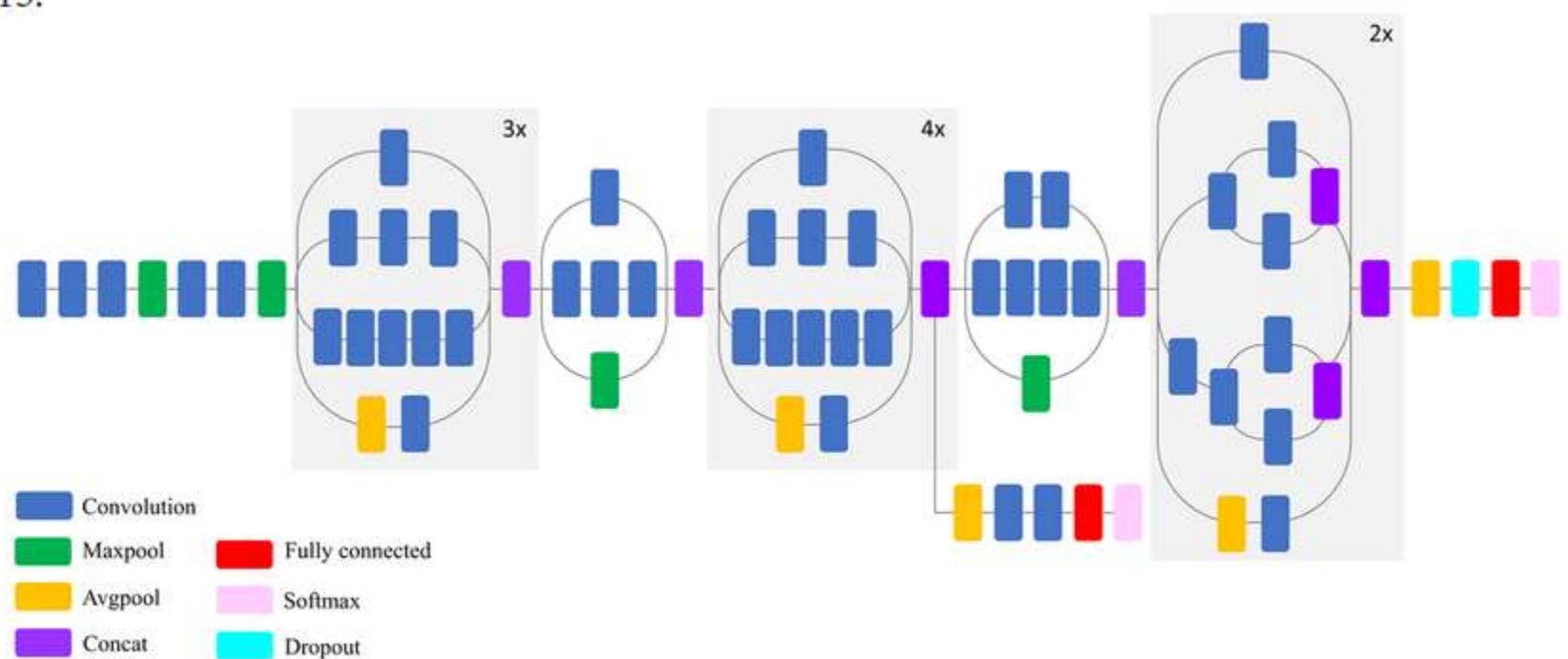
Convolutional Neural Networks | From LeNet to ...

ResNet [winner of ILSVRC 2015; (42)], an architecture that is approximately twenty times deeper than AlexNet; its main novelty is the introduction of residual layers, a kind of network-in-network architecture that forms building blocks to construct the network. ResNet uses special skip connections and batch normalization, and the fully-connected layers at the end of the network are substituted by global average pooling. Instead of learning unreference functions, ResNet explicitly reformulates layers as learning residual functions with reference to the input layer, which makes the model smaller in size and thus easier to optimize than other architectures.

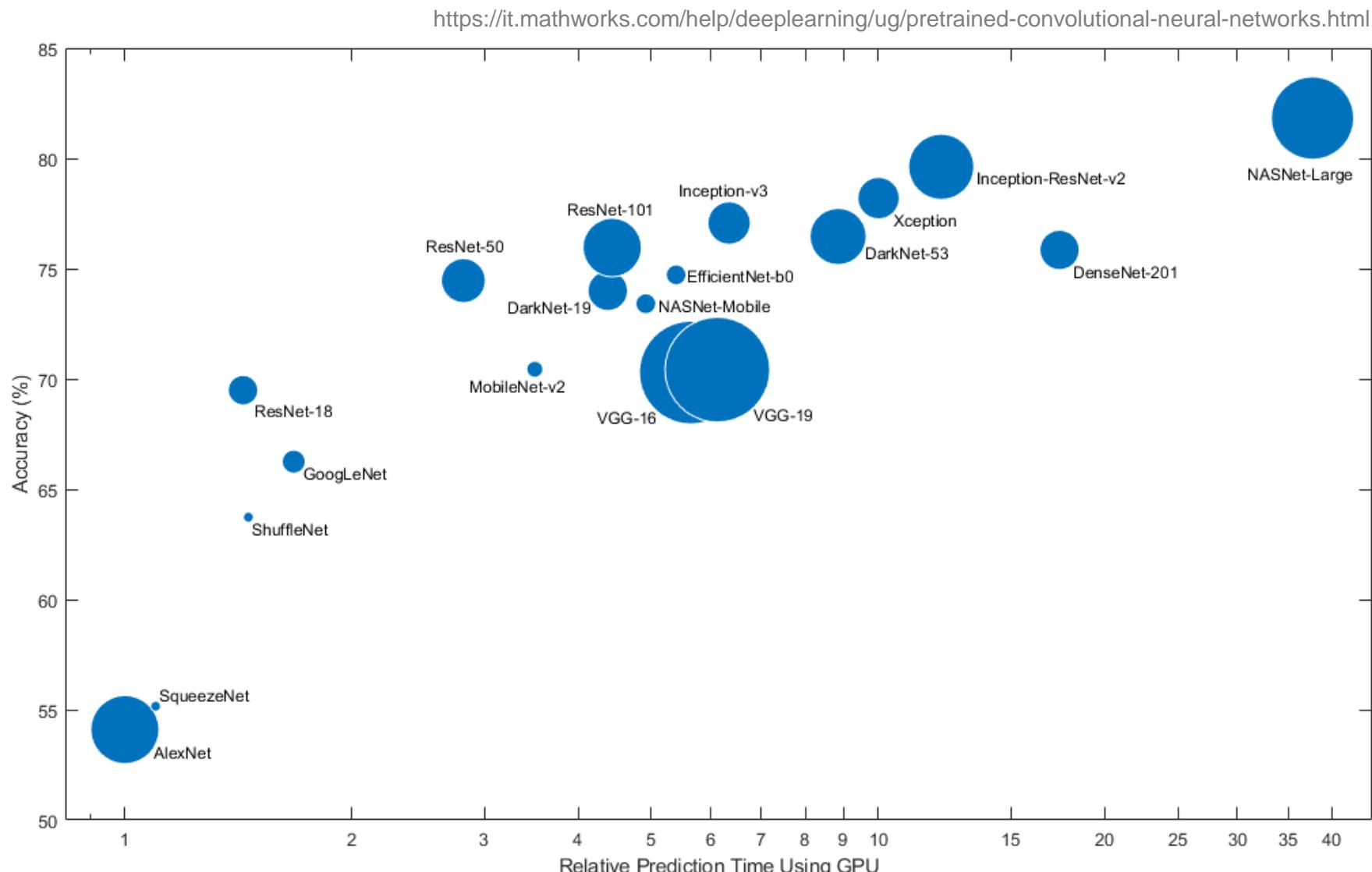


Convolutional Neural Networks | From LeNet to ...

Inception-v3 (43), a deep architecture of 48 layers able to classify images into 1,000 object categories; the net was trained on more than a million images obtained from the ImageNet database (resulting in a rich feature representation for a wide range of images). Inception-v3 classified as the first runner up for the ImageNet ILSVRC challenge in 2015.



Convolutional Neural Networks | From LeNet to ...

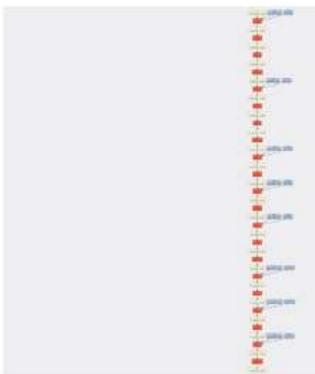


Convolutional Neural Networks | From LeNet to ...

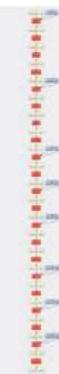
| Network | Depth | Size | Parameters (Millions) | Image Input Size |
|-------------------|-------|--------|-----------------------|------------------|
| squeezeNet | 18 | 5.2 MB | 1.24 | 227-by-227 |
| googlenet | 22 | 27 MB | 7.0 | 224-by-224 |
| inceptionv3 | 48 | 89 MB | 23.9 | 299-by-299 |
| densenet201 | 201 | 77 MB | 20.0 | 224-by-224 |
| mobilenetv2 | 53 | 13 MB | 3.5 | 224-by-224 |
| resnet18 | 18 | 44 MB | 11.7 | 224-by-224 |
| resnet50 | 50 | 96 MB | 25.6 | 224-by-224 |
| resnet101 | 101 | 167 MB | 44.6 | 224-by-224 |
| xception | 71 | 85 MB | 22.9 | 299-by-299 |
| inceptionresnetv2 | 164 | 209 MB | 55.9 | 299-by-299 |
| shufflenet | 50 | 5.4 MB | 1.4 | 224-by-224 |
| nasnetmobile | * | 20 MB | 5.3 | 224-by-224 |
| nasnetlarge | * | 332 MB | 88.9 | 331-by-331 |
| darknet19 | 19 | 78 MB | 20.8 | 256-by-256 |
| darknet53 | 53 | 155 MB | 41.6 | 256-by-256 |
| efficientnetb0 | 82 | 20 MB | 5.3 | 224-by-224 |
| alexnet | 8 | 227 MB | 61.0 | 227-by-227 |
| vgg16 | 16 | 515 MB | 138 | 224-by-224 |
| vgg19 | 19 | 535 MB | 144 | 224-by-224 |

Convolutional Neural Networks | From LeNet to ...

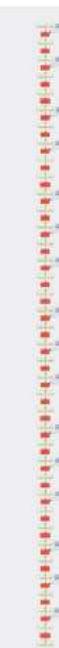
AlexNet (2012)



VGG-M (2013)



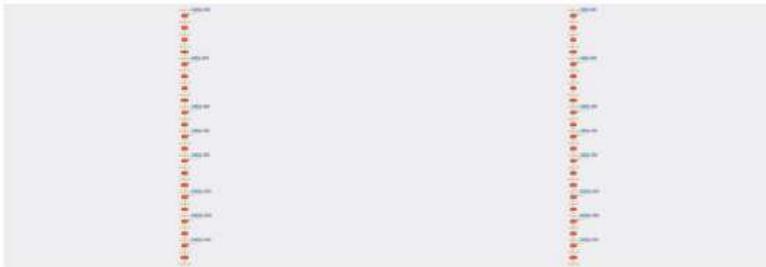
VGG-VD-16 (2014)



FROM A. VEDALDI

Convolutional Neural Networks | From LeNet to ...

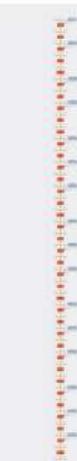
AlexNet (2012)



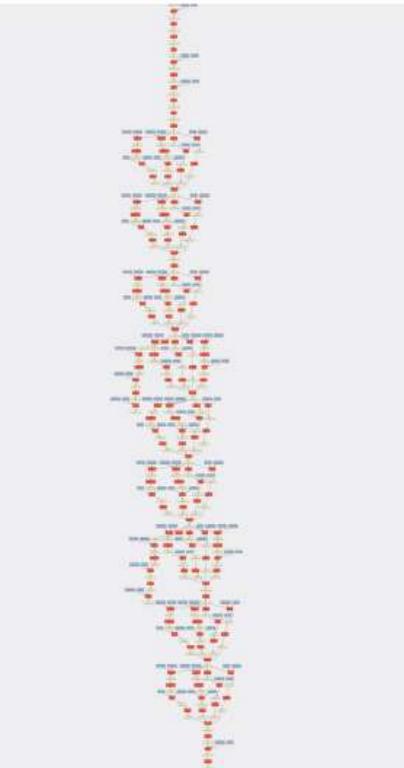
VGG-M (2013)



VGG-VD-16 (2014)

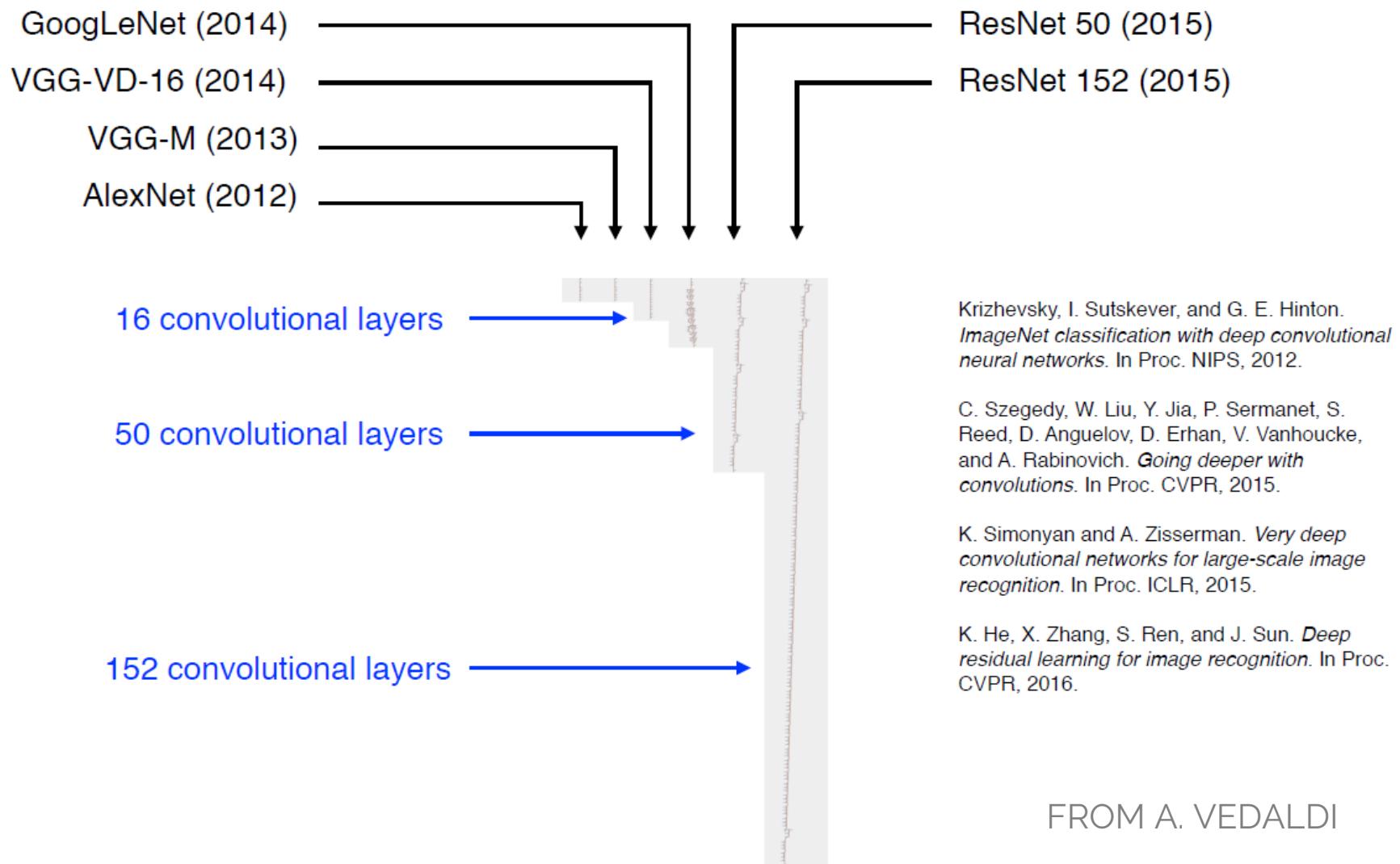


GoogLeNet (2014)



FROM A. VEDALDI

Convolutional Neural Networks | From LeNet to ...



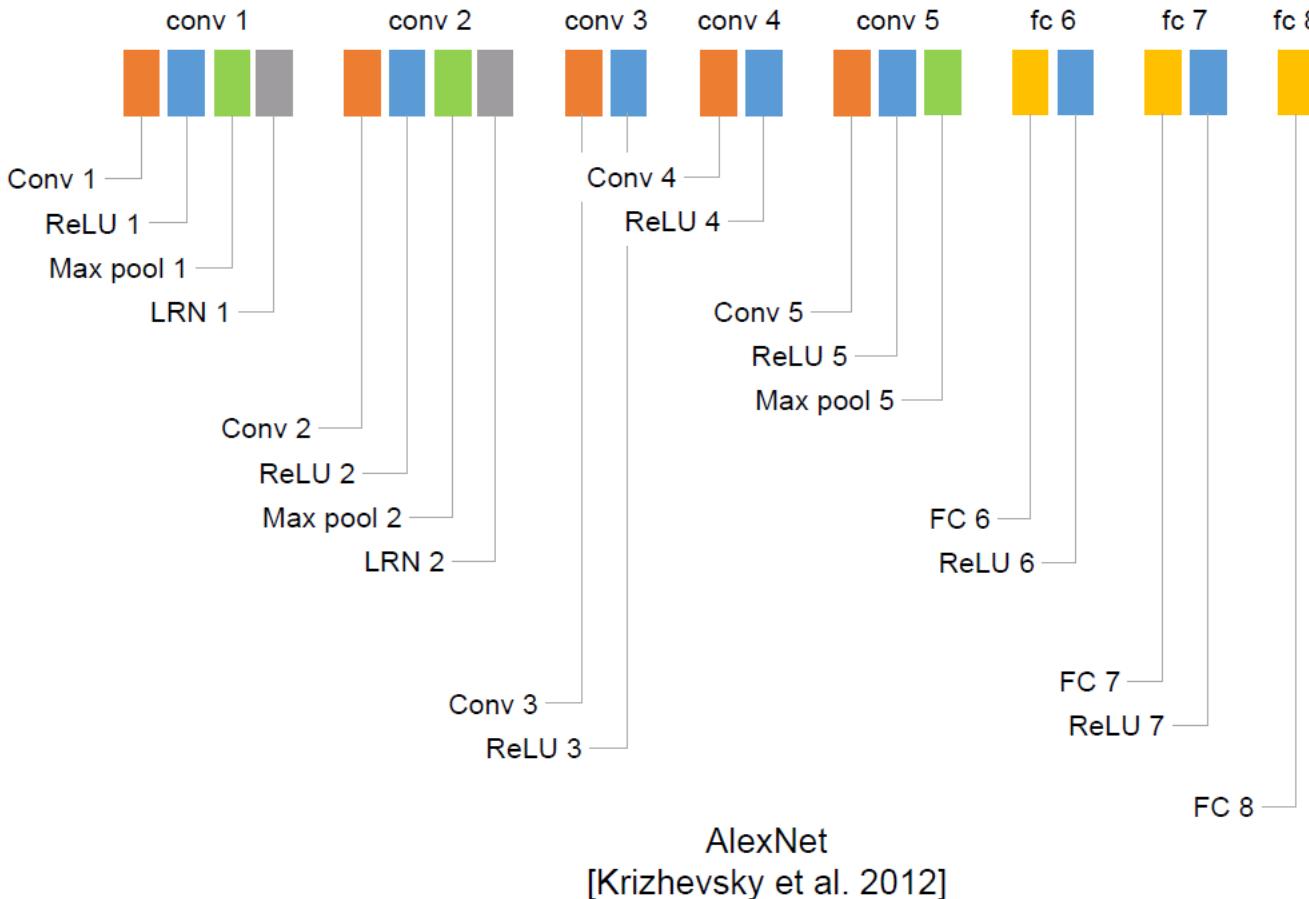
A CONSIDERATION ON EXPLAINABILITY AND (OVER)FITTING

UNDERFITTING, OVERFITTING AND BEST FITTING

FROM A. VEDALDI

10

Inversion



UNDERFITTING, OVERFITTING AND BEST FITTING

11

Inversion



Original
Image



UNDERFITTING, OVERFITTING AND BEST FITTING

12

Inversion



Original
Image



UNDERFITTING, OVERFITTING AND BEST FITTING

13

Inversion

FROM A. VEDALDI



Original
Image



UNDERFITTING, OVERFITTING AND BEST FITTING

14

Inversion

FROM A. VEDALDI



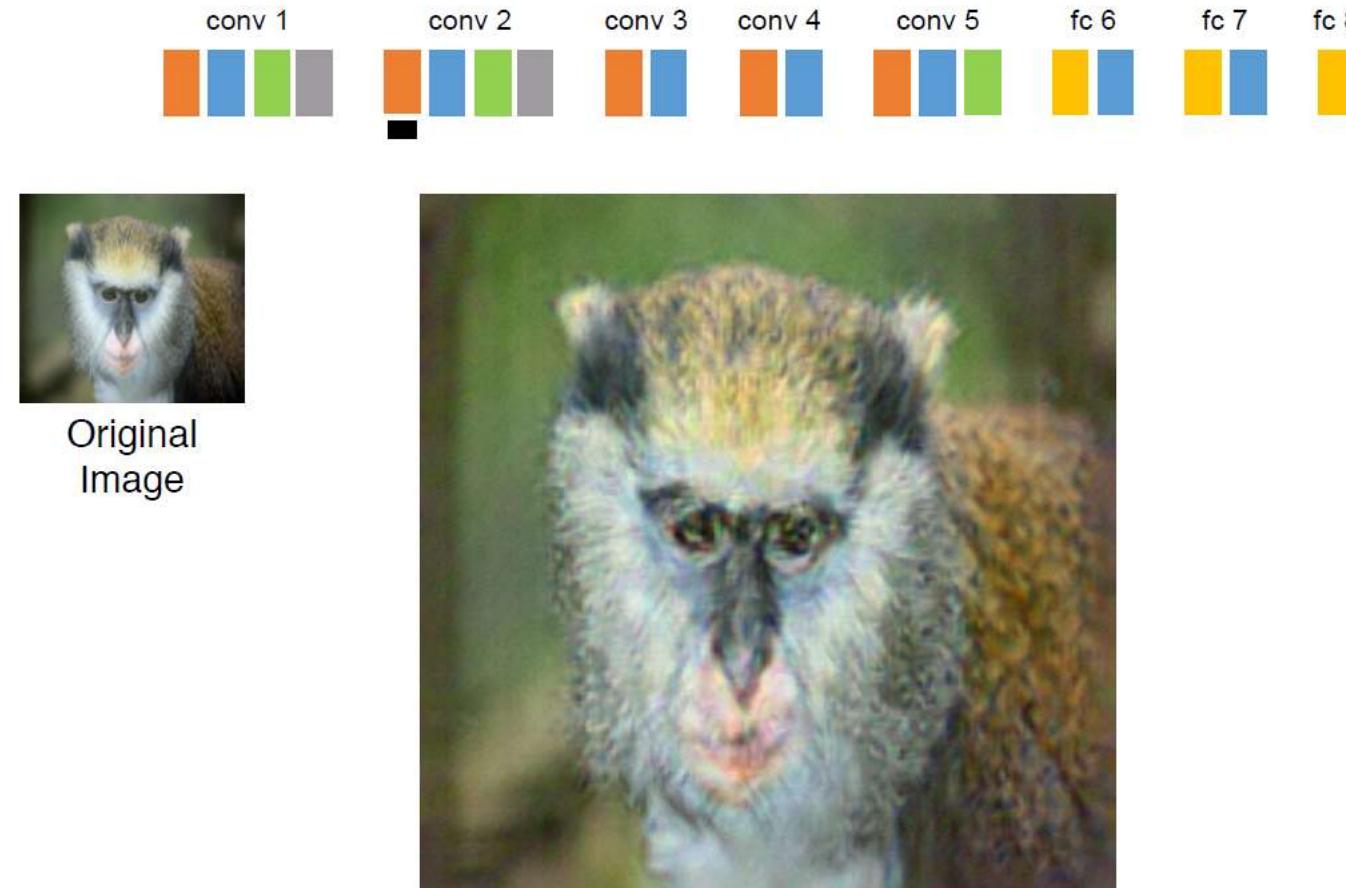
Original
Image



UNDERFITTING, OVERFITTING AND BEST FITTING

Inversion

15



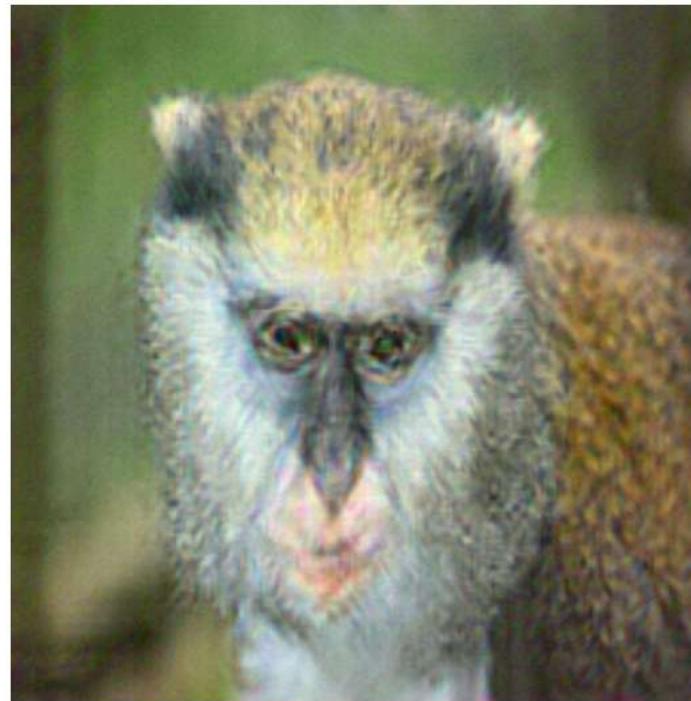
UNDERFITTING, OVERFITTING AND BEST FITTING

16

Inversion



Original
Image



UNDERFITTING, OVERFITTING AND BEST FITTING

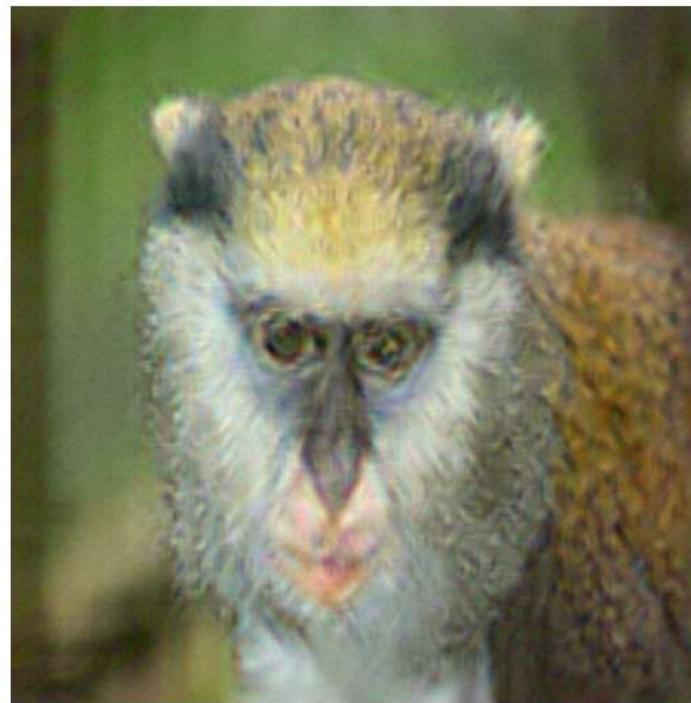
17

Inversion

FROM A. VEDALDI



Original
Image



UNDERFITTING, OVERFITTING AND BEST FITTING

18

Inversion



Original
Image



UNDERFITTING, OVERFITTING AND BEST FITTING

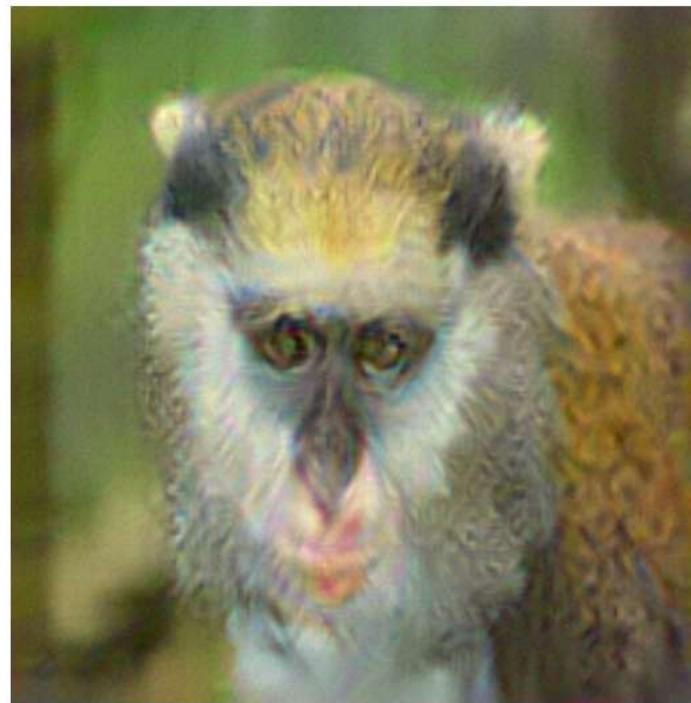
19

Inversion

FROM A. VEDALDI



Original
Image



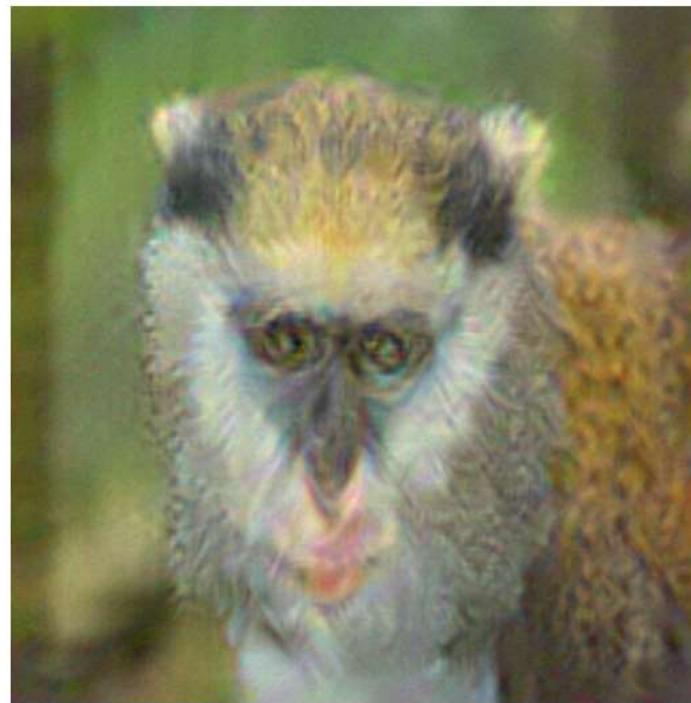
UNDERFITTING, OVERFITTING AND BEST FITTING

20

Inversion



Original
Image

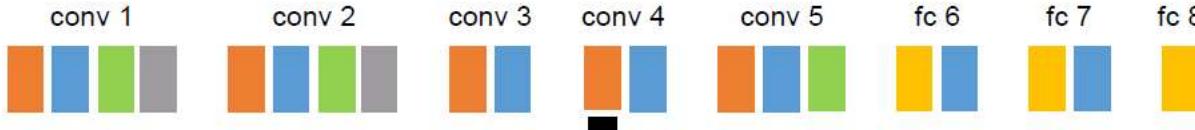


UNDERFITTING, OVERFITTING AND BEST FITTING

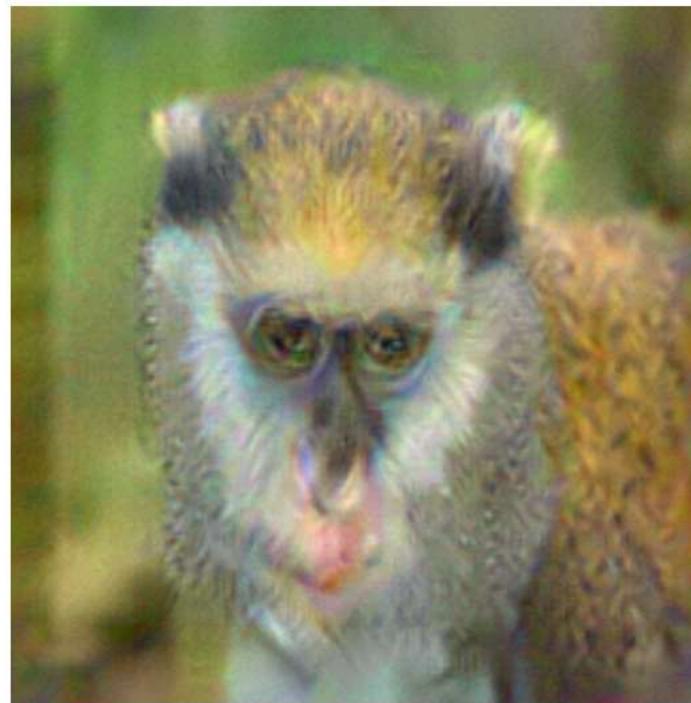
21

Inversion

FROM A. VEDALDI



Original
Image



UNDERFITTING, OVERFITTING AND BEST FITTING

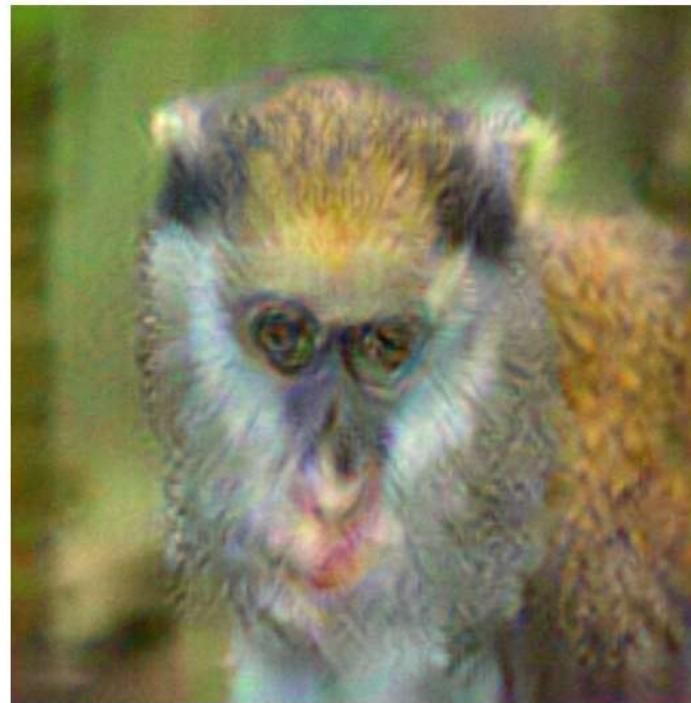
22

FROM A. VEDALDI

Inversion



Original
Image



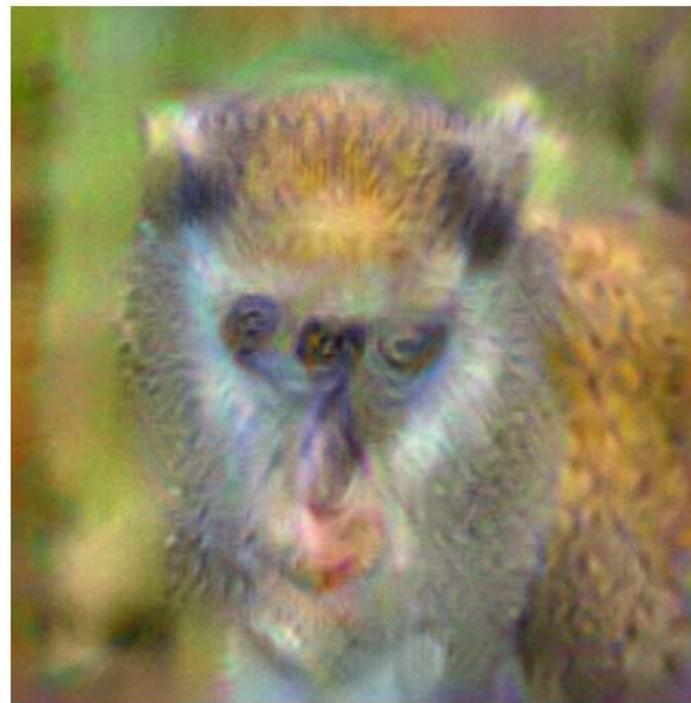
UNDERFITTING, OVERFITTING AND BEST FITTING

23

Inversion



Original
Image



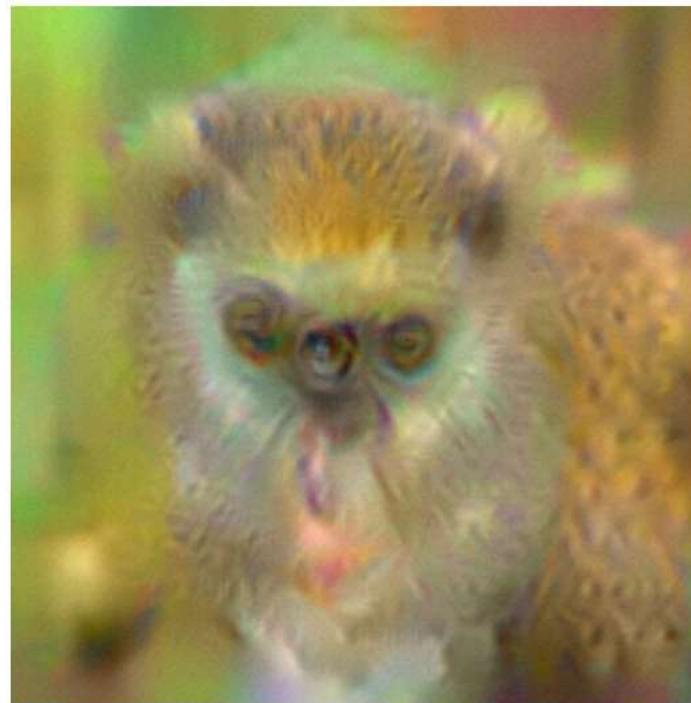
UNDERFITTING, OVERFITTING AND BEST FITTING

24

Inversion



Original
Image



UNDERFITTING, OVERFITTING AND BEST FITTING

25

Inversion



Original
Image



UNDERFITTING, OVERFITTING AND BEST FITTING

26

Inversion



Original
Image



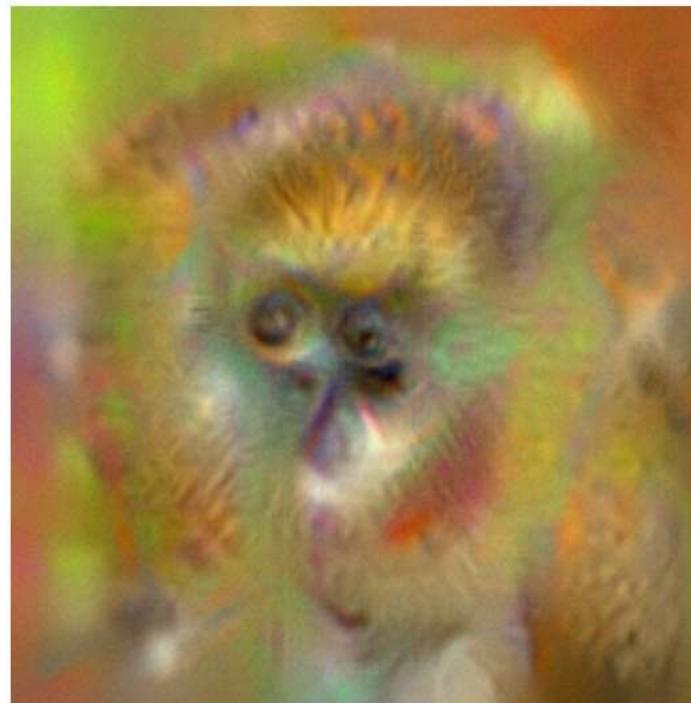
UNDERFITTING, OVERFITTING AND BEST FITTING

27

Inversion



Original
Image



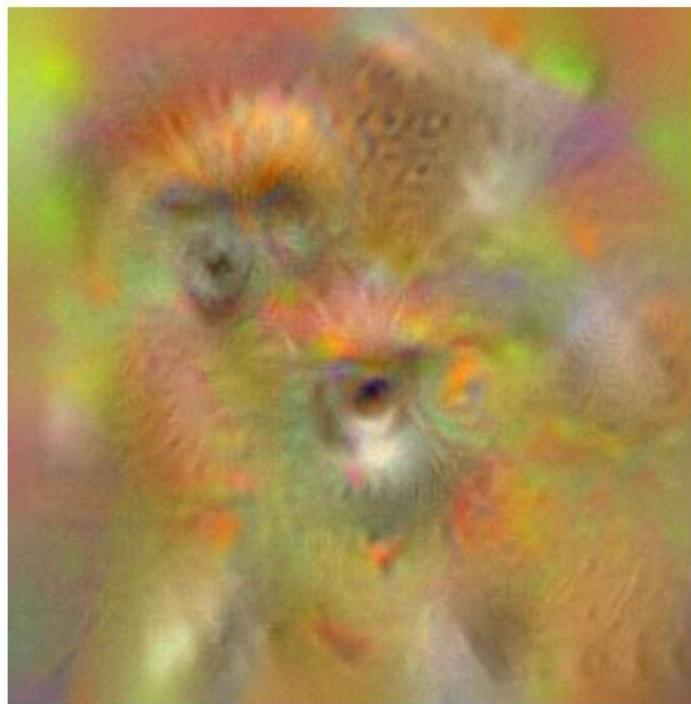
UNDERFITTING, OVERFITTING AND BEST FITTING

28

Inversion



Original
Image



UNDERFITTING, OVERFITTING AND BEST FITTING

29

Inversion



Original
Image



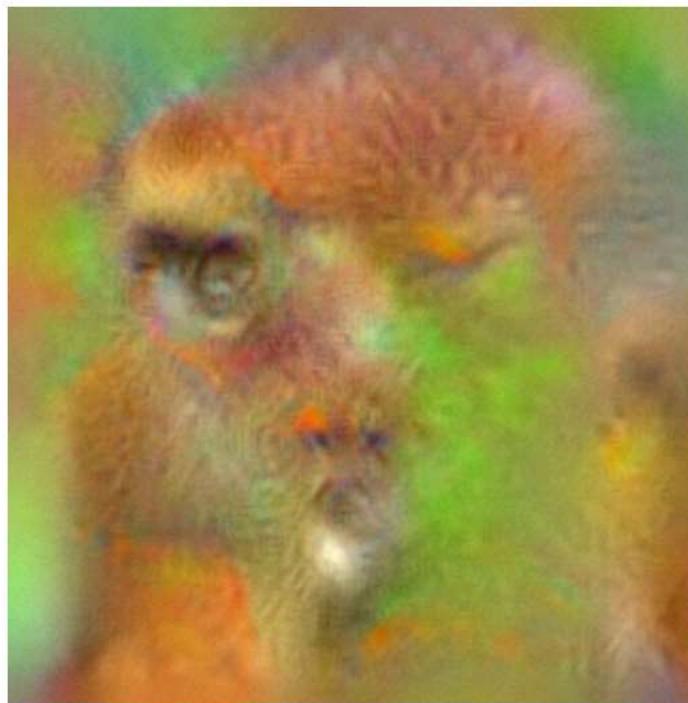
UNDERFITTING, OVERFITTING AND BEST FITTING

30

Inversion



Original
Image

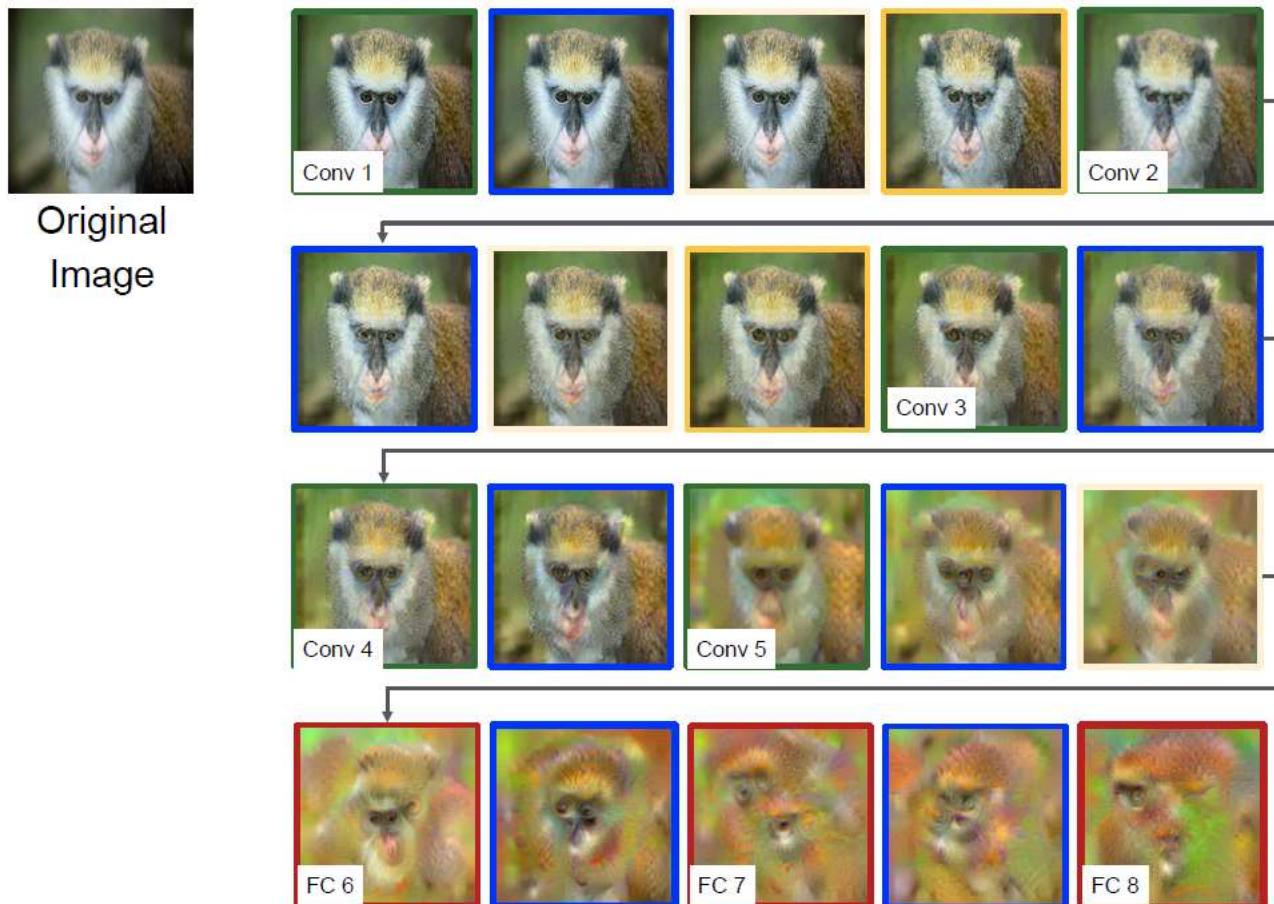


UNDERFITTING, OVERFITTING AND BEST FITTING

31

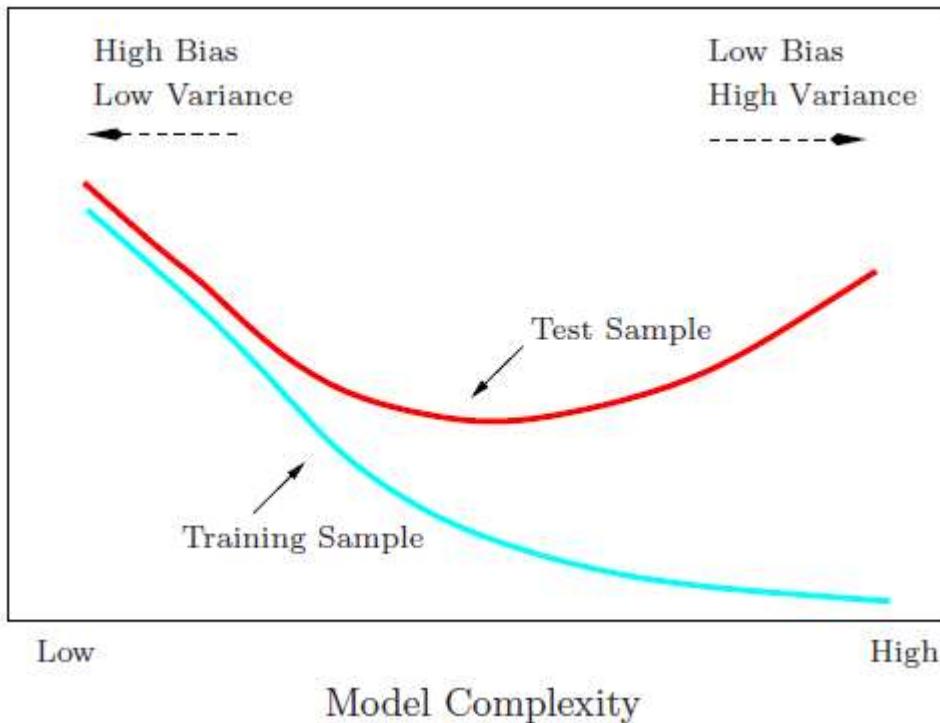
Inverting a Deep CNN

FROM A. VEDALDI



UNDERFITTING, OVERFITTING AND BEST FITTING

Prediction Error

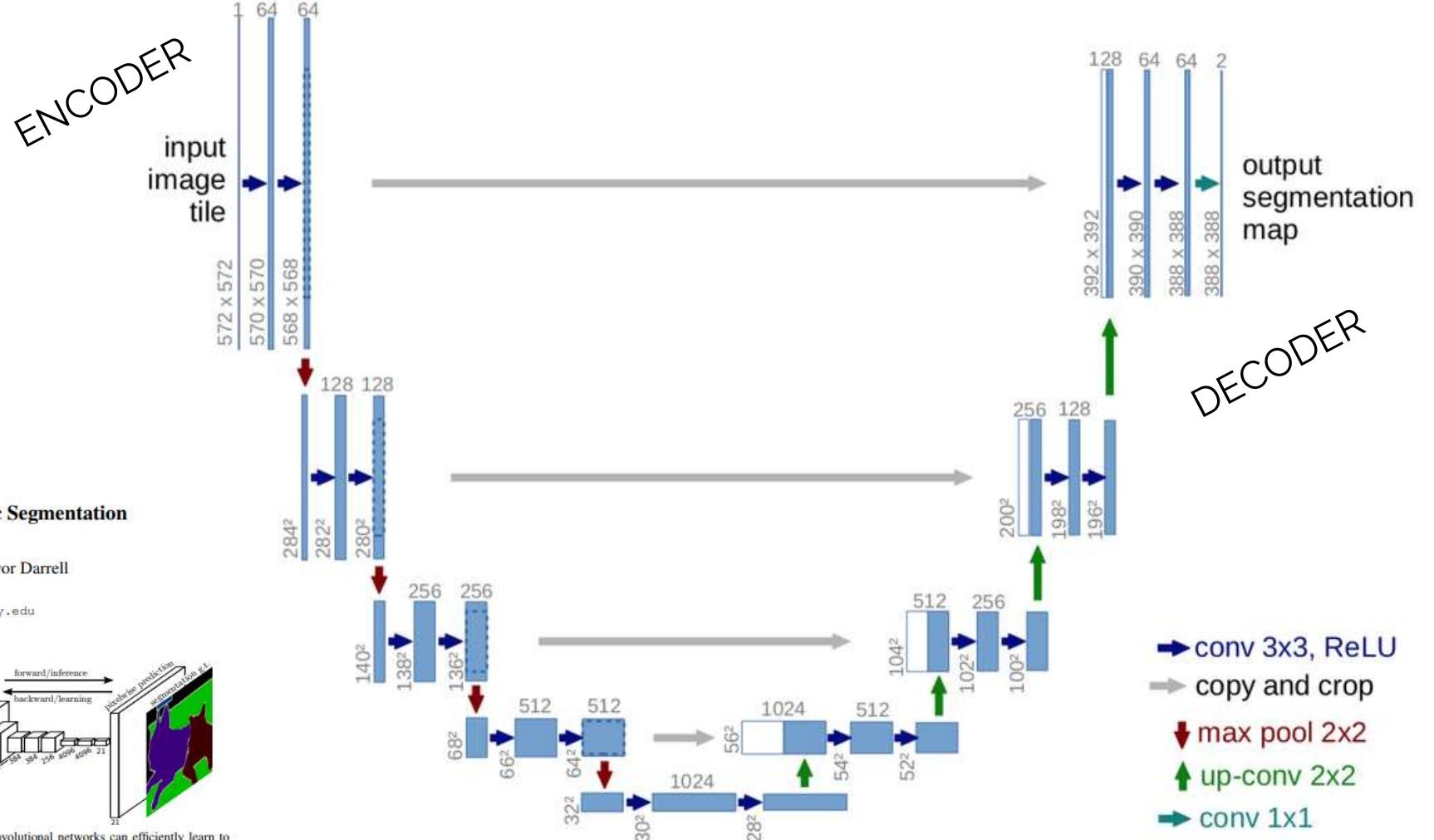


| Symptoms | Underfitting | Just right | Overfitting |
|-----------------------------|---|------------|--|
| Regression illustration | | | |
| Classification illustration | | | |
| Deep learning illustration | | | |
| Possible remedies | <ul style="list-style-type: none">• Complexify model• Add more features• Train longer | | <ul style="list-style-type: none">• Perform regularization• Get more data |

Convolutional Neural Networks | U-Net

U-Net

Suited for
(medical) image
segmentation



Fully Convolutional Networks for Semantic Segmentation

Jonathan Long* Evan Shelhamer* Trevor Darrell
UC Berkeley

{jonlong, shelhamer, trevor}@cs.berkeley.edu

Abstract

Convolutional networks are powerful visual models that yield hierarchies of features. We show that convolutional networks by themselves, trained end-to-end, pixels-to-pixels, exceed the state-of-the-art in semantic segmentation. Our key insight is to build “fully convolutional” networks that take input of arbitrary size and produce correspondingly-sized output with efficient inference and learning. We define and detail the space of fully convolutional networks, explain their application to spatially dense prediction tasks, and draw connections to prior models. We adapt contemporary classification networks (AlexNet [19],

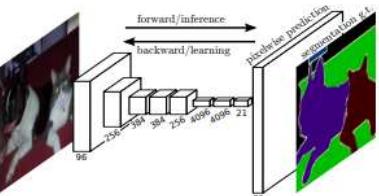


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

`christian.salvatore@iusspavia.it`

`https://christiansalvatore.github.io/machinelearning-iuss/`