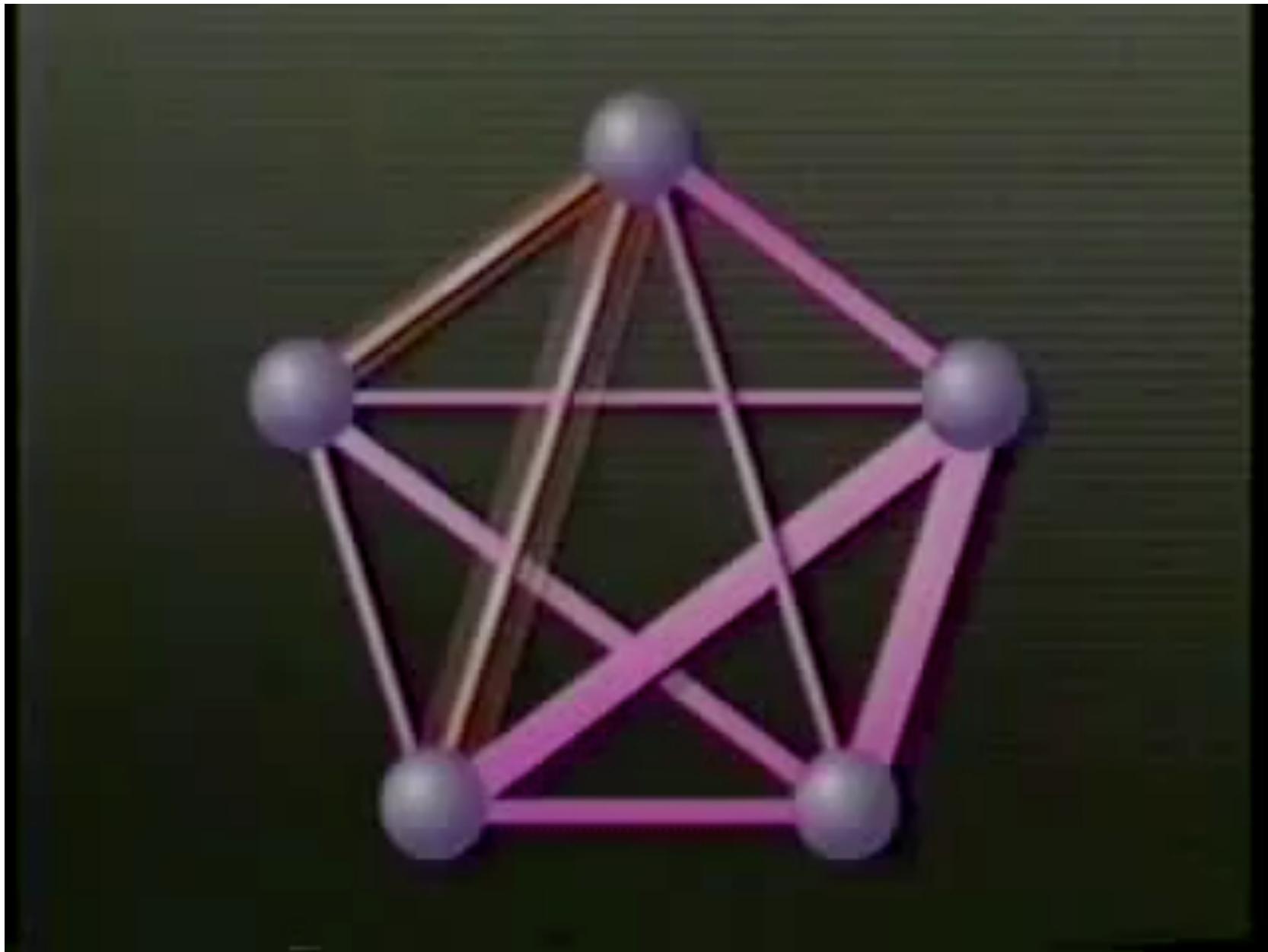


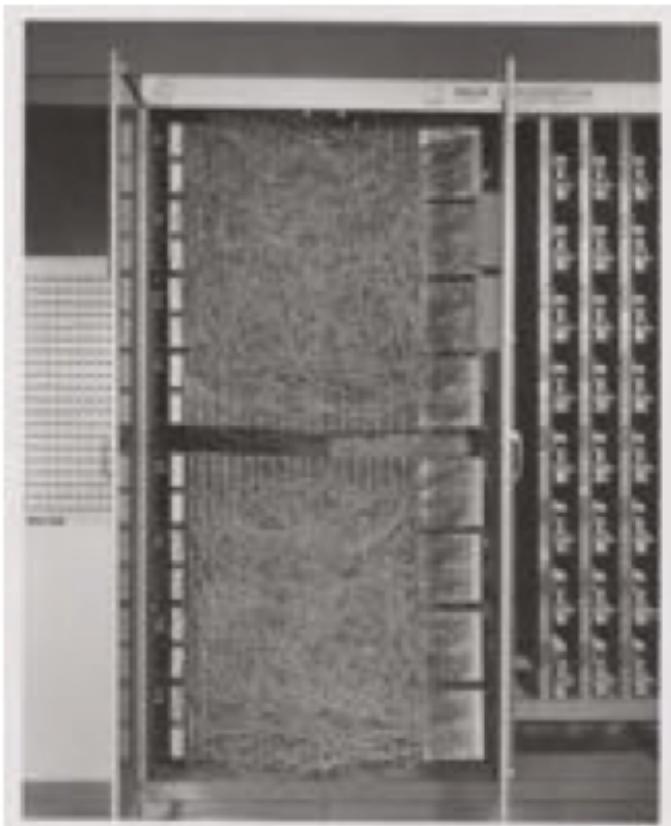
PART II: FOUNDATIONS OF “SHALLOW” NEURAL NETWORKS



Rosenblatt Perceptron

FIRST IMPLEMENTATION OF NEURAL NETWORK [Rosenblatt, 1957!]

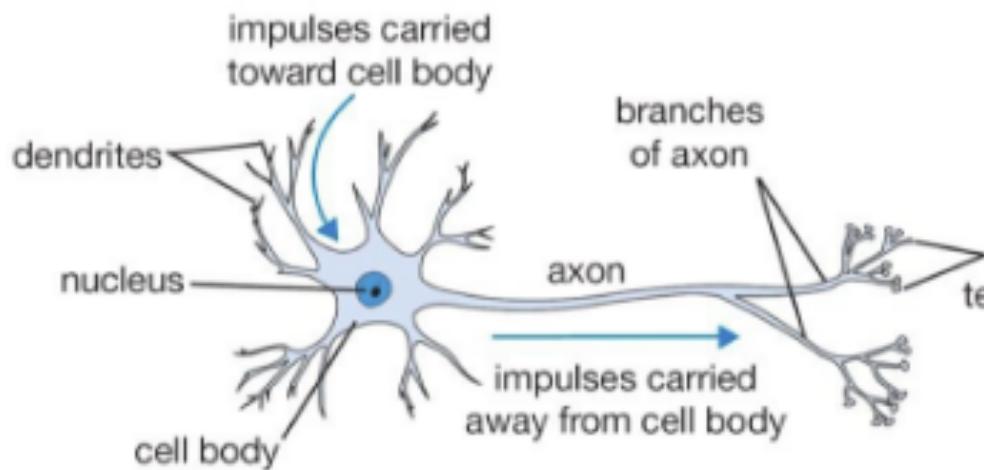
INTENDED TO BE A MACHINE (NOT AN ALGORITHM)



it had an array of 400 photocells,
randomly connected to the "neurons".

Weights were encoded in
potentiometers, and weight updates
during learning were performed by
electric motors

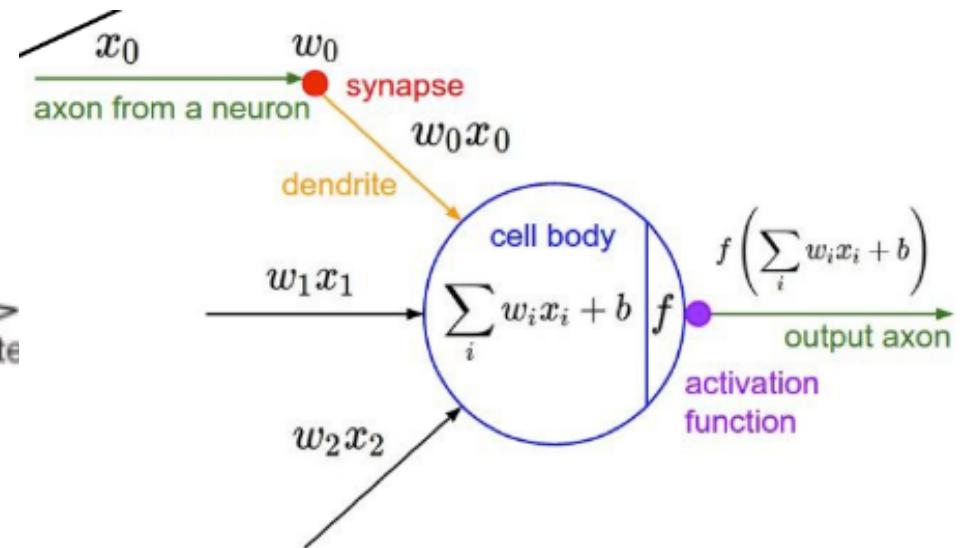
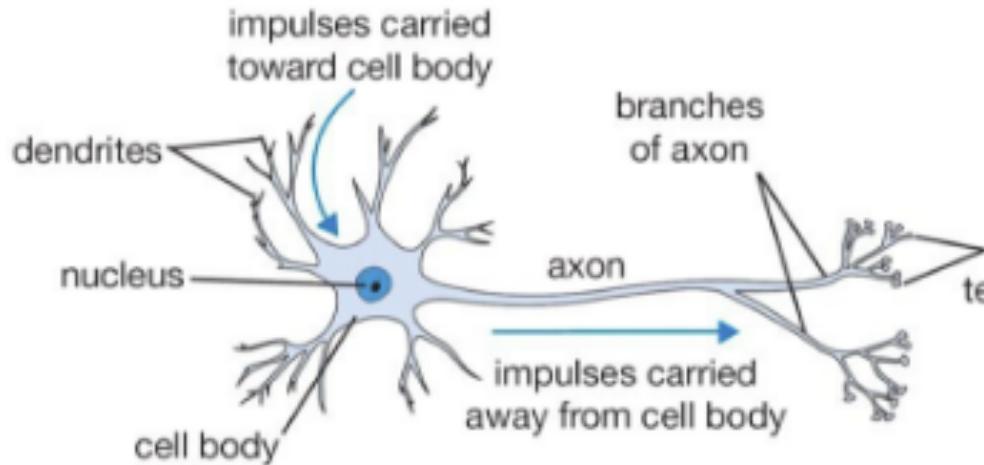
THE NEURON



INSPIRED BY NEURO - SCIENCE?

Credit: Karpathy

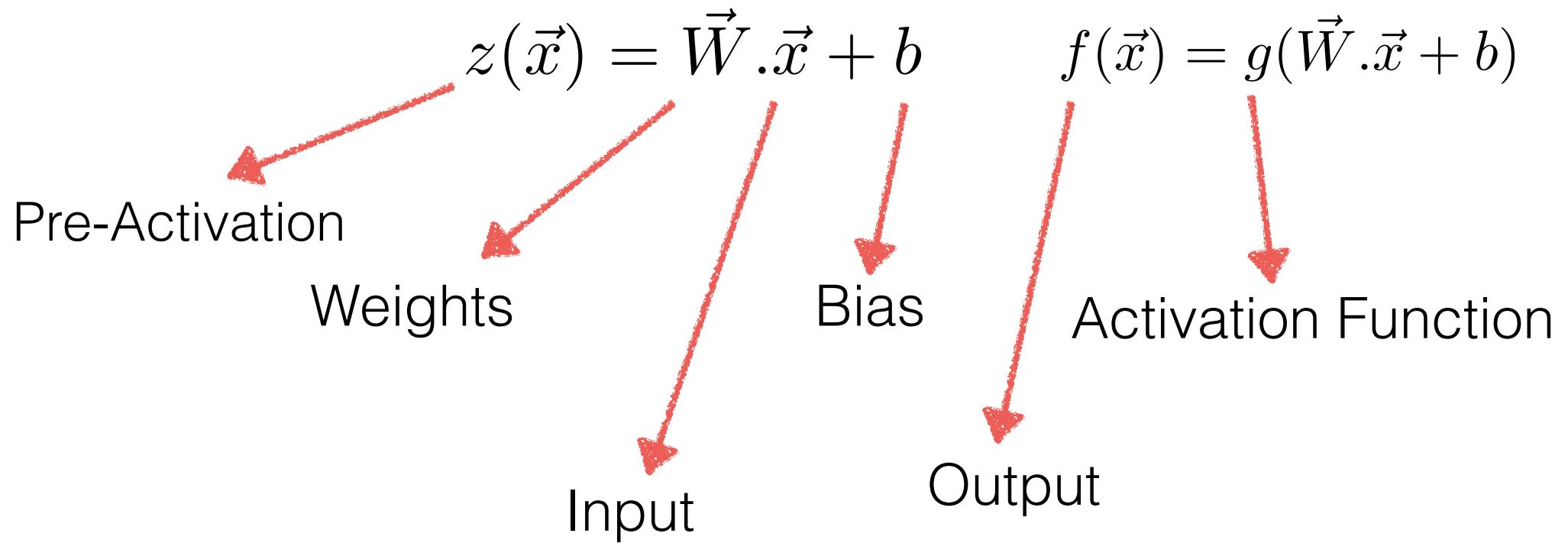
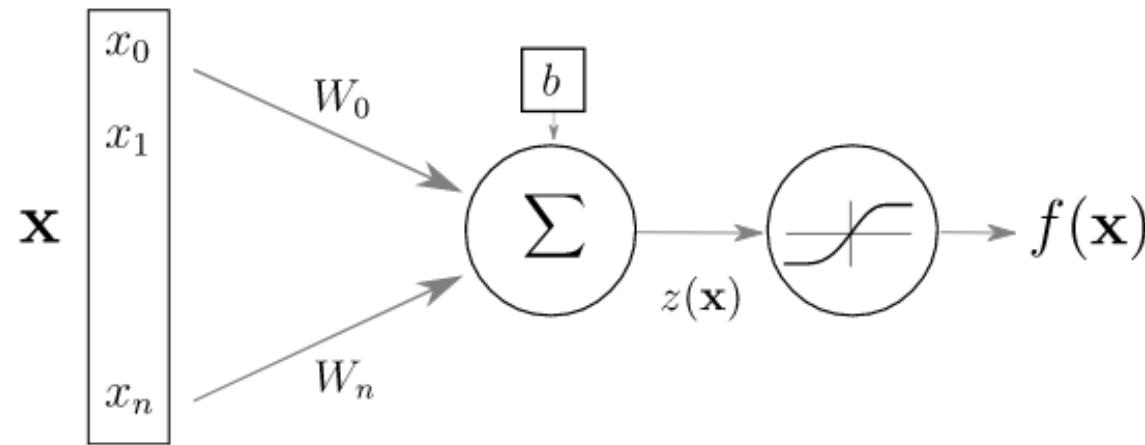
THE NEURON



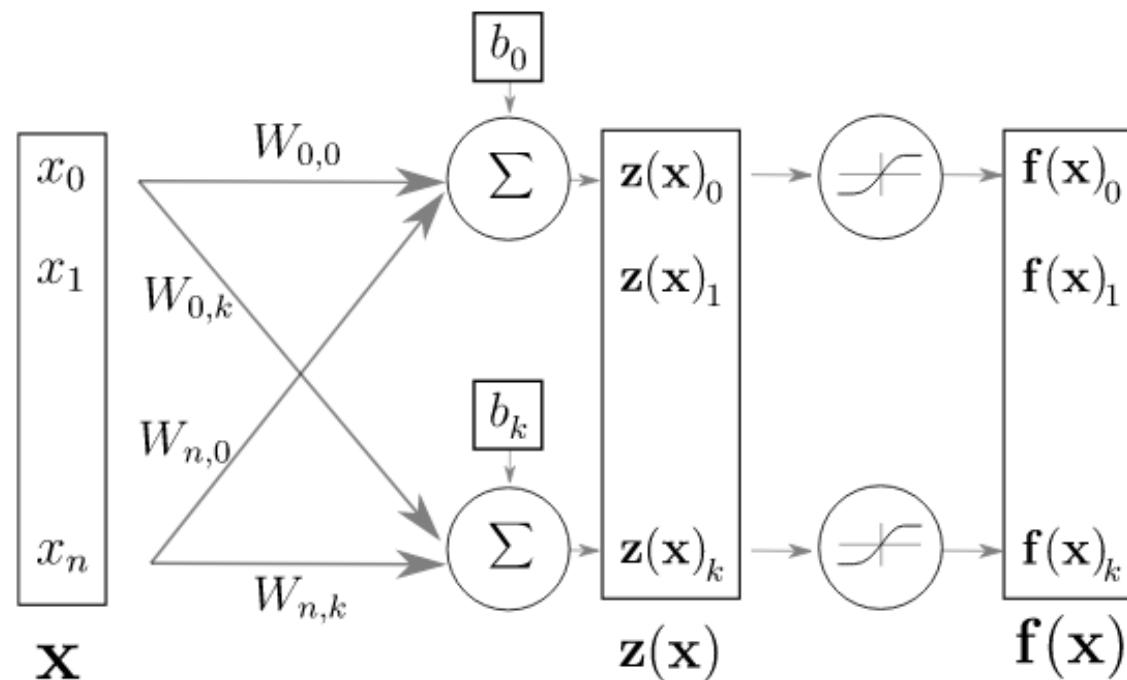
INSPIRED BY NEURO - SCIENCE?

Credit: Karpathy

TODAY'S ARTIFICIAL NEURON



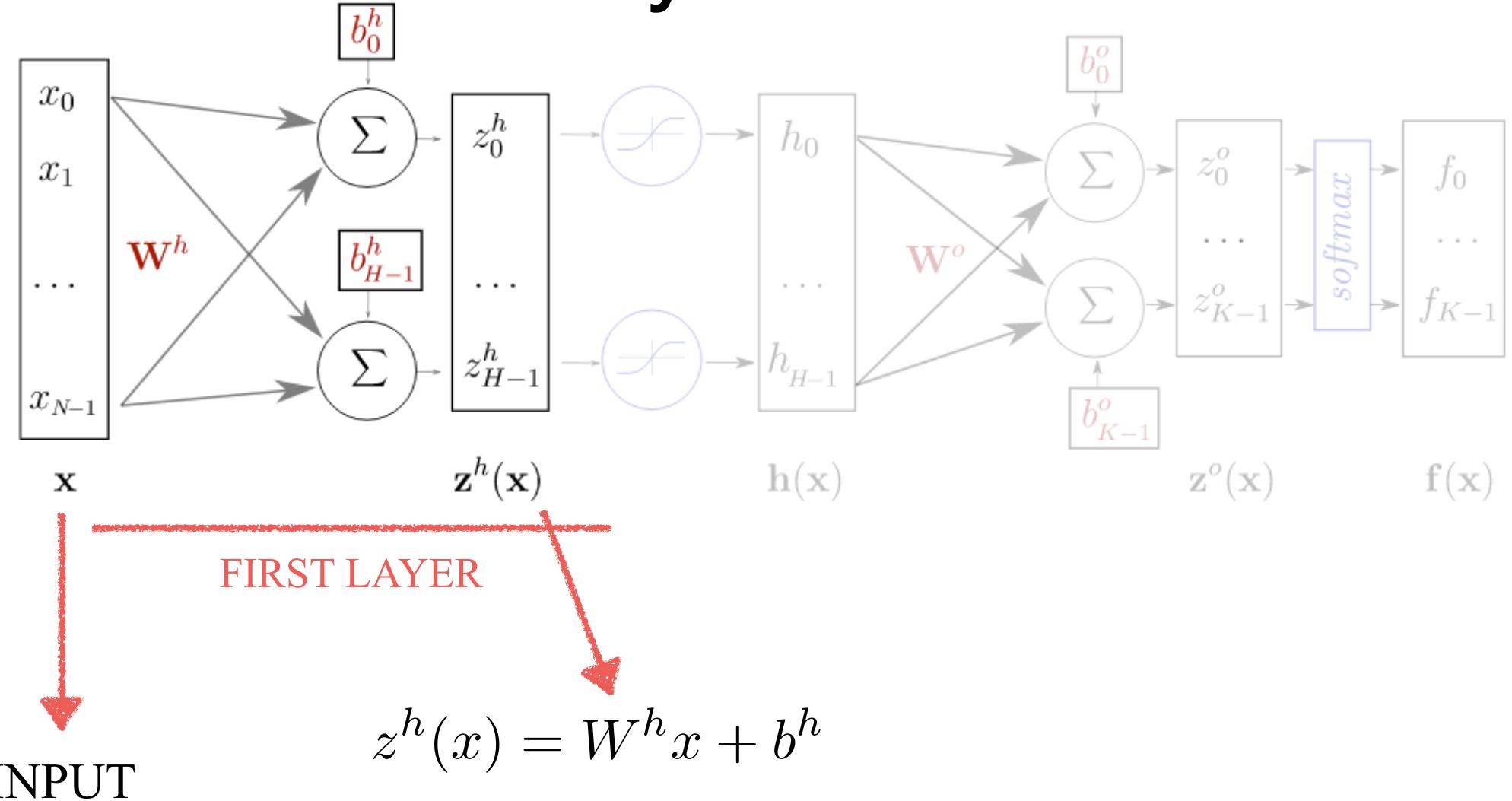
LAYER OF NEURONS



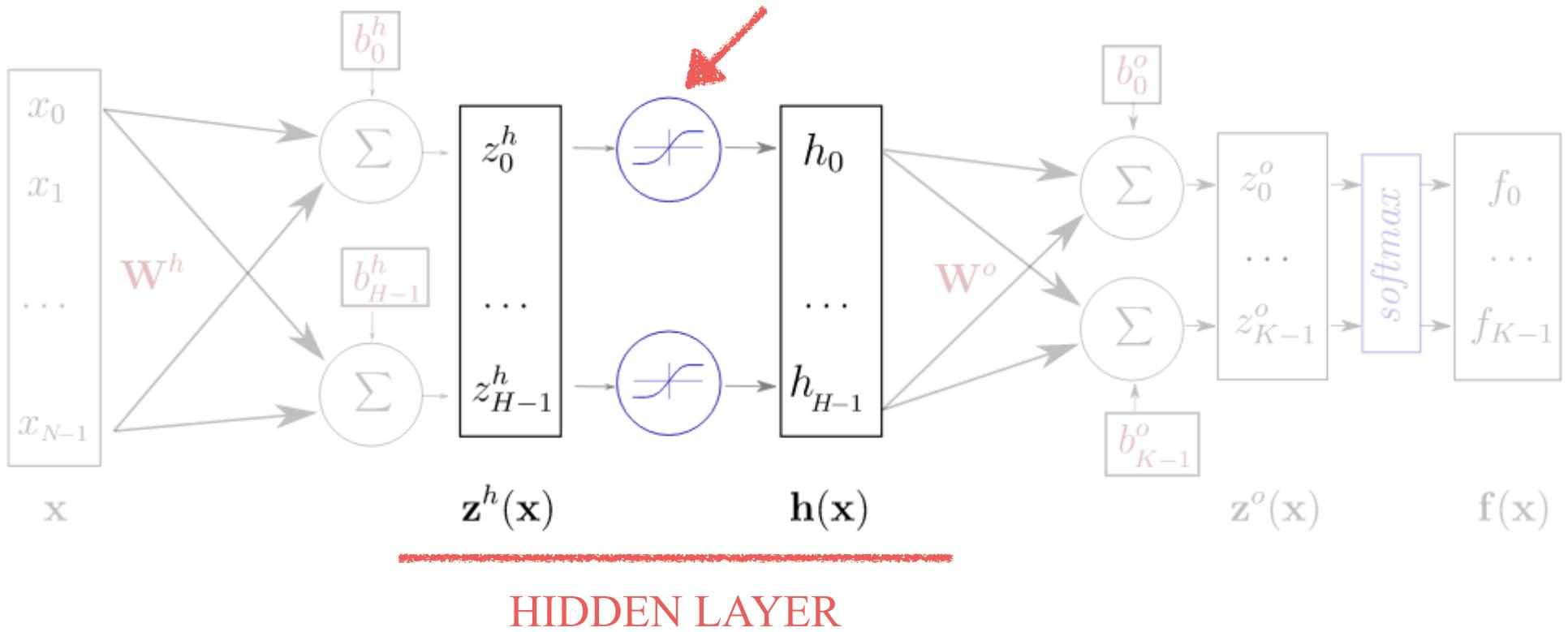
$$f(\vec{x}) = g(\mathbf{W} \cdot \vec{x} + \vec{b})$$

SAME IDEA. NOW **W** becomes a matrix and **b** a vector

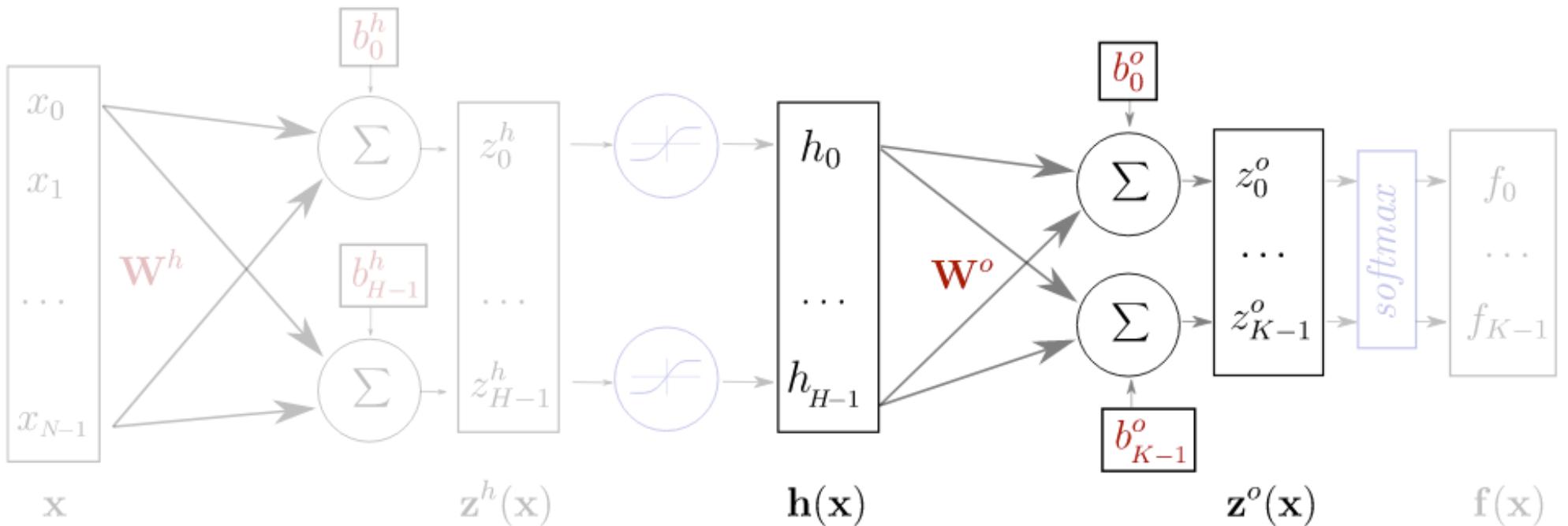
Hidden Layers of Neurons



ACTIVATION FUNCTION

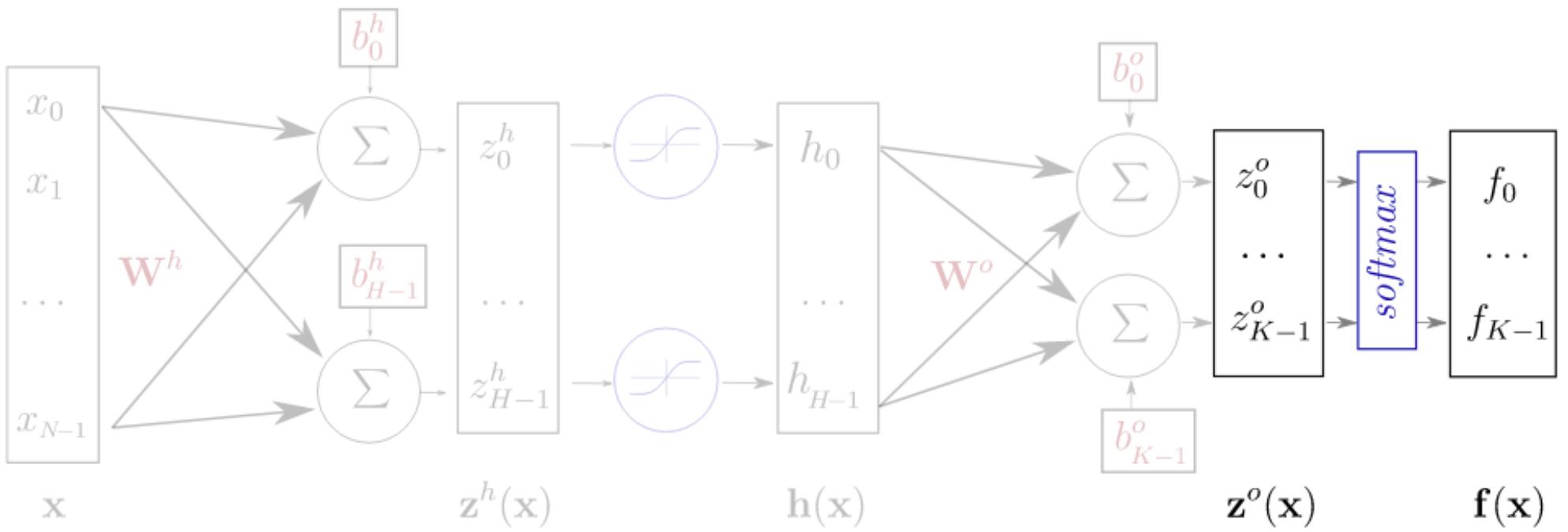


$$h(x) = g(z^h(x)) = g(W^h x + b^h)$$



OUTPUT LAYER

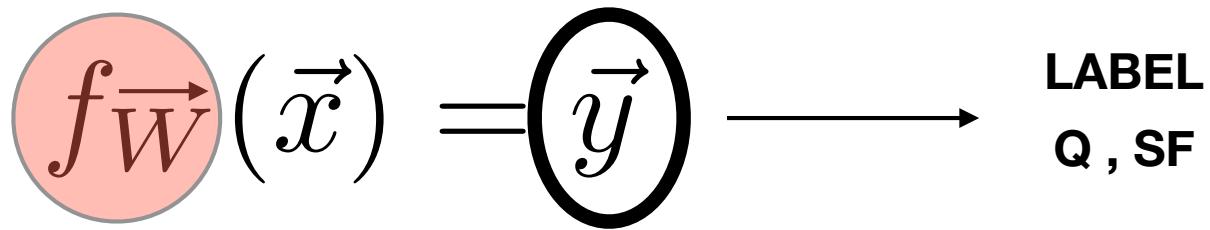
$$z^0(\mathbf{x}) = W^0 h(\mathbf{x}) + b^0$$



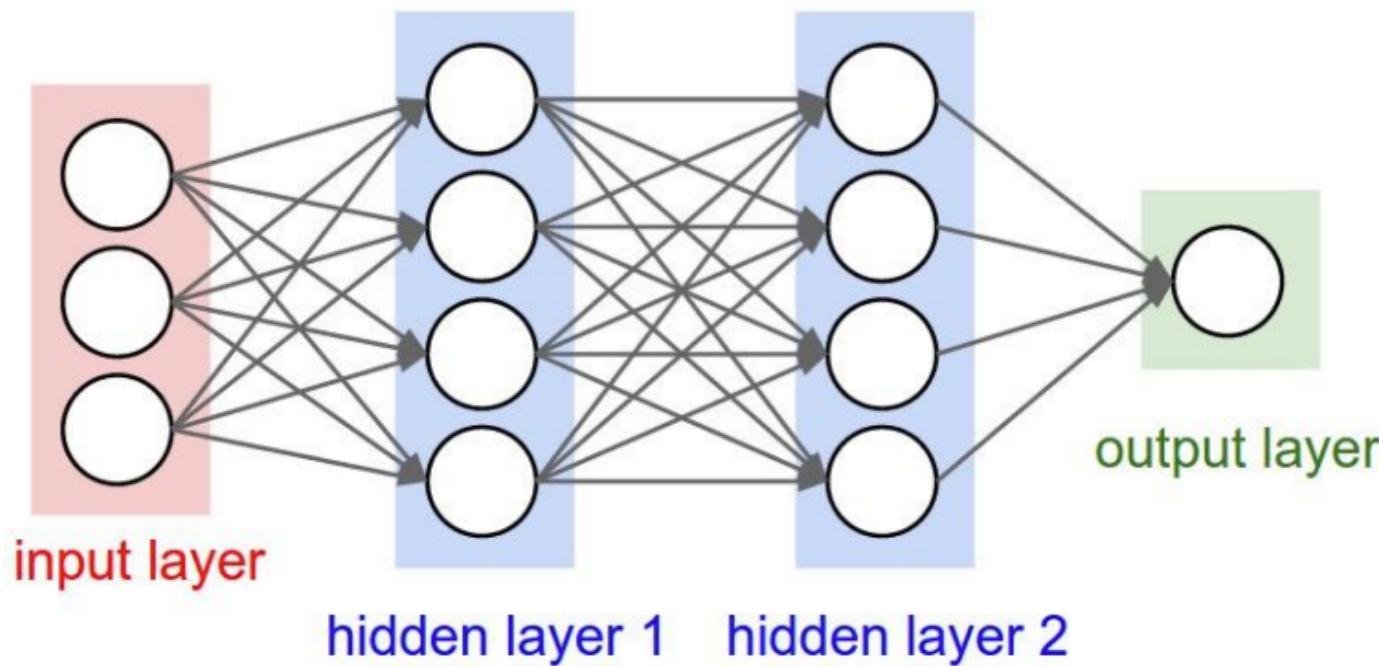
PREDICTION LAYER

$$f(\mathbf{x}) = \text{softmax}(\mathbf{z}^o)$$

**"CLASSICAL"
MACHINE LEARNING**



**REPLACE THIS BY A GENERAL
NON LINEAR FUNCTION WITH SOME PARAMETERS W**



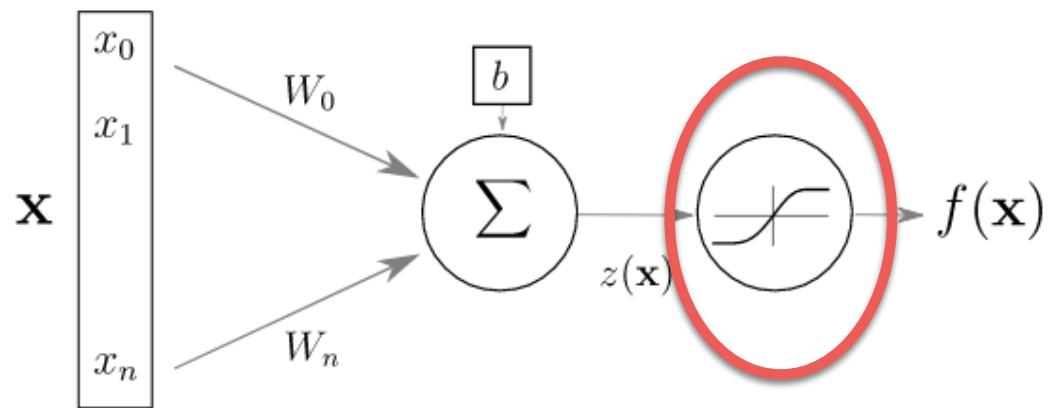
$$p = g_3(W_3g_2(W_2g_1(W_1\vec{x}_0)))$$

← NETWORK
FUNCTION

SO LET'S GO DEEPER AND DEEPER!

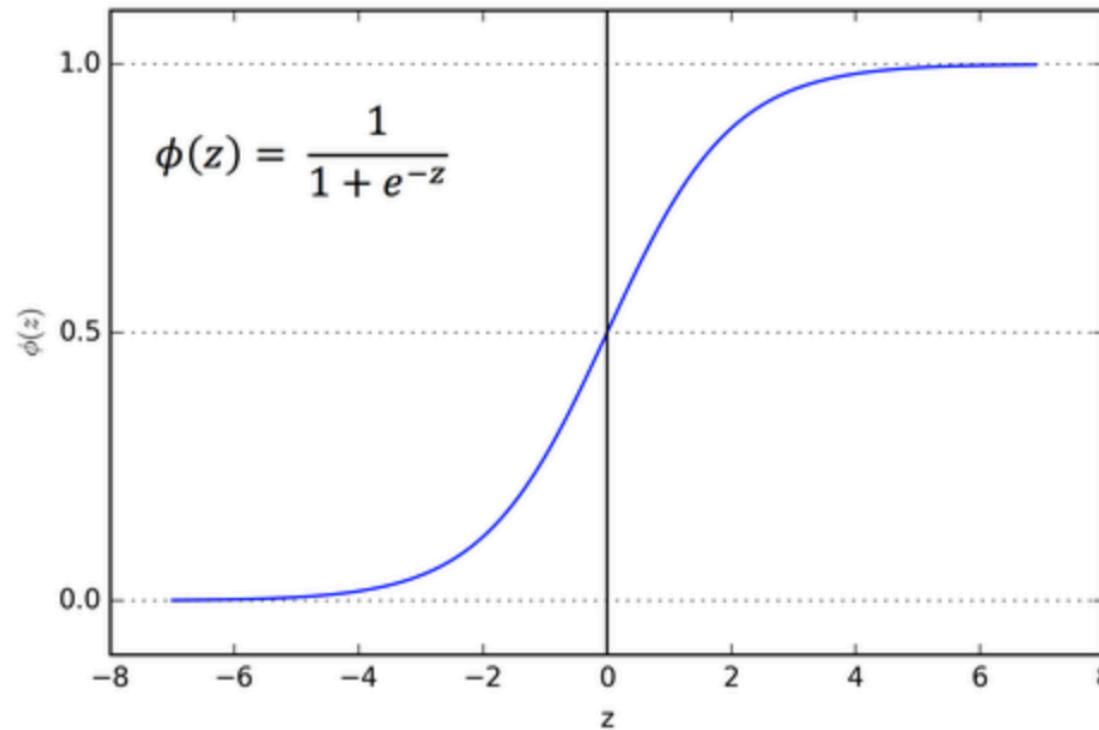
LET'S FIRST EXAMINE IN MORE DETAIL HOW SIMPLE
“SHALLOW” NETWORKS WORK

ACTIVATION FUNCTIONS?

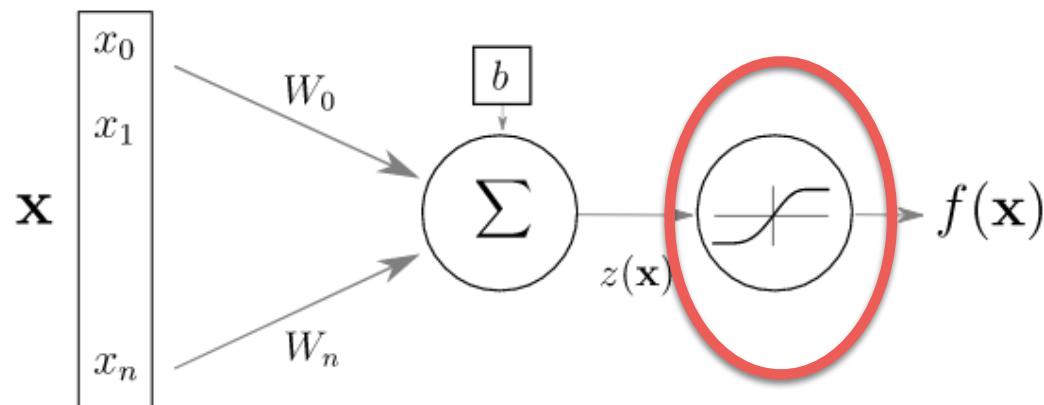


ADD NON LINEARITIES TO THE PROCESS

ACTIVATION FUNCTIONS?

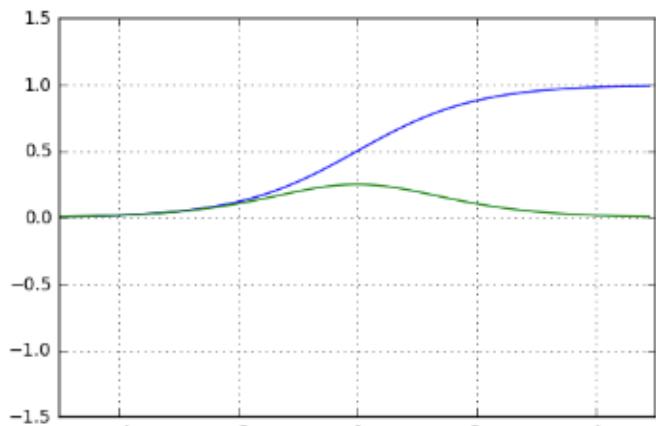


the sigmoid function



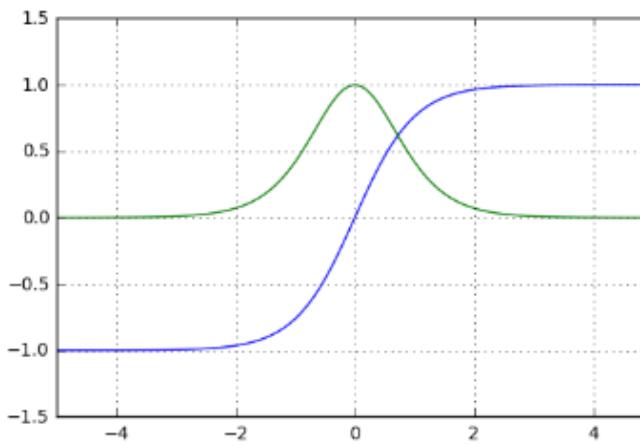
ADD NON LINEARITIES TO THE PROCESS

ACTIVATION FUNCTIONS



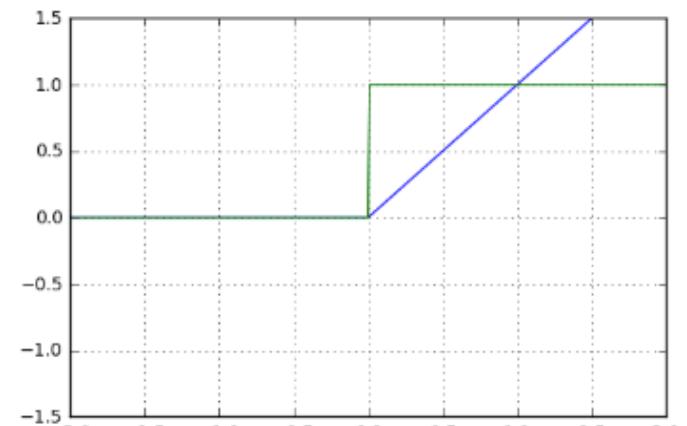
$$\text{sigm}(x) = \frac{1}{1 + e^{-x}}$$

$$\text{sigm}'(x) = \text{sigm}(x)(1 - \text{sigm}(x))$$



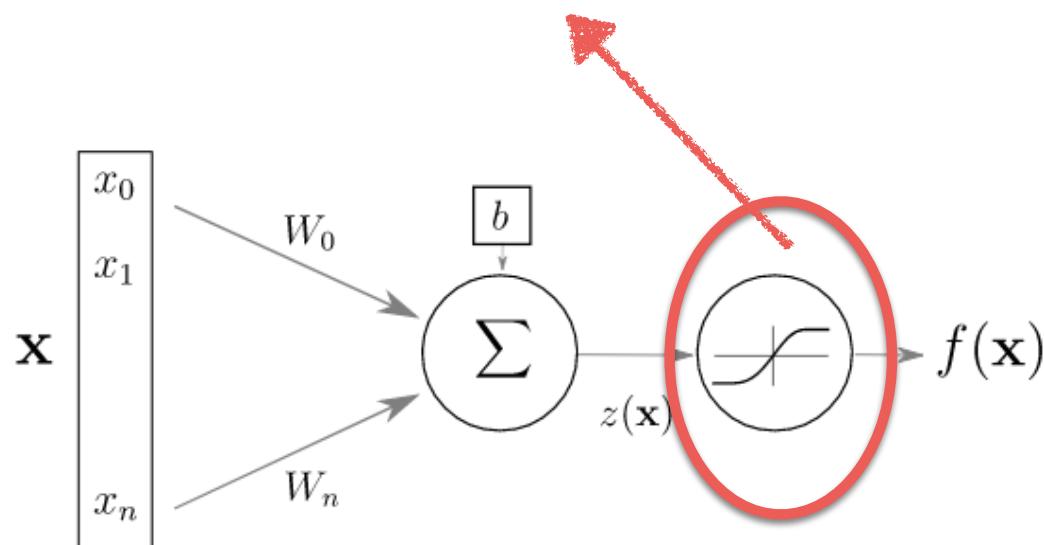
$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

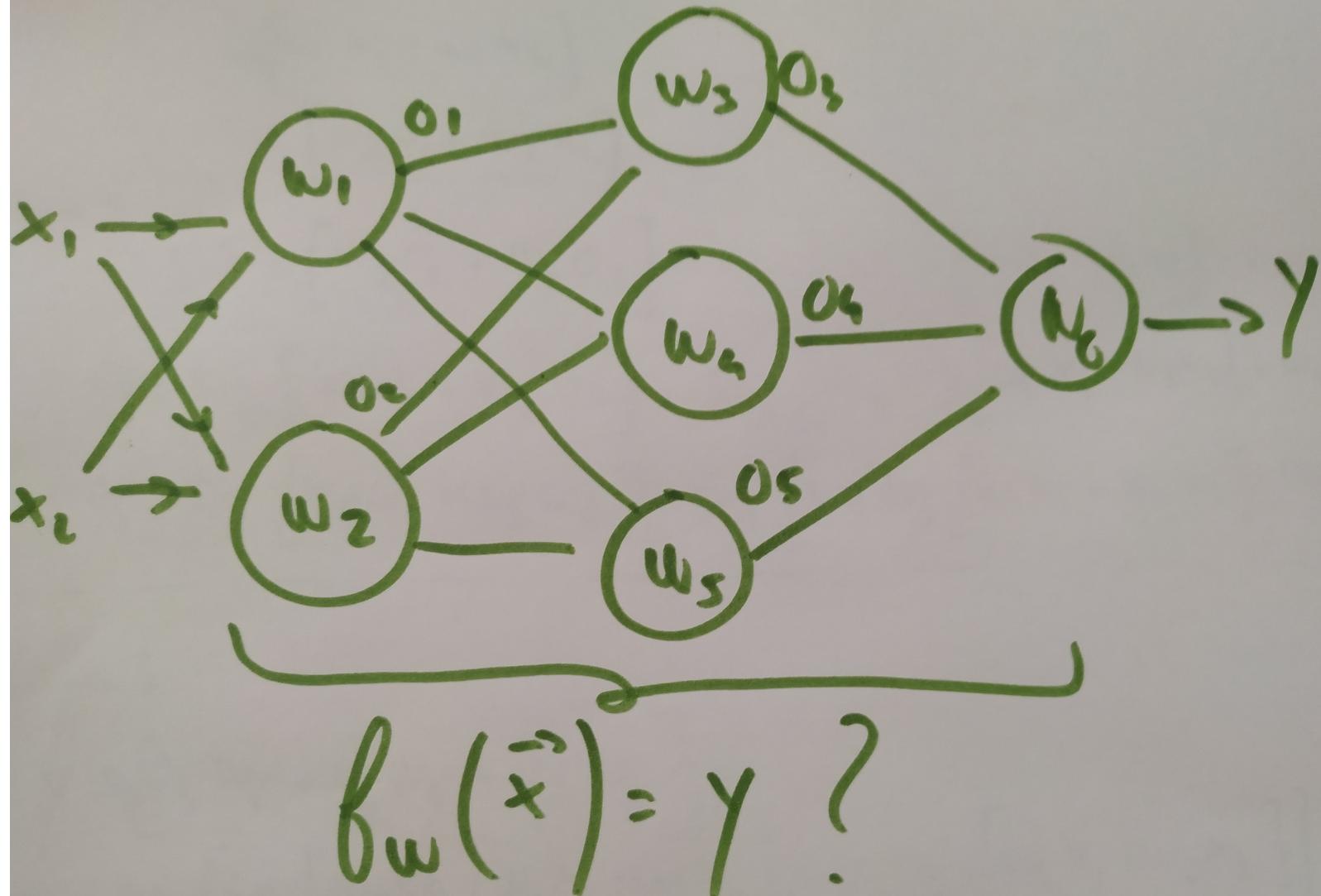
$$\tanh'(x) = 1 - \tanh(x)^2$$



$$\text{relu}(x) = \max(0, x)$$

$$\text{relu}'(x) = 1_{x>0}$$

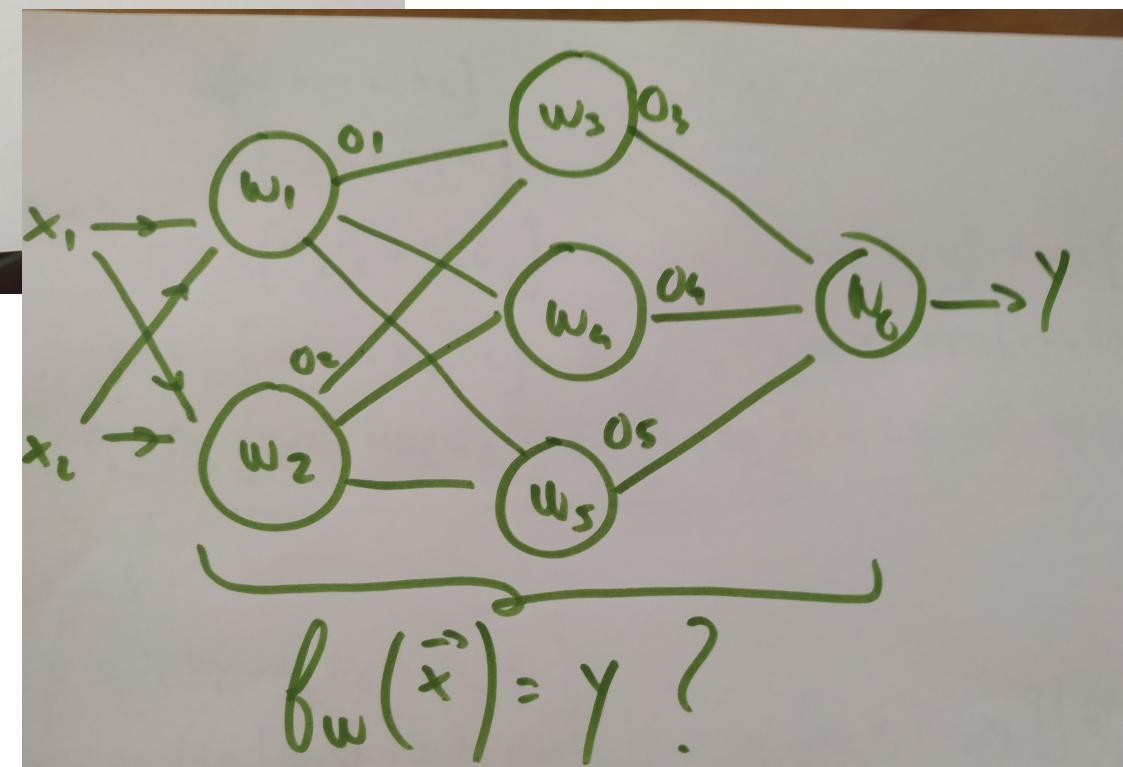




$$A: \text{sig} [\omega_3 \text{sig} [\omega_1 x_1 + \omega_2 x_2] \cdot w_6 + \omega_4 \text{sig} [\omega_2 x_1 + \omega_2 x_2] \cdot w_6 \\ + \omega_5 \text{sig} [\omega_1 x_1 + \omega_1 x_2] \cdot w_6]$$

$$B: \text{sig} [\text{sig} [\omega_3 \text{sig} [\omega_1 x_1 + \omega_1 x_2] + \omega_5 \text{sig} [\omega_2 x_1 + \omega_2 x_2]] \cdot w_6 + \\ \text{sig} [\omega_4 \text{sig} [\omega_1 x_1 + \omega_1 x_2] + \omega_6 \text{sig} [\omega_2 x_1 + \omega_2 x_2]] \cdot w_6 + \\ \text{sig} [\omega_5 \text{sig} [\omega_1 x_1 + \omega_1 x_2] + \omega_5 \text{sig} [\omega_2 x_1 + \omega_2 x_2]] \cdot w_6]$$

$$C: \text{sig} [\text{sig} [\omega_3 \text{sig} [\omega_1 x_1 + \omega_1 x_2] \cdot w_6 + \omega_4 \text{sig} [\omega_2 x_1 + \omega_2 x_2] \cdot w_6 \\ + \omega_5 \text{sig} [\omega_1 x_1 + \omega_1 x_2] \cdot w_6]$$



When poll is active, respond at **pollev.com/marchuertasc257**

Text **MARCHUERTASC257** to **22333** once to join

Which expression corresponds to the Neural Network shown in the figure?

A
B
C

REMEMBER WE NEED A LOSS FUNCTION ...

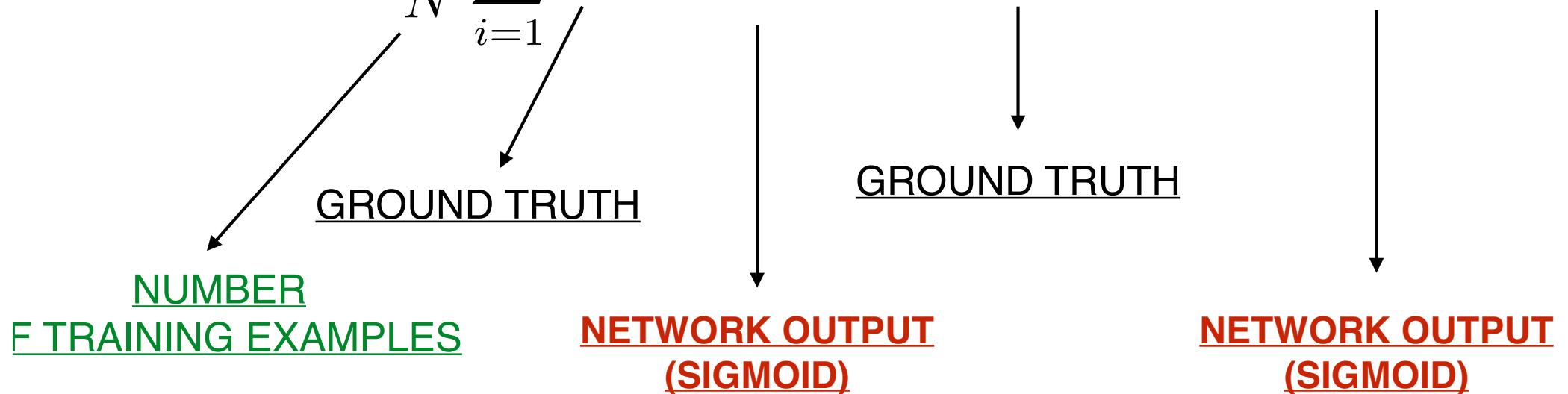
REMEMBER WE NEED A LOSS FUNCTION ...

LET'S SART WITH A SIMPLE CLASSIFICATION PROBLEM

BINARY CLASSIFICATION LOSS

WE TYPICALLY USE THE BINARY CROSS-ENTROPY LOSS:

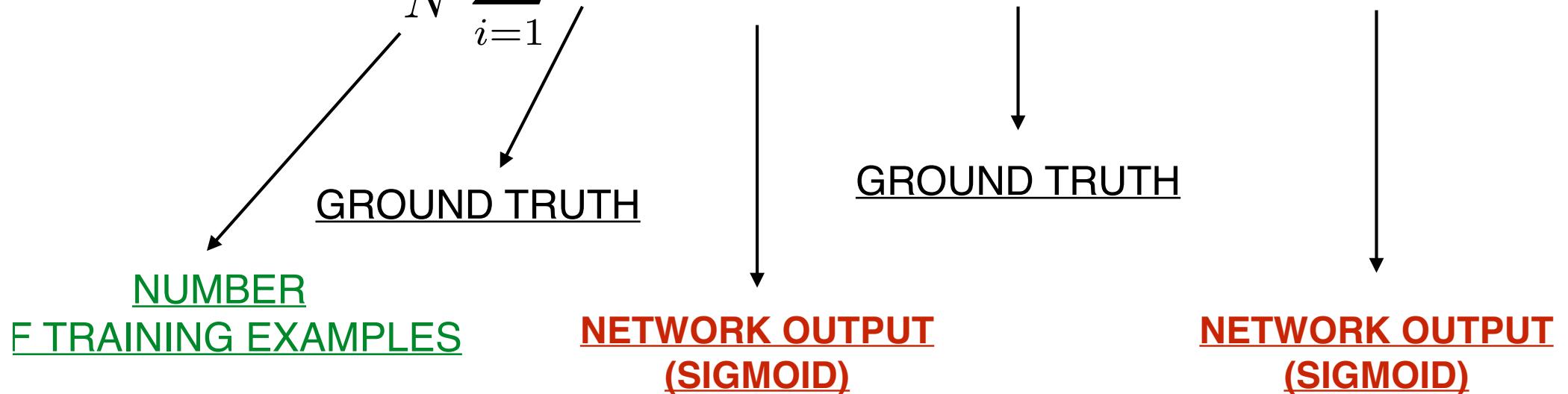
$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \log(p(y_i)) + (1 - y_i) \log(1 - p(y_i))$$



BINARY CLASSIFICATION LOSS

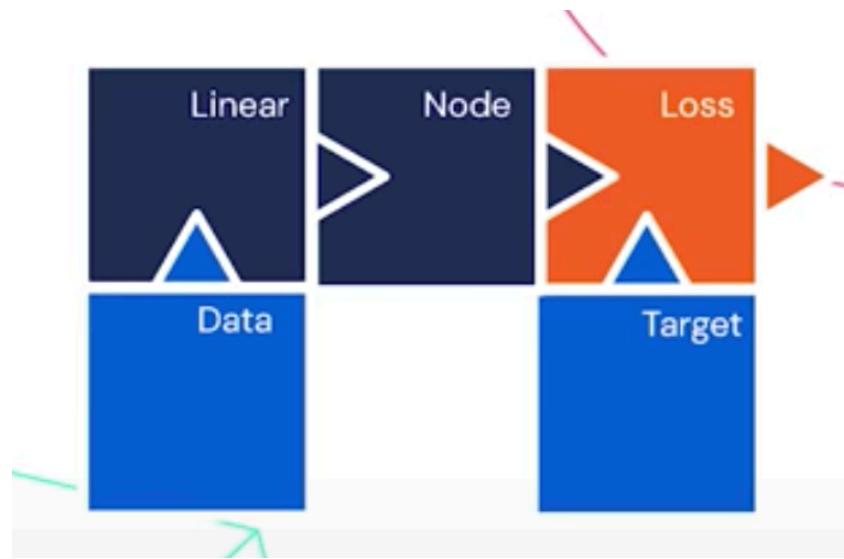
WE MINIMIZE THE CROSS ENTROPY BETWEEN THE TRUE [q(y)] AND PREDICTED [p(y)] DISTRIBUTIONS

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \log(p(y_i)) + (1 - y_i) \log(1 - p(y_i))$$



OK, SO NOW WE HAVE THE 3 MAIN INGREDIENTS THAT COMPOSE AN ANN:

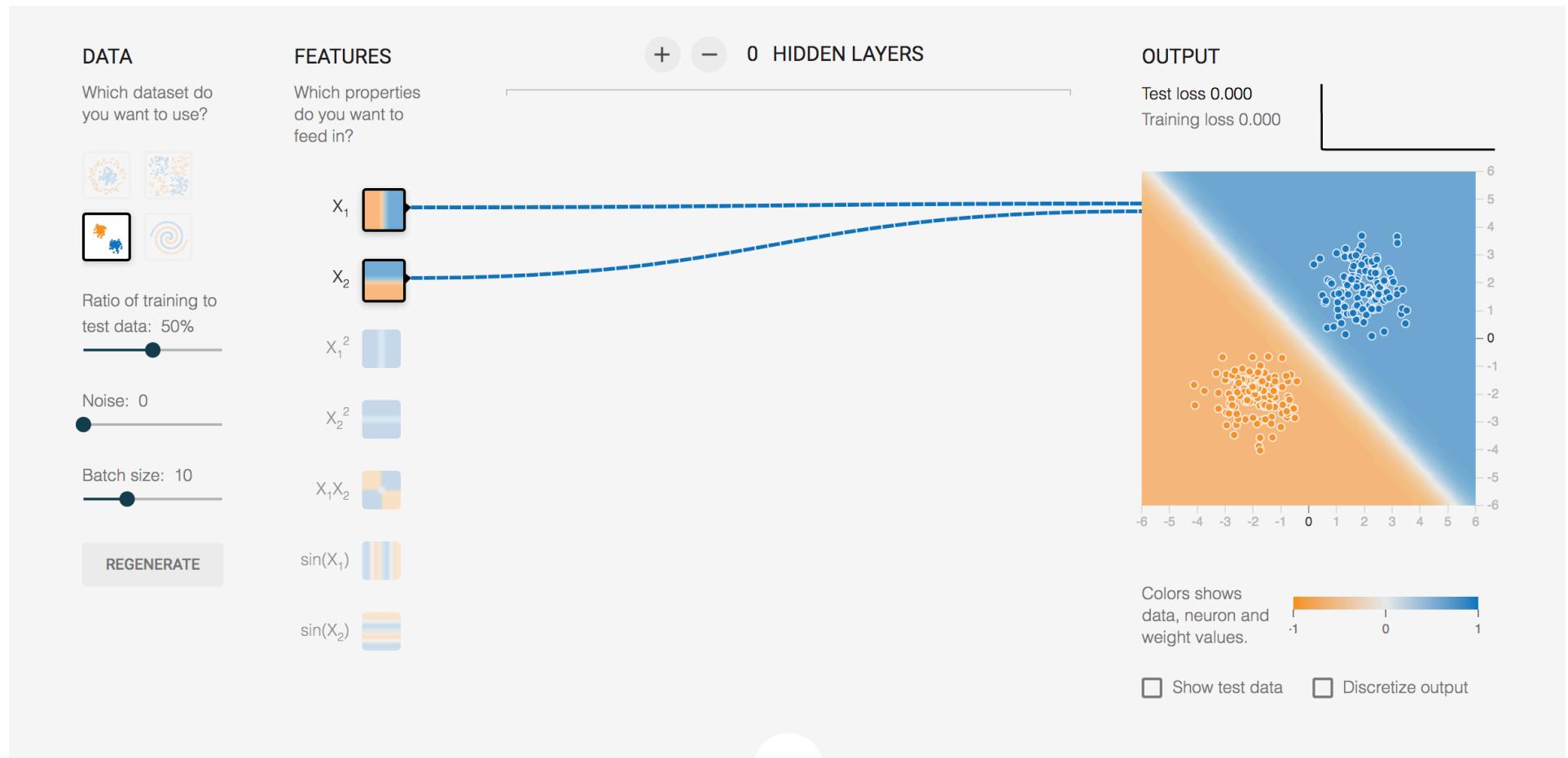
1. LINEAR NEURON OR UNIT
2. NON LINEARITIES (ACTIVATION FUNCTION)
3. LOSS FUNCTION



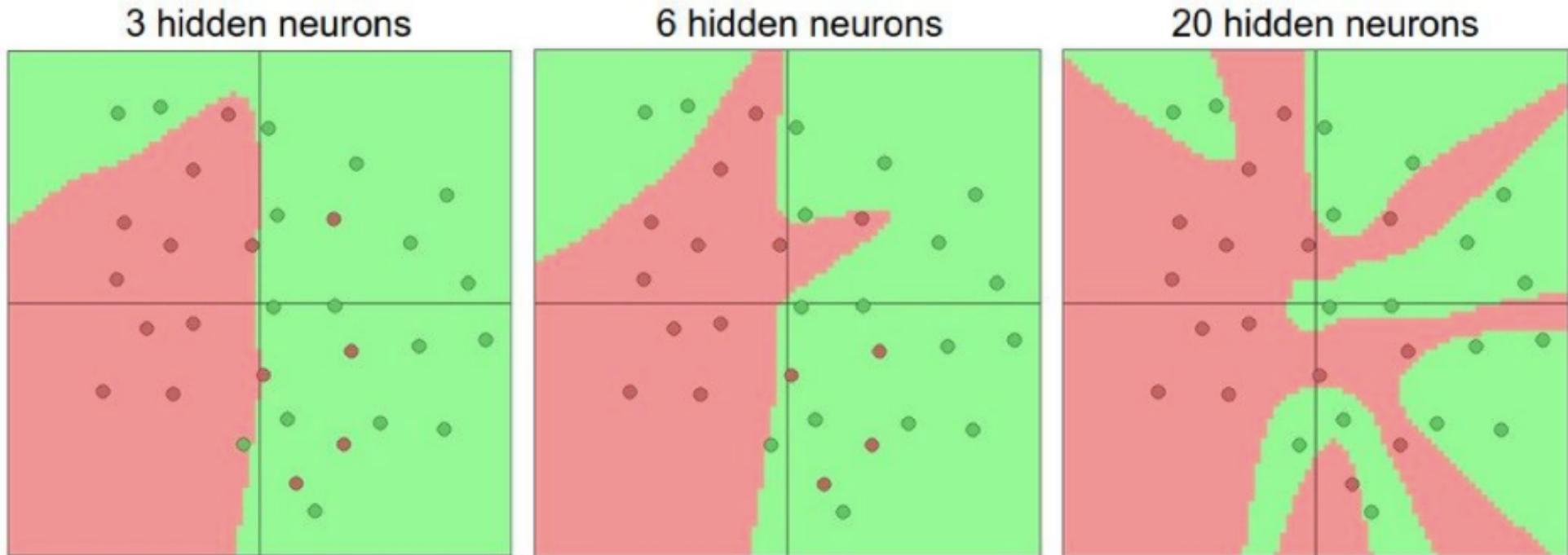
(deepmind
@Czarnecki)

LET'S SEE WHAT WE CAN DO WITH THIS SIMPLE MODEL!

<https://playground.tensorflow.org/>



HIDDEN LAYERS ALLOW INCREASING COMPLEXITY



More complex functions allow increasing complexity

Credit: Karpathy

UNIVERSAL APPROXIMATION THEOREM

FOR ANY CONTINUOUS FUNCTION FOR A HYPERCUBE $[0,1]^d$ TO REAL NUMBERS, AND EVERY POSITIVE EPSILON, THERE EXISTS A SIGMOID BASED 1-HIDDEN LAYER NEURAL NETWORK THAT OBTAINS AT MOST EPSILON ERROR IN FUNCTIONAL SPACE

Cybenko+89

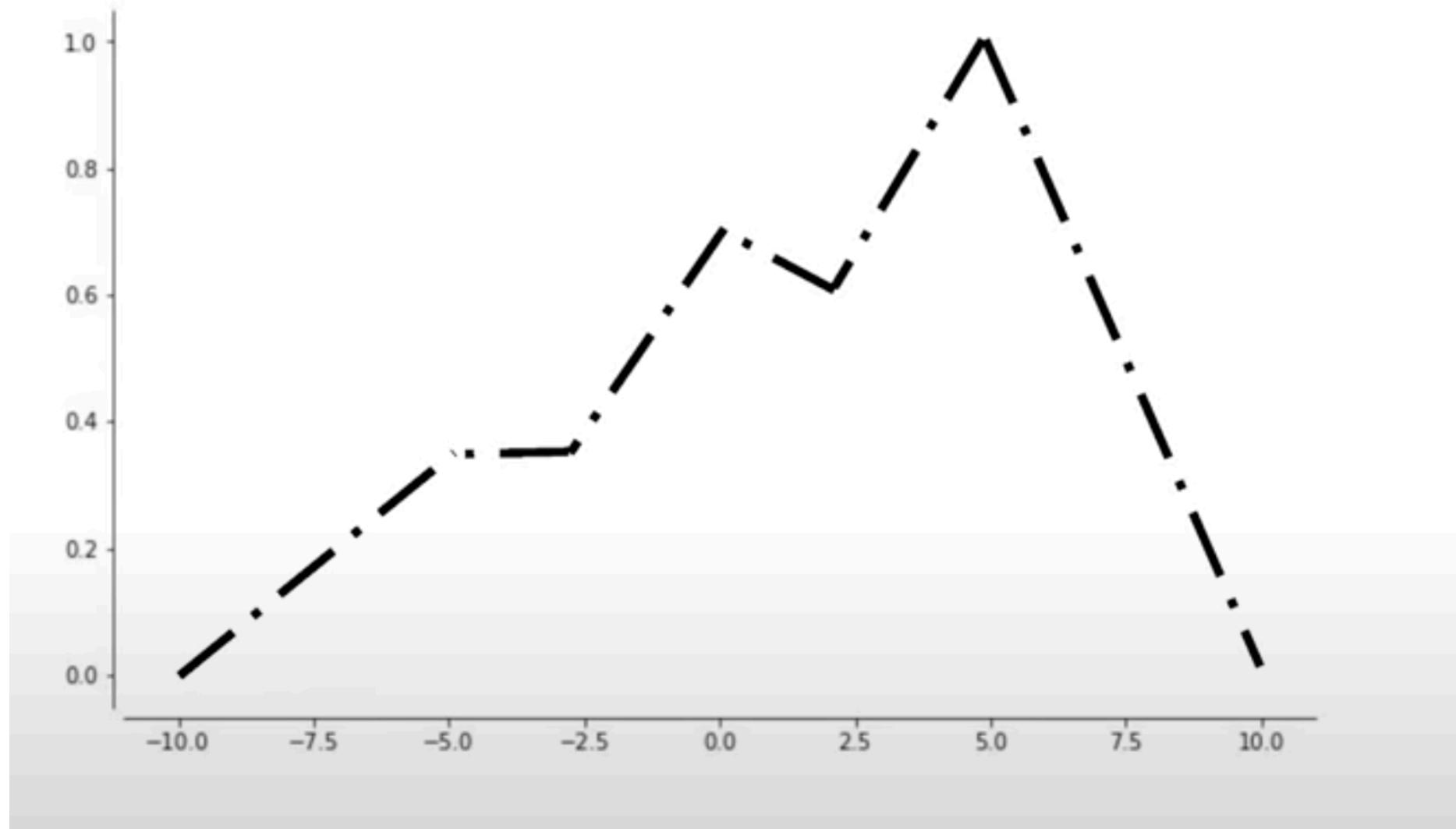
**“BIG ENOUGH NETWORK CAN APPROXIMATE,
BUT NOT REPRESENT ANY SMOOTH FUNCTION.
THE MATH DEMONSTRATION IMPLIES SHOWING
THAT NETWORKS ARE DENSE IN THE SPACE OF
TARGET FUNCTIONS”**

UNIVERSAL APPROXIMATION THEOREM

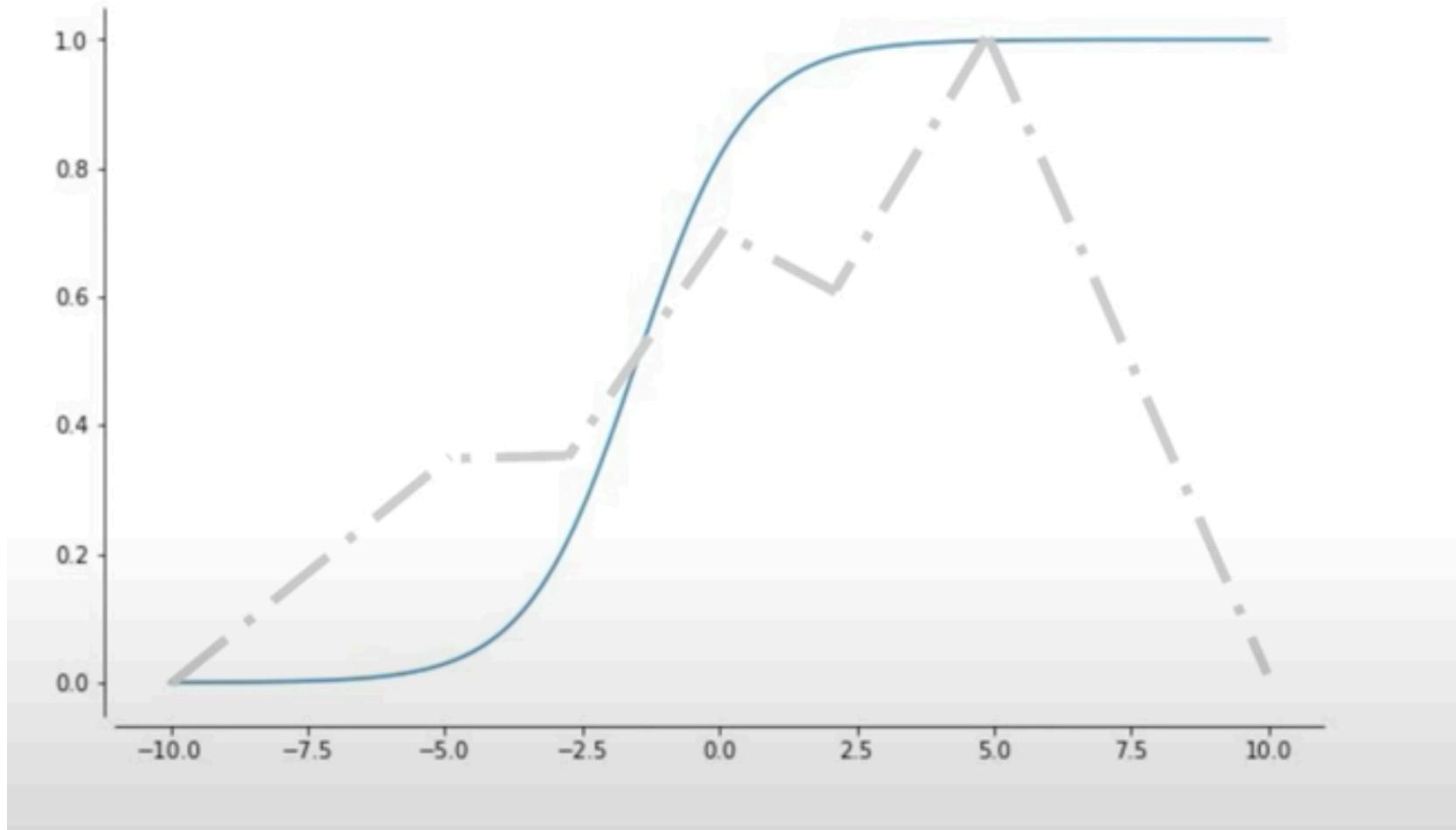
FOR ANY CONTINUOS FUNCTION FOR A HYPERCUBE $[0,1]^d$ TO REAL NUMBERS, NON-CONSTANT, BOUNDED AND CONTINUOUS ACTIVATION FUNCTION f , AND EVERY POSITIVE EPSILON, THERE EXISTS A 1-HIDDEN LAYER NEURAL NETWORK USING f THAT OBTAINES AT MOST EPSILON ERROR IN FUNCTIONAL SPACE

Horváth+91

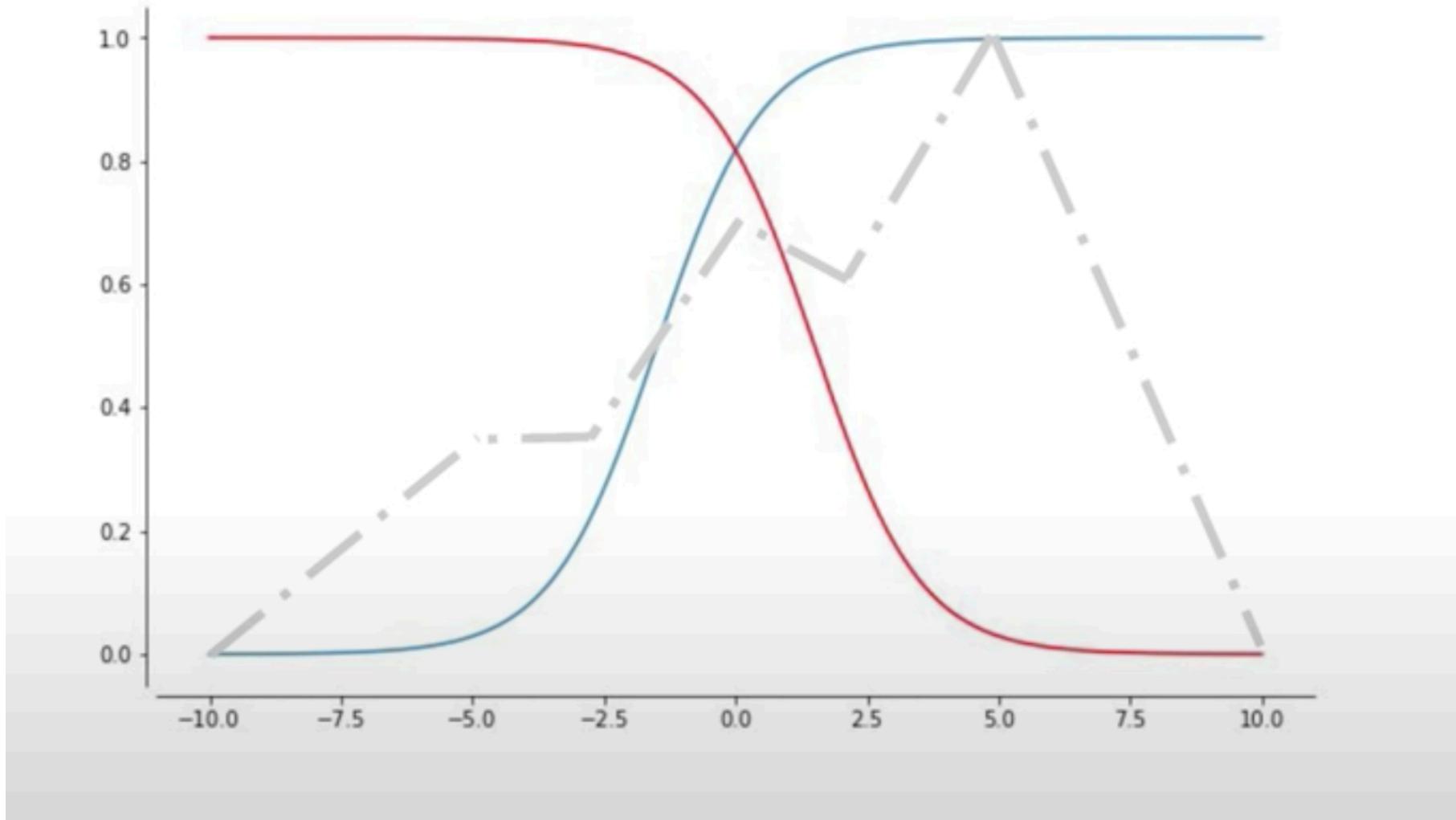
**“BIG ENOUGH NETWORK CAN APPROXIMATE,
BUT NOT REPRESENT ANY SMOOTH FUNCTION.
THE MATH DEMONSTRATION IMPLIES SHOWING
THAT NETWORS ARE DENSE IN THE SPACE OF
TARGET FUNCTIONS”**



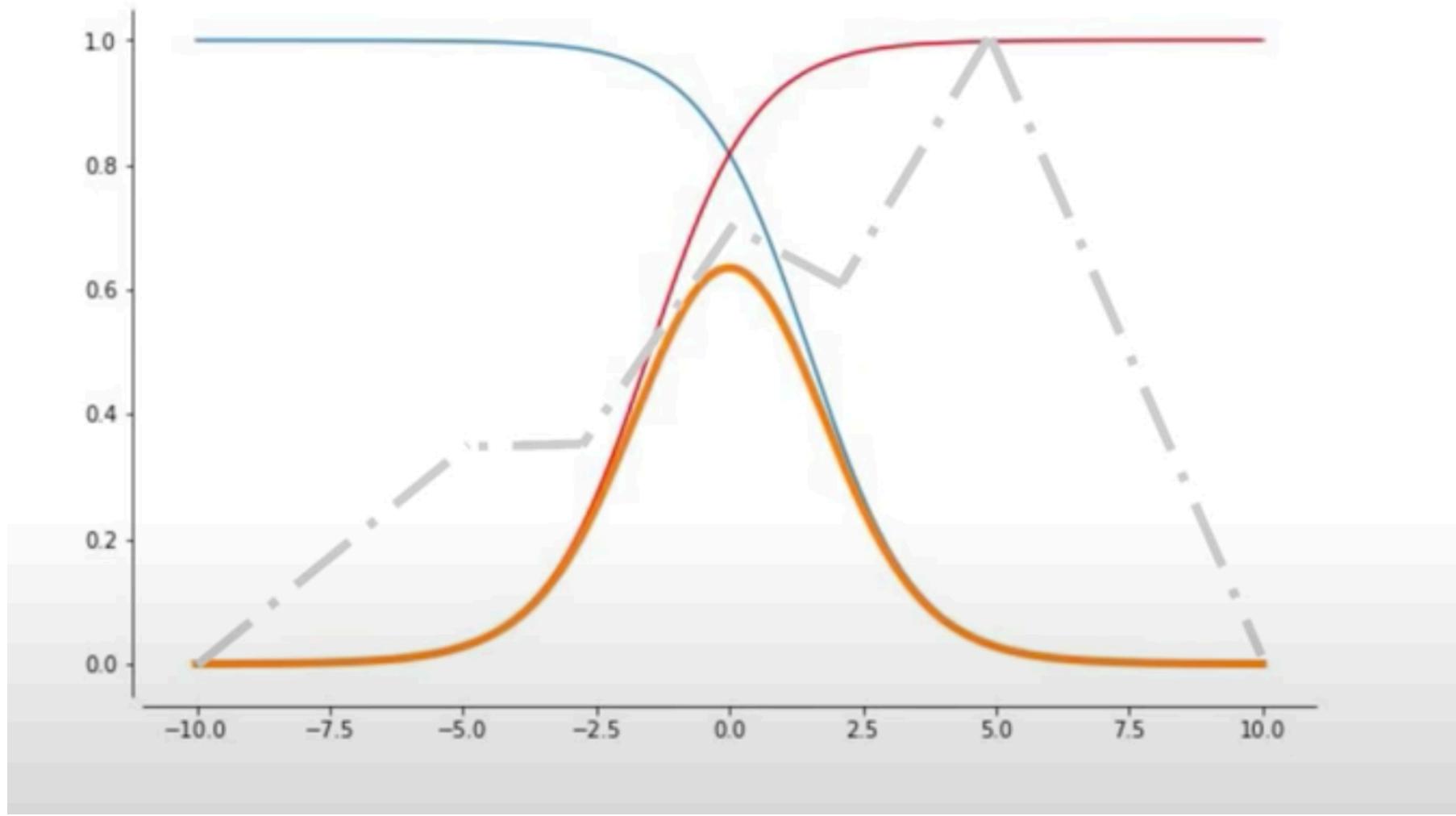
(deepmind
@Czarnecki)



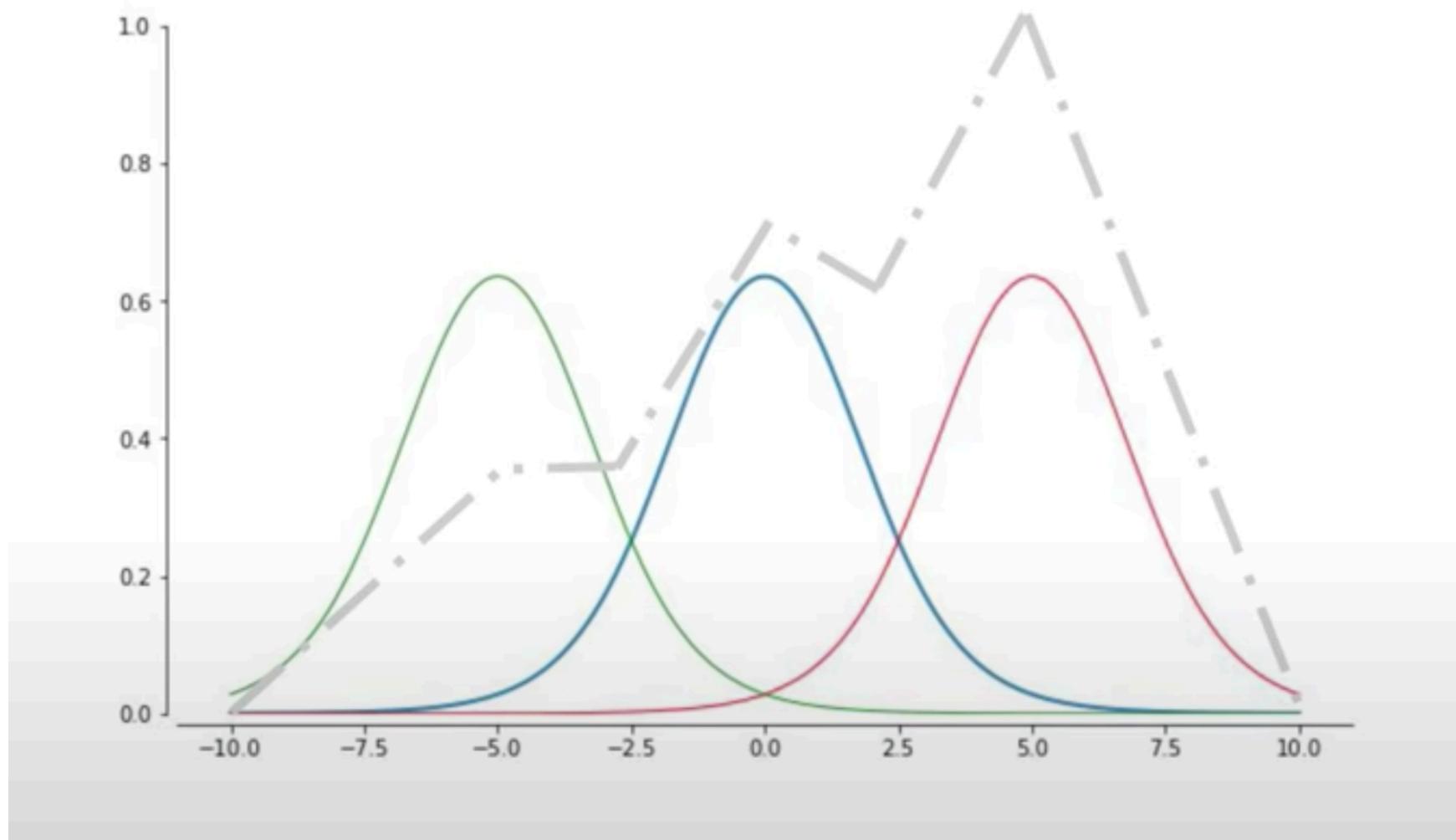
(deepmind
@Czarnecki)



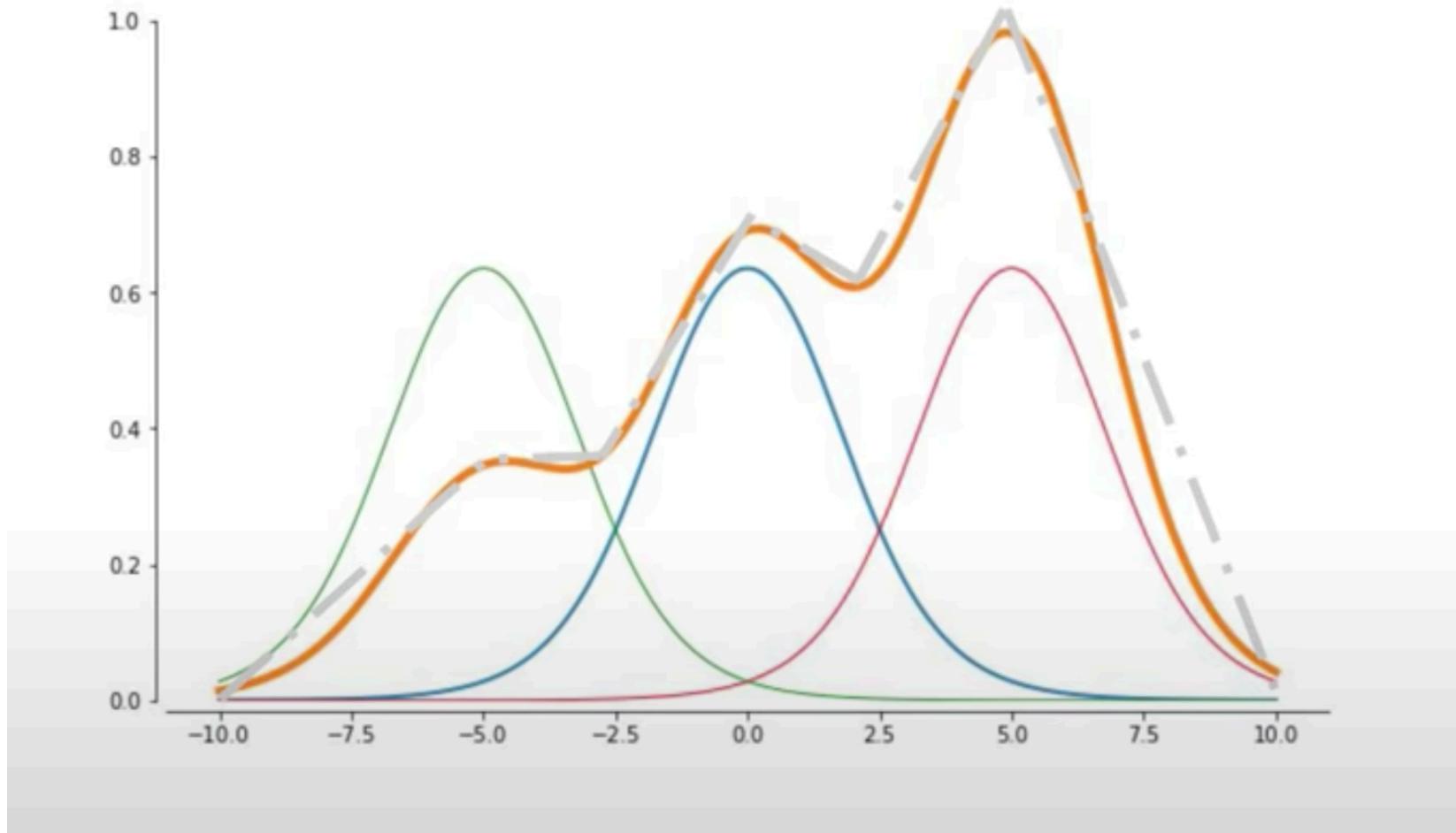
(deepmind
@Czarnecki)



(deepmind
@Czarnecki)

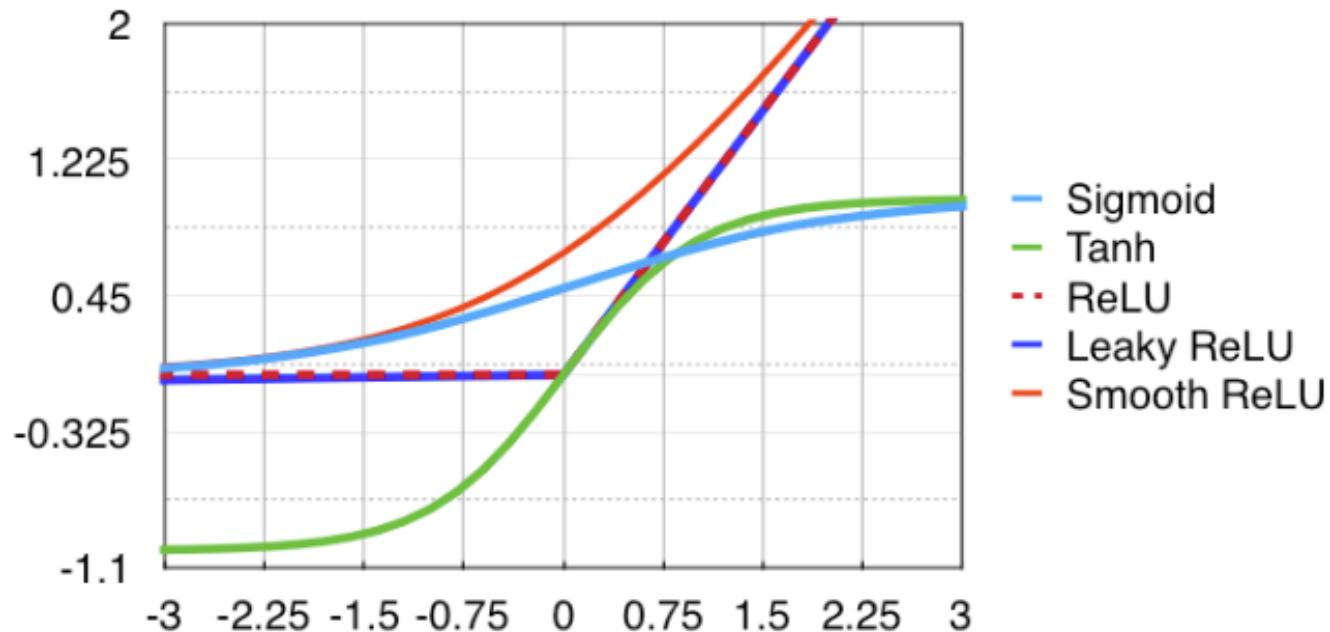


(deepmind
@Czarnecki)



(deepmind
@Czarnecki)

ACTIVATION FUNCTIONS



Sigmoid: $f(x) = \frac{1}{1 + e^{-x}}$

Tanh: $f(x) = \tanh(x)$

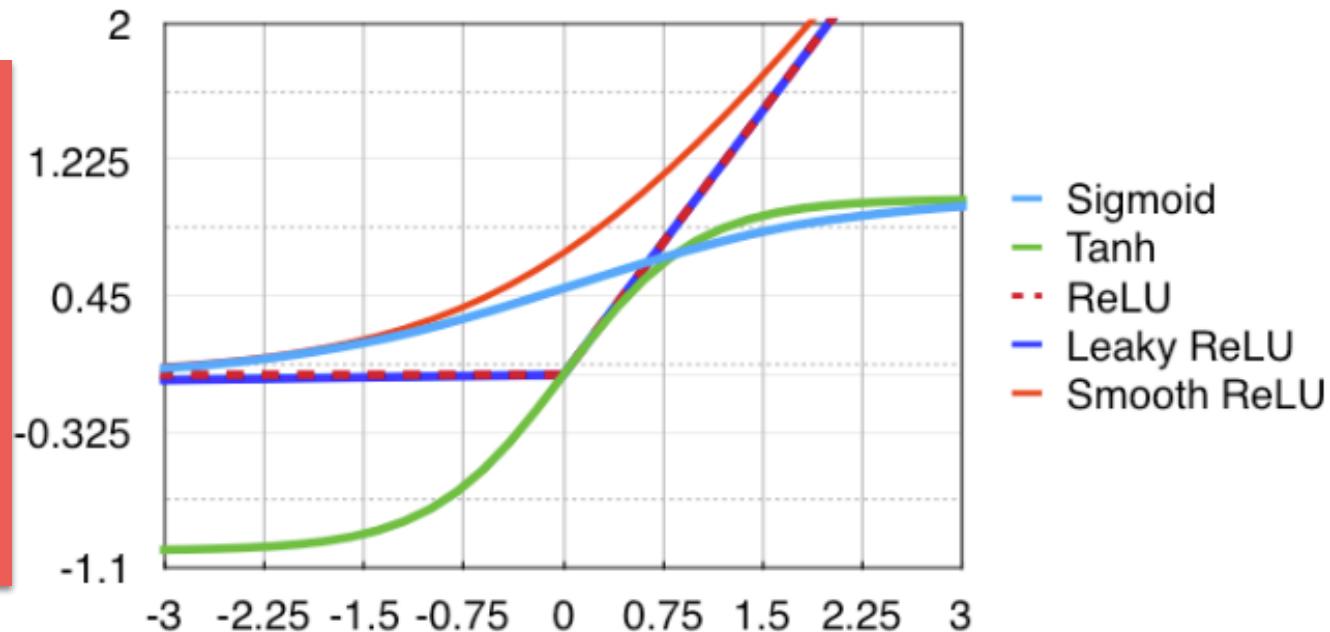
ReLU: $f(x) = \max(0, x)$

Soft ReLU: $f(x) = \log(1 + e^x)$

Leaky ReLU: $f(x) = \epsilon x + (1 - \epsilon)\max(0, x)$

ACTIVATION FUNCTIONS

+
**MANY
OTHERS!**



Sigmoid: $f(x) = \frac{1}{1 + e^{-x}}$

Tanh: $f(x) = \tanh(x)$

ReLU: $f(x) = \max(0, x)$

Soft ReLU: $f(x) = \log(1 + e^x)$

Leaky ReLU: $f(x) = \epsilon x + (1 - \epsilon) \max(0, x)$

SOFTMAX

A generalization of the SIGMOID ACTIVATION

$$\text{softmax}(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\sum_{i=1}^n e^{x_i}}$$

THE OUTPUT IS NORMALIZED BETWEEN 0 AND 1

THE COMPONENTS ADD TO 1

CAN BE INTERPRETED AS A PROBABILITY

$$p(Y = c | X = \mathbf{x}) = \text{softmax}(z(\mathbf{x}))_c$$

SO LET'S GO DEEPER AND DEEPER!

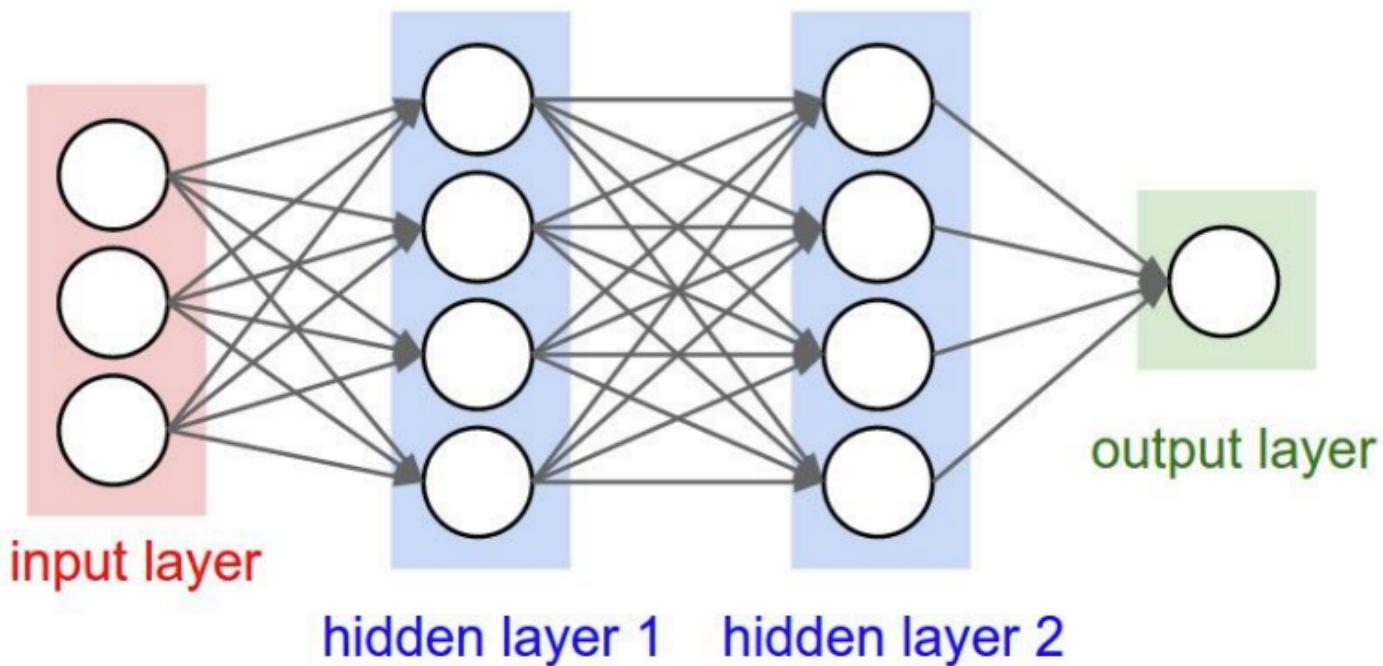
YES BUT...

NOT SO STRAIGHTFORWARD, DEEPER MEANS MORE
WEIGHTS, MORE DIFFICULT OPTIMIZATION, RISK OF
OVERFITTING...

**OK, SO NOW LET'S FIND
THE WEIGHTS**

OPTIMIZATION

[OR HOW TO FIND THE WEIGHTS?]

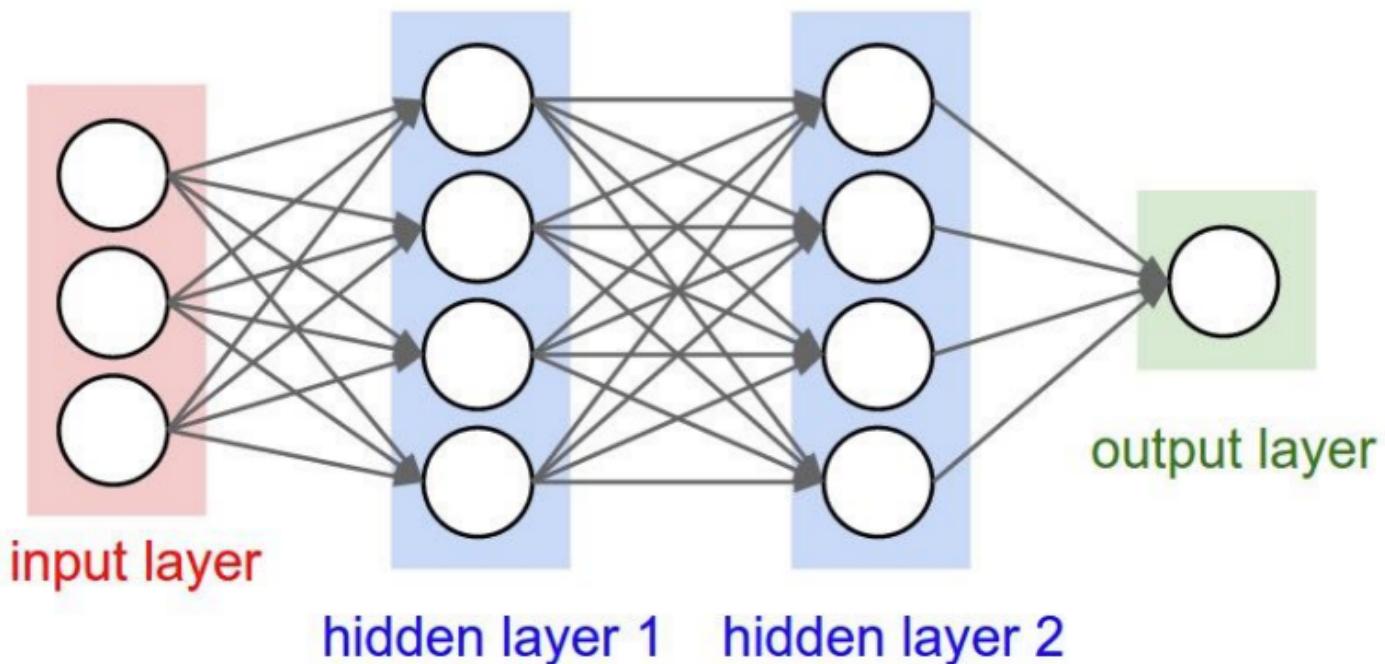


$$p = g_3(W_3g_2(W_2g_1(W_1\vec{x}_0)))$$

← NETWORK
FUNCTION

OPTIMIZATION

[OR HOW TO FIND THE WEIGHTS?]



$$p = g_3(W_3g_2(W_2g_1(W_1\vec{x}_0)))$$

$$\frac{1}{N} \sum_{i=1}^N (y_i - p_i)^2$$

←
LOSS
FUNCTION

WE SIMPLY WANT TO MINIMIZE THE LOSS FUNCTION WITH
RESPECT TO THE WEIGHTS, i.e. FIND THE WEIGHTS THAT
GENERATE THE MINIMUM LOSS

WE SIMPLY WANT TO MINIMIZE THE LOSS FUNCTION WITH
RESPECT TO THE WEIGHTS, i.e. FIND THE WEIGHTS THAT
GENERATE THE MINIMUM LOSS

WE THEN USE STANDARD MINIMIZATION ALGORITHMS
THAT YOU ALL KNOW...

FOR EXAMPLE....

Gradient Descent

$$W_{t+1} = W_t - \lambda_t \nabla f(W_t)$$



[gradient]

Newton

$$W_{t+1} = W_t - \lambda [Hf(W_t)]^{-1} \nabla f(W_t)$$



[hessian]

NEWTON CONVERGES FASTER...

FOR EXAMPLE....

Gradient Descent

$$W_{t+1} = W_t - \lambda_t \nabla f(W_t)$$



[gradient]

Newton

$$W_{t+1} = W_t - \lambda [Hf(W_t)]^{-1} \nabla f(W_t)$$



[hessian]

NEWTON CONVERGES FASTER...

BUT NEEDS THE HESSIAN

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial W_1^2} & \frac{\partial^2 f}{\partial W_1 \partial W_2} & \dots & \frac{\partial^2 f}{\partial W_1 \partial W_n} \\ \frac{\partial^2 f}{\partial W_2 \partial W_1} & \frac{\partial^2 f}{\partial W_2^2} & \dots & \frac{\partial^2 f}{\partial W_2 \partial W_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial W_n \partial W_1} & \frac{\partial^2 f}{\partial W_n \partial W_2} & \dots & \frac{\partial^2 f}{\partial W_n^2} \end{bmatrix}$$

FOR EXAMPLE....

Gradient Descent

$$W_{t+1} = W_t - \lambda_t \nabla f(W_t)$$



[gradient]

Newton

$$W_{t+1} = W_t - \lambda [Hf(W_t)]^{-1} \nabla f(W_t)$$

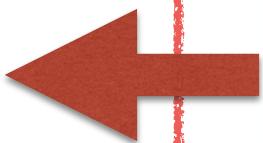


[hessian]

NEWTON CONVERGES FASTER...

BUT NEEDS THE HESSIAN

MOST USED BY FAR....



$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial W_1^2} & \frac{\partial^2 f}{\partial W_1 \partial W_2} & \dots & \frac{\partial^2 f}{\partial W_1 \partial W_n} \\ \frac{\partial^2 f}{\partial W_2 \partial W_1} & \frac{\partial^2 f}{\partial W_2^2} & \dots & \frac{\partial^2 f}{\partial W_2 \partial W_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial W_n \partial W_1} & \frac{\partial^2 f}{\partial W_n \partial W_2} & \dots & \frac{\partial^2 f}{\partial W_n^2} \end{bmatrix}$$

FOR EXAMPLE....

Gradient Descent

$$W_{t+1} = W_t - \lambda_t \nabla f(W_t)$$



[gradient]

EVERYTHING RELIES
ON COMPUTING THE GRADIENT

MOST USED BY FAR....

Newton

$$W_{t+1} = W_t - \lambda [Hf(W_t)]^{-1} \nabla f(W_t)$$



[hessian]

NEWTON CONVERGES FASTER...

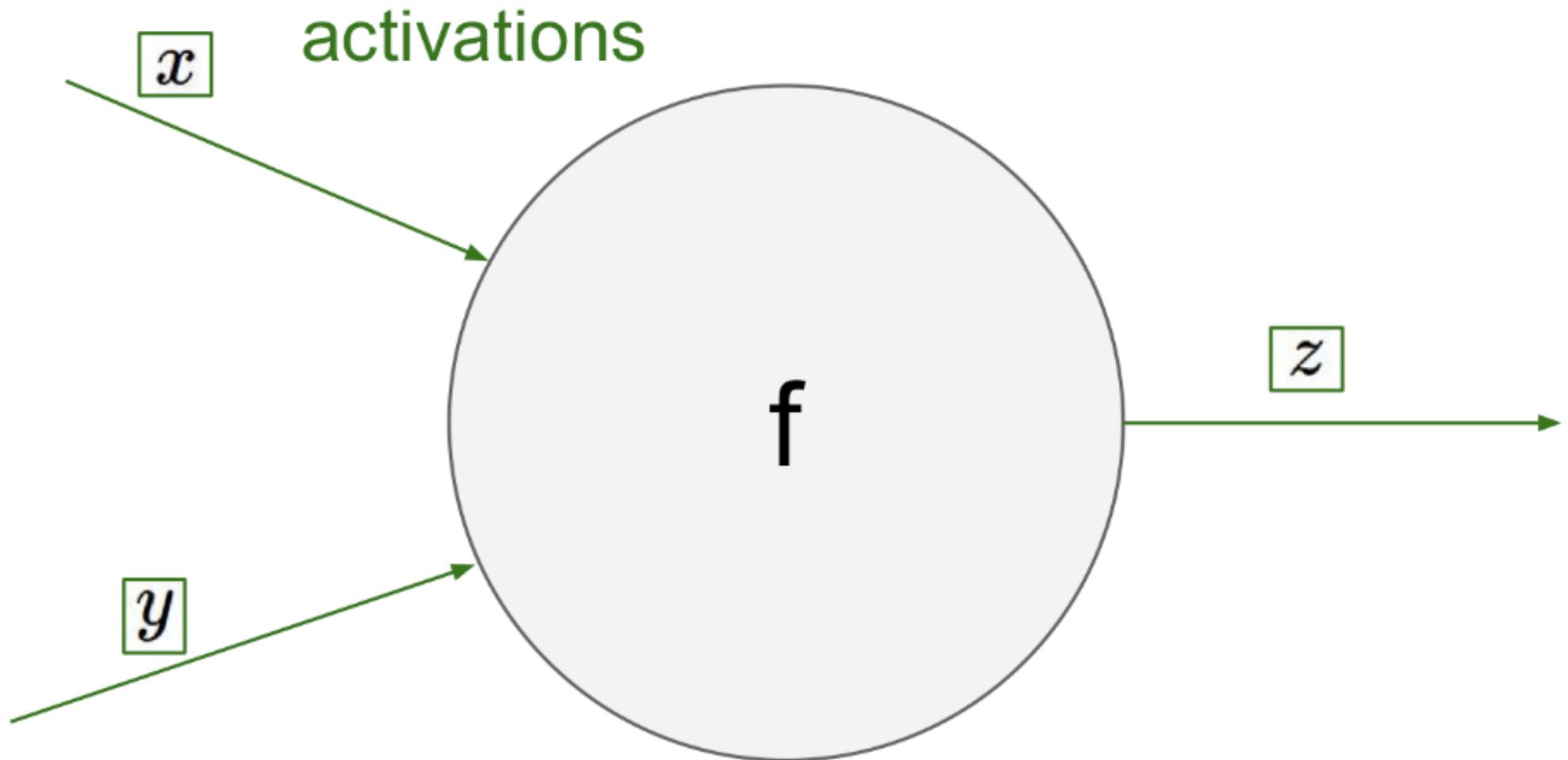
BUT NEEDS THE HESSIAN

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial W_1^2} & \frac{\partial^2 f}{\partial W_1 \partial W_2} & \dots & \frac{\partial^2 f}{\partial W_1 \partial W_n} \\ \frac{\partial^2 f}{\partial W_2 \partial W_1} & \frac{\partial^2 f}{\partial W_2^2} & \dots & \frac{\partial^2 f}{\partial W_2 \partial W_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial W_n \partial W_1} & \frac{\partial^2 f}{\partial W_n \partial W_2} & \dots & \frac{\partial^2 f}{\partial W_n^2} \end{bmatrix}$$

NICE, BUT I NEED TO COMPUTE THE
GRADIENT AT EVERY ITERATION OF
AN ARBITRARY COMPLEX FUNCTION!

BACKPROPAGATION

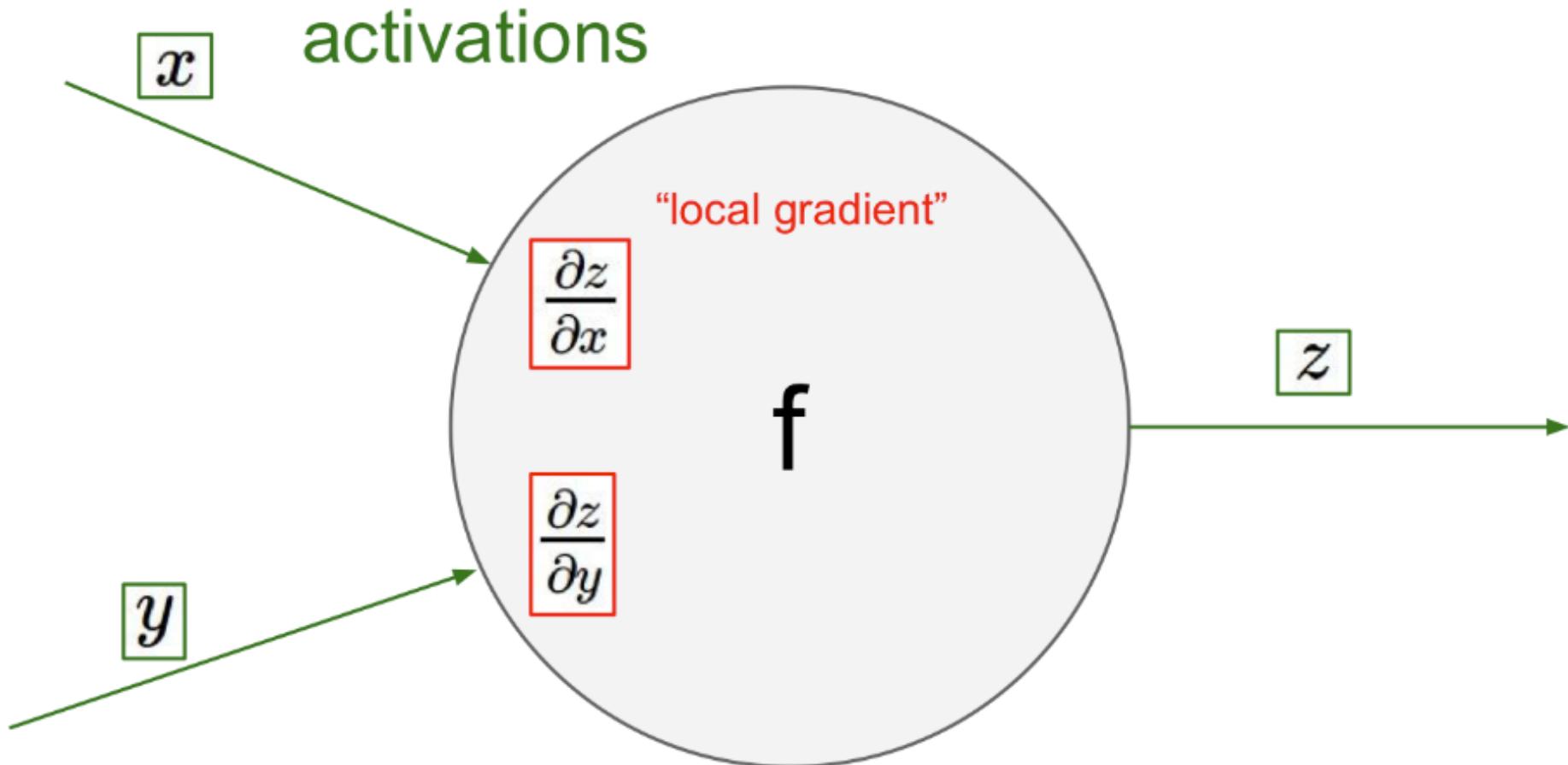
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

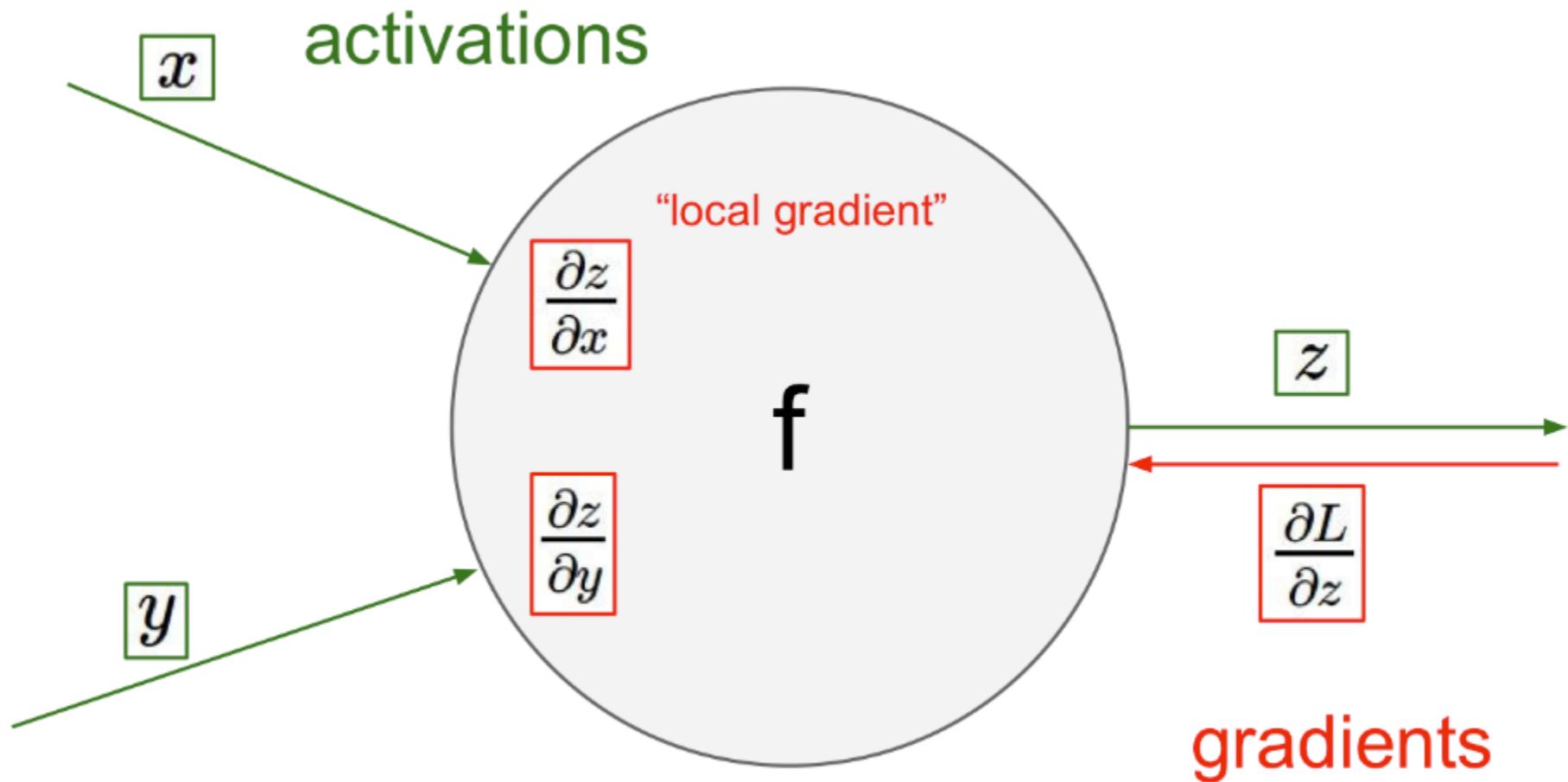
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

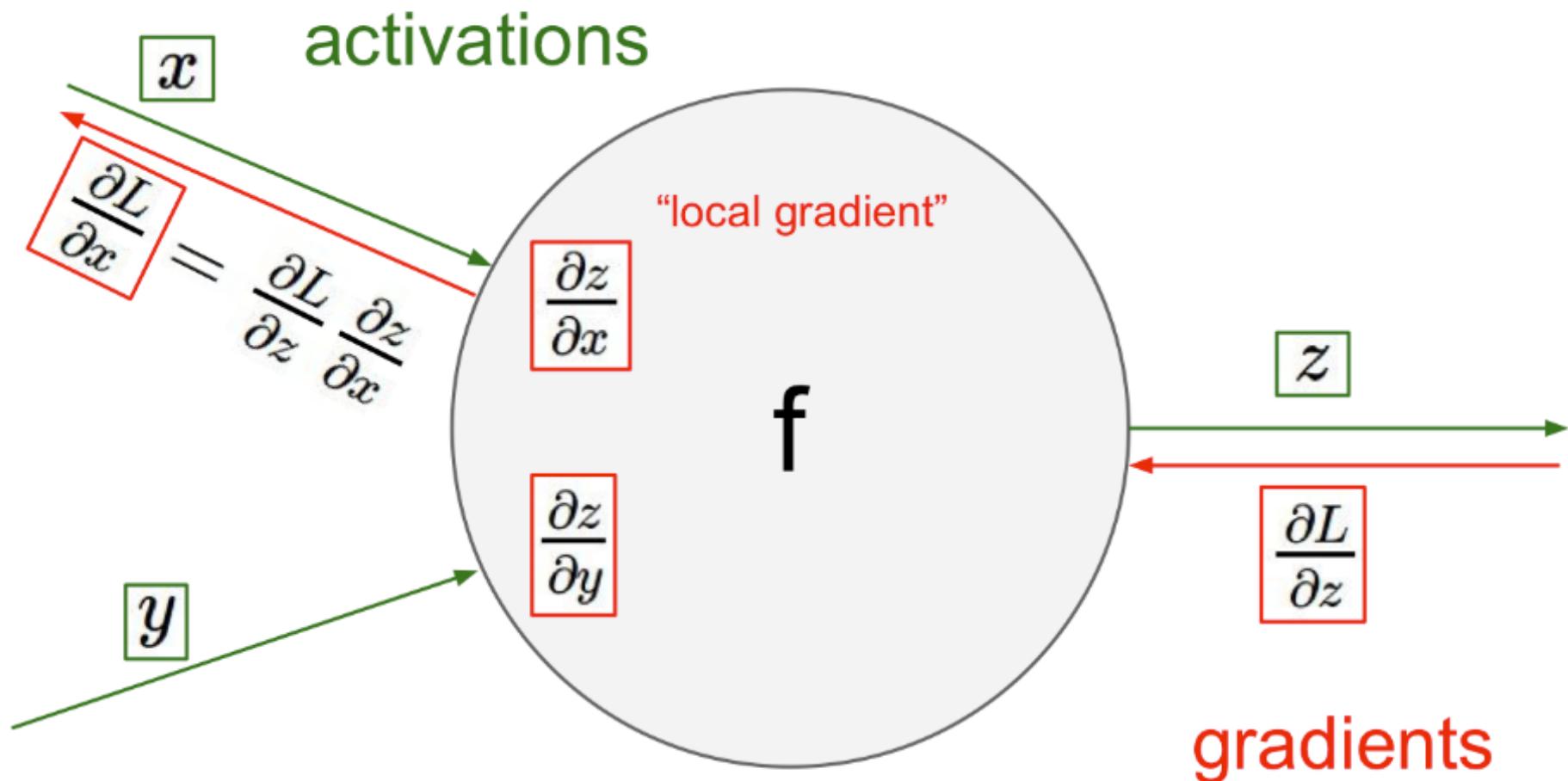
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

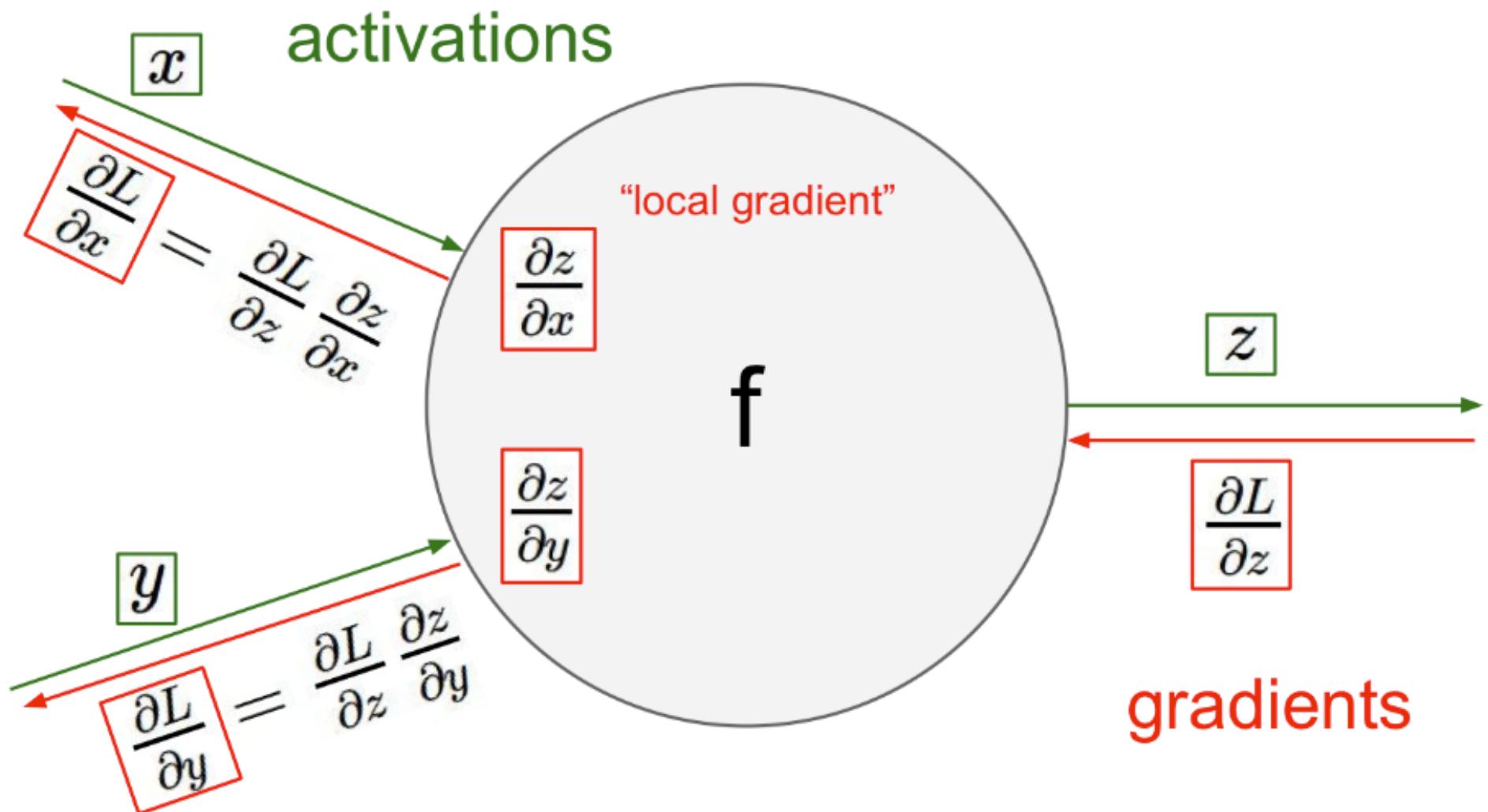
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

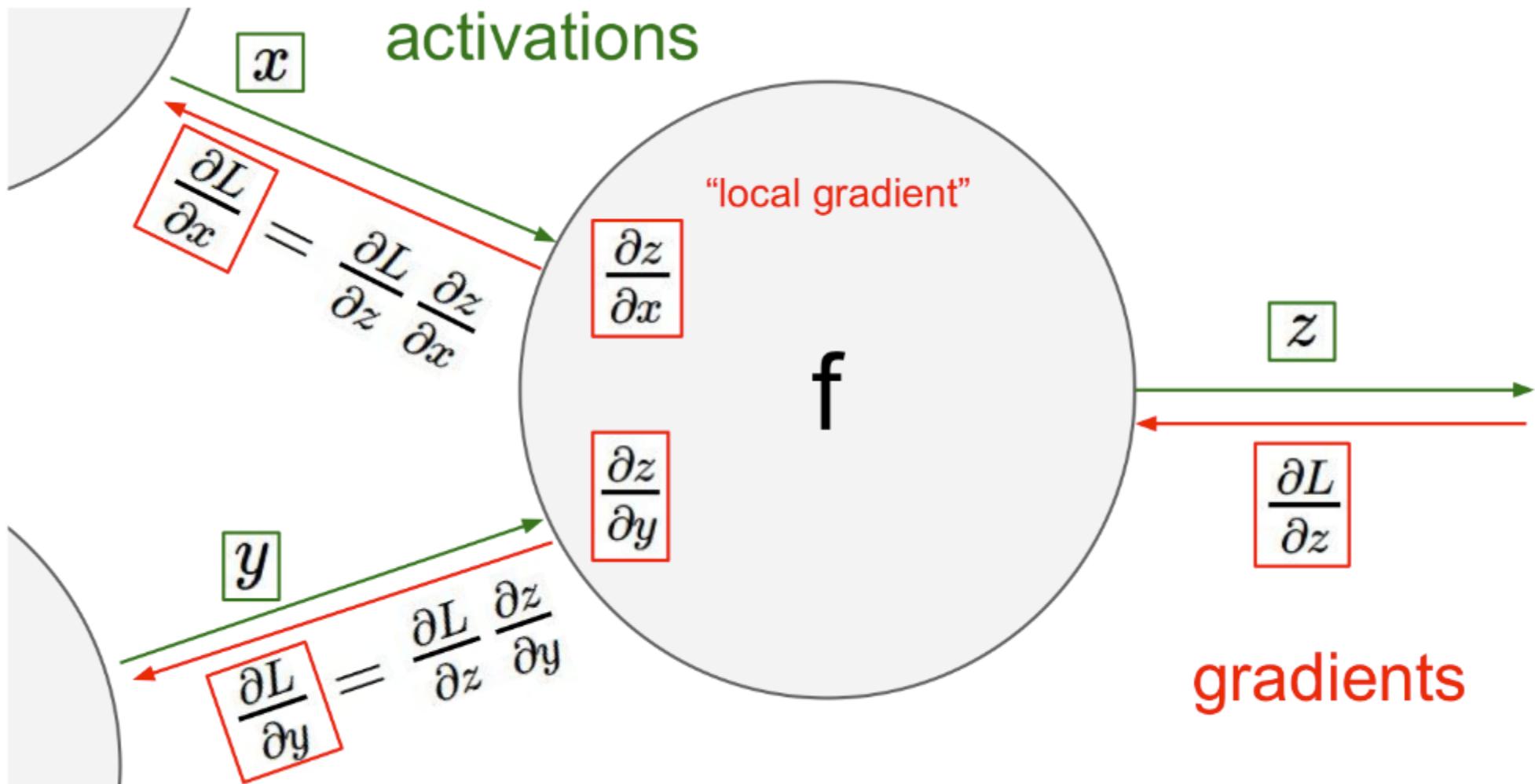
[AT THE NEURON LEVEL]



Credit: A. Karpathy

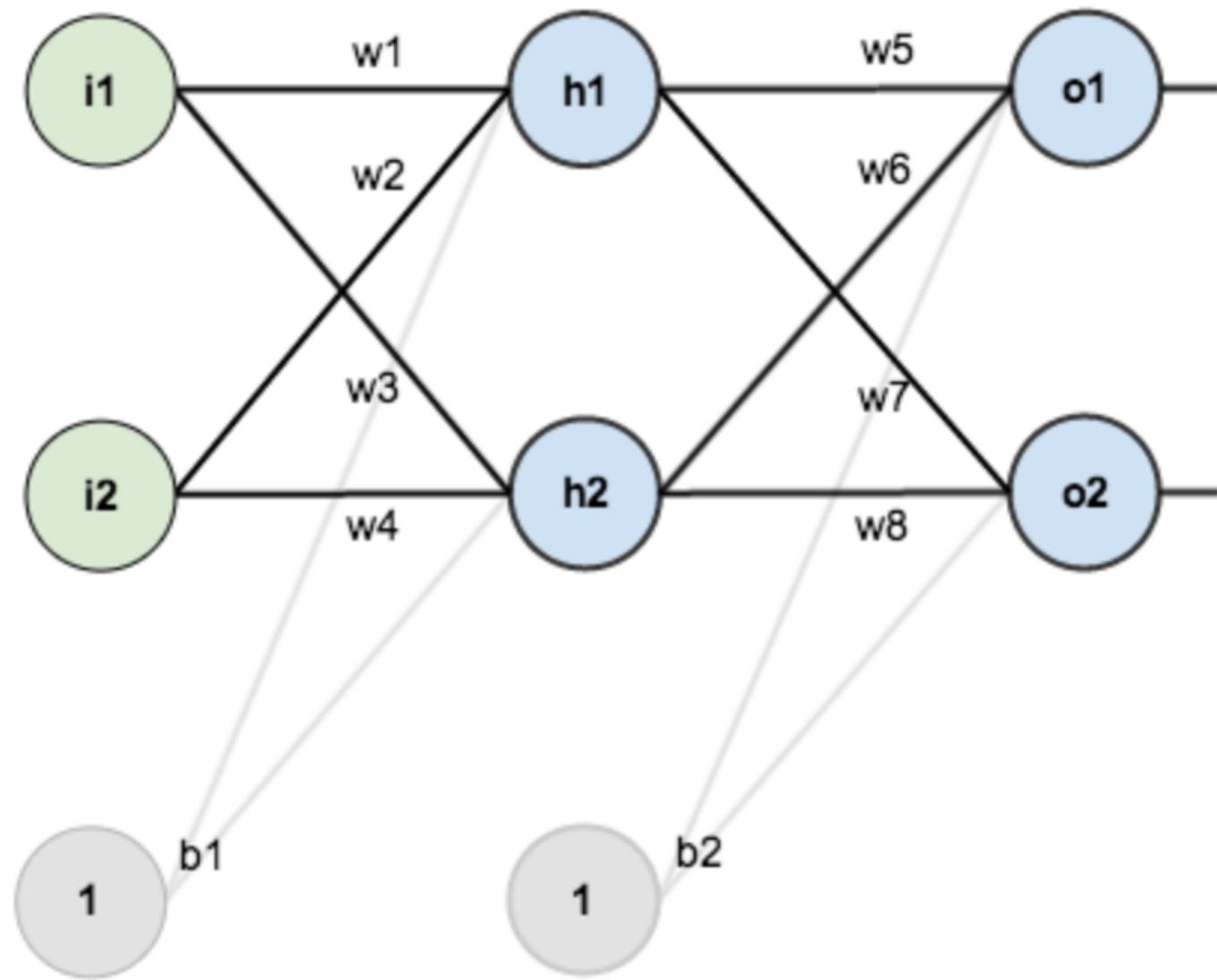
BACKPROPAGATION

[AT THE NEURON LEVEL]

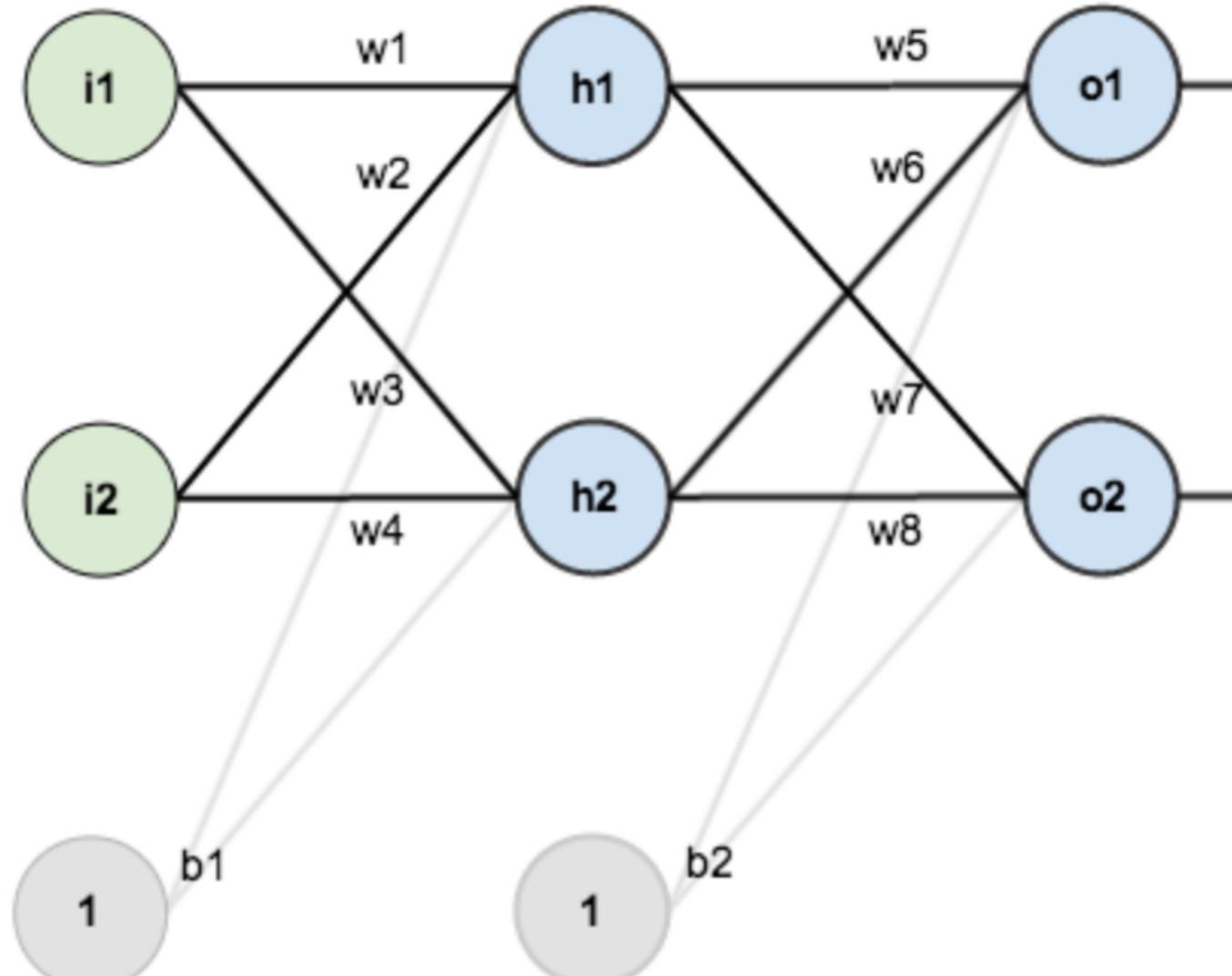


Credit: A. Karpathy

**LET'S FOLLOW A NETWORK
WHILE IT LEARNS...**

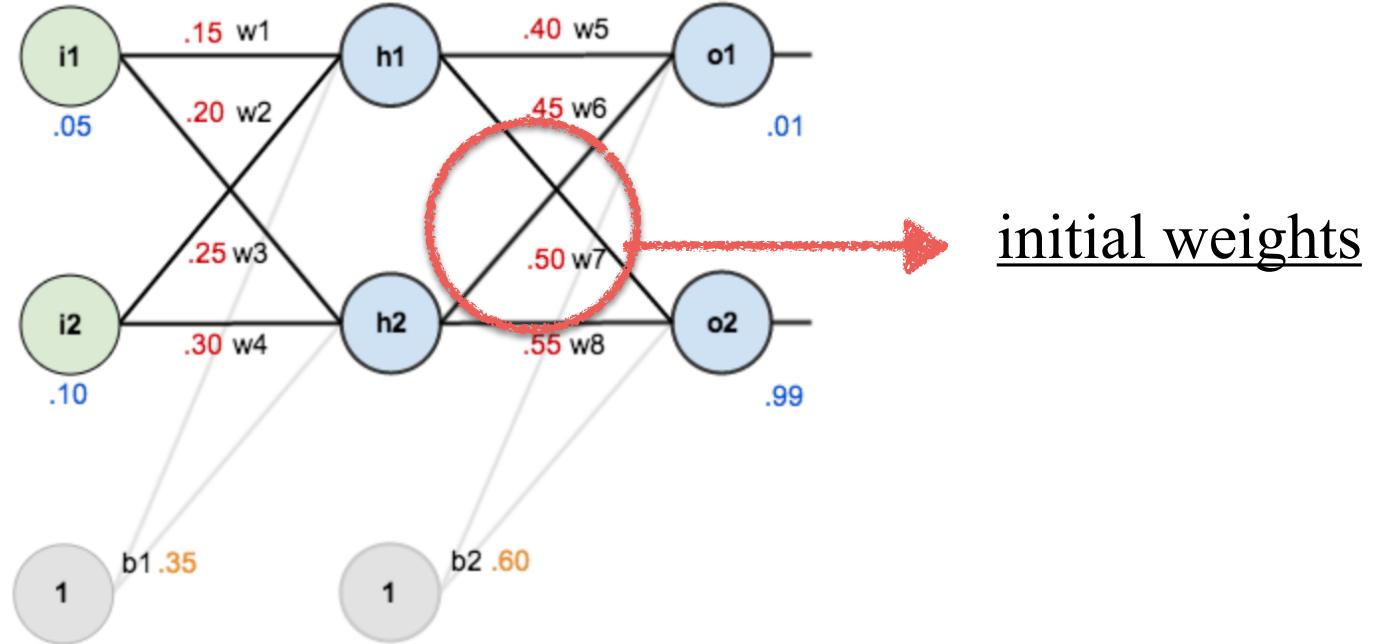


EXAMPLE TAKEN FROM HERE



LET'S ASSUME A VERY SIMPLE TRAINING SET:
 $X=(0.05, 0.10) \rightarrow Y=(0.01, 0.99)$

EXAMPLE TAKEN FROM HERE

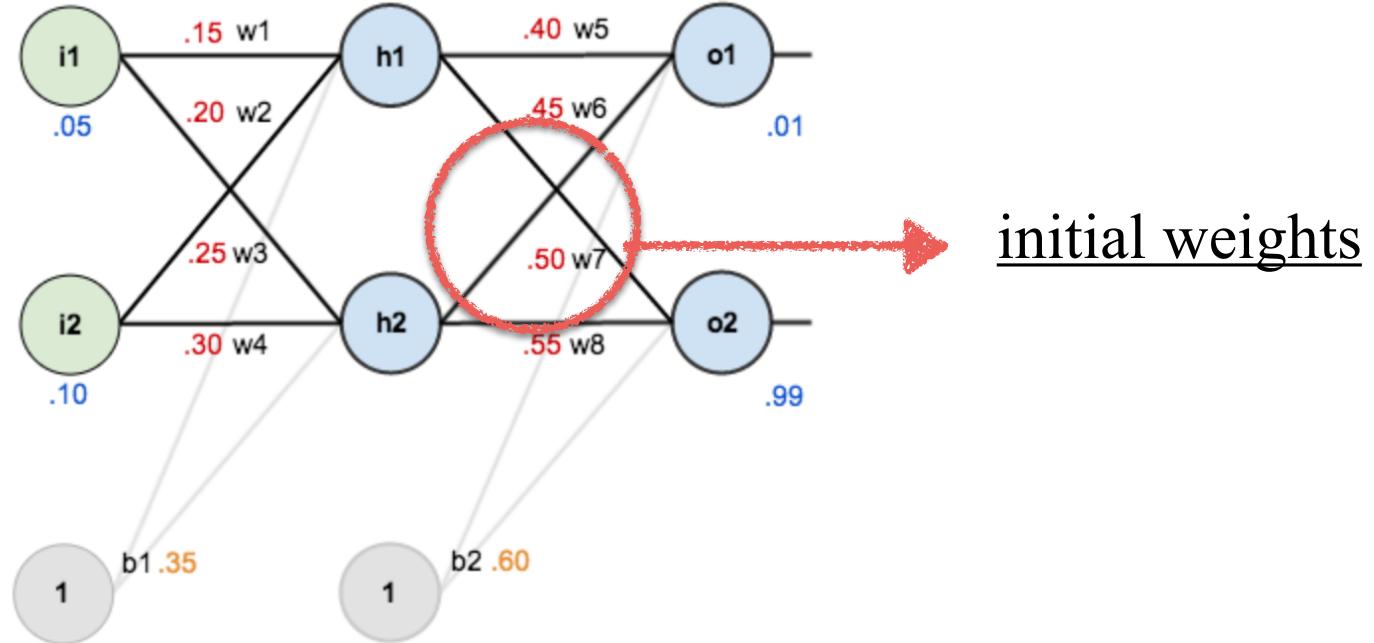


1. THE FORWARD PASS

$$in_{h1} = w_1 i_1 + w_2 i_2 + b_1$$

$$in_{h1} = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$$

[with initial weights]



1. THE FORWARD PASS

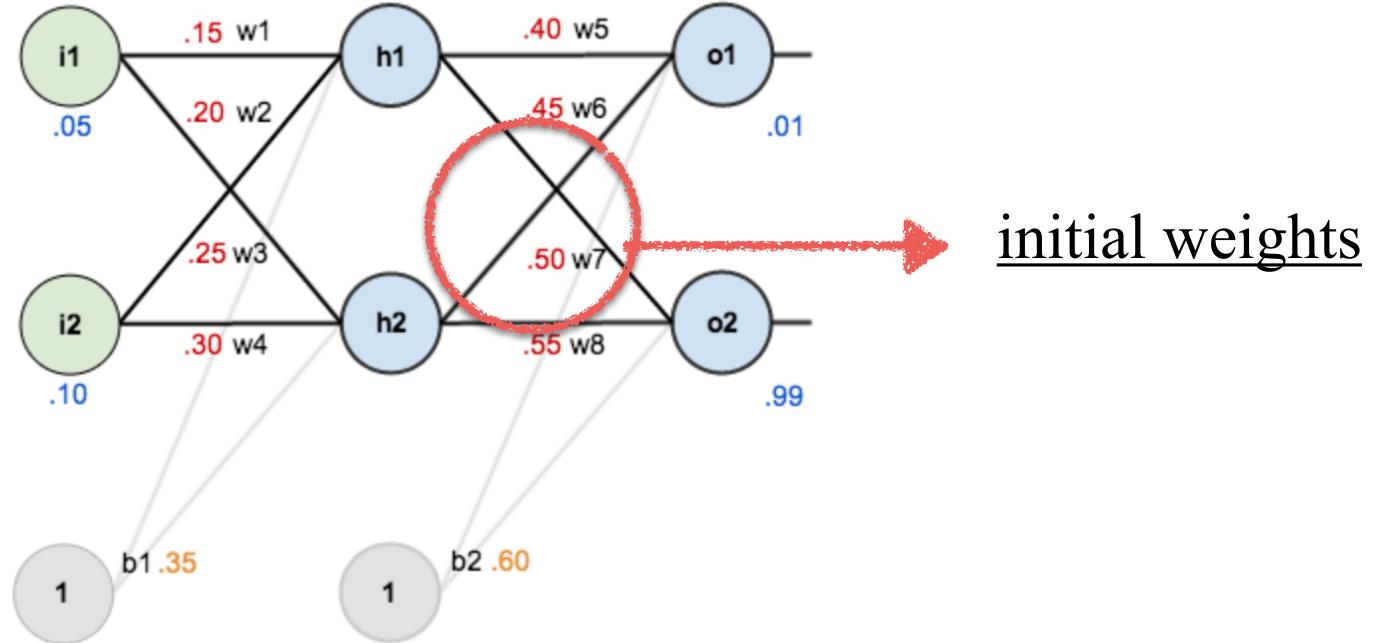
$$in_{h1} = w_1 i_1 + w_2 i_2 + b_1$$

$$in_{h1} = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$$

[with initial weights]

$$out_{h1} = \frac{1}{1 + e^{-in_{h1}}} = 0.5932$$

[after the activation function]



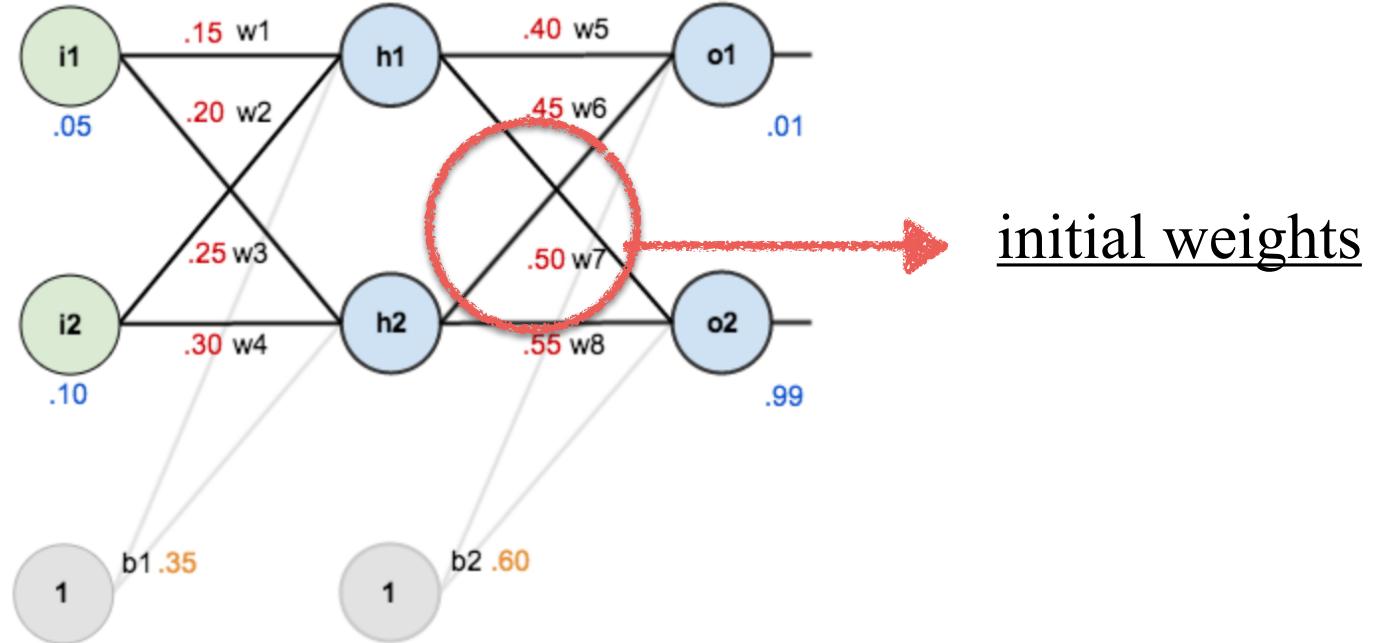
1. THE FORWARD PASS

WE CONTINUE TO o_1

$$in_{o1} = w_5 out_{h1} + w_6 out_{h2} + b_2$$

$$in_{o1} = 0.4 \times 0.593 + 0.45 \times 0.596 + 0.6 = 1.105$$

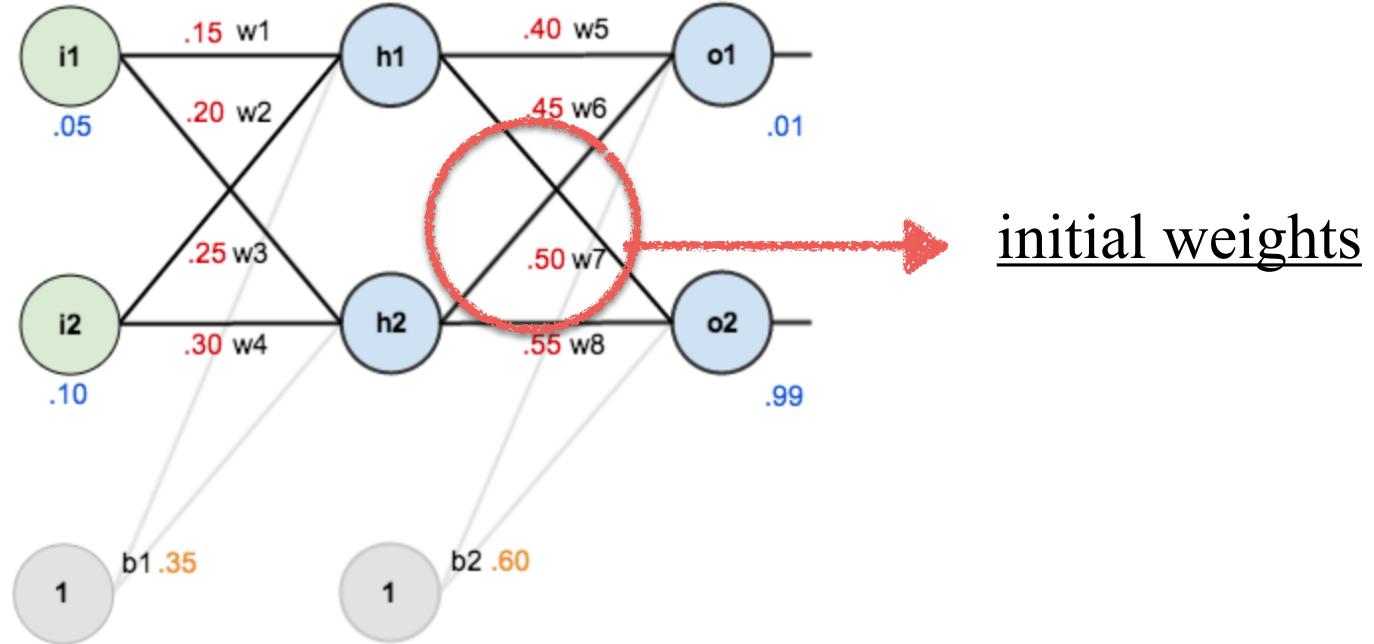
$$out_{o1} = \frac{1}{1 + e^{-1.105}} = 0.751$$



1. THE FORWARD PASS

AND THE SAME FOR o_2

$$out_{o2} = 0.7729$$

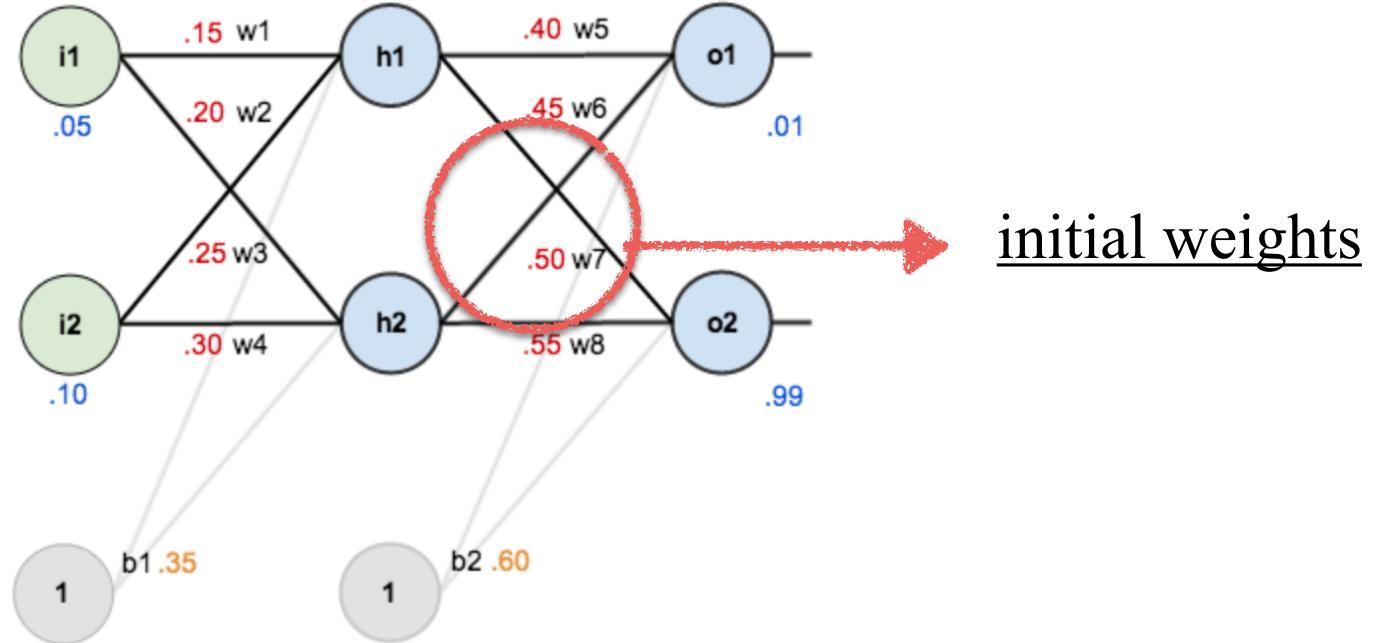


2. THE LOSS FUNCTION

$$L_{total} = \sum 0.5(target - output)^2$$

$$L_{o1} = 0.5(target_{o1} - output_{o1})^2 = 0.5 \times (0.01 - 0.751)^2 = 0.274$$

$$L_{o2} = 0.023$$

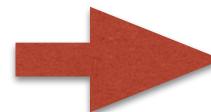


2. THE LOSS FUNCTION

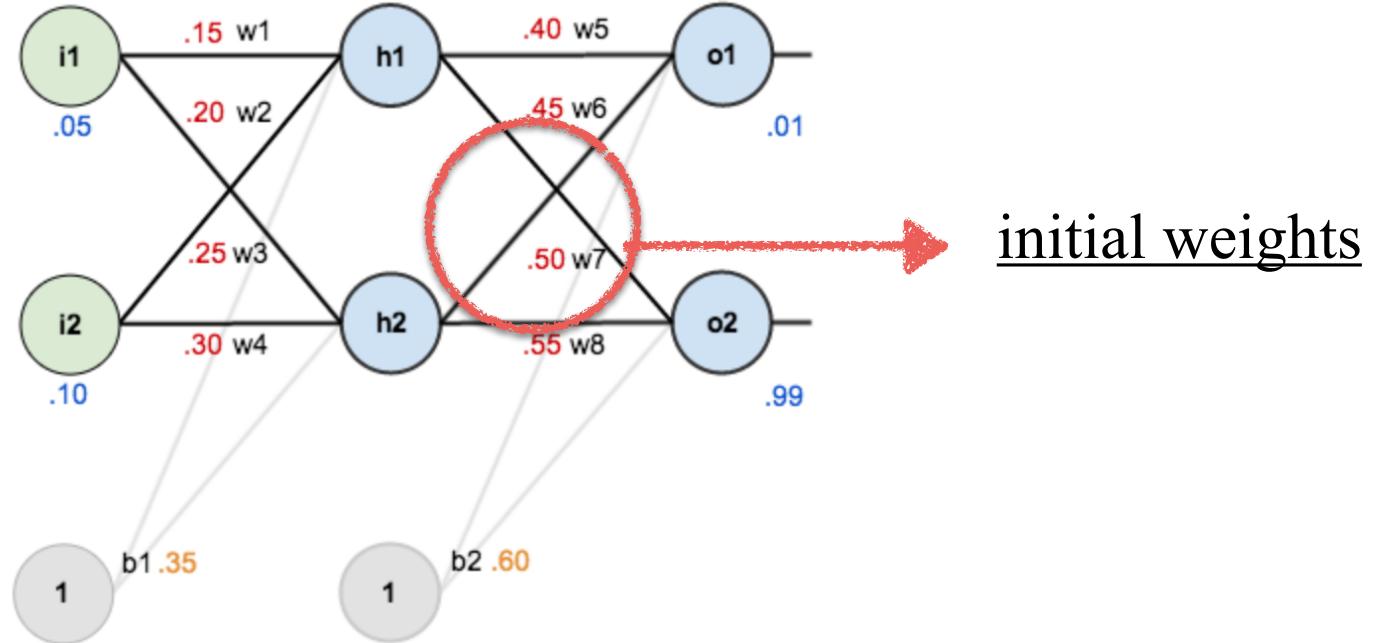
$$L_{total} = \sum 0.5(target - output)^2$$

$$L_{o1} = 0.5(target_{o1} - output_{o1})^2 = 0.5 \times (0.01 - 0.751)^2 = 0.274$$

$$L_{o2} = 0.023$$



$$L_{total} = L_{o1} + L_{o2} = 0.298$$



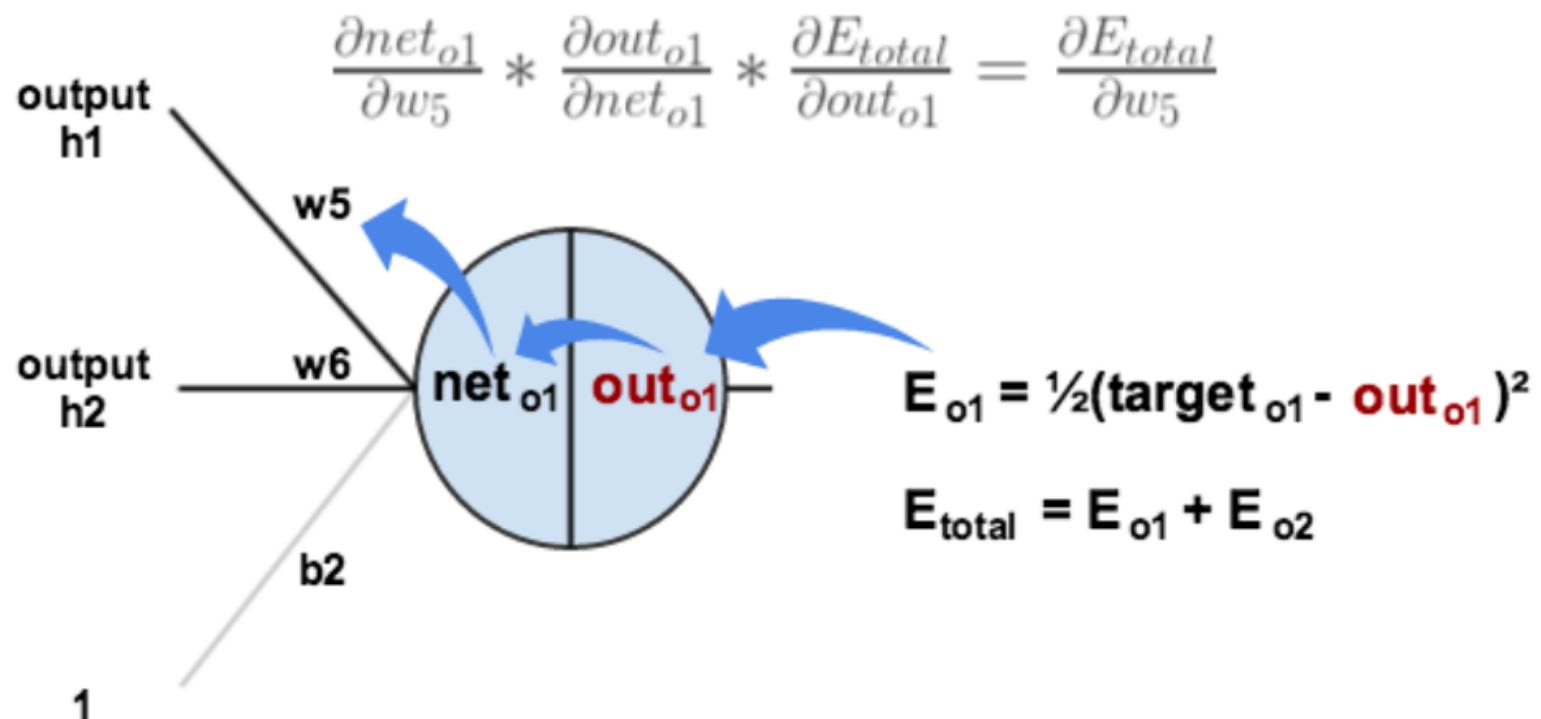
3. THE BACKWARD PASS

FOR W5

WE WANT:

$$\frac{\partial L_{total}}{\partial w_5}$$

[gradient of loss function]



3. THE BACKWARD PASS

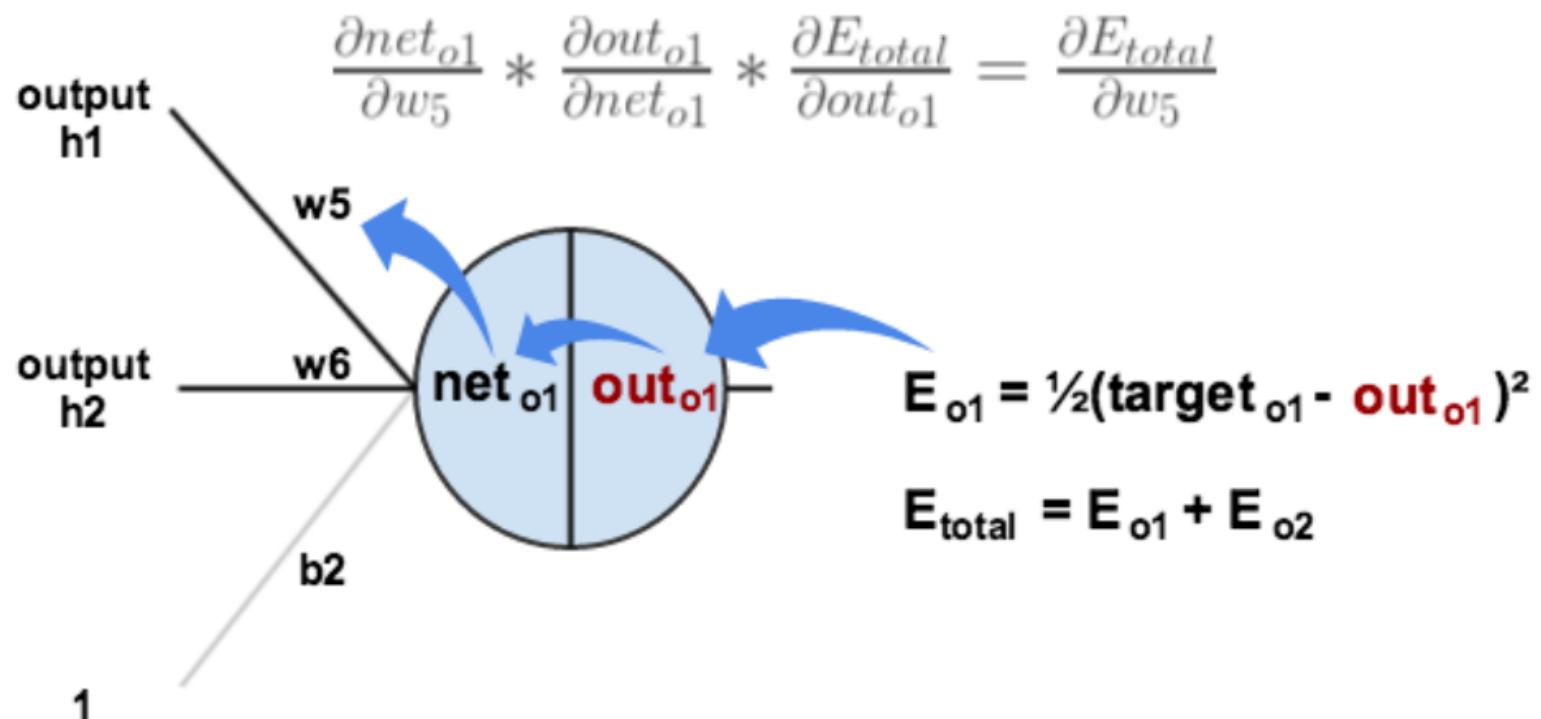
FOR w_5

WE WANT:

$$\frac{\partial L_{total}}{\partial w_5} \quad [\text{gradient of loss function}]$$

WE APPLY THE CHAIN RULE:

$$\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}$$

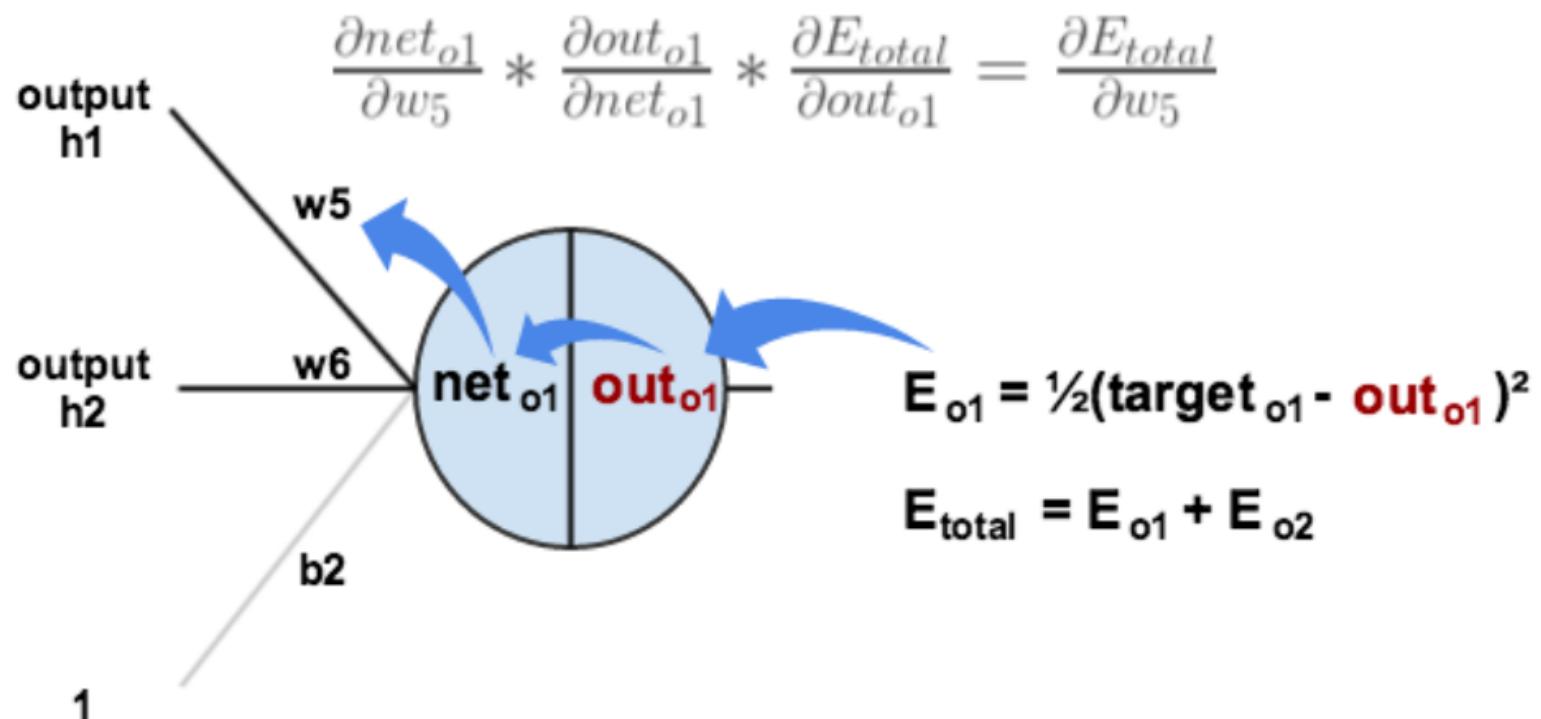


3. THE BACKWARD PASS

$$\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}$$

$$L_{total} = 0.5(target_{o1} - out_{o1})^2 + 0.5(target_{o2} - out_{o2})^2$$

$$\frac{\partial L_{total}}{\partial out_{o1}} = 2 \times 0.5(target_{o1} - out_{o1}) \times (-1) = 0.741$$

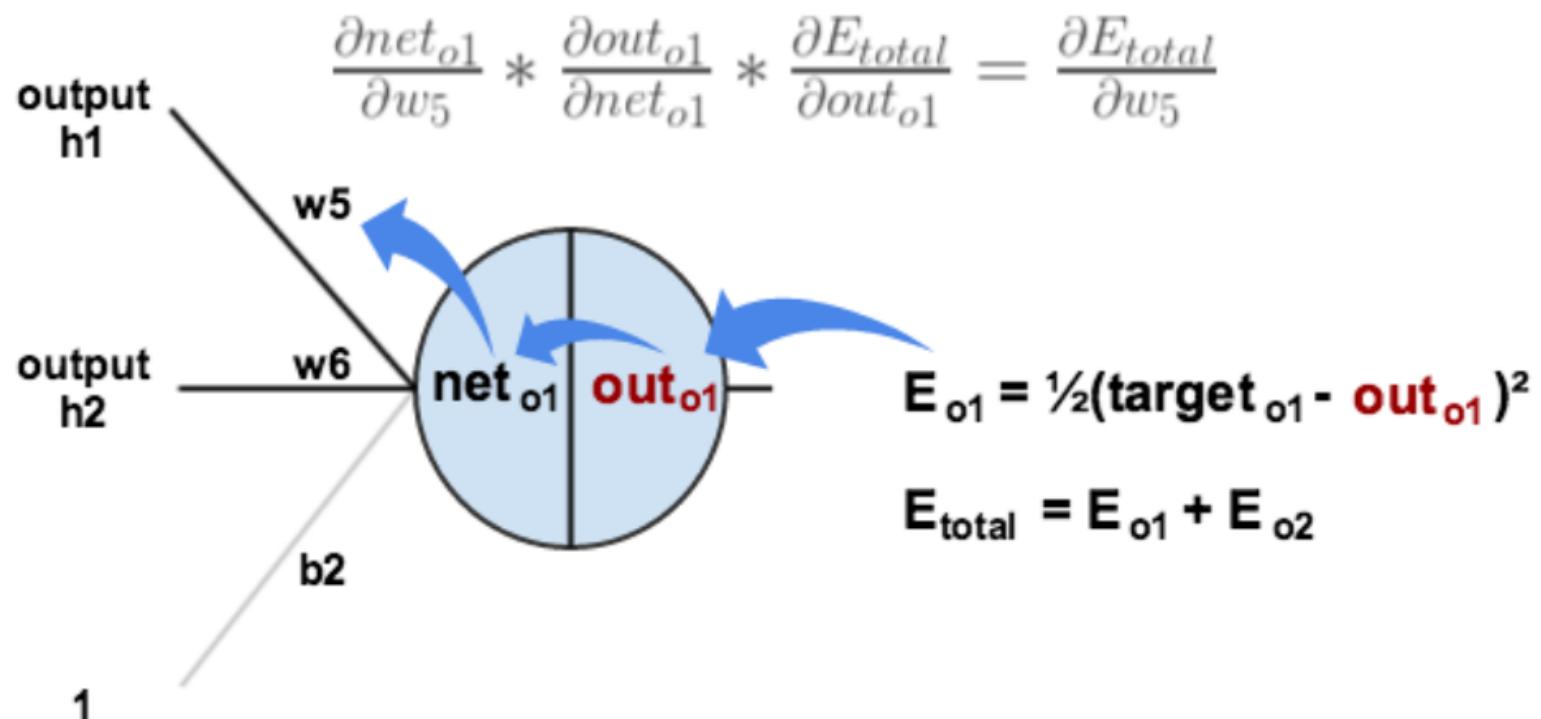


3. THE BACKWARD PASS

$$\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}$$

$$out_{o1} = \frac{1}{1 + e^{-in_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial in_{o1}} = out_{o1} \times (1 - out_{o1}) = 0.186$$

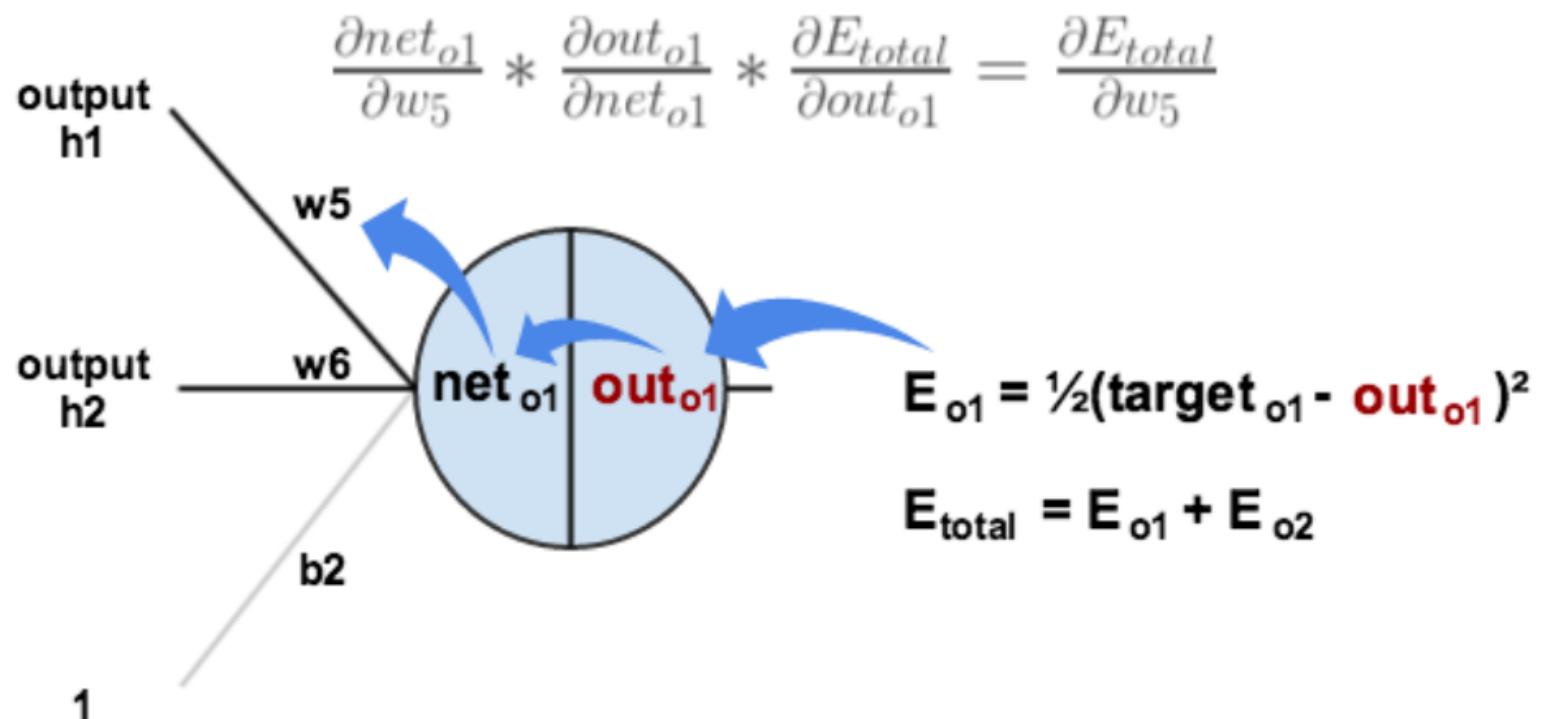


3. THE BACKWARD PASS

$$\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}$$

$$in_{o1} = w_5 \times out_{h1} + w_6 \times out_{h2} + b_2$$

$$\frac{\partial in_{o1}}{\partial w_5} = out_{h1} \times w_5^{1-1} = out_{h1} = 0.593$$

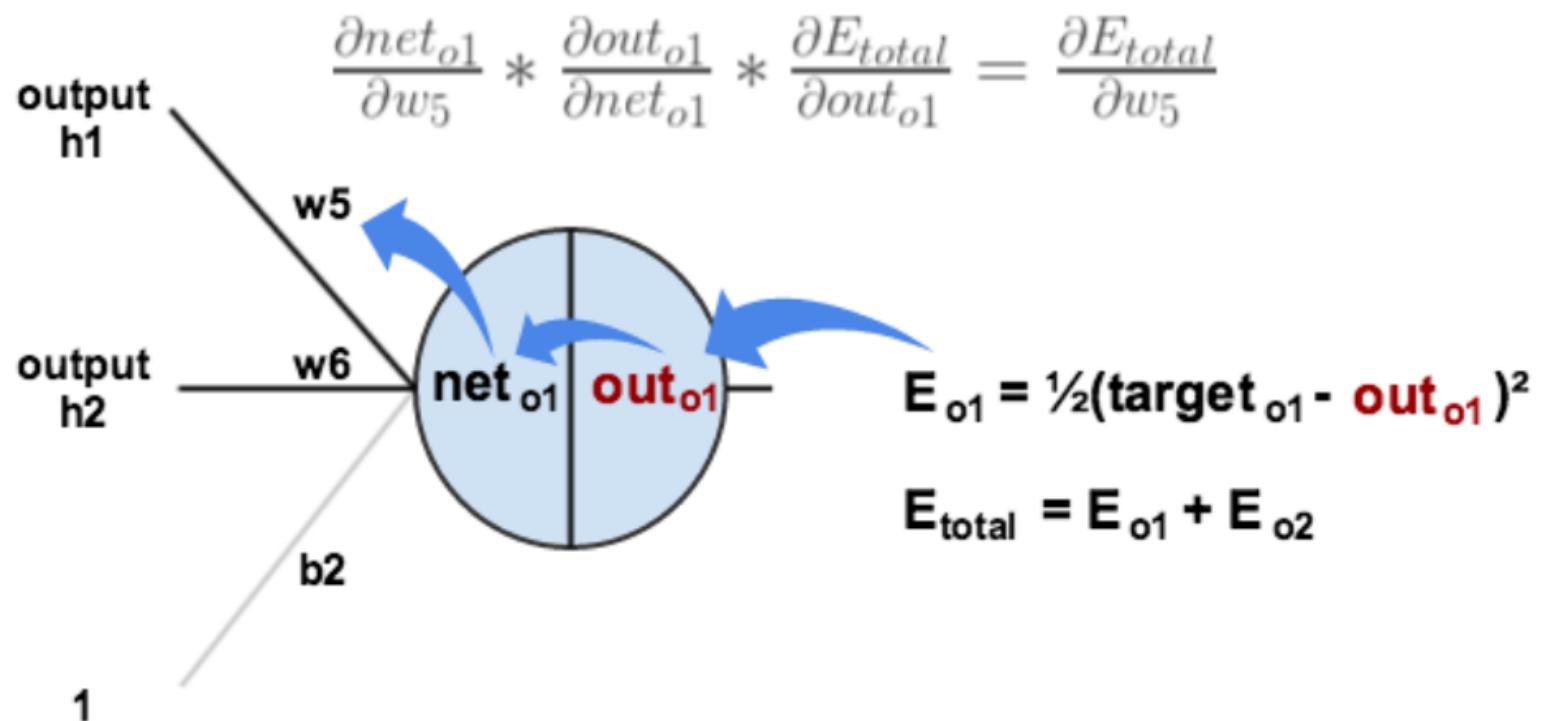


3. THE BACKWARD PASS

ALL TOGETHER:

$$\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}$$

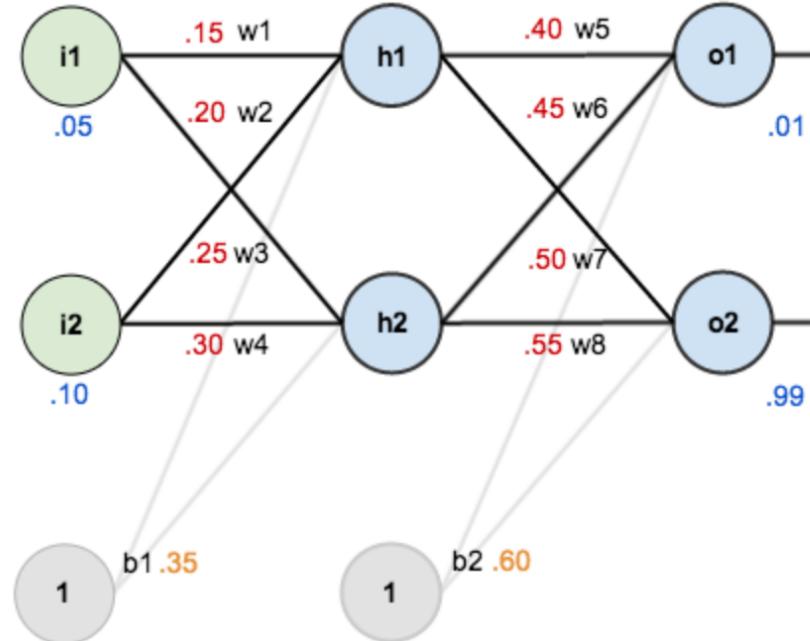
$$\frac{\partial L_{total}}{\partial w_5} = 0.741 \times 0.186 \times 0.593 = 0.082$$



4. UPDATE WEIGHTS WITH GRADIENT AND LEARNING RATE

$$w_5^{t+1} = w_5 - \lambda \times \frac{\partial L_{total}}{\partial w_5}$$

$$w_5^{t+1} = 0.4 - 0.5 \times 0.082 = 0.358$$



**THIS IS REPEATED FOR THE OTHER WEIGHTS
OF THE OUTPUT LAYER**

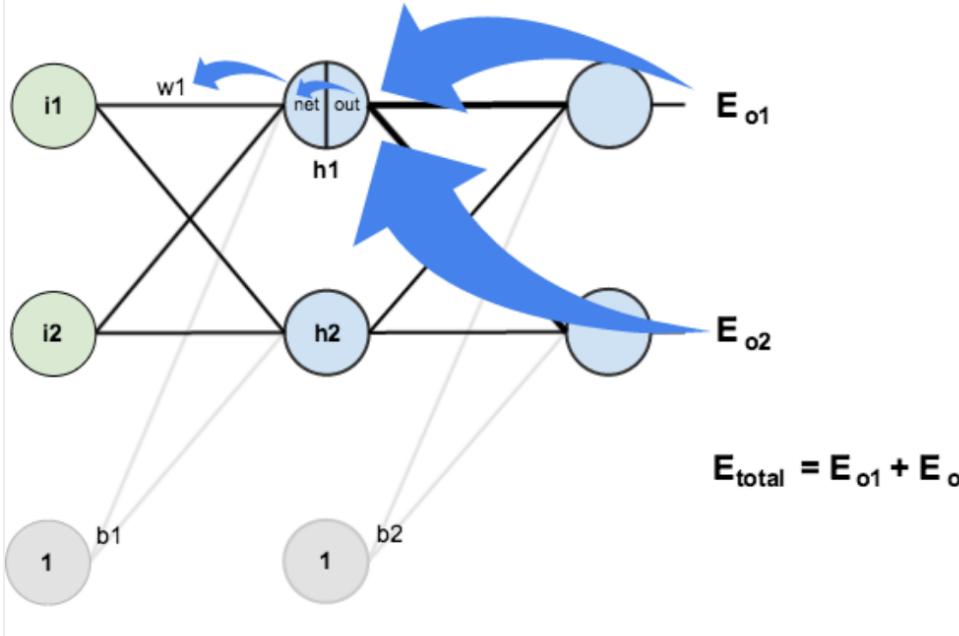
$$w_6^{t+1} = 0.408$$

$$w_7^{t+1} = 0.511$$

$$w_8^{t+1} = 0.561$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



AND BACK-PROPAGATED TO THE HIDDEN LAYERS

VISUALIZE SIMPLE NETWORK LEARNING

IN ORDER TO KNOW MORE
ABOUT BACKPROPAGATION...

[CLICK HERE](#)

RECAP:

