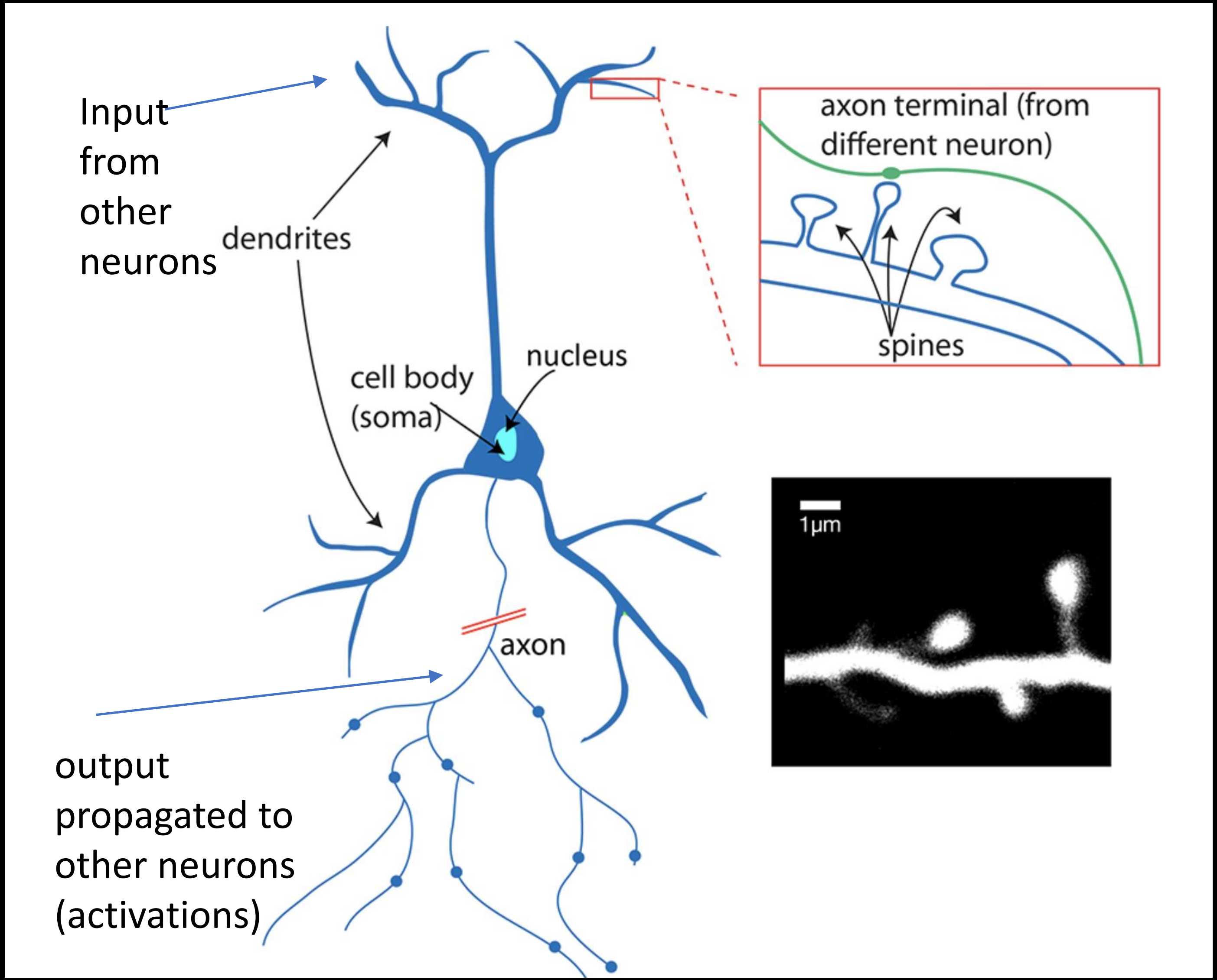
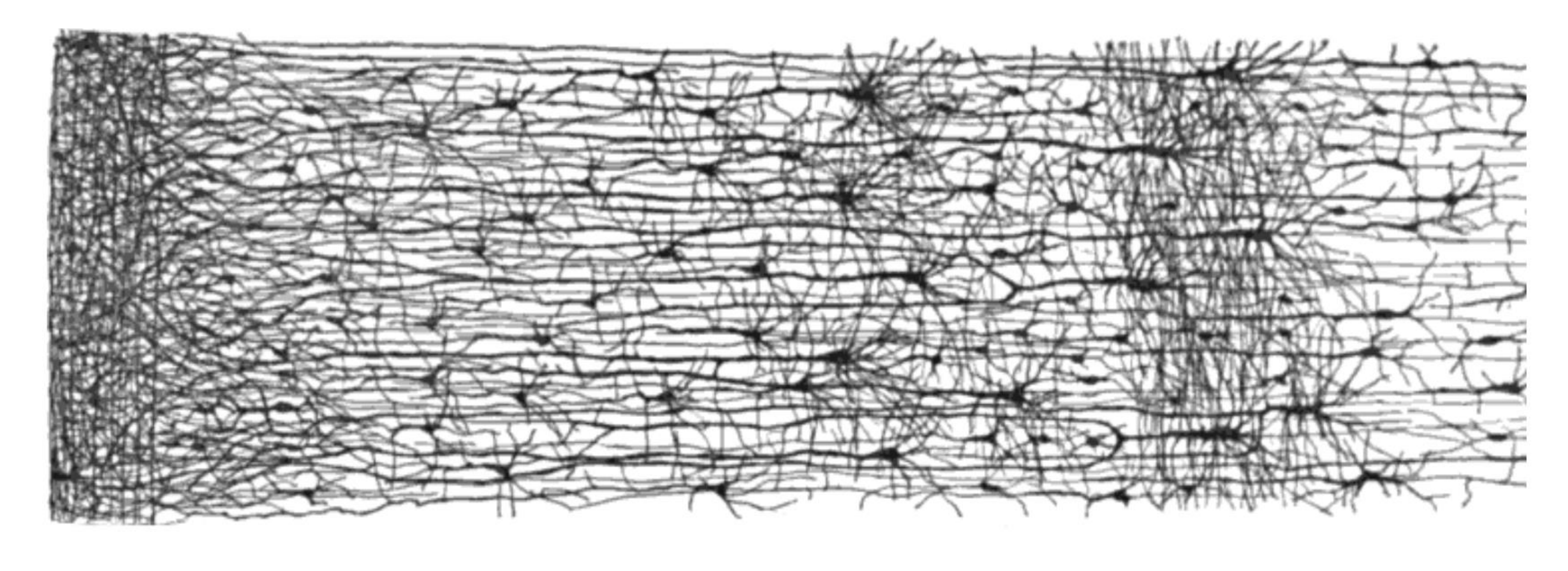
Neural Network

Why ’Neural Network’?

A Biological Neuron



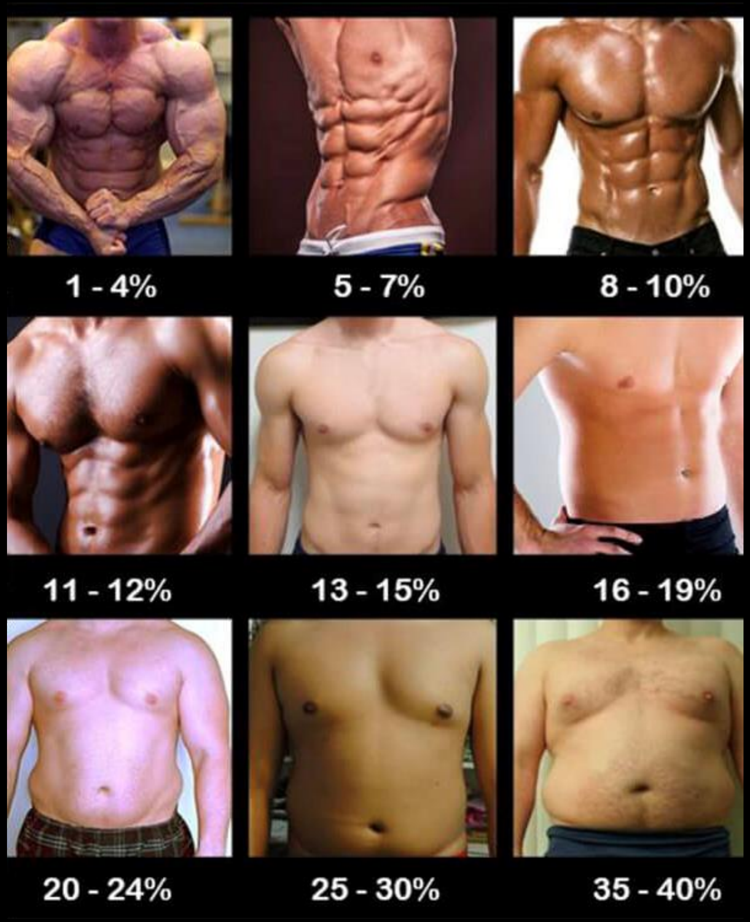
A biological Neuron Network



100 B neurons

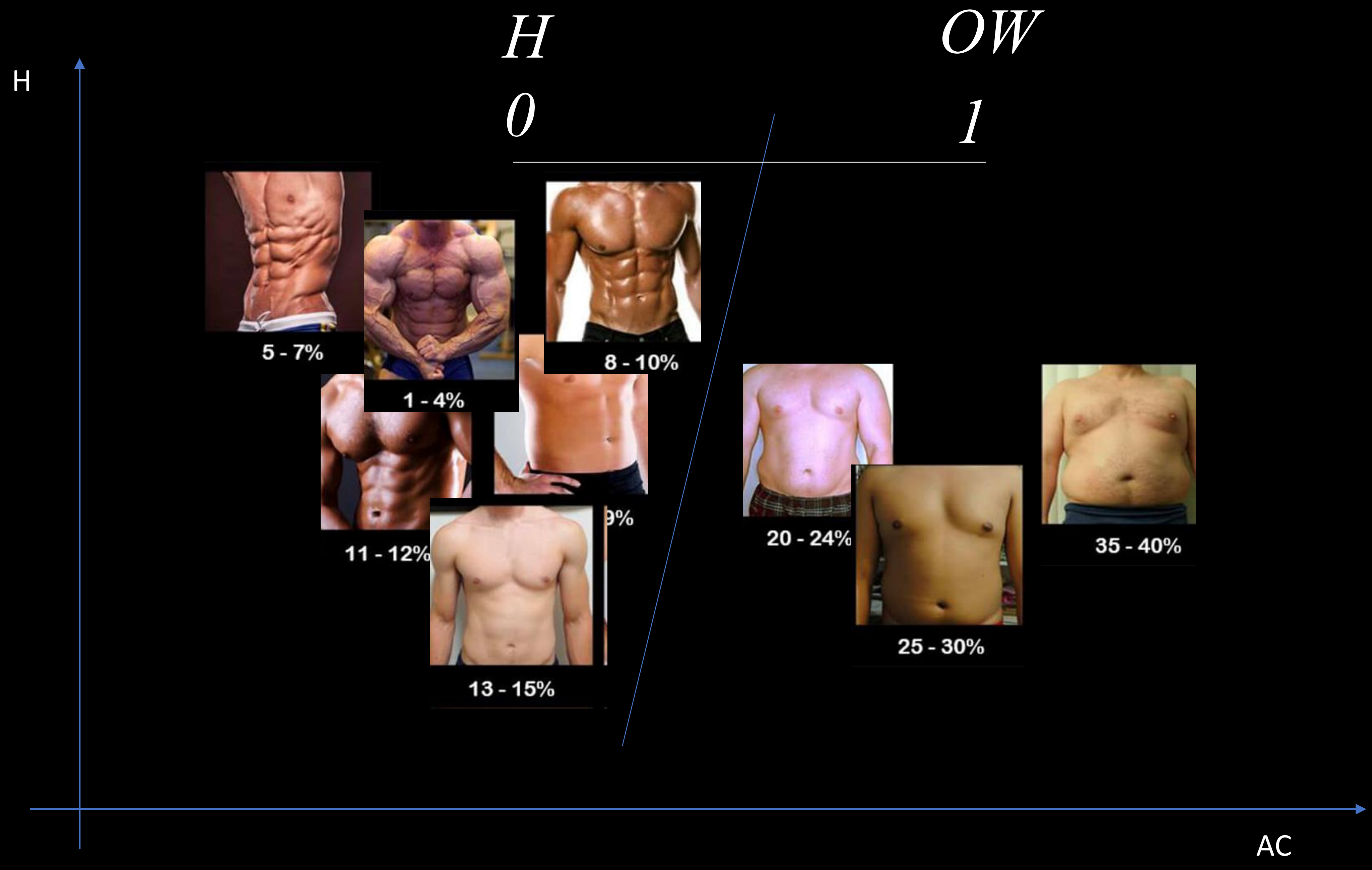
A human vision task:

‘healthy’ vs overweighting men

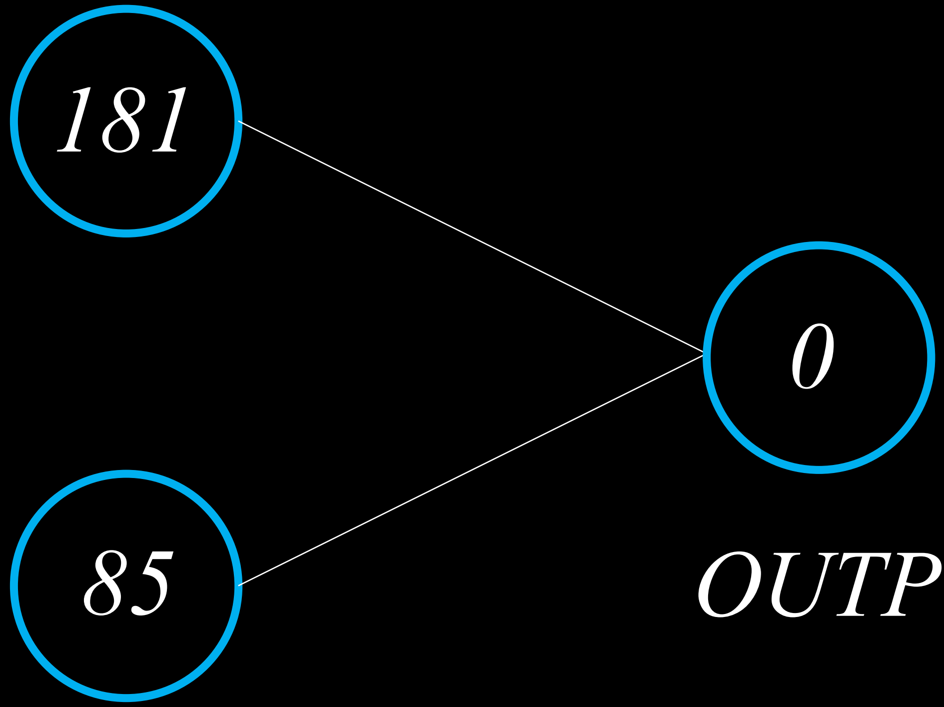


‘healthy’ vs overweighting men

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


H
0
OW
1
H
5 - 7%
1 - 4%
8 - 10%
11 - 12%
9%
13 - 15%
20 - 24%
25 - 30%
35 - 40%
AC

$H=a_1$ $AC=a_2$



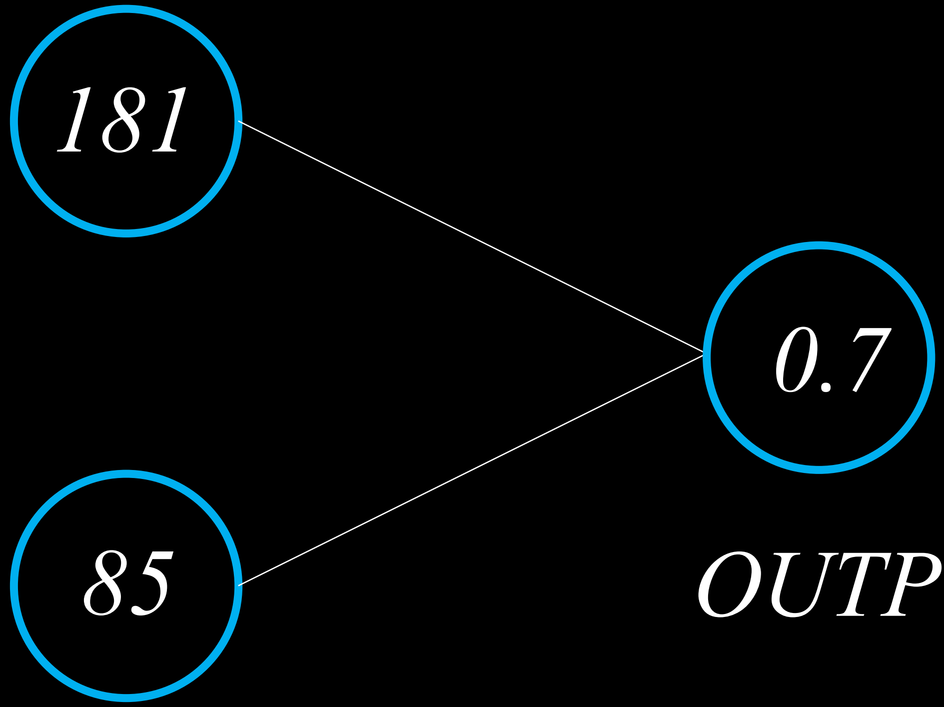
OUTPUT

(Expected)

INPUT

(signals)

$H = a_1$ $AC = a_2$



INPUT

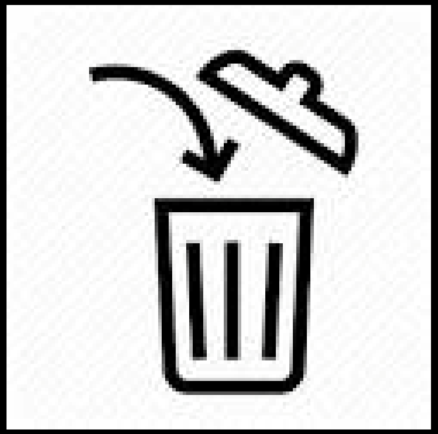
OUTPUT

Predicted

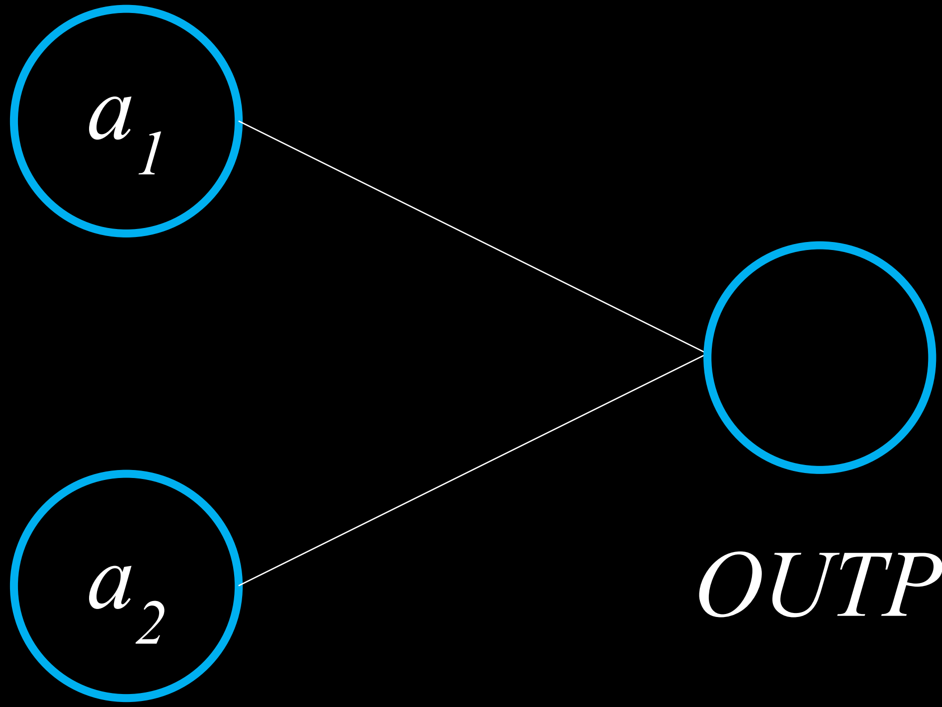
the activation of neuron

H OW

0 1



$H = a_1$ $AC = a_2$



OUTPUT

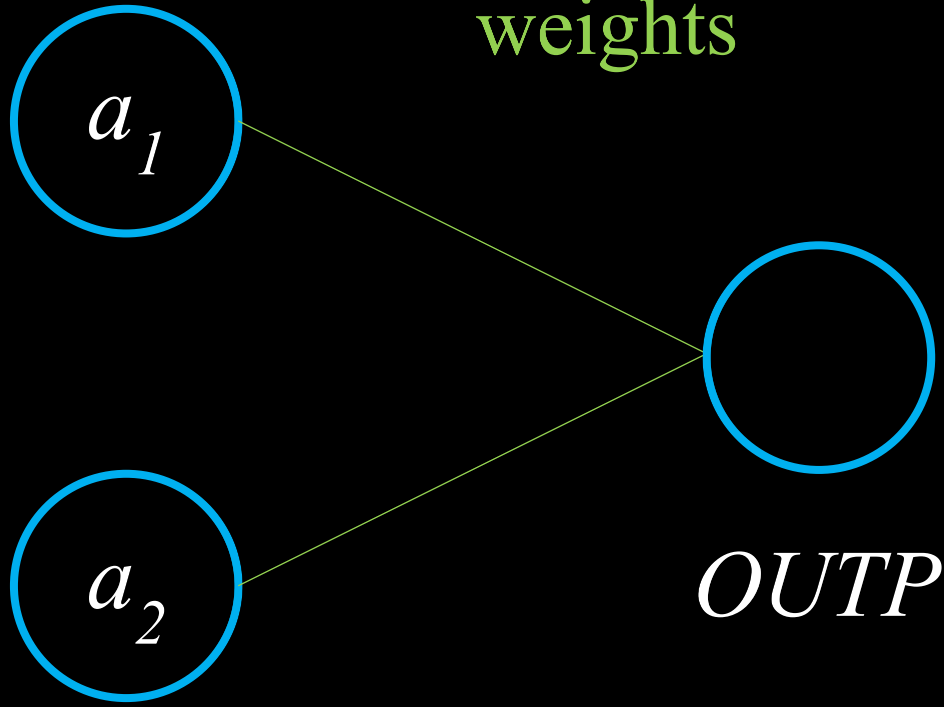
$$OUTPUT = f\,(INPUT)$$

INPUT

The Weights and the Bias

$H = a_1$ $AC = a_2$

weights



Bias (for inactivity)

$$a_1 w_1 + a_2 w_2 + b$$

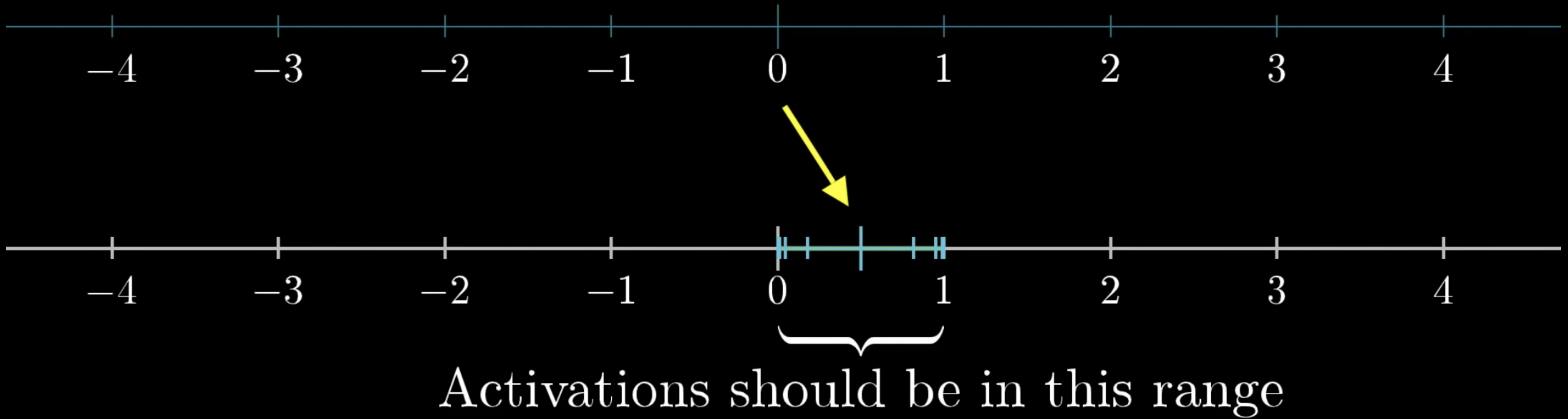
OUTPUT

$$OUTPUT = f\,(INPUT)$$

INPUT

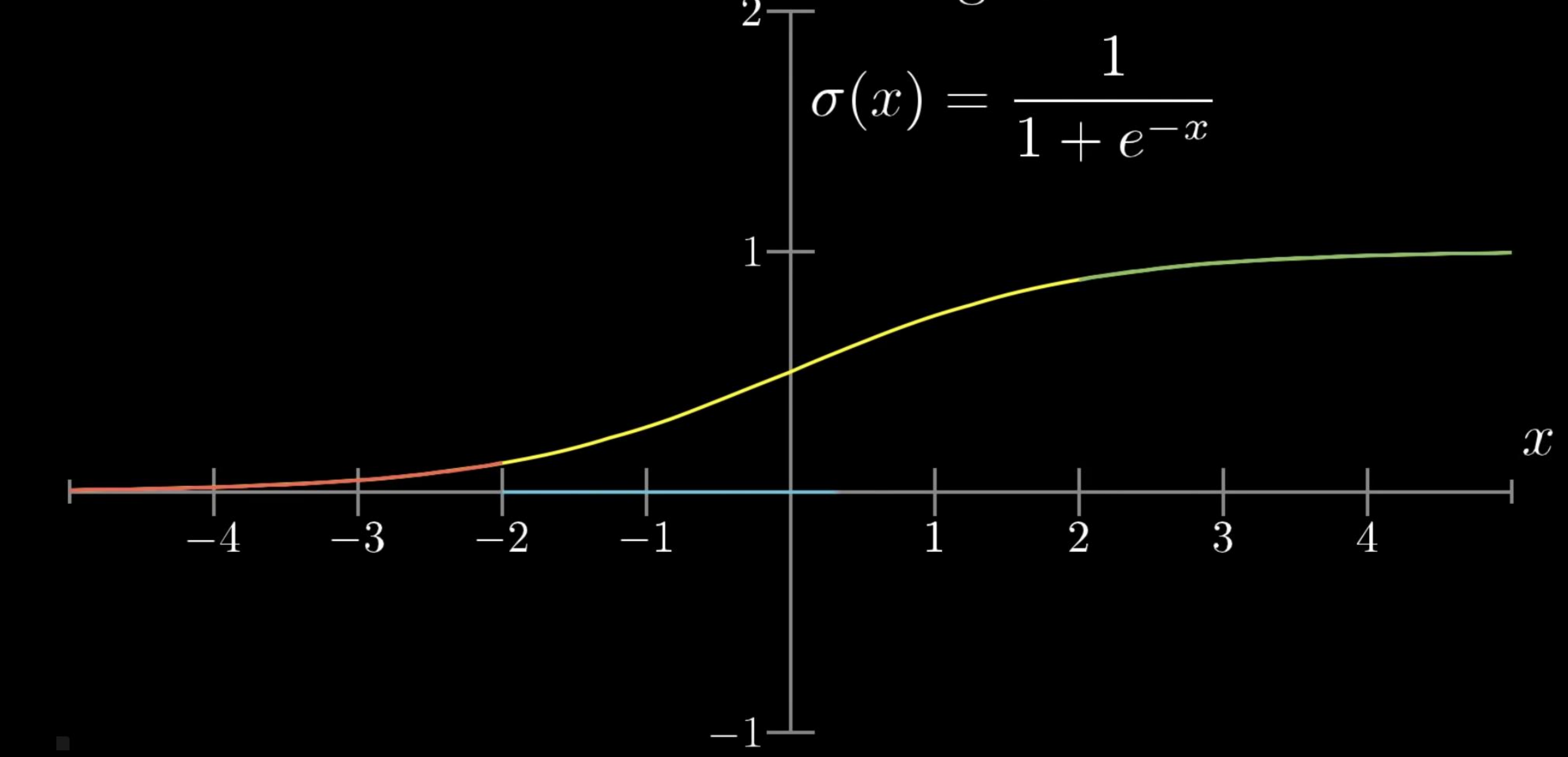
The activation function

$w_1 a_1 + w_2 a_2$



Sigmoid

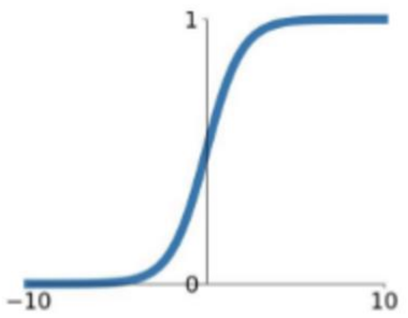
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Other function that progressively changes from 0 to 1 with no discontinuity

Sigmoid

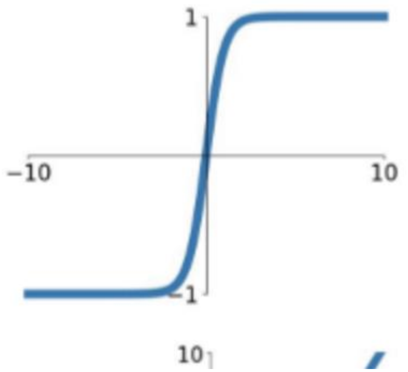
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Hyperbolic tangent funtion

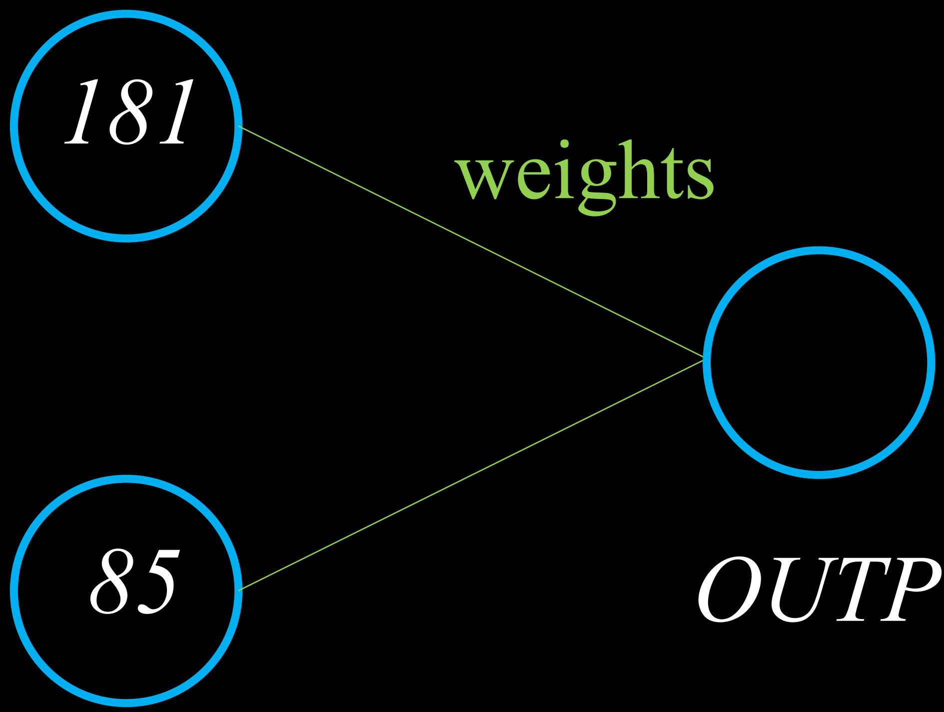
tanh

$$\tanh(x)$$



$H = a_1$ $AC = a_2$

Forward propagation



INPUT

OUTPUT

Activation function

$$Sigmoid\,(181w_1 + 85w_2 + b)$$

$$OUTPUT = f\,(INPUT)$$

w_1 ? w_2 ? b ?

The Cost Function

INPUT

a_1 a_2

The Cost Function

$$Sigmoid\,(\,a_1 w_1 + a_2 w_2 + \mathrm{b}\,)$$

OUTPUT

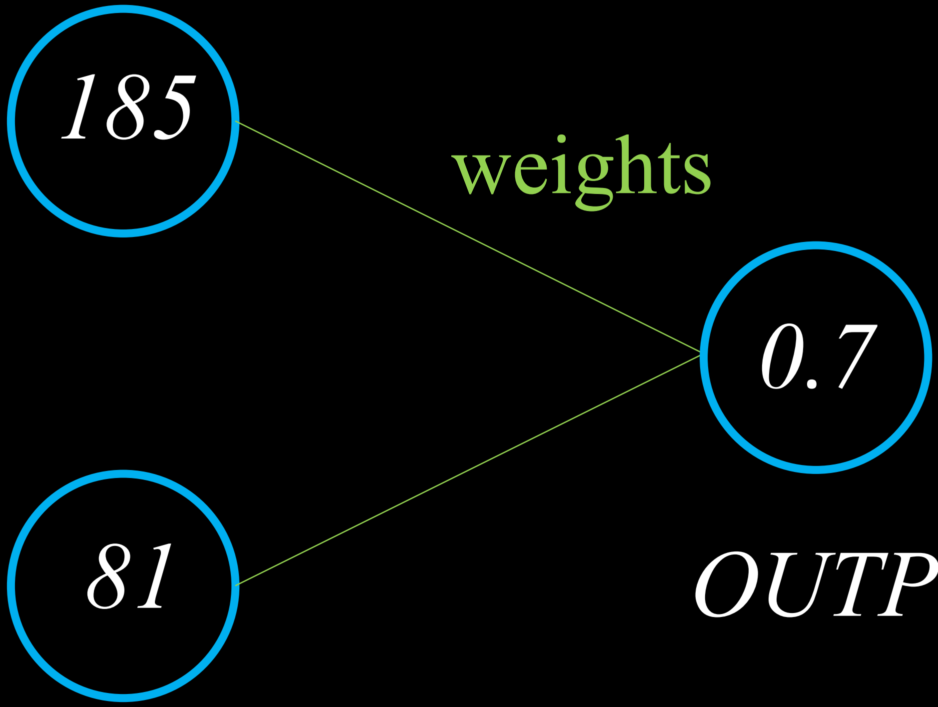
INPUT

a_1 a_2

The Cost Function

w_1 w_2 b

$H = a_1$ $AC = a_2$



0.0

OUTPUT OUTPUT

Predicted *Expected*

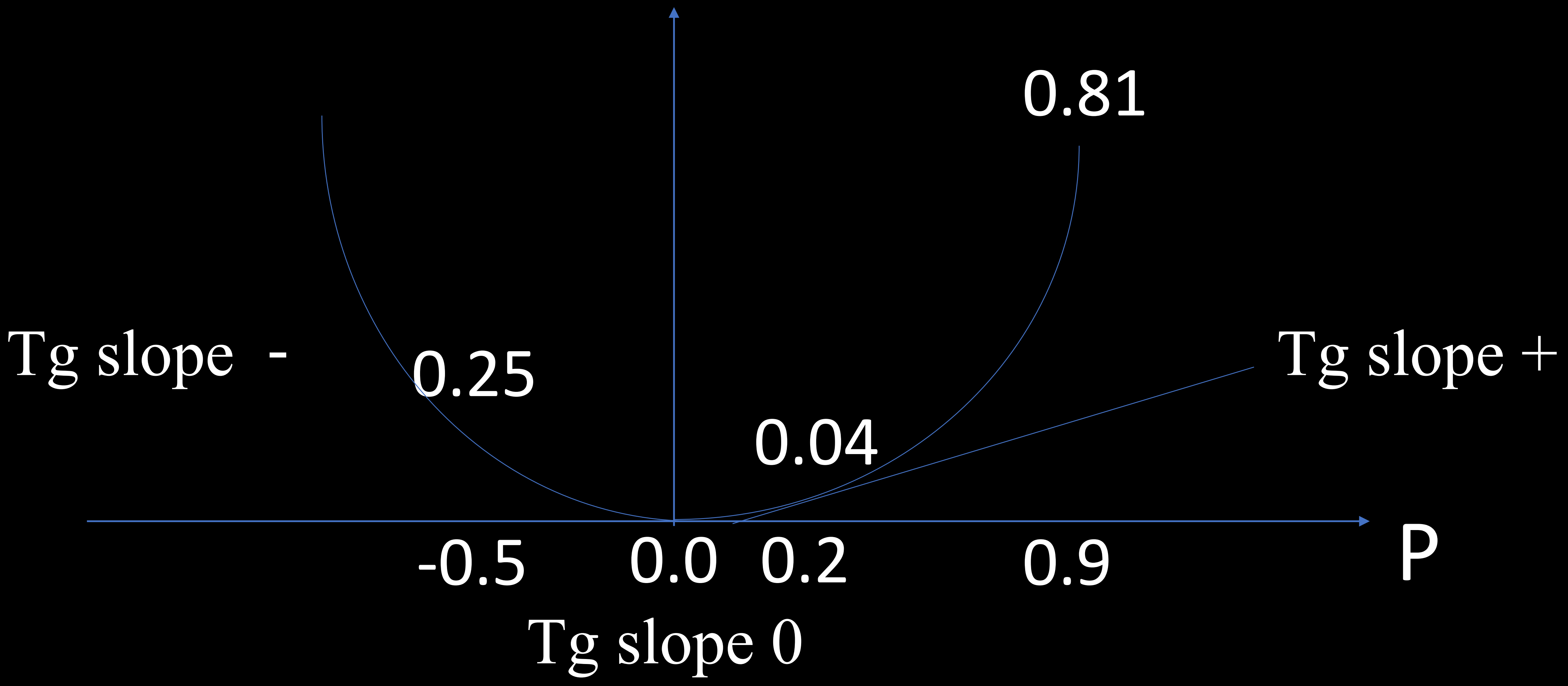
INPUT

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

$$\text{Cost Function} = (\text{Predicted-Expected})^2 = 0.49$$

Squared error cost

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



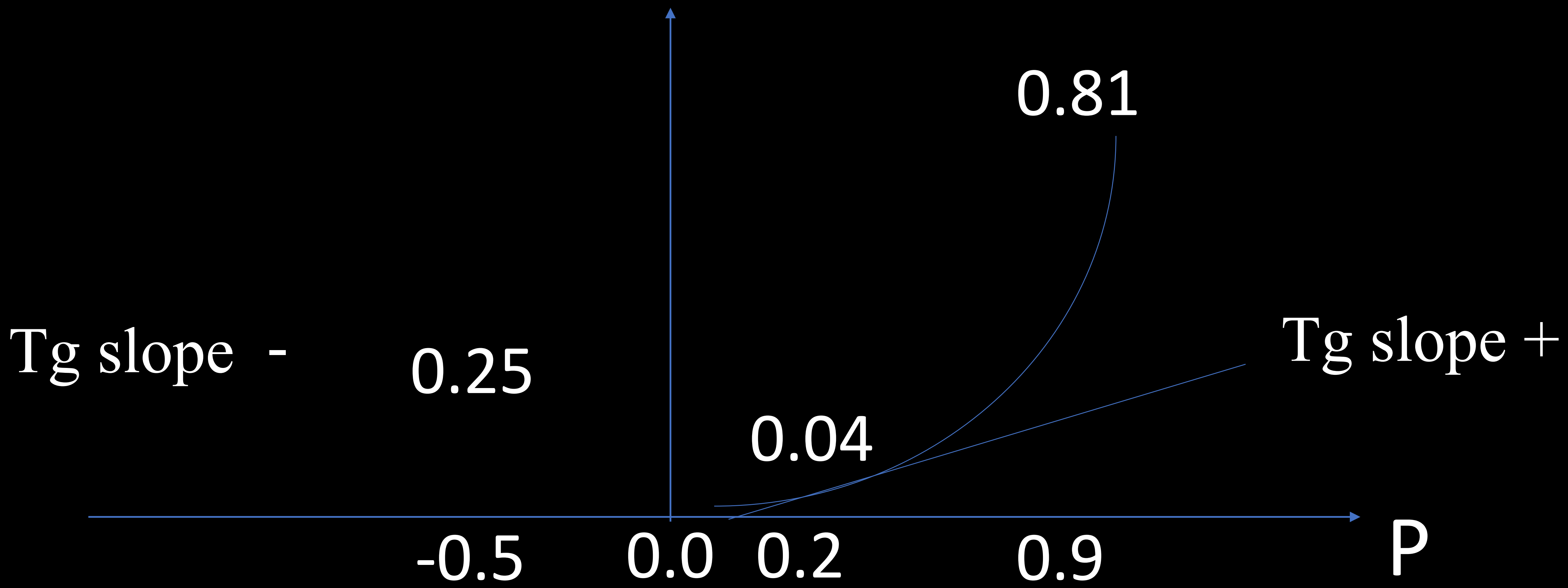
If the slope is + we must decrease P of a fraction of the slope

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

The Cost Function = $(P\text{-}0.0)^2$



Next P = P – LR*(Tg slope)P   LR*=Learning Rate

Next P = P – LR* (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

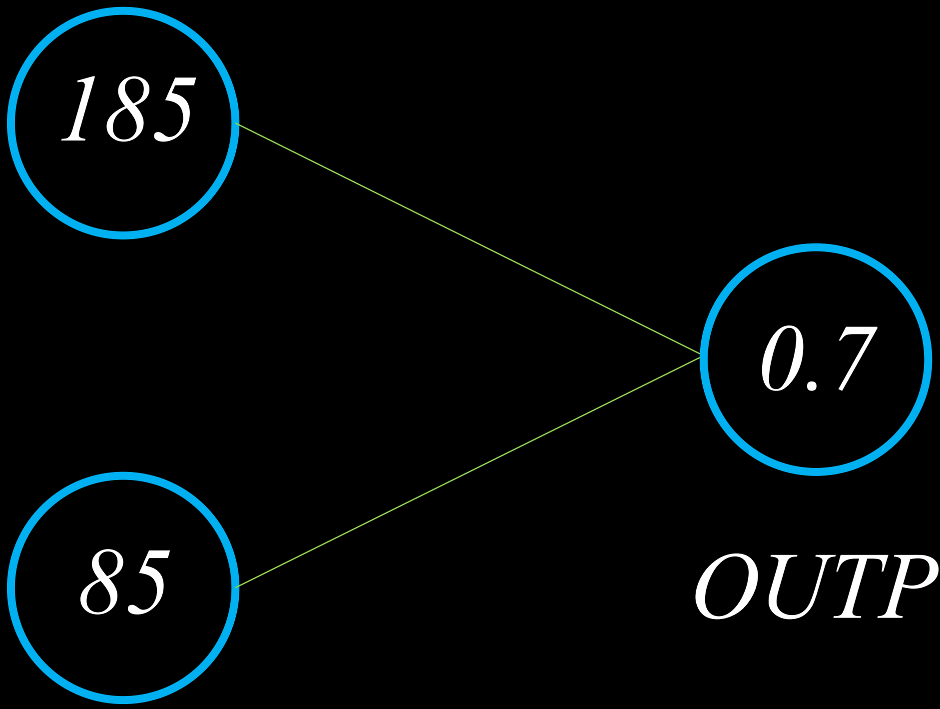
Different training methods

Supervised

learning rule that trains the neural network on

already known correct output

$H = a_1$ $CA = a_2$



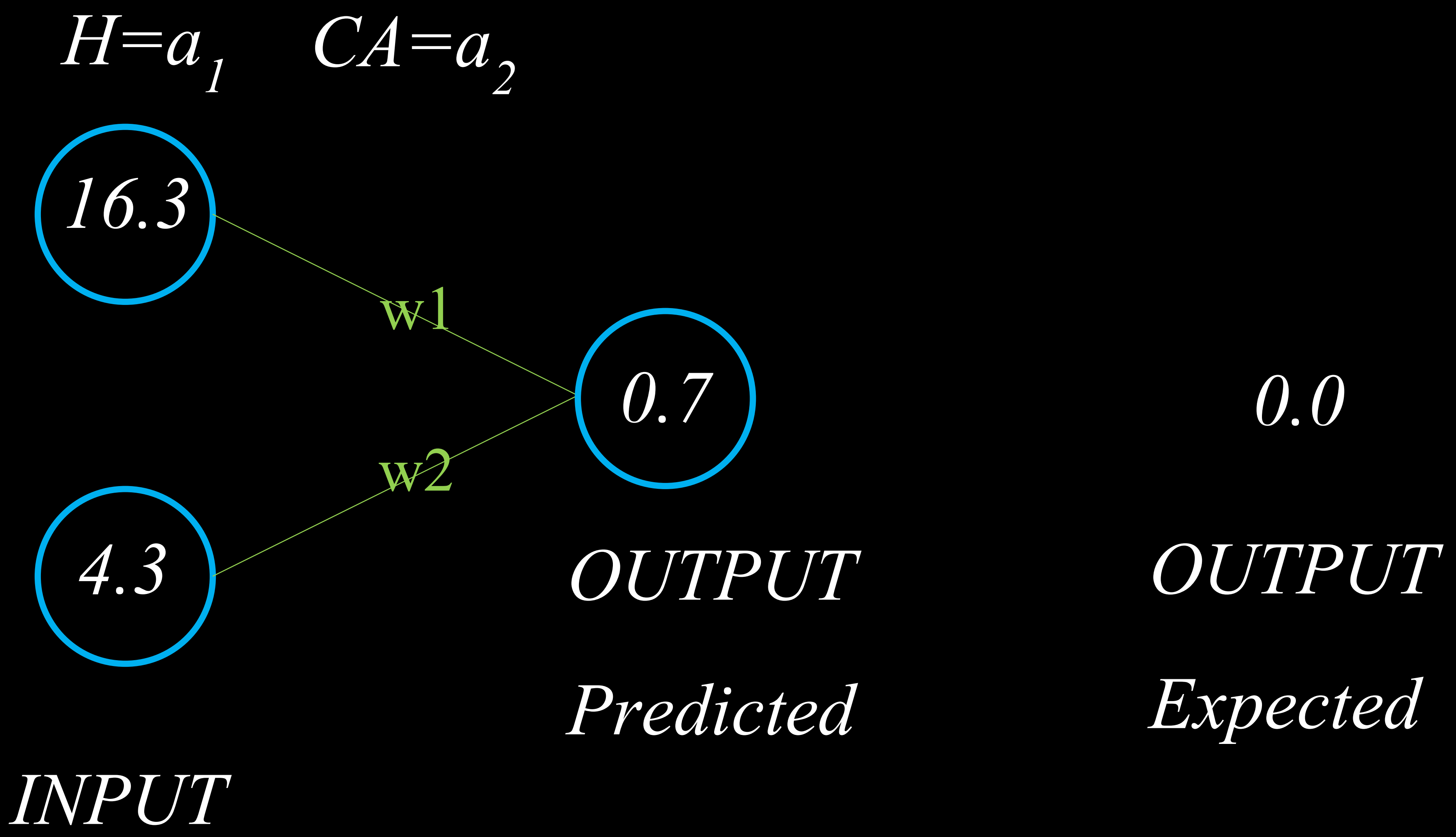
0.0

OUTPUT Predicted

OUTPUT Expected

INPUT

Initialization of weights (STEP 1)



Initialization of weights (STEP 1)

w1 = 0.0006
w2 = 0.0002
b=1

$$0.7 = Sigmoid\,(185x0.0006 + 85x0.0002 + 1) =$$

$$Sigmoid\,(0.111 + 0.281 + 1) = Sigmoid\,(1.281)$$

the back propagation

Computation of the error (STEP 2)

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

$=(\,Sigmoid\,(185x0.0006+85x0.0002+1) - 0.0)^2$

Weights and bias adjustment: the back propagation

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

$$0.0005 = 0.0006 - 0.0001$$

$$0.0001 = 0.0002 - 0.0001$$

$$0.0008 = 1 - 0.0002$$

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

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

$$b = b - LR * slope = b - LR * derivative_b \text{ of the Cost Function}$$

$$w1 = w1 - LR^* \frac{\partial costo}{\partial w1}$$

$$w2 = w2 - LR^* \frac{\partial costo}{\partial w2}$$

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

$$\frac{\partial \text{costo}}{\partial \text{w1}} = \frac{\partial \text{costo}}{\partial \text{p}} \times \frac{\partial \text{p}}{\partial \text{t}}$$

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

$$\frac{\partial \text{costo}}{\partial \text{w1}} = \frac{\partial \text{costo}}{\partial \text{p}} \times \frac{\partial \text{p}}{\partial \text{t}} \times \frac{\partial \text{t}}{\partial \text{w1}}$$

$$\frac{\partial \text{costo}}{\partial \text{w1}} = \text{2(sigmoide(2w1+5w2+b) - 1) x sigmoide(2w1+5w2+b)(1-sigmoide(2w1+5w2+b)) x 2}$$

$$\frac{\partial \text{costo}}{\partial \text{w1}} = 2(\text{sigmoide}(2w1+5w2+b) - 1) \times \text{sigmoide}(2w1+5w2+b)(1-\text{sigmoide}(2w1+5w2+b)) \times 2$$

$$\frac{\partial \text{costo}}{\partial \text{w2}} = 2(\text{sigmoide}(2w1+5w2+b) - 1) \times \text{sigmoide}(2w1+5w2+b)(1-\text{sigmoide}(2w1+5w2+b)) \times 5$$

$$\frac{\partial \text{costo}}{\partial \text{b}} = 2(\text{sigmoide}(2w1+5w2+b) - 1) \times \text{sigmoide}(2w1+5w2+b)(1-\text{sigmoide}(2w1+5w2+b)) \times 1$$

$$\frac{\partial costo}{\partial w1} = \frac{\partial costo}{\partial w1} = \frac{\partial costo}{\partial p} \times \frac{\partial p}{\partial t} \times \frac{\partial t}{\partial w1}$$

$$= 2(sigmoide(2w1+5w2+b) - 1) \times sigmoide(2w1+5w2+b)(1-sigmoide(2w1+5w2+b)) \times 2$$

$$\frac{\partial costo}{\partial w2} = \frac{\partial costo}{\partial w2} = \frac{\partial costo}{\partial p} \times \frac{\partial p}{\partial t} \times \frac{\partial t}{\partial w2}$$

$$= 2(sigmoide(2w1+5w2+b) - 1) \times sigmoide(2w1+5w2+b)(1-sigmoide(2w1+5w2+b)) \times 5$$

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

$$= 2(sigmoide(2w1+5w2+b) - 1) \times sigmoide(2w1+5w2+b)(1-sigmoide(2w1+5w2+b)) \times 1$$

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

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

Back propagation

Weights and bias adjustment

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

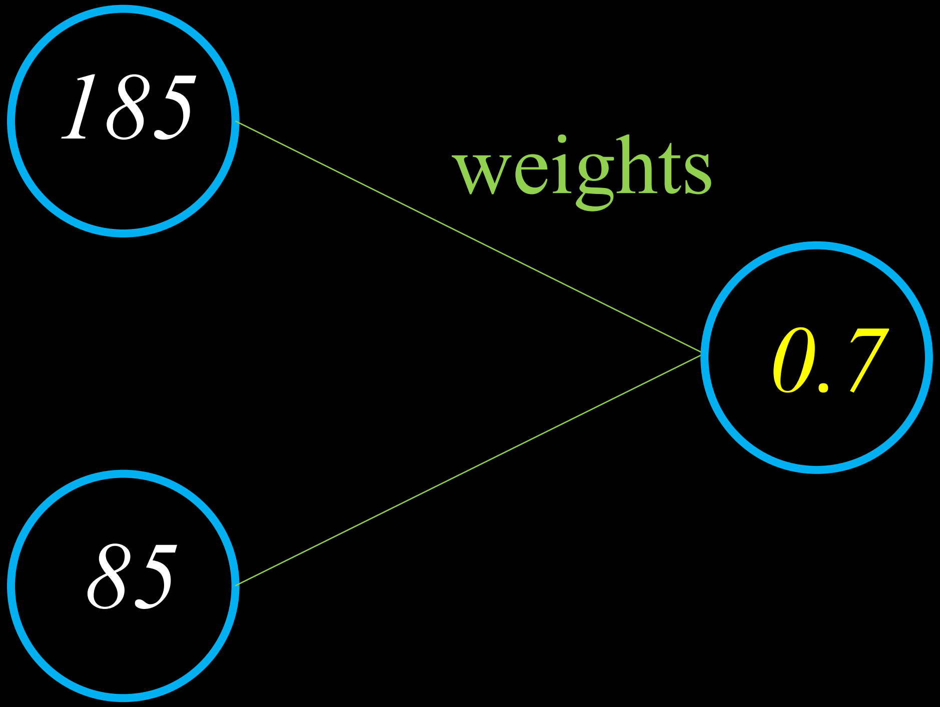
The Cost Function $(w_1\ w_2\ b)$ = (Sigmoid-0.0)^2=0.43

$$Sigmoid\ (185x0.0005 + 85x0.0001 + 1) = Sigmoid\ (0.0925 + 0.085 + 0.008) = Sigmoid\ (1.178) = 0.65$$

the back propagation

$H = a_1$ $CA = a_2$

Forward propagation

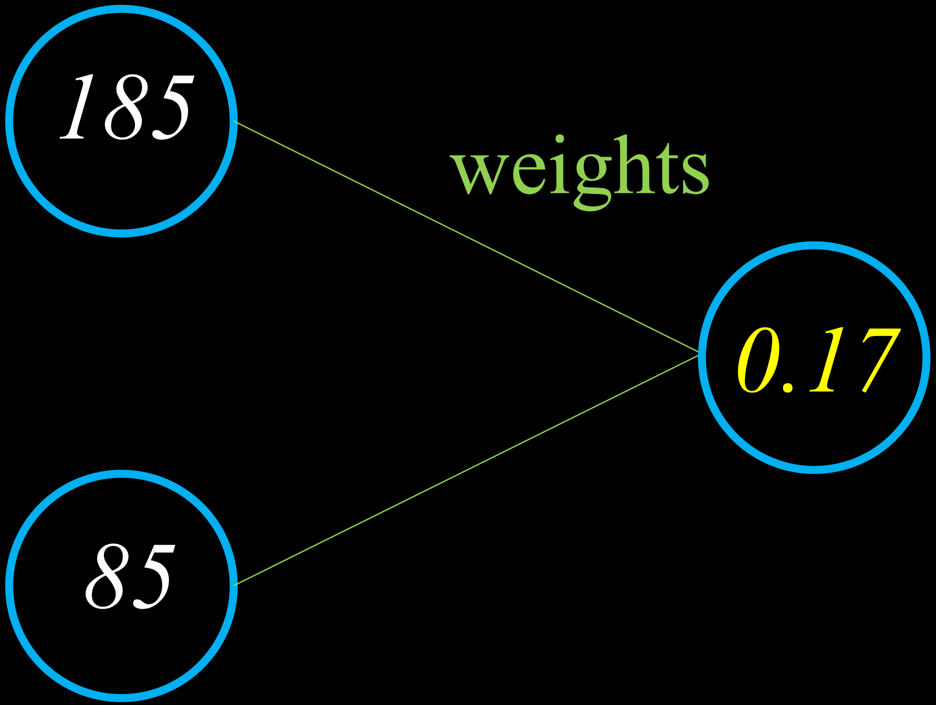


Activation function

$$Sigmoid\,(185w_1 + 85w_2 + b)$$

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

Back propagation



Cost function

$$\left(Sigmoid\left(185w_1 + 85w_2 + b\right) - 0.0\right)^2$$

Training set: ‘healthy’ vs overweighting men

H	185	178	184	190	169	188	185	192	175	H	168	179	170	180	169	189	188	190	178	H	185	192	175	185	192	168	190	186	168
C	85	94	102	80	98	110	116	77	84	C	88	90	68	120	112	109	89	92	94	C	77	103	119	98	99	100	98	96	78

H	188	190	178	169	188	185	192	190	188	H	185	192	168	179	188	185	192	175	180	H	169	189	188	190	169	188	185	192	188
C	90	79	68	120	98	99	102	104	89	C	87	90	102	98	96	78	90	103	110	C	90	89	90	79	68	120	96	99	79

H	168	179	170	182	168	178	190	188	177	H	175	188	178	169	176	168	179	170	180	H	178	193	190	169	188	185	177	186	165
C	113	90	90	79	68	120	98	96	78	C	119	98	106	116	77	103	119	98	77	C	90	79	68	120	112	109	89	92	78

H	190	170	179	170	180	179	177	185	159	H	176	168	179	188	185	177	168	179	188	H	176	181	174	185	192	175	185	192	169
C	68	120	98	96	77	103	119	98	89	C	104	77	98	96	78	90	79	68	120	C	102	98	96	78	82	95	99	100	96

Weights initialization (STEP 1)

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 Trainig iteration)

Training set: ‘healthy’ vs overweighting men

H	185	178	184	190	169	188	185	192	175
C	85	94	102	80	98	110	116	77	84

H	168	179	170	180	169	189	188	190	178
C	88	90	68	120	112	109	89	92	94

H	185	192	175	185	192	168	190	186	168
C	77	103	119	98	99	100	98	96	78

H	188	190	178	169	188	185	192	190	188
C	90	79	68	120	98	99	102	104	89

H	185	192	168	179	188	185	192	175	180
C	87	90	102	98	96	78	90	103	110

H	169	189	188	190	169	188	185	192	188
C	90	89	90	79	68	120	96	99	79

H	168	179	170	182	168	178	190	188	177
C	113	90	90	79	68	120	98	96	78

H	175	188	178	169	176	168	179	170	180
C	119	98	106	116	77	103	119	98	77

H	178	193	190	169	188	185	177	186	165
C	90	79	68	120	112	109	89	92	78

H	190	170	179	170	180	179	177	185	159
C	68	120	98	96	77	103	119	98	89

H	176	168	179	188	185	177	168	179	188
C	104	77	98	96	78	90	79	68	120

H	176	181	174	185	192	175	185	192	169
C	102	98	96	78	82	95	99	100	96

$$Sigmoid\,(185x0.0006+85x0.0002+1)=0.7$$

$$Sigmoid\,(178x0.0004+94x0.0003+0.008)=0.65$$

$$Sigmoid\,(184x0.0003+102x0.0001+0.007)=0.64$$

$$Sigmoid\,(100x0.0005+80x0.0002+0.008)=0.55$$

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

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

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

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

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 avarage of all weight updates are used only once in each epoch)

Mini-Batch method (mix)

The loss function computes the error for a single training example, the cost function is the average of the loss functions of the entire training

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

Batch method applied

21-40

41-60

61-80

5 weight update will be performed to complete the training process

Speed of SGD

Stability of batch

SOFTWARE CODES

Matlab: essential functions and scripts

Matlab: simple examples

Sigmoid.m

```
function y = Sigmoid(x)
y = 1/(1+exp(-x));
end
```

Sigmoid.m SGD_method.m

```
function Weight = SGD_method(Weight, input, correct_Output)
alpha = 0.9;

N = 4;
for k = 1:N
    transposed_Input = input(k, :)';
    d = correct_Output(k);
weighted_Sum = Weight*transposed_Input;
output = Sigmoid(weighted_Sum);

error     = d - output;
delta = output*(1-output)*error;

dWeight = alpha*delta*transposed_Input;

Weight(1) = Weight(1) + dWeight(1);
Weight(2) = Weight(2) + dWeight(2);
```

Sigmoid.m SGD_method.m*

```
N = 4;
for k = 1:N
    transposed_Input = input(k, :)';
    d = correct_Output(k);
weighted_Sum = Weight*transposed_Input;
output = Sigmoid(weighted_Sum);

error      = d - output;
delta = output*(1-output)*error;

dWeight = alpha*delta*transposed_Input;

Weight(1) = Weight(1) + dWeight(1);
Weight(2) = Weight(2) + dWeight(2);
Weight(3) = Weight(3) + dWeight(3);
end
end
```

Sigmoid.m | SGD_method.m | Training.m

```matlab
input = [ 0 0 1;
          0 1 1;
          1 0 1;
          1 1 1;
        ];
correct_Output = [0
                  0
                  1
                  1
                 ];
Weight = 2*rand(1, 3) - 1;
for epoch = 1:10000
    Weight = SGD_method(Weight, input, correct_Output);
end

save('Trained_Network.mat')
```

Sigmoid.m | SGD_method.m | Training.m | testing.m*

```
load('Trained_Network.mat');
input = [ 0 0 1;
          0 1 1;
          1 0 1;
          1 1 1;
        ];
N = 4;
for k = 1:N
    transposed_Input = input(k, :)';
    weighted_Sum = Weight*transposed_Input;
    output = Sigmoid(weighted_Sum)
end
```