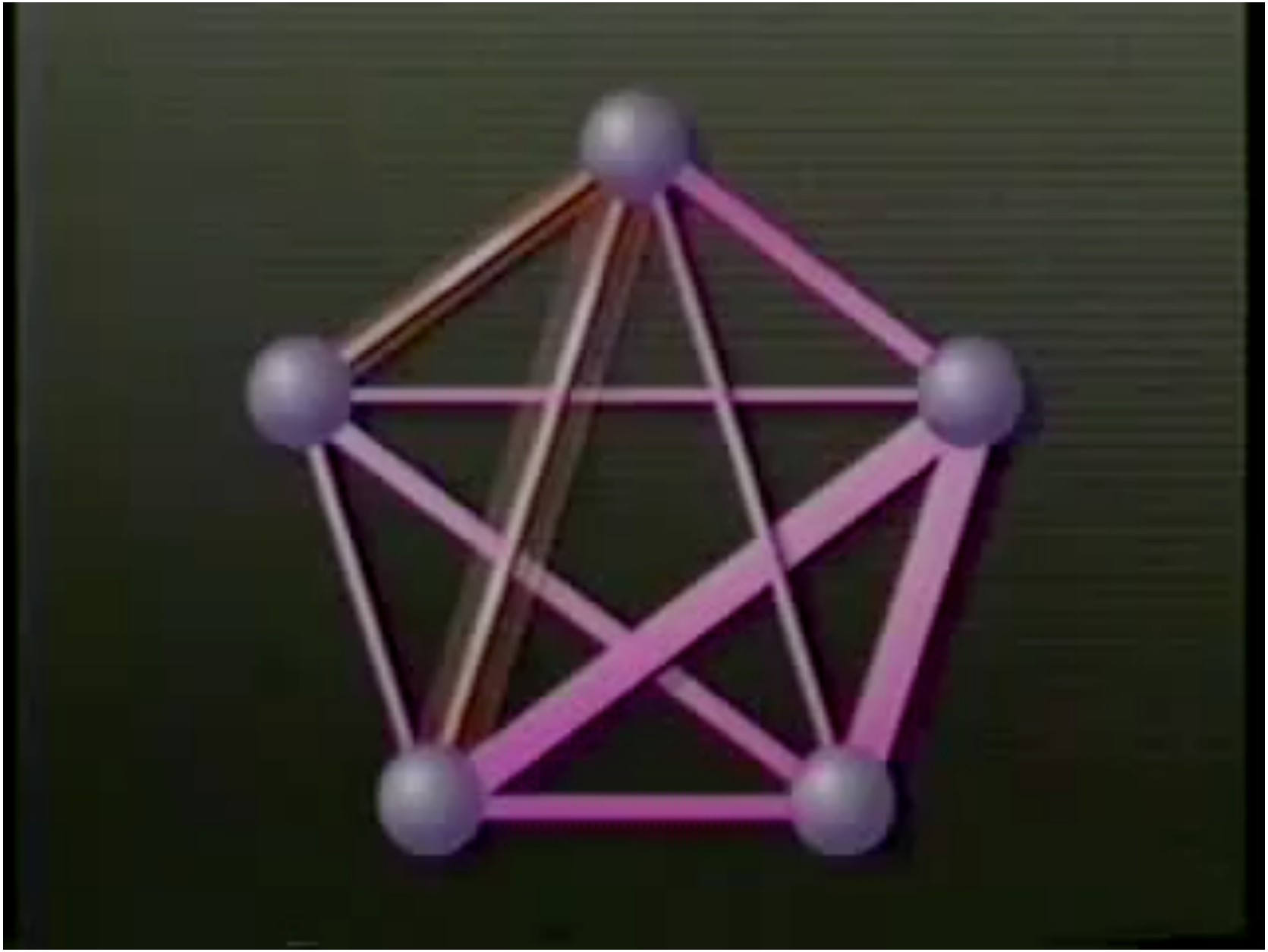
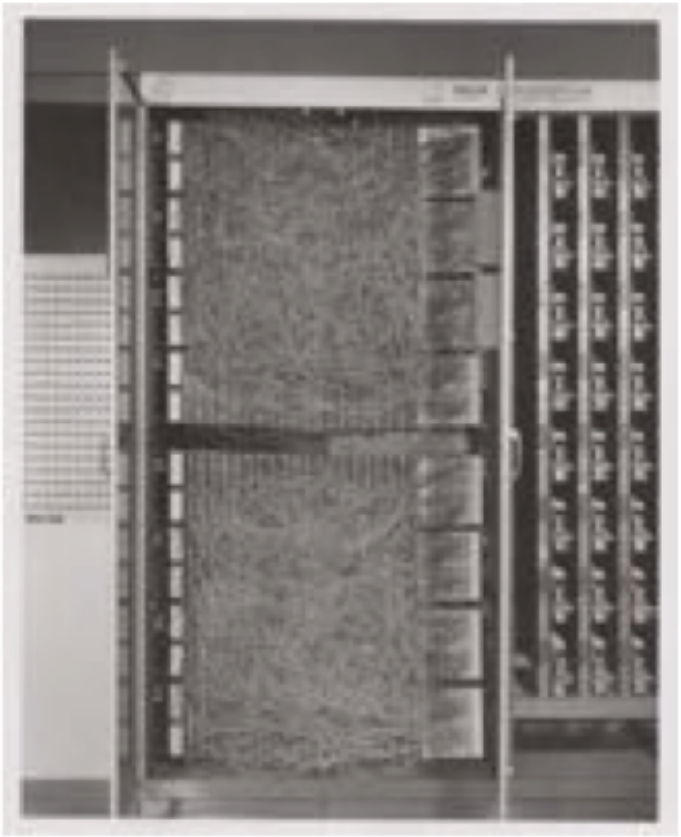
PART II: FOUNDATIONS OF "SHALLOW" NEURAL NETWORKS



Rosenblatt Perceptron

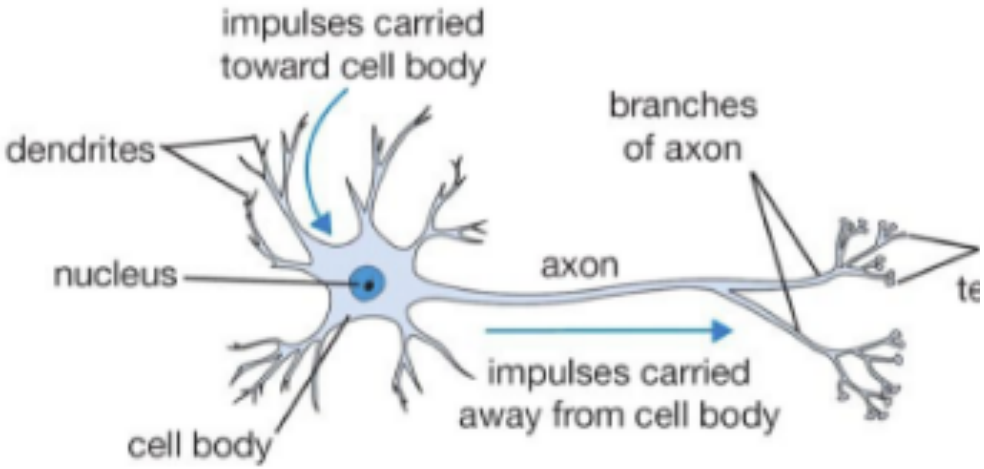
FIRST IMPLEMENTATION OF NEURAL NETWORK [Rosenblatt, **1957!**]

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

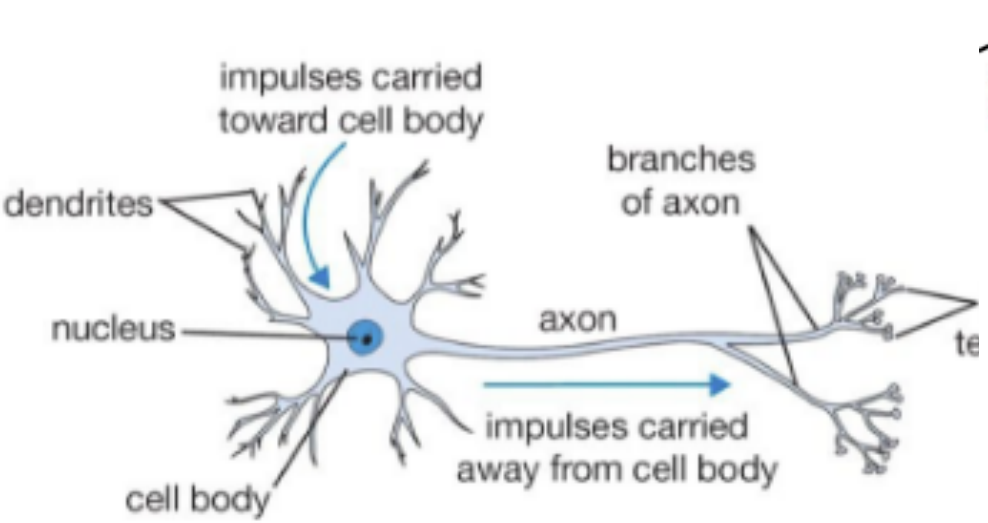
THE NEURON

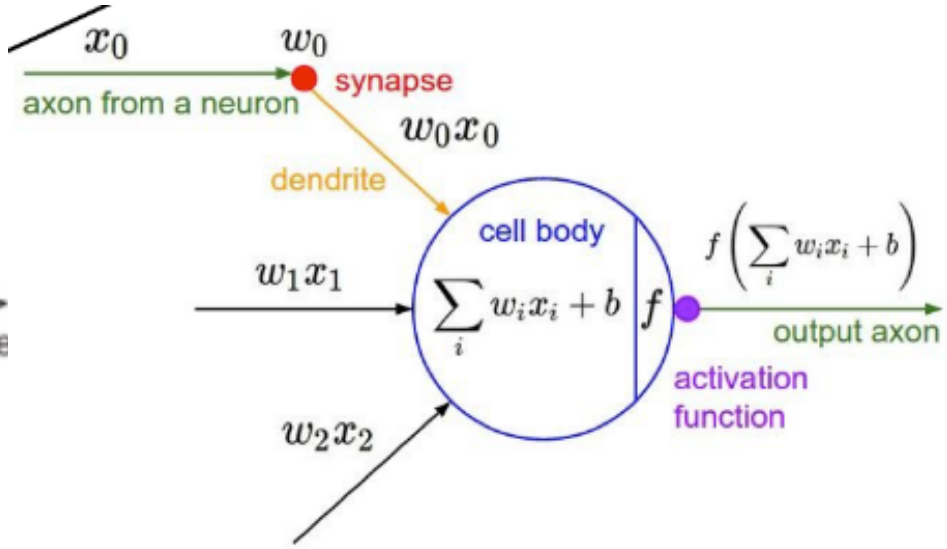


INSPIRED BY NEURO - SCIENCE?

Credit: Karpathy

THE NEURON

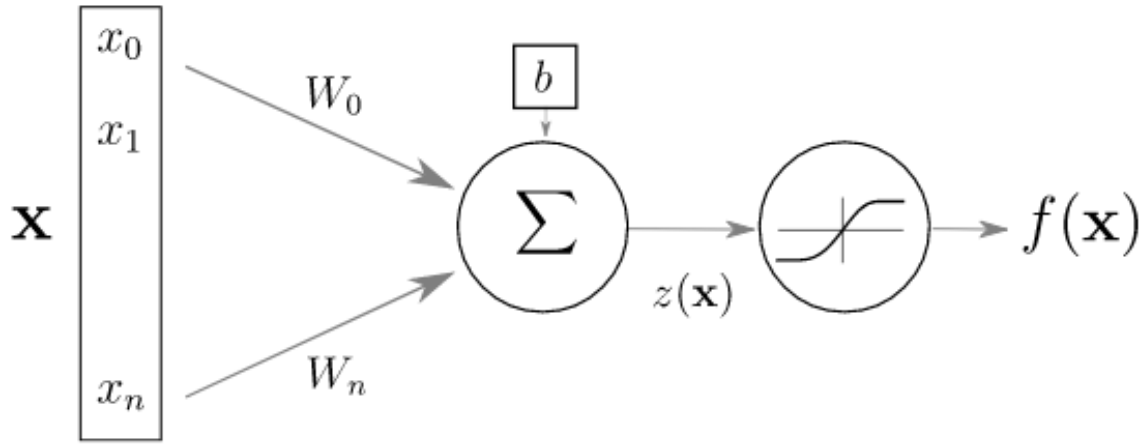




INSPIRED BY NEURO - SCIENCE?

Credit: Karpathy

TODAY'S ARTIFICIAL NEURON



$$z(\vec{x}) = \vec{W}.\vec{x} + b \qquad f(\vec{x}) = g(\vec{W}.\vec{x} + b)$$

Pre-Activation

Weights

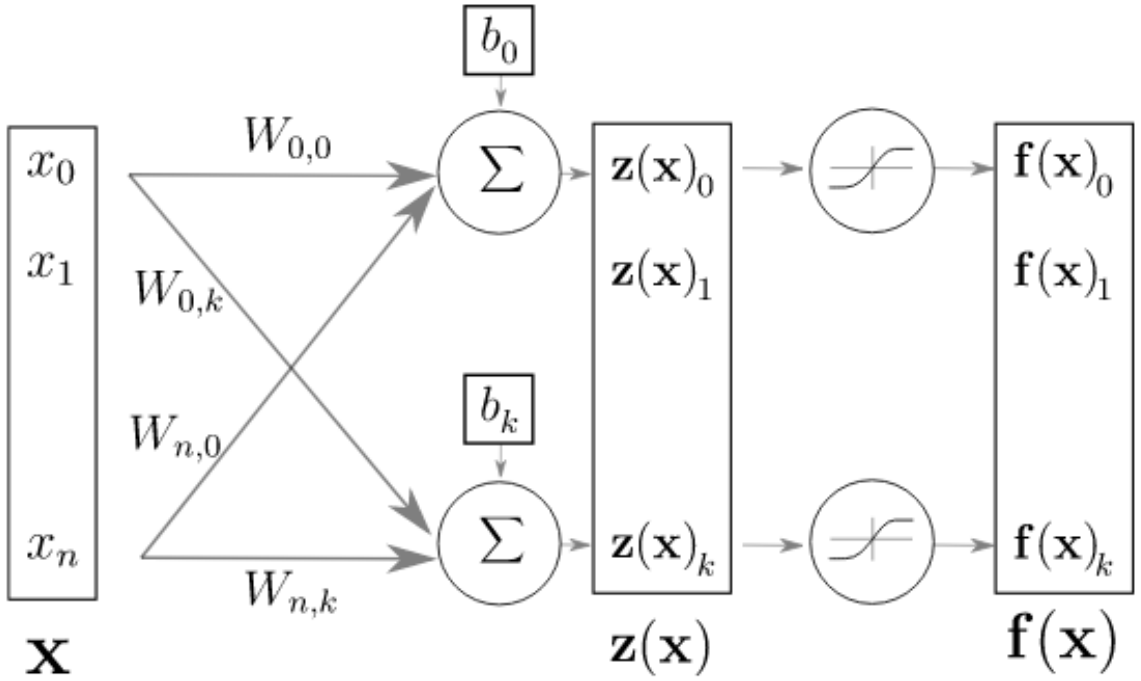
Input

Bias

Output

Activation Function

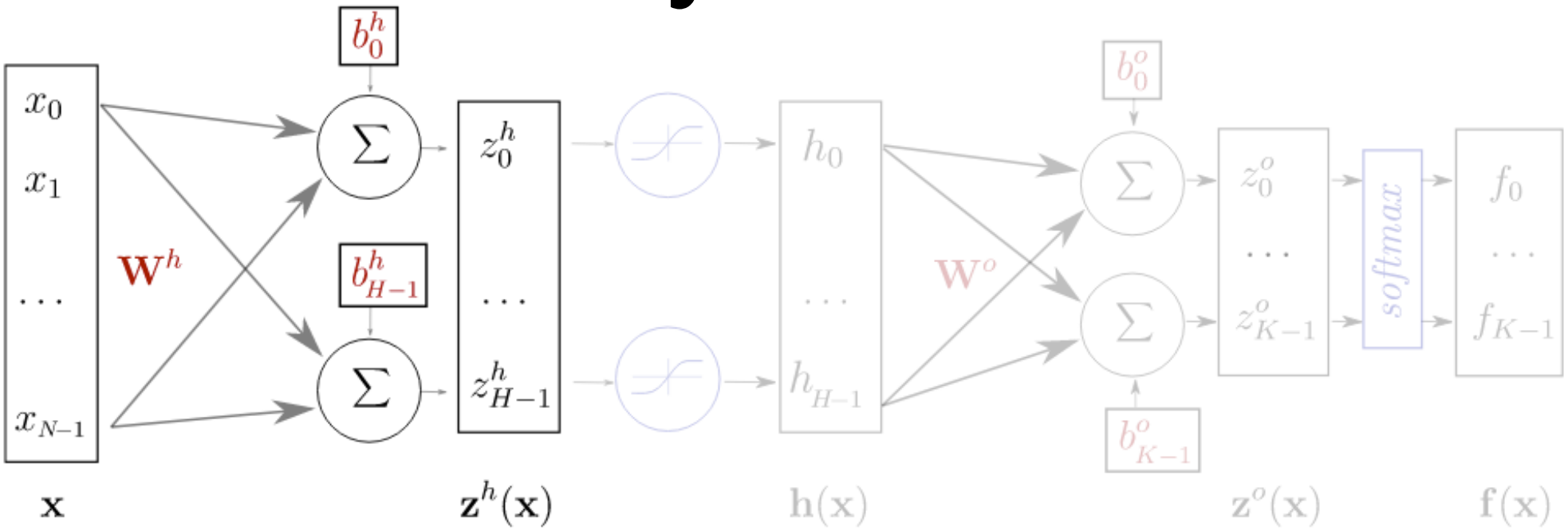
LAYER OF NEURONS



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

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

Hidden Layers of Neurons

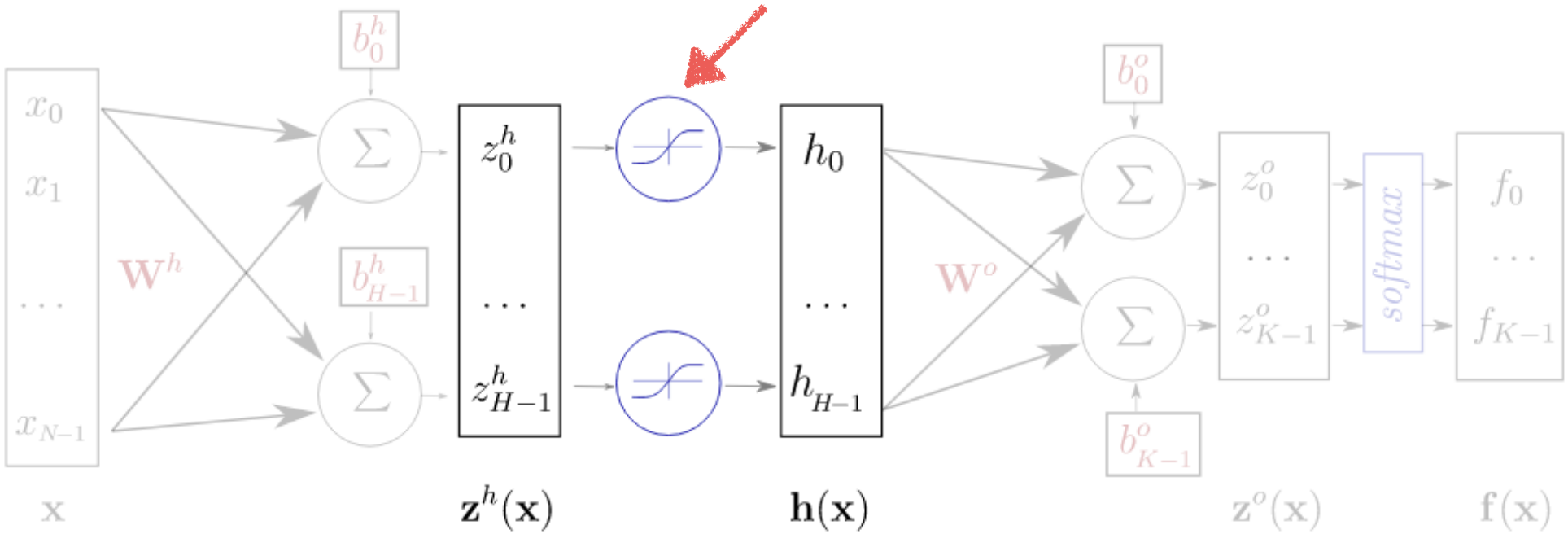


FIRST LAYER

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

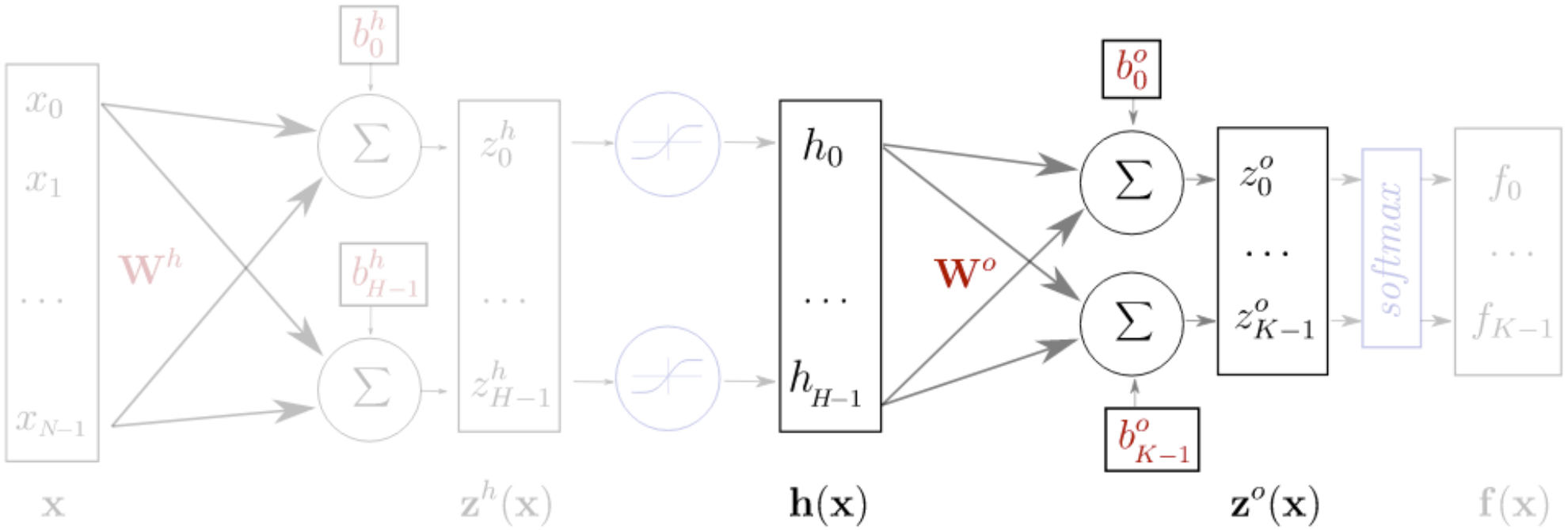
INPUT

ACTIVATION FUNCTION



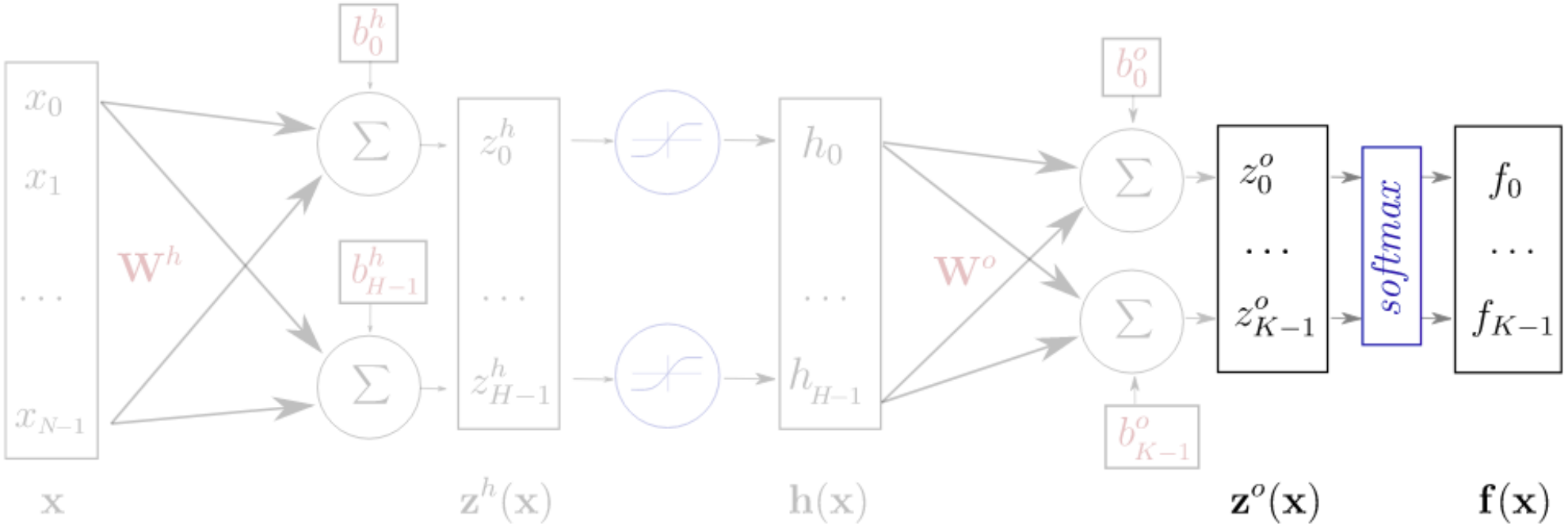
HIDDEN LAYER

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



OUTPUT LAYER

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



PREDICTION LAYER

$$f(\mathbf{x}) = softmax(\mathbf{z^0})$$

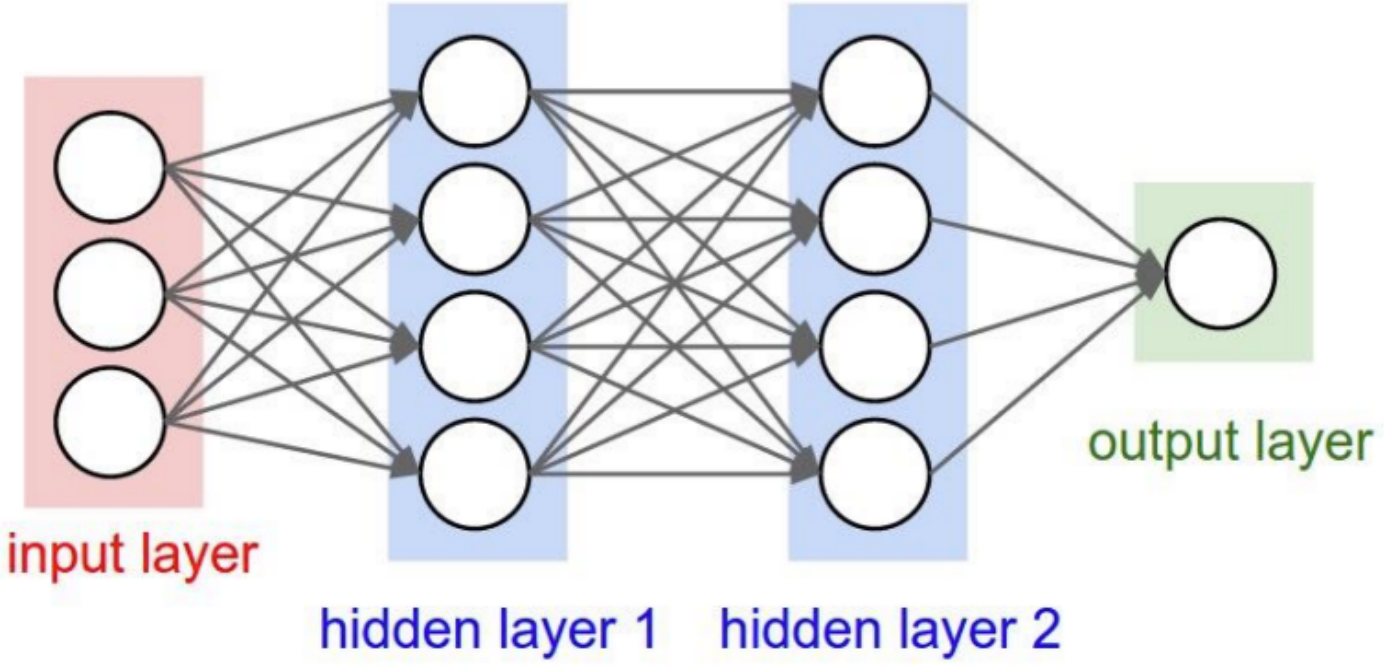
“CLASSICAL” MACHINE LEARNING

$$f_{\vec{W}}(\vec{x}) = \vec{y}$$

LABEL

Q , SF

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



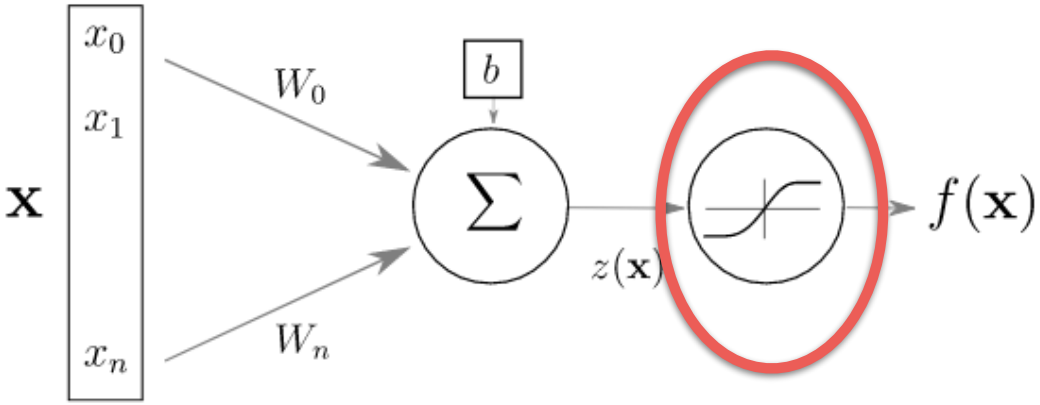
$$p = g_3(W_3 g_2(W_2 g_1(W_1 \vec{x}_0)))$$

NETWORK FUNCTION

SO LET’S GO DEEPER AND DEEPER!

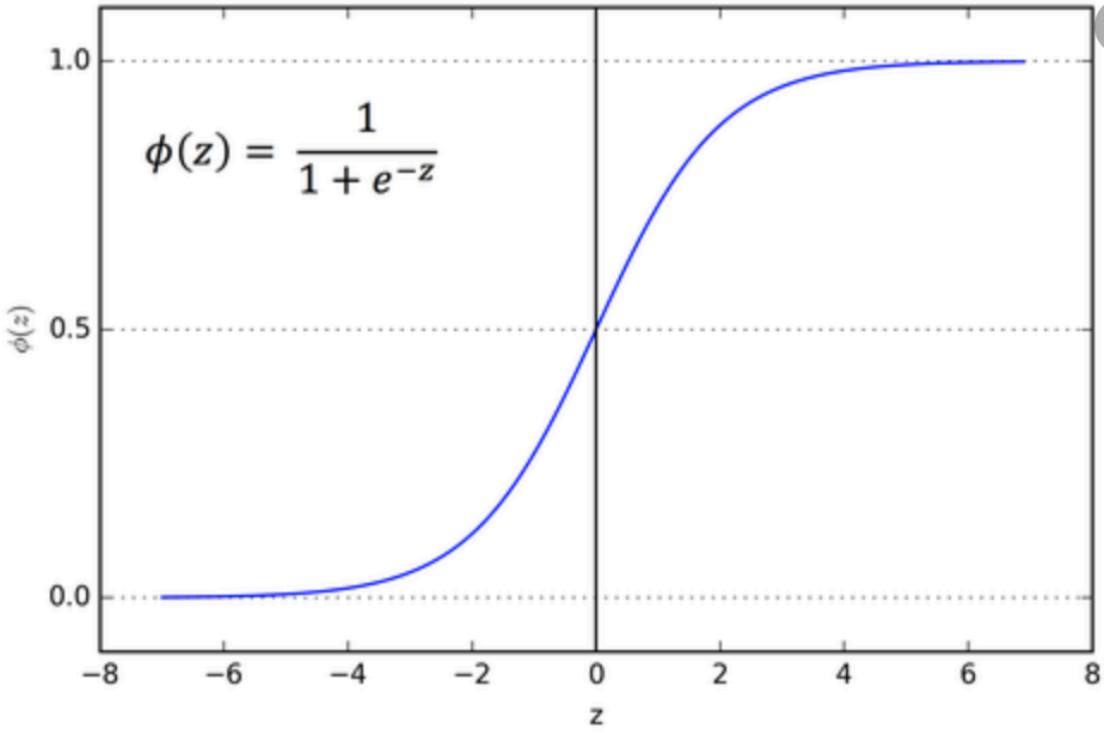
LET’S FIRST EXAMINE IN MORE DETAIL HOW SIMPLE “SHALLOW” NETWORKS WORK

ACTIVATION FUNCTIONS?

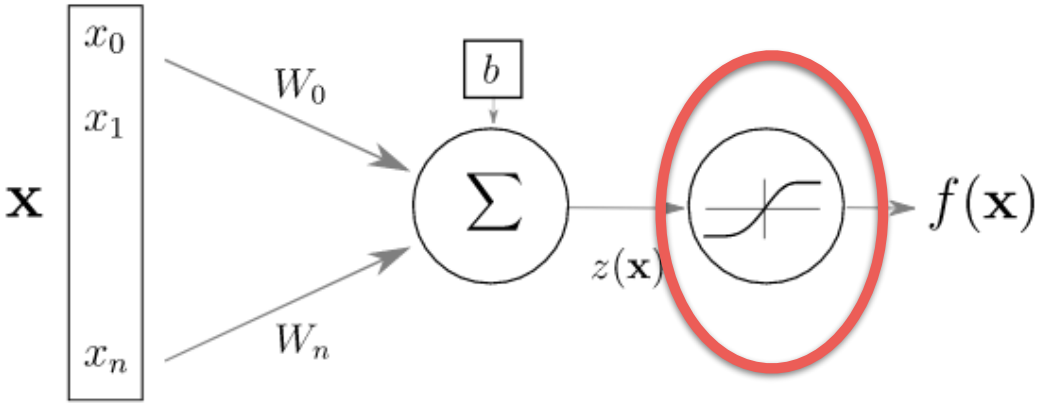


ADD NON LINEARITIES TO THE PROCESS

ACTIVATION FUNCTIONS?

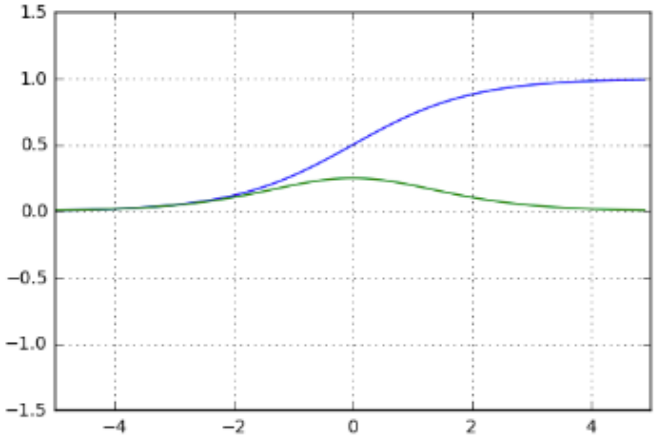


the sigmoid function



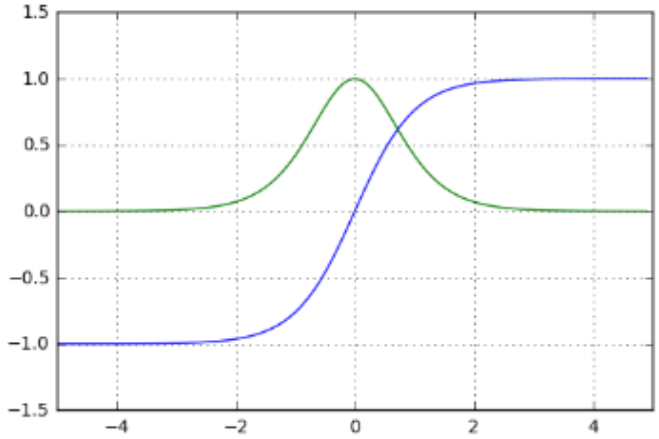
ADD NON LINEARITIES TO THE PROCESS

ACTIVATION FUNCTIONS



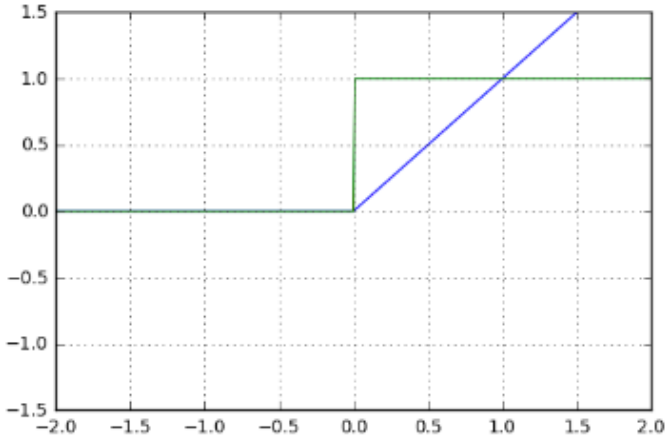
$$\text{sigm}(x) = \frac{1}{1+e^{-x}}$$

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



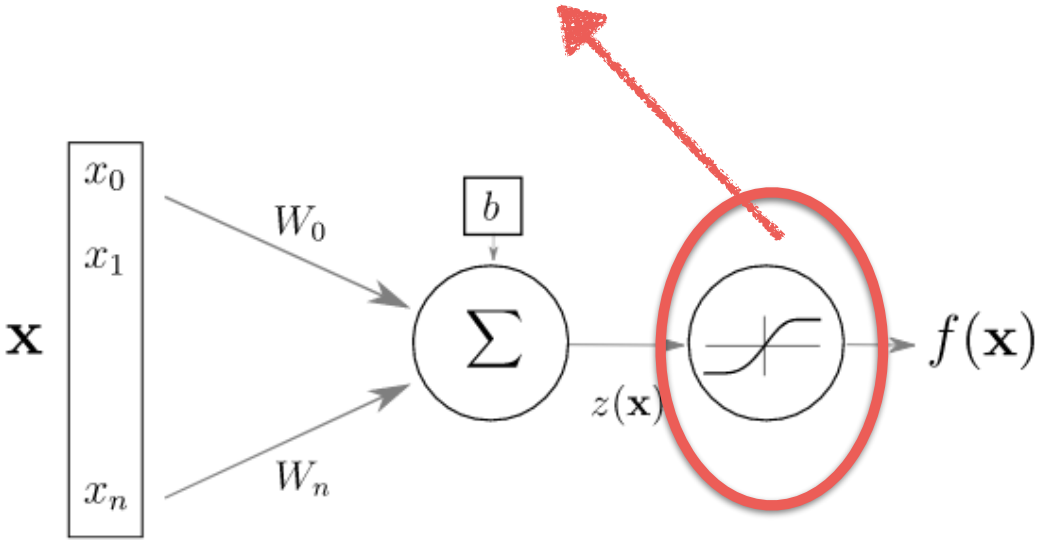
$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

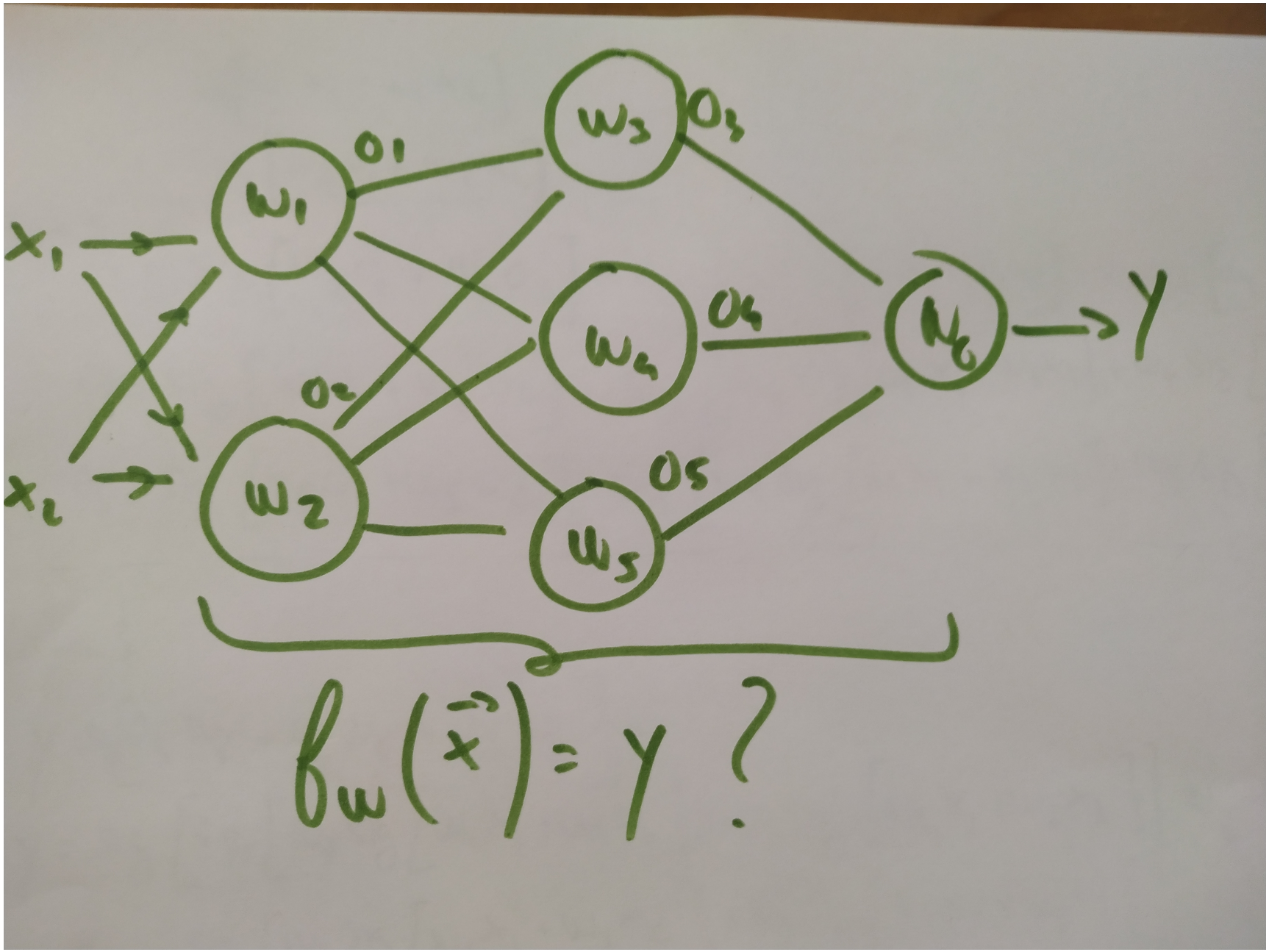
$$\tanh'(x) = 1-\tanh(x)^2$$



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

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




x_1
x_2
w_1
w_2
w_3
w_4
w_5
w_6
o_1
o_2
o_3
o_4
o_5
$\rightarrow y$
$f_w(\vec{x}) = y$?

A: $sig\left[w_5 sig\left[w_1 x_1 + w_1 x_2\right] * w_6 + w_4 sig\left[w_2 x_1 + w_2 x_2\right] * w_6 + w_5 sig\left[w_5 sig\left[w_1 x_1 + w_1 x_2\right]\right] \cdot w_6\right]$

B: $sig\left[sig\left[w_5 sig\left[w_1 x_1 + w_1 x_2\right] + w_5 sig\left[w_2 x_1 + w_2 x_2\right]\right] \cdot w_6 + sig\left[w_4 sig\left[w_1 x_1 + w_1 x_2\right] + w_4 sig\left[w_2 x_1 + w_2 x_2\right]\right] \cdot w_6 + sig\left[w_5 sig\left[w_1 x_1 + w_1 x_2\right] + w_5 sig\left[w_2 x_1 + w_2 x_2\right]\right] \cdot w_6\right]$

C: $sig\left[sig\left[w_5 sig\left[w_1 x_1 + w_1 x_2\right] \cdot w_6 + w_4 sig\left[w_2 x_1 + w_2 x_2\right] * w_6 + w_5 sig\left[w_1 x_1 + w_1 x_2\right] * w_6\right]\right.$

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

A

B

C

AND REMEMBER WE NEED A LOSS FUNCTION ...

AND 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:

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

NUMBER OF TRAINING EXAMPLES

GROUND TRUTH

NETWORK OUTPUT (SIGMOID)

GROUND TRUTH

NETWORK OUTPUT (SIGMOID)

BINARY CLASSIFICATION LOSS

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

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

NUMBER OF TRAINING EXAMPLES

GROUND TRUTH

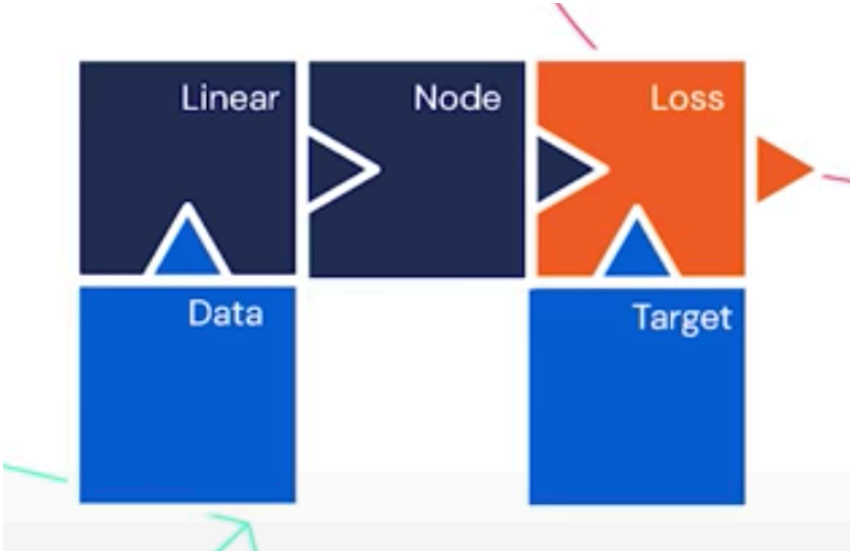
NETWORK OUTPUT (SIGMOID)

GROUND TRUTH

NETWORK OUTPUT (SIGMOID)

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

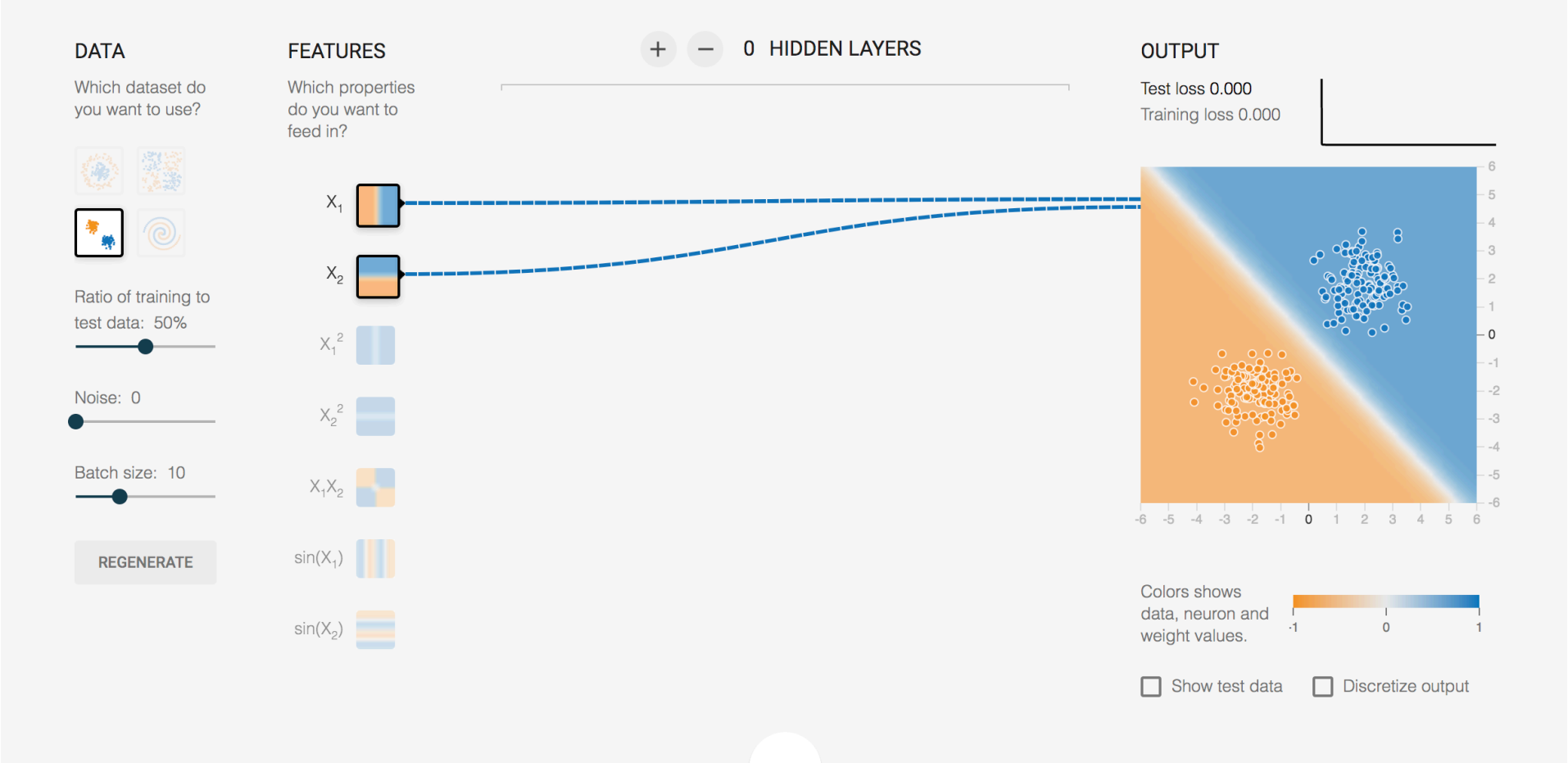
1. LINEAR NEURON OR UNIT
2. NON LINEARITIES (ACTIVATION FUNCTION)
3. LOSS FUNCTION



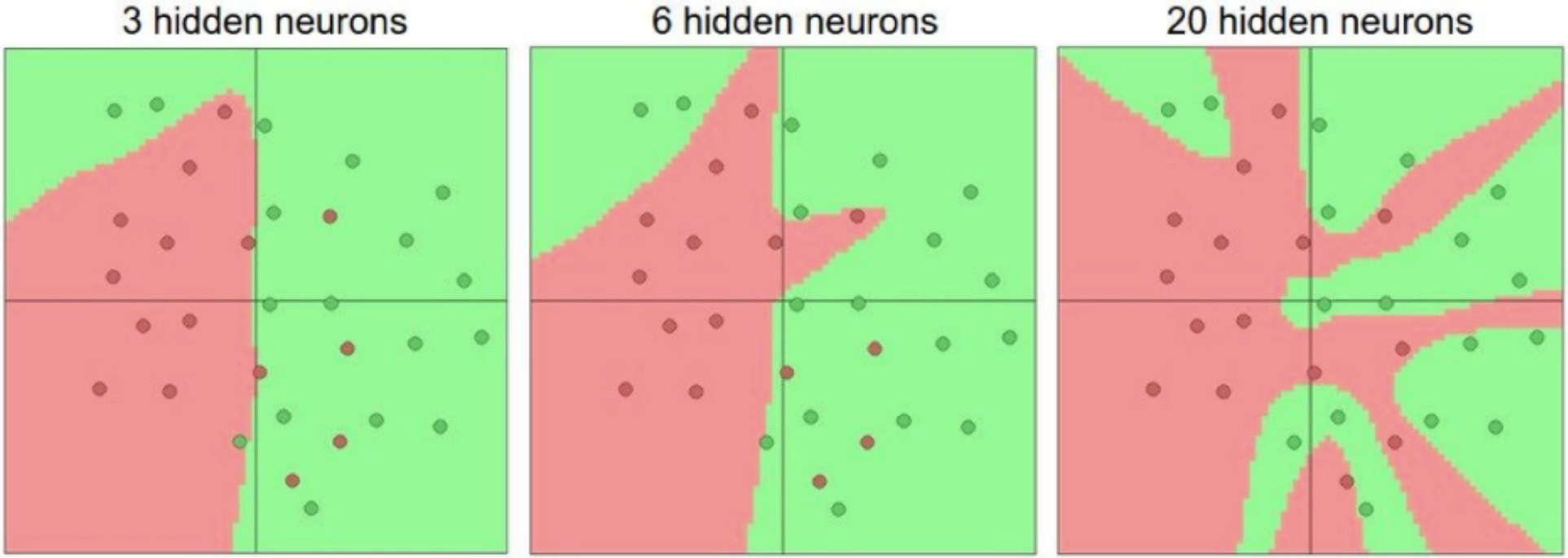
(deepmind @Czarnecki)

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

GO HERE:

https://github.com/mhuertascompany/Saas-Fee/blob/main/hands-on/session1/hello_ANN.ipynb

UNIVERSAL APPROXIMATION THEOREM

FOR ANY CONTINUOS 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

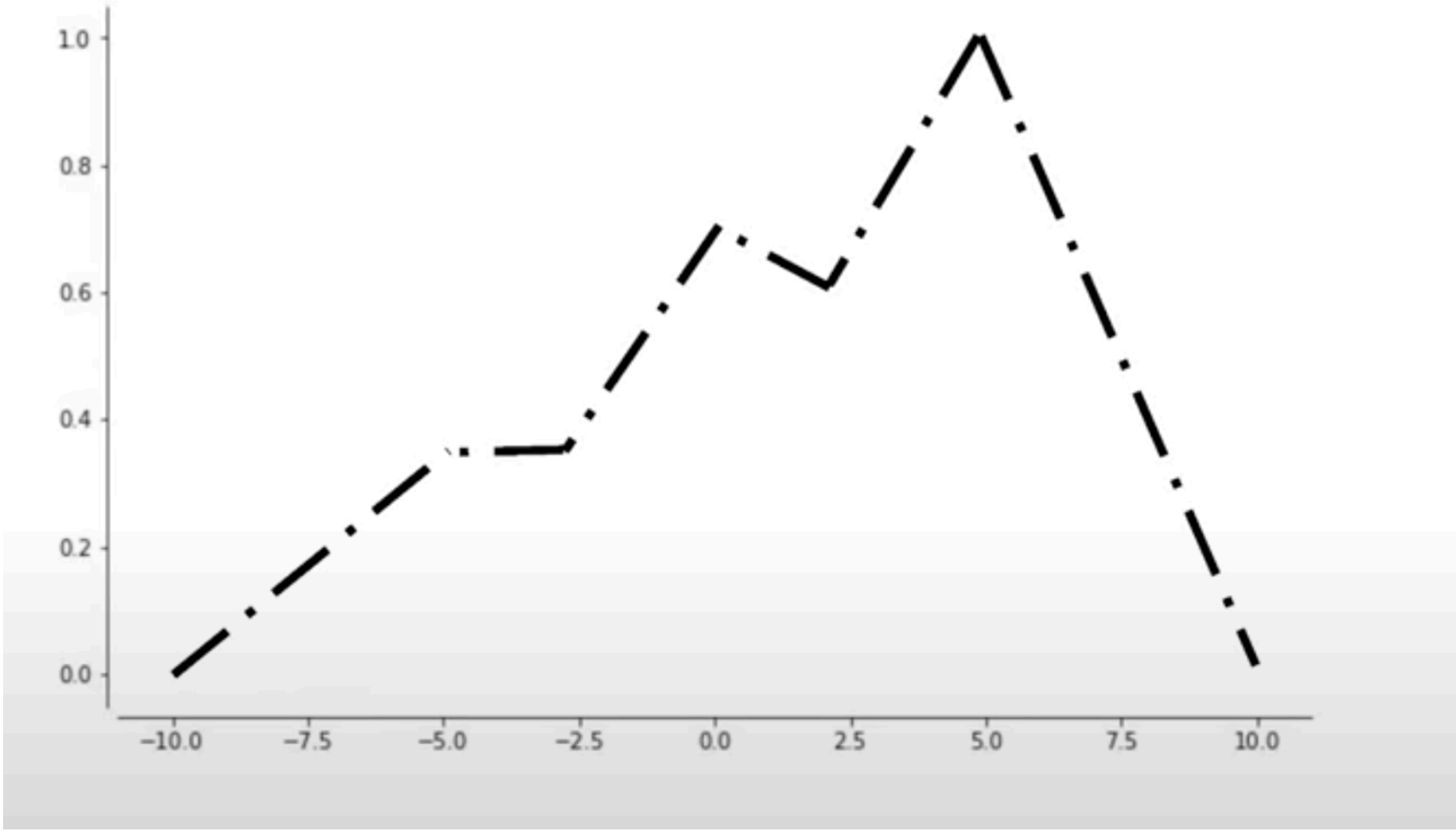
"BIG ENOUGH NETWORK CAN APPROXIMATE, BUT NOT REPRESENT ANY SMOOTH FUNCTION. THE MATH DEMONSTRATION IMPLIES SHOWING THAT NETWORS ARE DENSE IN THE SPACE OF TARGET FUNCTIONS"

UNIVERSAL APPROXIMATION THEOREM

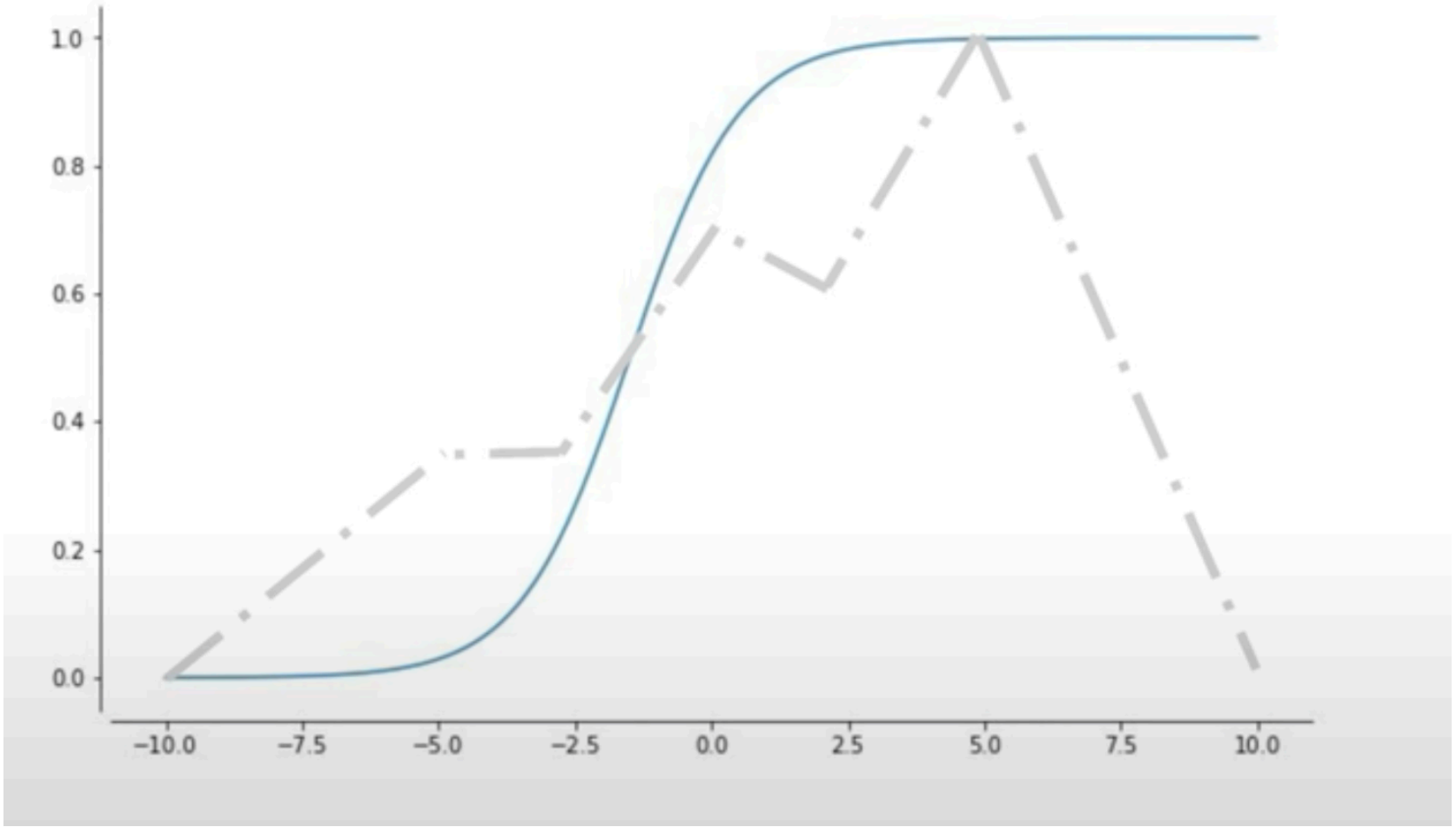
FOR ANY CONTINUOS FUNCTION FOR A HYPERCUBE [0,1]d TO REAL NUMBERS, NON-CONSTANT, BOUNDED AND CONTINUOUS ACTIVATION FUNCTION f, AND EVERY POSITIVE EPSILON, THERE EXISTS A1-HIDDEN LAYER NEURAL NETWORK USING f THAT OBTAINS AT MOST EPSILON ERROR IN FUNCTIONAL SPACE

Horvik+91

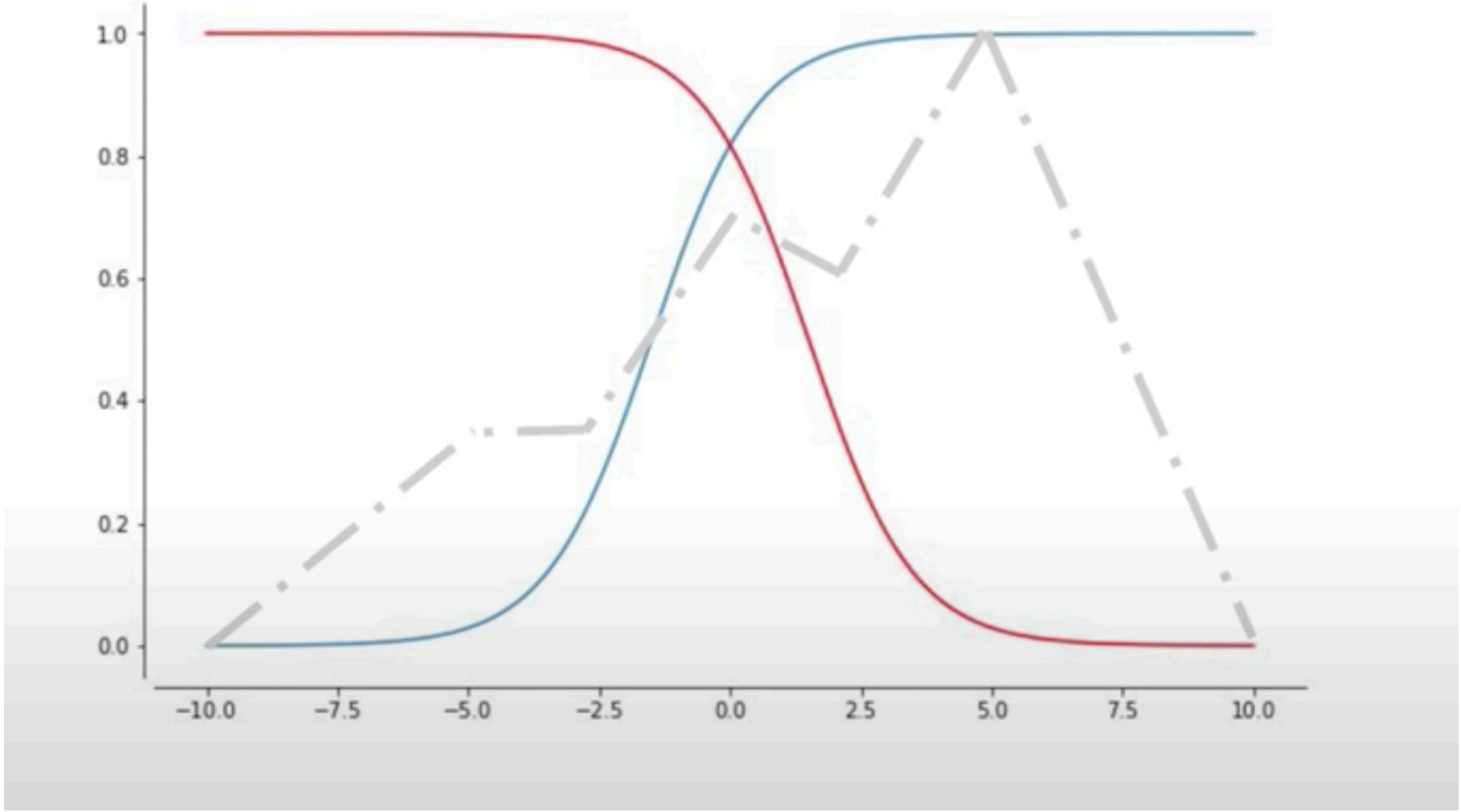
"BIG ENOUGH NETWORK CAN APPROXIMATE, BUT NOT REPRESENT ANY SMOOTH FUNCTION. THE MATH DEMONSTRATION IMPLIES SHOWING THAT NETWORS ARE DENSE IN THE SPACE OF TARGET FUNCTIONS"



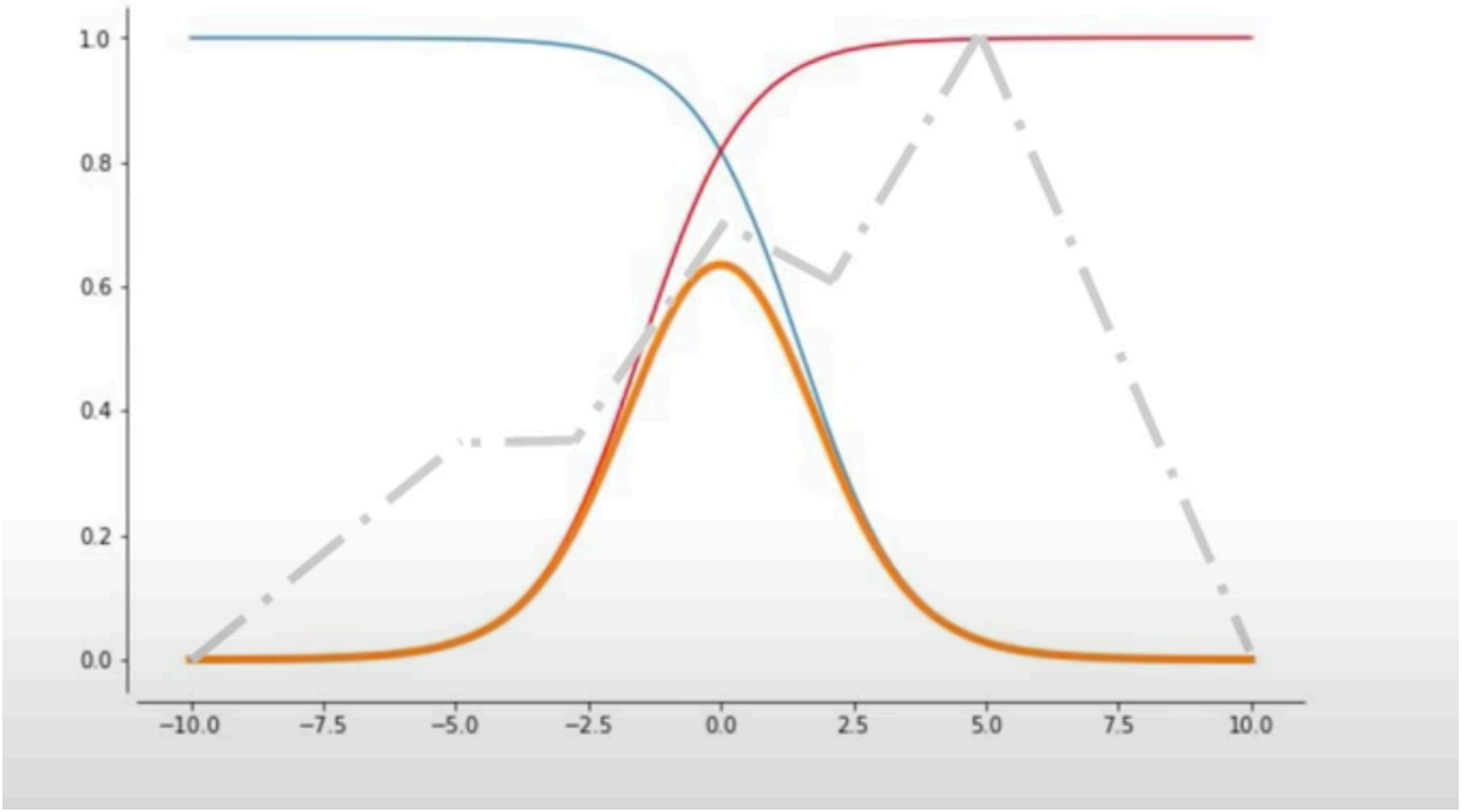
(deepmind @Czarnecki)



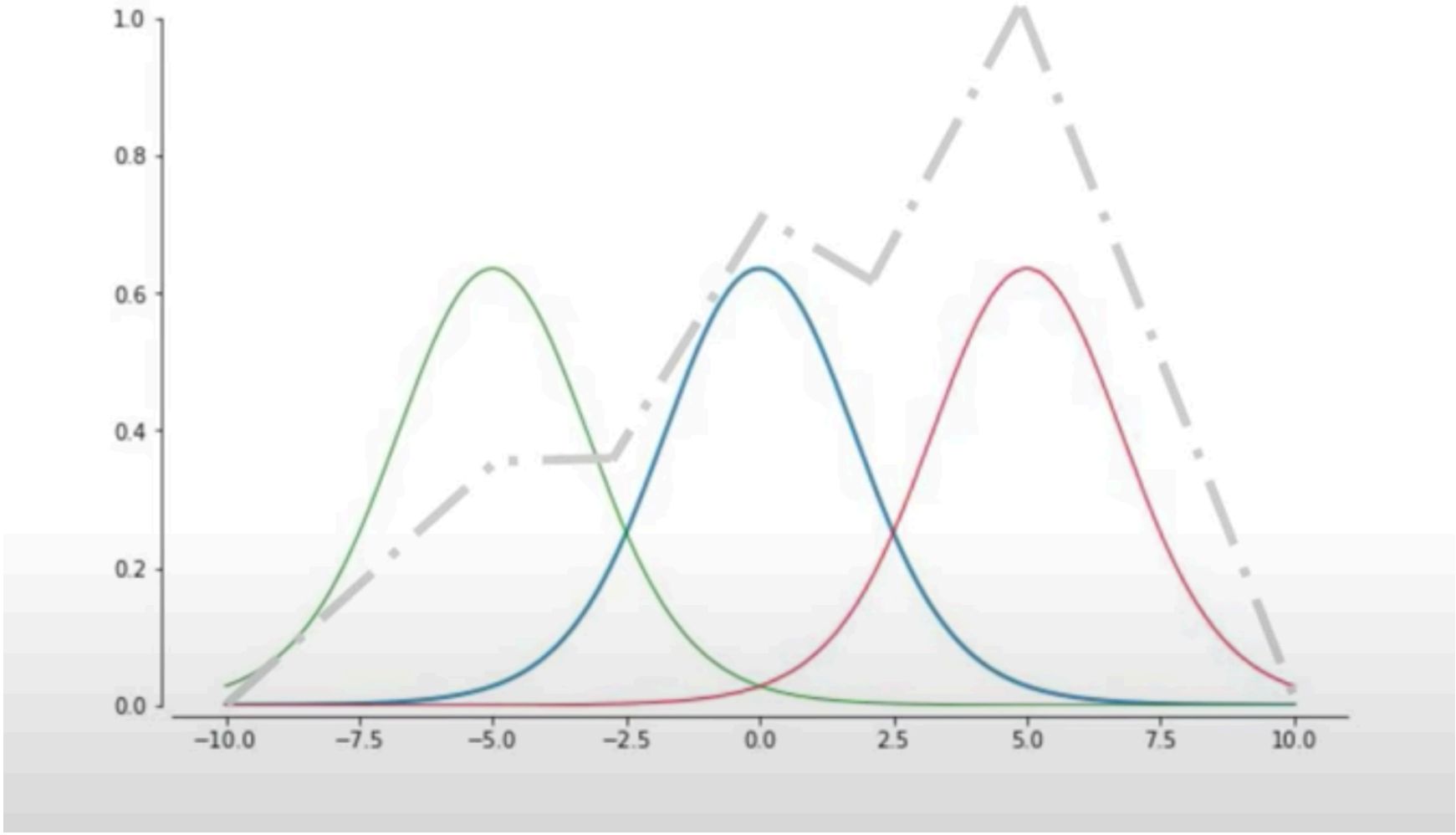
(deepmind @Czarnecki)



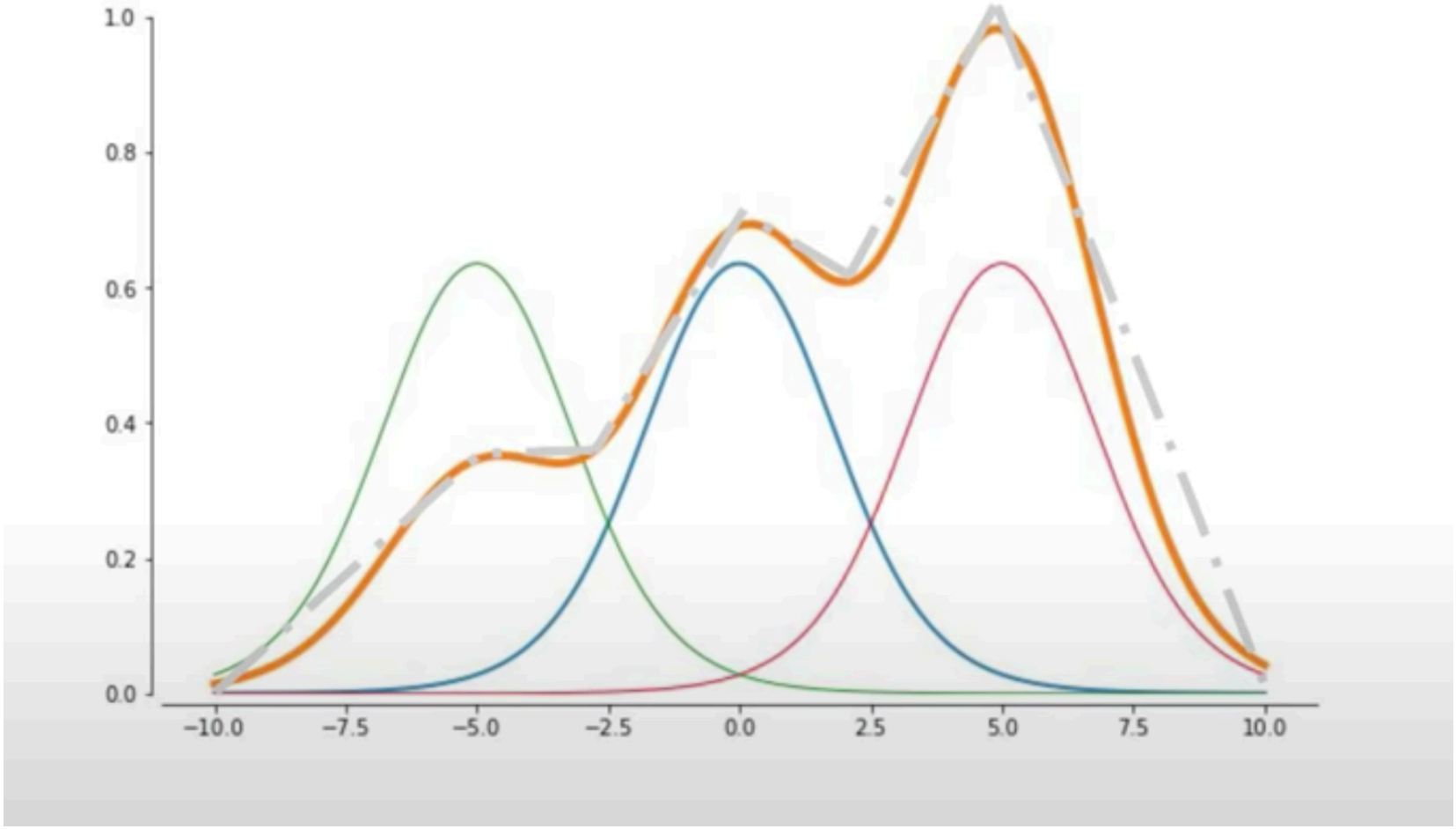
(deepmind
@Czarnecki)



(deepmind
@Czarnecki)

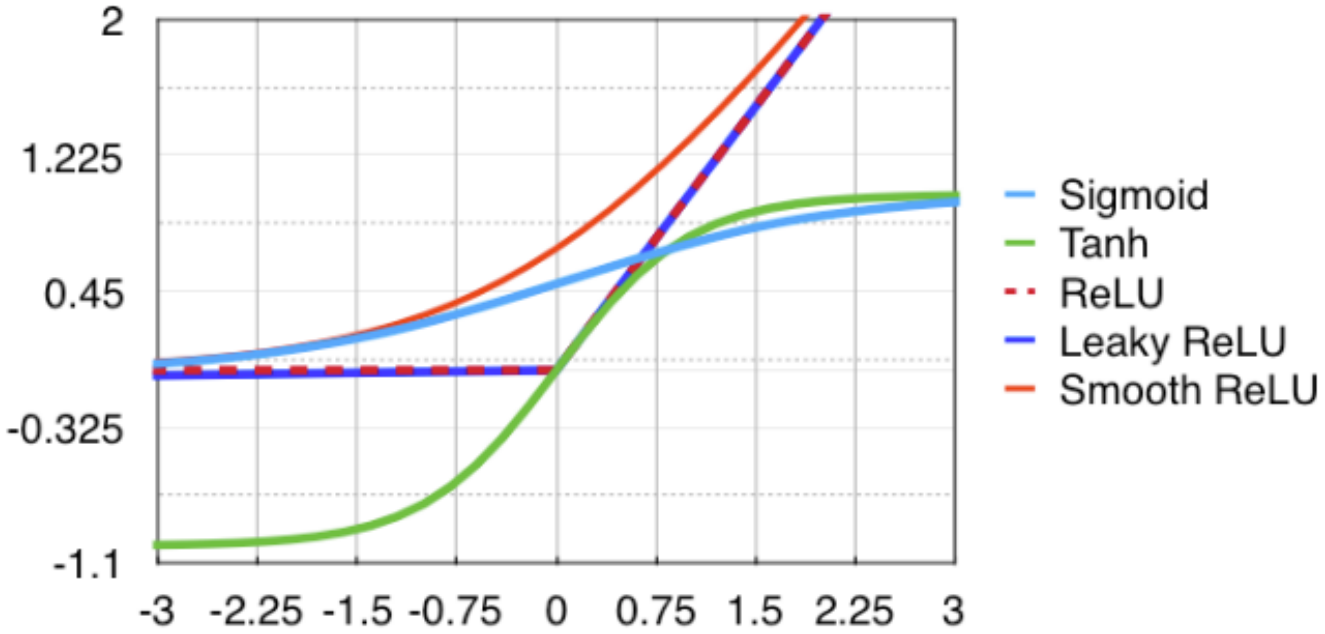


(deepmind
@Czarnecki)



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

SOFTMAX

A generalization of the SIGMOID ACTIVATION

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

THE OUTPUT IS NORMALIZED BETWEEN 0 AND 1

THE COMPONENTS ADD TO 1

CAN BE INTERPRETED AS A PROBABILITY

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

SO LET’S GO DEEPER AND DEEPER!

YES BUT…

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