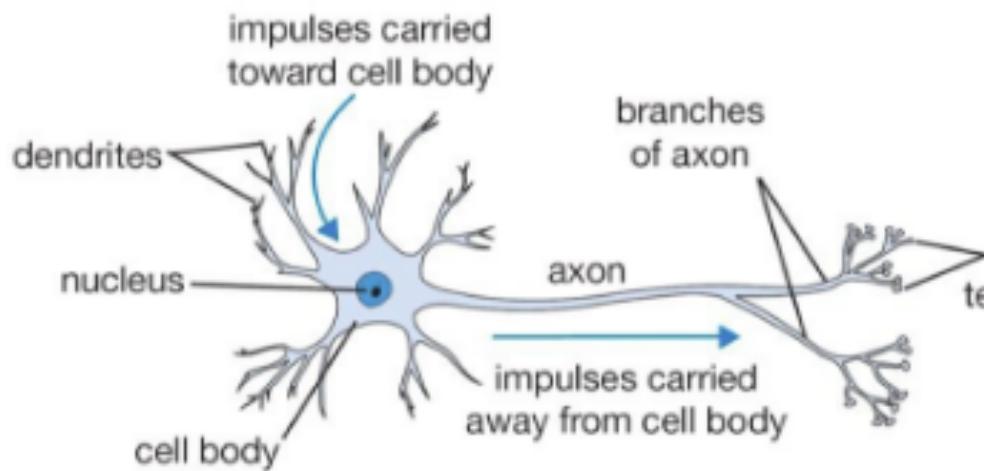


PART II: A FOCUS ON “SHALLOW” NEURAL NETWORKS

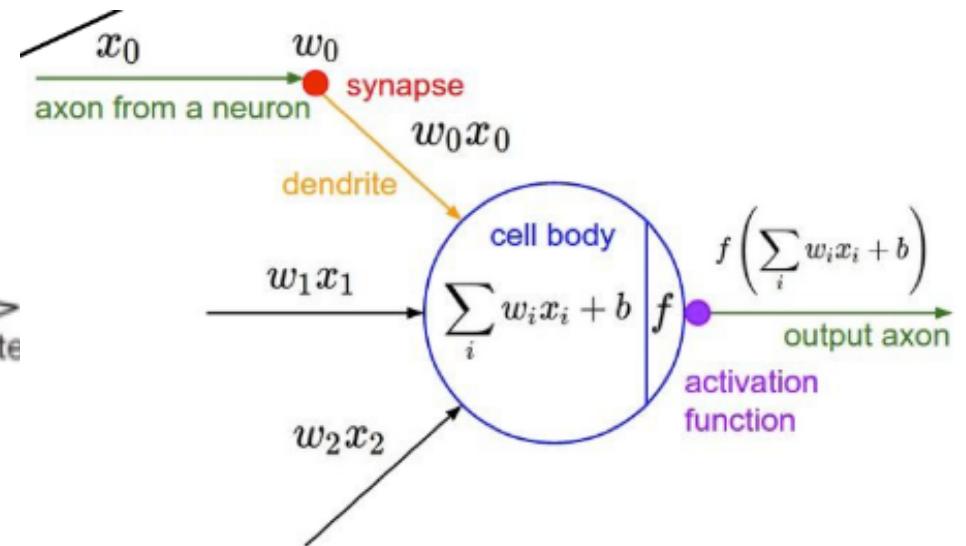
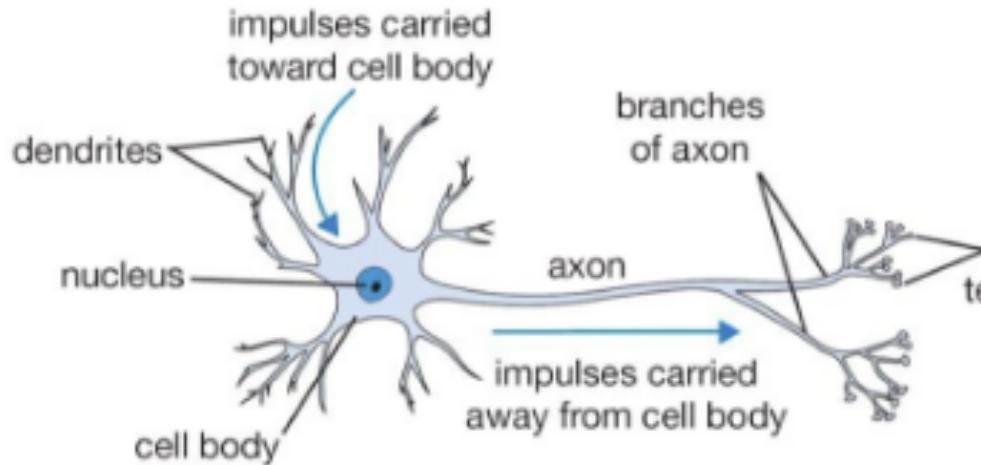
THE NEURON



INSPIRED BY NEURO - SCIENCE?

Credit: Karpathy

THE NEURON



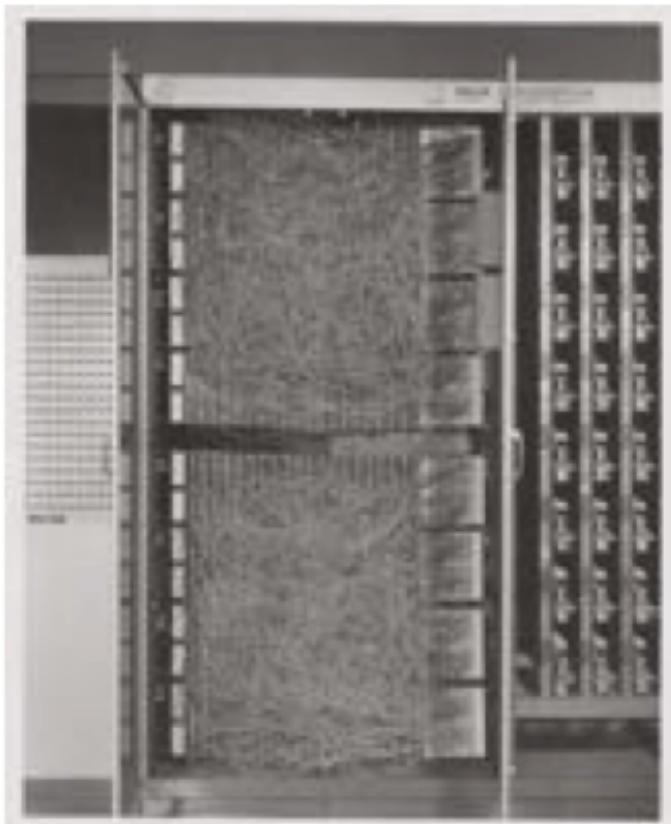
INSPIRED BY NEURO - SCIENCE?

Credit: Karpathy

Mark I Perceptron

FIRST IMPLEMENTATION OF NEURAL NETWORK [Rosenblatt,
1957!]

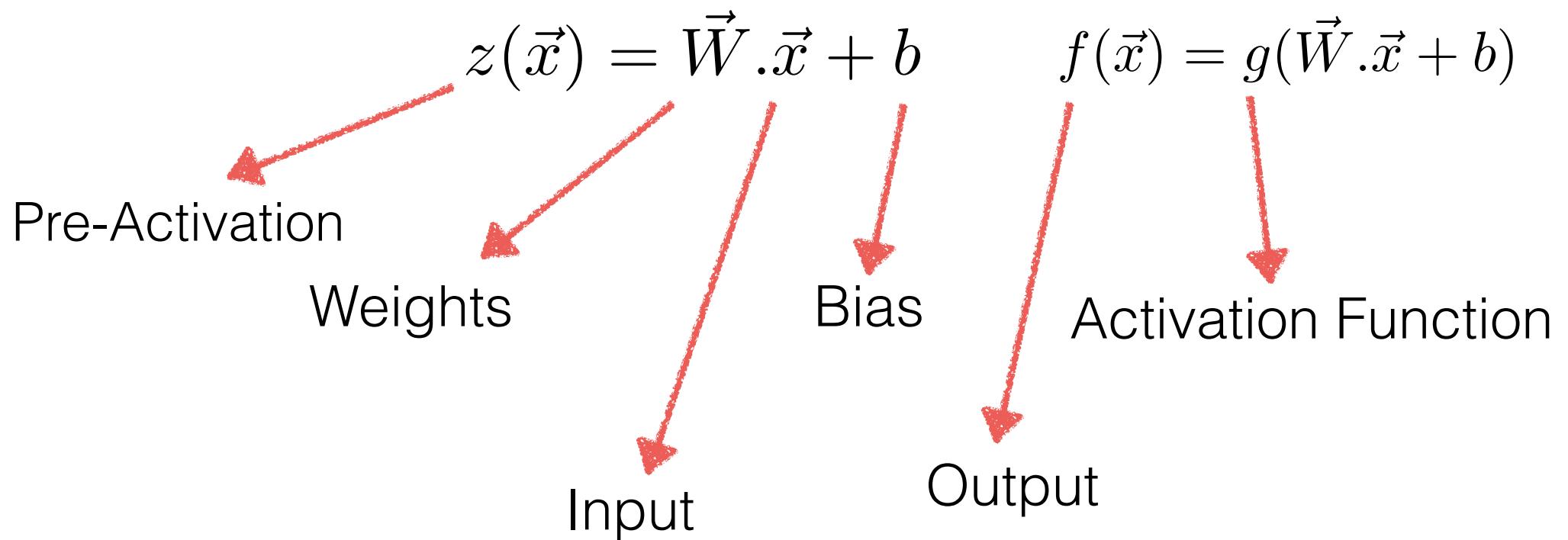
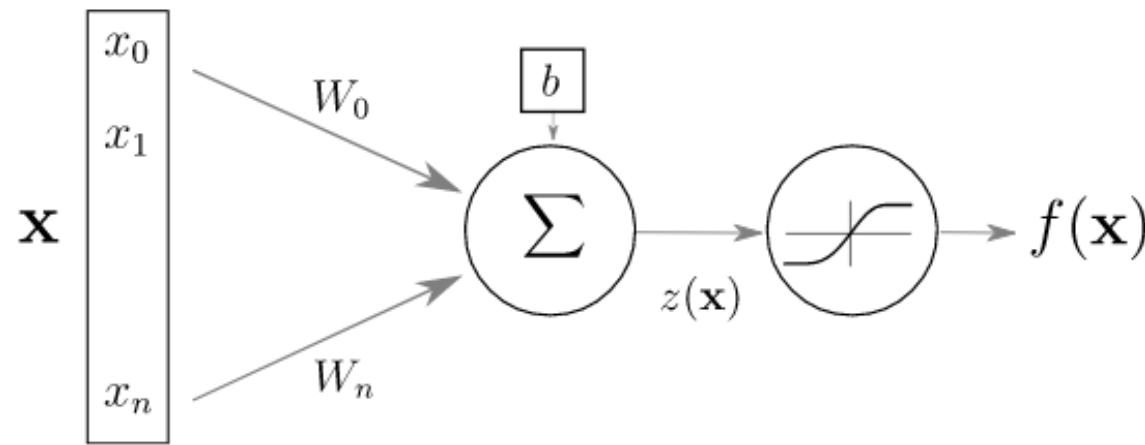
INTENDED TO BE A MACHINE (NOT AN ALGORITHM)



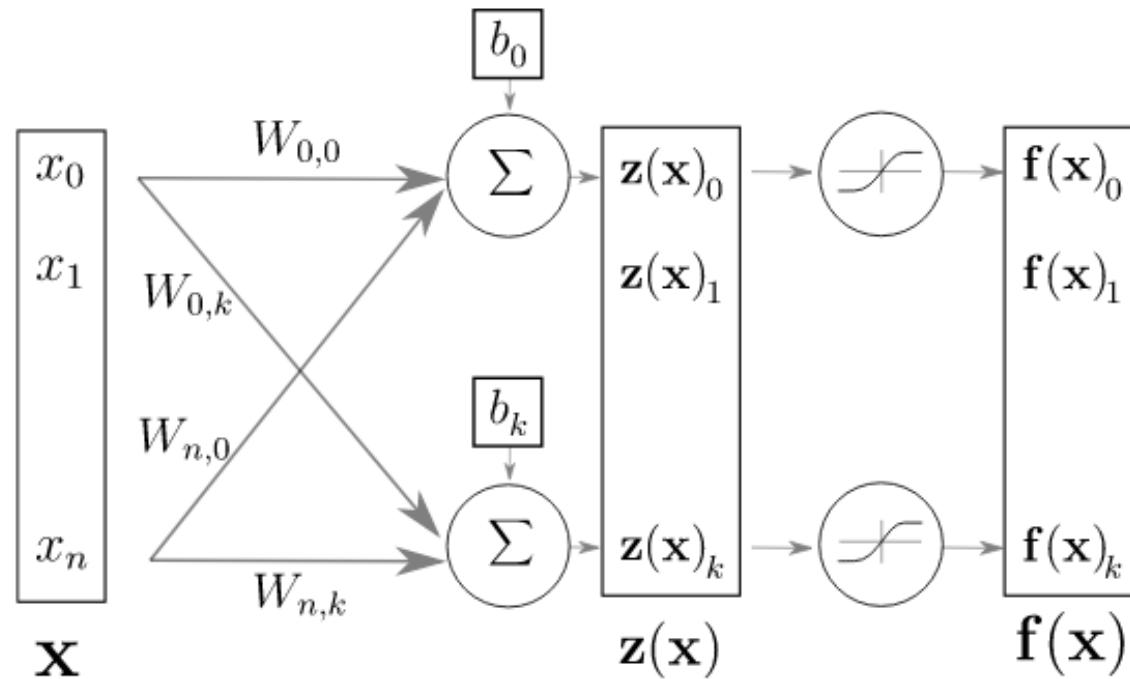
it had an array of 400 photocells,
randomly connected to the "neurons".

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

TODAY'S ARTIFICIAL NEURON



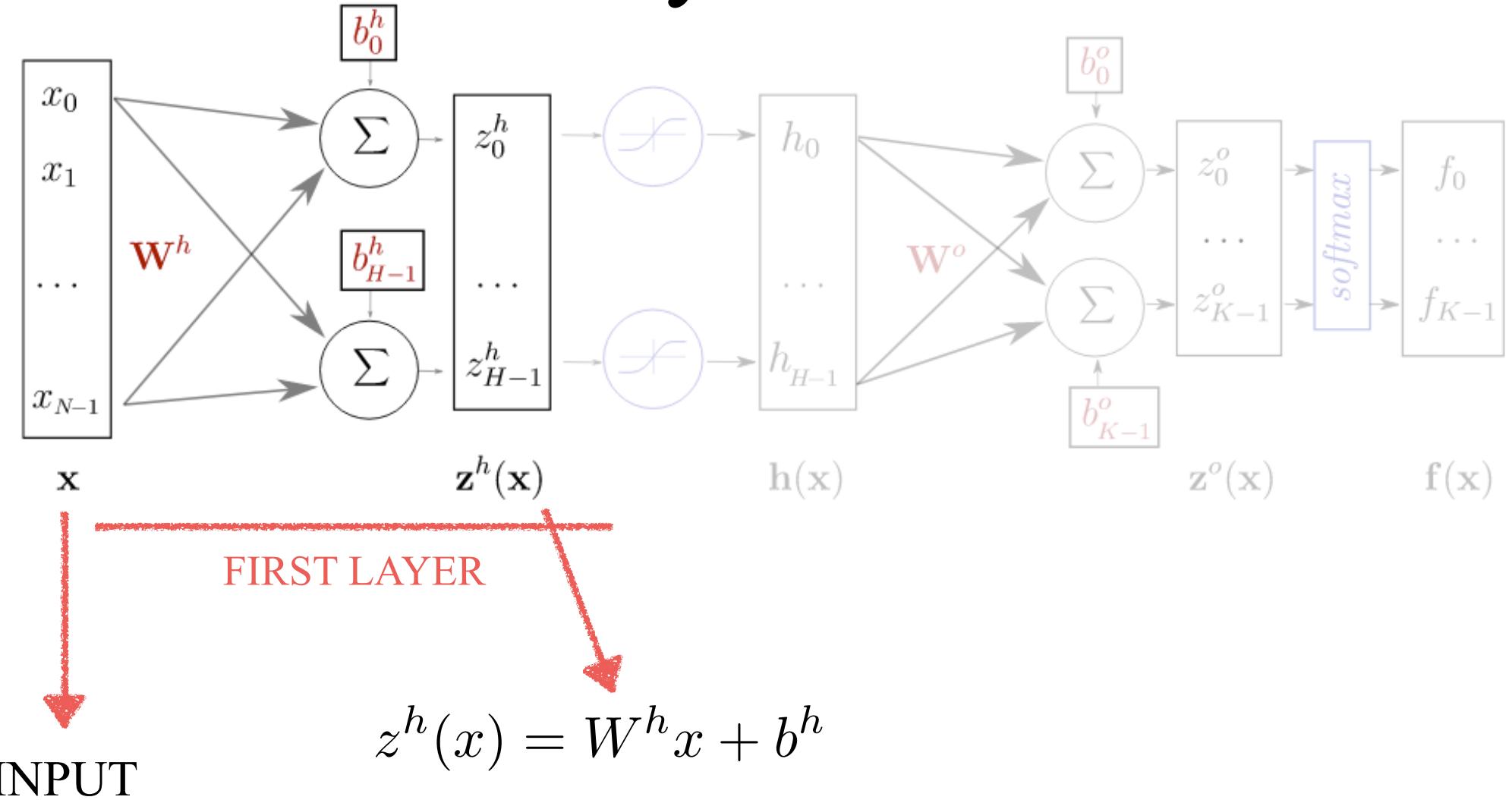
LAYER OF NEURONS



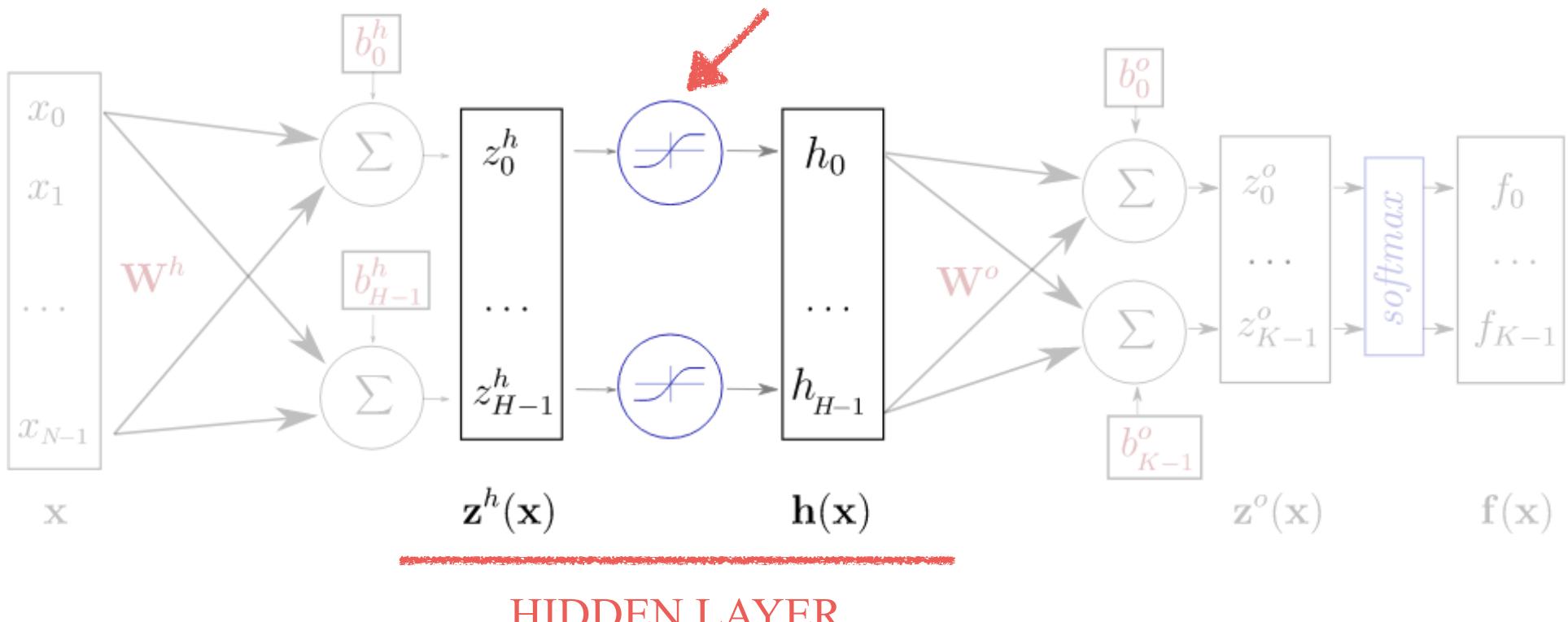
$$f(\vec{x}) = g(\mathbf{W} \cdot \vec{x} + \vec{b})$$

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

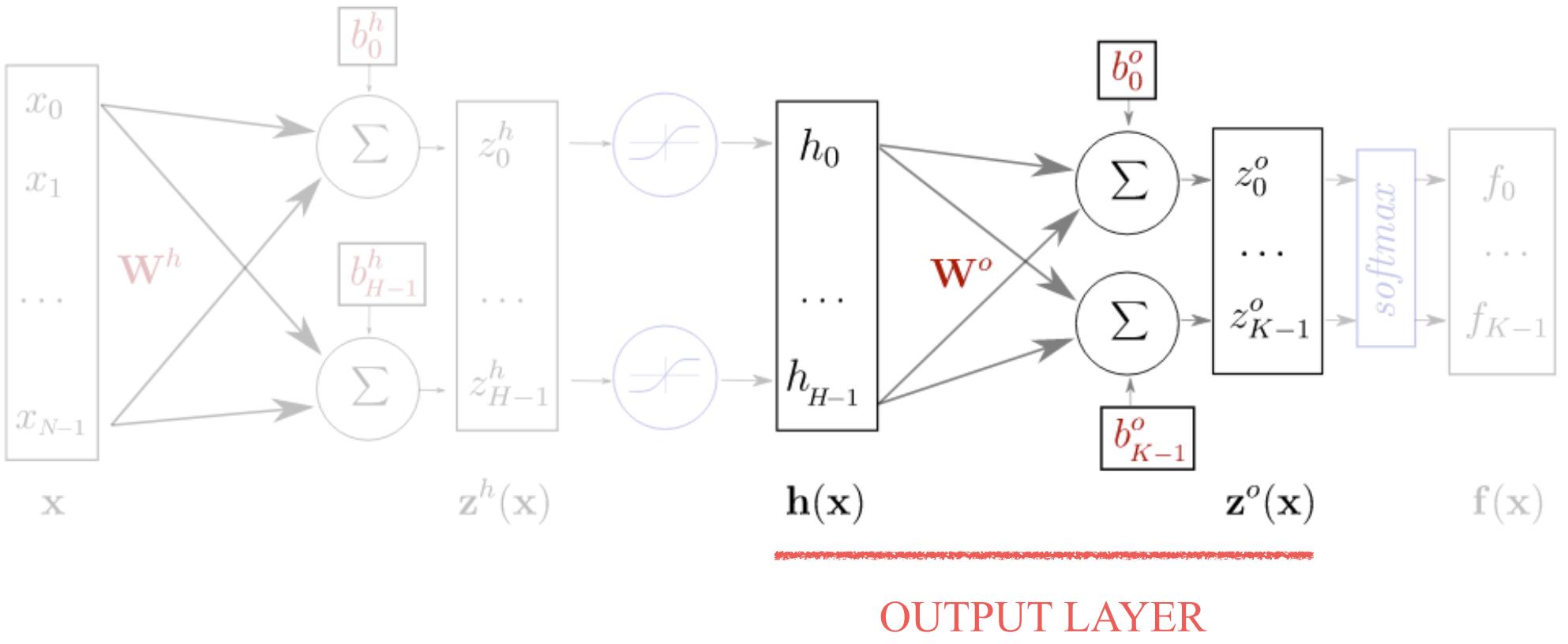
Hidden Layers of Neurons



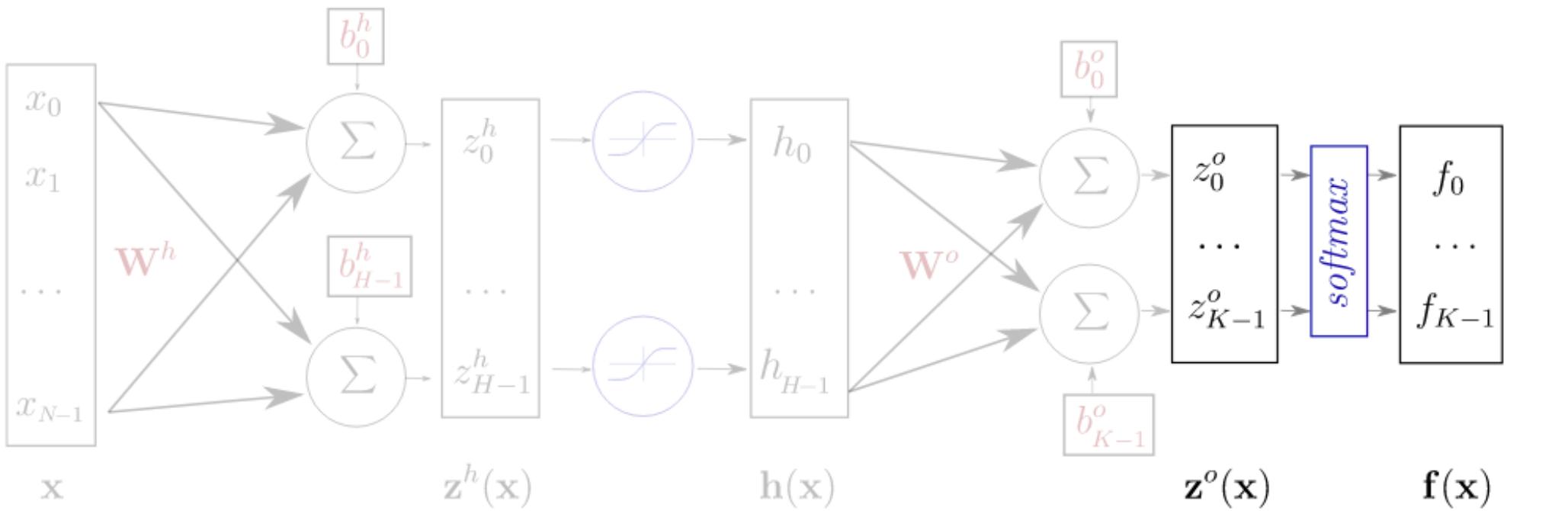
ACTIVATION FUNCTION



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



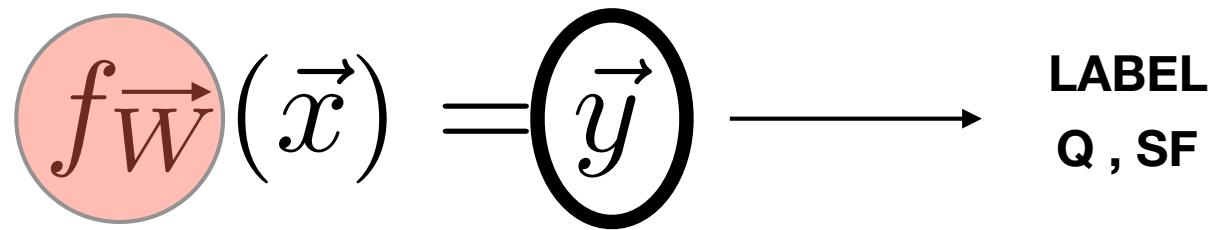
$$z^0(\mathbf{x}) = W^0 h(\mathbf{x}) + b^0$$



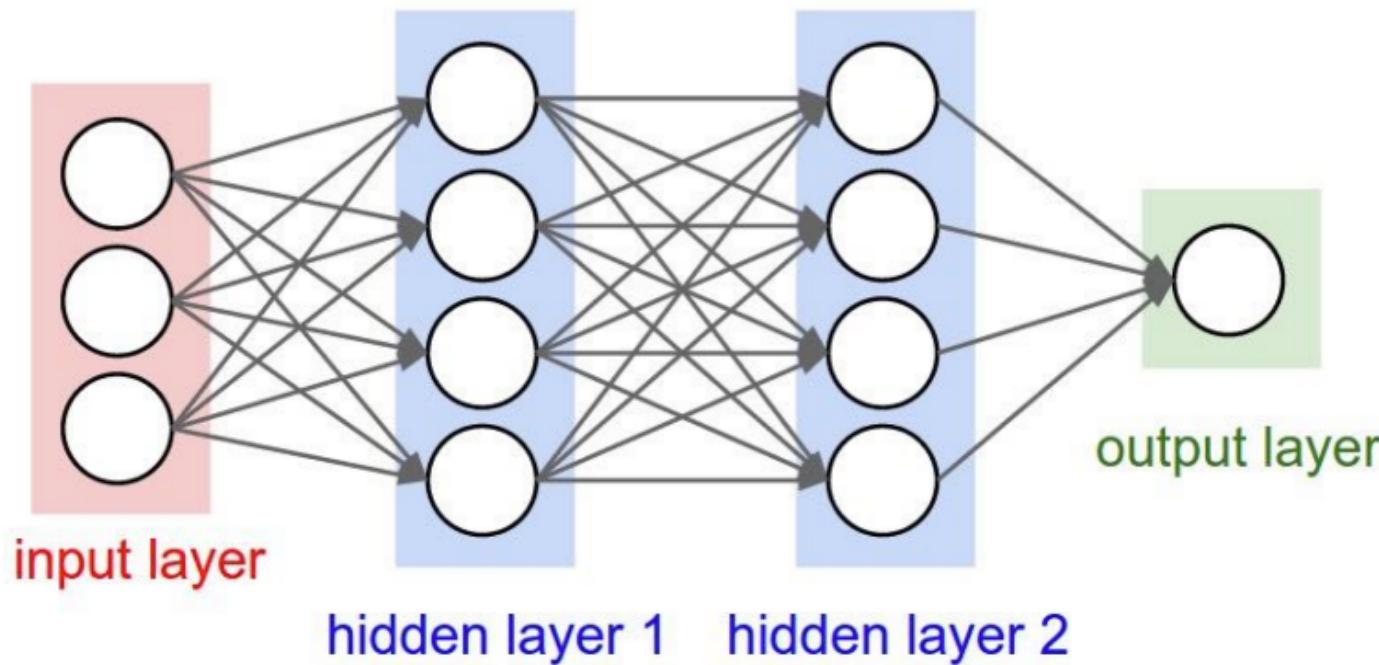
PREDICTION LAYER

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

**"CLASSICAL"
MACHINE LEARNING**

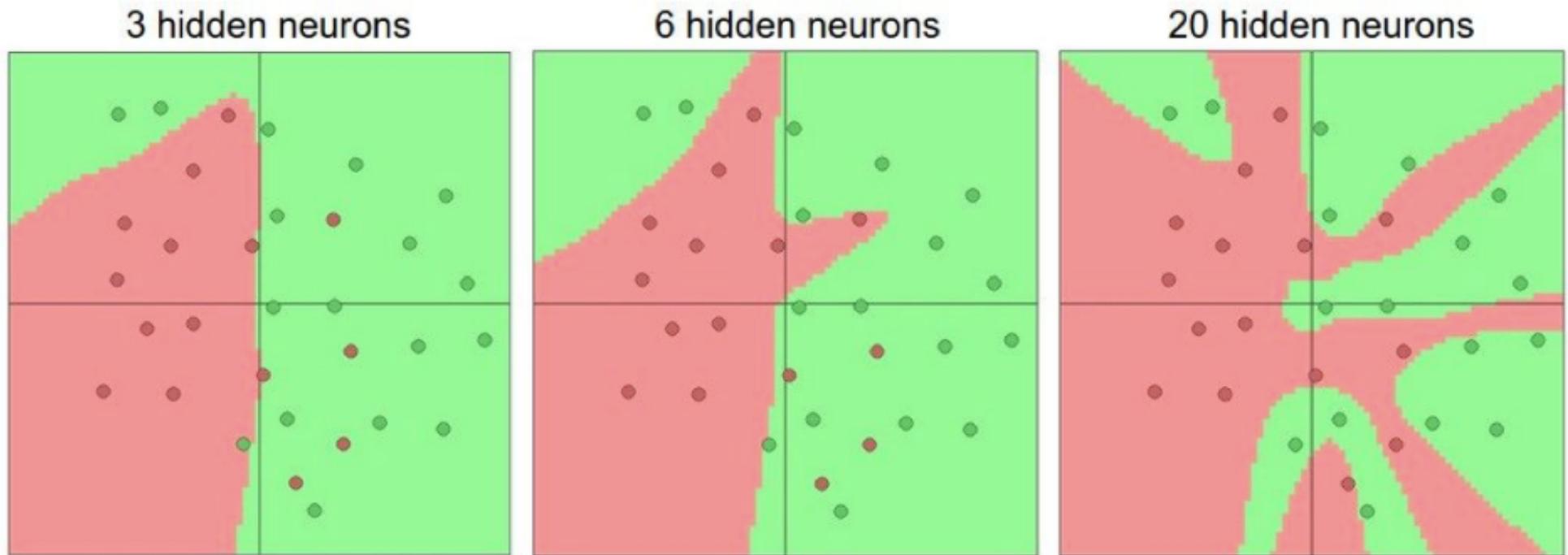


**REPLACE THIS BY A GENERAL
NON LINEAR FUNCTION WITH SOME PARAMETERS W**



$$p = g_3(W_3g_2(W_2g_1(W_1\vec{x}_0))) \xleftarrow{\text{NETWORK FUNCTION}}$$

Why hidden layers?



More complex functions allow increasing complexity

Credit: Karpathy

SO LET'S GO DEEPER AND DEEPER!

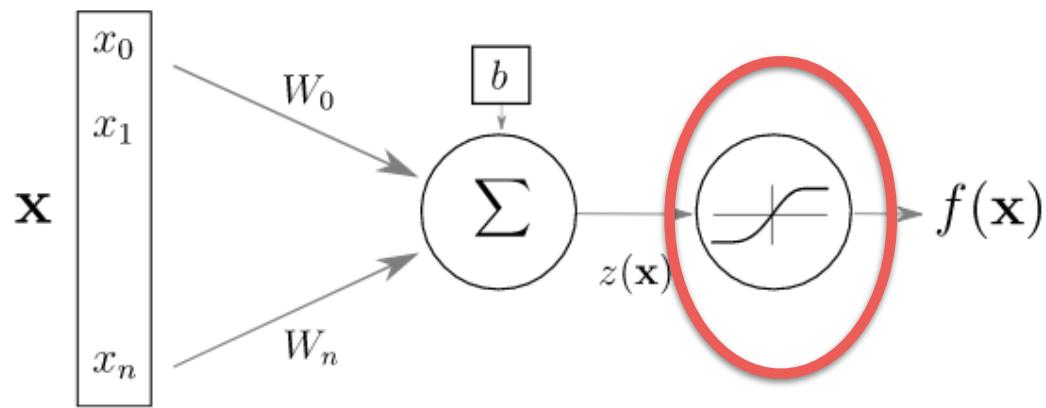
SO LET'S GO DEEPER AND DEEPER!

YES BUT...

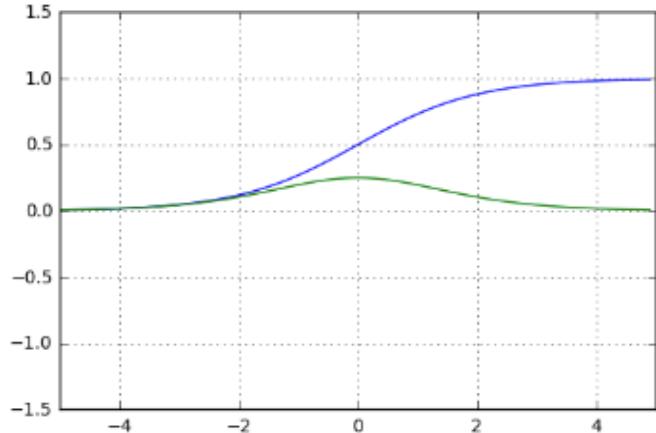
NOT SO STRAIGHTFORWARD, DEEPER MEANS MORE
WEIGHTS, MORE DIFFICULT OPTIMIZATION, OVERFITTING
RISKS...

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

Activation functions?

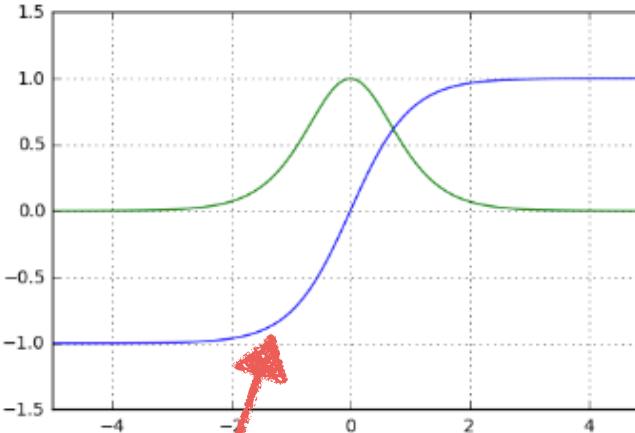


Activation functions



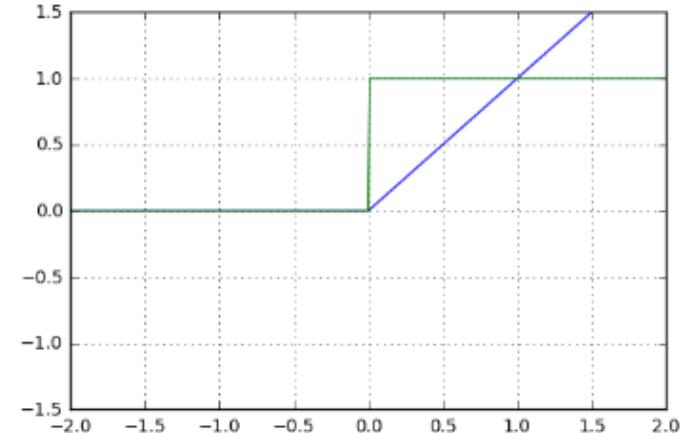
$$\text{sigm}(x) = \frac{1}{1 + e^{-x}}$$

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



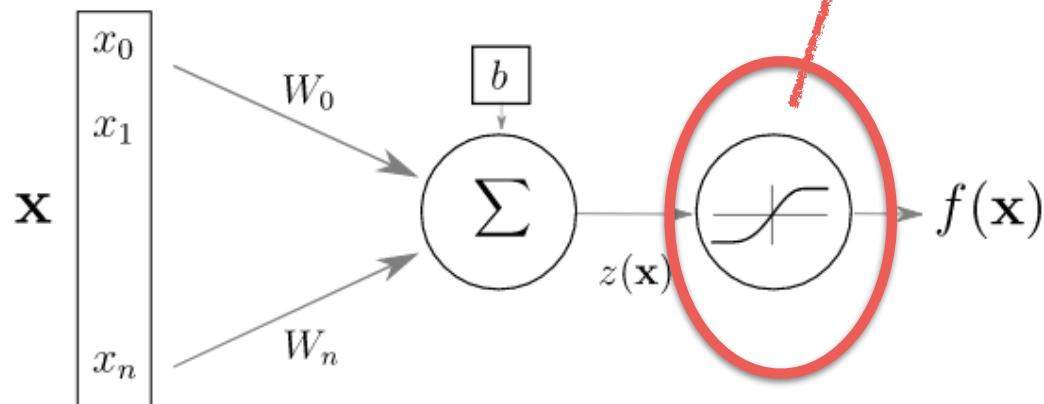
$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

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

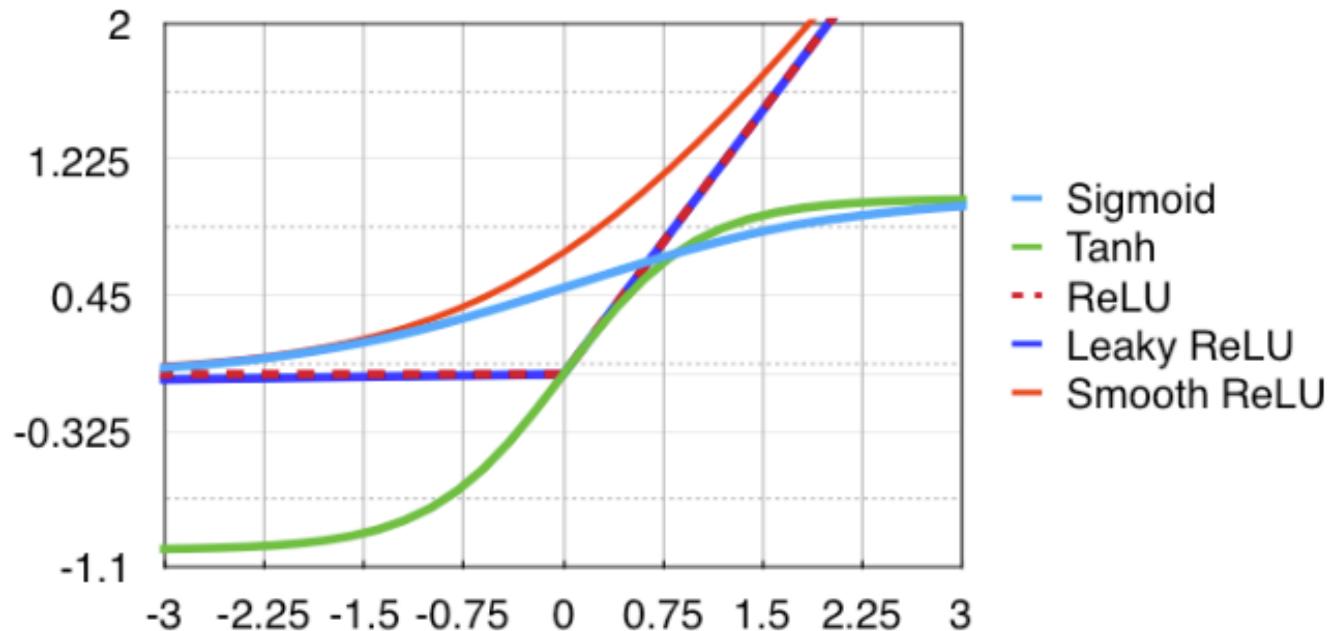


$$\text{relu}(x) = \max(0, x)$$

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



Activation functions



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

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

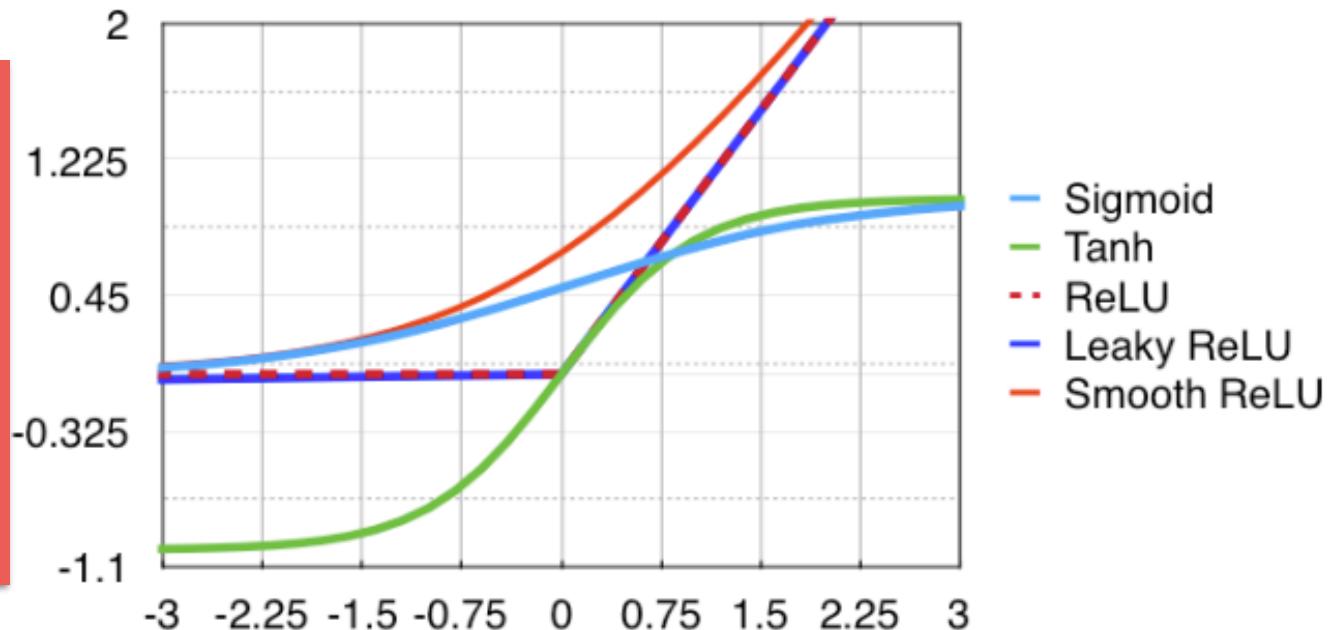
ReLU: $f(x) = \max(0, x)$

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

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

Activation functions

+
**MANY
OTHERS!**



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

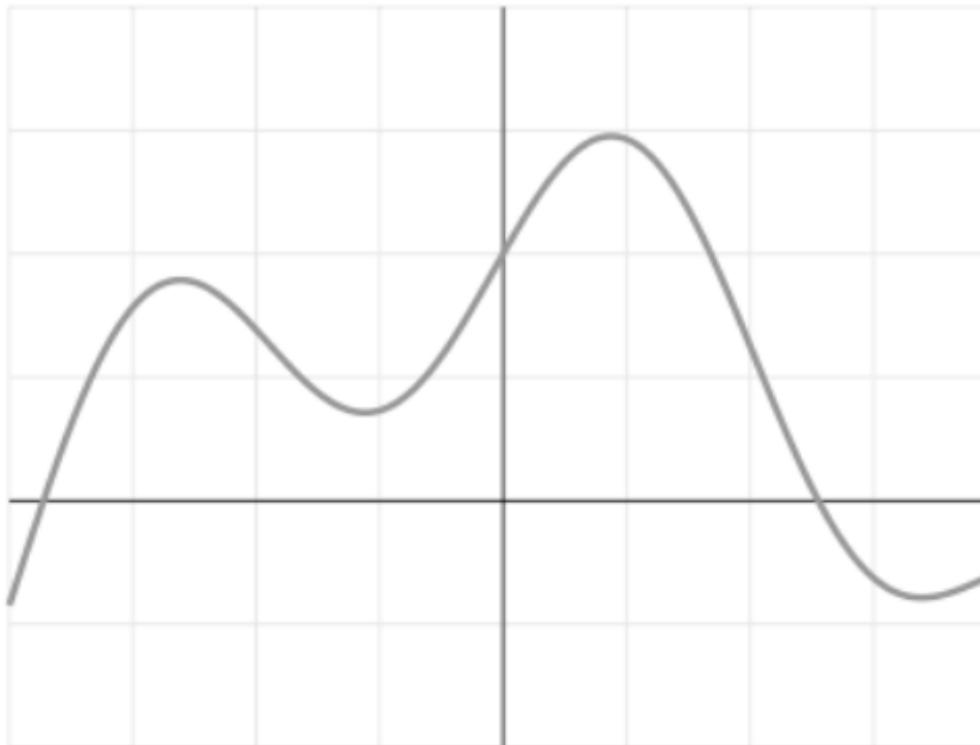
Tanh: $f(x) = \tanh(x)$

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

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

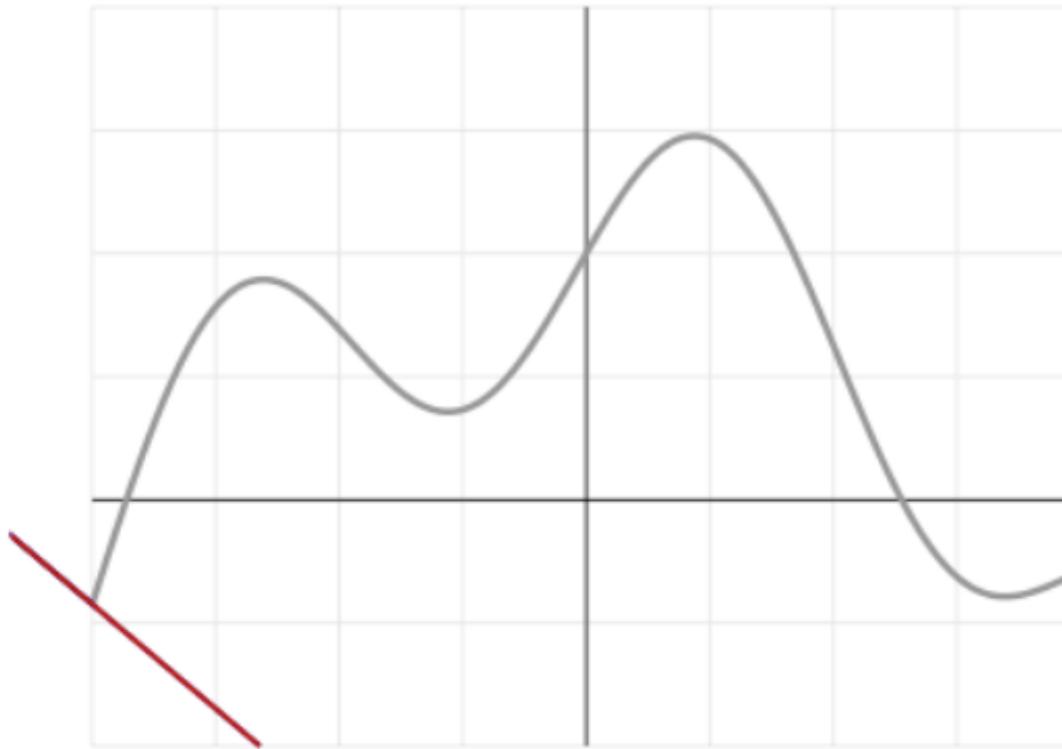
Leaky ReLU: $f(x) = \epsilon x + (1 - \epsilon) \max(0, x)$

What is the meaning of the activation function?



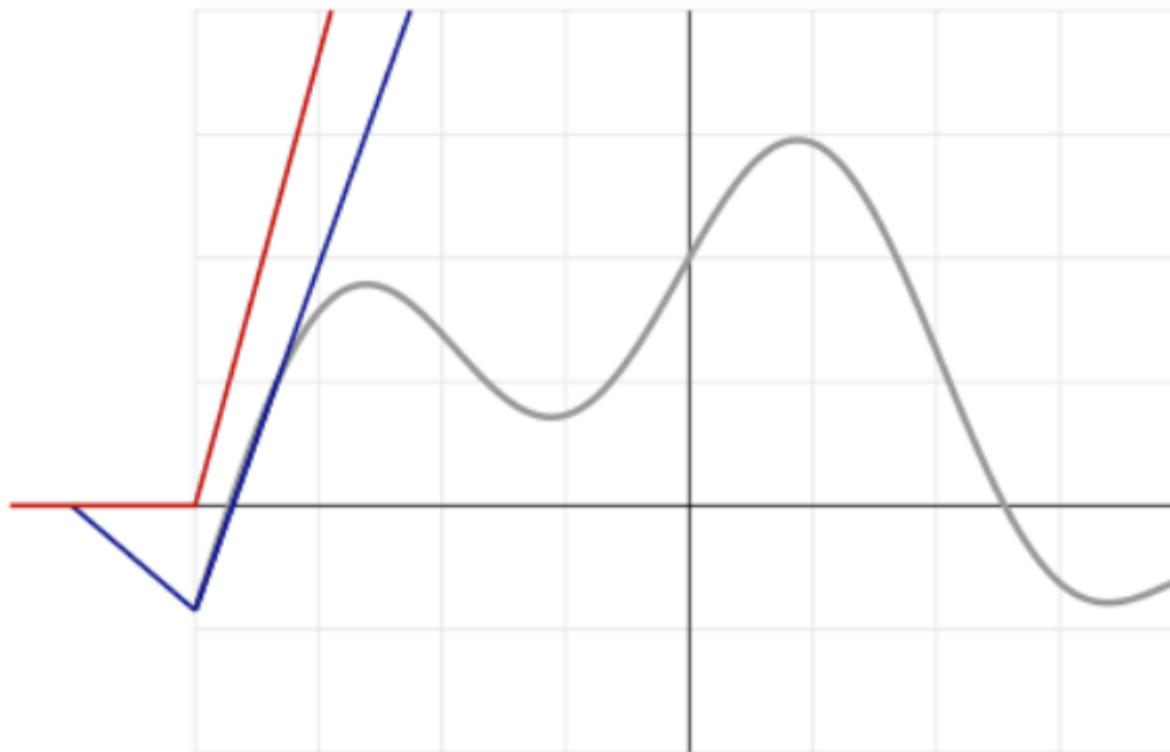
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



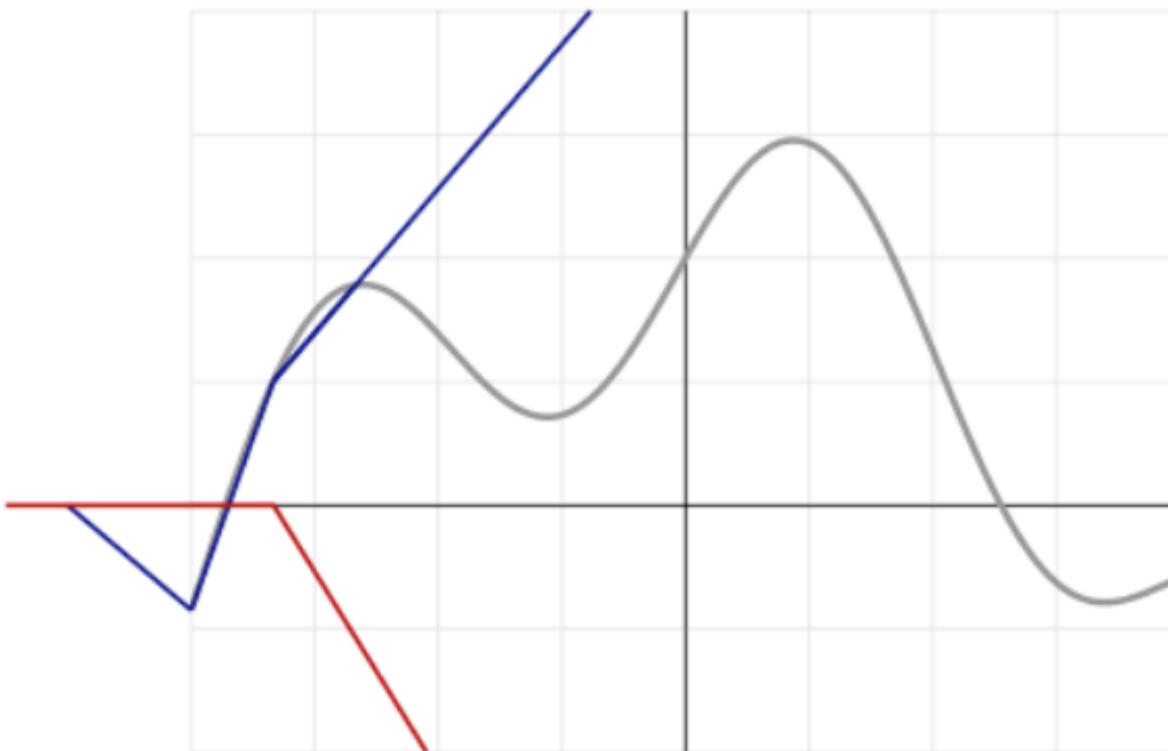
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



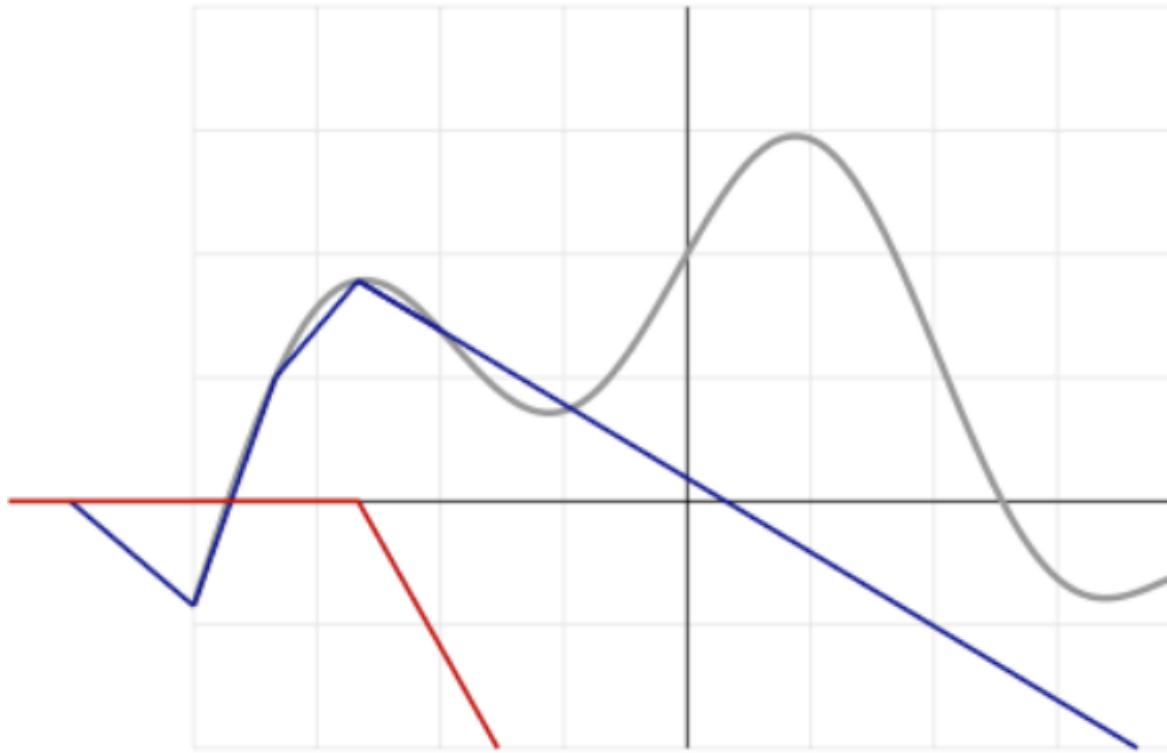
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



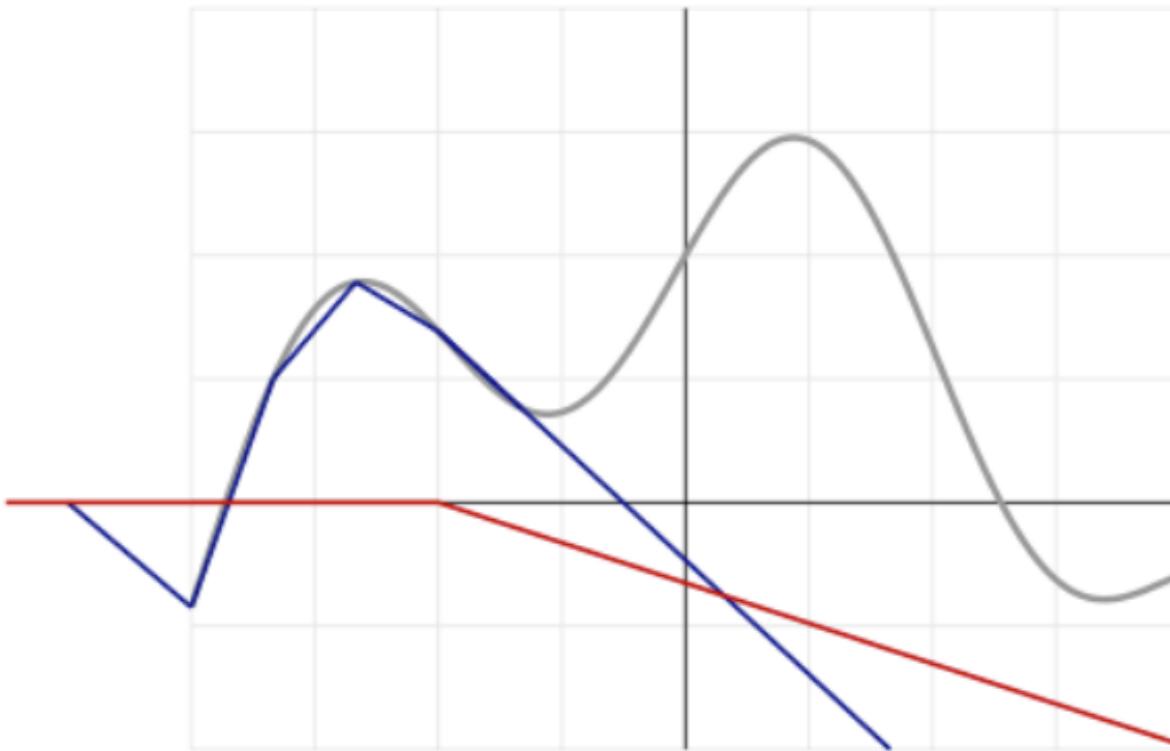
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



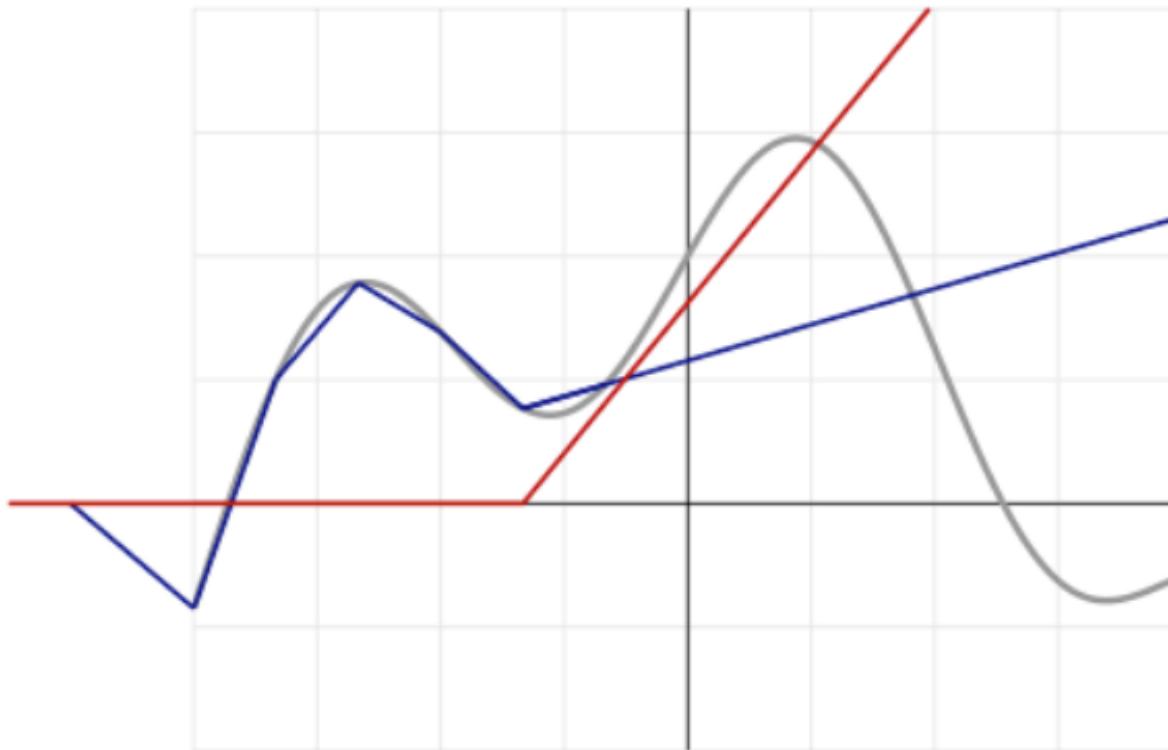
Any real function in an interval (a, b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



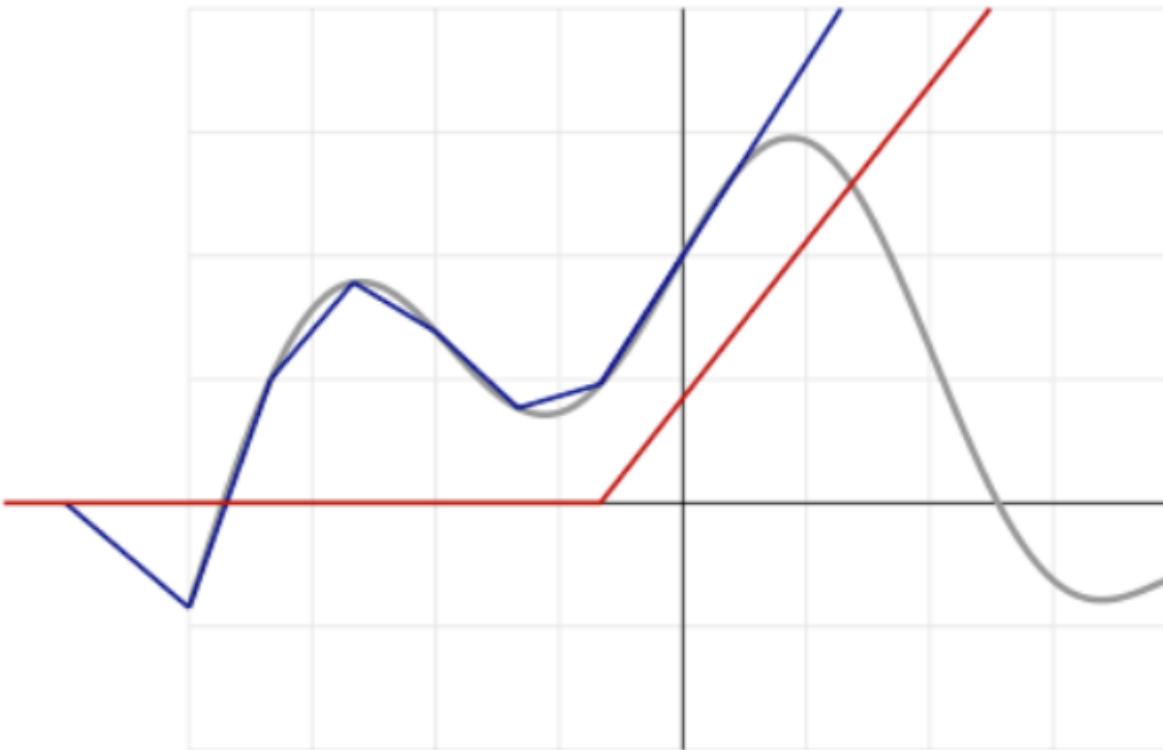
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



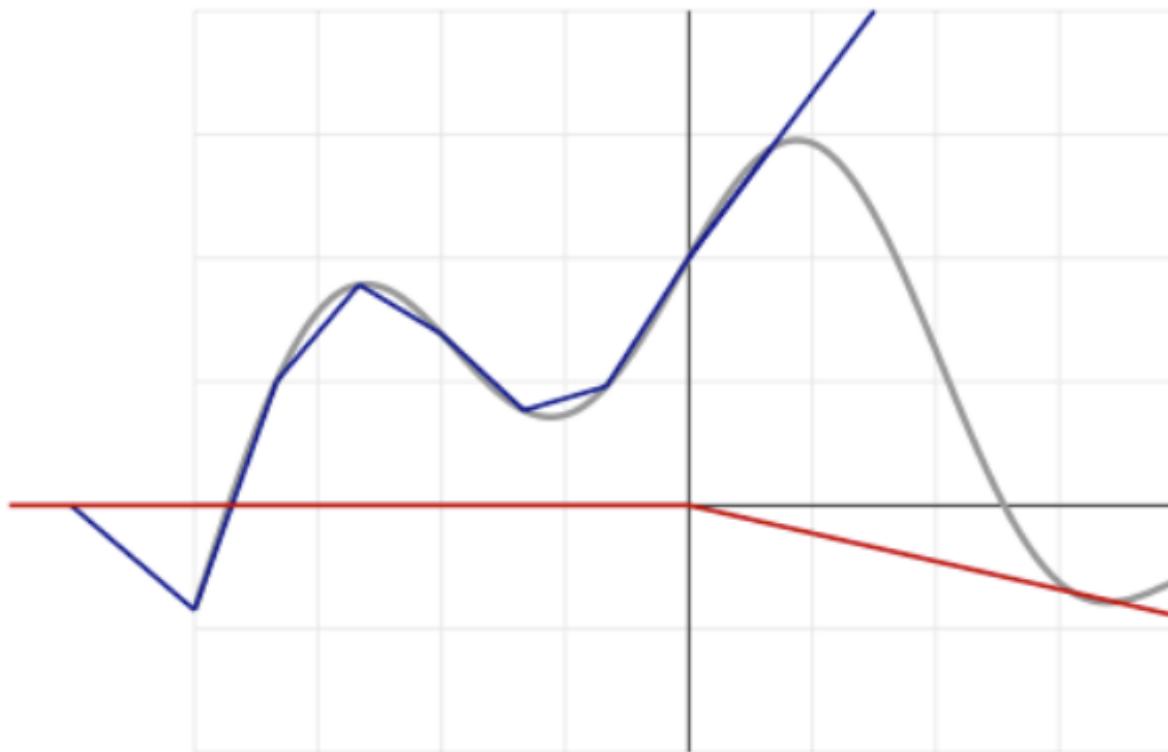
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



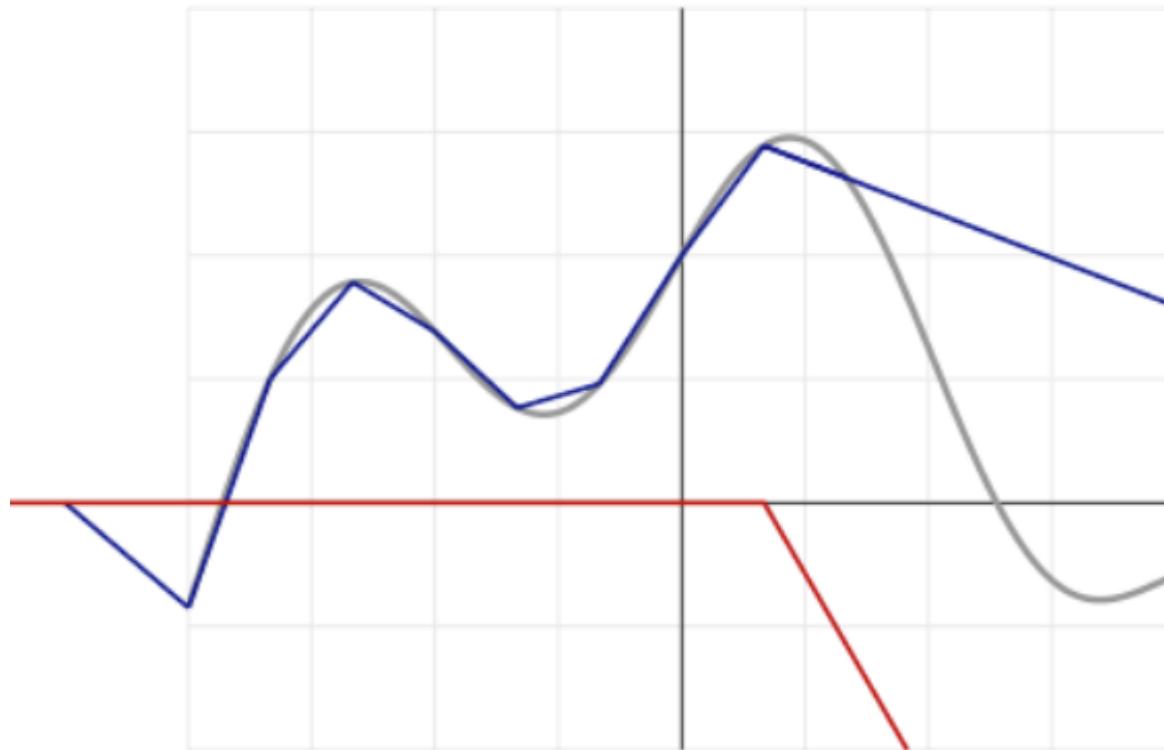
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



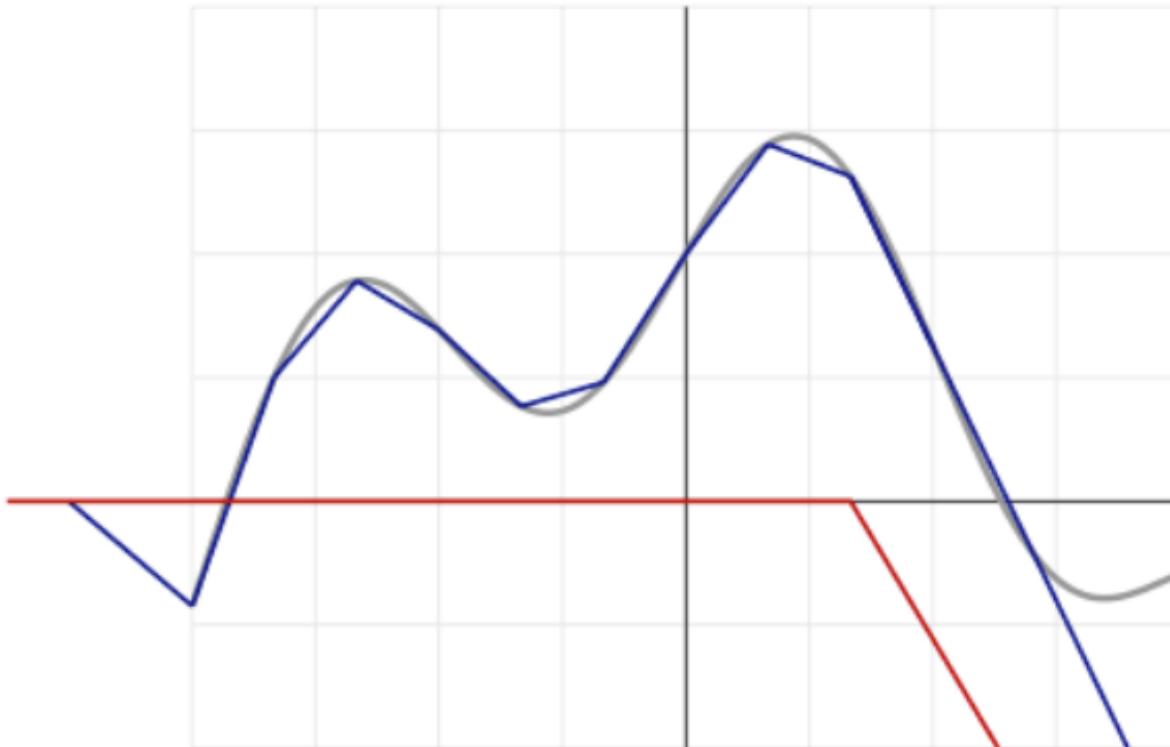
Any real function in a interval (a, b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



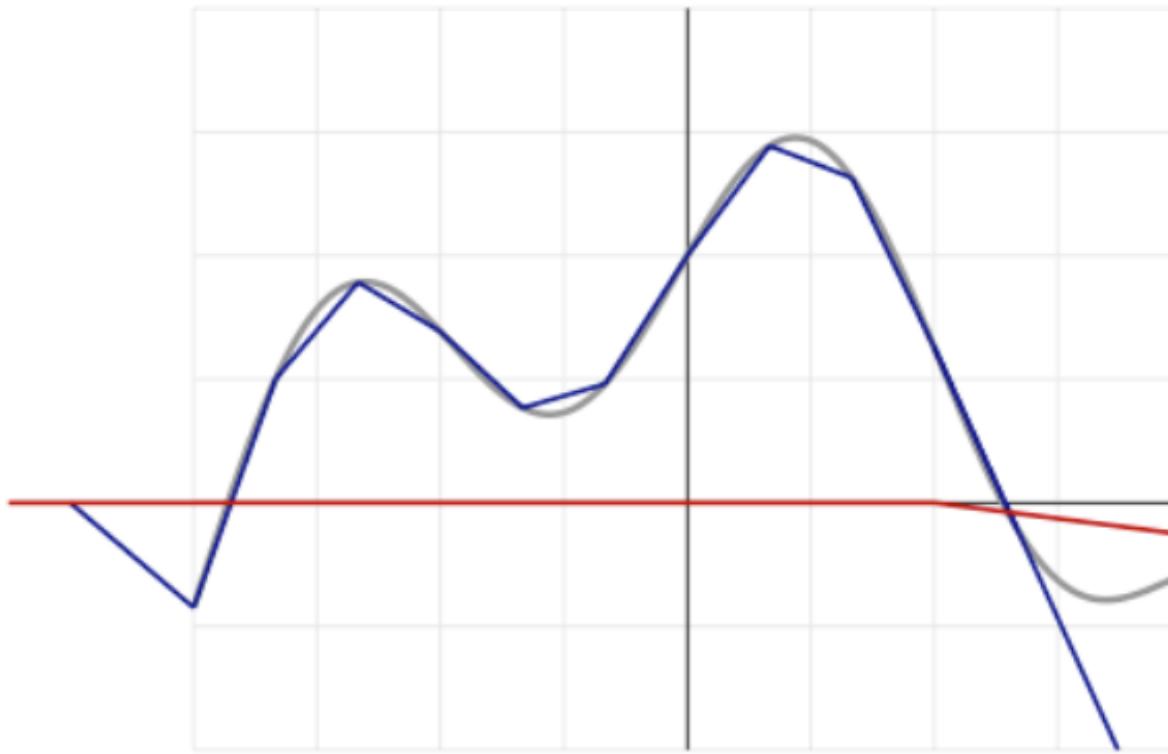
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



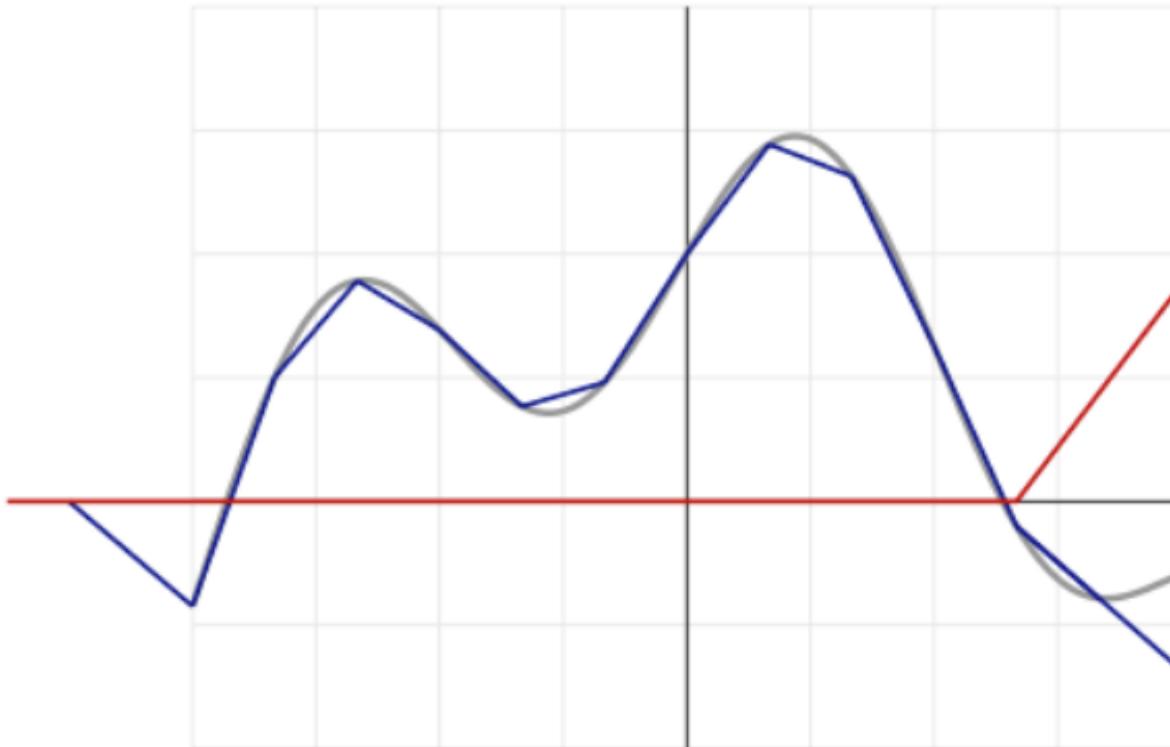
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



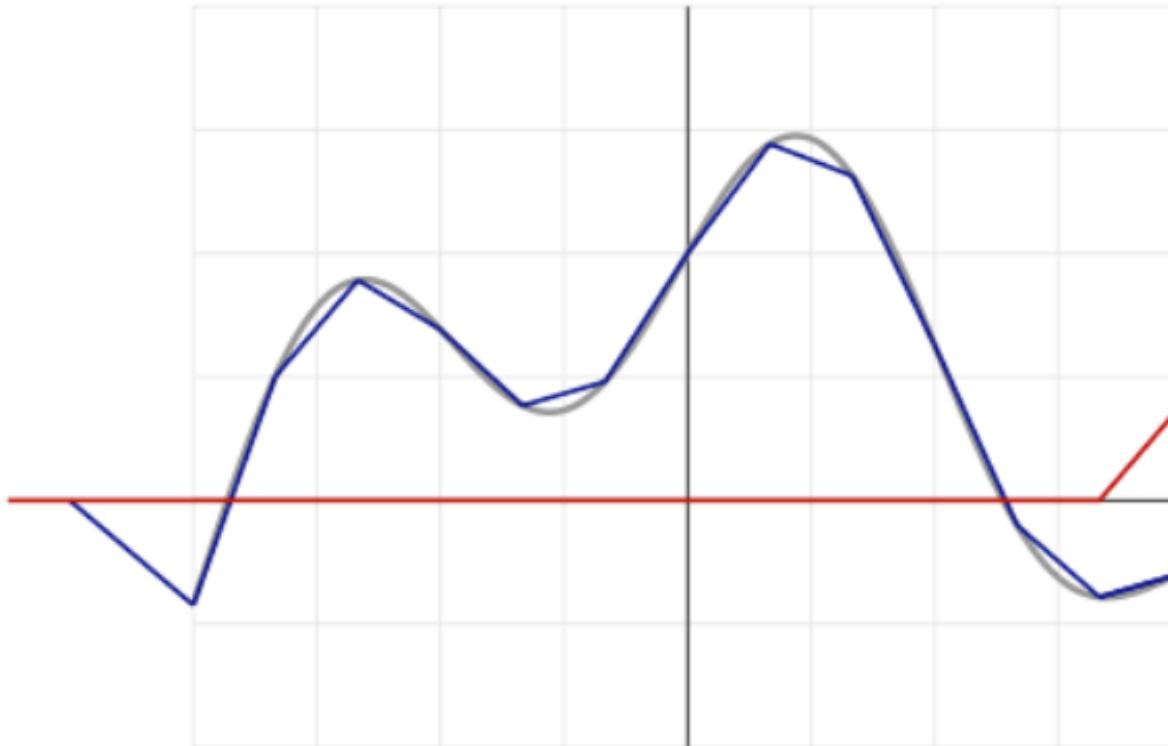
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



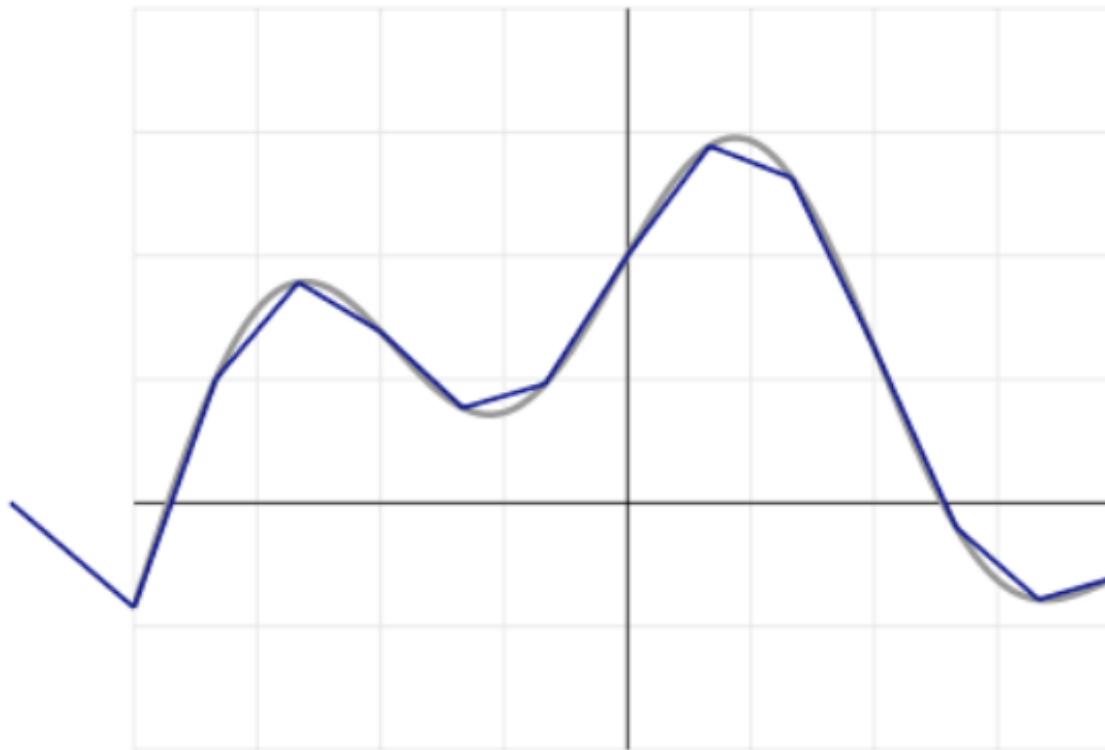
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



Any real function in a interval (a, b) can be approximated with a linear combination of translated and scaled ReLu functions

What is the meaning of the activation function?



Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

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

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

SOFTMAX

A generalization of the SIGMOID ACTIVATION

THE OUTPUT IS A VECTOR OF PROBABILITIES
AND 1

GENERALLY
USED AS ACTIVATION
OF LAST LAYER

(will come back later)

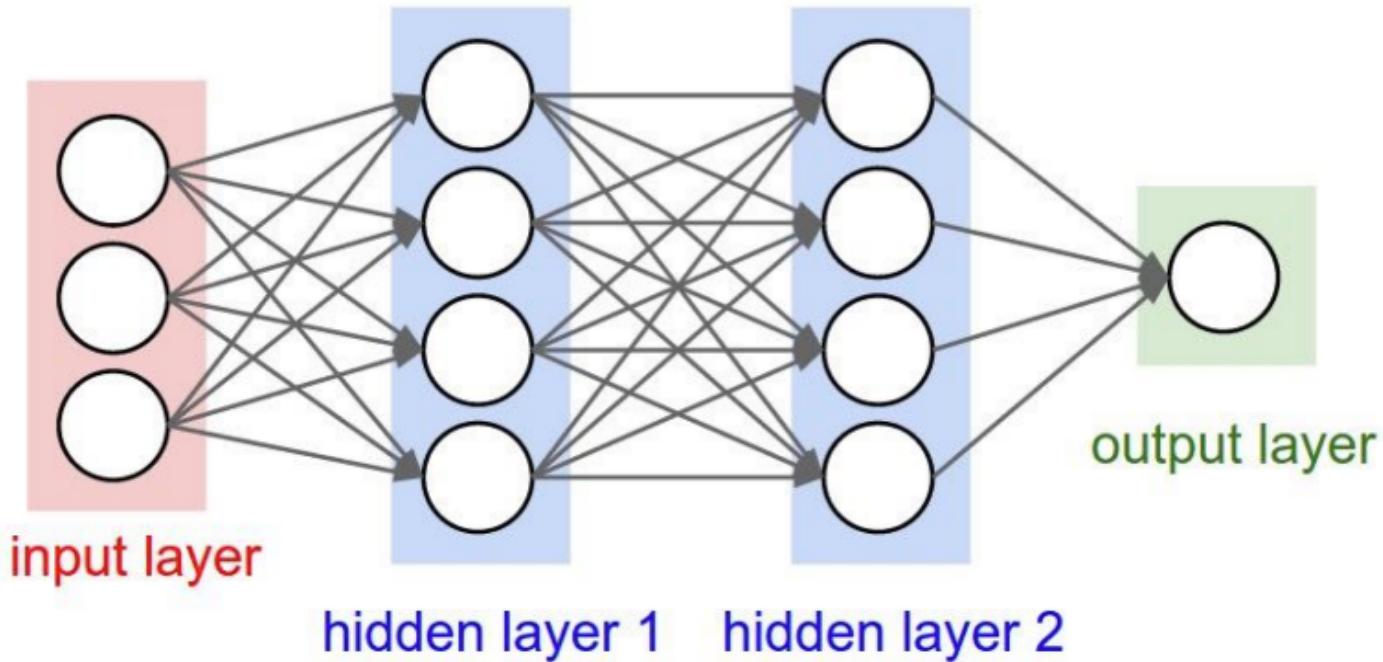
CAN BE INTERPRETED AS A PROBABILITY

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

OK, SO NOW LET'S FIND
THE WEIGHTS

OPTIMIZATION

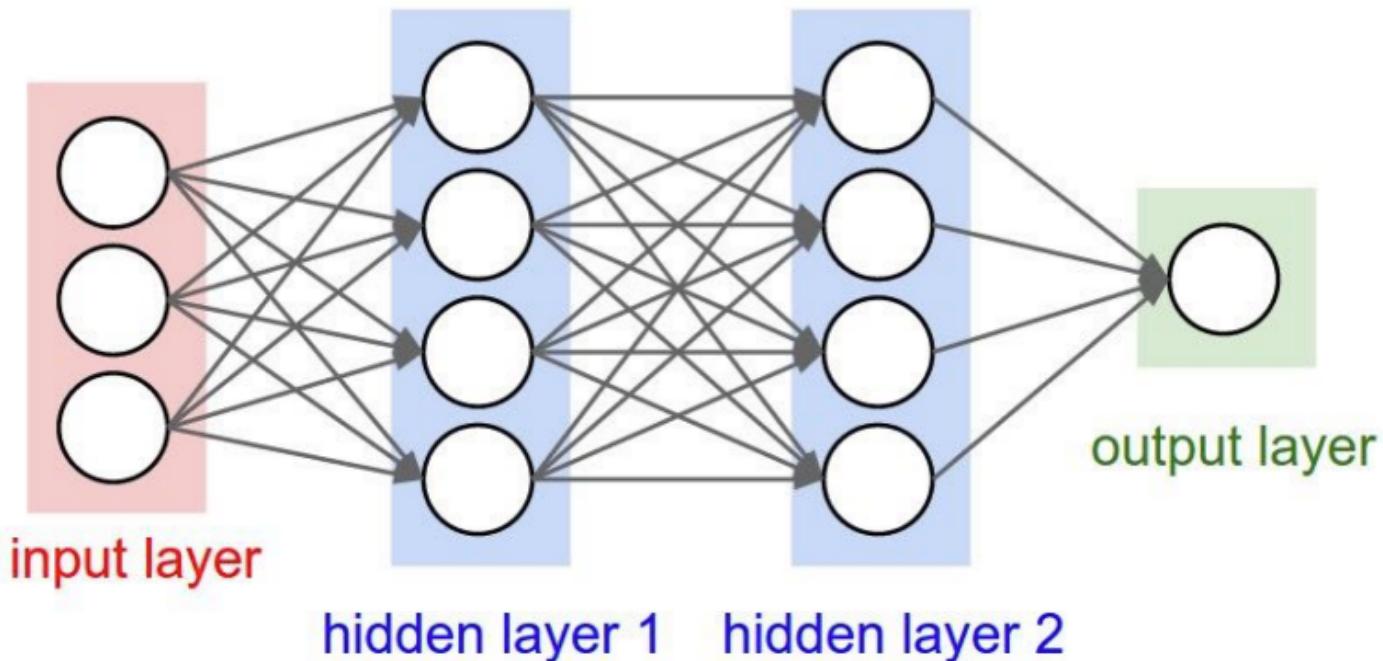
[OR HOW TO FIND THE WEIGHTS?]



$$p = g_3(W_3g_2(W_2g_1(W_1\vec{x}_0))) \xleftarrow{\text{NETWORK FUNCTION}}$$

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

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

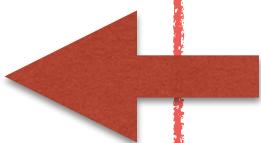


[hessian]

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

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

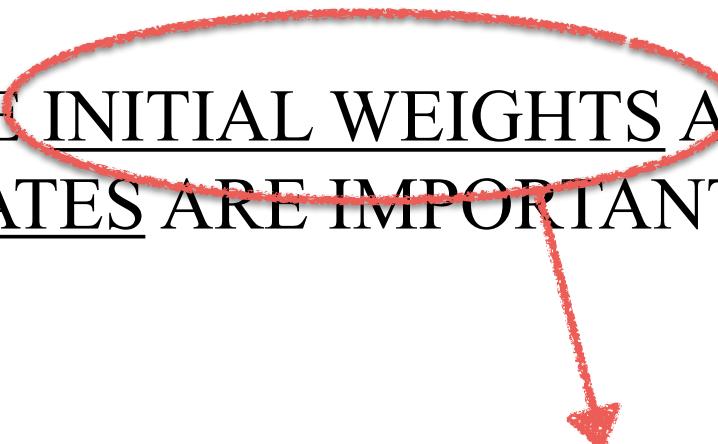
ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT
EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING
THE STEEPEST DESCENT

ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT
EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING
THE STEEPEST DESCENT

THE CHOICES OF THE INITIAL WEIGHTS AND THE
LEARNING RATES ARE IMPORTANT

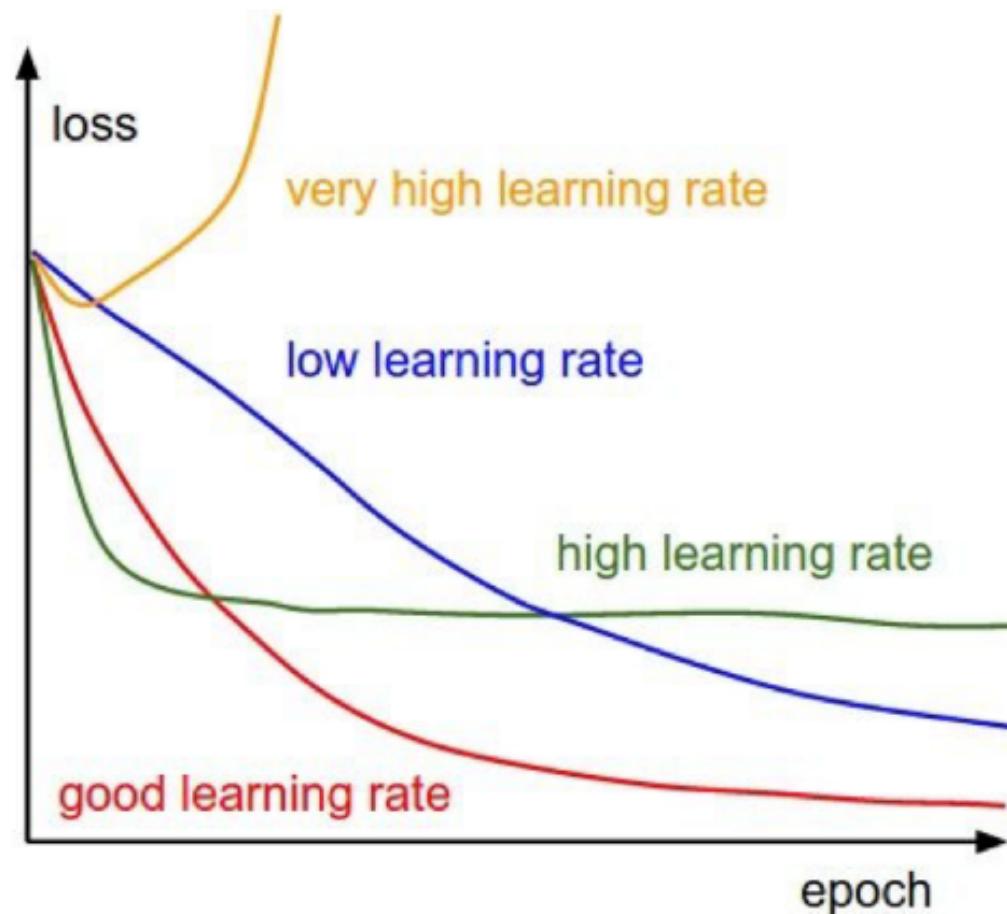
ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT
EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING
THE STEEPEST DESCENT

THE CHOICES OF THE INITIAL WEIGHTS AND THE
LEARNING RATES ARE IMPORTANT



WE WILL TALK ABOUT
THIS LATER

LEARNING RATES



Credit:

LEARNING RATES

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



THERE ARE DIFFERENT WAYS
TO UPDATE THE LEARNING RATE

Credit:

LEARNING RATES

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



THERE ARE DIFFERENT WAYS
TO UPDATE THE LEARNING RATE

ADAGRAD:

THE LEARNING RATE IS SCALED DEPENDING ON THE HISTORY OF PREVIOUS GRADIENTS

$$W_{t+1} = W_t - \frac{\lambda}{\sqrt{G_t + \epsilon}} \nabla f(W_t)$$

G IS A MATRIX CONTAINING ALL PREVIOUS GRADIENTS. WHEN THE GRADIENT BECOMES LARGE THE LEARNING RATE IS DECREASED AND VICE VERSA.

$$G_{t+1} = G_t + (\nabla f)^2$$

Credit:

LEARNING RATES

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



THERE ARE DIFFERENT WAYS
TO UPDATE THE LEARNING RATE

RMSPROP:

THE LEARNING RATE IS SCALED DEPENDING ON THE HISTORY OF PREVIOUS GRADIENTS

$$W_{t+1} = W_t - \frac{\lambda}{\sqrt{G_t + \epsilon}} \nabla f(W_t)$$

SAME AS ADAGRAD BUT G IS CALCULATED BY EXPONENTIALLY DECAYING AVERAGE

$$G_{t+1} = \lambda G_t + (1 - \lambda)(\nabla f)^2$$

Credit:

ADAM [Adaptive moment estimator]:

SAME IDEA, USING FIRST AND SECOND ORDER
MOMENTUMS

$$G_{t+1} = \beta_2 G_t + (1 - \beta_2)(\nabla f)^2 \quad M_{t+1} = \beta_1 M_t + (1 - \beta_1)(\nabla f)$$

$$W_{t+1} = W_t - \frac{\lambda}{\sqrt{\hat{G}_t + \epsilon}} \hat{M}_t$$

with: $\hat{M}_{t+1} = \frac{M_t}{1 - \beta_1}$ $\hat{G}_{t+1} = \frac{G_t}{1 - \beta_2}$

ADAM [Adaptive moment estimator]:

SAME IDEA, USING FIRST AND SECOND ORDER
MOMENTUMS

$$G_{t+1} = \beta_2 G_t + (1 - \beta_2)(\nabla f)^2 \quad M_{t+1} = \beta_1 M_t + (1 - \beta_1)(\nabla f)$$

ONLY FOR YOUR
RECORDS

$$\sqrt{G_t} + \epsilon$$

with: $\hat{M}_{t+1} = \frac{M_t}{1 - \beta_1}$ $\hat{G}_{t+1} = \frac{G_t}{1 - \beta_2}$

IN KERAS:

RMSprop

[source]

```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

RMSProp optimizer.

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned).

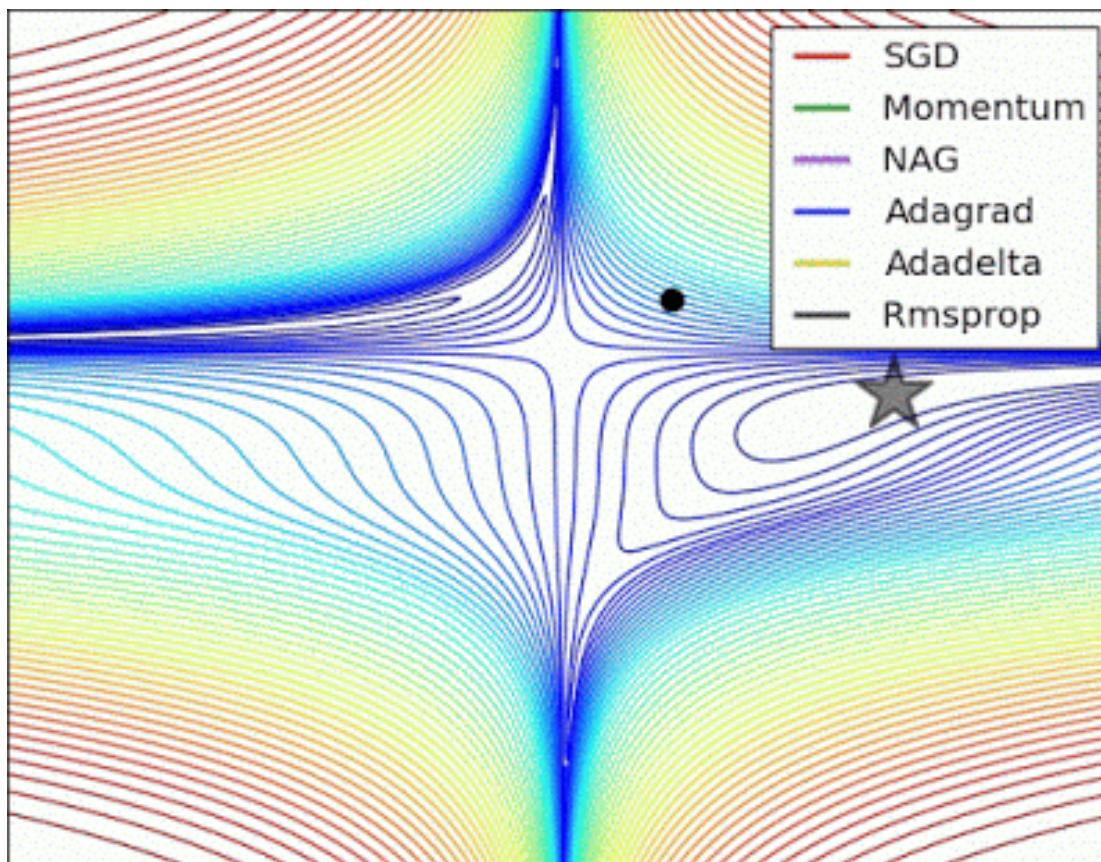
This optimizer is usually a good choice for recurrent neural networks.

Arguments

- `lr`: float ≥ 0 . Learning rate.
- `rho`: float ≥ 0 .
- `epsilon`: float ≥ 0 . Fuzz factor. If `None`, defaults to `K.epsilon()`.
- `decay`: float ≥ 0 . Learning rate decay over each update.

References

- [rmsprop](#): Divide the gradient by a running average of its recent magnitude



Credit

BATCH GRADIENT DESCENT

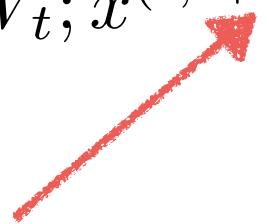
LOCAL MINIMA CAN ALSO BE AVOIDED BY COMPUTING THE
GRADIENT IN SMALL BATCHES INSTEAD OF OVER THE FULL
DATASET

BATCH GRADIENT DESCENT

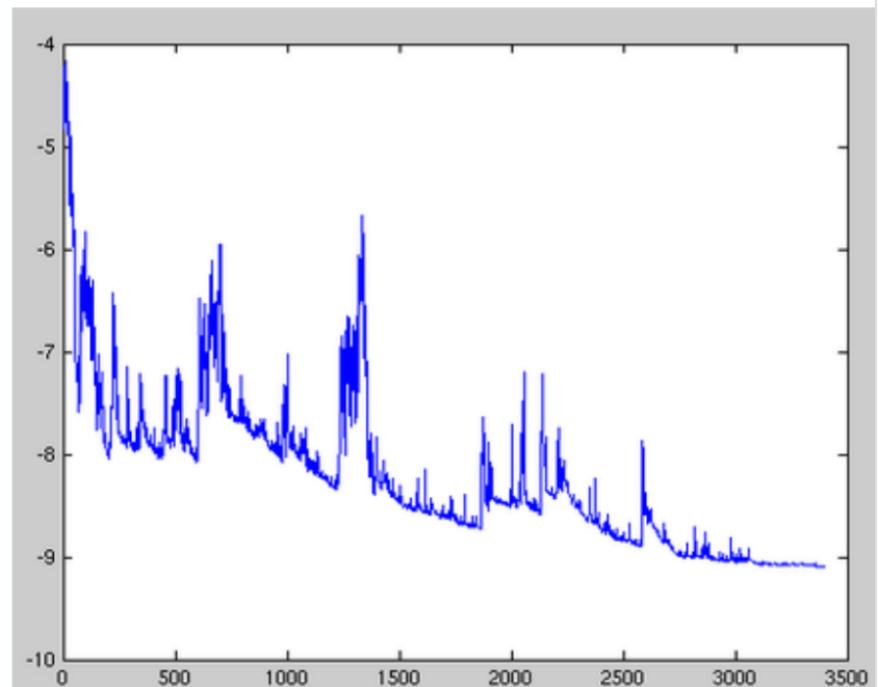
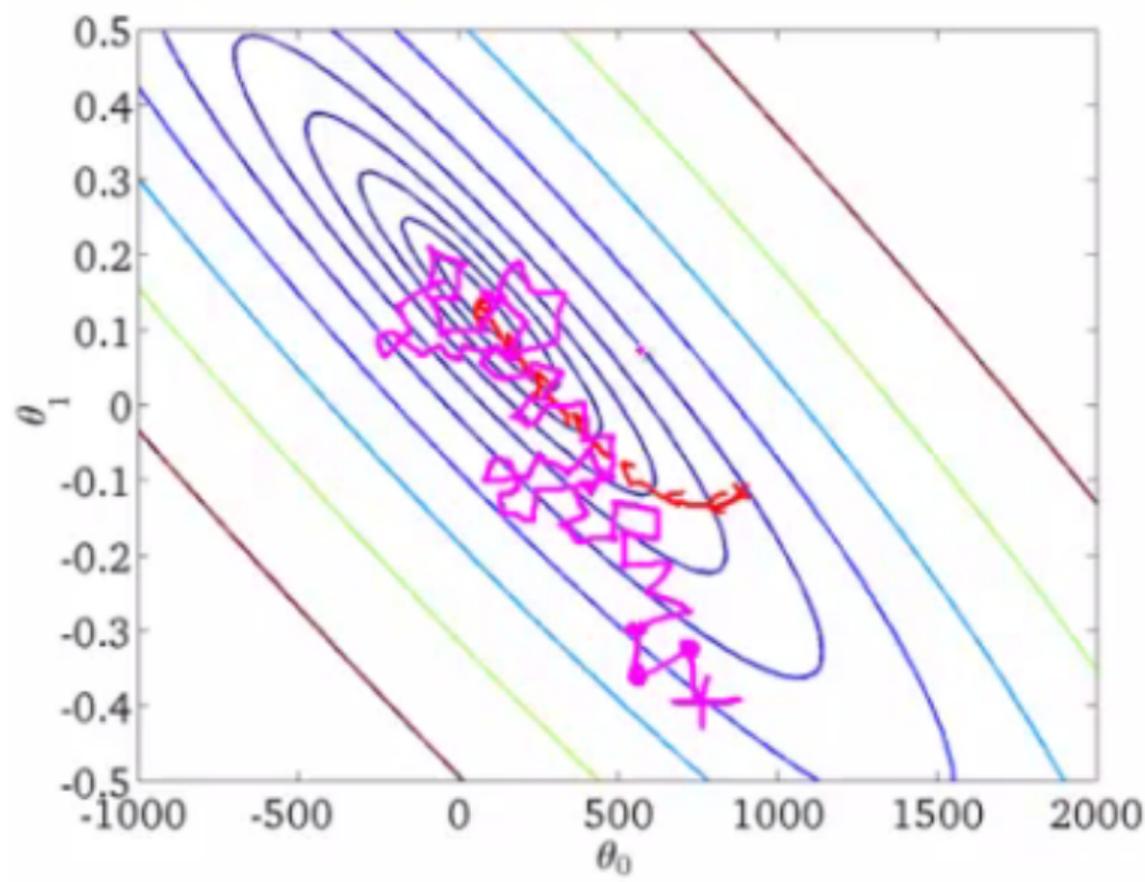
LOCAL MINIMA CAN ALSO BE AVOIDED BY COMPUTING THE GRADIENT IN SMALL BATCHES INSTEAD OF OVER THE FULL DATASET

MINI-BATCH GRADIENT DESCENT

$$W_{t+1/num} = W_t - \lambda_t \nabla f(W_t; x^{(i,i+b)}, y^{(i,i+b)})$$



THE GRADIENT IS COMPUTED OVER A BATCH OF SIZE B



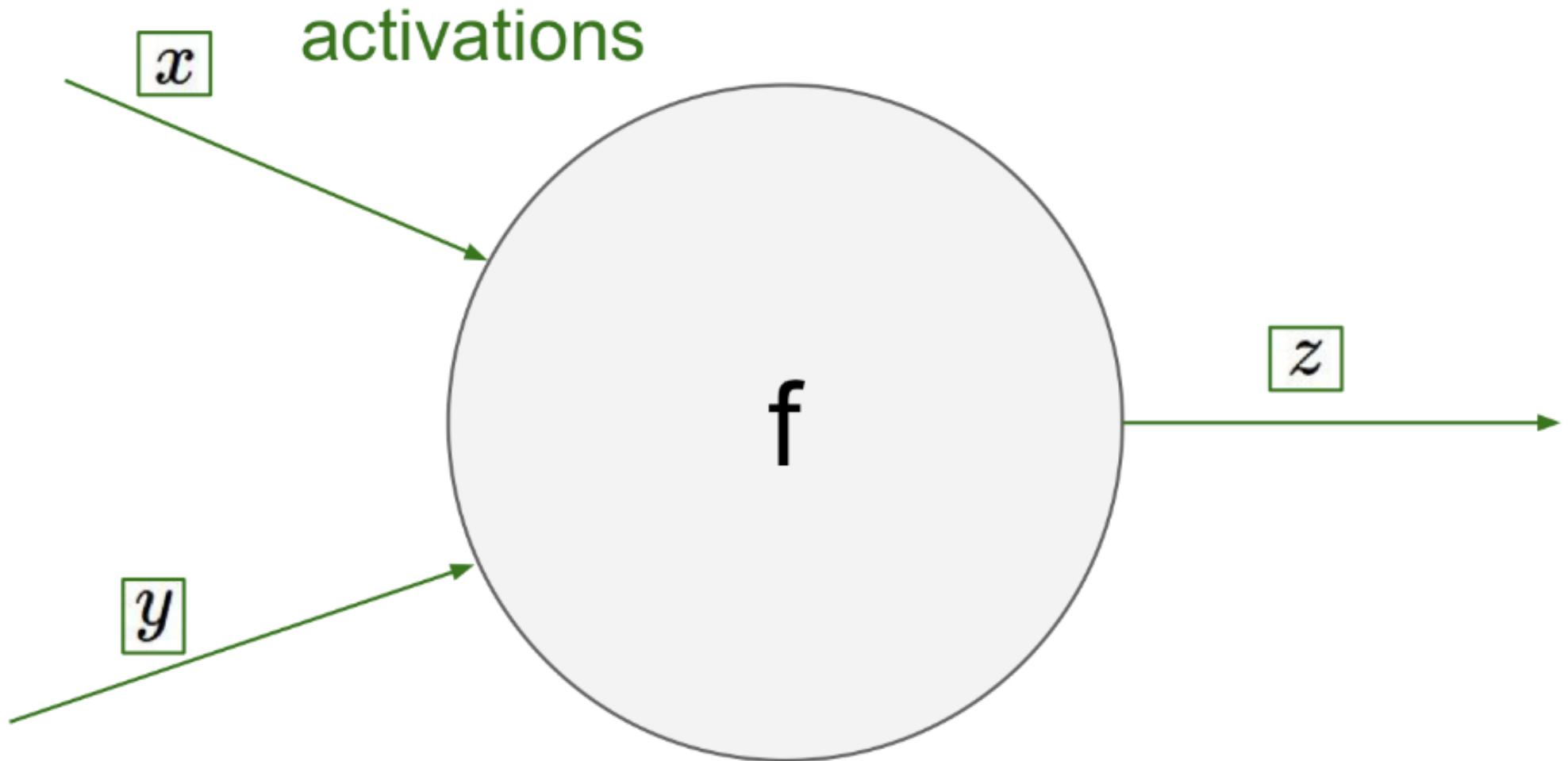
Fluctuations in the total objective function as gradient steps with respect to mini-batches are taken.

Credit

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

BACKPROPAGATION

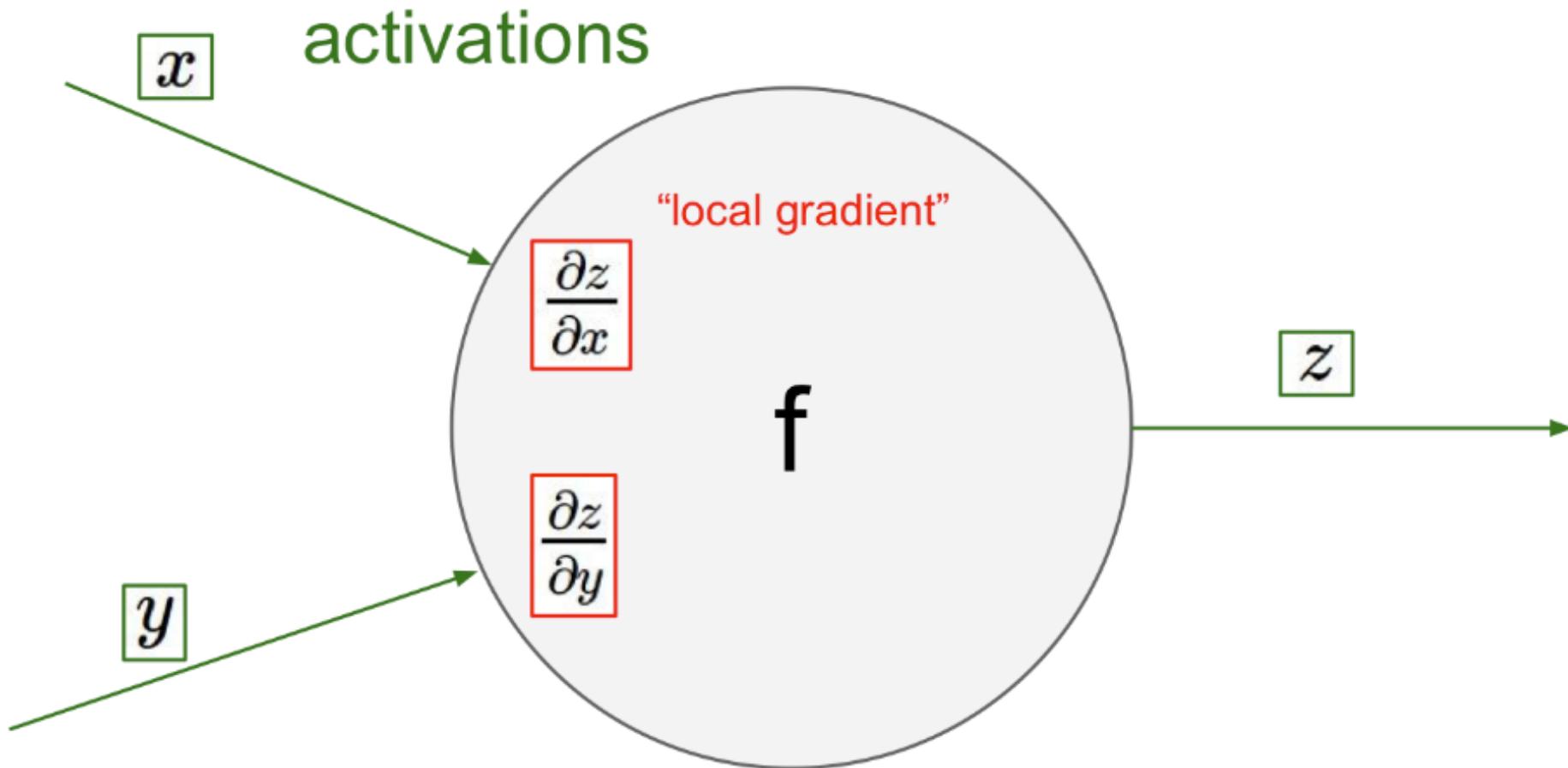
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

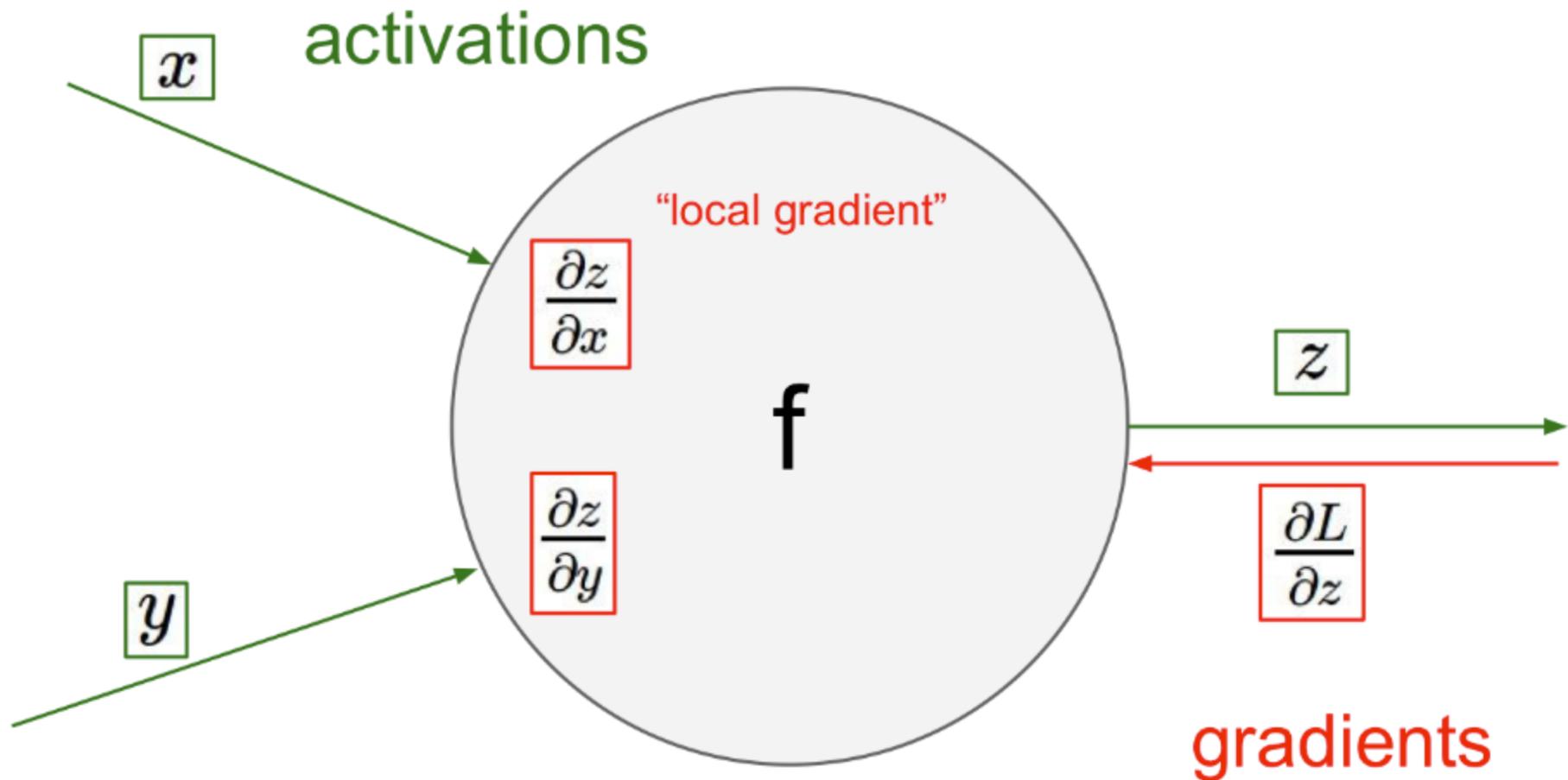
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

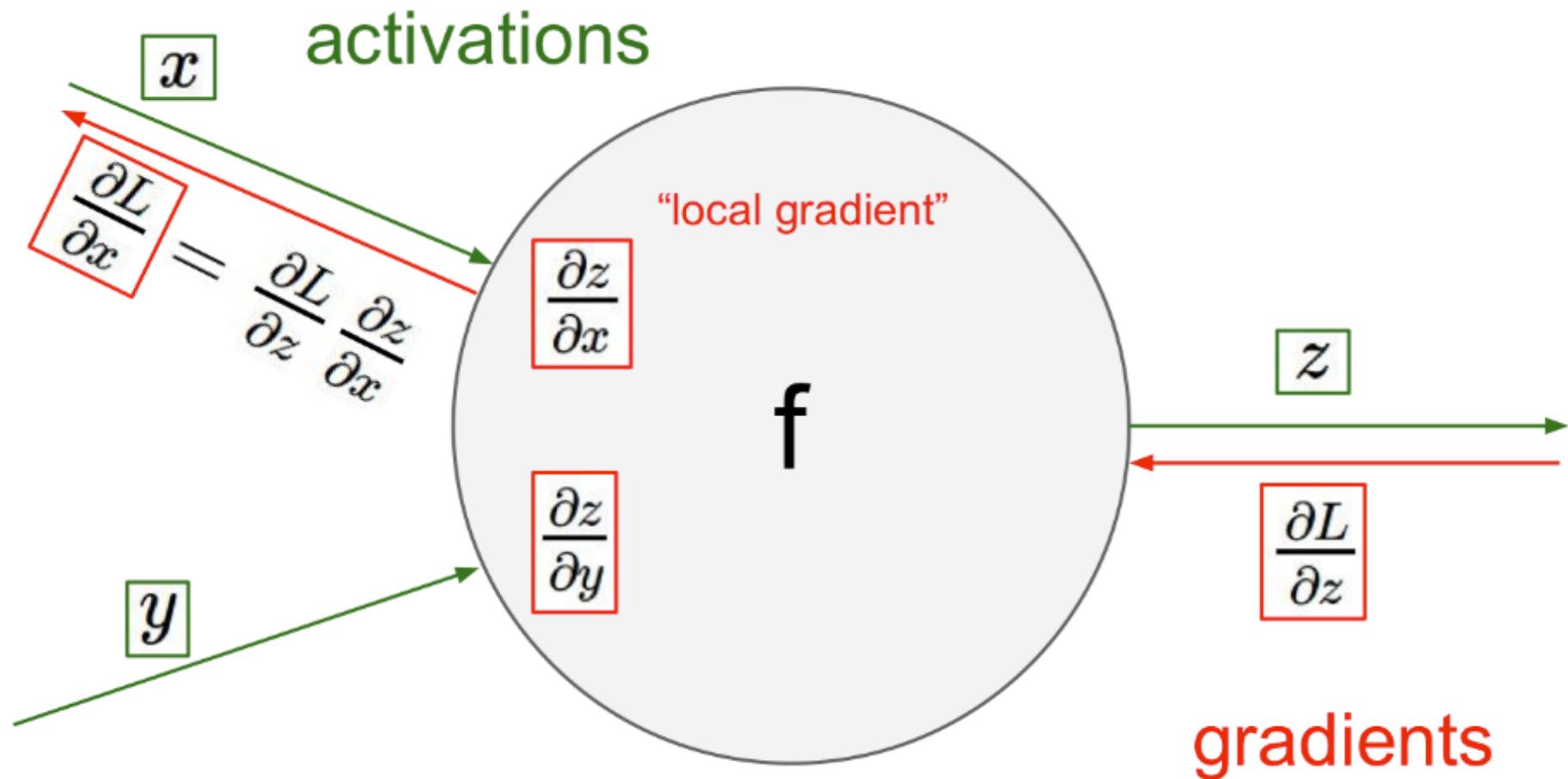
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

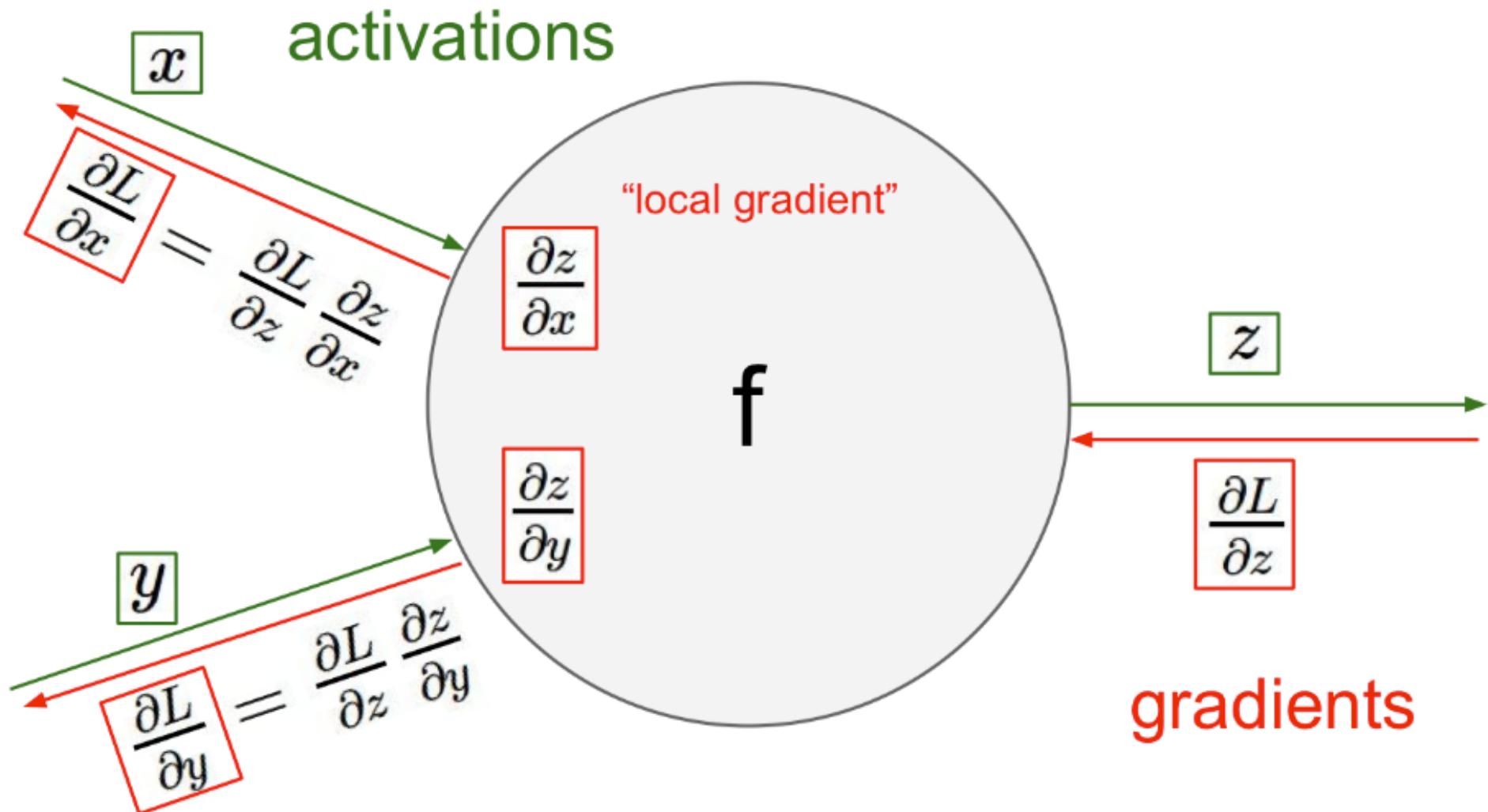
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

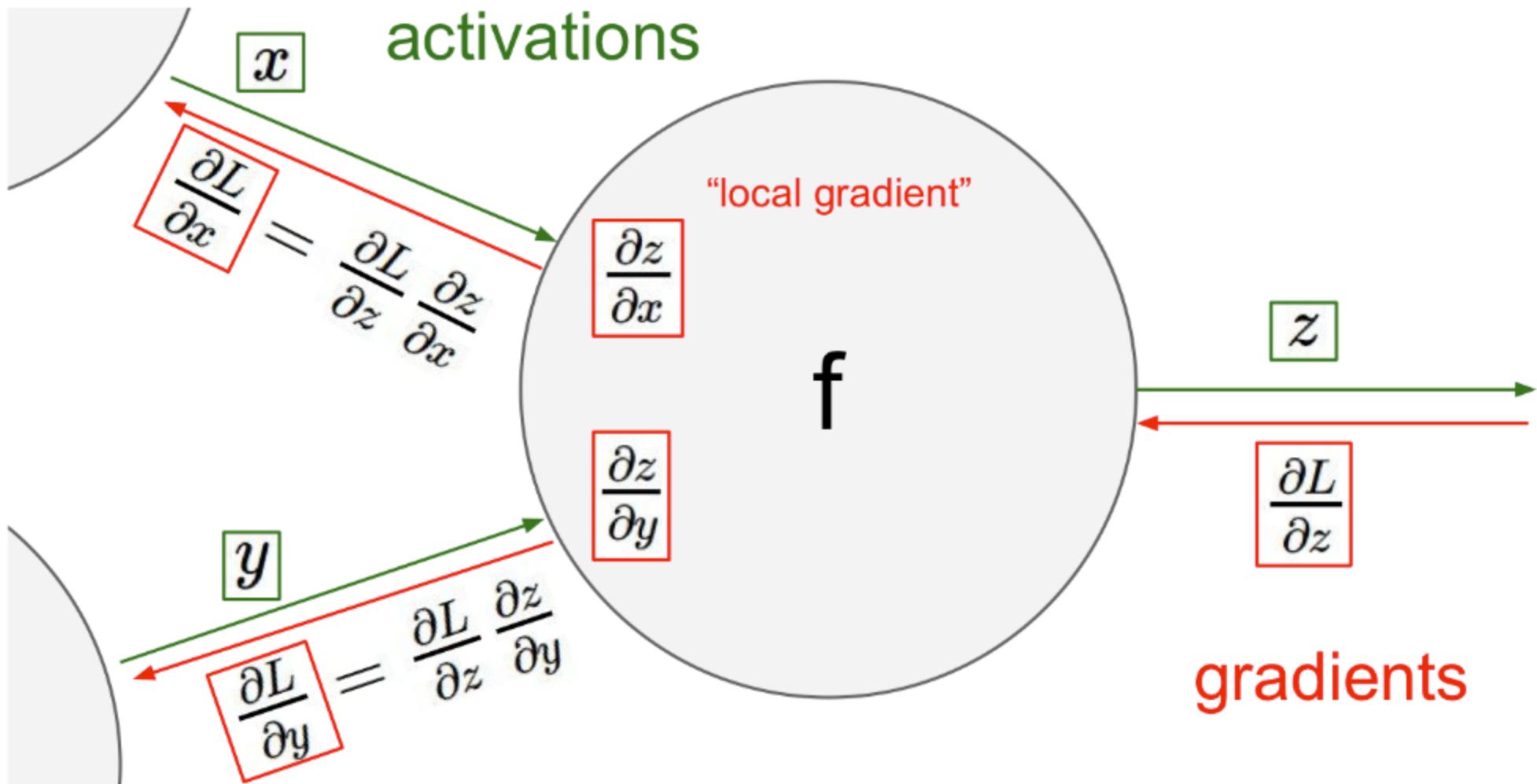
[AT THE NEURON LEVEL]



Credit: A. Karpathy

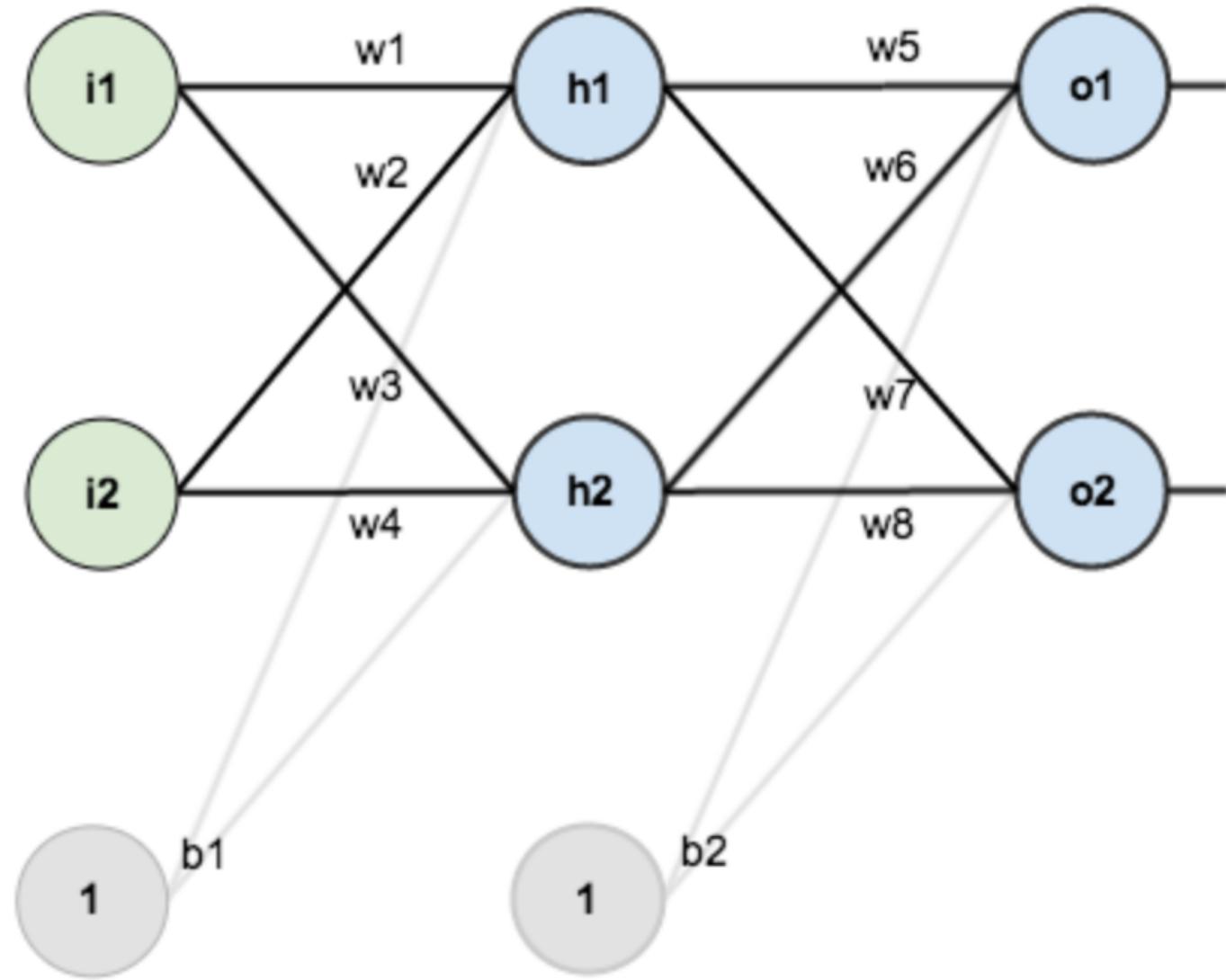
BACKPROPAGATION

[AT THE NEURON LEVEL]

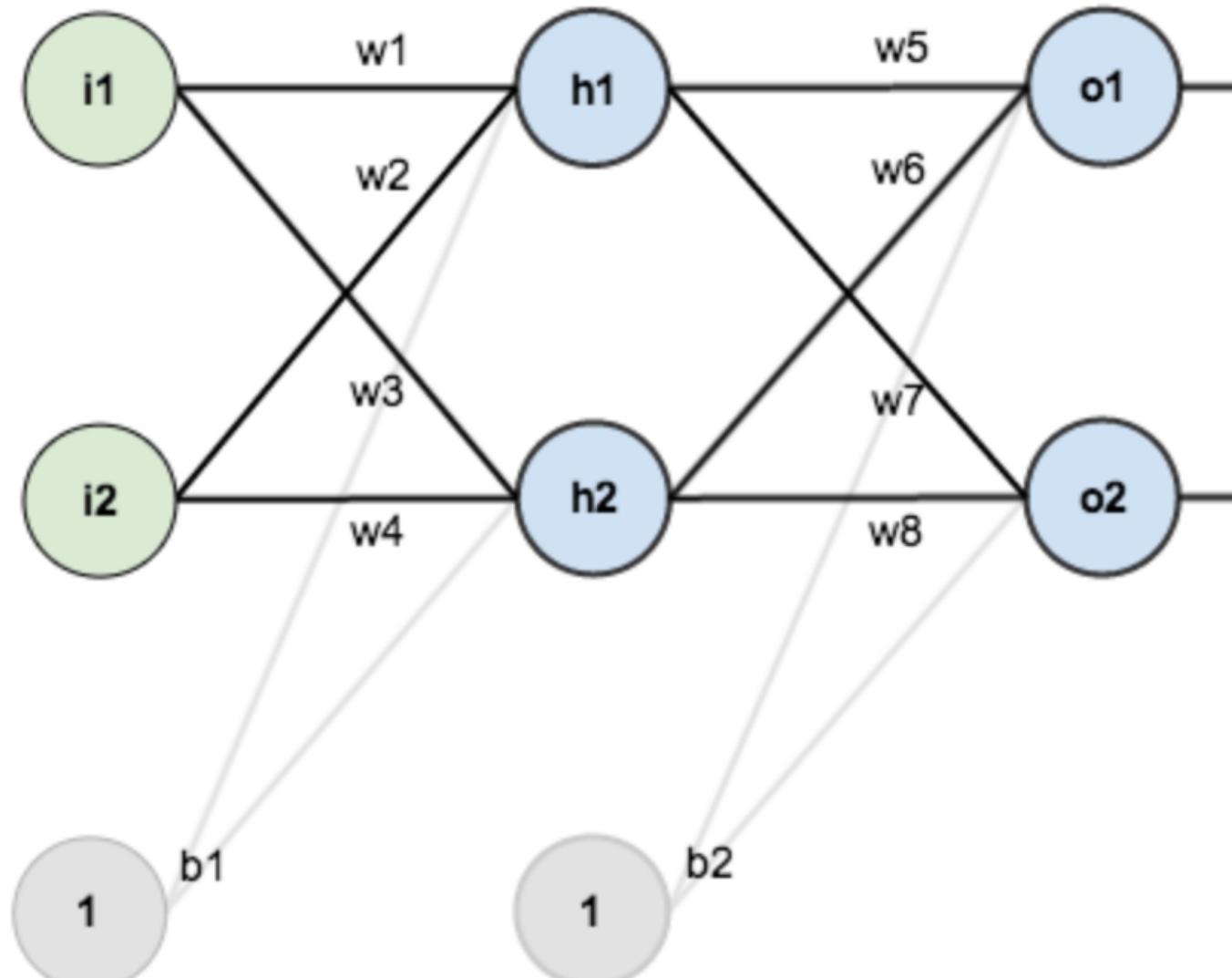


Credit: A. Karpathy

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

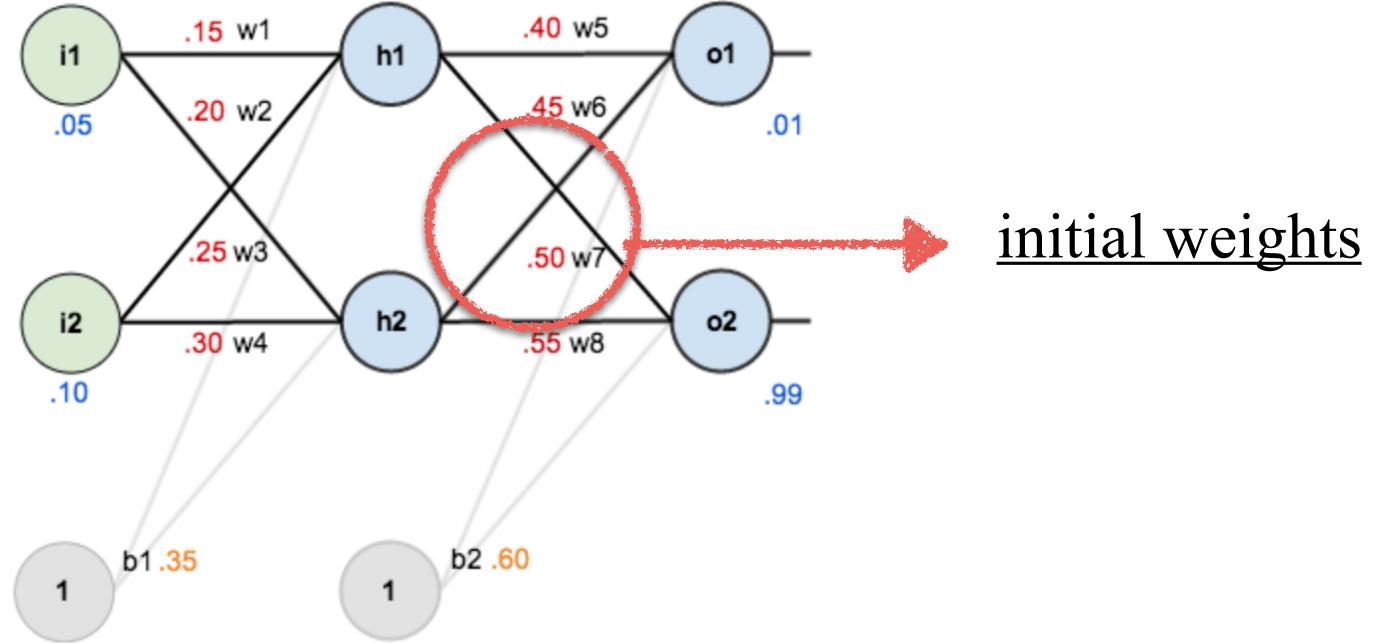


EXAMPLE TAKEN FROM HERE



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

EXAMPLE TAKEN FROM HERE

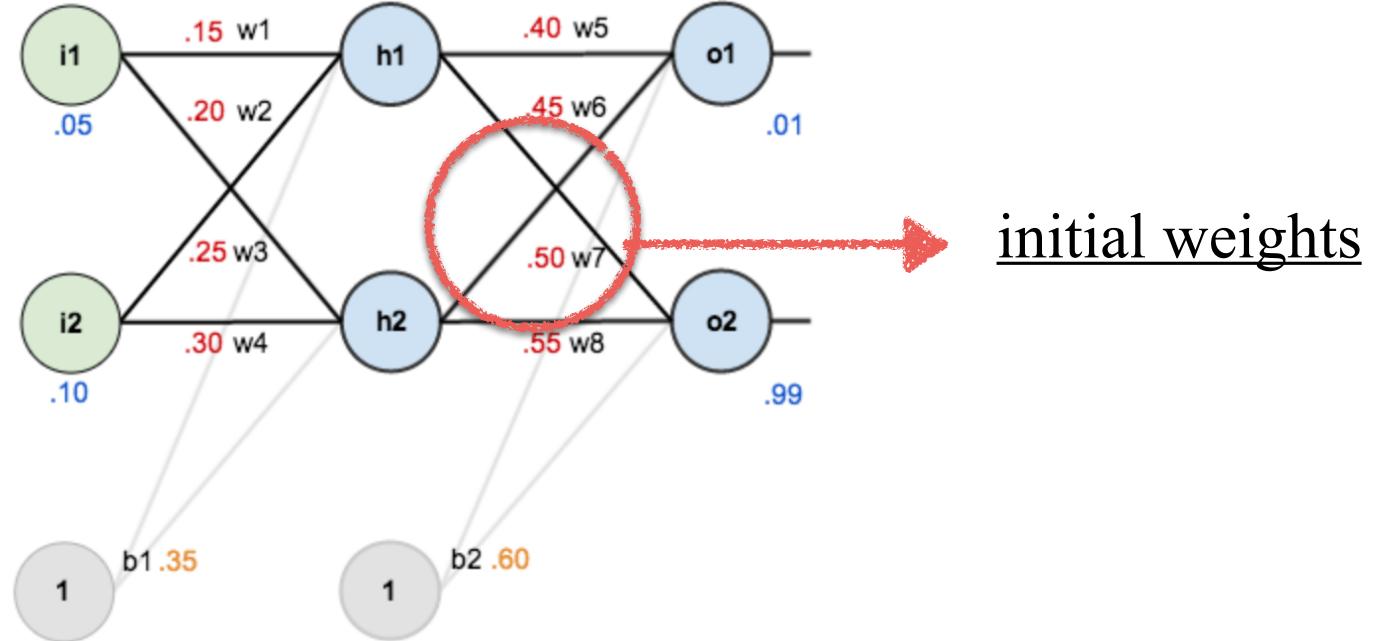


1. THE FORWARD PASS

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

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

[with initial weights]



1. THE FORWARD PASS

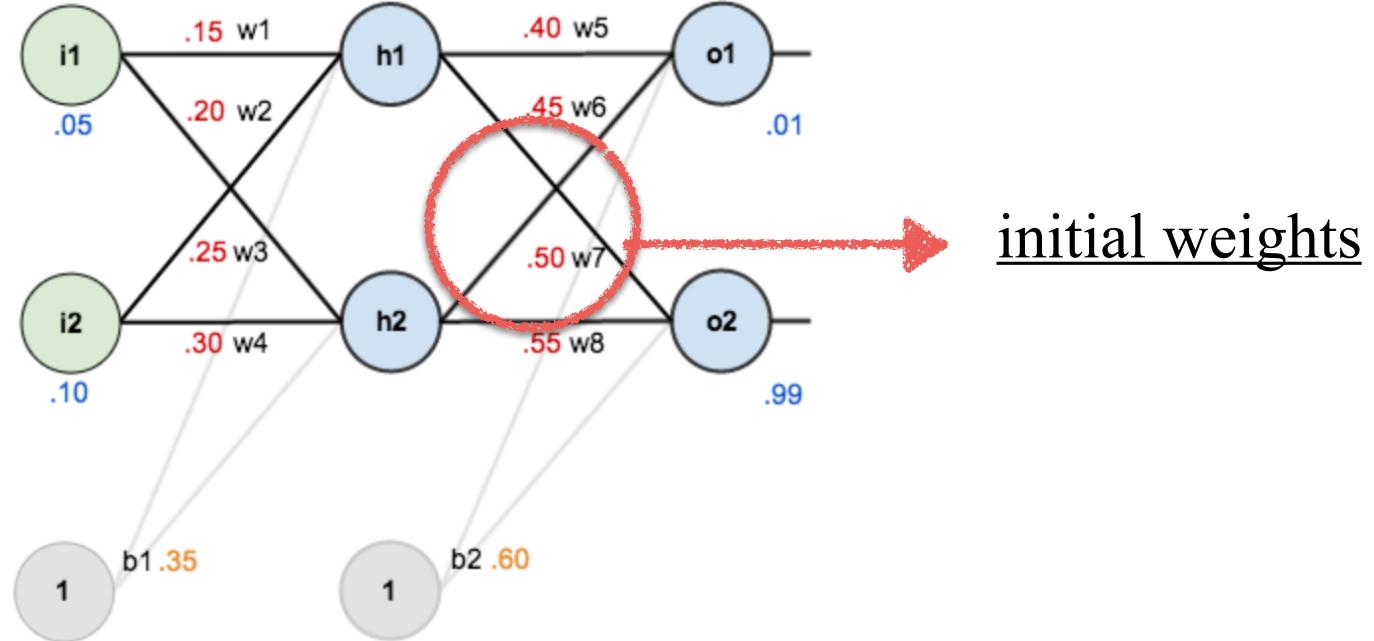
$$in_{h1} = w_1 i_1 + w_2 i_2 + b_1$$

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

[with initial weights]

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

[after the activation function]



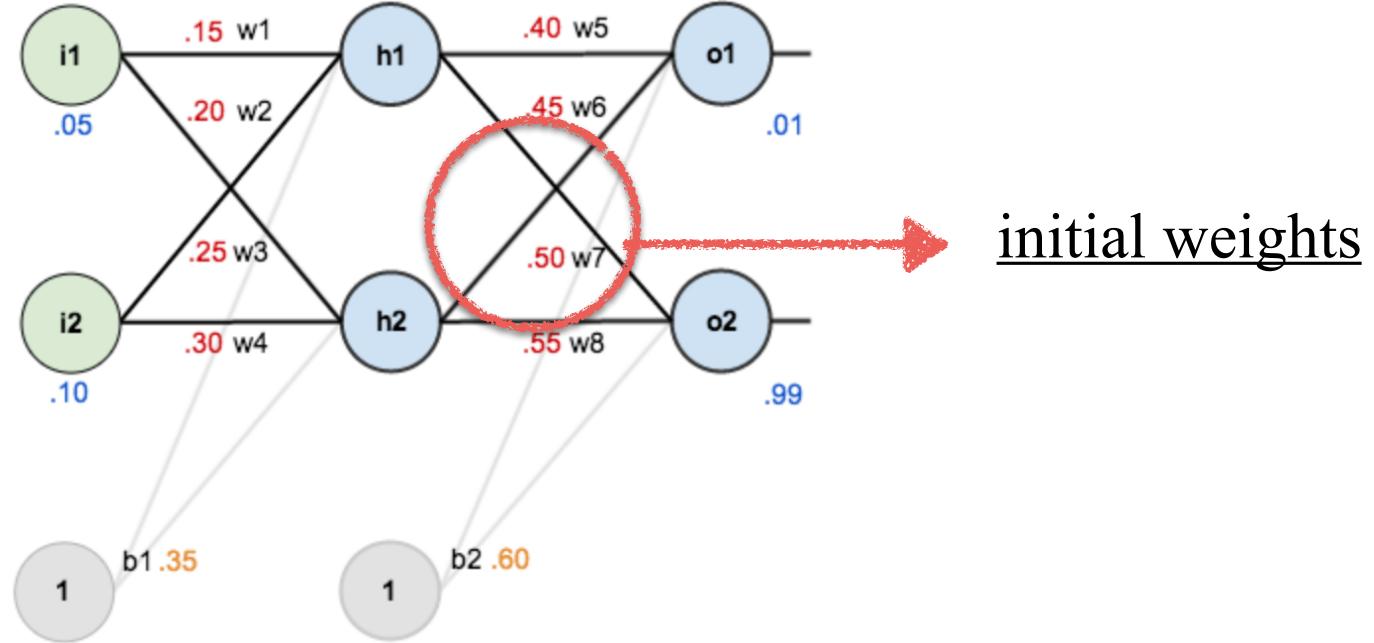
1. THE FORWARD PASS

WE CONTINUE TO o_1

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

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

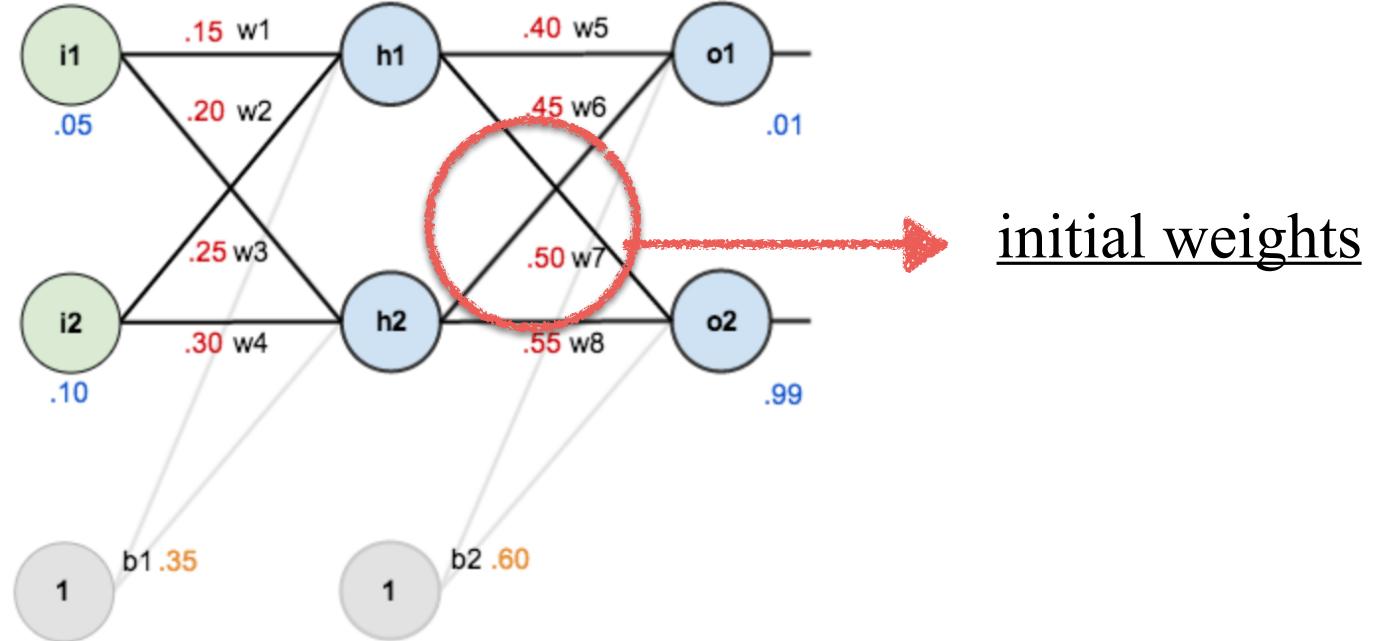
$$out_{o1} = \frac{1}{1 + e^{-1.105}} = 0.751$$



1. THE FORWARD PASS

AND THE SAME FOR o_2

$$out_{o2} = 0.7729$$

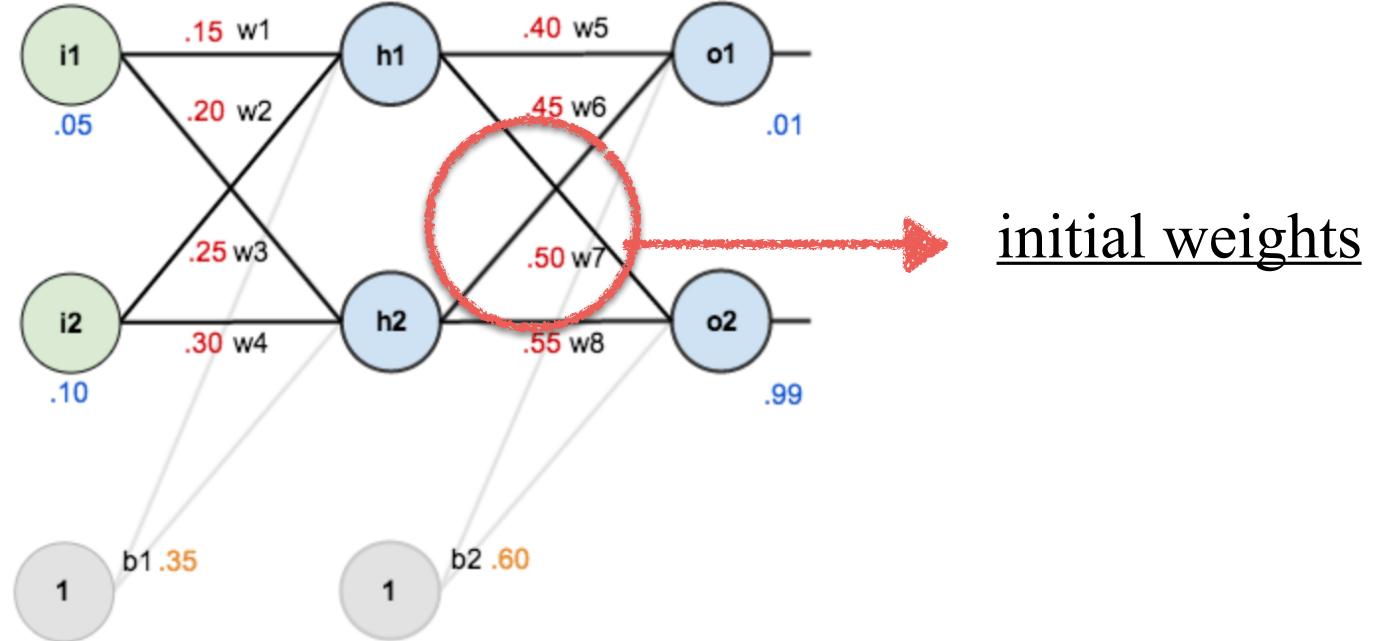


2. THE LOSS FUNCTION

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

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

$$L_{o2} = 0.023$$



2. THE LOSS FUNCTION

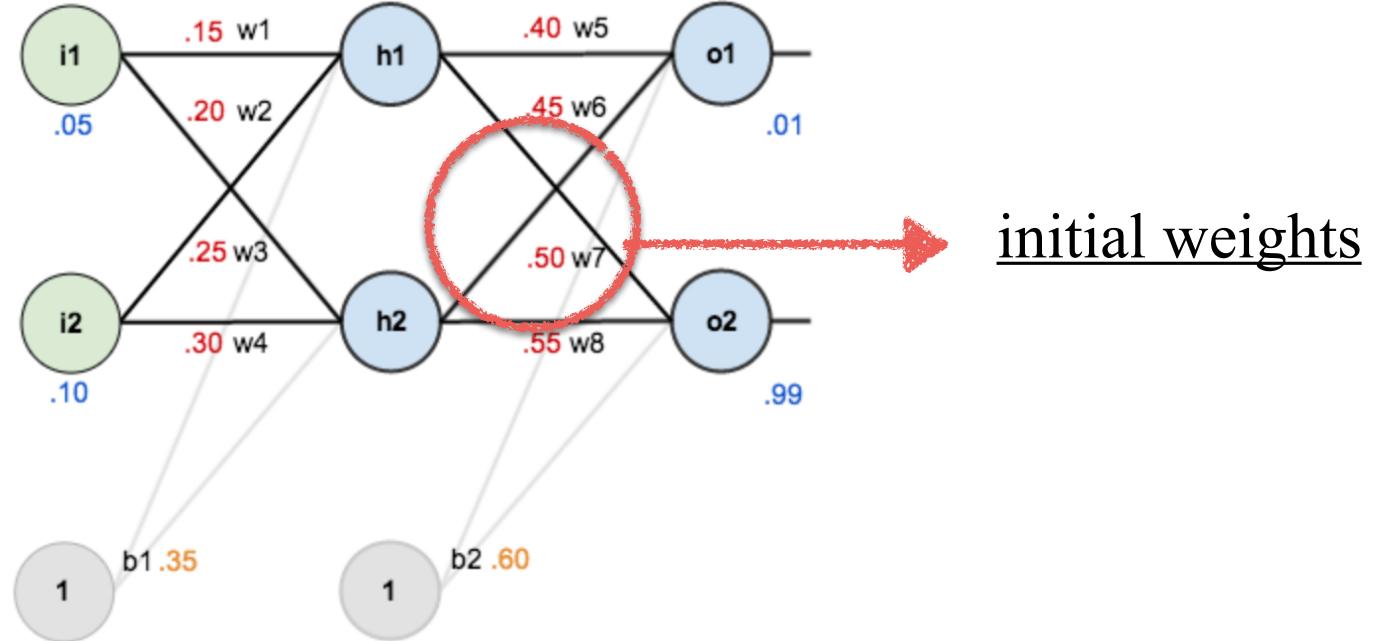
$$L_{total} = \sum 0.5(target - output)^2$$

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

$$L_{o2} = 0.023$$



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



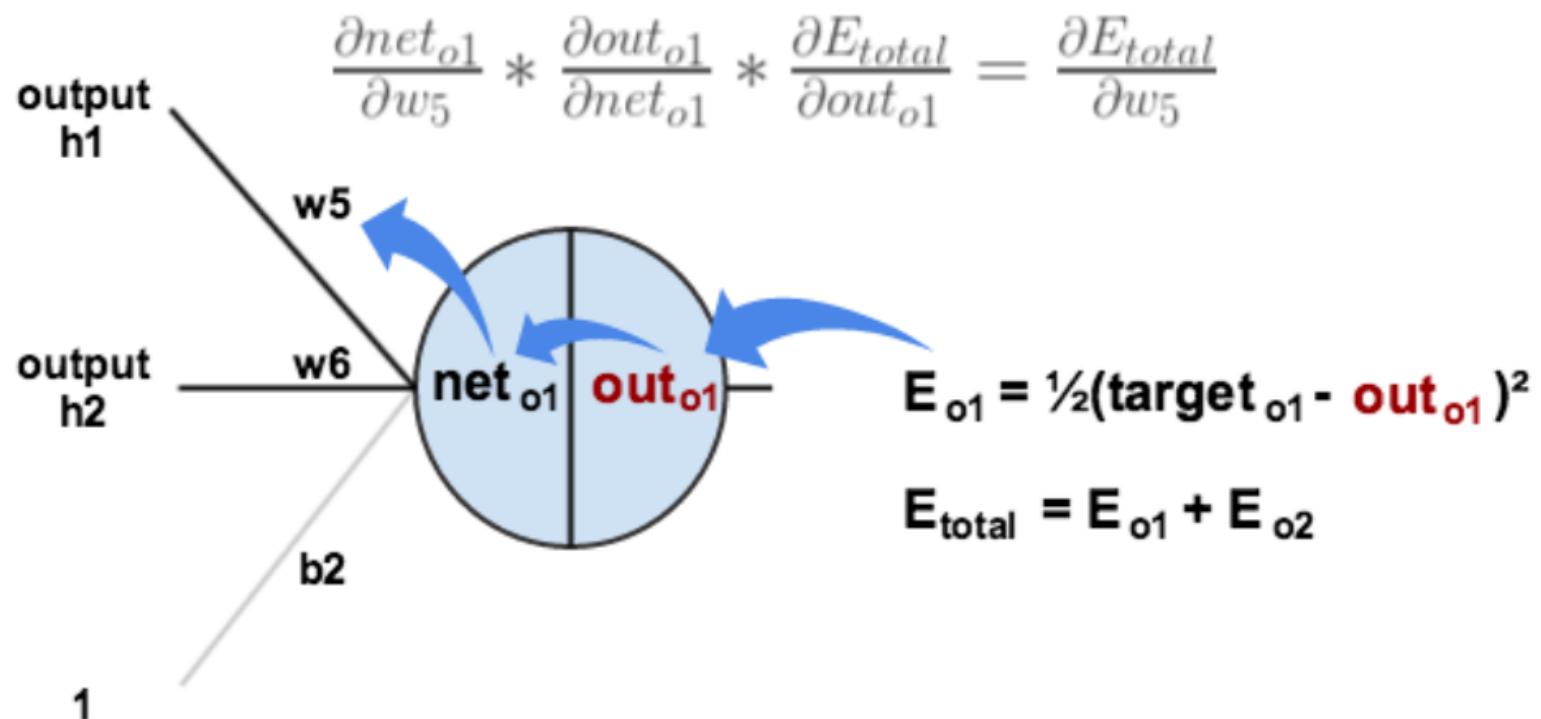
3. THE BACKWARD PASS

FOR w_5

WE WANT:

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

[gradient of loss function]



3. THE BACKWARD PASS

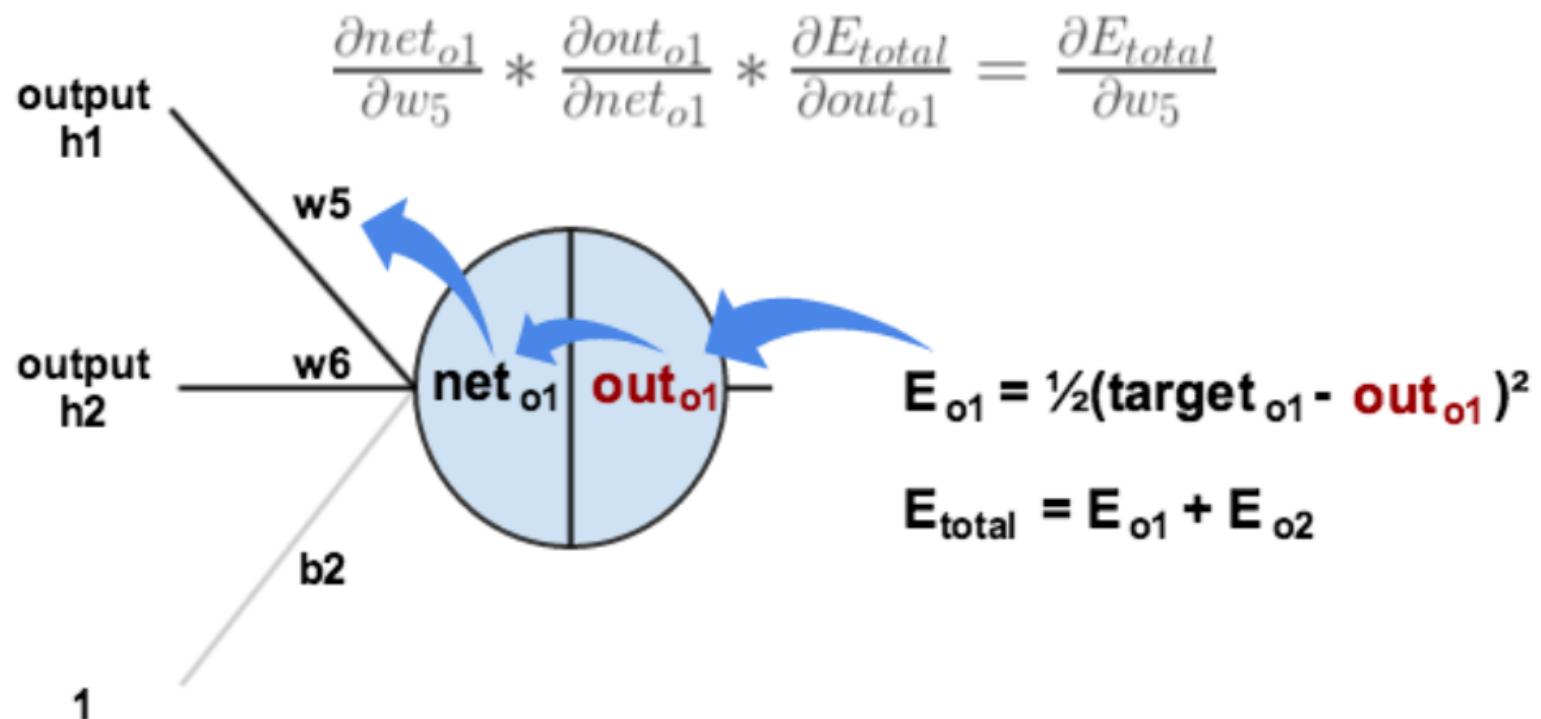
FOR W5

WE WANT:

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

WE APPLY THE CHAIN RULE:

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

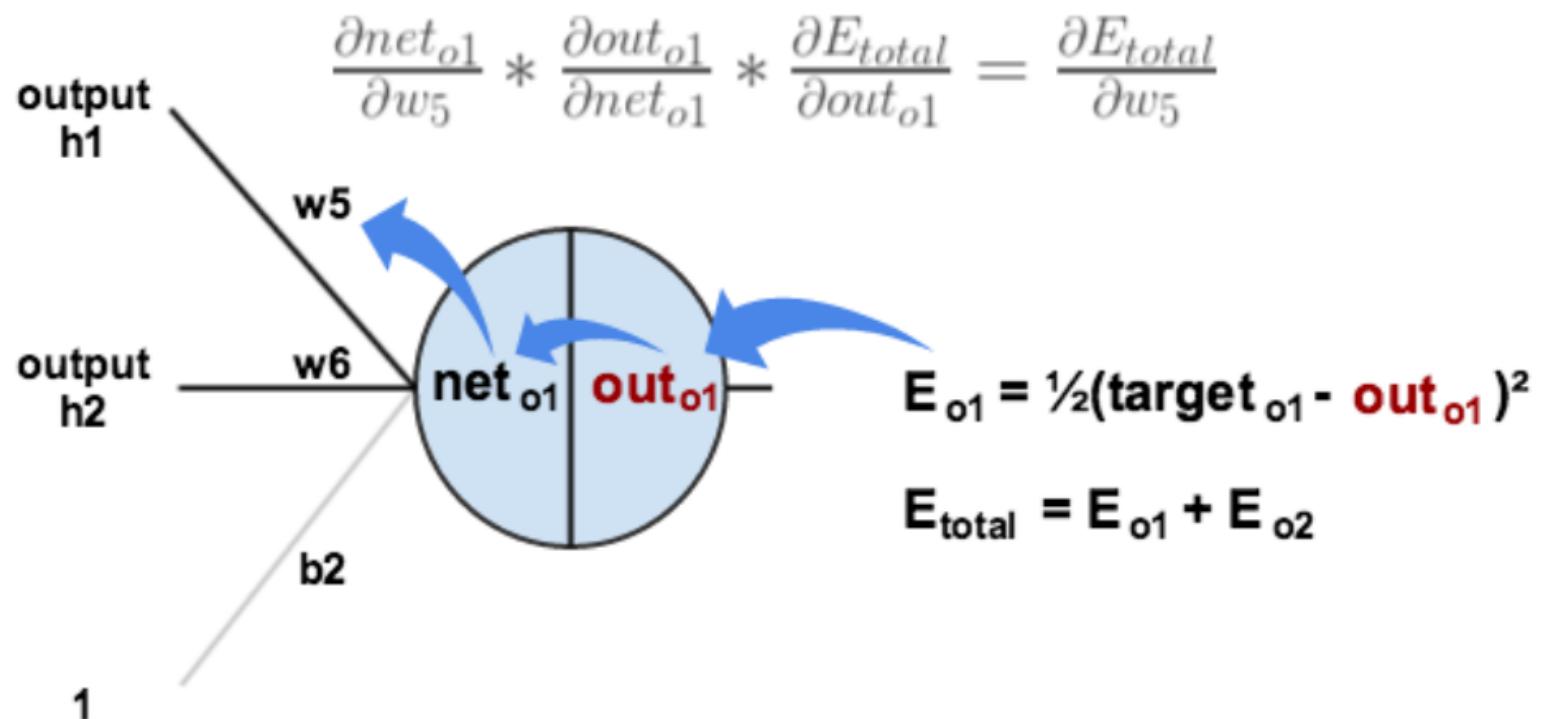


3. THE BACKWARD PASS

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

$$L_{total} = 0.5(\text{target}_{o1} - out_{o1})^2 + 0.5(\text{target}_{o2} - out_{o2})^2$$

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

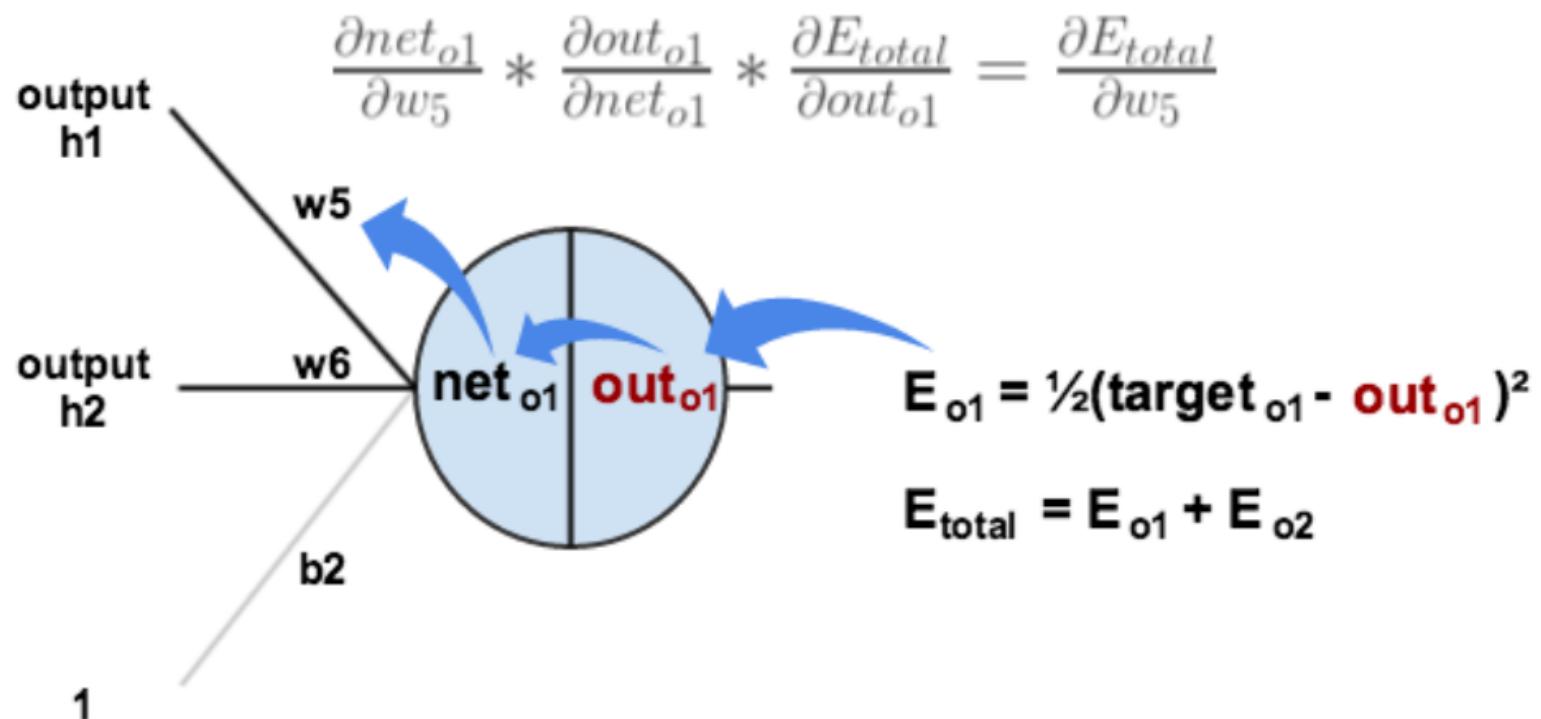


3. THE BACKWARD PASS

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

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

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

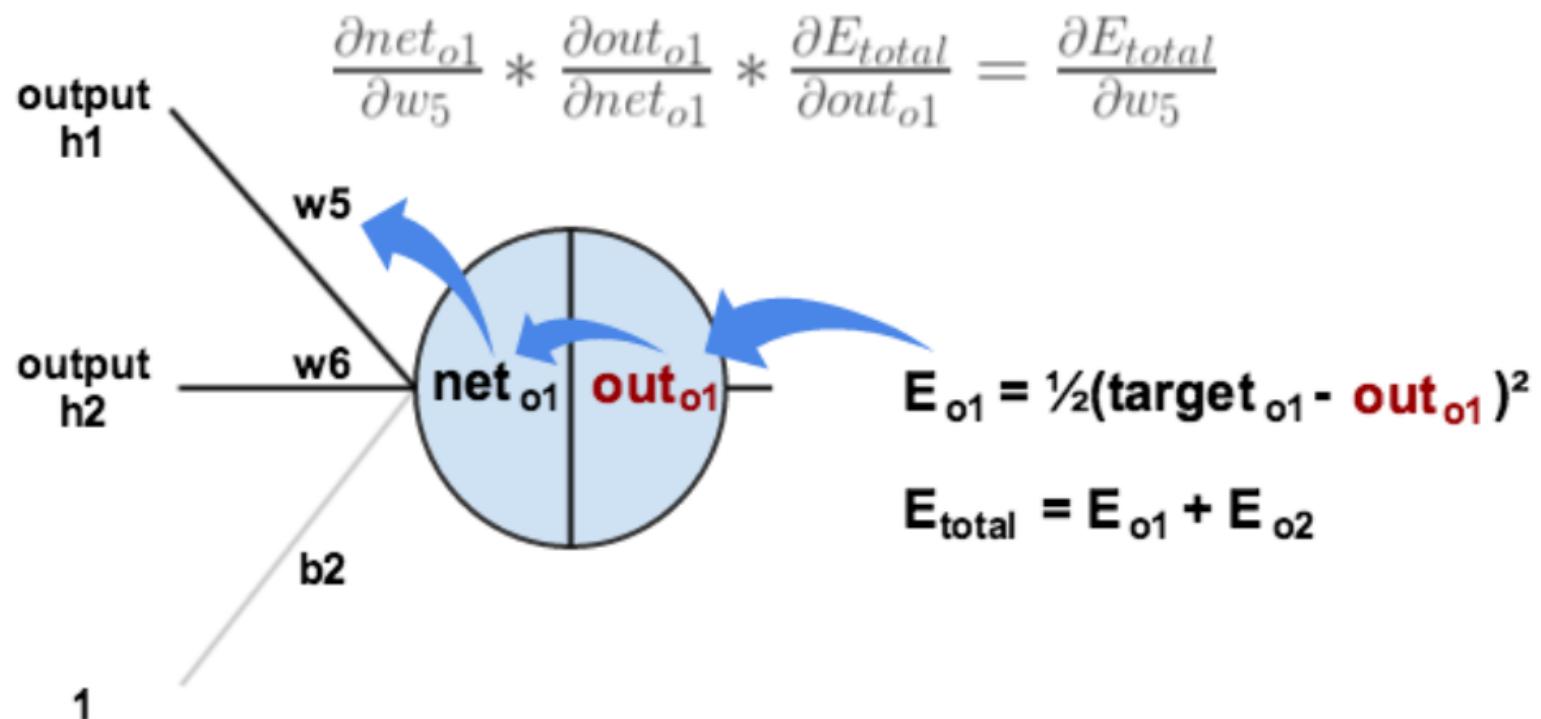


3. THE BACKWARD PASS

$$\frac{\partial L_{total}}{\partial w5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w5}$$

$$in_{o1} = w5 \times out_{h1} + w6 \times out_{h2} + b2$$

$$\frac{\partial in_{o1}}{\partial w5} = out_{h1} \times w_5^{1-1} = out_{h1} = 0.593$$

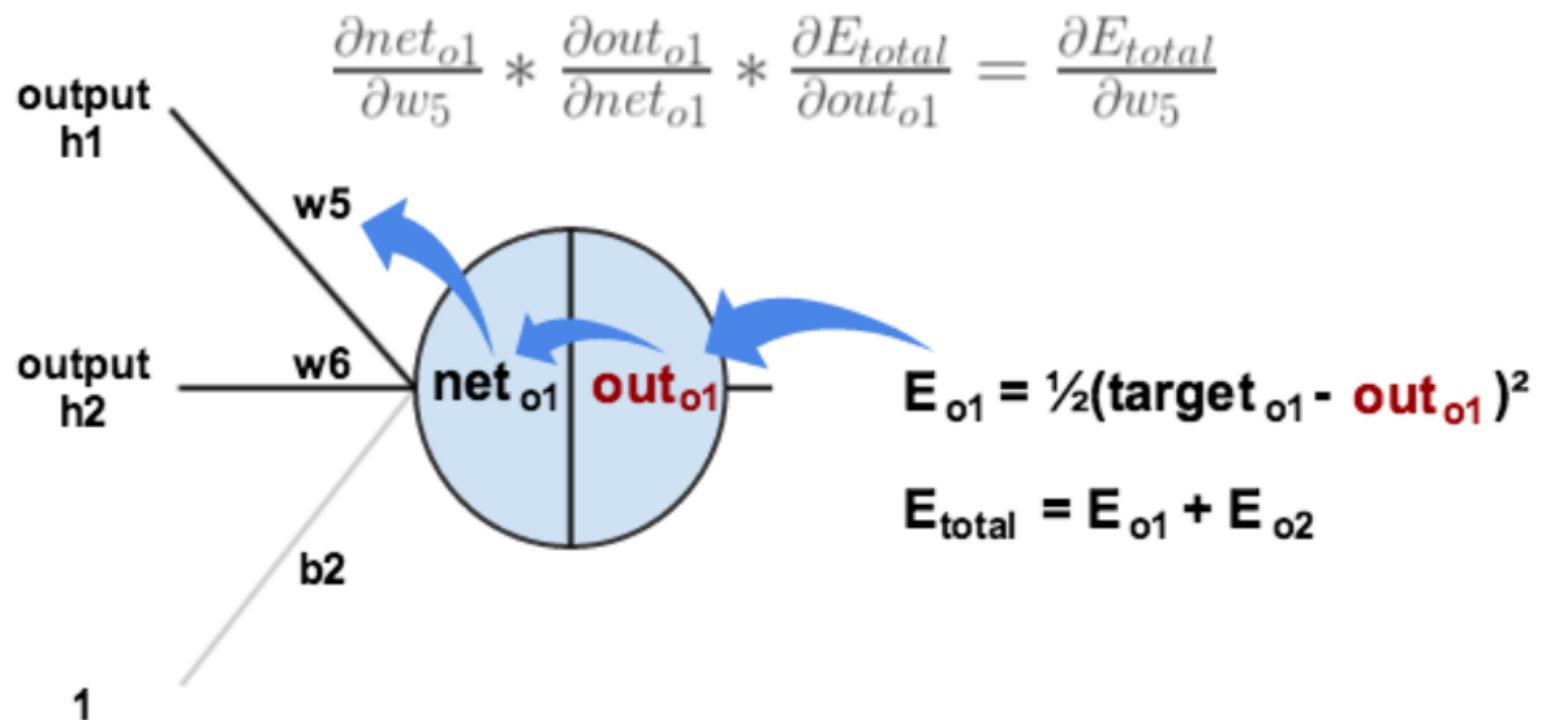


3. THE BACKWARD PASS

ALL TOGETHER:

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

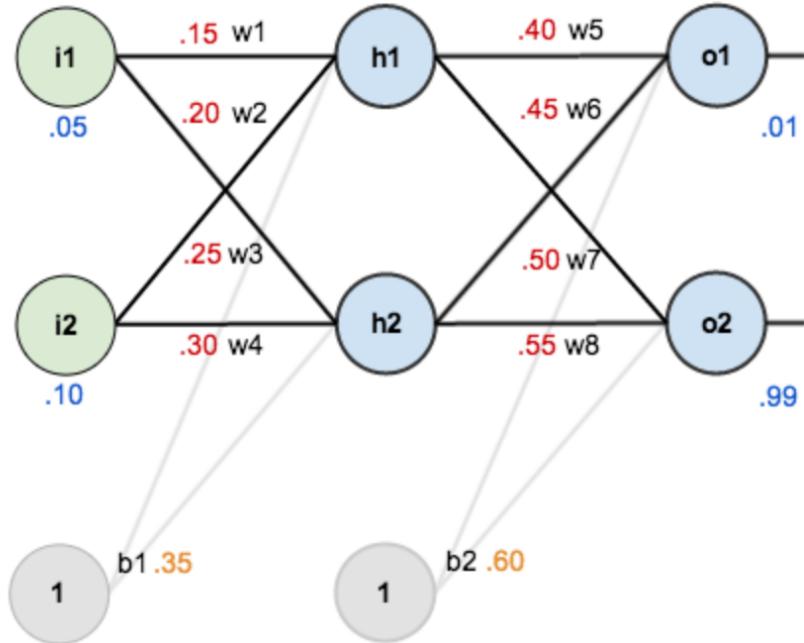
$$\frac{\partial L_{total}}{\partial w_5} = 0.741 \times 0.186 \times 0.593 = 0.082$$



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**

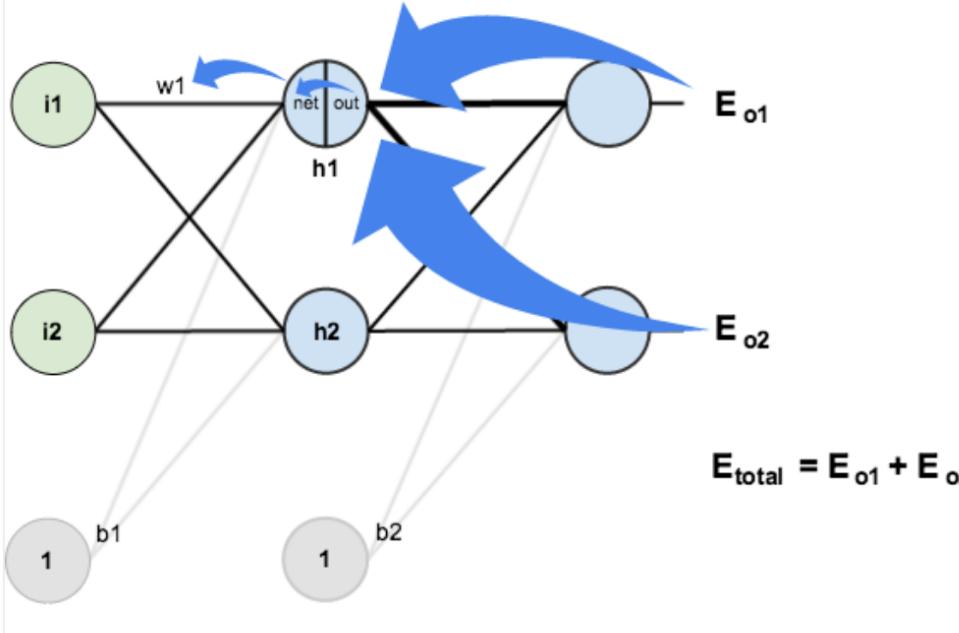
$$w_6^{t+1} = 0.408$$

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

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

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

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



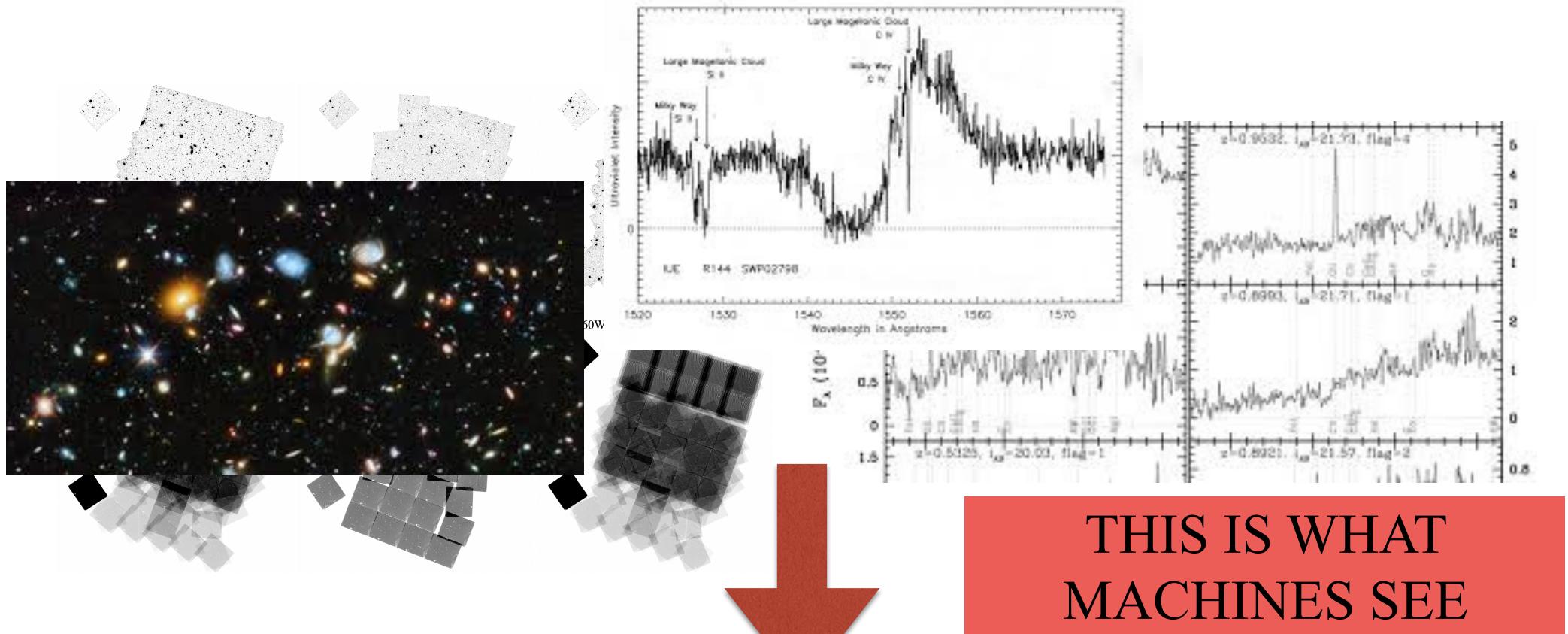
AND BACK-PROPAGATED TO THE HIDDEN LAYERS

CAN WE GO DEEP NOW?

CAN WE GO DEEP NOW?

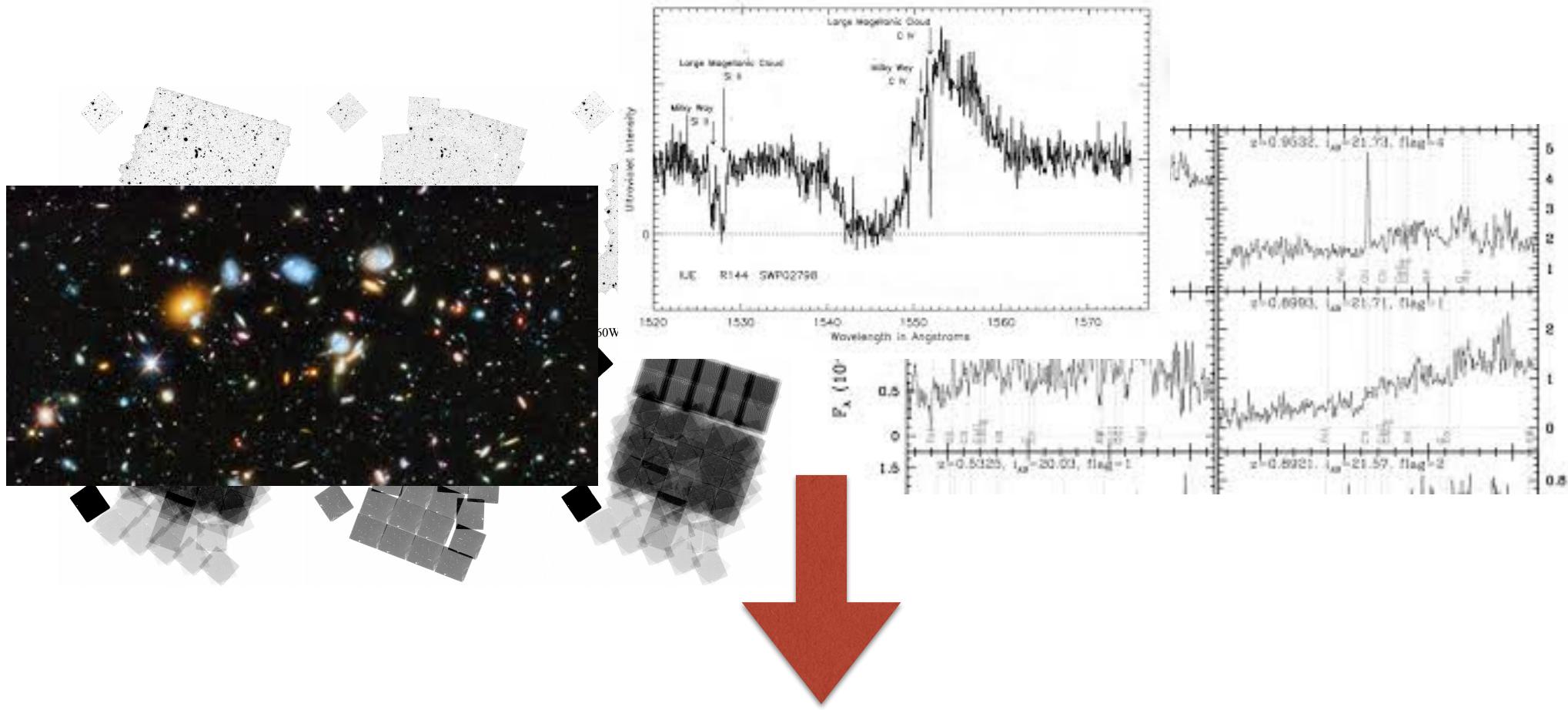
ALMOST THERE...LET'S THINK FOR A
MOMENT ABOUT WHAT WE PUT AS
INPUT...

What do we put as input?



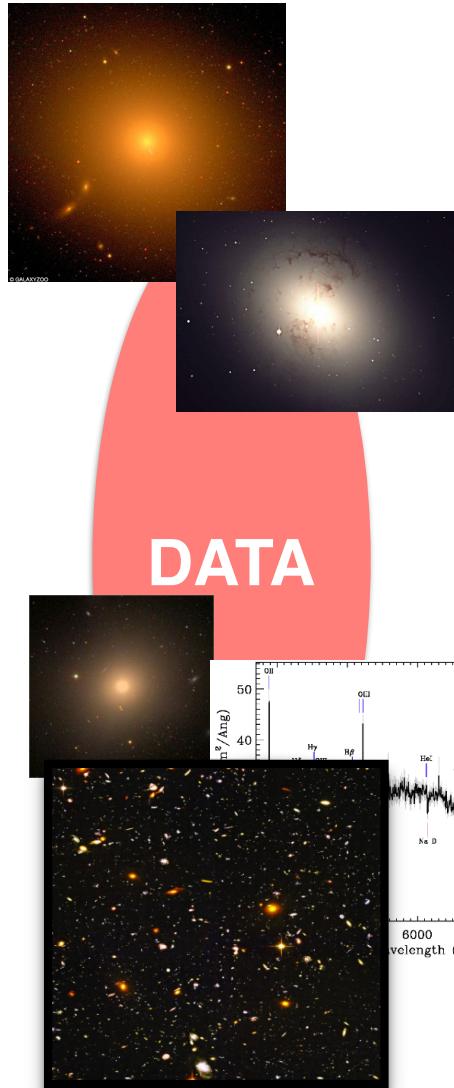
THIS IS WHAT MACHINES SEE

What do we put as input?



PRE-PROCESS DATA TO EXTRACT MEANINGFUL INFORMATION

THIS IS GENERALLY CALLED **FEATURE EXTRACTION**



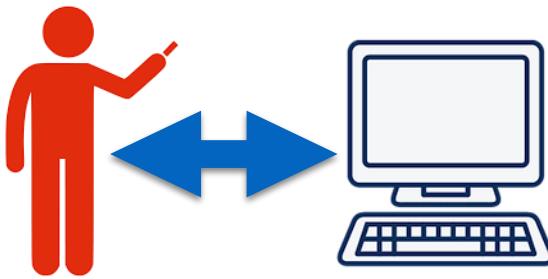
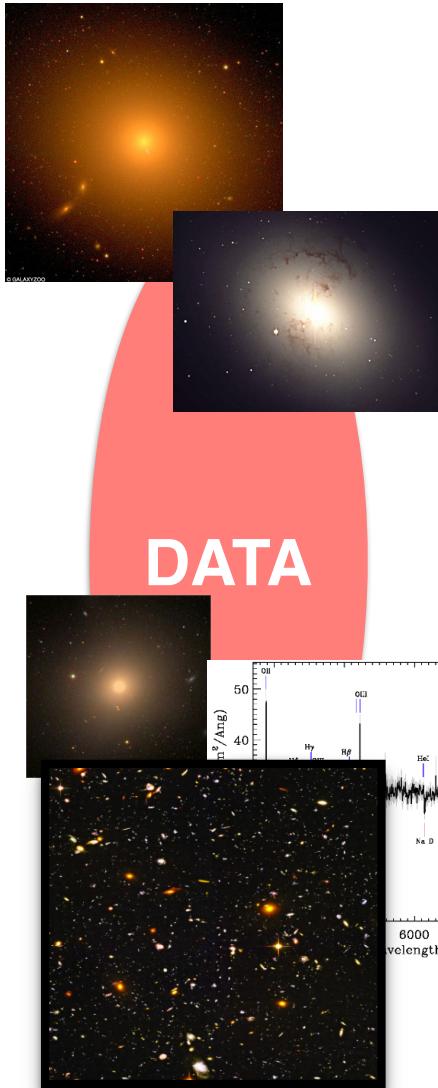
Spiral!

Emission line!

Merger!

Clump!

AGN!



Spiral!

Emission line!

Merger!

Clump!

AGN!

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

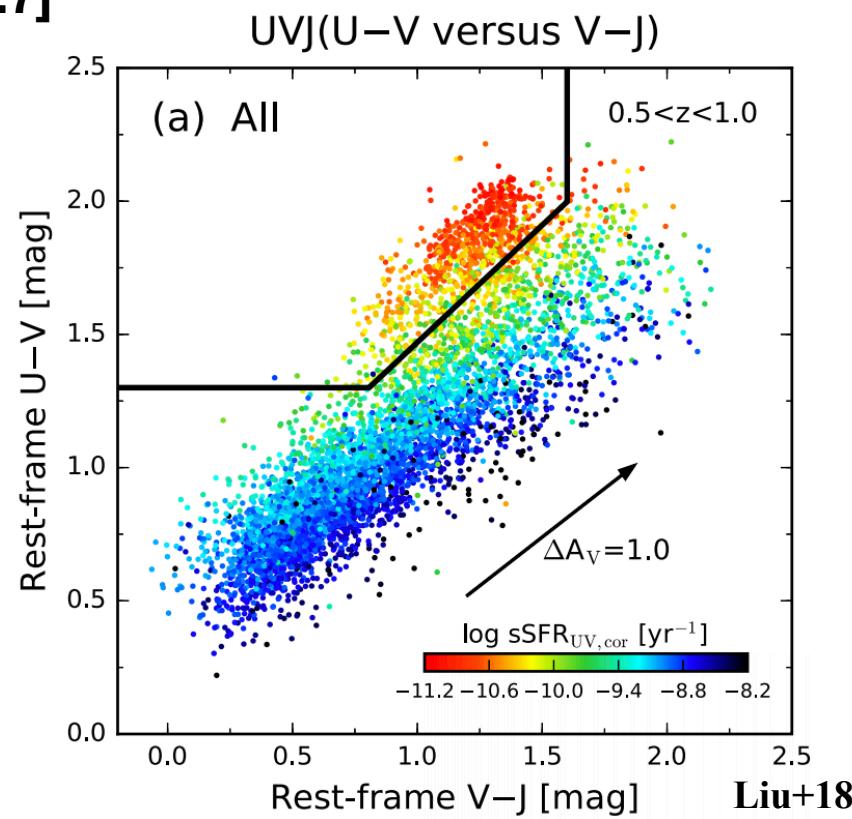
Q(0) , SF(1)

NETWORK FUNCTION

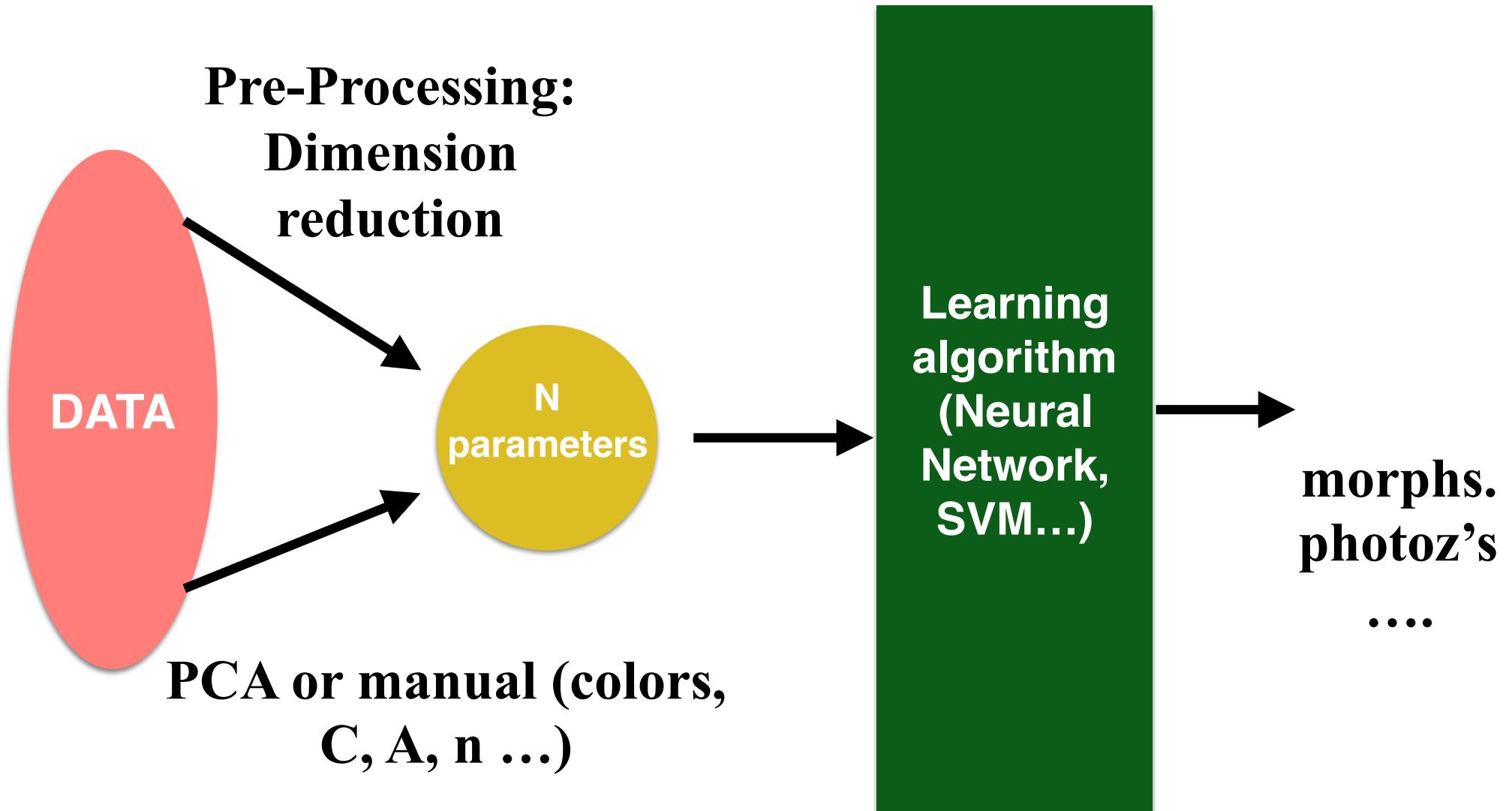
(U-V, V-J) FEATURES

$$\text{sgn}[(u-v)-0.8*(v-j)-0.7]$$

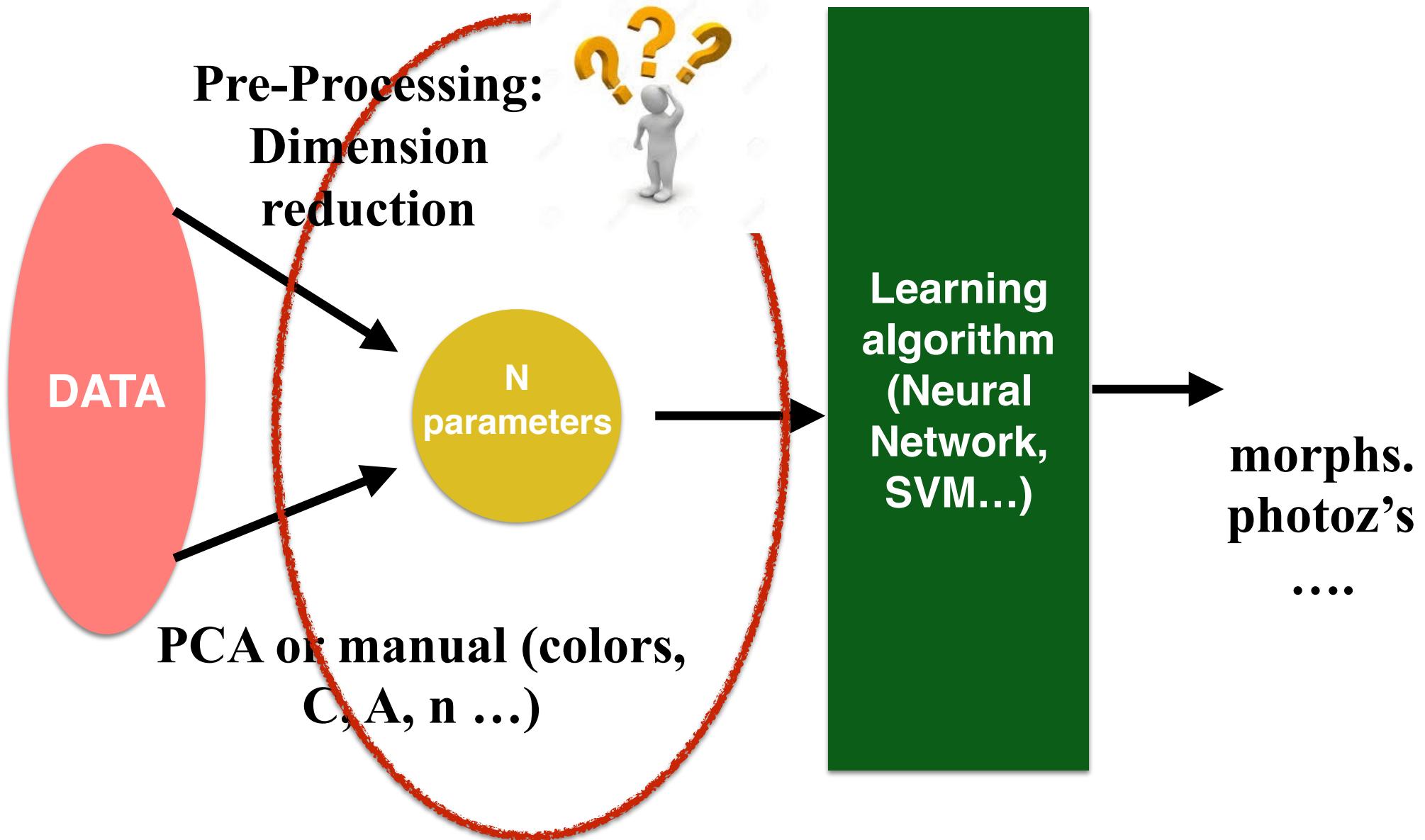
WEIGHTS

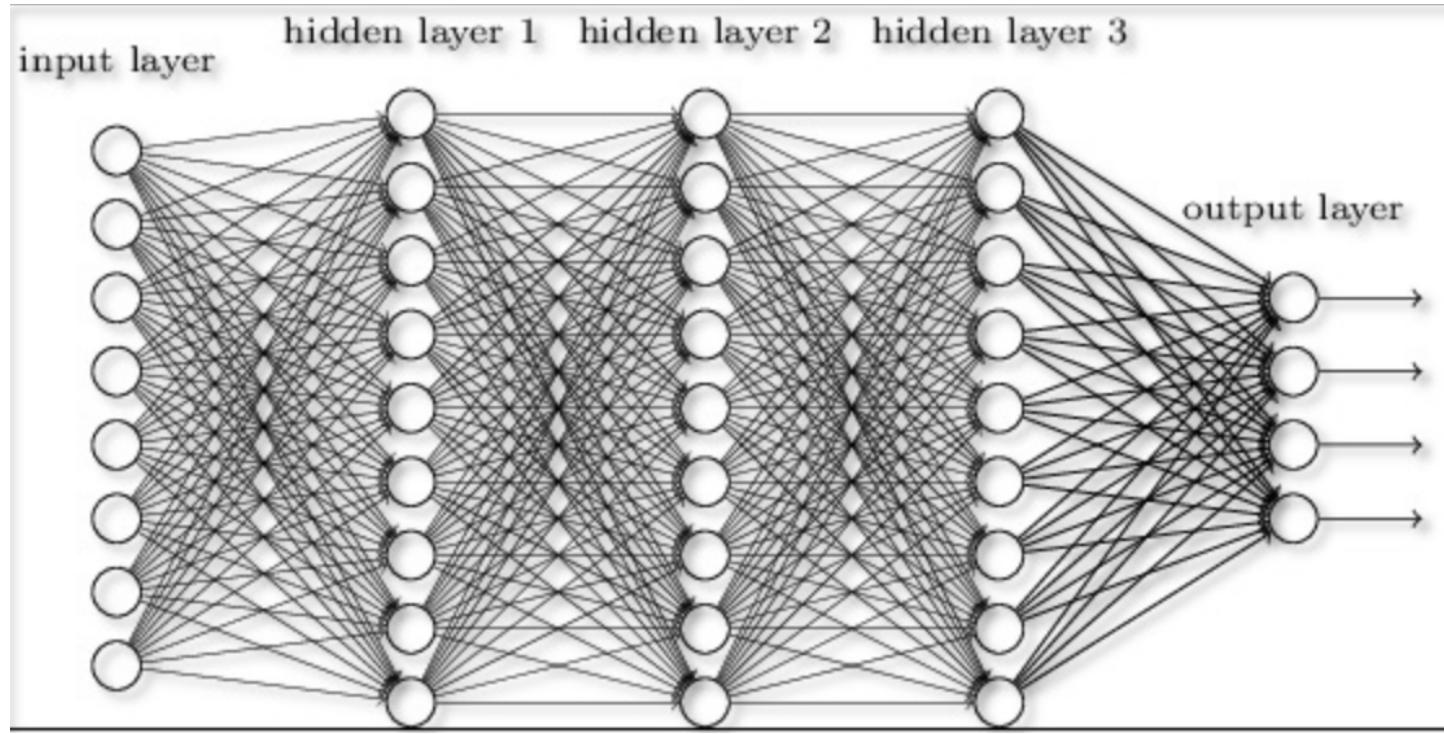


THE “CLASSICAL” APPROACH

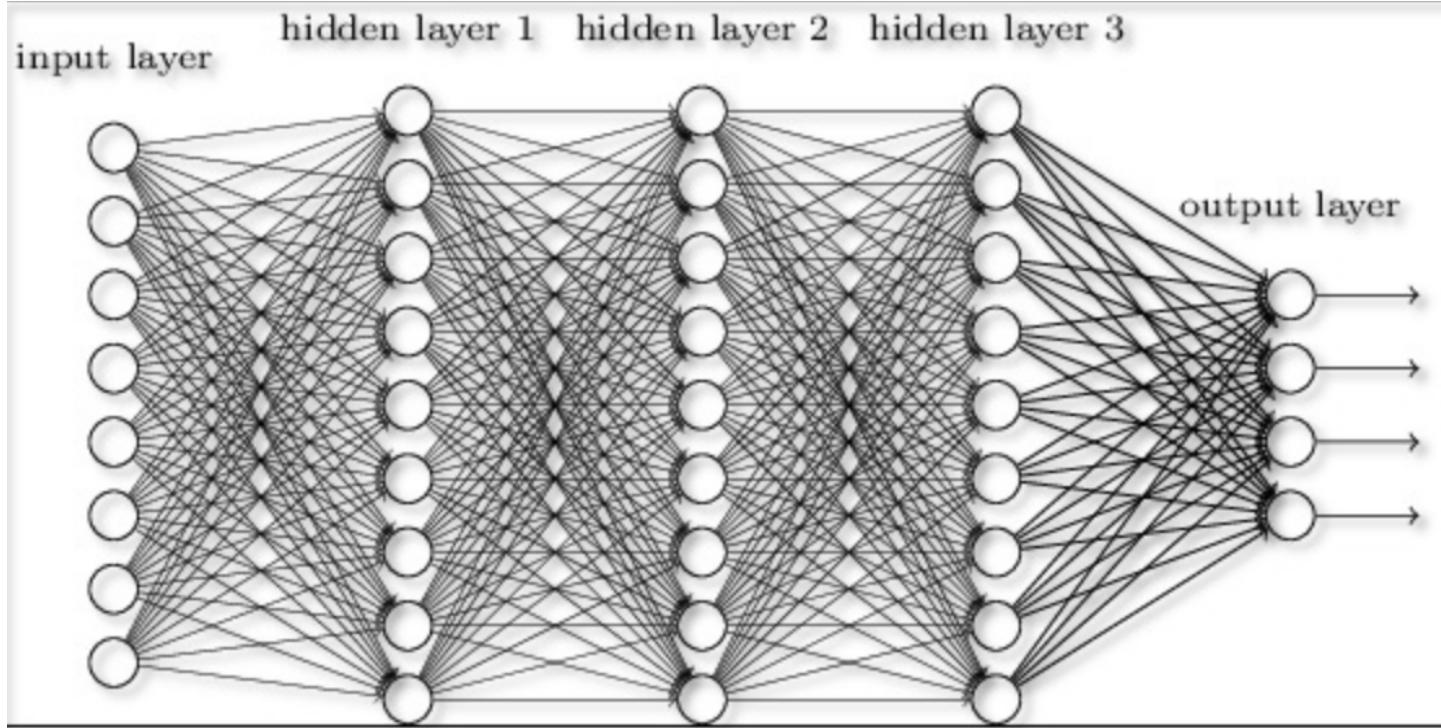


“CLASSICAL” MACHINE LEARNING



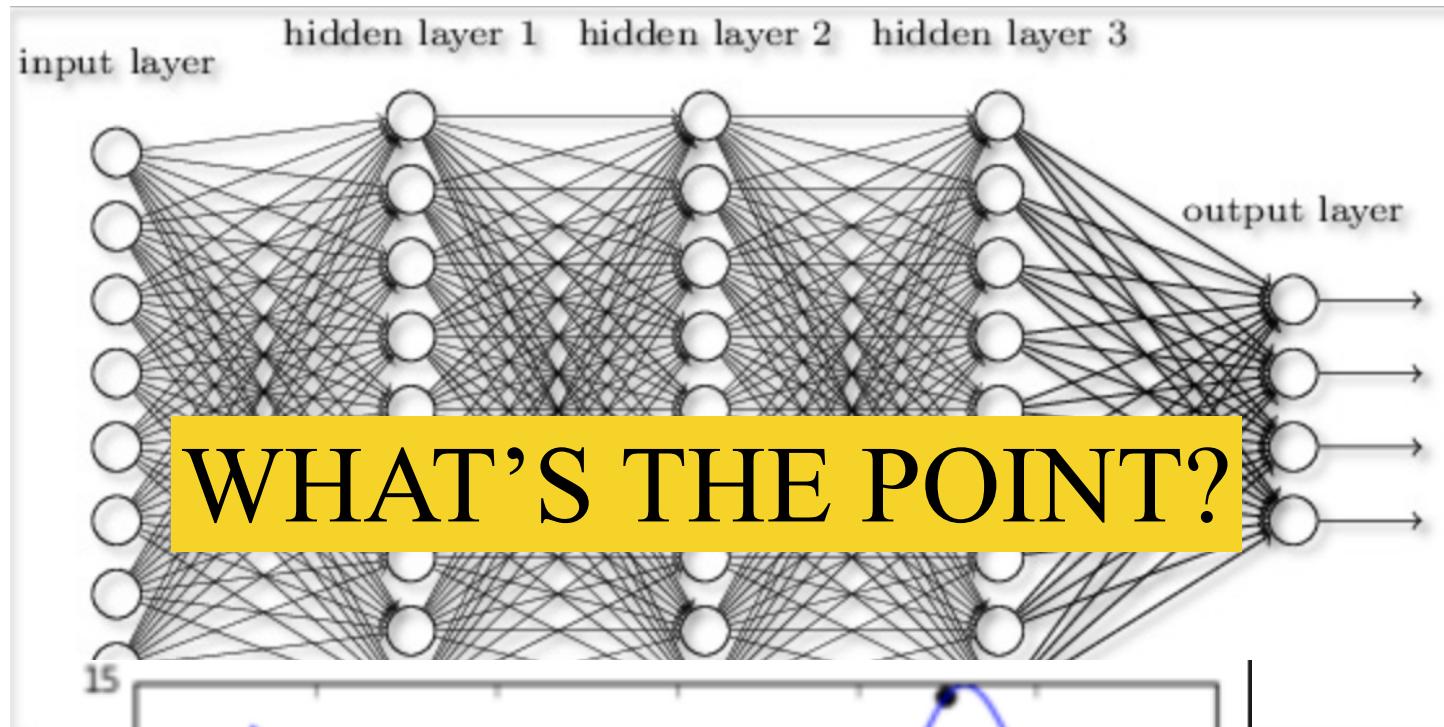


DEEPER → INCREASING NUMBER OF PARAMETERS



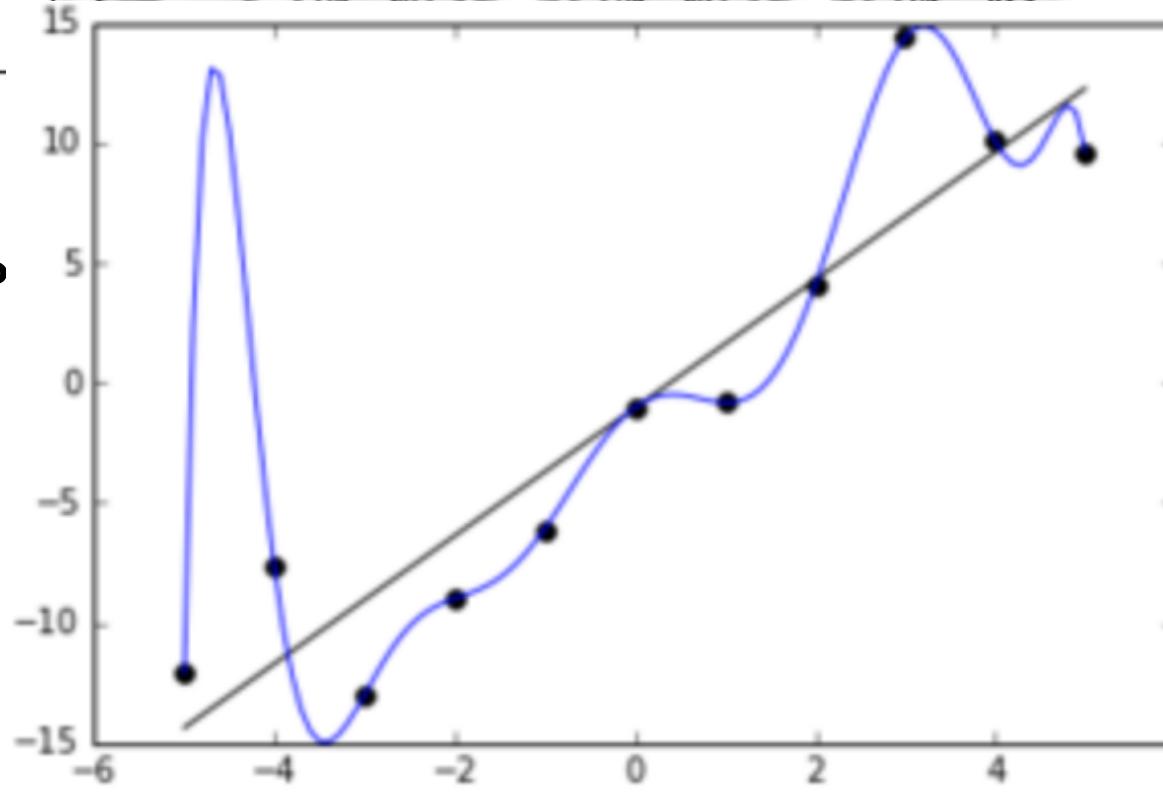
DEEPER → INCREASING NUMBER OF PARAMETERS

IF OUR INPUT CONTAINS A HANDFUL
OF PARAMETERS [colors, sizes, lines etc..]



DEEP

ETERS



What about using raw data?

ALL INFORMATION IS IN THE INPUT DATA

WHY REDUCING ?

LET THE NETWORK FIND THE INFO

What about using raw data?

ALL INFORMATION IS IN THE INPUT DATA

WHY REDUCING ?

LET THE NETWORK FIND THE INFO

LARGE DIMENSION SIGNALS SUCH AS IMAGES OR SPECTRA WOULD REQUIRE TREMENDOUSLY LARGE MODELS

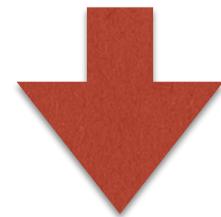
A 512x512 image as input of a fully connected layer producing output of same size:

$$(512 \times 512)^2 = 7e10$$

BUT

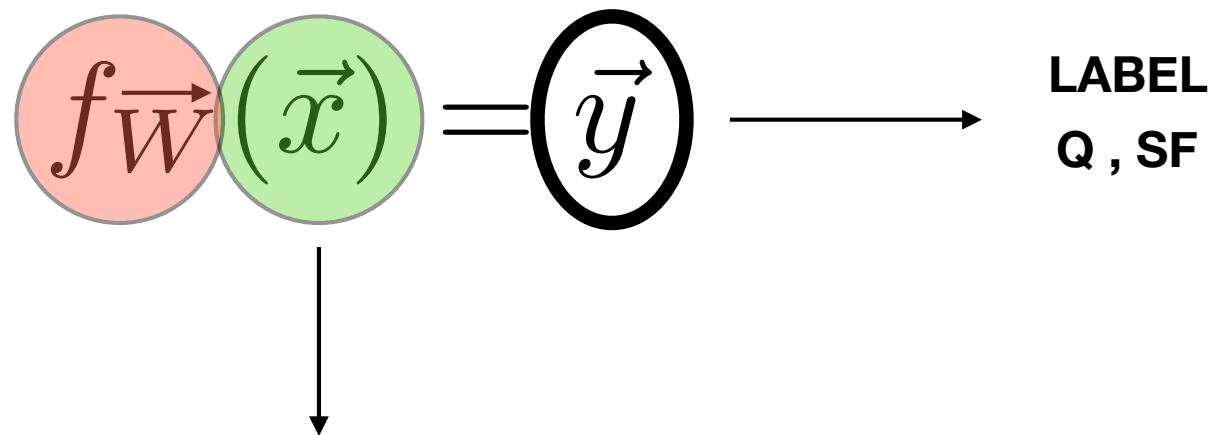
FEEDING INDIVIDUAL RESOLUTION ELEMENTS IS NOT
VERY EFFICIENT SINCE IT LOOSES ALL INVARIANCE TO
TRANSLATION

FEEDING INDIVIDUAL RESOLUTION ELEMENTS IS NOT
VERY EFFICIENT SINCE IT LOOSES ALL INVARIANCE TO
TRANSLATION



SO?

DEEP LEARNING



LET THE MACHINE FIGURE THIS OUT ("unsupervised feature extraction")