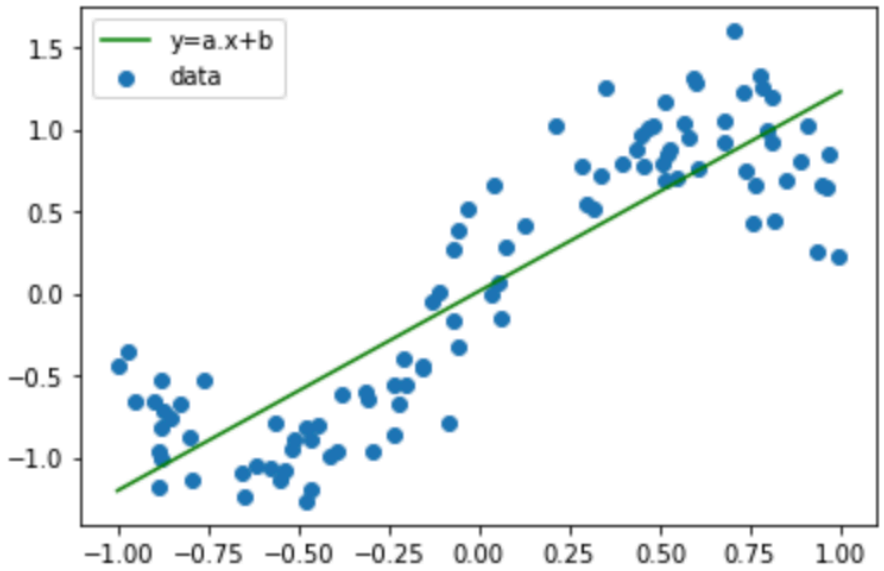
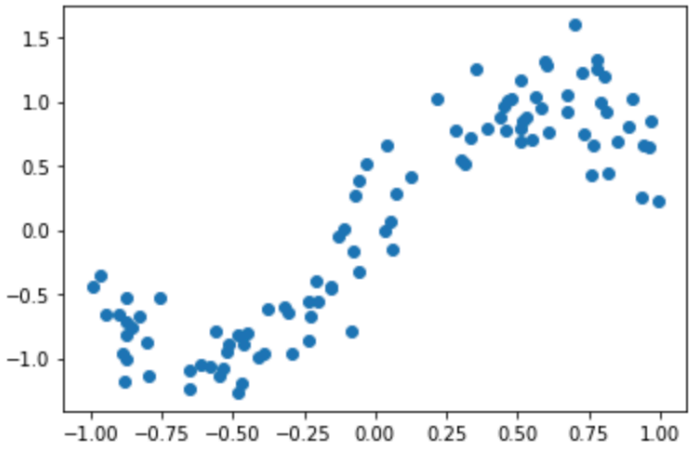
Introduction to deep learning

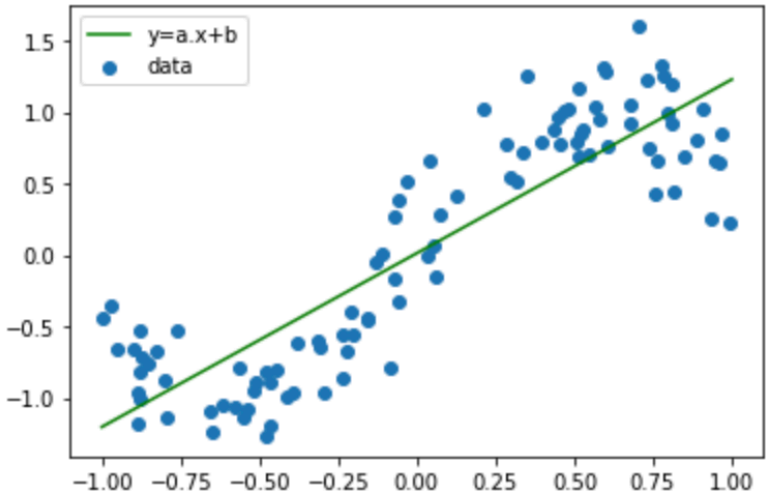
Marc Huertas-Company

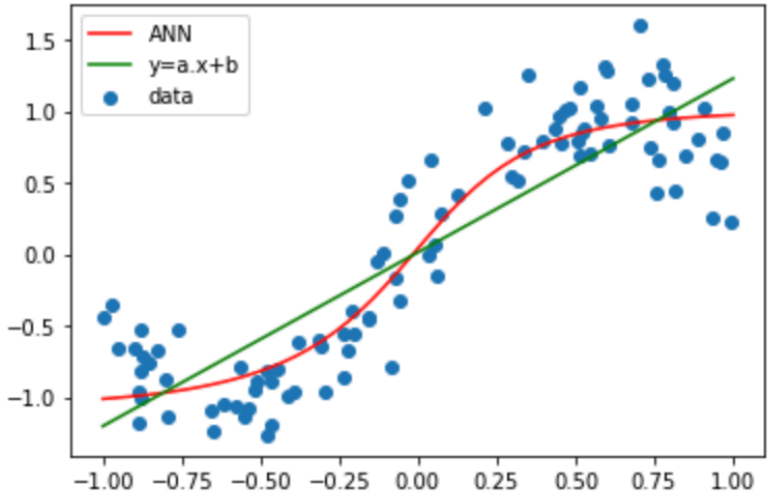


Is this machine learning?

y=F(x;w)? w=(a,b)





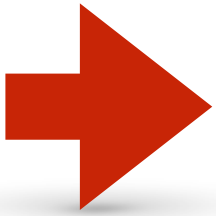


p is derived from physical insight

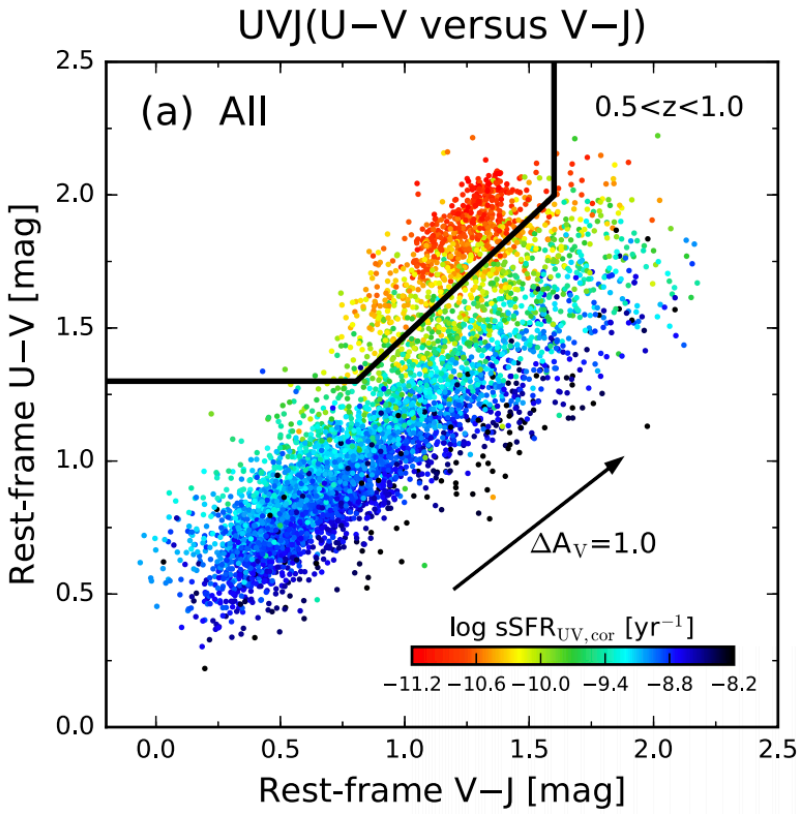
p is data driven, no physics

Why data driven?

- No suitable physical model available: **accuracy**
- Physical Model too complex or dataset too large, minimisation difficult: **speed**
- There might be hidden information in the data (beyond usual summary statistics): **discovery**



Motivated by large and complex datasets



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

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

Is this color-color plot machine learning?

- Yes
- No
- It depends
- I don't know

Powered by Poll Everywhere

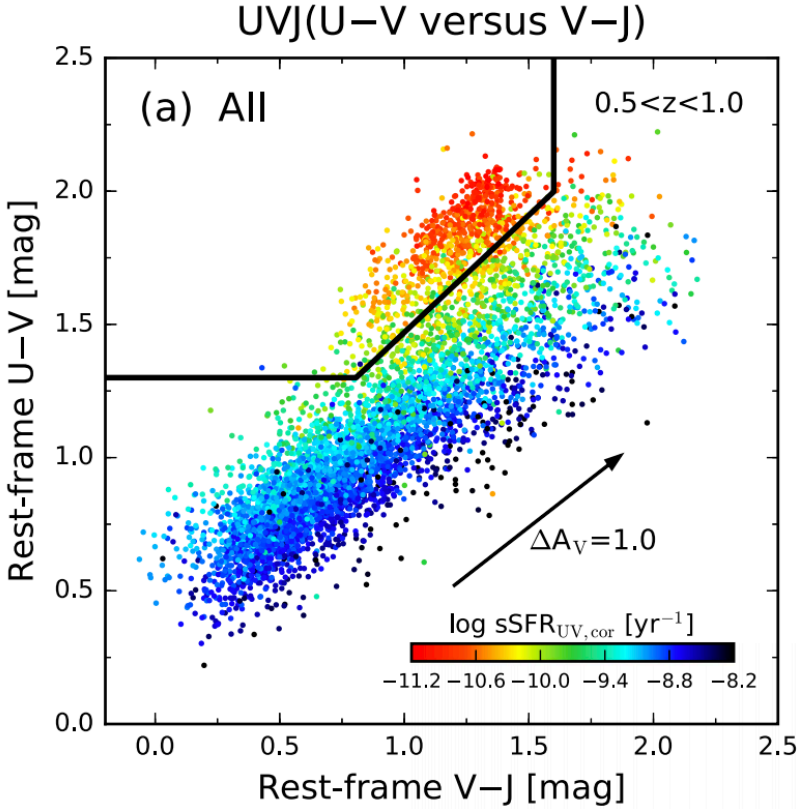
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

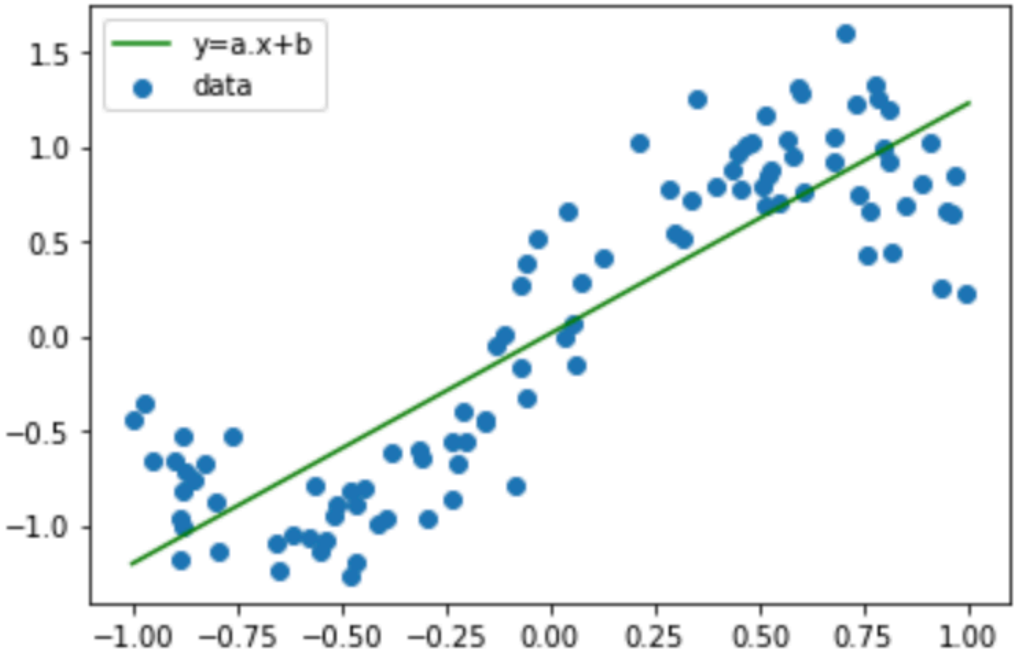
Liu+18

Supervised Machine Learning:

$$f_W(\vec{x}) = \vec{y}$$

If y is discrete: **classification**
If y is a real number: **regression**





Liu+18

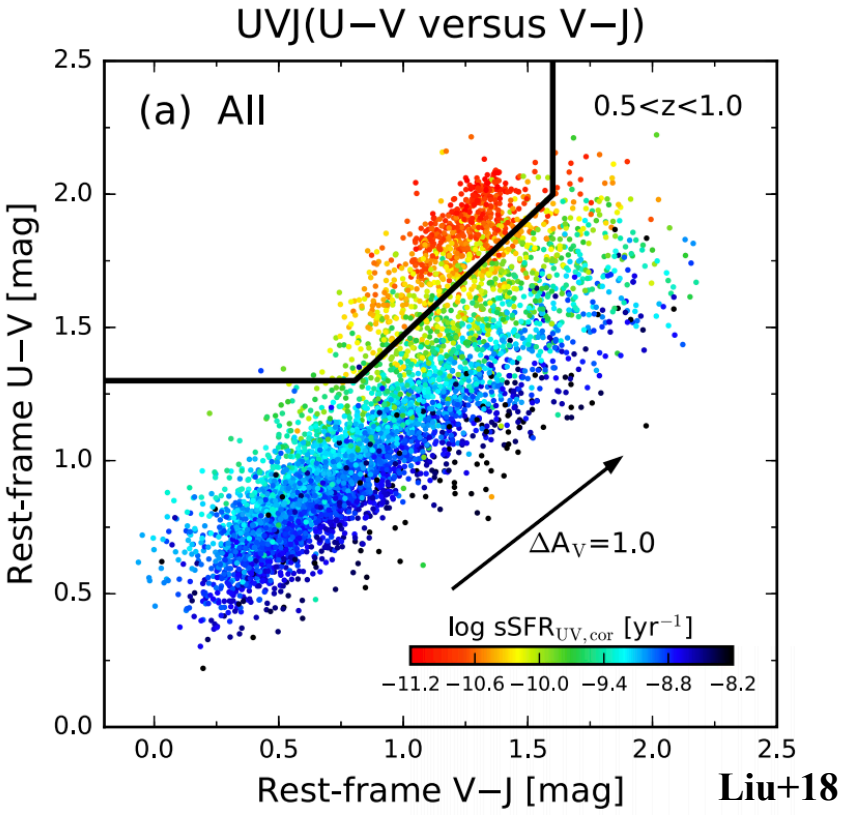
Why data driven?

- No suitable physical model available: **accuracy**
- Physical Model too complex or dataset too large, minimisation difficult: **speed**
- There might be hidden information in the data (beyond usual summary statistics): **discovery**

$$f_W(\vec{x}) = \vec{y}$$

LABEL

Q , SF

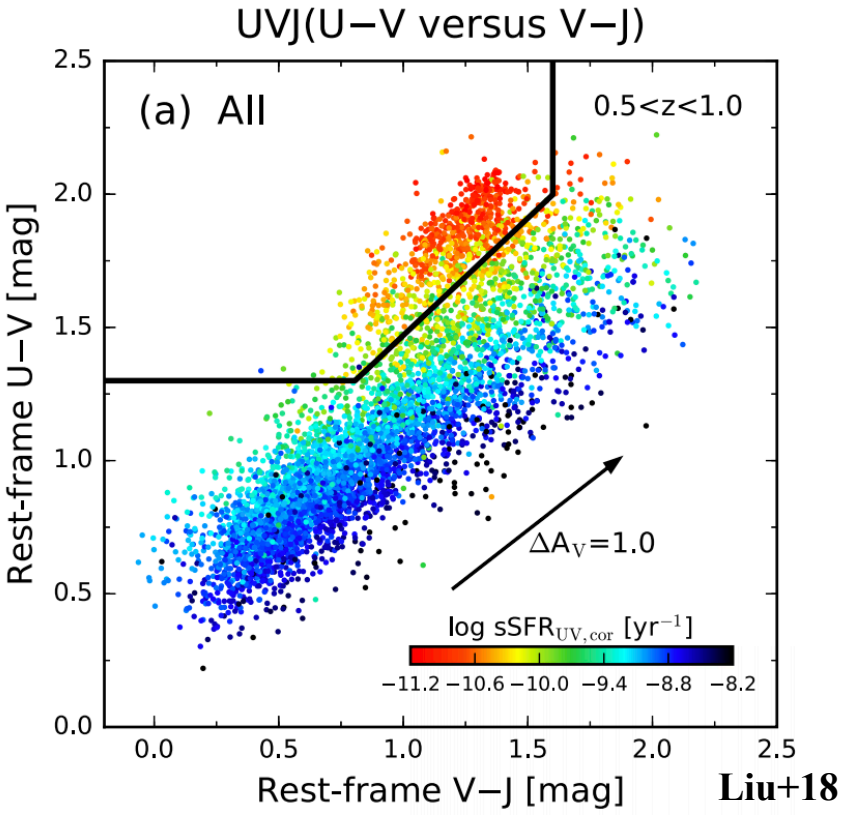


$$f_{\vec{W}}(\vec{x}) = \vec{y}$$

LABEL

Q(0) , SF(1)

(U-V, V-J) FEATURES



Liu+18

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

LABEL

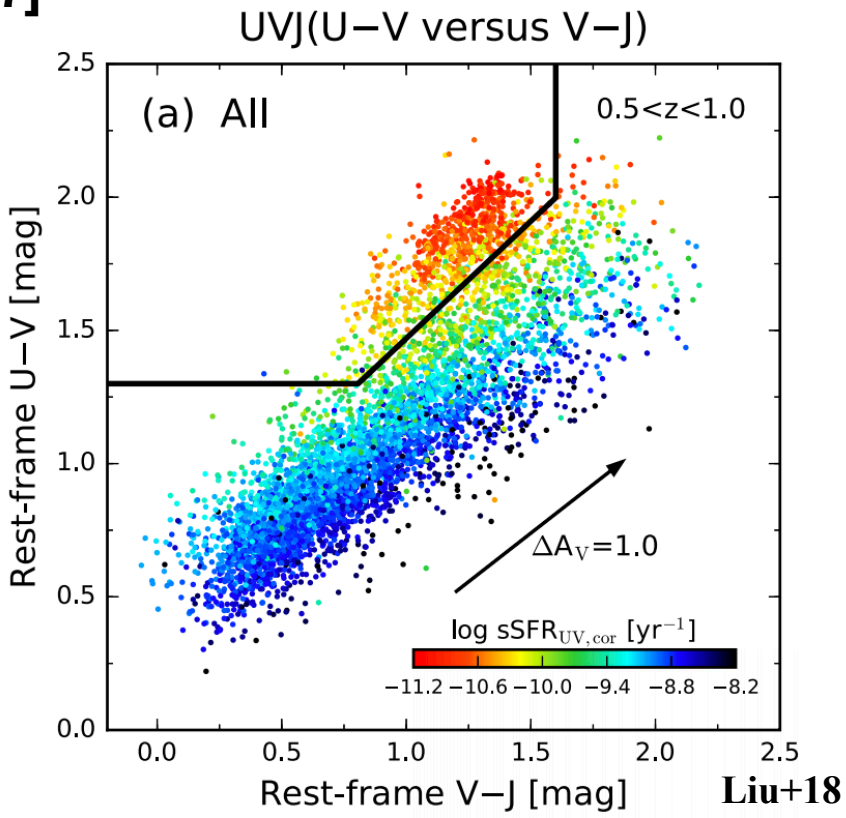
Q(0) , SF(1)

NETWORK FUNCTION

(U-V, V-J) FEATURES

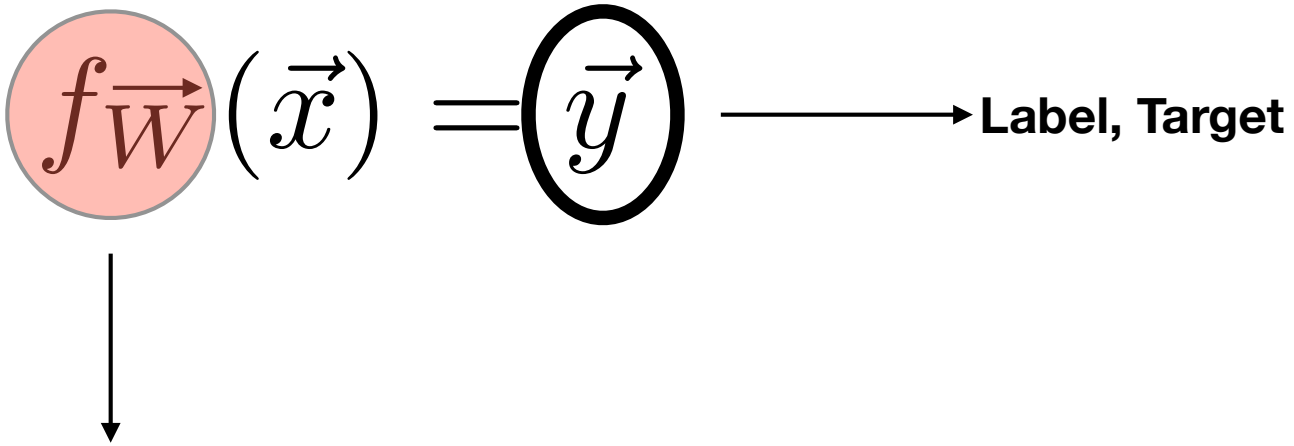
sgn[(u-v)-0.8*(v-j)-0.7]

WEIGHTS



Liu+18

“CLASSICAL” MACHINE LEARNING



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

SUPERVISED LEARNING

Given a dataset with known labels - find a function that can assign (predict) labels for an unlabelled dataset using a set of features (measurements)

Training set

$$(\vec{x_1}, \vec{x_2}, \vec{x_3}, \ldots, \vec{x_n})$$

Features
(colors, fluxes, spectra indices…)

$$(\vec{y_1}, \vec{y_2}, \vec{y_3}, \ldots, \vec{y_n})$$

Label
(morphology, object type, redshift …)

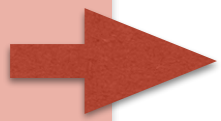
SUPERVISED LEARNING

Given a dataset with known labels - find a function that can assign (predict) labels for an unlabelled dataset using a set of features (measurements)

Training set

$$(\vec{x_1}, \vec{x_2}, \vec{x_3}, \ldots, \vec{x_n})$$

$$(\vec{y_1}, \vec{y_2}, \vec{y_3}, \ldots, \vec{y_n})$$



$$f_W(\vec{x}) = \vec{y}$$

?

SUPERVISED LEARNING

Unlabelled set

$(\vec{x_1}', \vec{x_2}', \vec{x_3}', ..., \vec{x_n}')$

$(\vec{y_1}', \vec{y_2}', \vec{y_3}', ..., \vec{y_n}')$

Training set

$(\vec{x_1}, \vec{x_2}, \vec{x_3}, ..., \vec{x_n})$

$(\vec{y_1}, \vec{y_2}, \vec{y_3}, ..., \vec{y_n})$

$f_W(\vec{x}) = \vec{y}$

?

$$(\vec{x_1}, \vec{x_2}, \vec{x_3}, \ldots, \vec{x_n}) \qquad \vec{x} \in \mathbb{R}^d$$

$$(\vec{y_1}, \vec{y_2}, \vec{y_3}, \ldots, \vec{y_n}) \qquad \vec{y} \in \mathbb{R} \qquad \vec{y} \in \mathbb{N}$$

GENERAL GOAL: Find a (non-linear) function that outputs the correct class / label (y) for a given input object:

It is translated into a minimization problem : find W such as the prediction error is minimal over all unseen vectors

we need two key elements

1. **A LOSS FUNCTION (OBJECTIVE FUNCTION TO MINIMISE)**

2. **A MINIMISATION OR OPTIMISATION ALGORITHM**

we need two key elements

1. A LOSS FUNCTION
(Objective to minimize)

2. A MINIMISATION OR OPTIMISATION ALGORITHM

THIS IS COMMON TO ALL MACHINE LEARNING ALGORITHMS

FOR EXAMPLE:

1. DEFINE A LOSS FUNCTION

$$loss(F_W(.), \vec{x_i}, \vec{y_i})$$

For example: $(F_W(\vec{x_i}) - \vec{y_i})^2$ (MSE LOSS FUNCTION)

2. MINIMISE THE EMPIRICAL RISK WITH OPTIMISATION

$$\Re_{empirical}(W) = \frac{1}{N} \sum_{i}^{N} [loss(W, \vec{x}, \vec{y})]$$

MINIMISE THE RISK

EMPIRICAL RISK?

$$\Re_{empirical}(W) = \frac{1}{N}\sum_{i}^{N}[loss(W, \vec{x}, \vec{y})]$$

WE ARE MINIMISING WITH RESPECT TO A FINITE NUMBER OF OBSERVED EXAMPLES

Empirical Risk

$$\Re_{empirical}(W) = \frac{1}{N}\sum_{i}^{N}[loss(W, \vec{x}, \vec{y})]$$

WE ARE MINIMISING WITH RESPECT TO A FINITE NUMBER OF OBSERVED EXAMPLES

OBSERVED DATASET

Empirical Risk

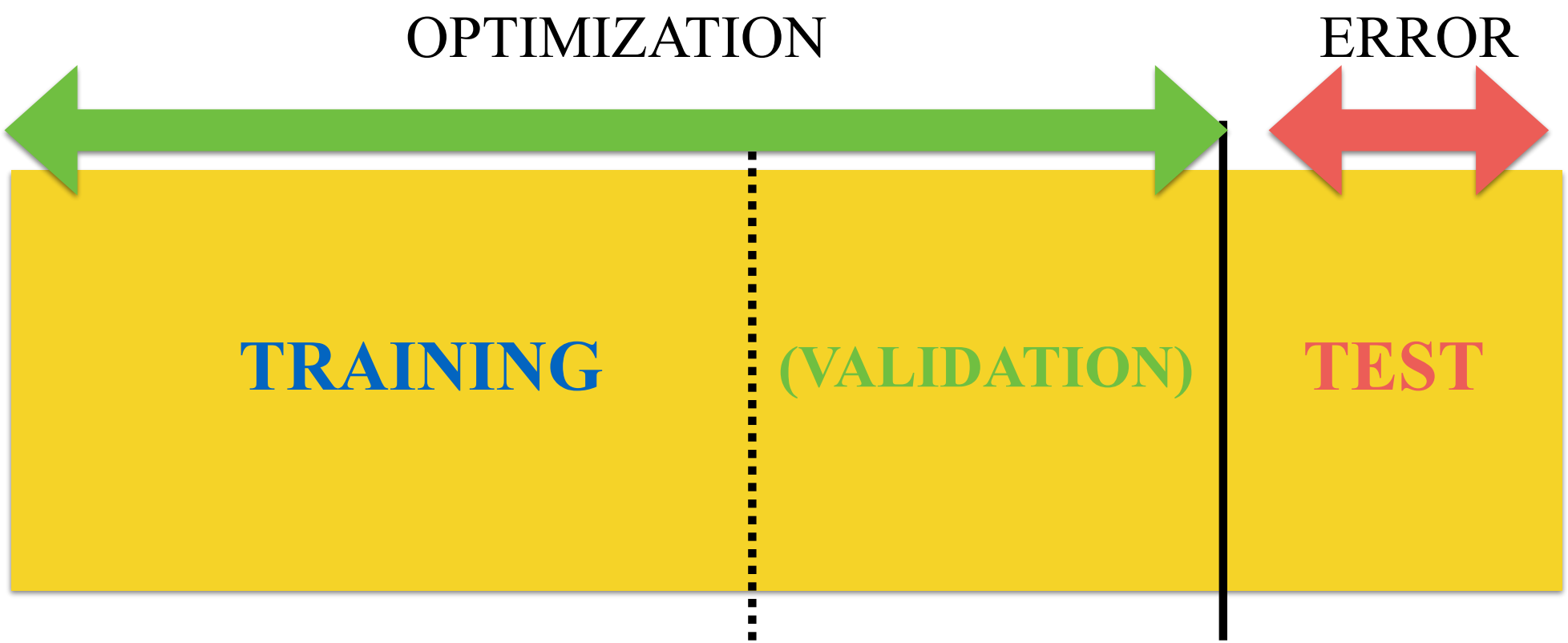
$$\Re_{empirical}(W) = \frac{1}{N}\sum_{i}^{N}[loss(W, \vec{x}, \vec{y})]$$

WE ARE MINIMISING WITH RESPECT TO A FINITE NUMBER OF OBSERVED EXAMPLES

ALL “GALAXIES IN THE UNIVERSE”

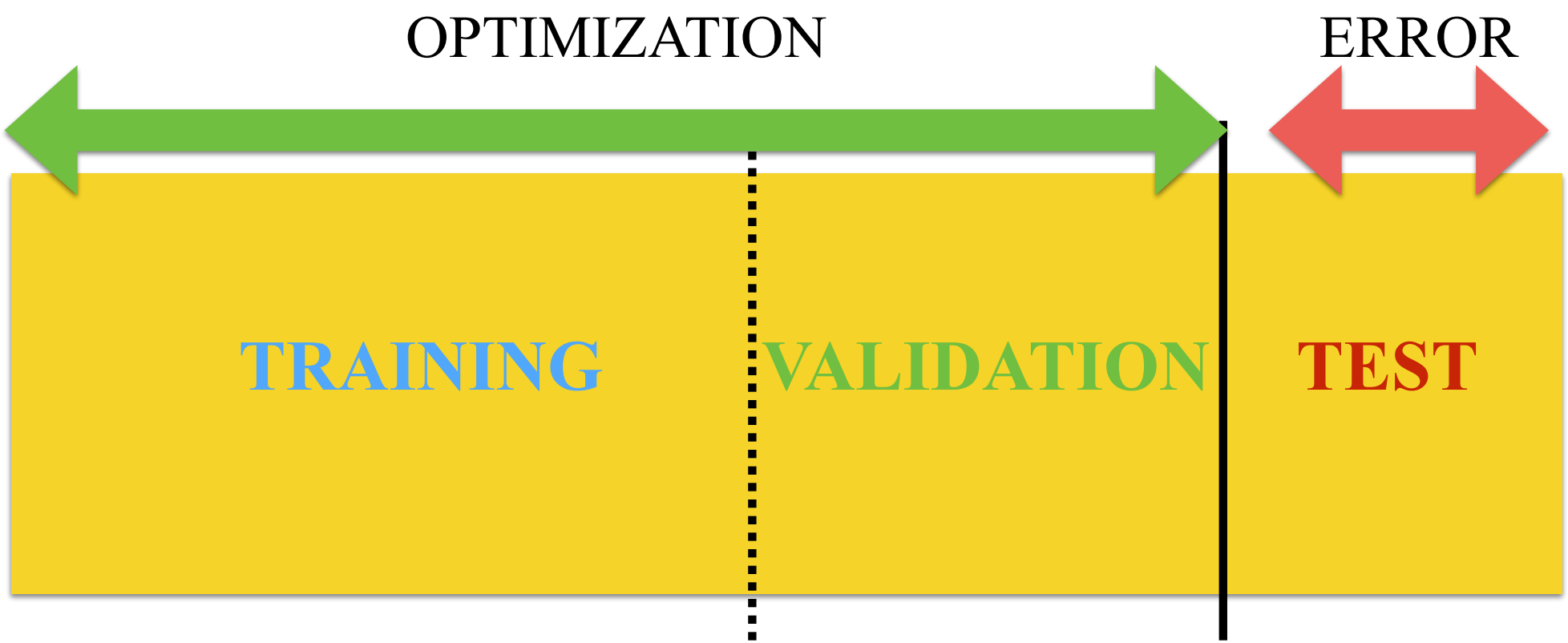
OBSERVED DATASET

In Practice



training set: use to train the classifier
validation set: use to monitor performance in real time - check for overfitting
test set: use to train the classifier

IN PRACTICE



NO CHEATING! NEVER USE TRAINING TO VALIDATE YOUR ALGORITHM!

The algorithm used to minimise is called OPTIMIzATION

THERE ARE SEVERAL OPTIMISATION TECHNIQUES

Optimisation

THERE ARE SEVERAL OPTIMISATION TECHNIQUES

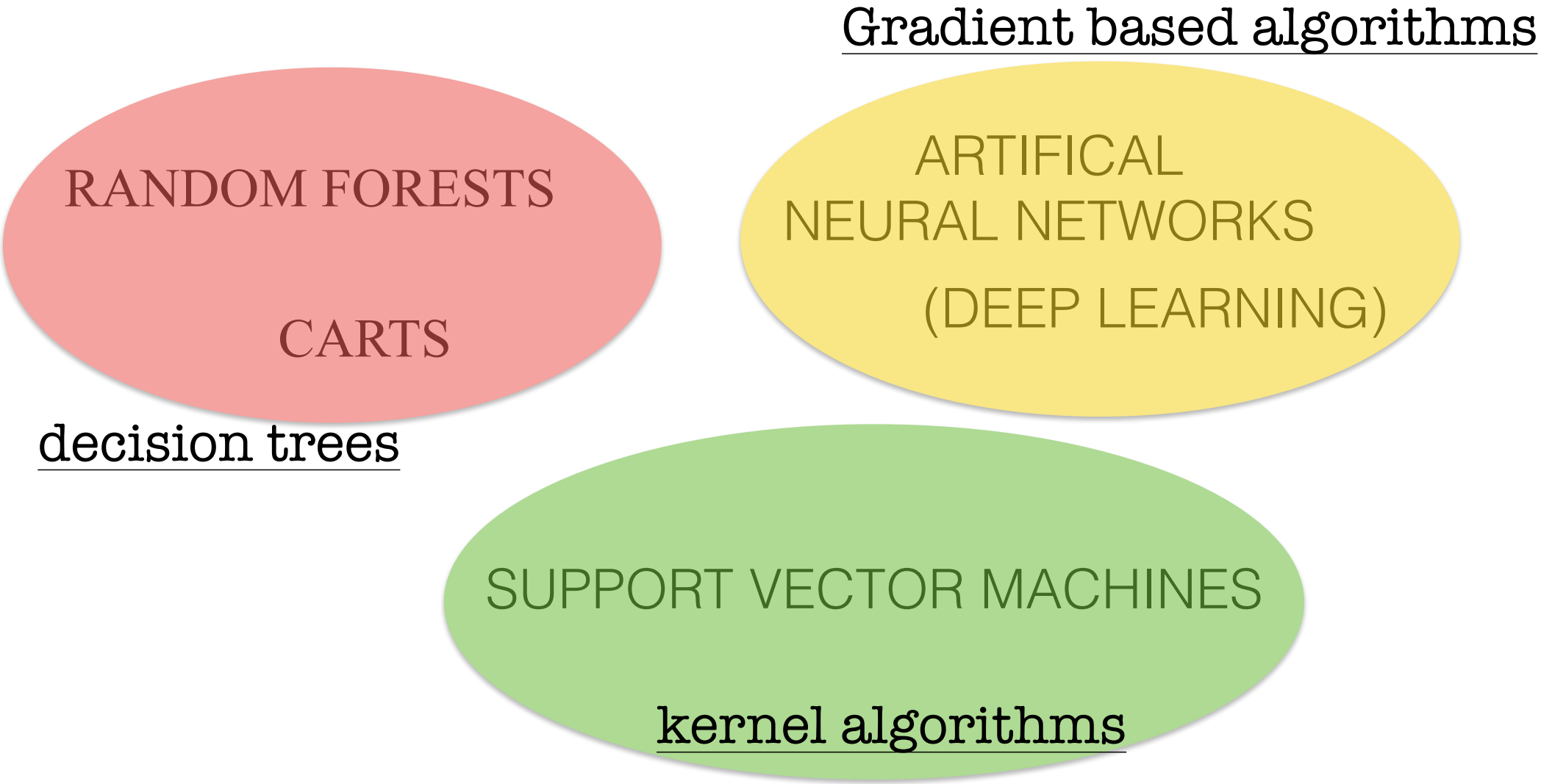
THEY DEPEND ON THE MACHINE LEARNING ALGORITHM

Optimisation

THERE ARE SEVERAL OPTIMISATION TECHNIQUES

THEY DEPEND ON THE MACHINE LEARNING ALGORITHM
(* See Gini minimisation from Viviana's lectures)

Different types of supervised machine learning methods



The differences are
in the function
that is used, which sets
the optimisation and loss

$$f_W(\vec{x})$$

Gradient based algorithms

RANDOM FORESTS

CARTS

decision trees

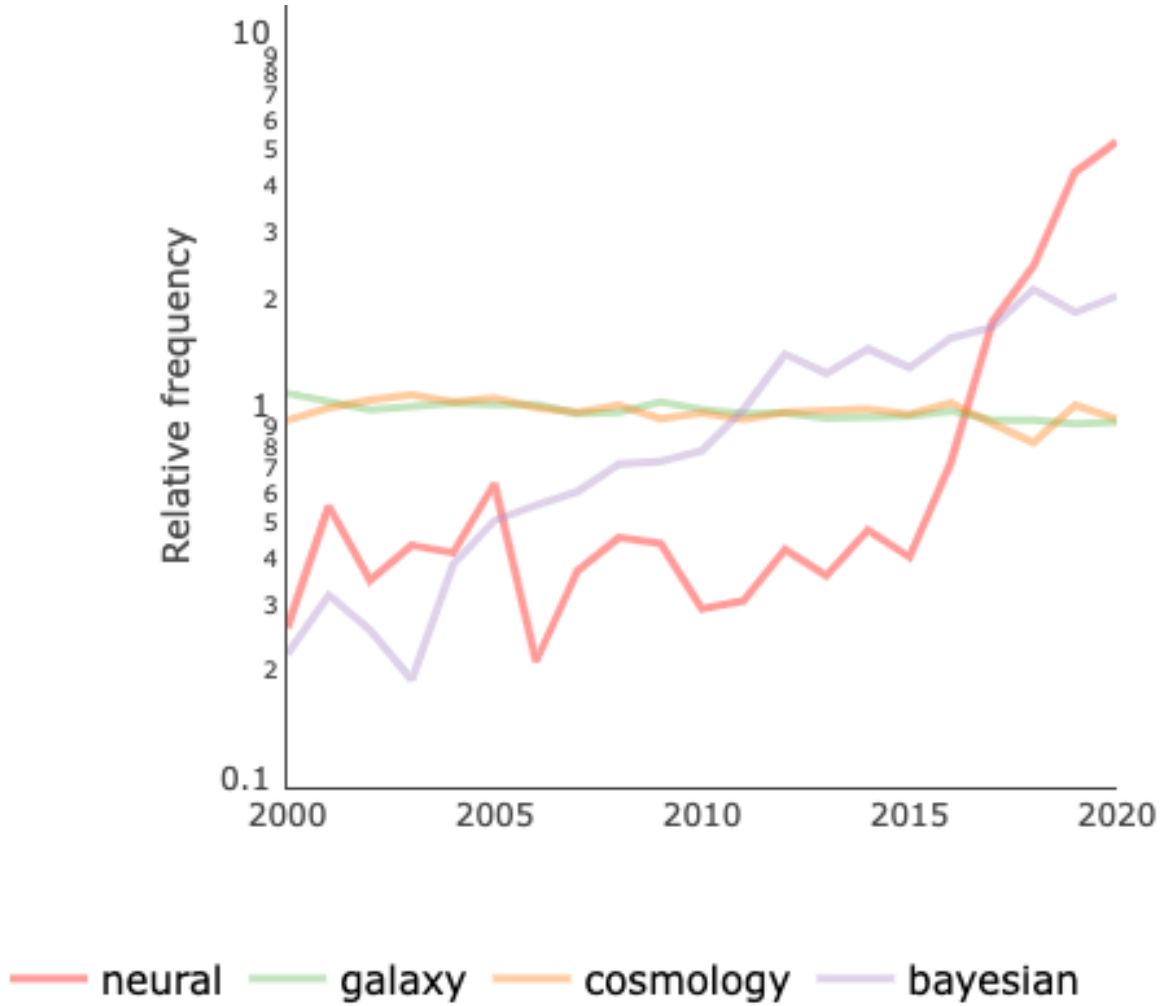
ARTIFICAL
NEURAL NETWORKS

(DEEP LEARNING)

SUPPORT VECTOR MACHINES

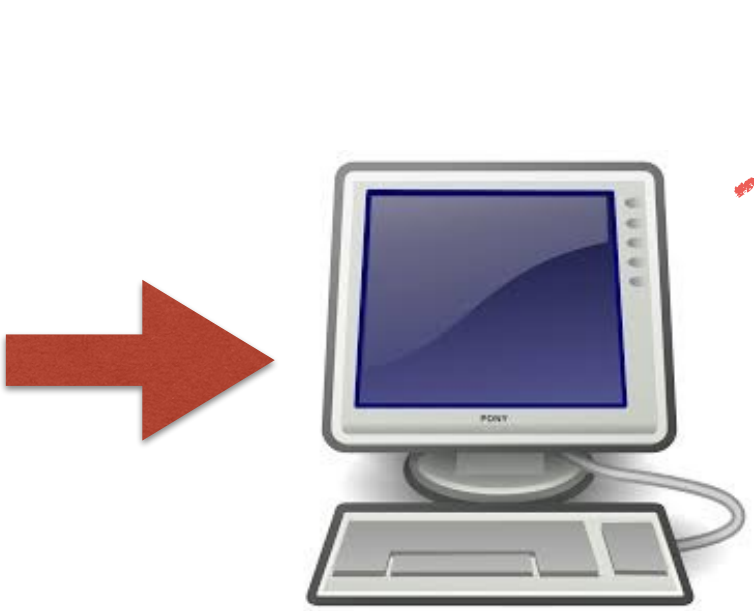
kernel algorithms

Why are neural networks special?



Huertas-Company+23

Before 2012

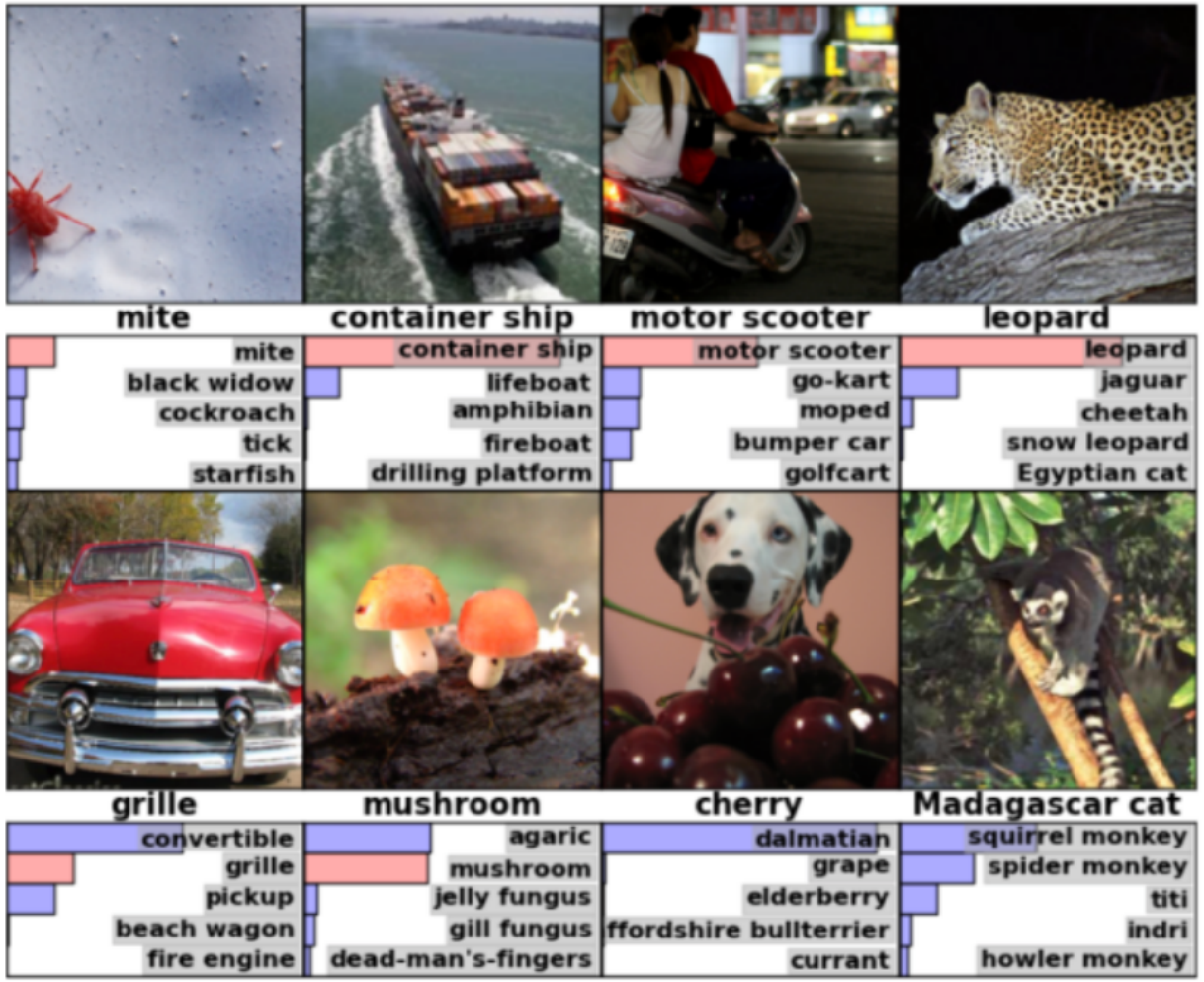


CAT?

DOG?

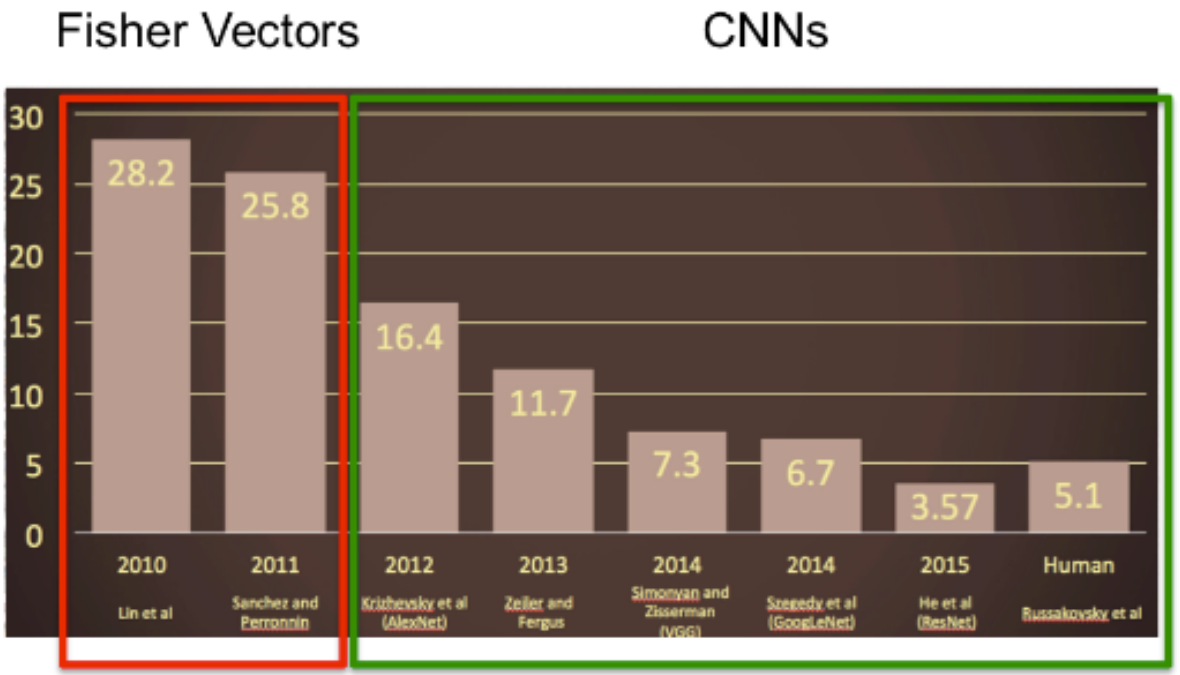
TRIVIAL HUMAN TASKS REMAINED CHALLENGING FOR COMPUTERS

After 2012



IT HAS BECOME TRIVIAL….

The end of the ML winter…



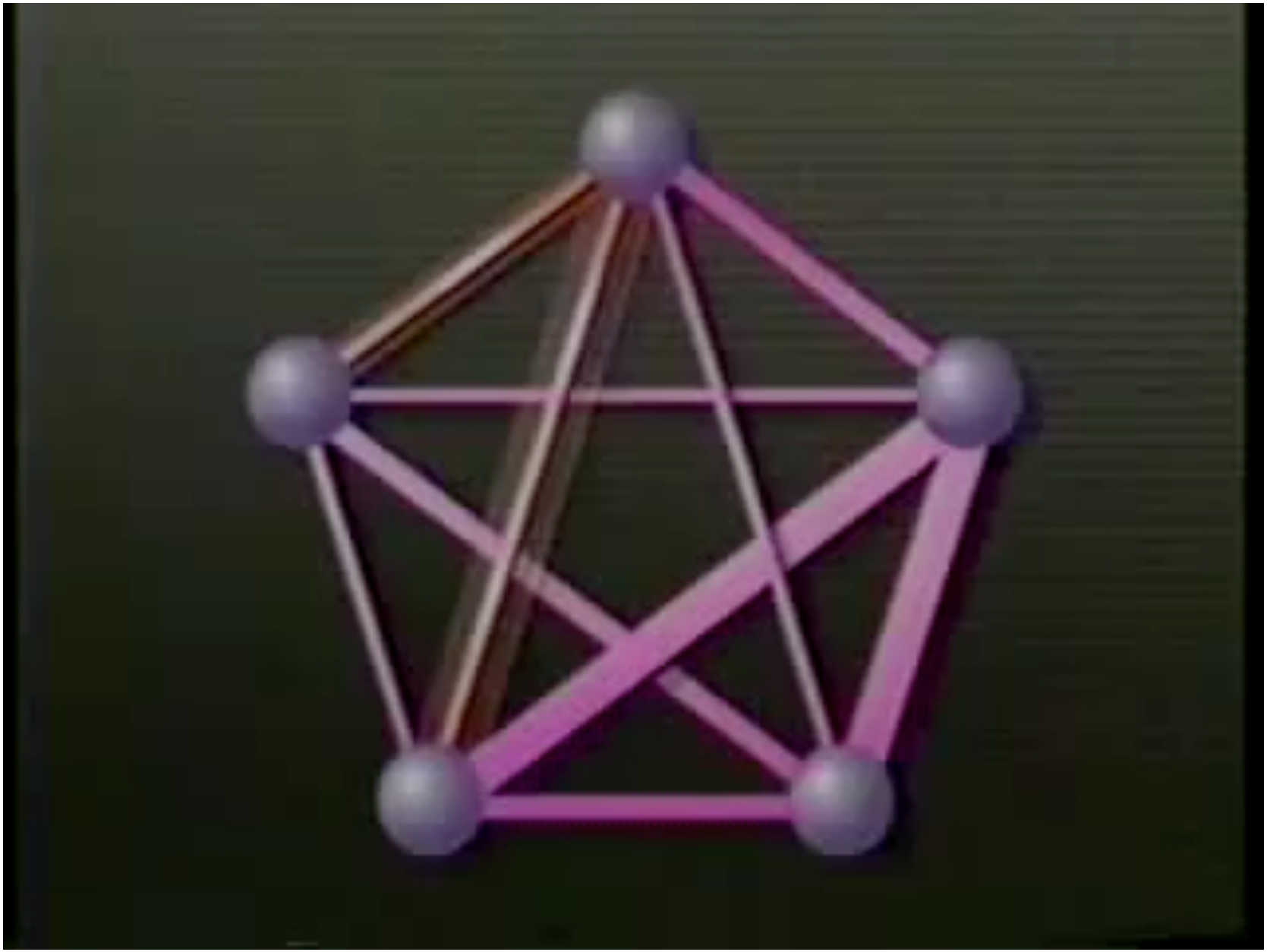
ImageNet
top-5 error (%)

deep learning

Tentative Plan

- Lecture 1: Intro1 - foundations of NNs
- Lecture 2: Intro2 - NNs as statistical models
- Lectures 3 and 4: NNs for computer vision
- Lecture 5: NNs for sequences
- Lecture 6: NNs for graphs
- Lectures 7 and 8: unsupervised deep learning

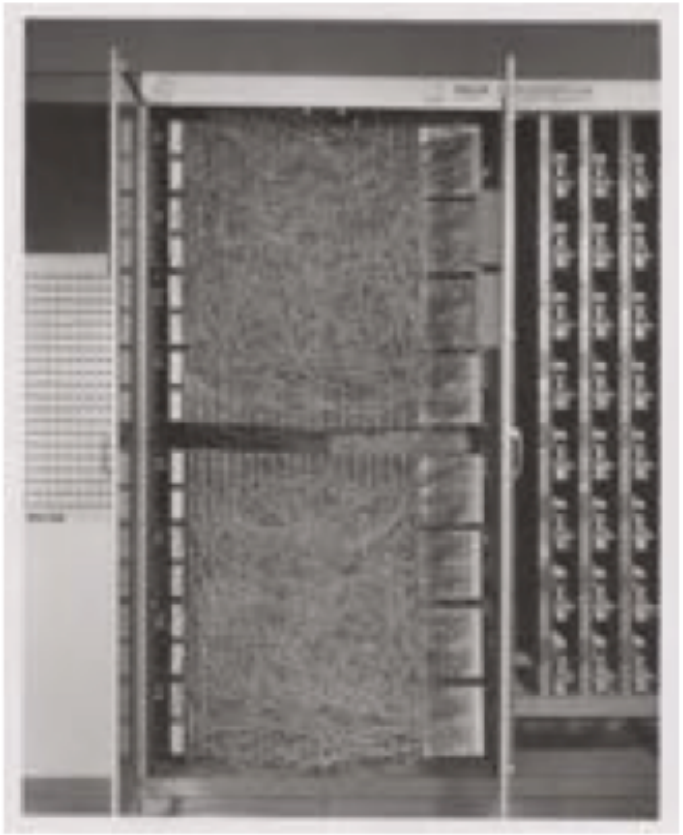
Foundation of dense neural networks



Rosenblatt Perceptron

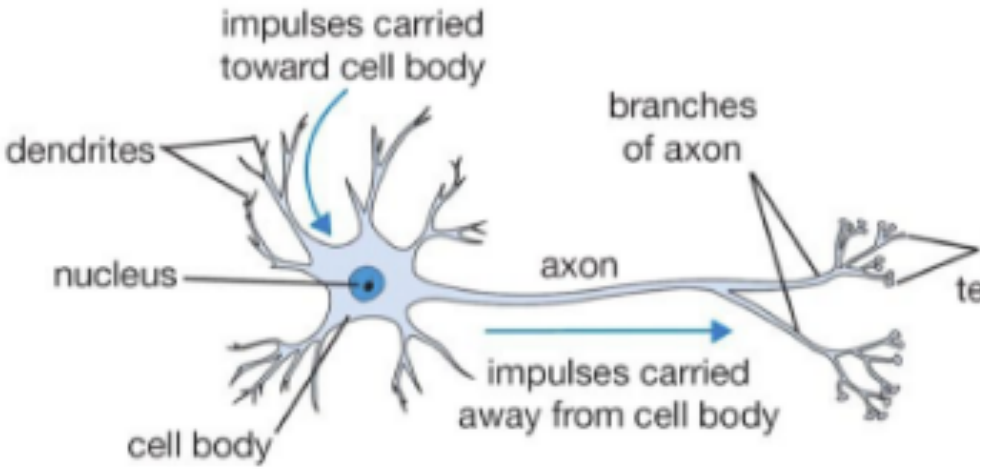
First implementation of a neural network [Rosenblatt, **1957!**]

INTENDED TO BE A MACHINE (NOT AN ALGORITHM)



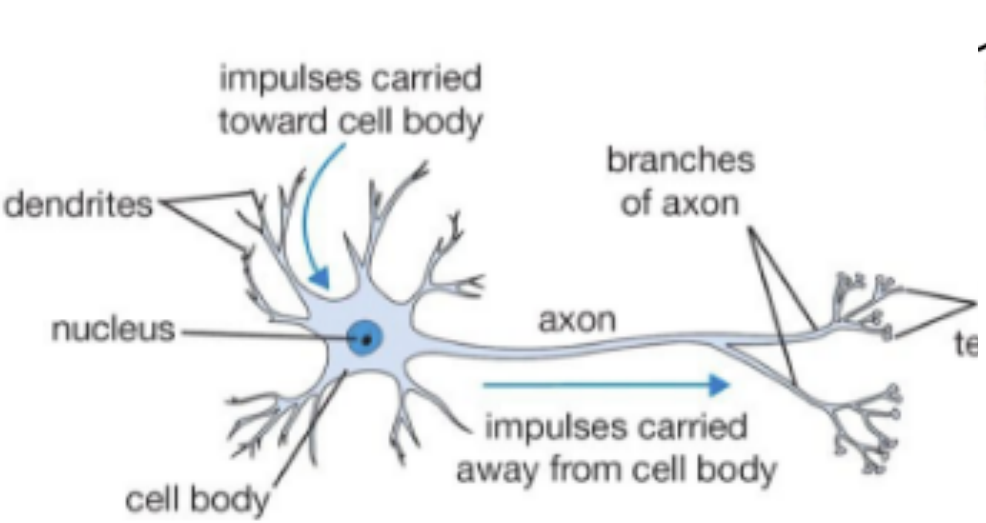
it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors

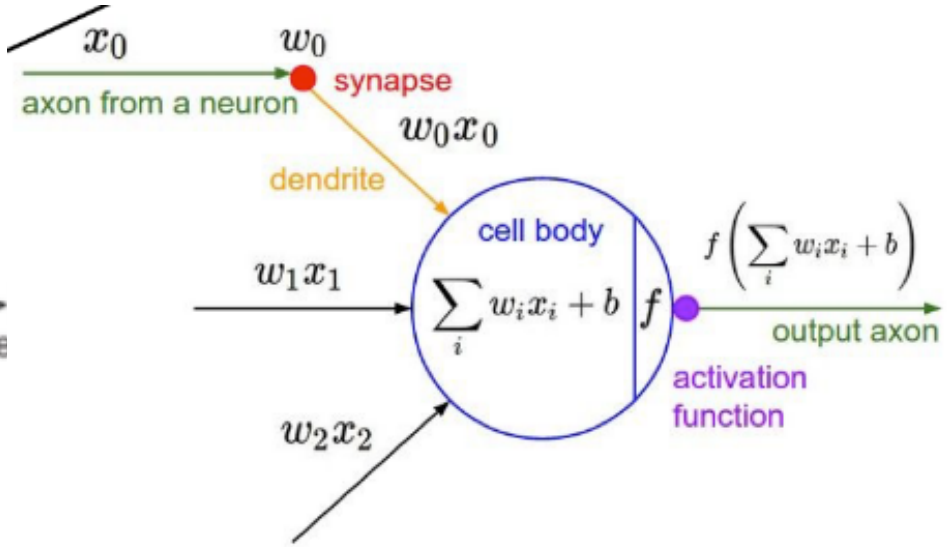
Inspired by neuro-science…



Credit: Karpathy

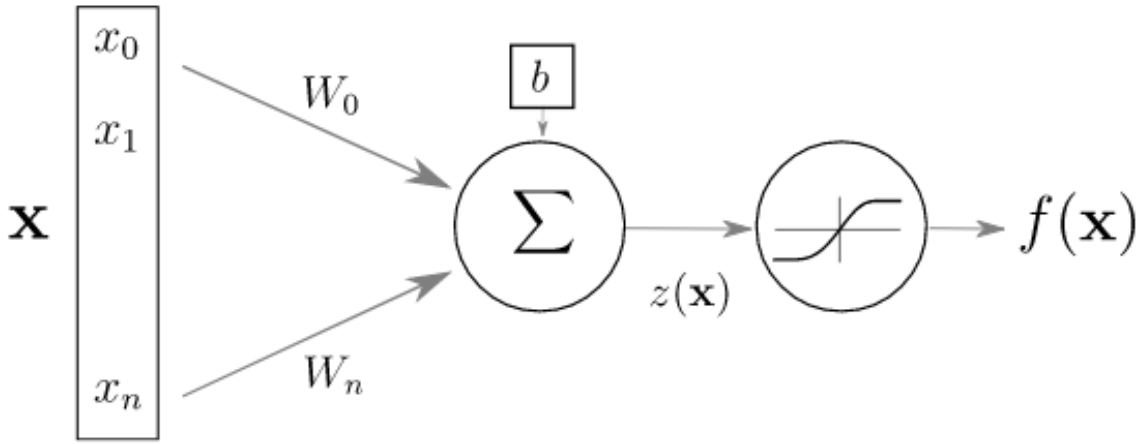
Inspired by neuro-science…





Credit: Karpathy

Artificial neuron (or unit or perceptron)



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

Pre-Activation

Weights

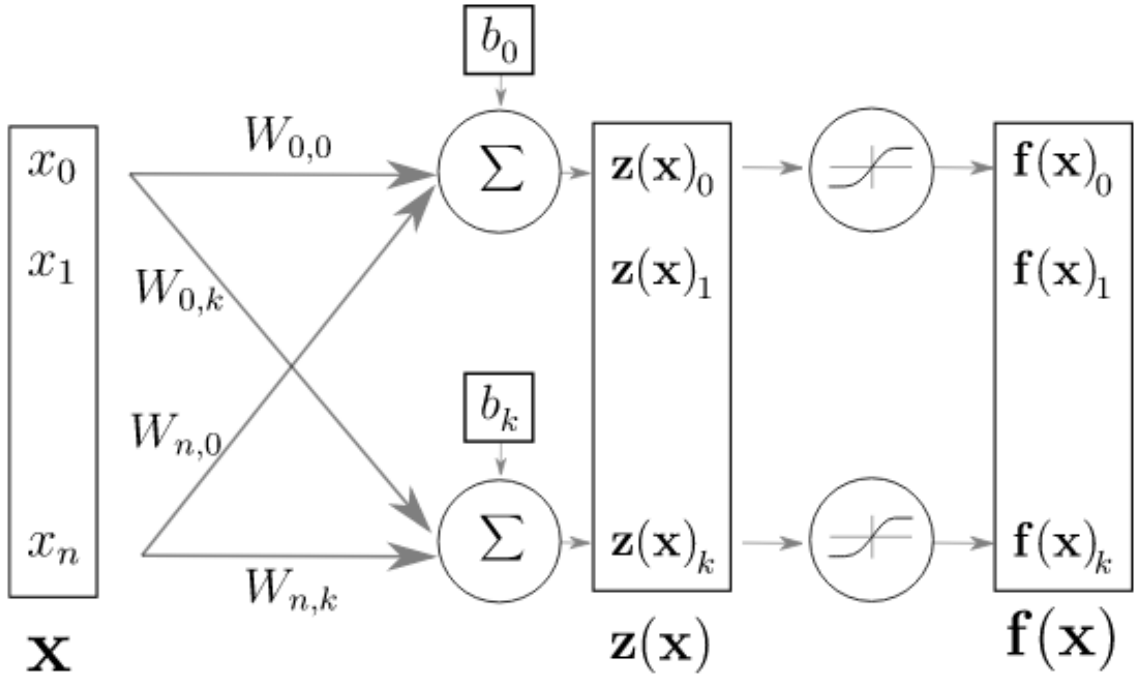
Input

Bias

Output

Activation Function

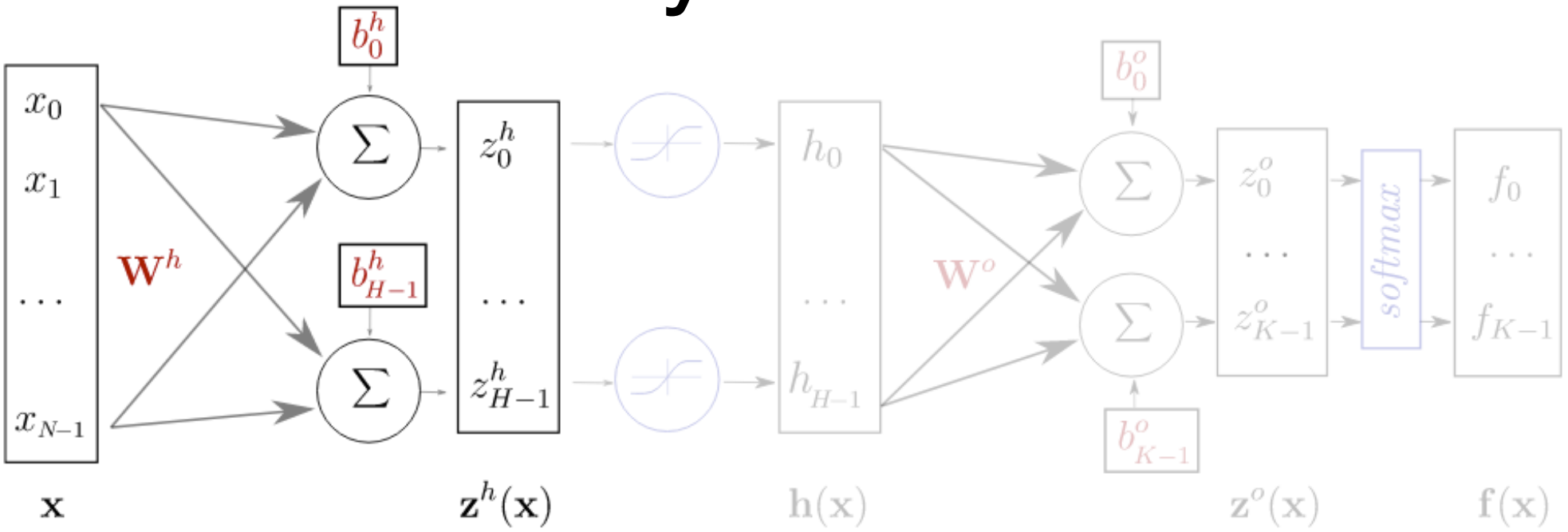
Layer of neurons



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

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

Hidden Layers of Neurons

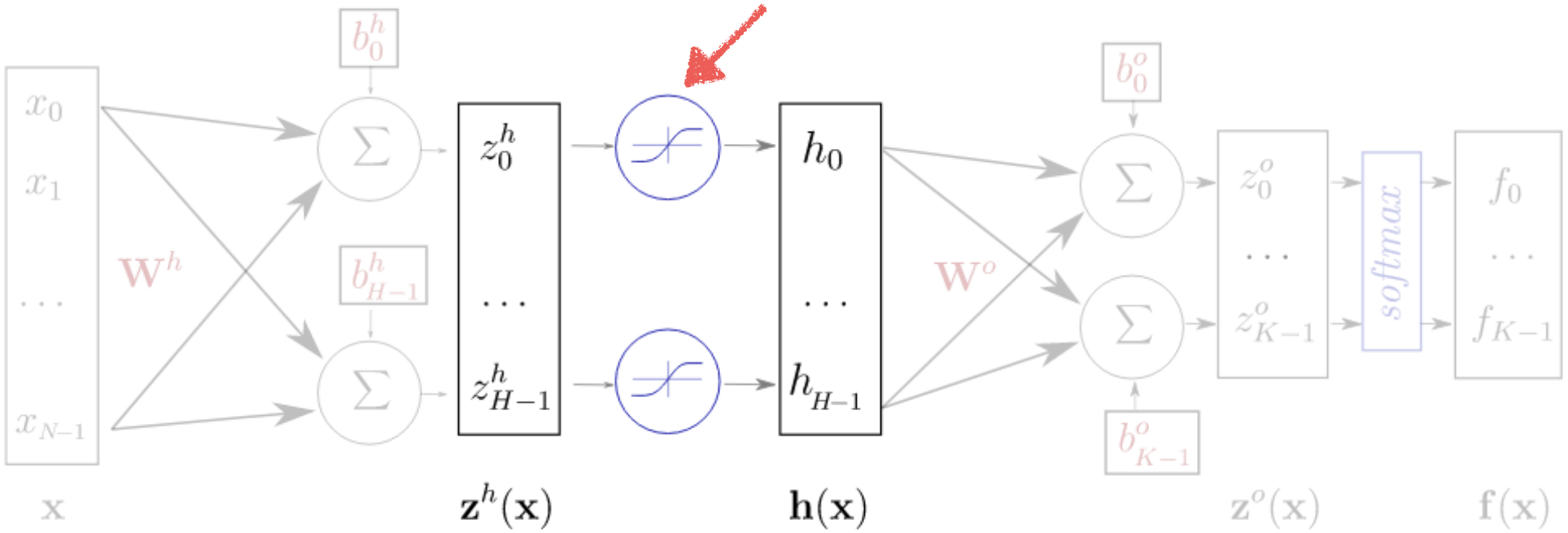


FIRST LAYER

INPUT

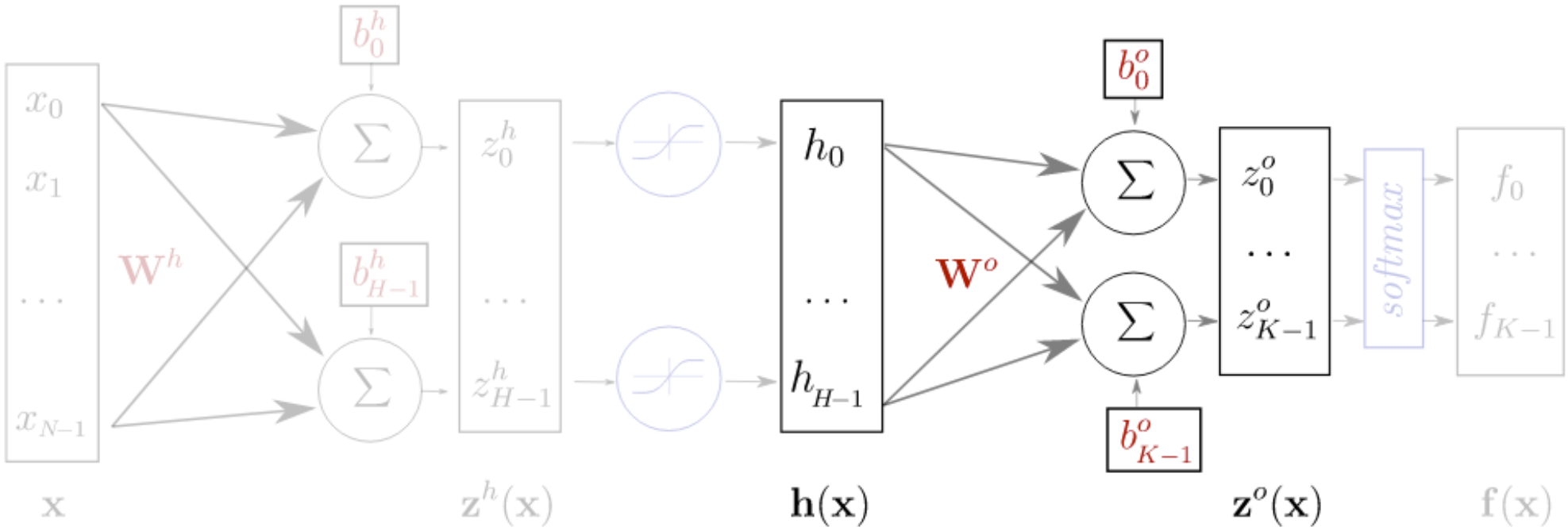
$$z^h(x) = W^h x + b^h$$

ACTIVATION FUNCTION



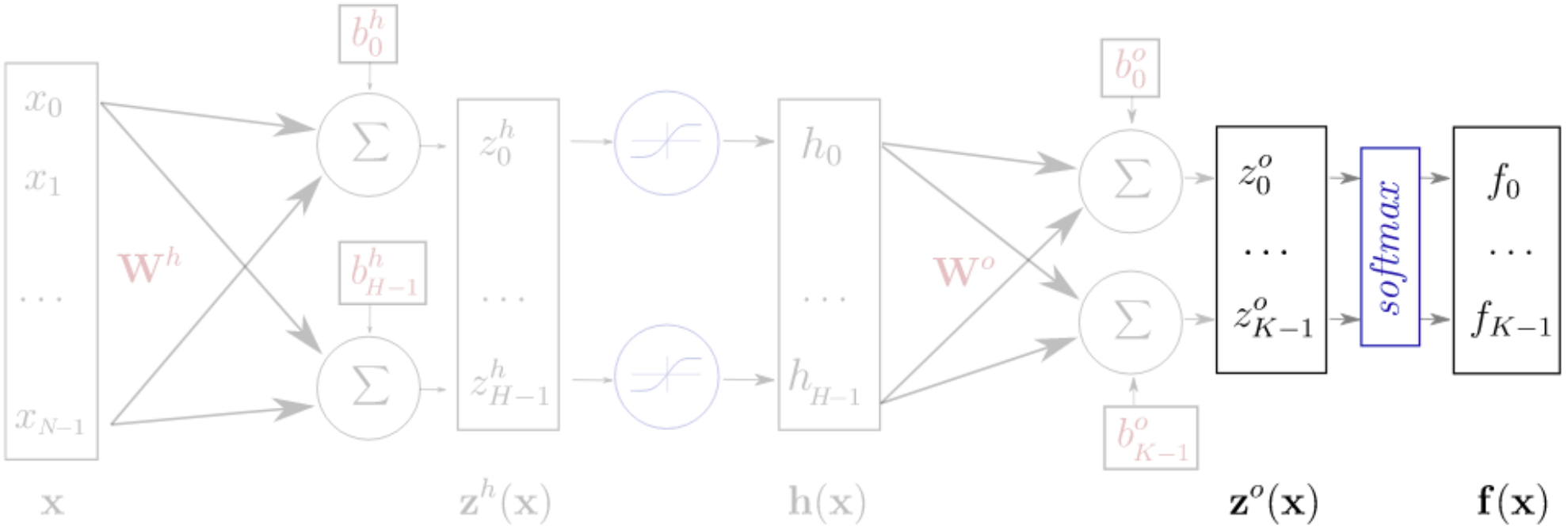
HIDDEN LAYER

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



OUTPUT LAYER

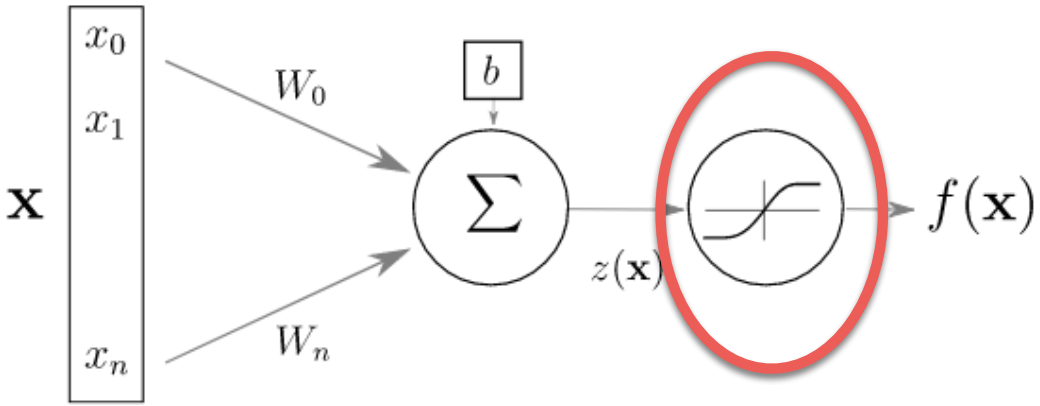
$$z^0(\mathbf{x}) = W^0 h(\mathbf{x}) + b^0$$



PREDICTION LAYER

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

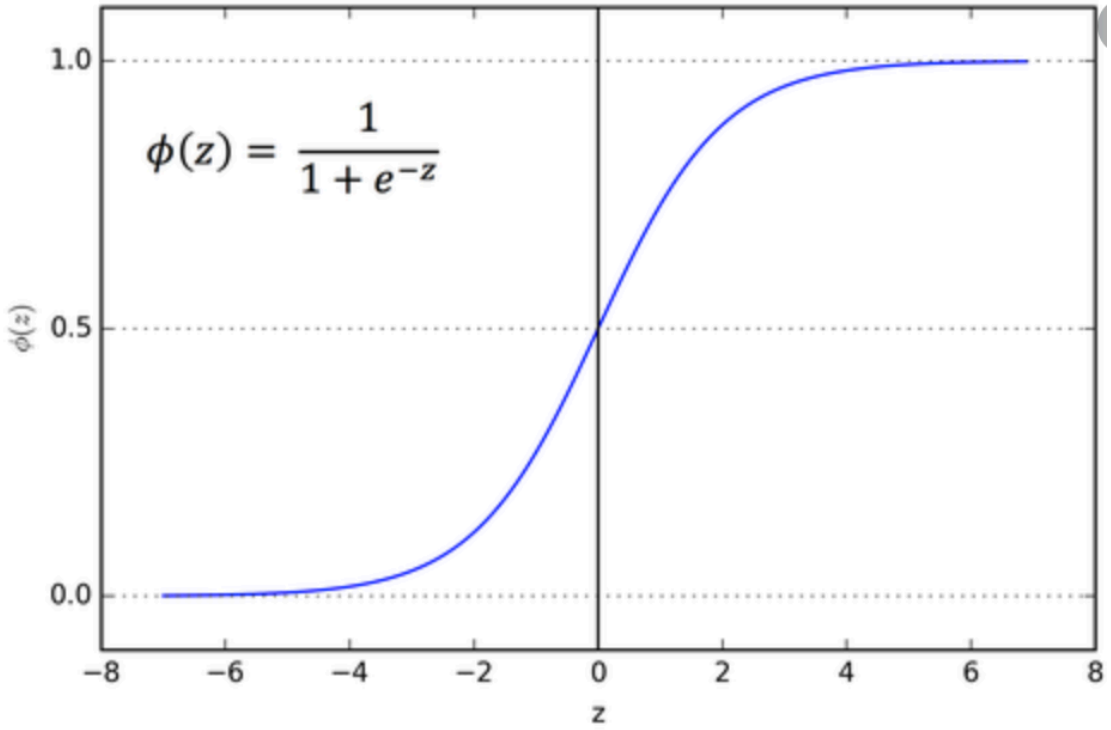
1. ACTIVATION FUNCTIONS



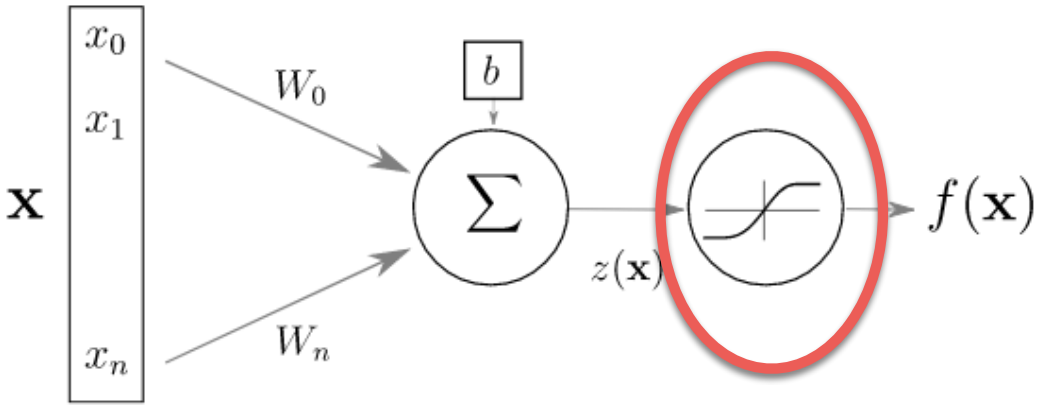
NON TRAINABLE

ADD NON LINEARITIES TO THE PROCESS

1. ACTIVATION FUNCTIONS

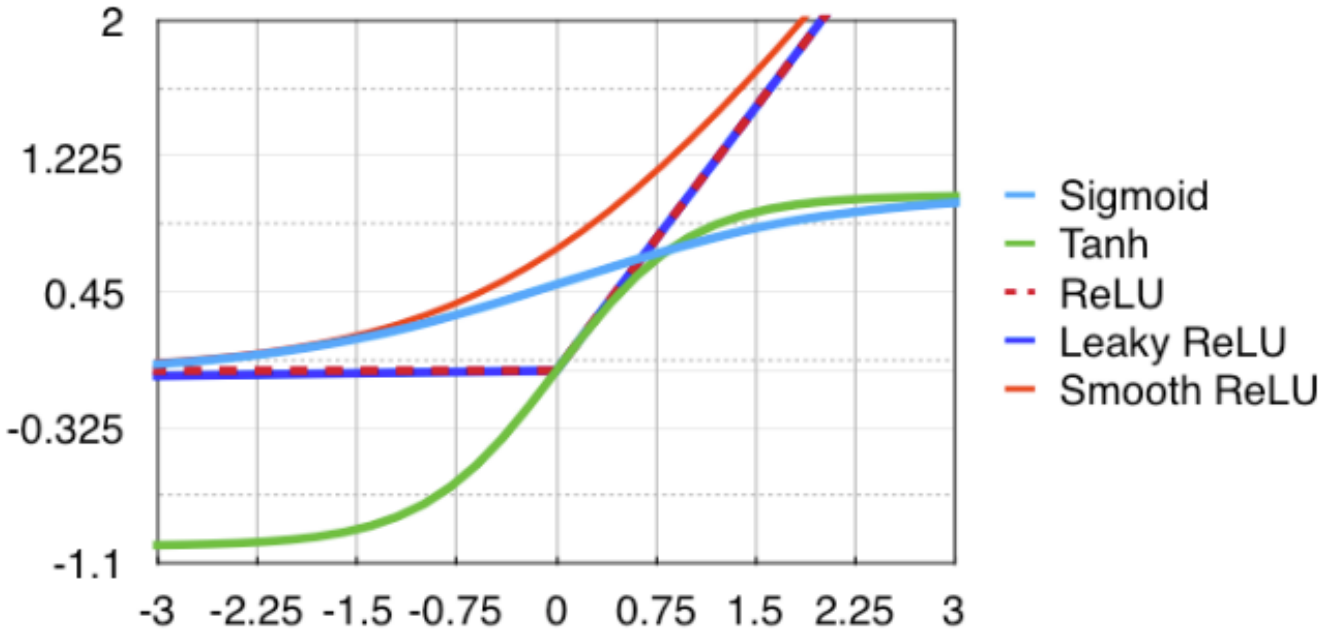


the sigmoid function



ADD NON LINEARITIES TO THE PROCESS

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

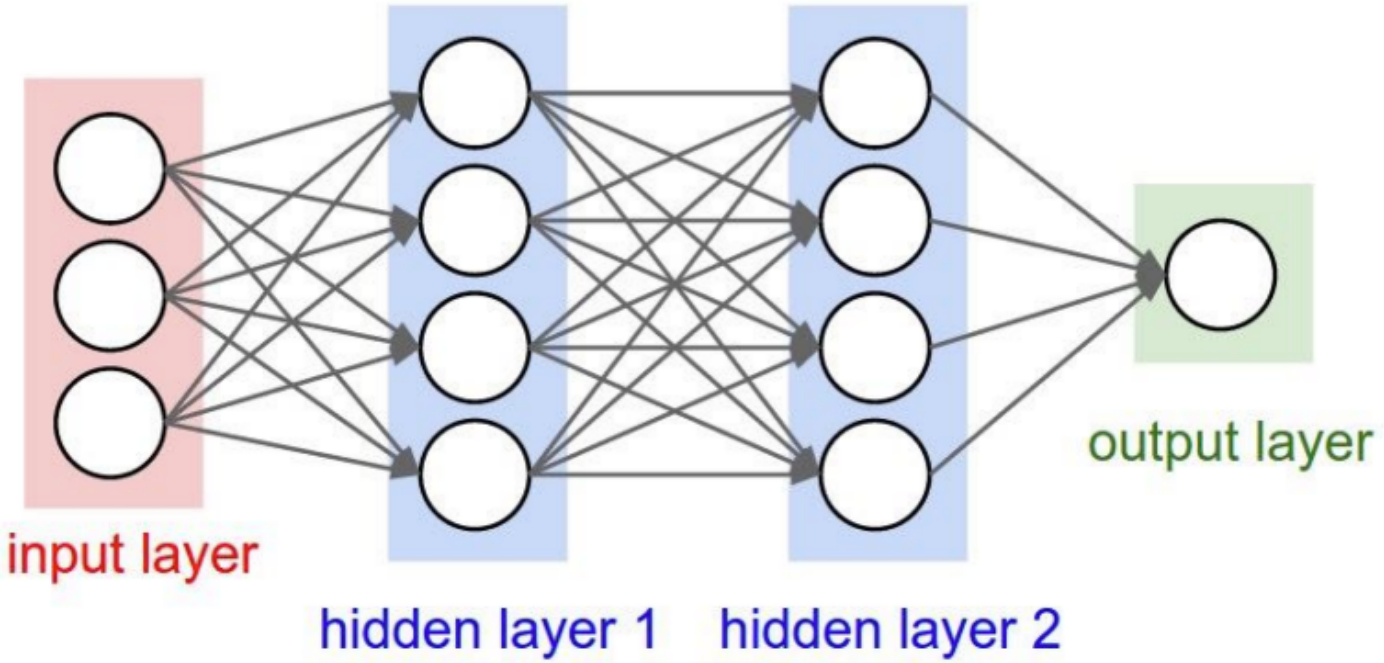
"CLASSICAL" MACHINE LEARNING

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

LABEL

Q , SF

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



$$f_w(x) = g_3(W_3 g_2(W_2 g_1(W_1(x))))$$

NETWORK FUNCTION

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

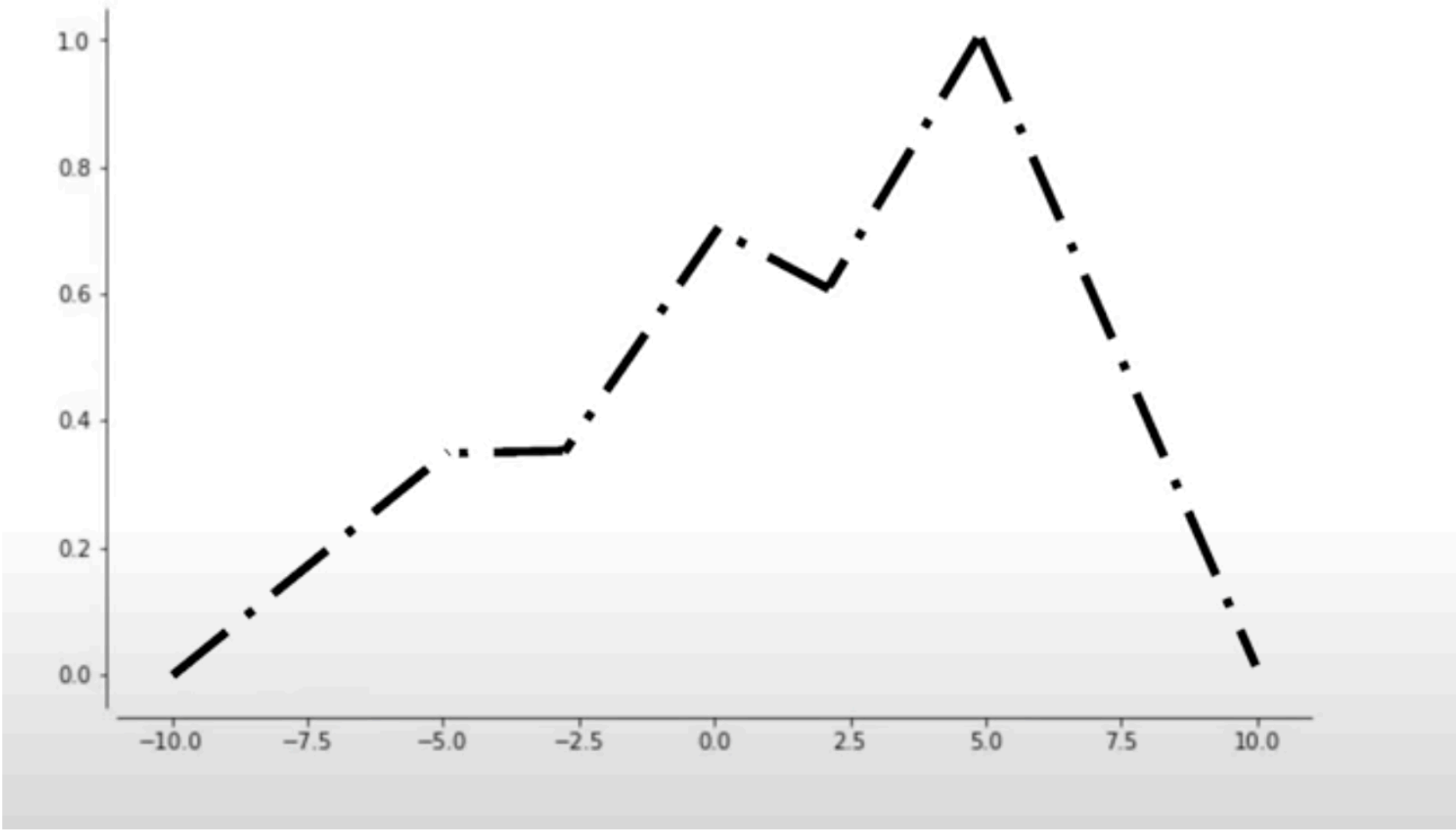
"BIG ENOUGH NETWORK CAN APPROXIMATE, BUT NOT REPRESENT ANY SMOOTH FUNCTION. THE MATH DEMONSTRATION IMPLIES SHOWING THAT NETWORS ARE DENSE IN THE SPACE OF TARGET FUNCTIONS"

UNIVERSAL APPROXIMATION THEOREM

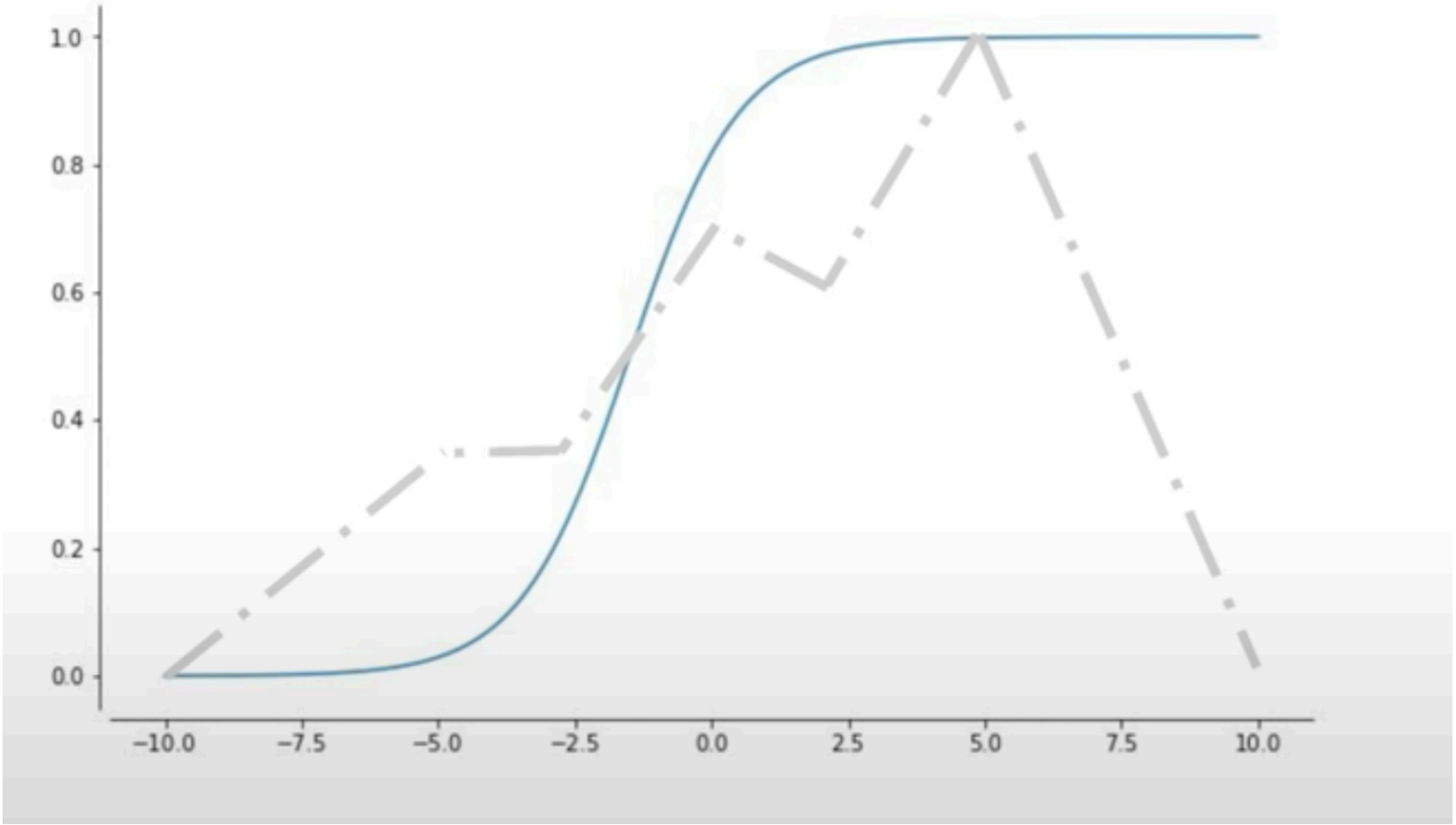
FOR ANY CONTINUOS FUNCTION FOR A HYPERCUBE [0,1]d TO REAL NUMBERS, NON-CONSTANT, BOUNDED AND CONTINUOUS ACTIVATION FUNCTION f, AND EVERY POSITIVE EPSILON, THERE EXISTS A1-HIDDEN LAYER NEURAL NETWORK USING f THAT OBTAINS AT MOST EPSILON ERROR IN FUNCTIONAL SPACE

Horvik+91

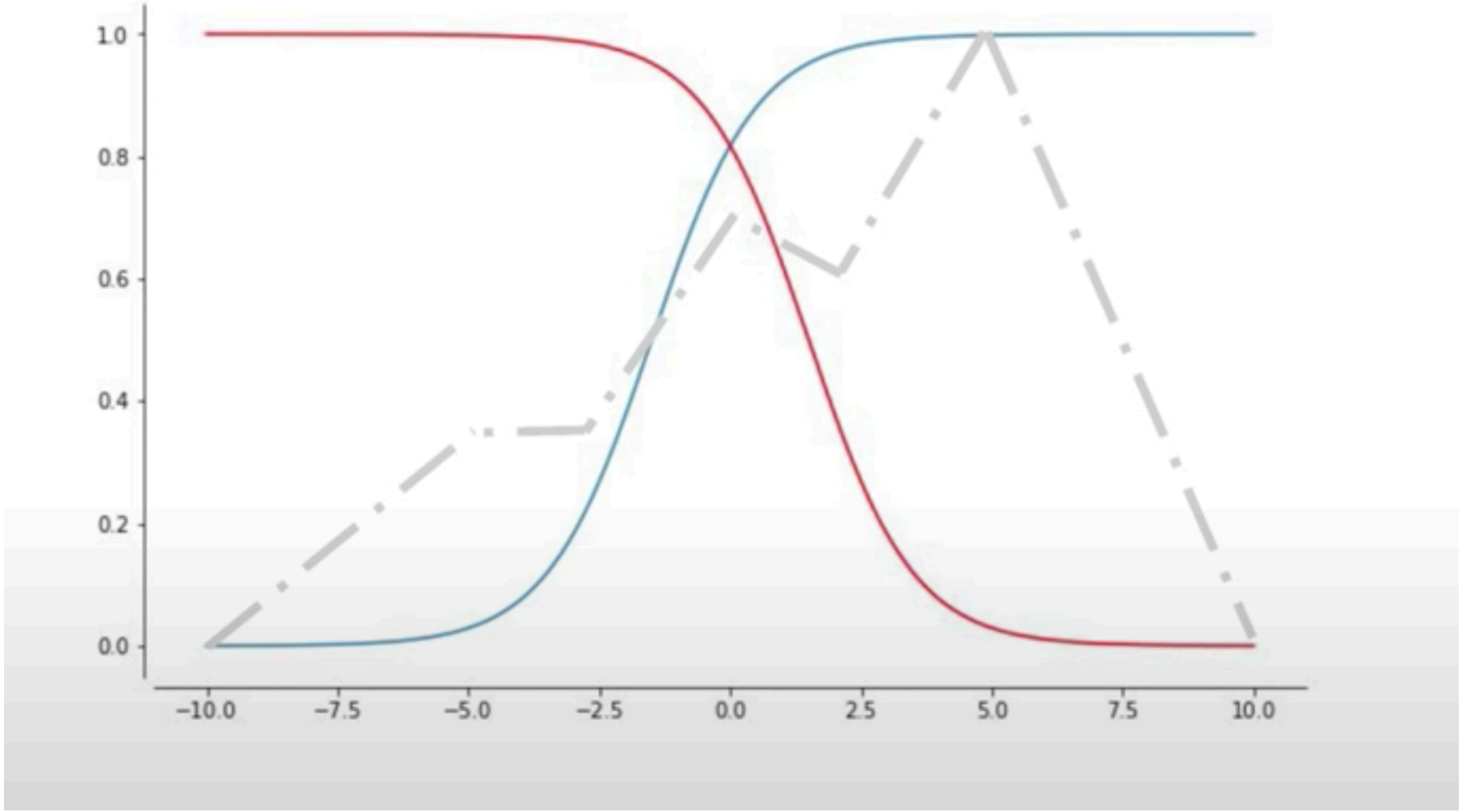
"BIG ENOUGH NETWORK CAN APPROXIMATE, BUT NOT REPRESENT ANY SMOOTH FUNCTION. THE MATH DEMONSTRATION IMPLIES SHOWING THAT NETWORS ARE DENSE IN THE SPACE OF TARGET FUNCTIONS"



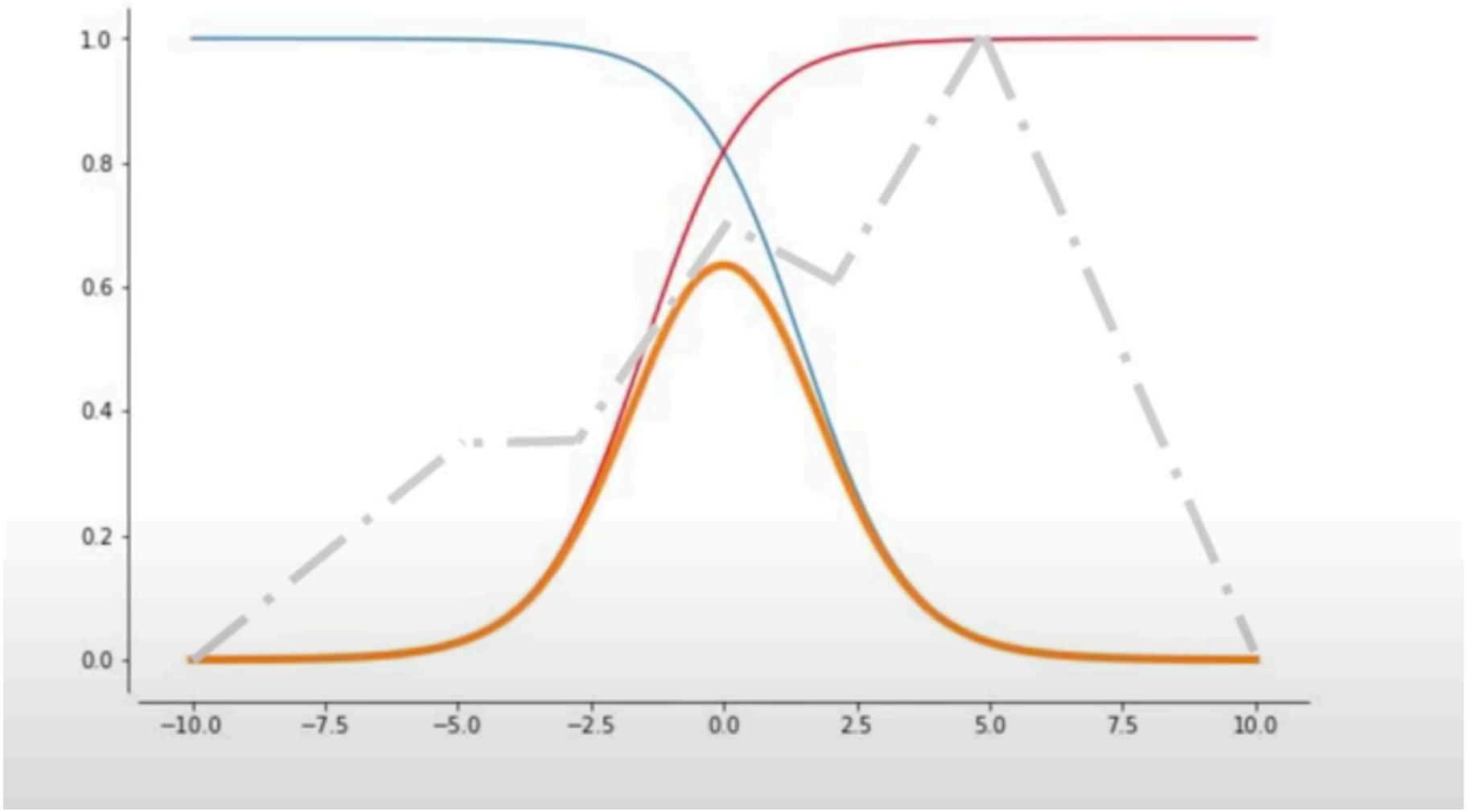
(deepmind
@Czarnecki)



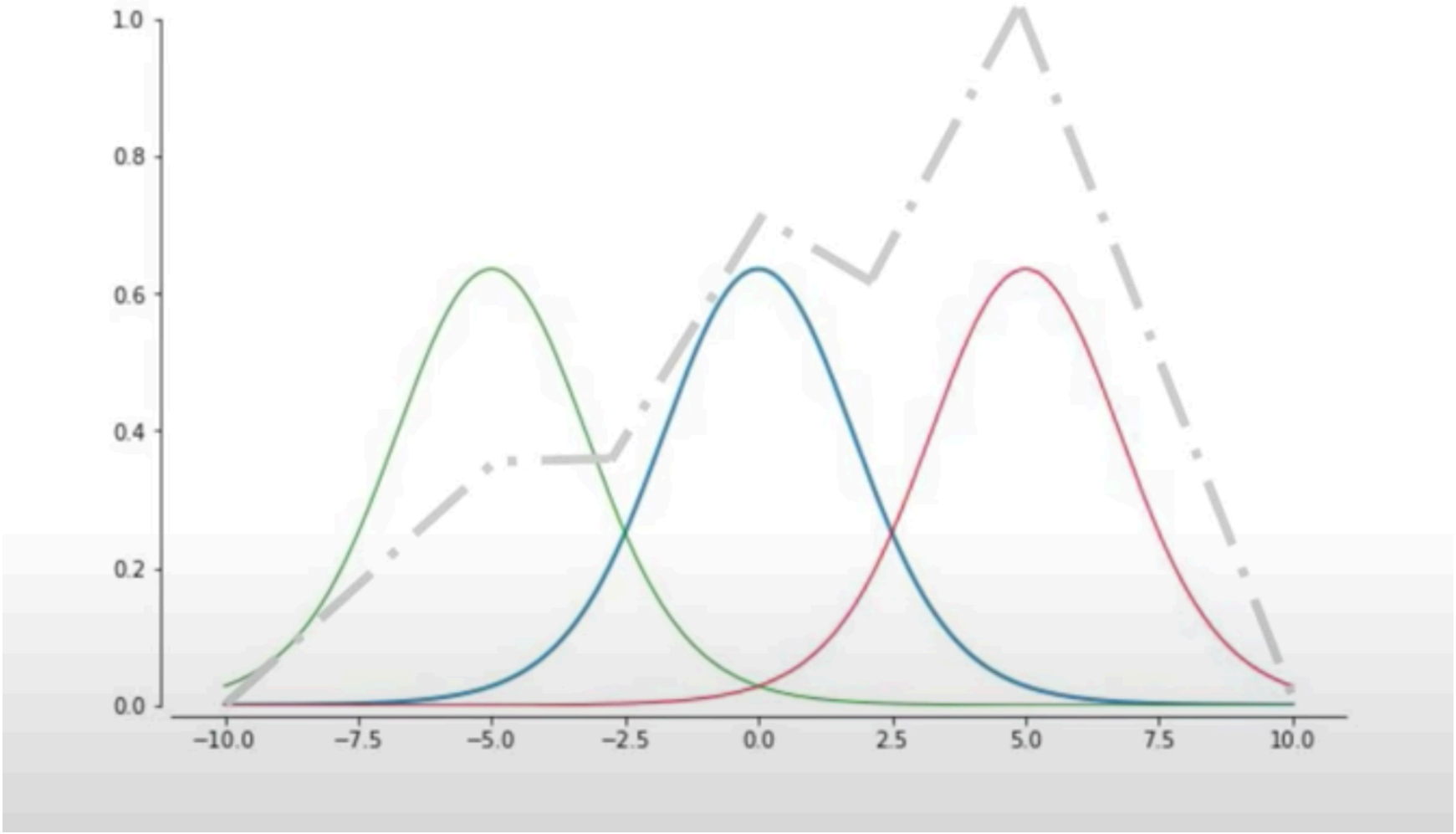
(deepmind
@Czarnecki)



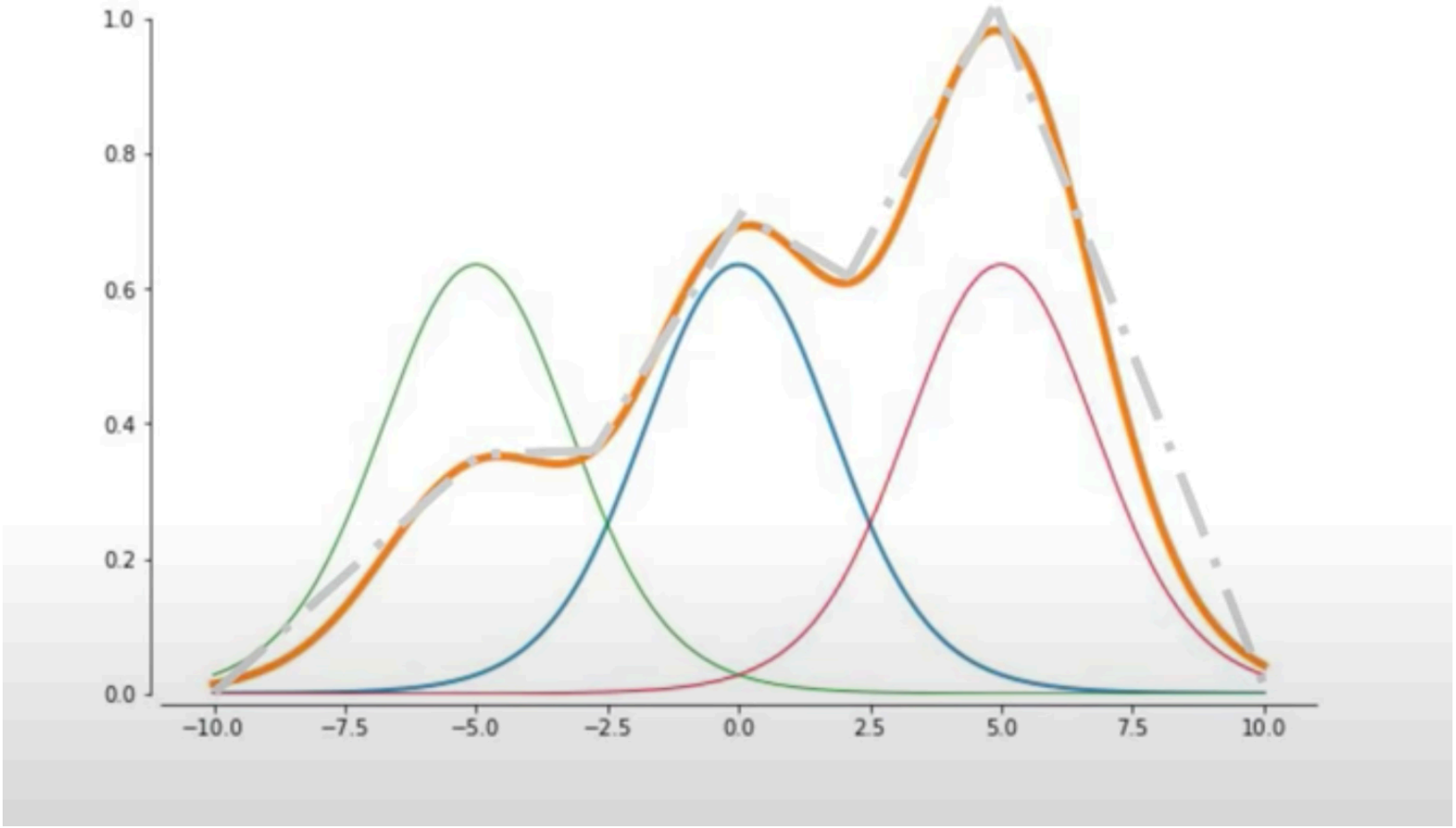
(deepmind
@Czarnecki)



(deepmind
@Czarnecki)

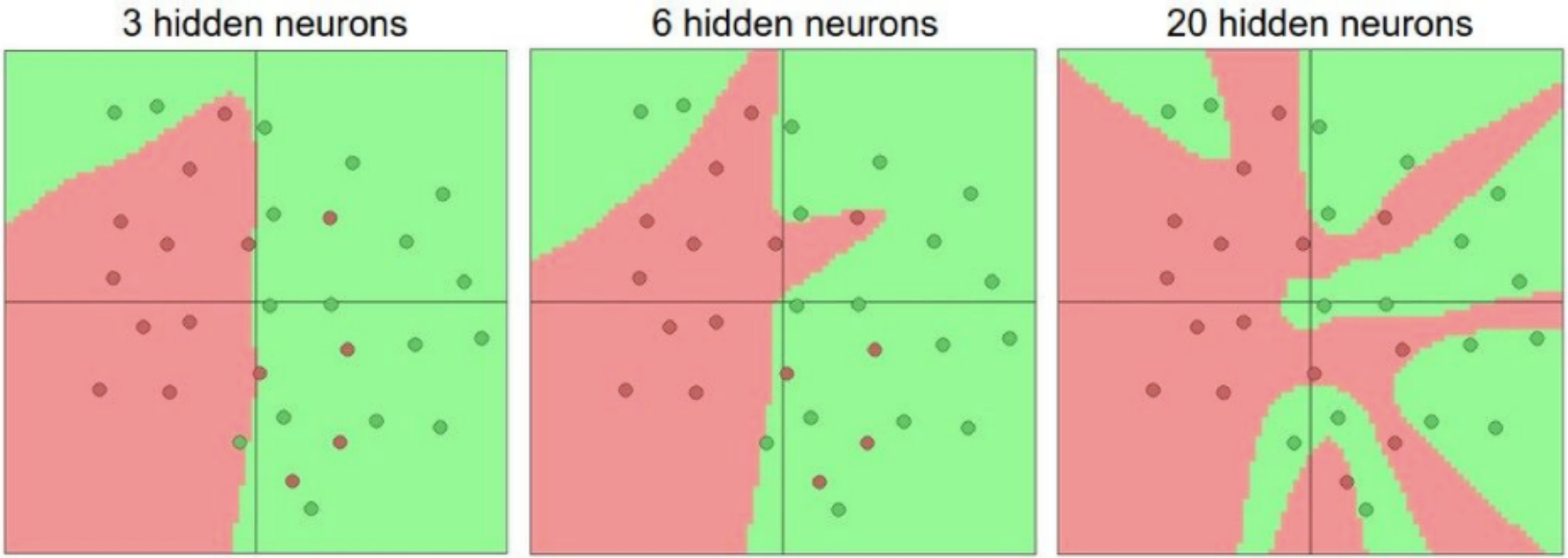


(deepmind
@Czarnecki)



(deepmind
@Czarnecki)

HIDDEN LAYERS ALLOW INCREASING COMPLEXITY



More complex functions allow increasing complexity

Credit: Karpathy

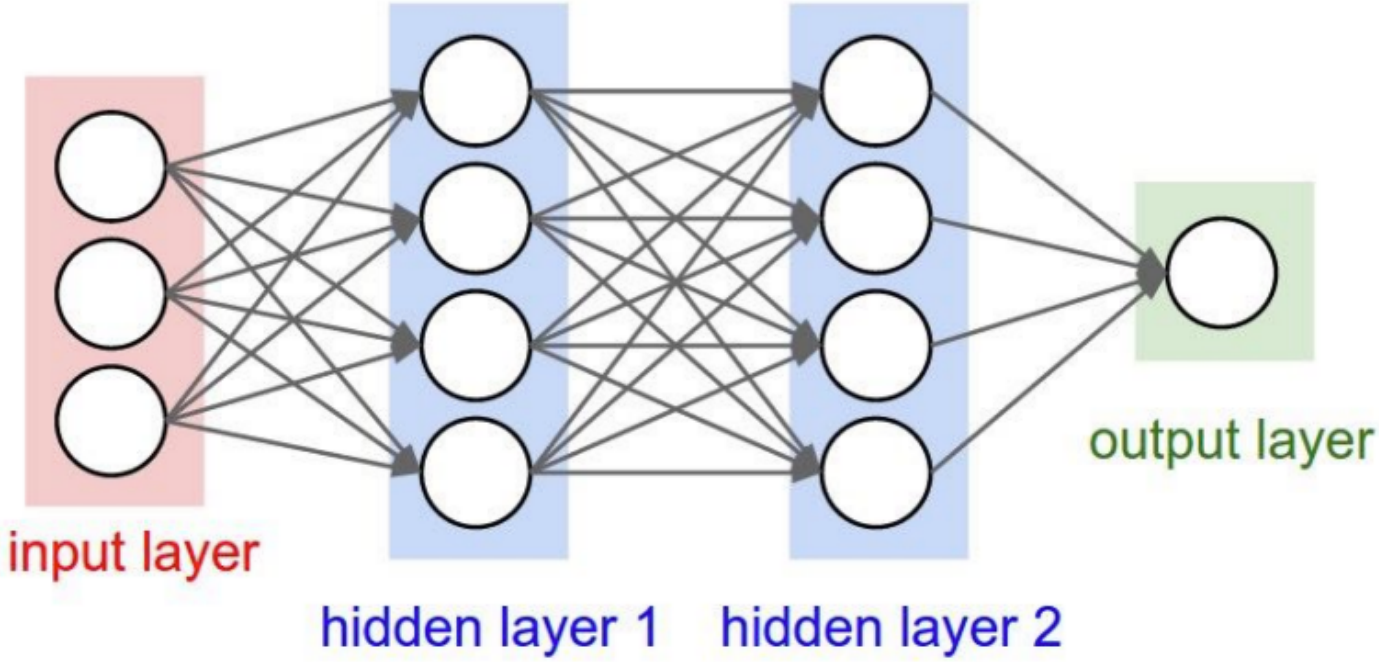
Optimisation of Neural Networks

$$f_W(\vec{x}) = \vec{y}$$

how to find the weights?

OPTIMIZATION

[OR HOW TO FIND THE WEIGHTS?]



$$f_w(x) = g_3(W_3 g_2(W_2 g_1(W_1(x))))$$

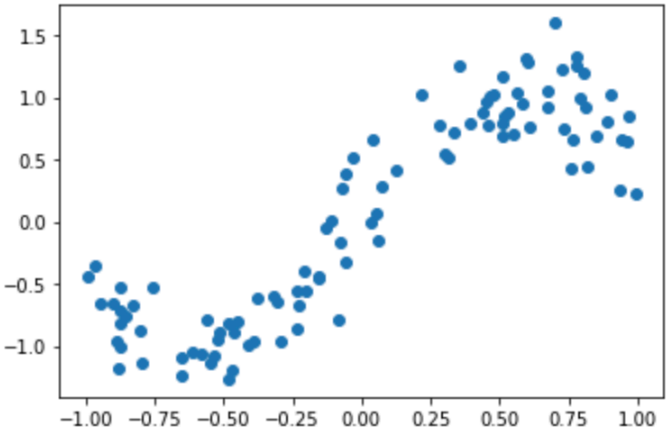
Network function

LOSS FUNCTIONS

WE FIRST NEED TO DEFINE THE OBJECTIVE TO MINIMISE:
THIS IS CALLED THE **LOSS FUNCTION**

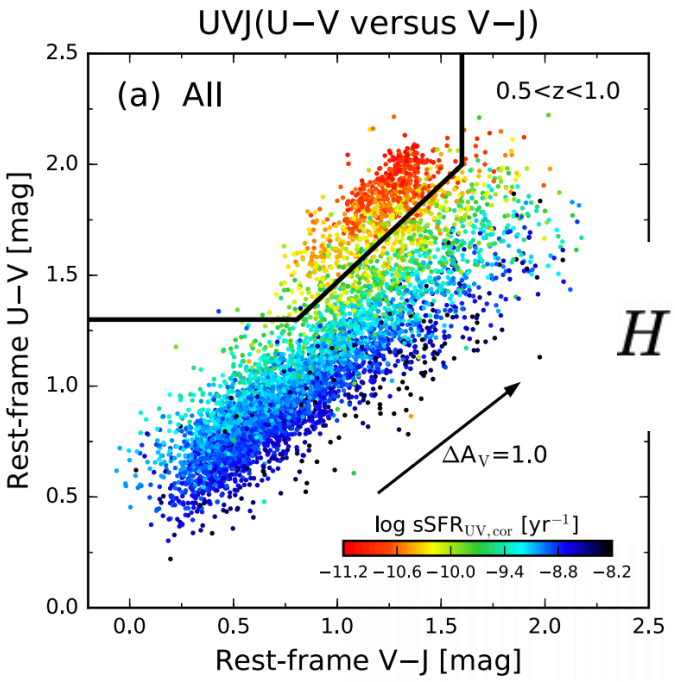
THE CHOICE OF THE LOSS FUNCTION IS CRITICAL AND WILL DETERMINE THE BEHAVIOUR OF THE ALGORITHM

IT IS IMPORTANT TO CHOOSE THE LOSS FUNCTION FOR YOUR PROBLEM



Regression

$$MSE = \frac{1}{N}\sum[y_i - f_w(x_i)]^2$$



Classification

$$H = -\frac{1}{N}\sum_{i=1}^{N} y_i.log(f_w(x_i)) + (1 - y_i).log(1 - f_w(x_i))$$

BINARY CLASSIFICATION LOSS

We typically use the binary cross-entropy loss:

$$H = -\frac{1}{N}\sum_{i=1}^{N} y_i.log(f_w(x_i)) + (1-y_i).log(1-f_w(x_i))$$

NUMBER OF TRAINING EXAMPLES

GROUND TRUTH

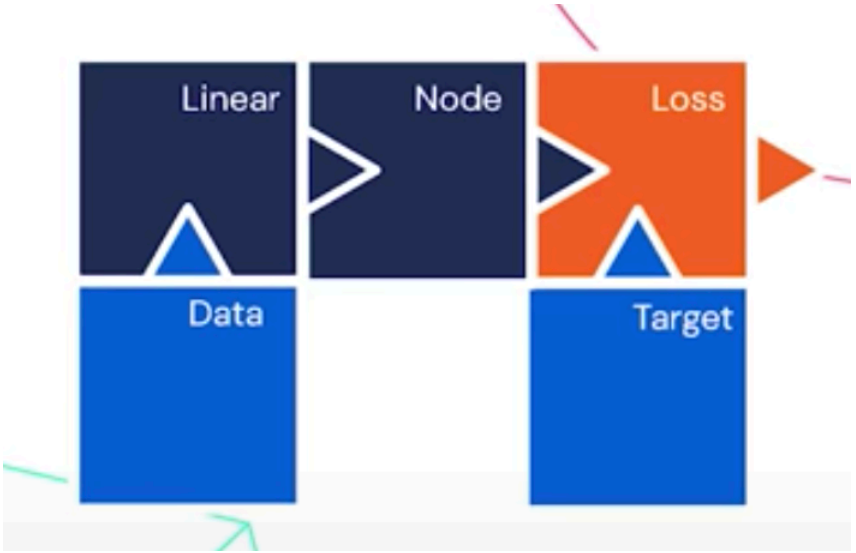
NETWORK OUTPUT (SIGMOID)

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



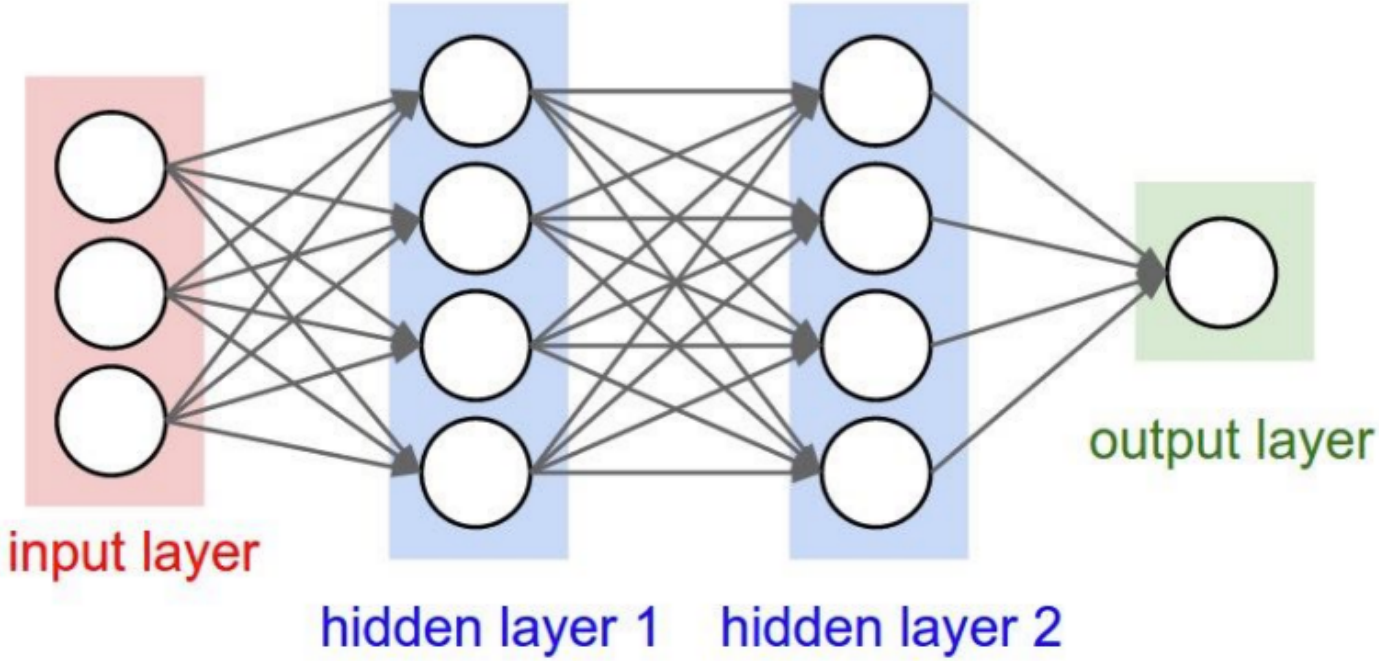
(deepmind @Czarnecki)

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

We still have not seen how to optimize this.

OPTIMIZATION

[OR HOW TO FIND THE WEIGHTS?]



$$f_w(x) = g_3(W_3 g_2(W_2 g_1(W_1(x))))$$

$$MSE = \frac{1}{N}\sum[y_i - f_w(x_i)]^2$$

$$H = -\frac{1}{N}\sum_{i=1}^{N} y_i.log(f_w(x_i)) + (1-y_i).log(1-f_w(x_i))$$

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

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

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….

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

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…

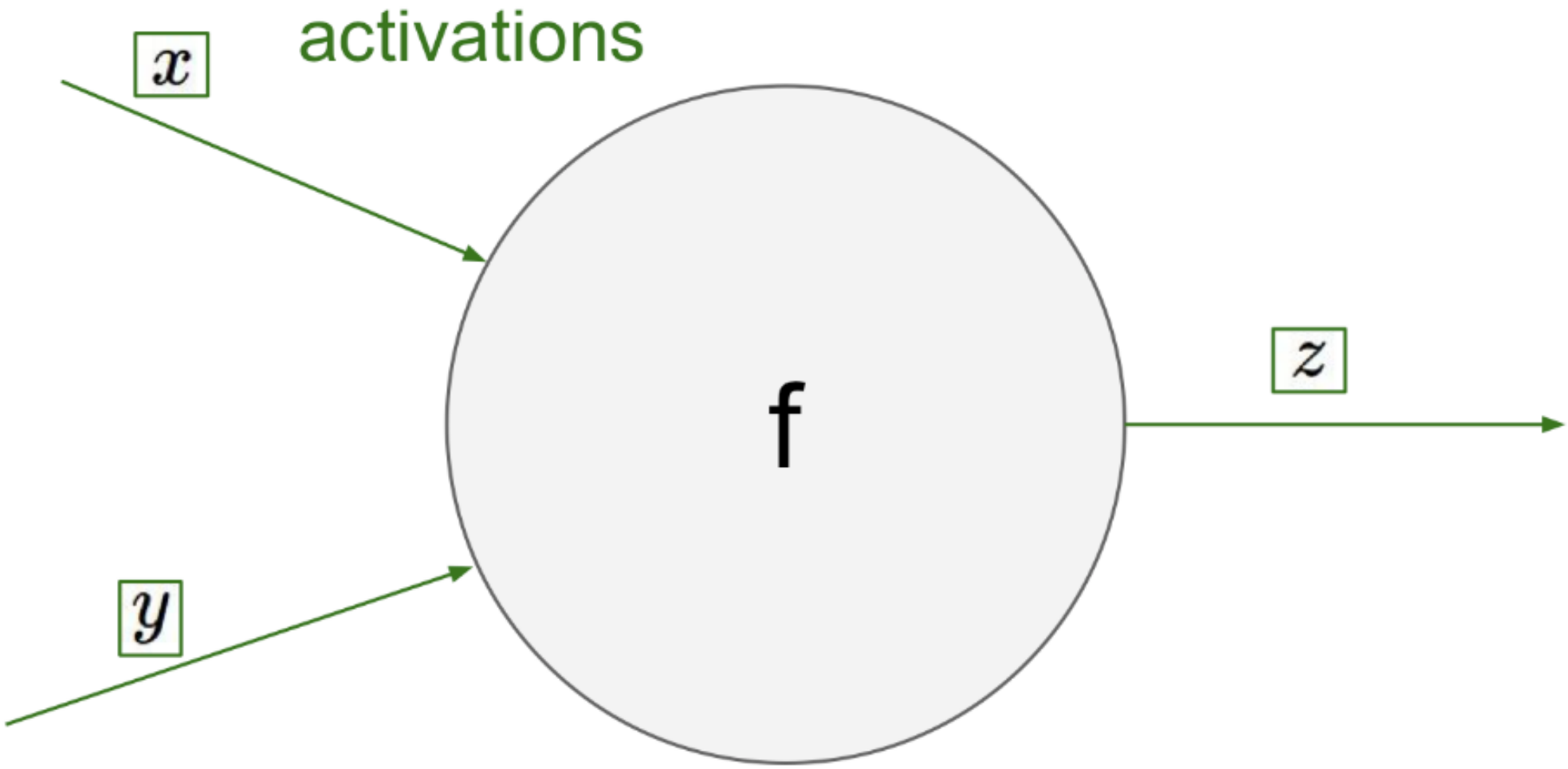
BUT NEEDS THE HESSIAN

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

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

BACKPROPAGATION

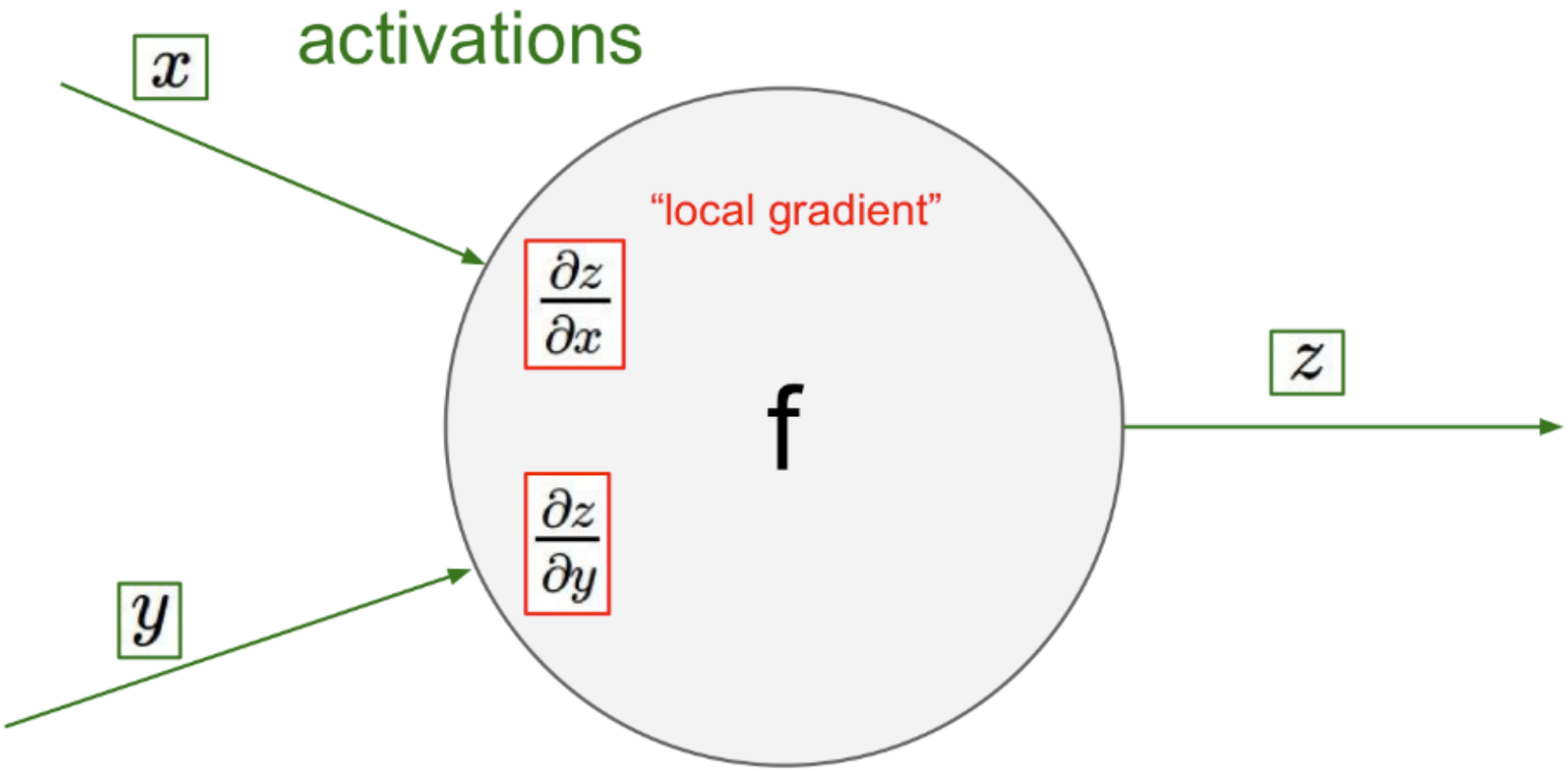
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

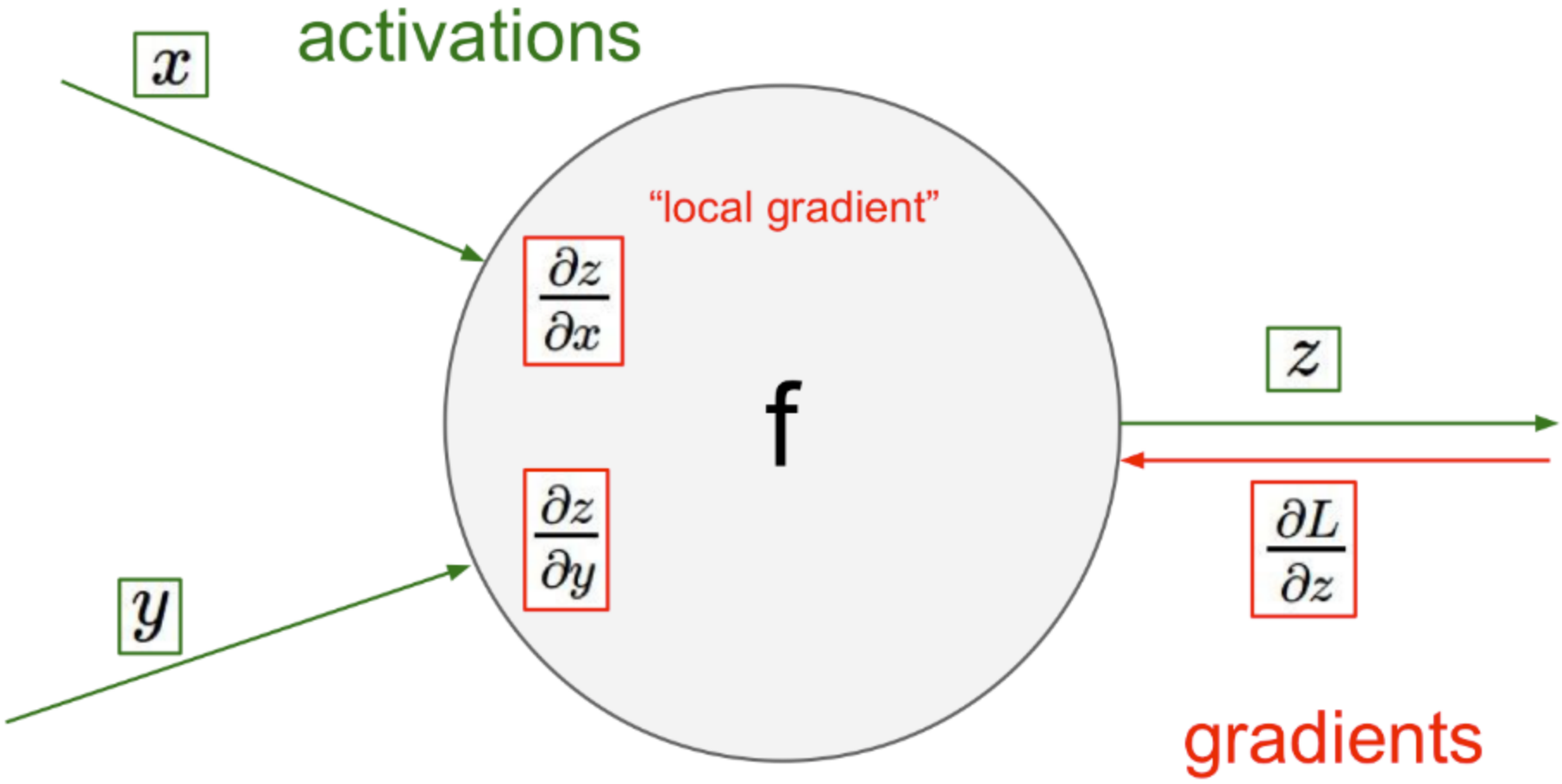
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

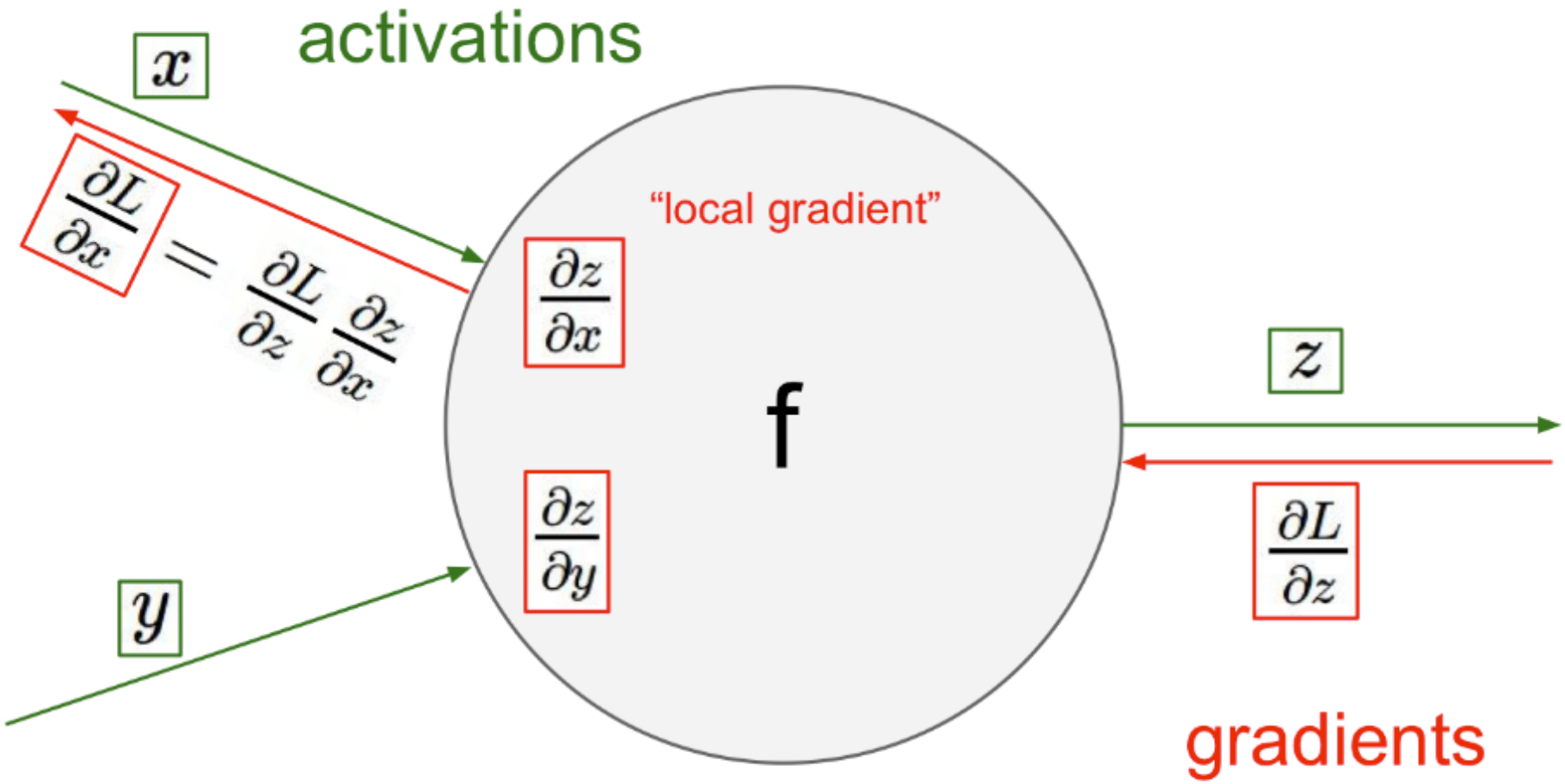
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

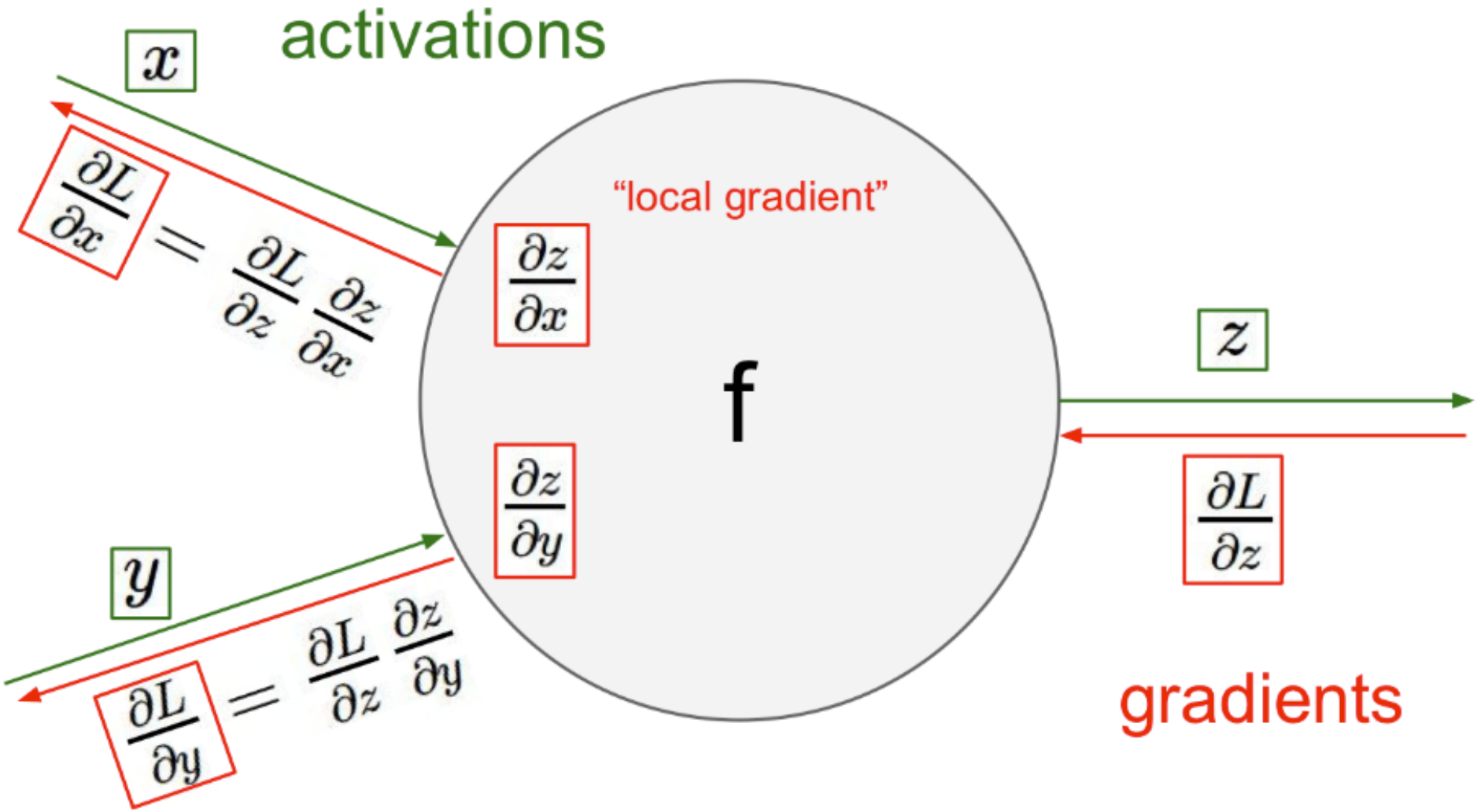
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

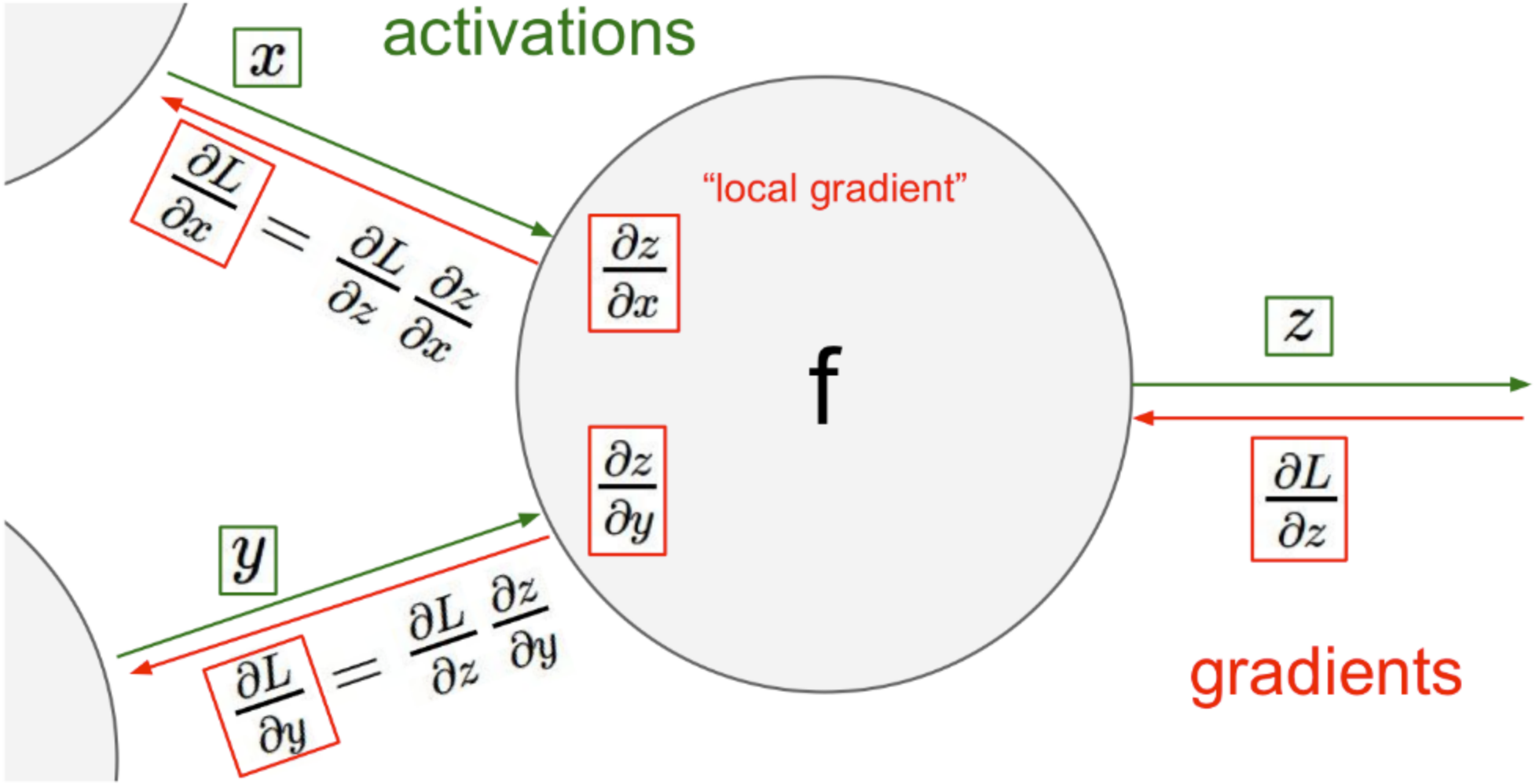
[AT THE NEURON LEVEL]



Credit: A. Karpathy

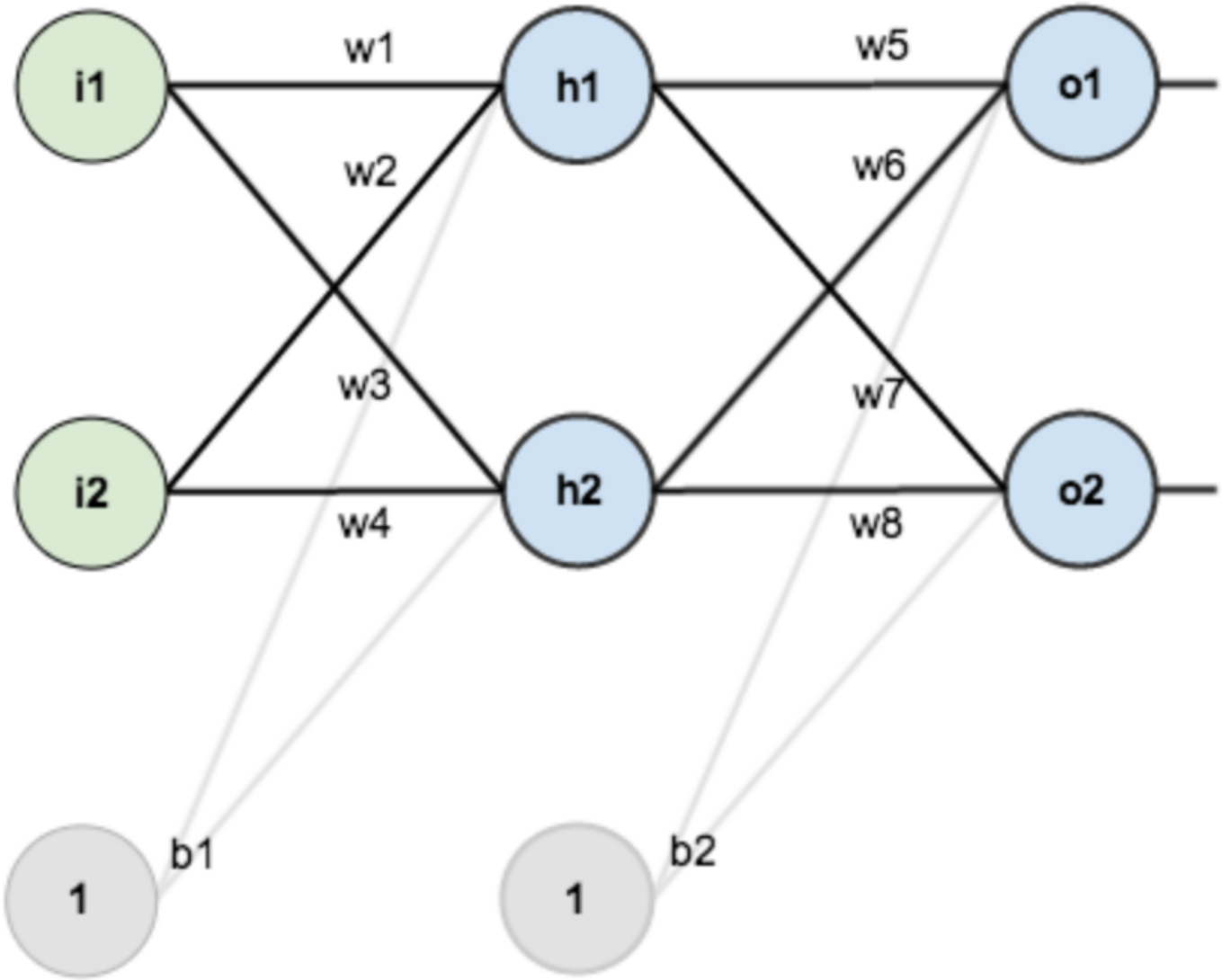
BACKPROPAGATION

[AT THE NEURON LEVEL]

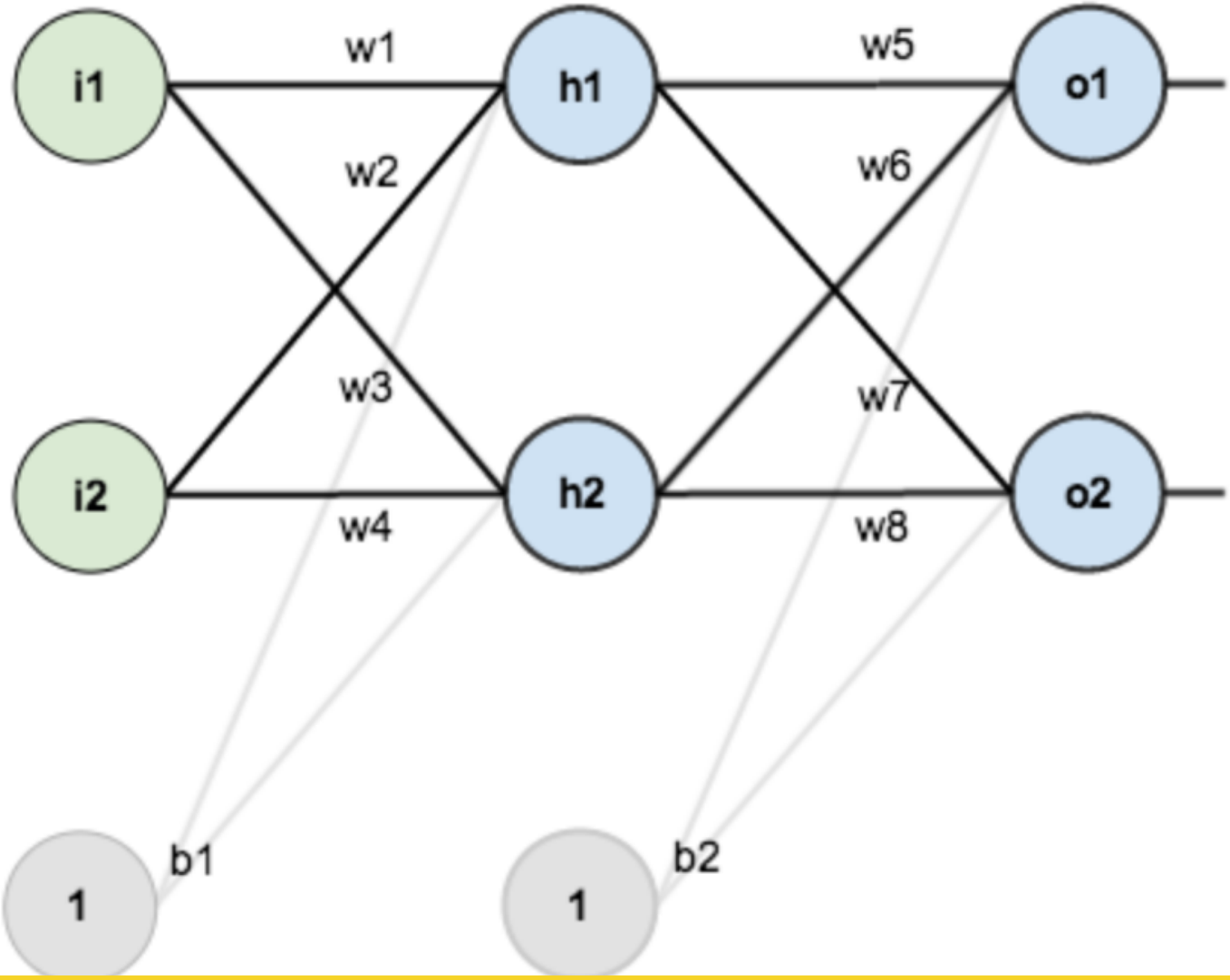


Credit: A. Karpathy

LET’S FOLLOW A NETWORK WHILE IT LEARNS…

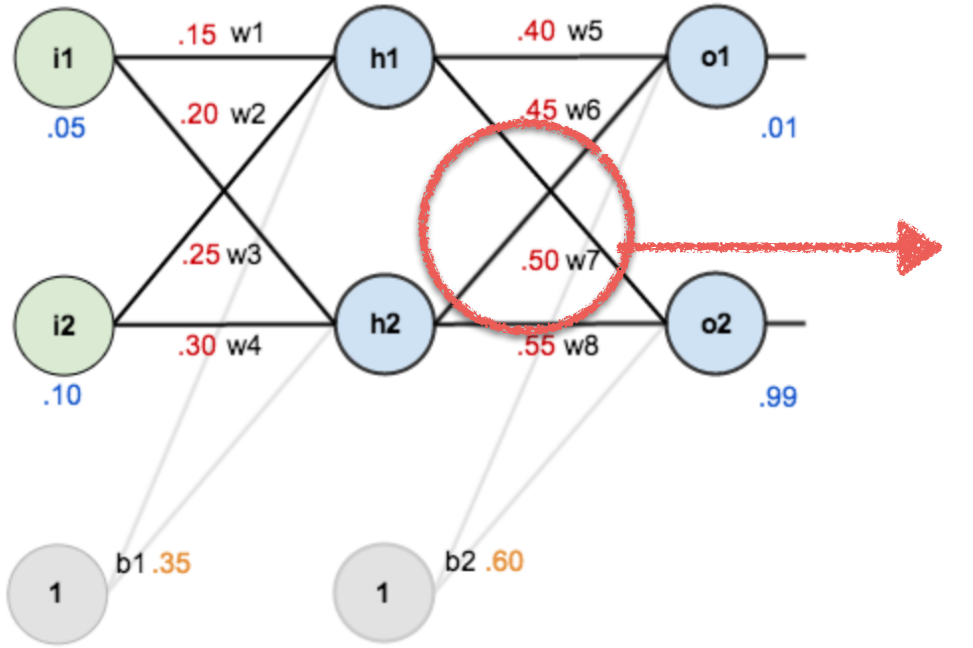


EXAMPLE TAKEN FROM HERE



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

EXAMPLE TAKEN FROM HERE



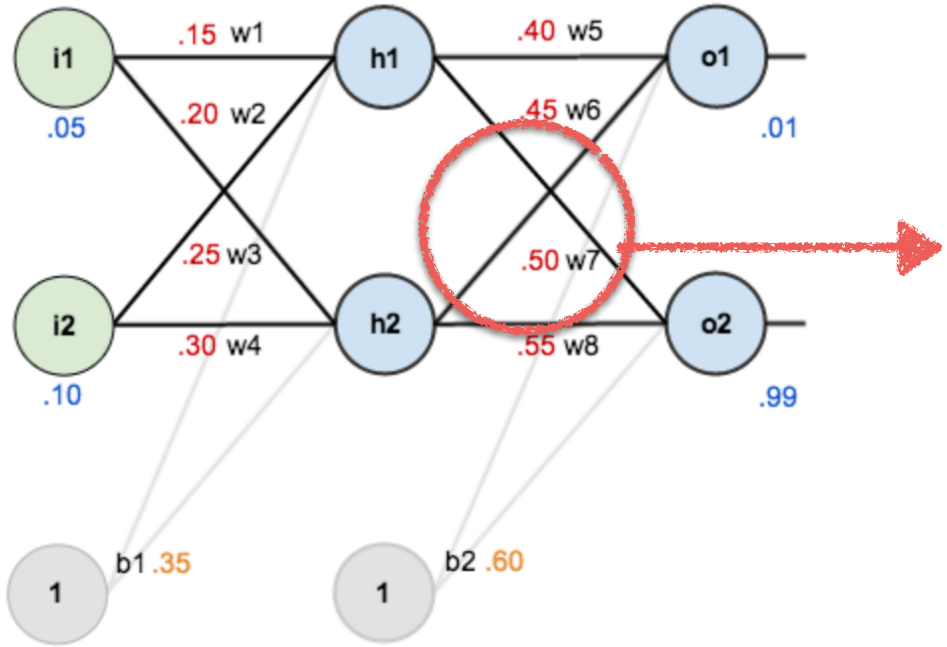
initial weights

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]



initial weights

1. THE FORWARD PASS

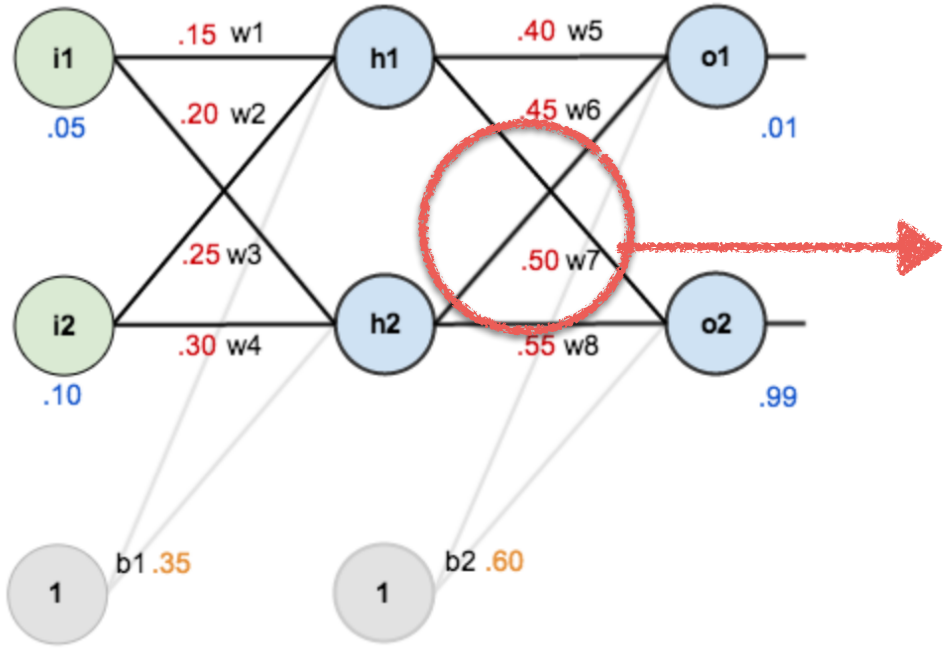
$$in_{h1} = w_1 i_1 + w_2 i_2 + b_1$$

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

[with initial weights]

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

[after the activation function]



initial weights

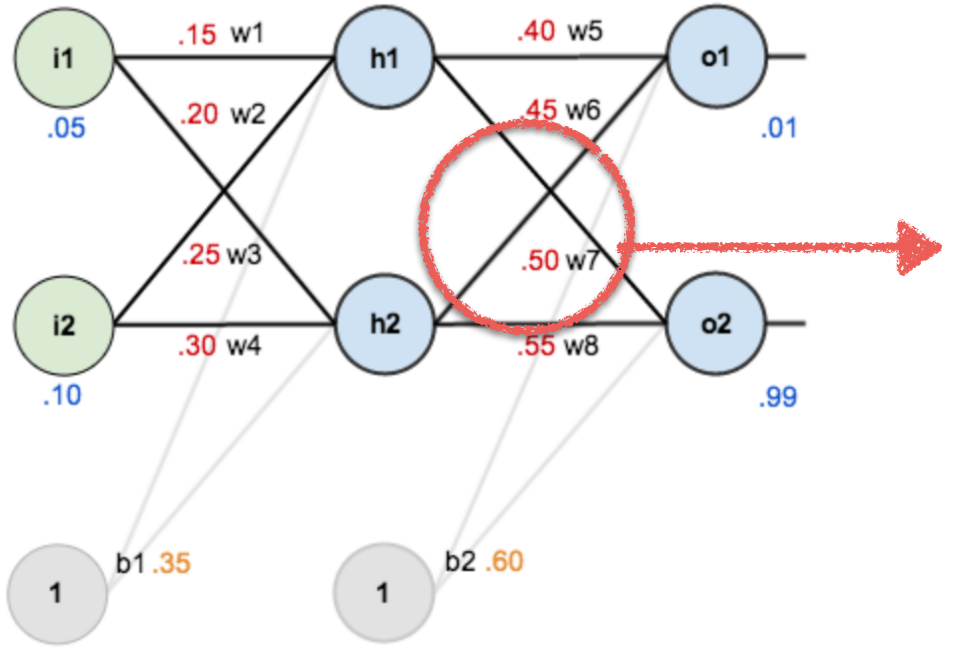
1. THE FORWARD PASS

WE CONTINUE TO o1

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

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

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

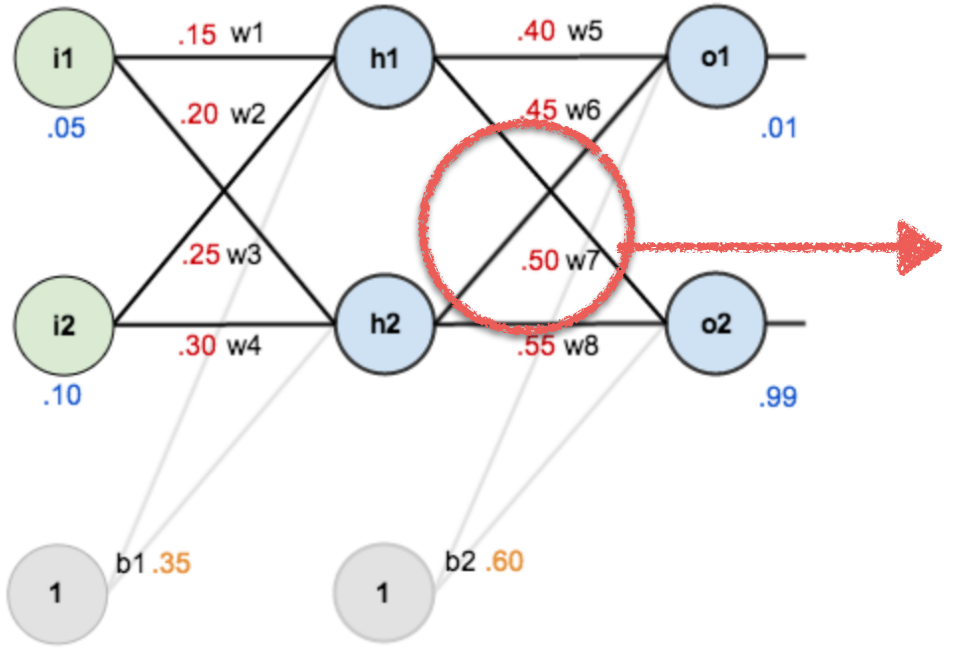


initial weights

1. THE FORWARD PASS

AND THE SAME FOR o2

$$out_{o2} = 0.7729$$



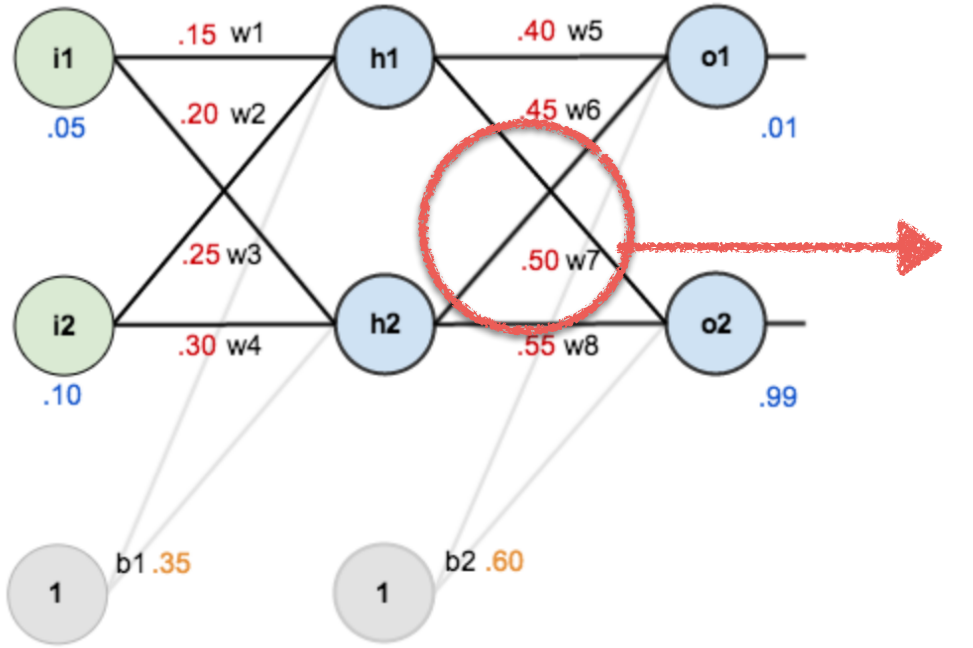
initial weights

2. THE LOSS FUNCTION

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

$$L_{o1} = 0.5(target_{o1} - output_{01})^2 = 0.5 \times (0.01 - 0.751)^2 = 0.274$$

$$L_{o2} = 0.023$$



initial weights

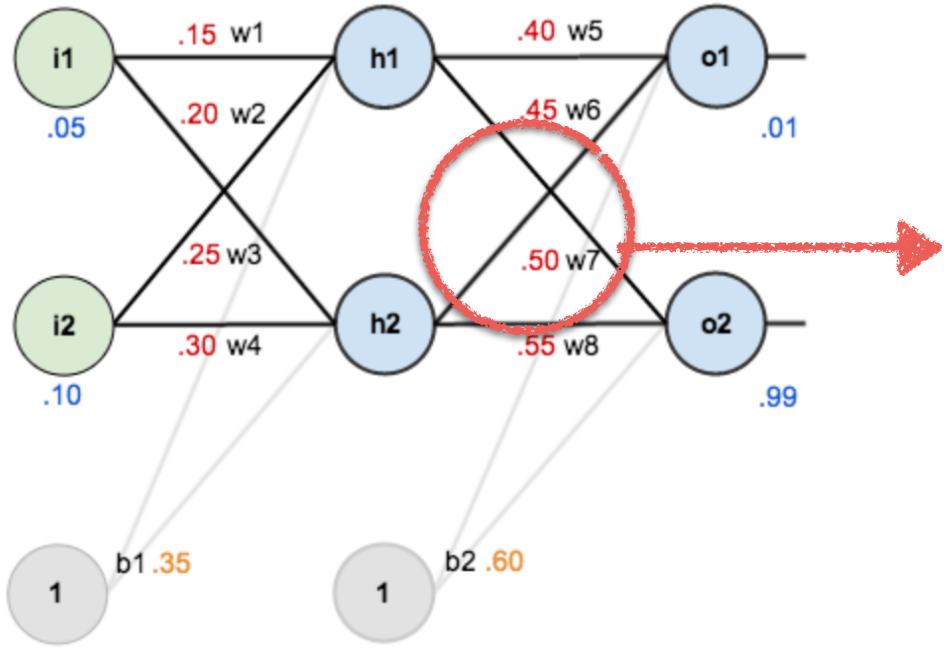
2. THE LOSS FUNCTION

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

$$L_{o1} = 0.5(target_{o1} - output_{01})^2 = 0.5 \times (0.01 - 0.751)^2 = 0.274$$

$$L_{o2} = 0.023$$

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



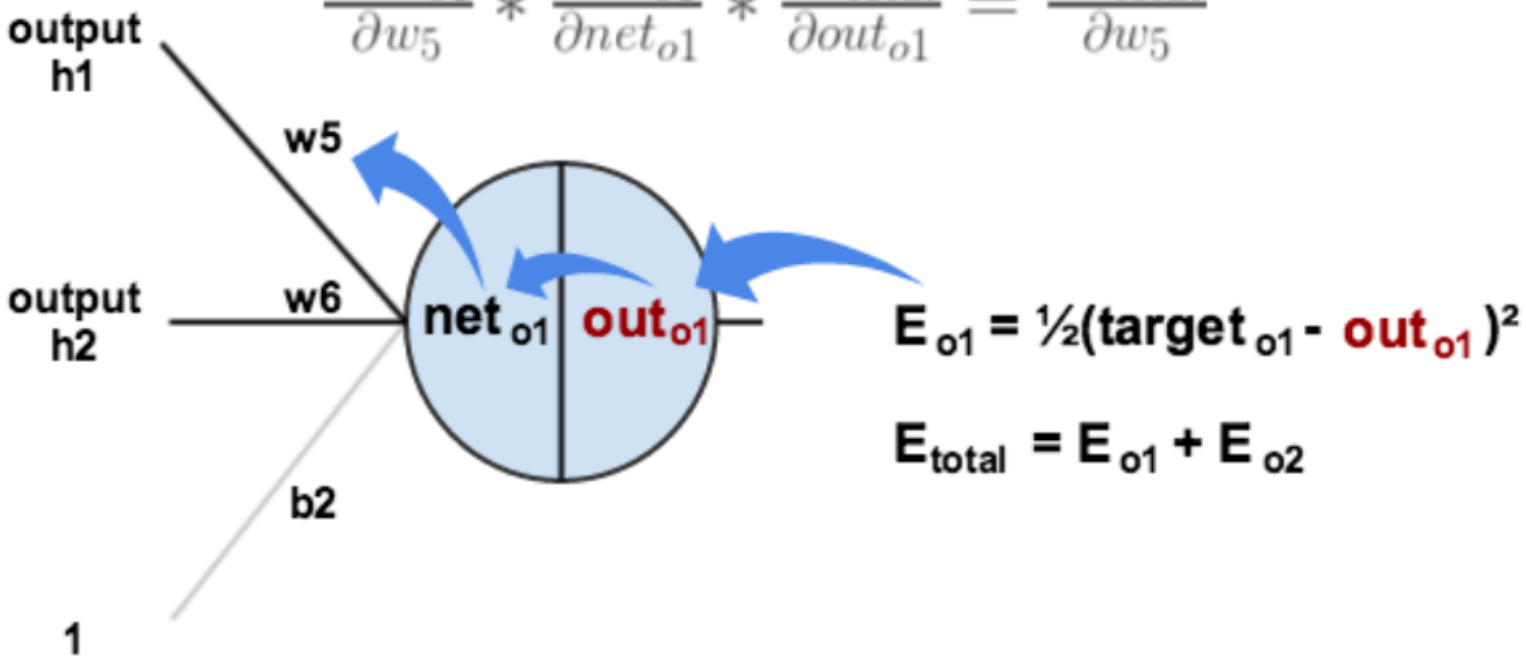
initial weights

3. THE BACKWARD PASS

FOR W5

WE WANT: $\frac{\partial L_{total}}{\partial w_5}$ [gradient of loss function]

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



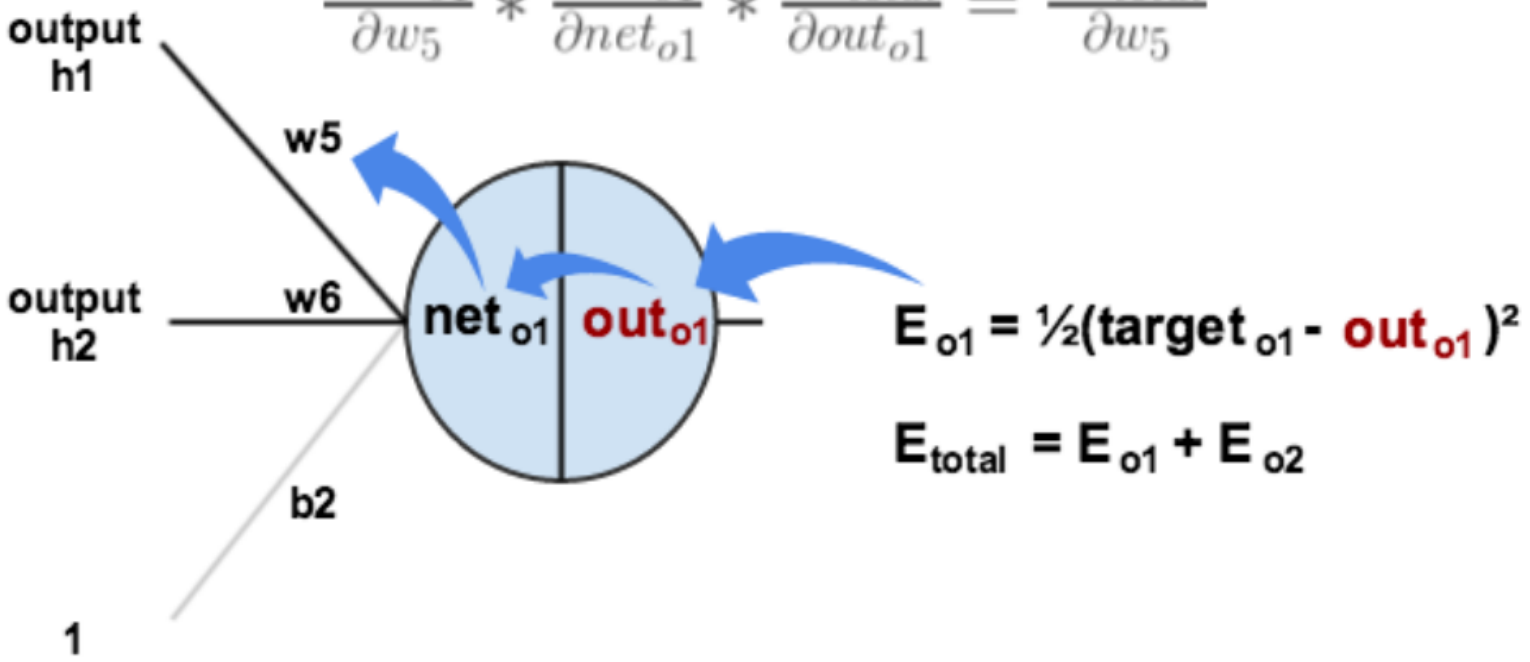
3. THE BACKWARD PASS

FOR W5 WE WANT: $\frac{\partial L_{total}}{\partial w_5}$ [gradient of loss function]

WE APPLY THE CHAIN RULE:

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

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



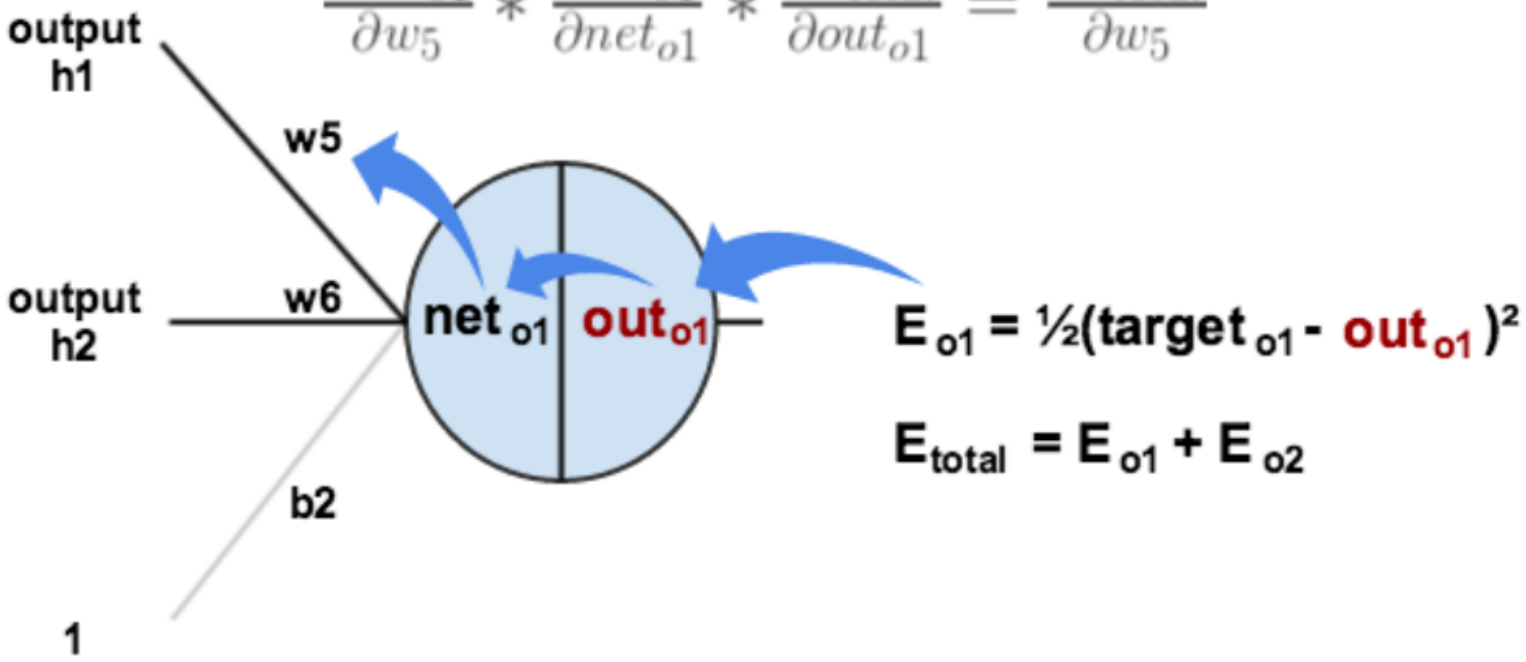
3. THE BACKWARD PASS

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

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

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

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



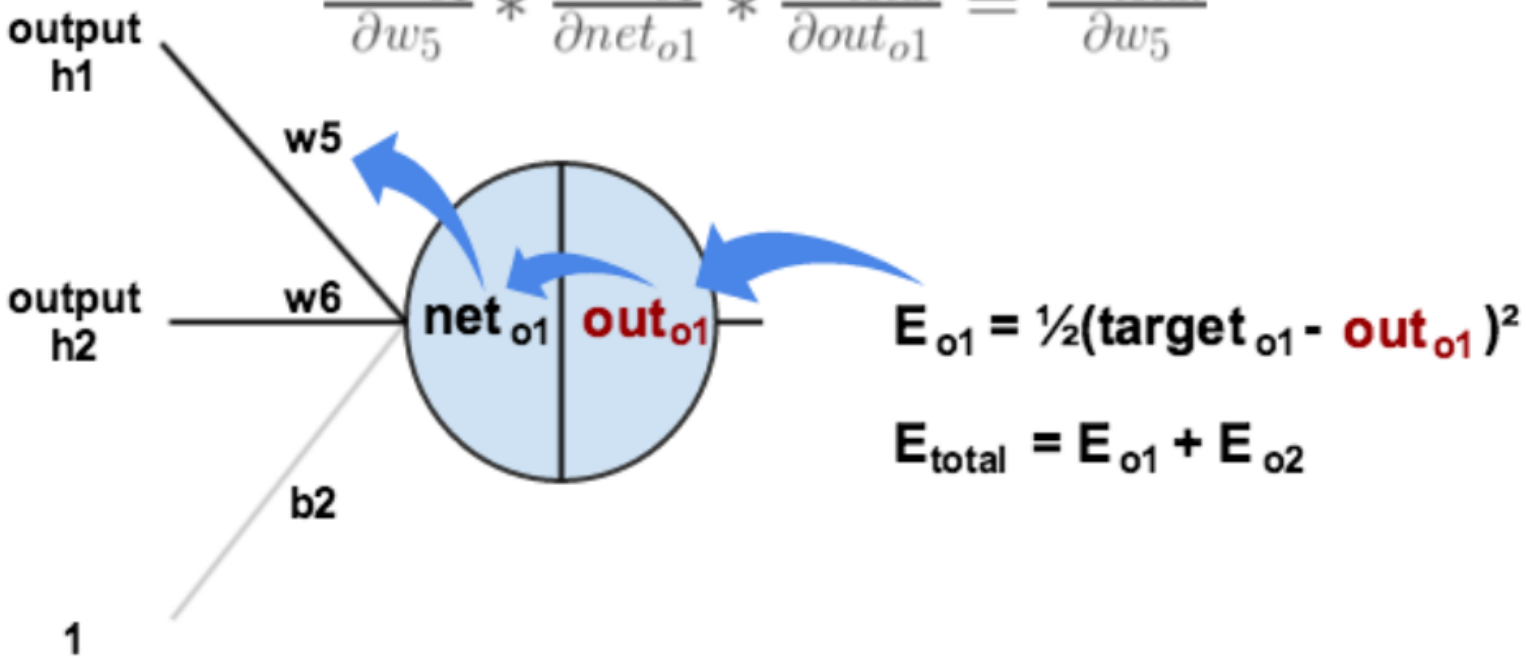
3. THE BACKWARD PASS

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

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

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

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



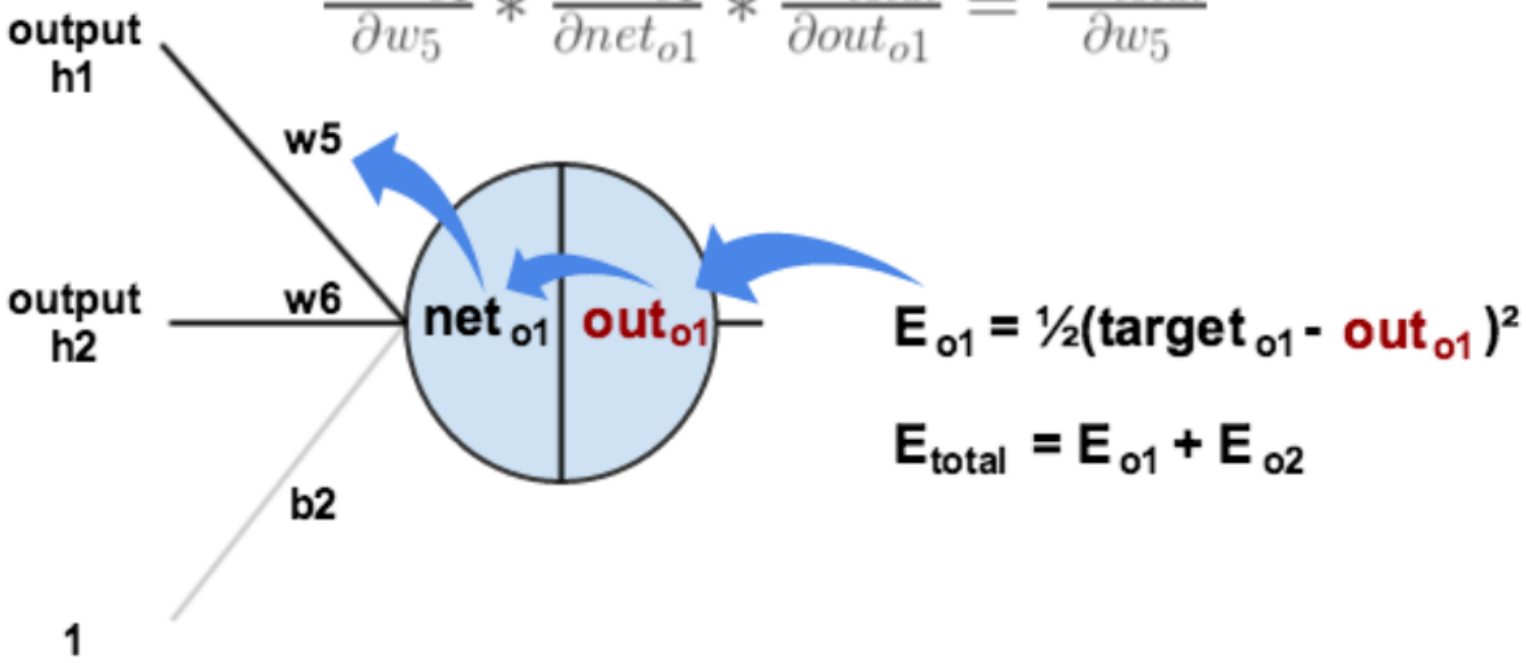
3. THE BACKWARD PASS

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

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

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

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



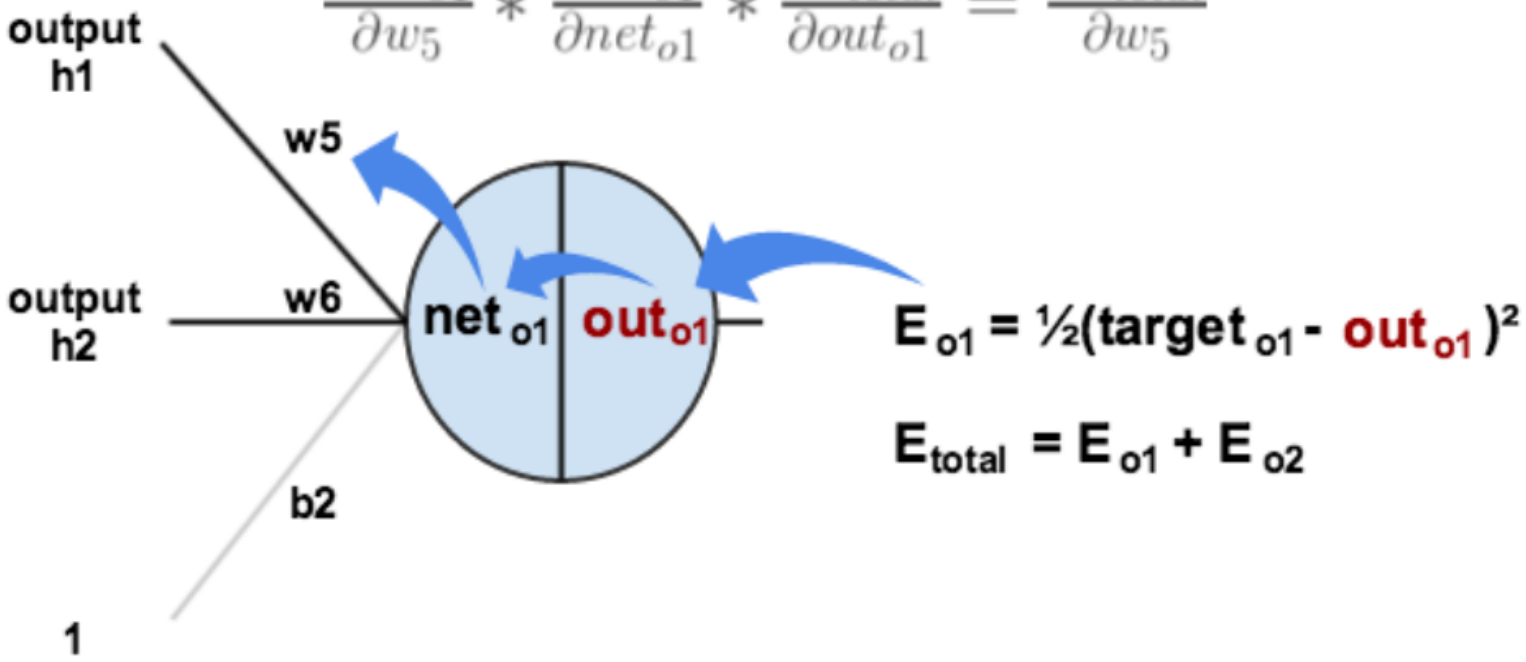
3. THE BACKWARD PASS

ALL TOGETHER:

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

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

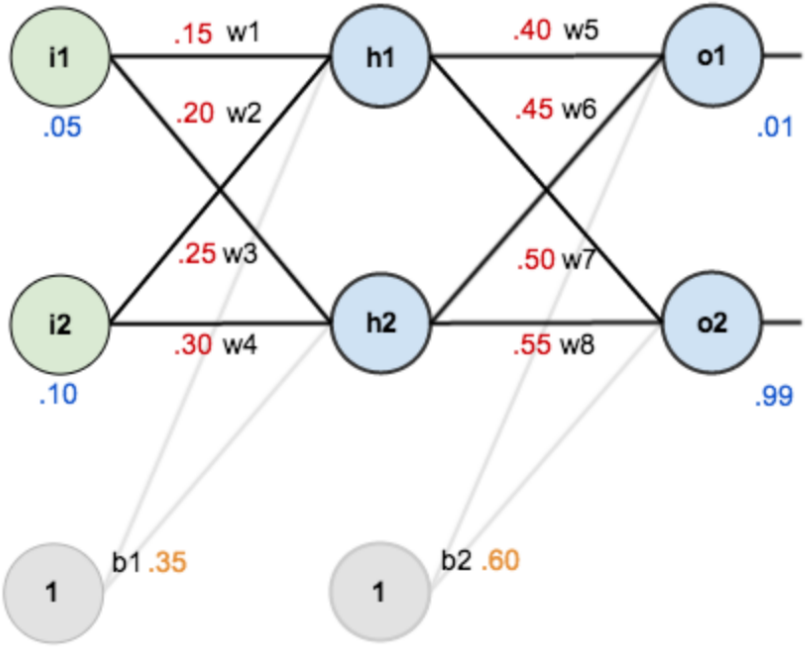
$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



4. UPDATE WEIGHTS WITH GRADIENT AND LEARNING RATE

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

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

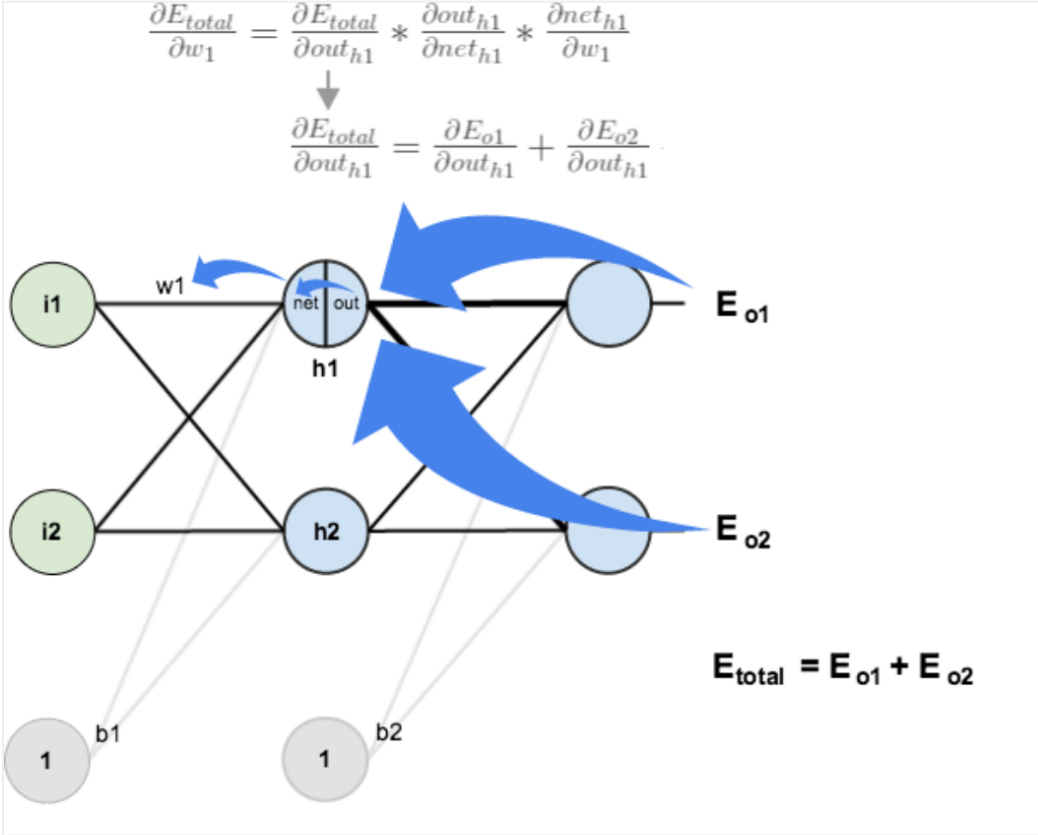


THIS IS REPEATED FOR THE OTHER WEIGHTS OF THE OUTPUT LAYER

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

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

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



AND BACK-PROPAGATED TO THE HIDDEN LAYERS