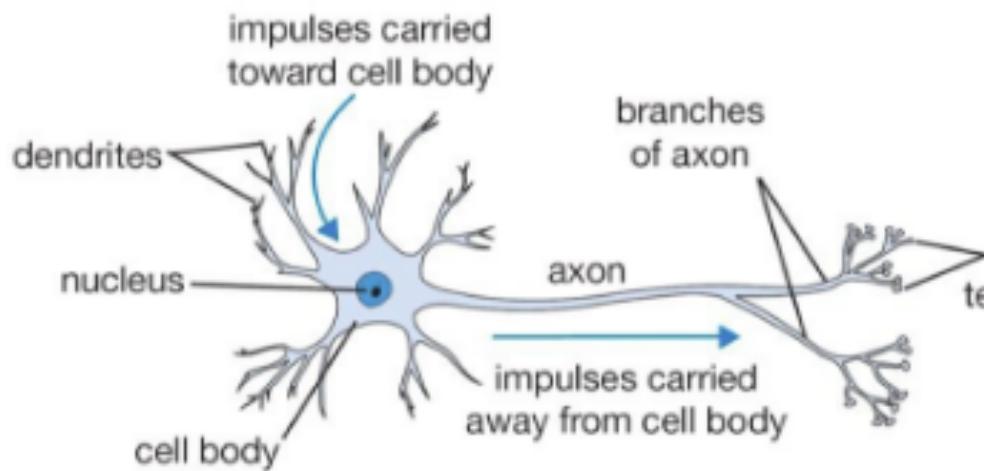


PART II: A FOCUS ON “SHALLOW” NEURAL NETWORKS

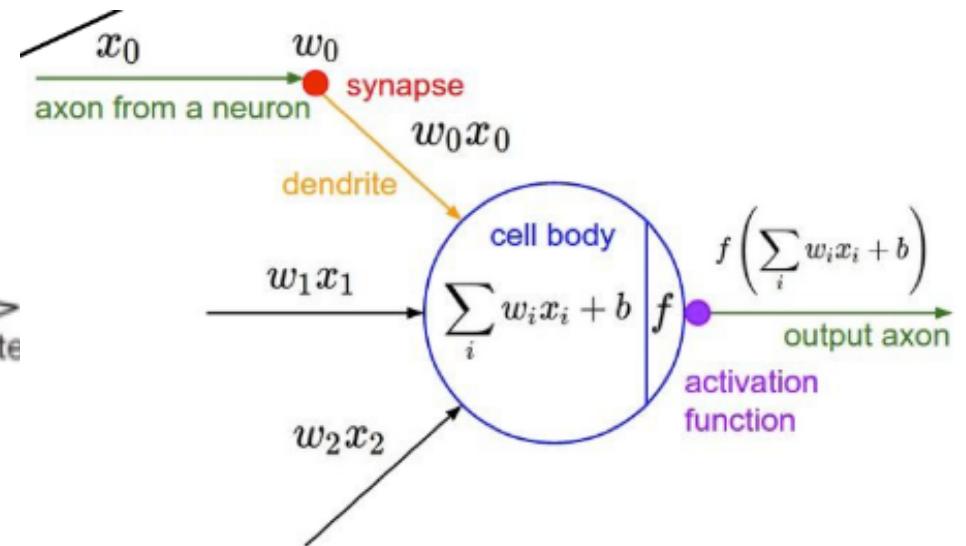
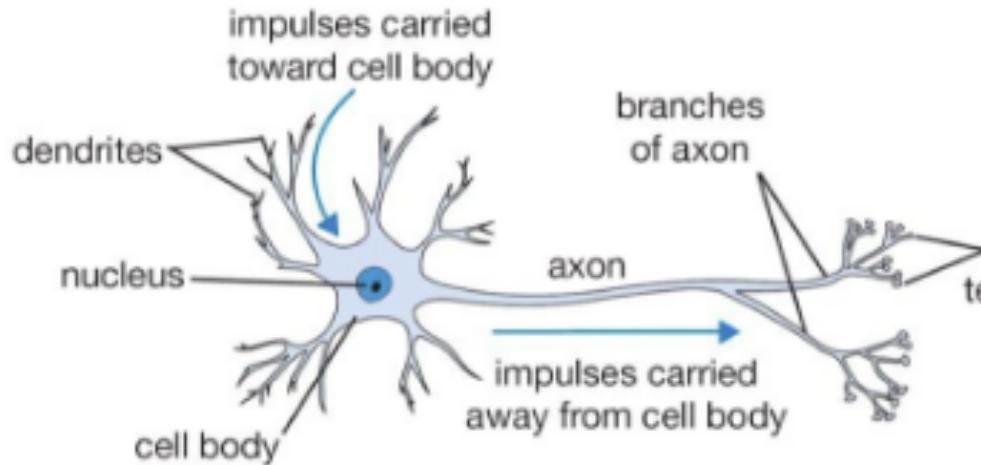
THE NEURON



INSPIRED BY NEURO - SCIENCE?

Credit: Karpathy

THE NEURON



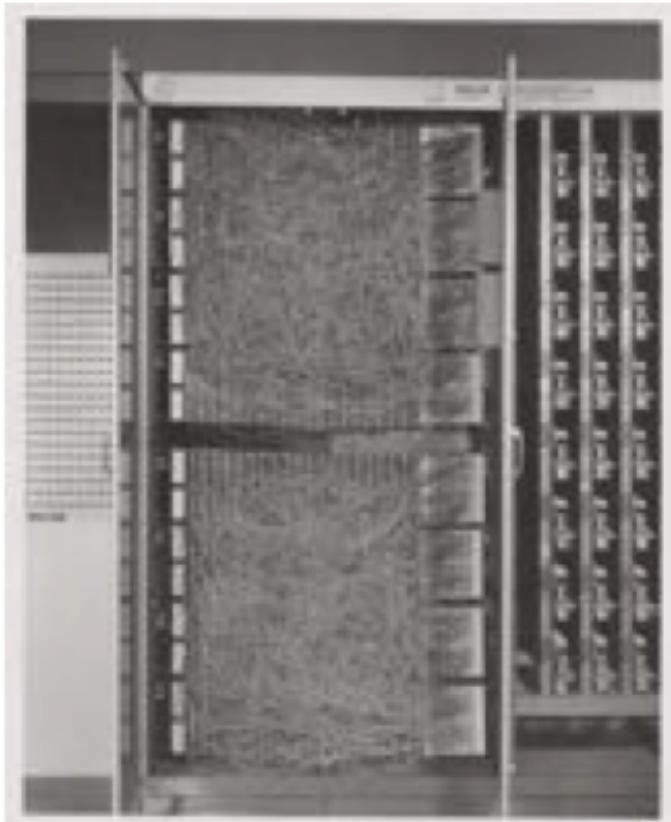
INSPIRED BY NEURO - SCIENCE?

Credit: Karpathy

Rosenblatt Perceptron

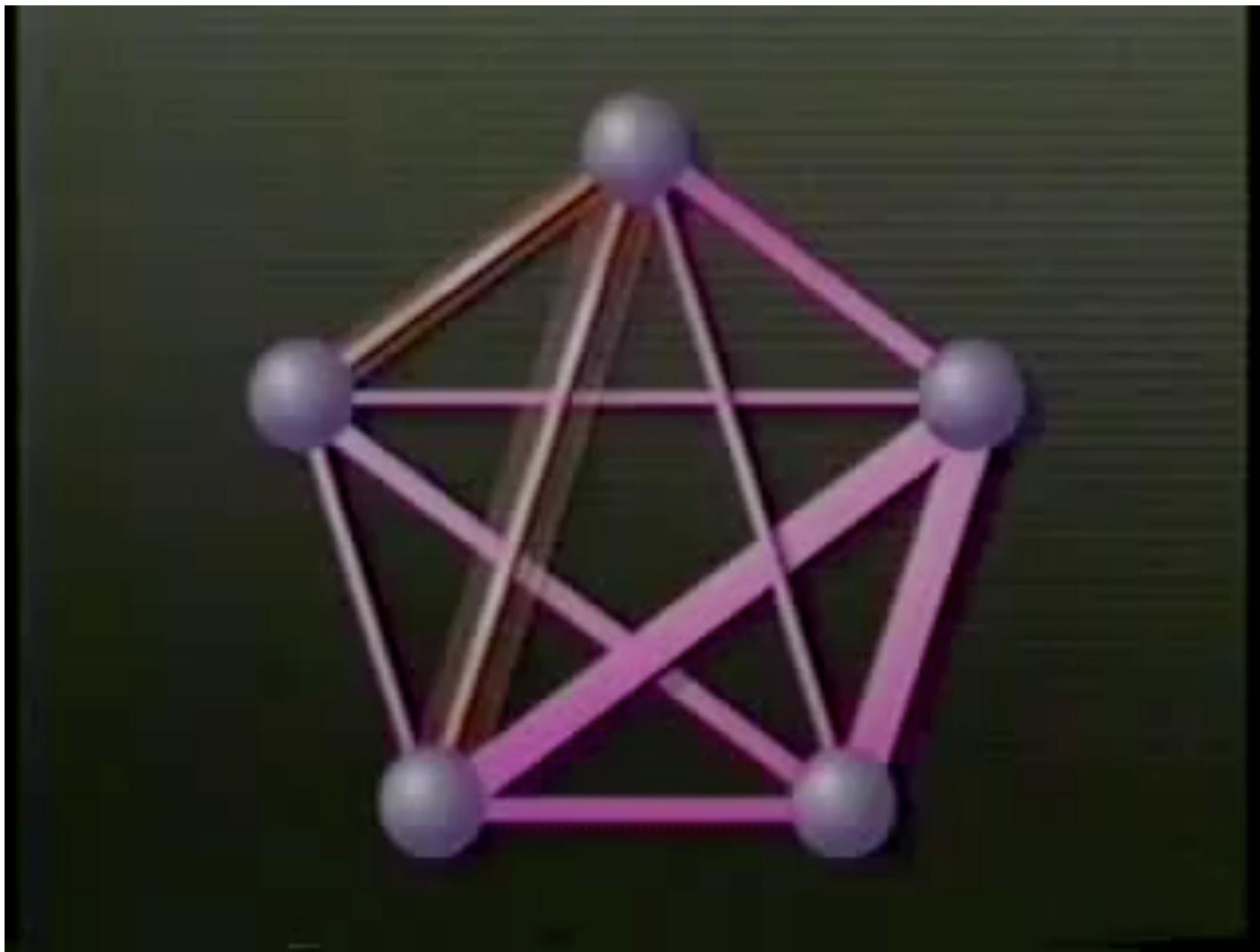
FIRST IMPLEMENTATION OF NEURAL NETWORK [Rosenblatt, 1957!]

INTENDED TO BE A MACHINE (NOT AN ALGORITHM)

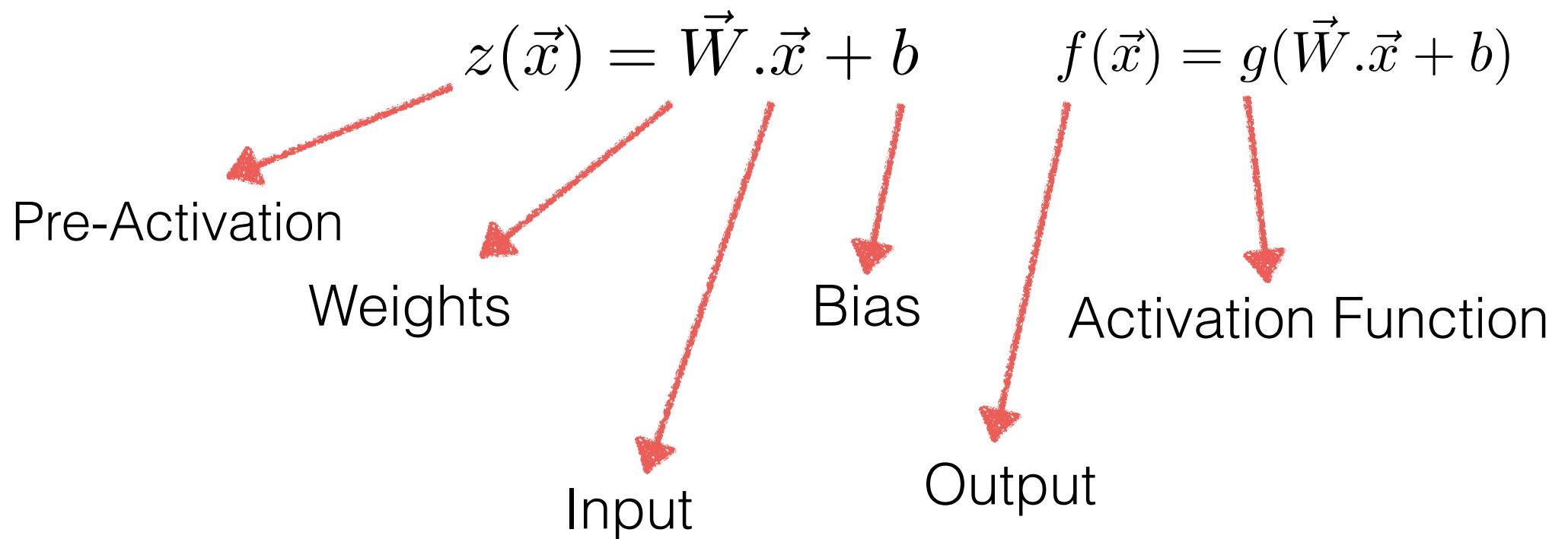
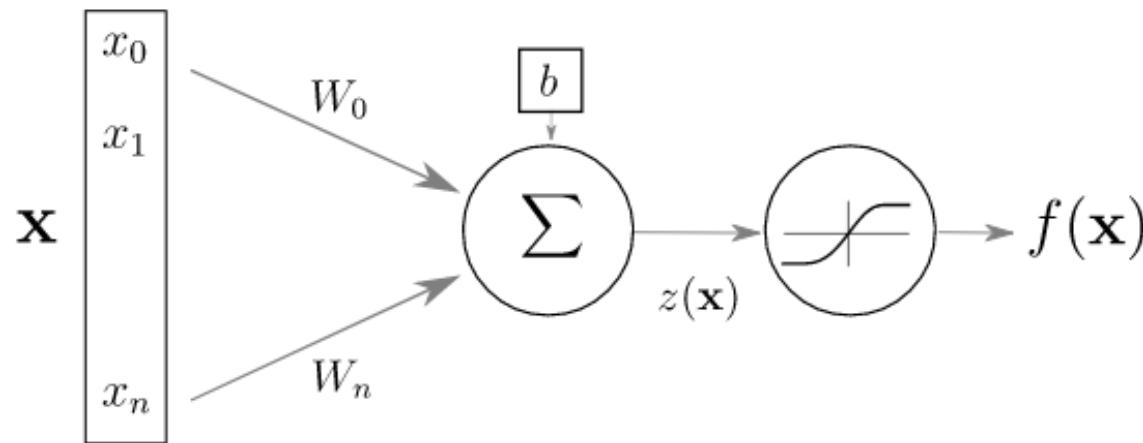


it had an array of 400 photocells,
randomly connected to the "neurons".

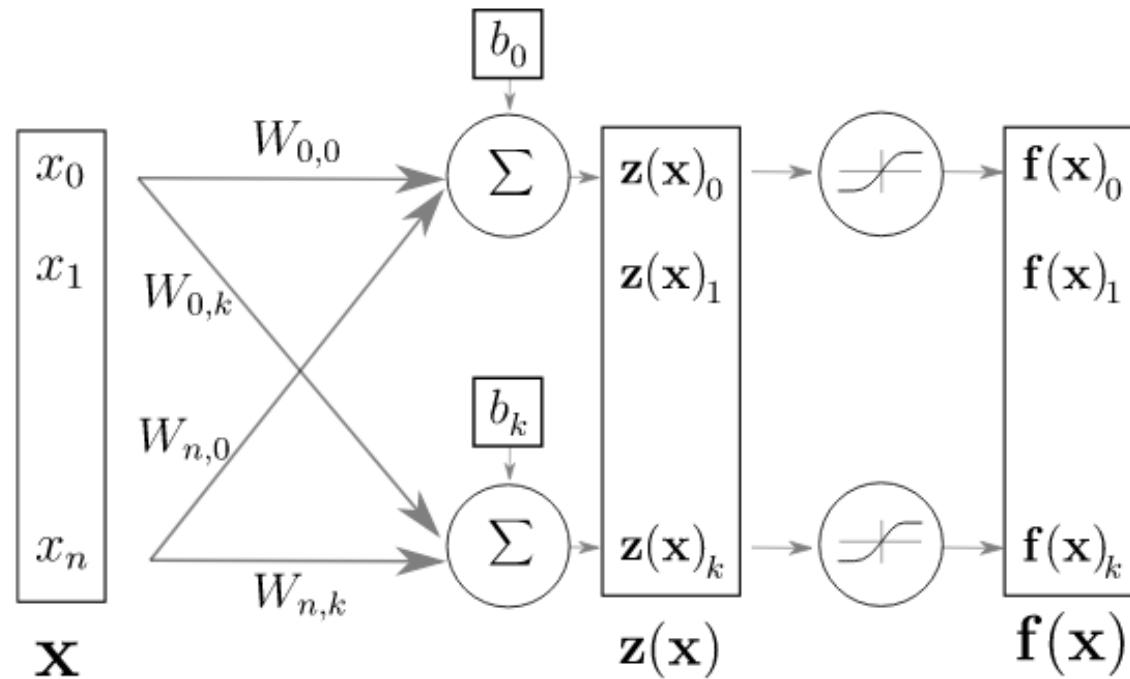
Weights were encoded in
potentiometers, and weight updates
during learning were performed by
electric motors



TODAY'S PERCEPTRON



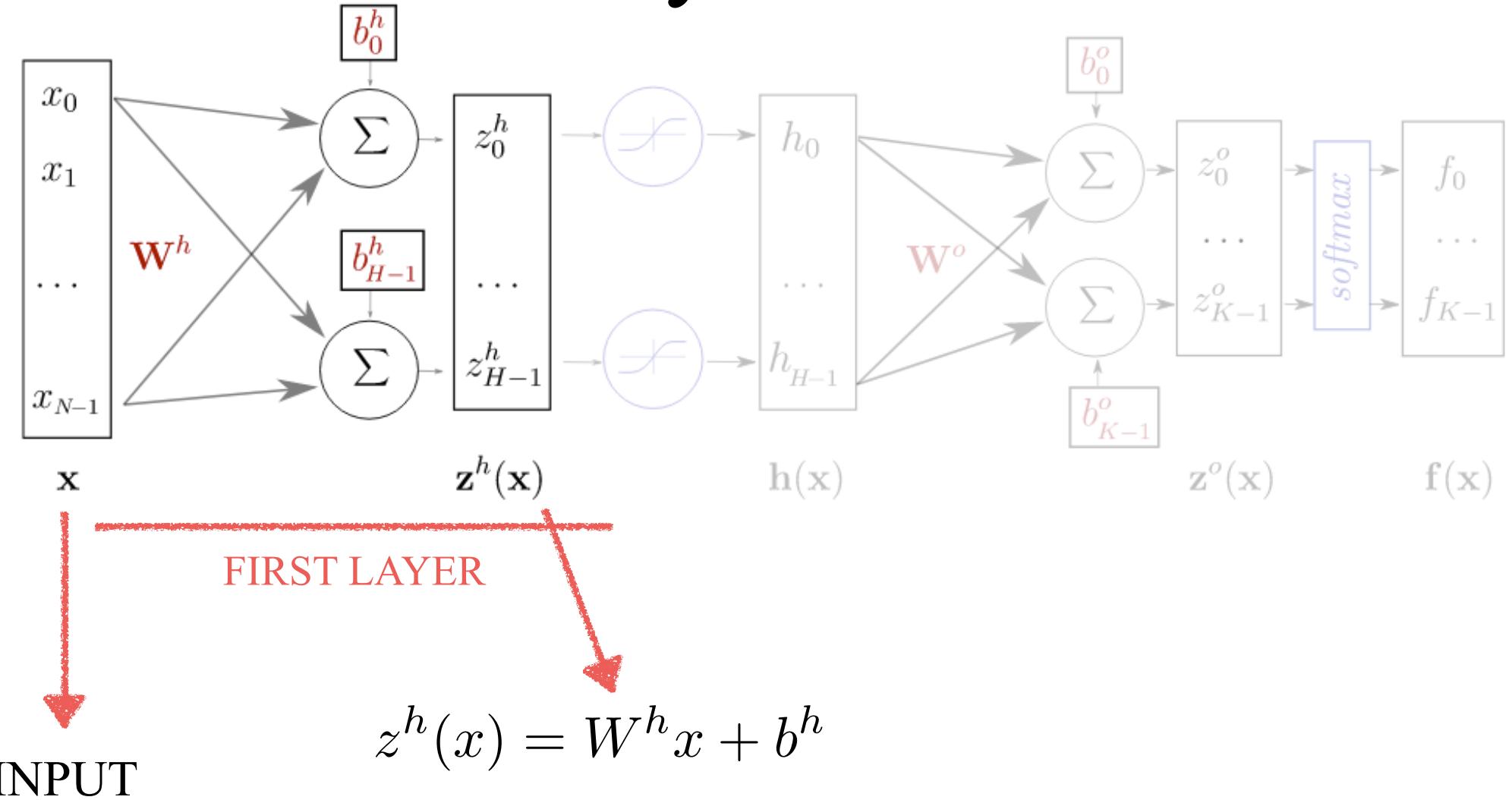
LAYER OF NEURONS



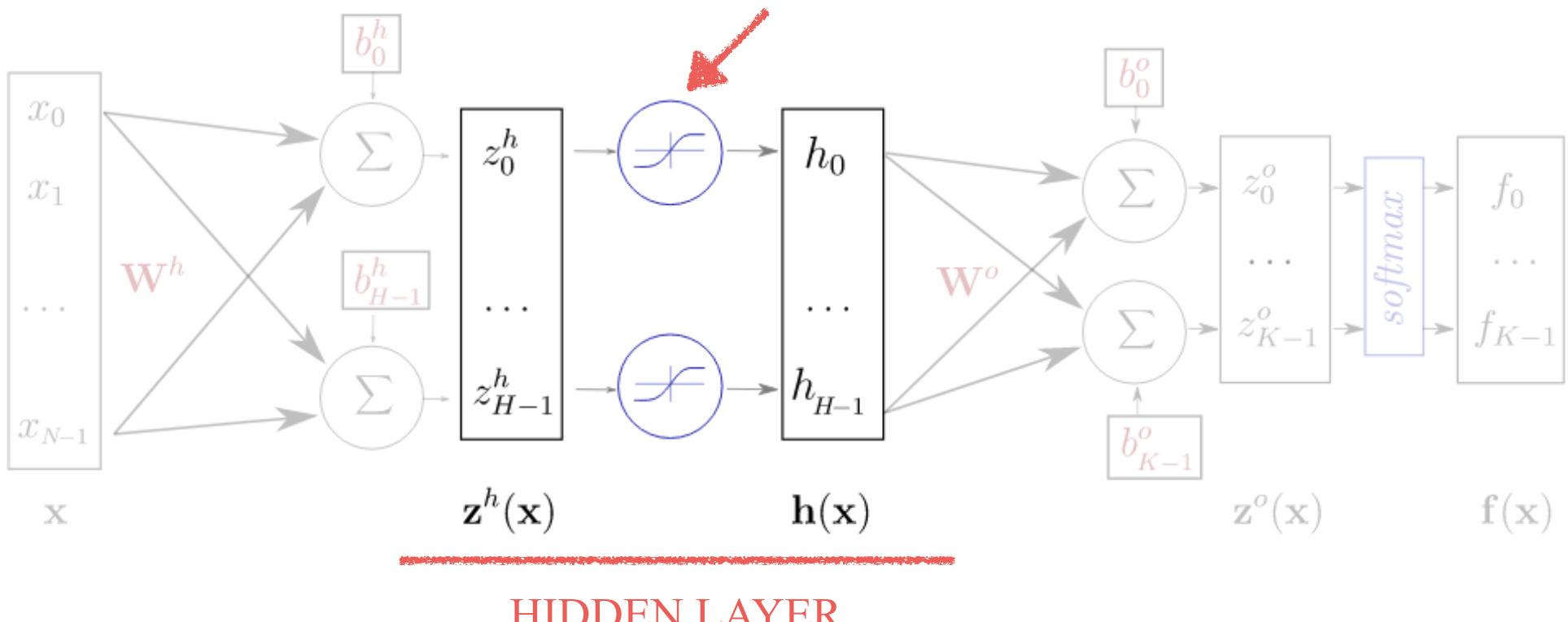
$$f(\vec{x}) = g(\mathbf{W} \cdot \vec{x} + \vec{b})$$

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

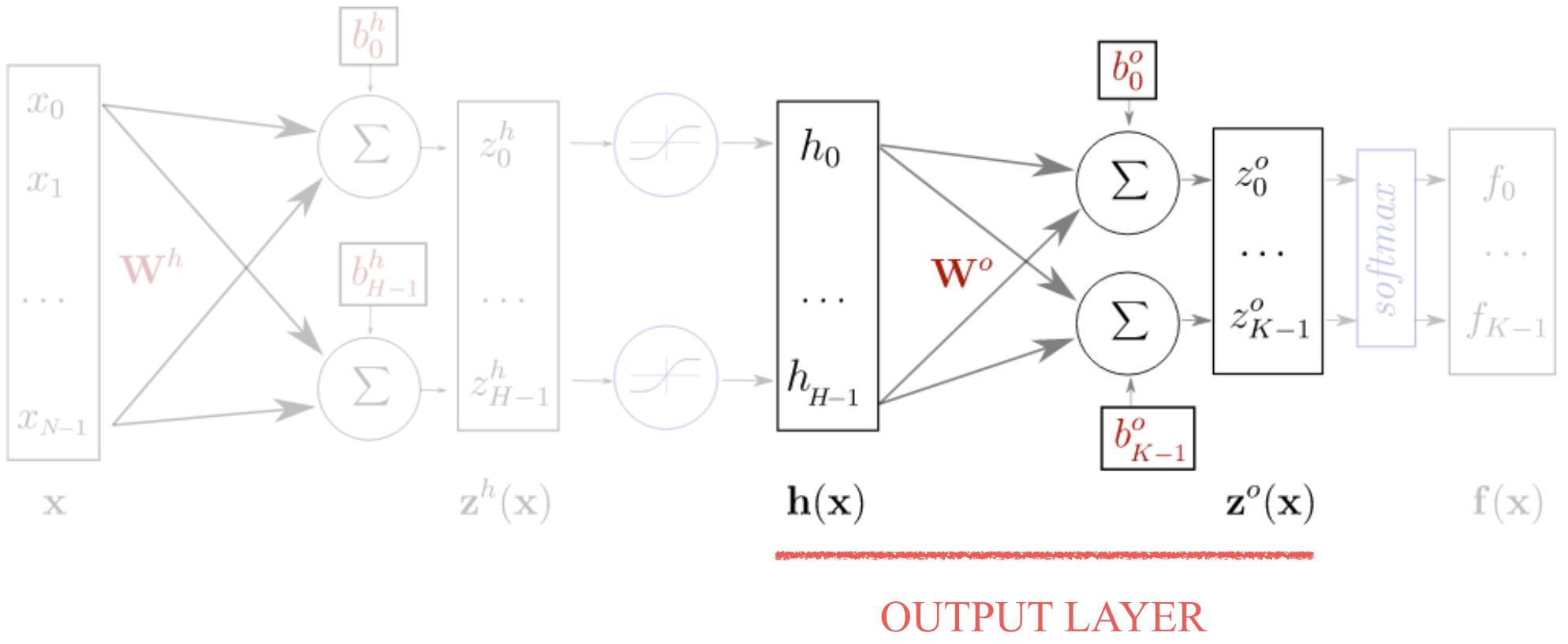
Hidden Layers of Neurons



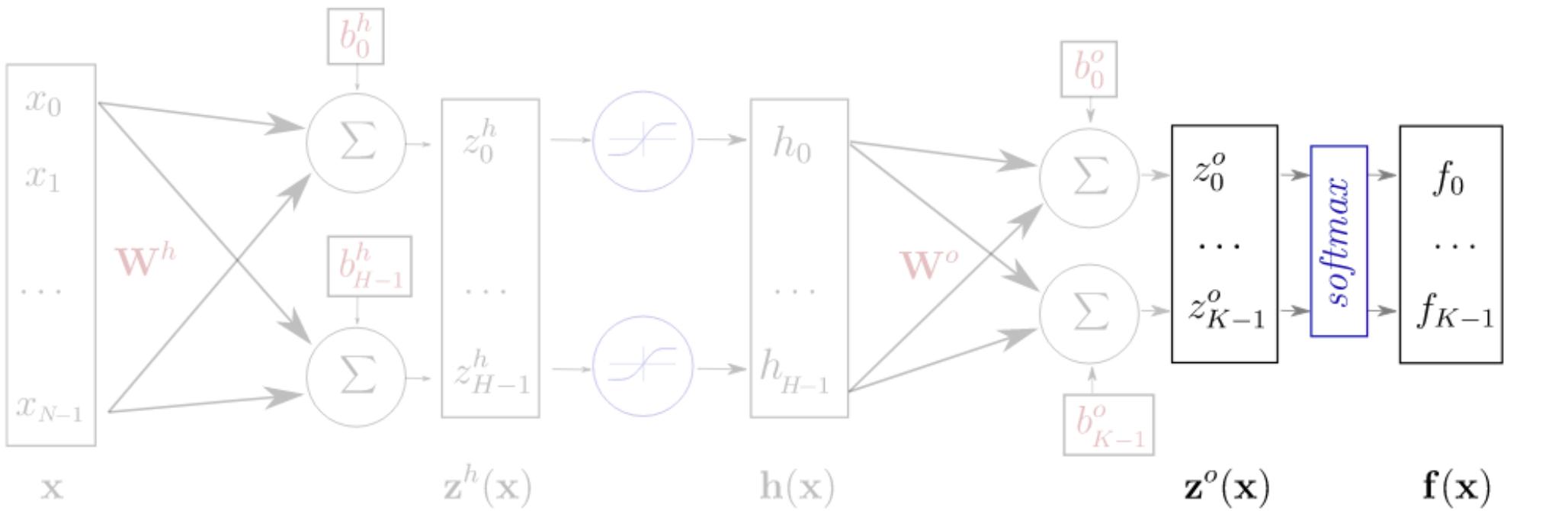
ACTIVATION FUNCTION



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



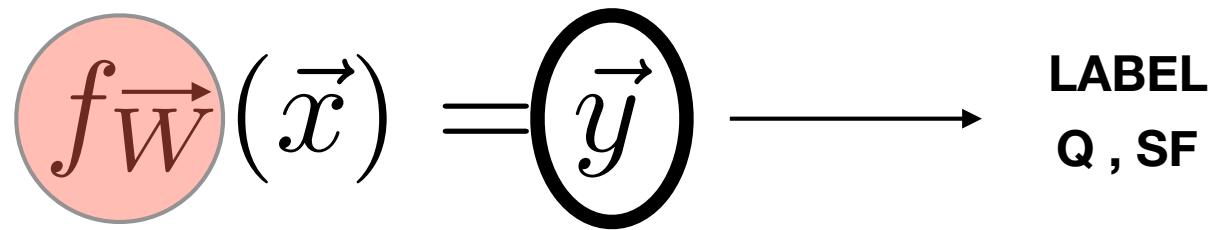
$$z^0(\mathbf{x}) = W^0 h(\mathbf{x}) + b^0$$



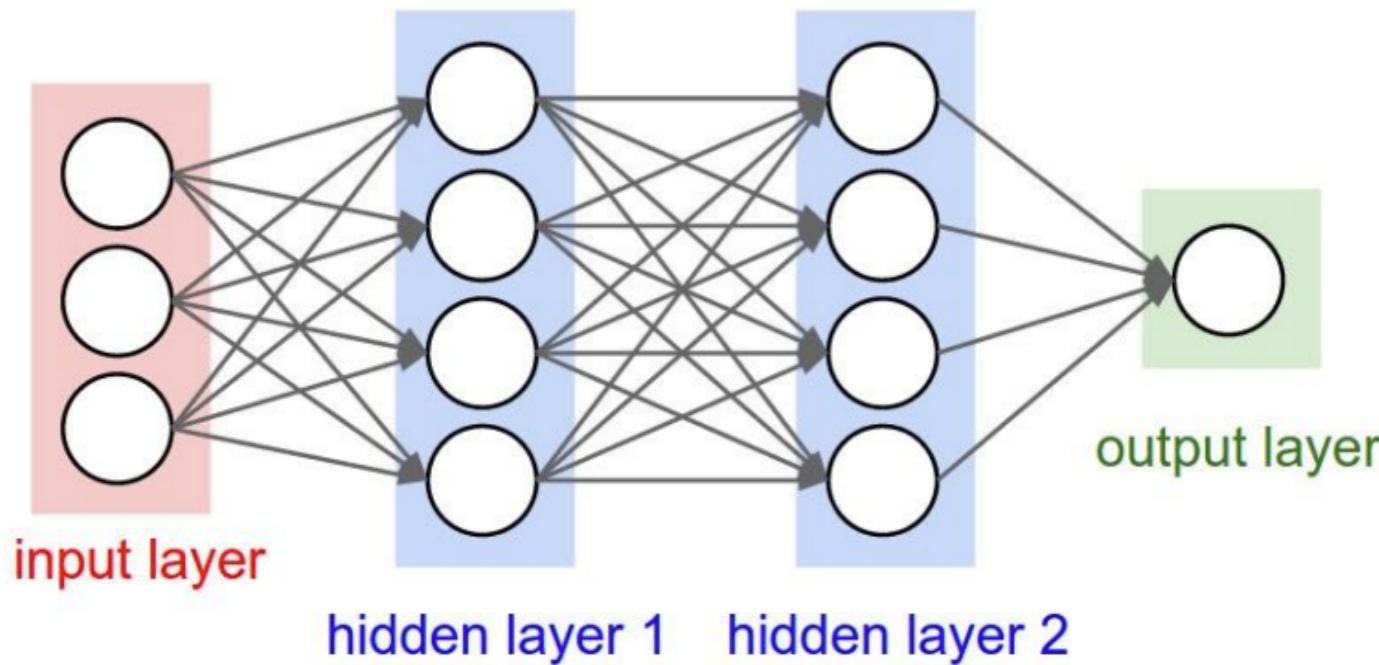
PREDICTION LAYER

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

**"CLASSICAL"
MACHINE LEARNING**

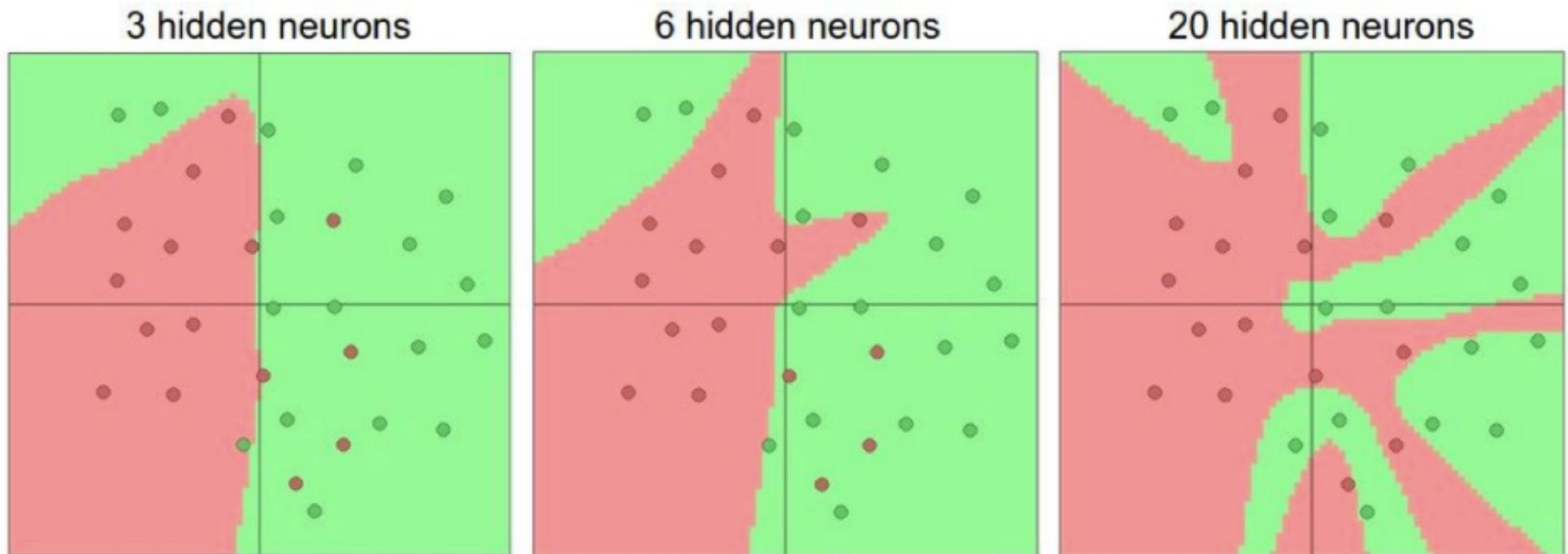


**REPLACE THIS BY A GENERAL
NON LINEAR FUNCTION WITH SOME PARAMETERS W**



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

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!

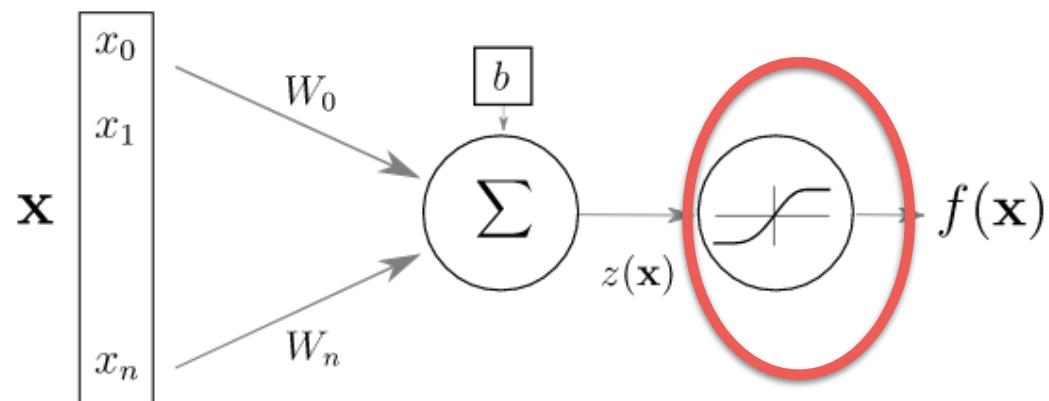
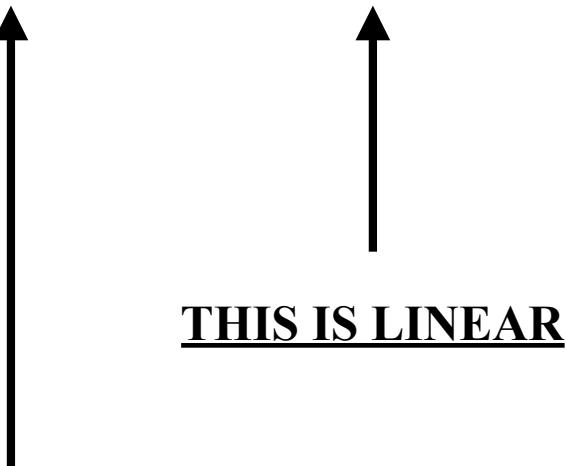
YES BUT...

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

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

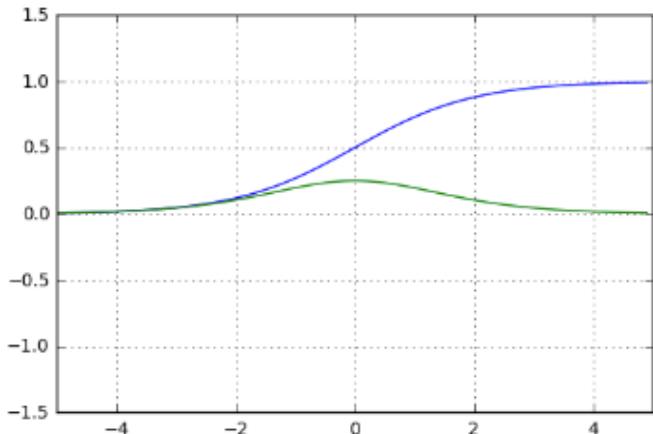
ACTIVATION FUNCTIONS?

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



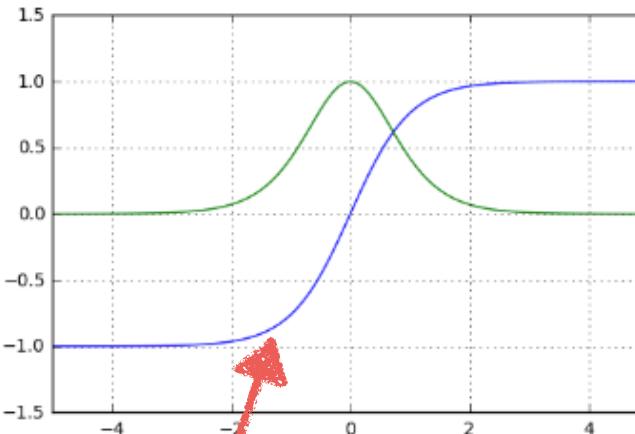
ADD NON LINEARITIES TO THE PROCESS

ACTIVATION FUNCTIONS



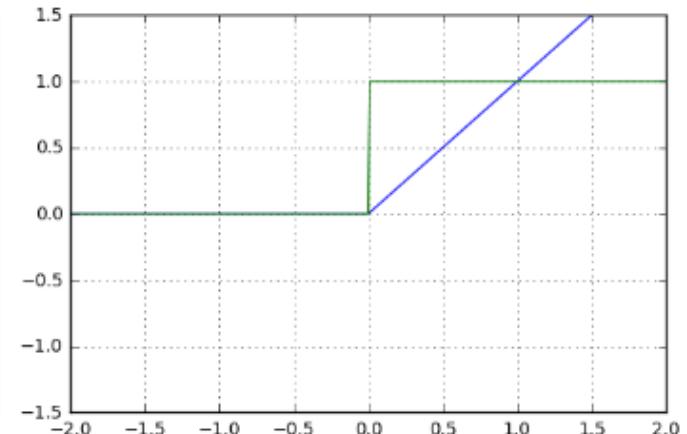
$$\text{sigm}(x) = \frac{1}{1 + e^{-x}}$$

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



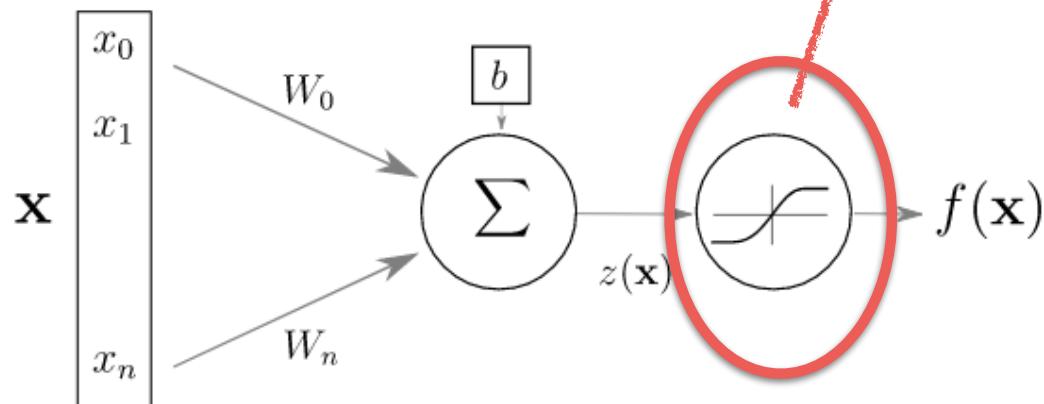
$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

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



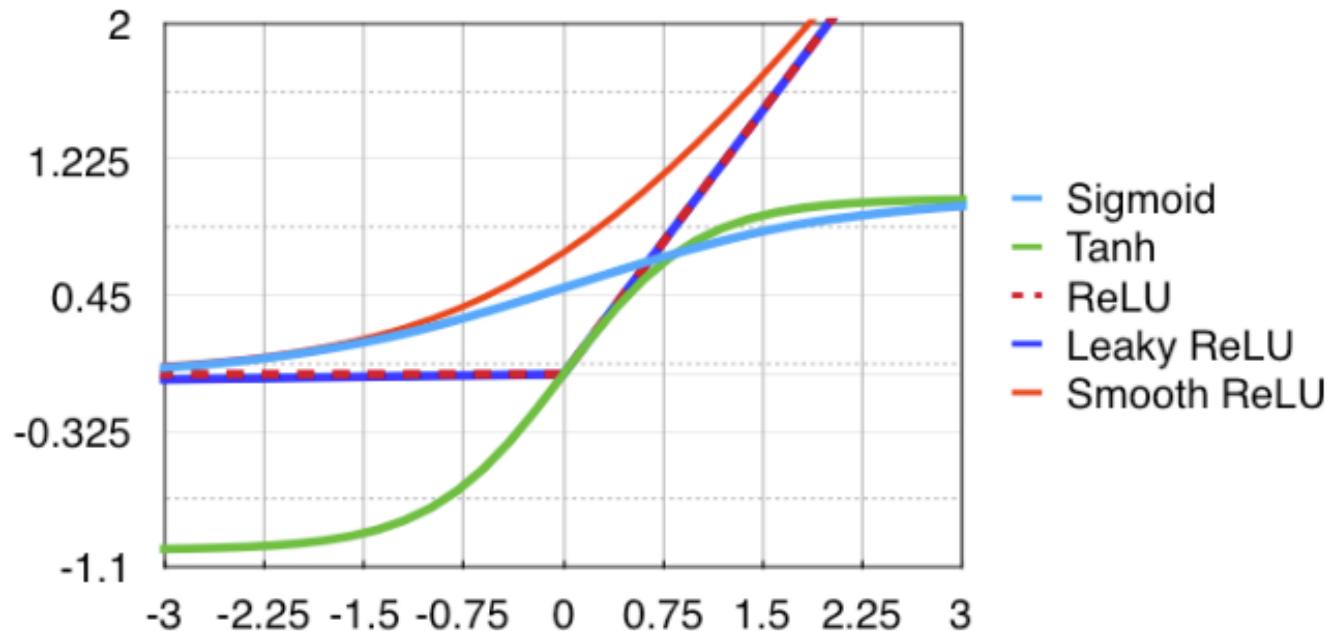
$$\text{relu}(x) = \max(0, x)$$

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



**ALL ACTIVATION FUNCTIONS
HAVE A TYPICAL
“THRESHOLD” FOR
ACTIVATION**

ACTIVATION FUNCTIONS



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

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

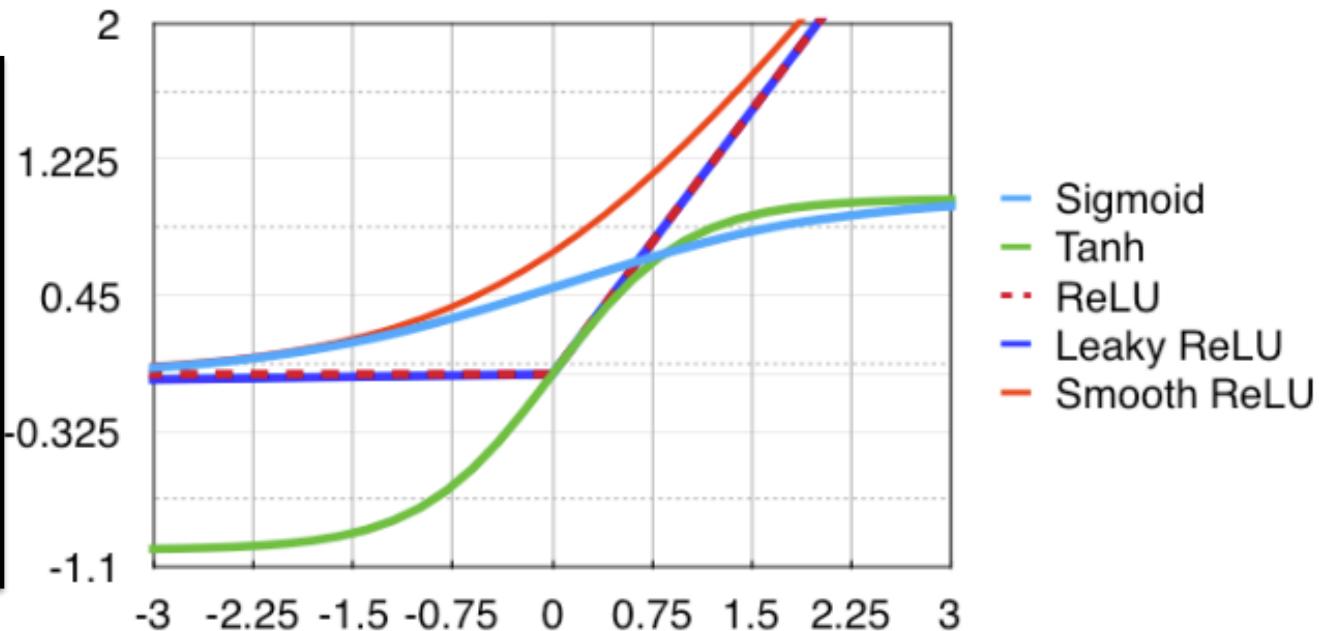
Tanh: $f(x) = \tanh(x)$

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

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

SOFTMAX

A generalization of the SIGMOID ACTIVATION

THE OUTI

GENERALLY
USED AS ACTIVATION
OF LAST LAYER

AND 1

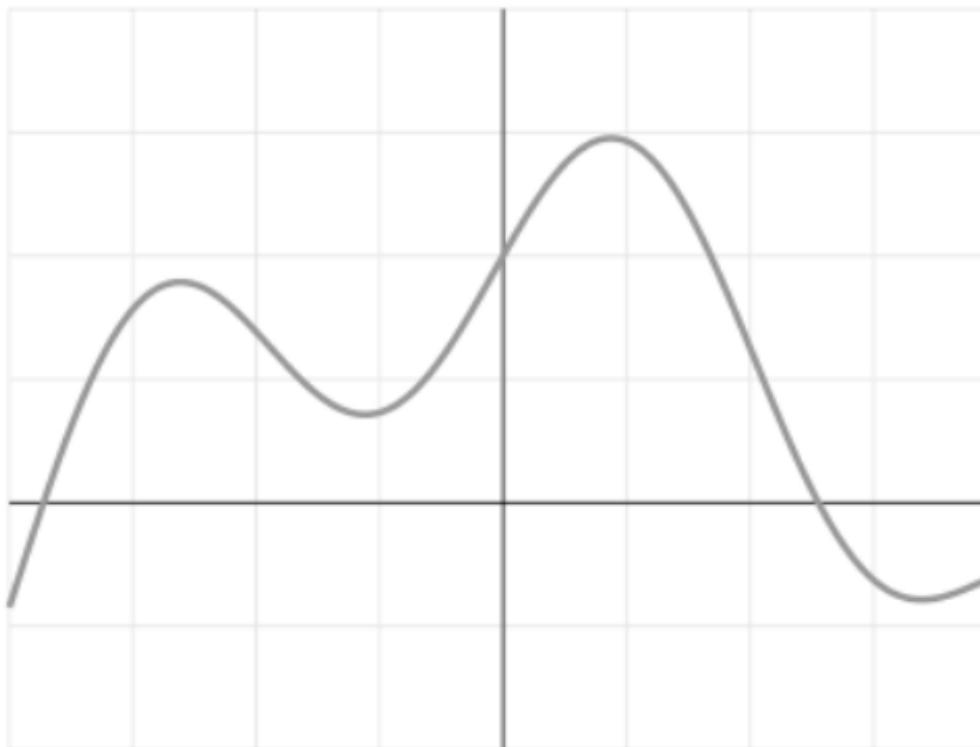
(will come back later)

THE SUM OF EXPONENTS ADDS UP TO 1

CAN BE INTERPRETED AS A PROBABILITY

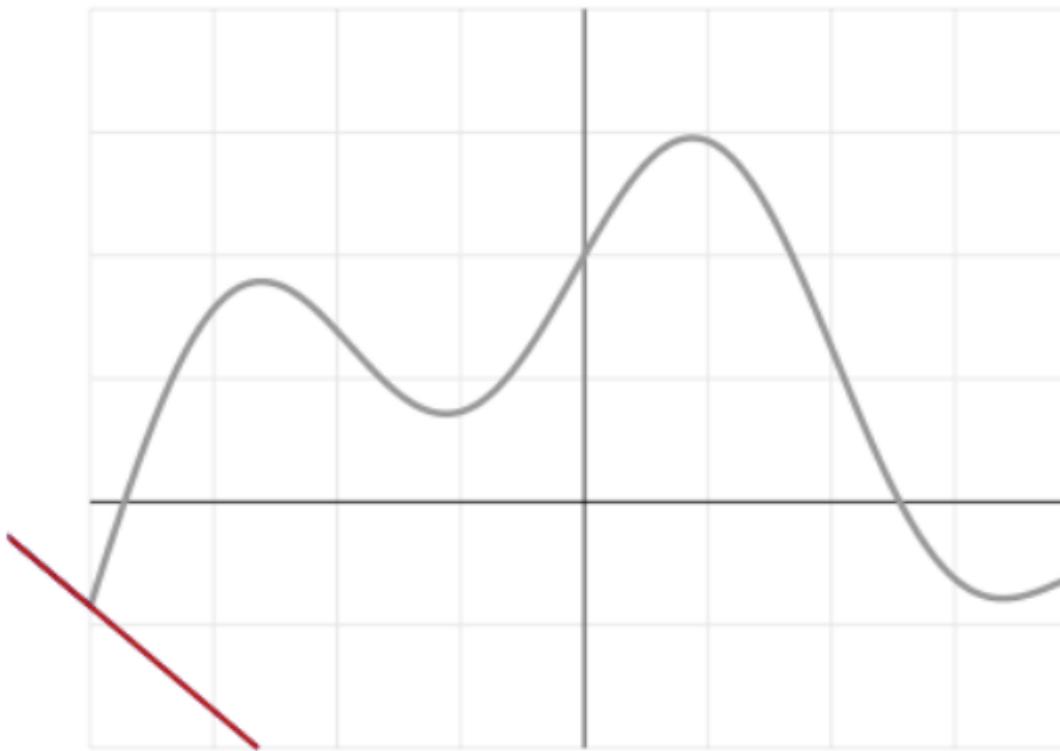
$$p(Y = c | X = \mathbf{x}) = softmax(z(\mathbf{x}))_c$$

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



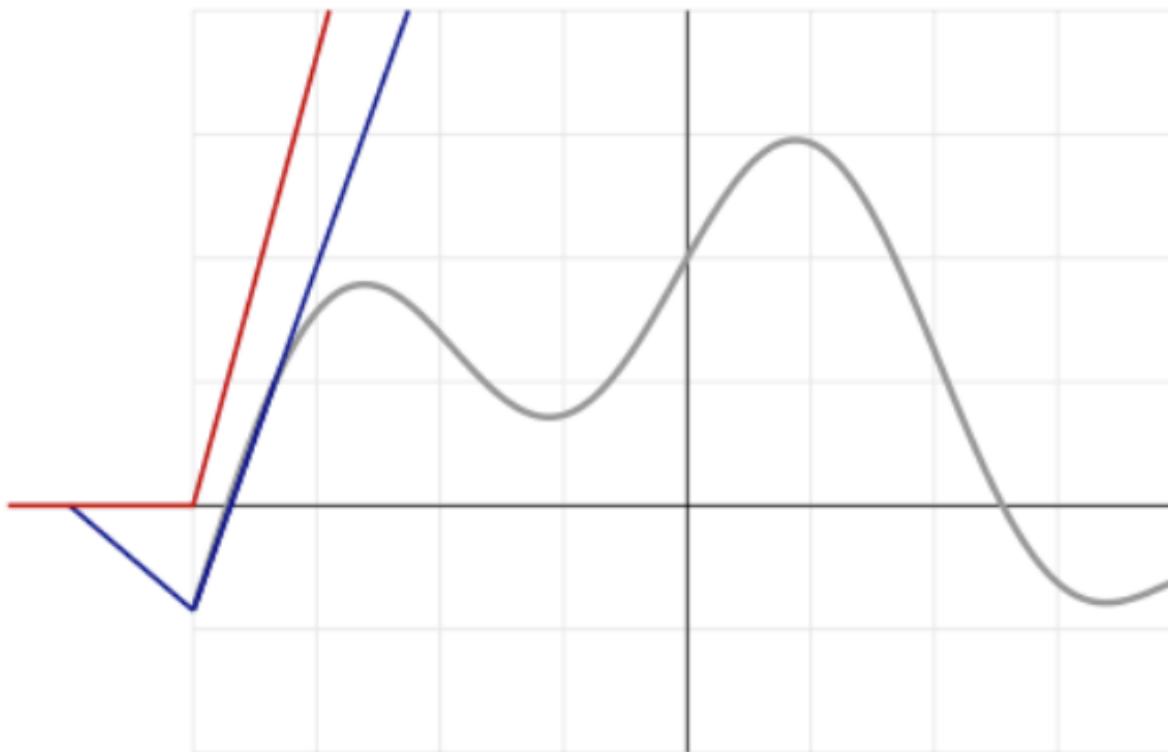
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



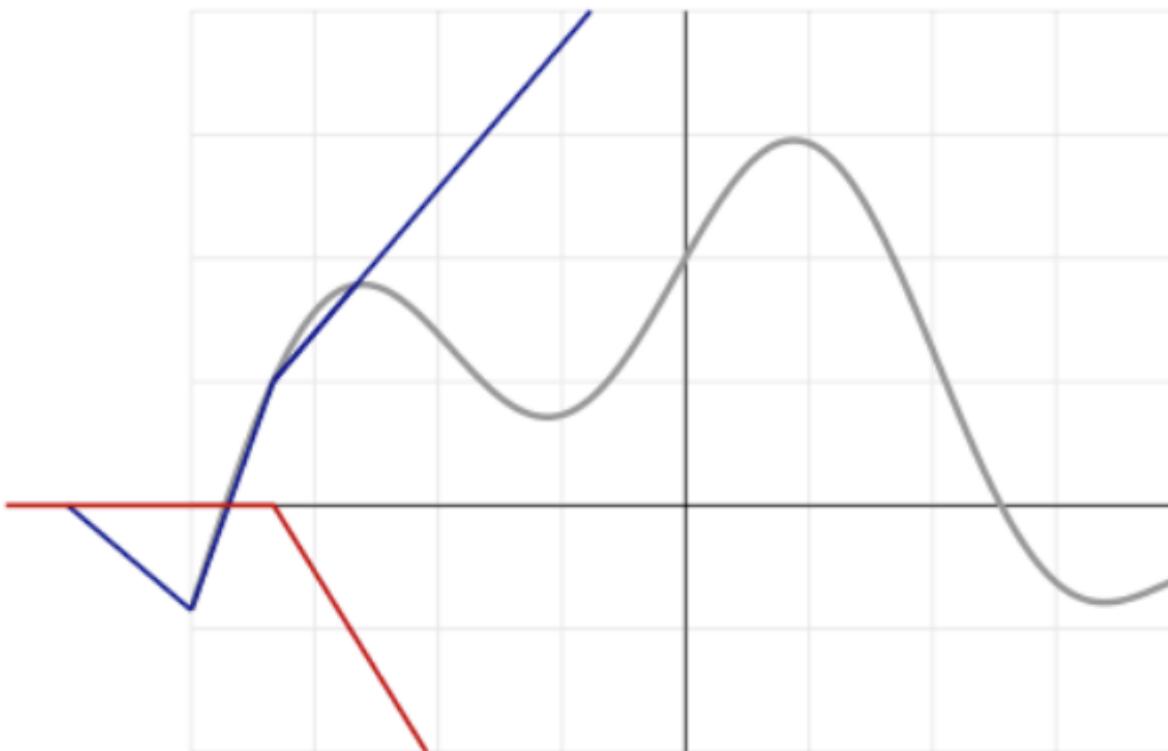
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



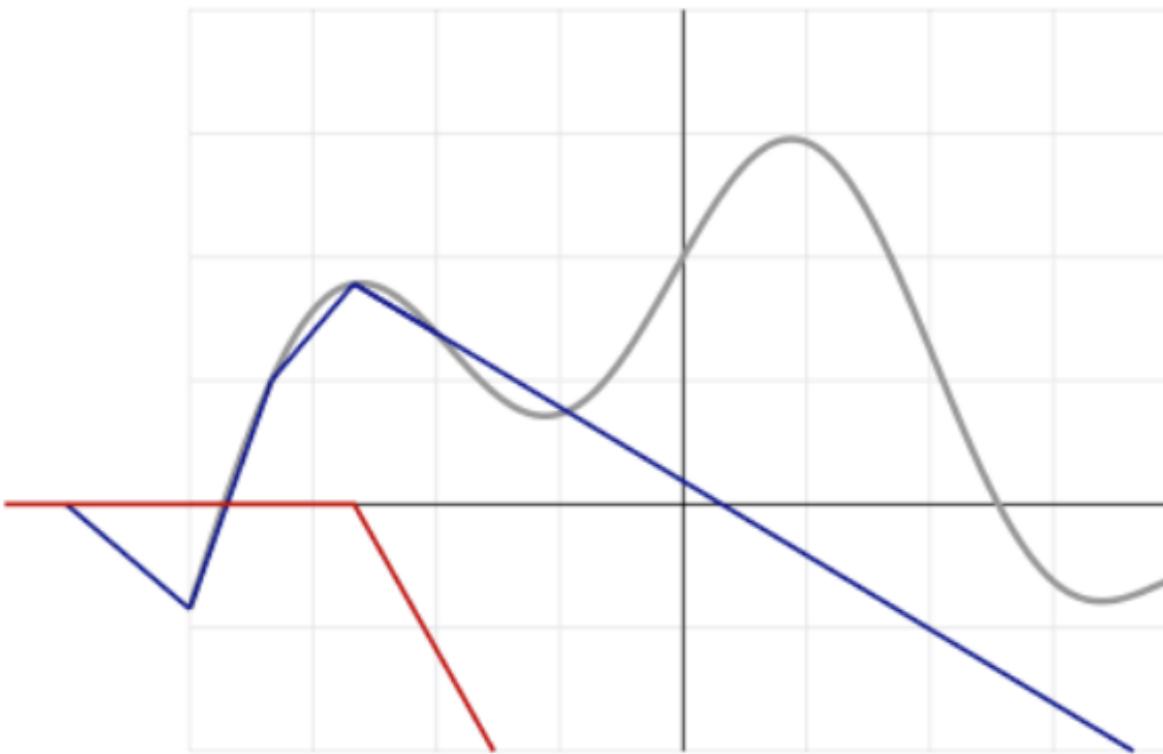
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



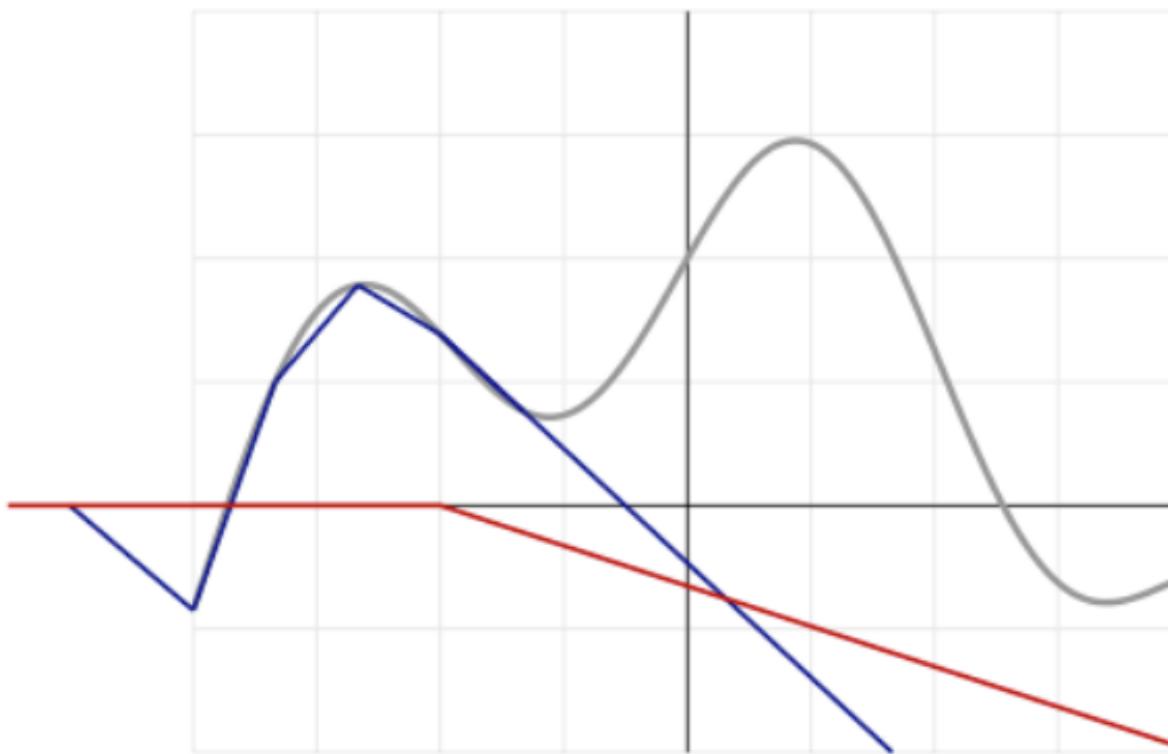
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



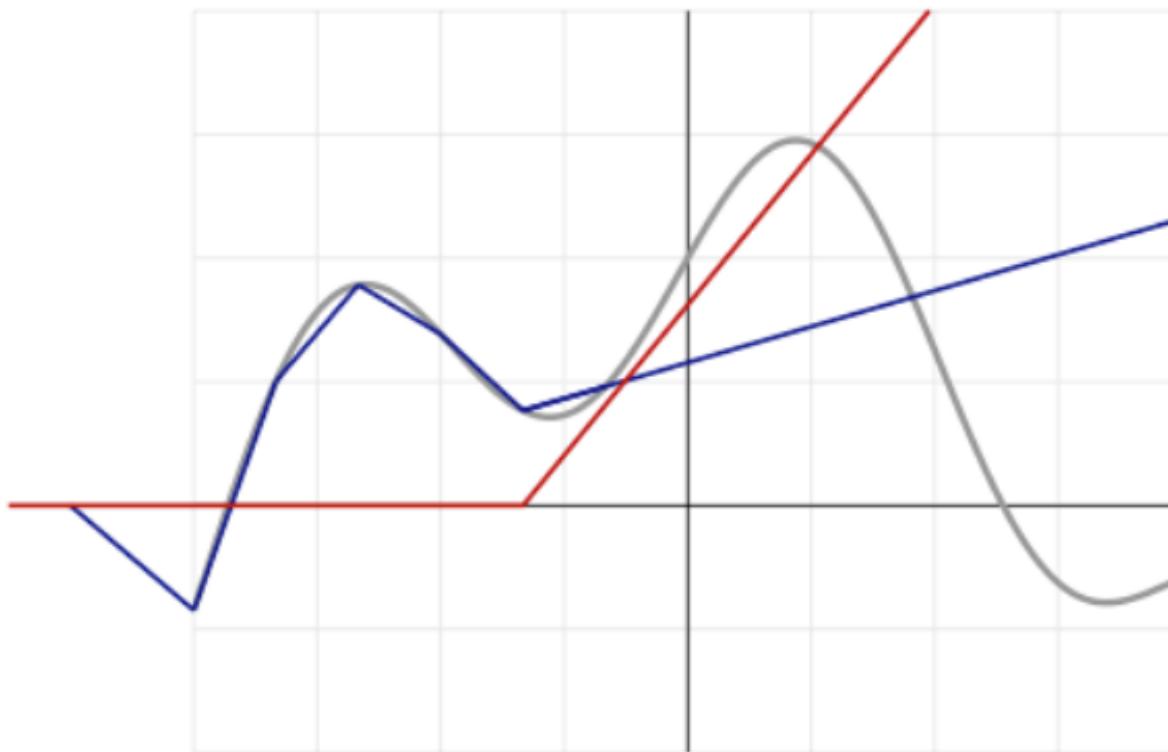
Any real function in an interval (a, b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



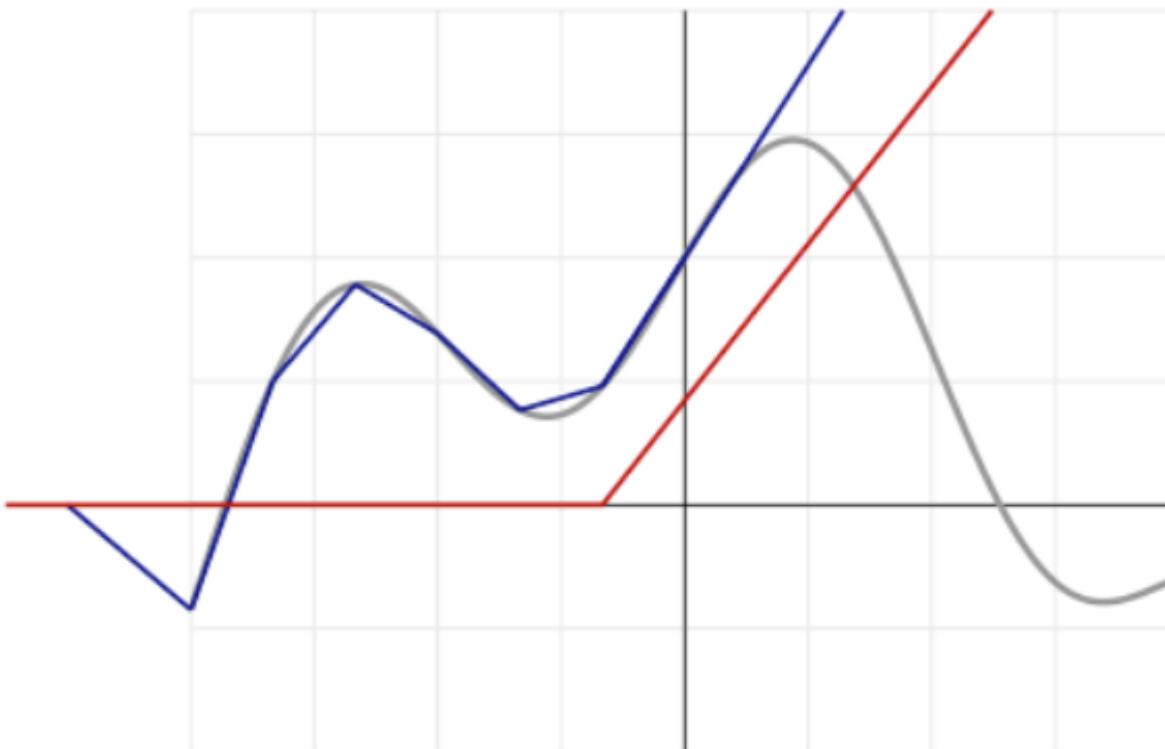
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



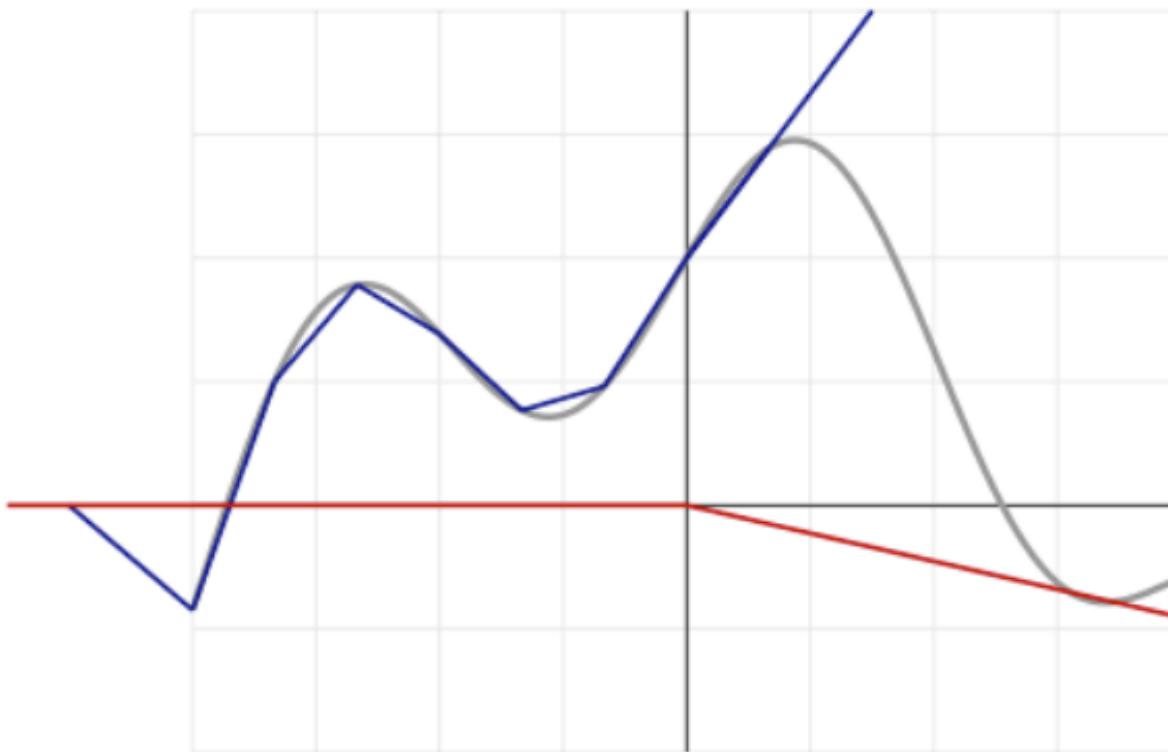
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



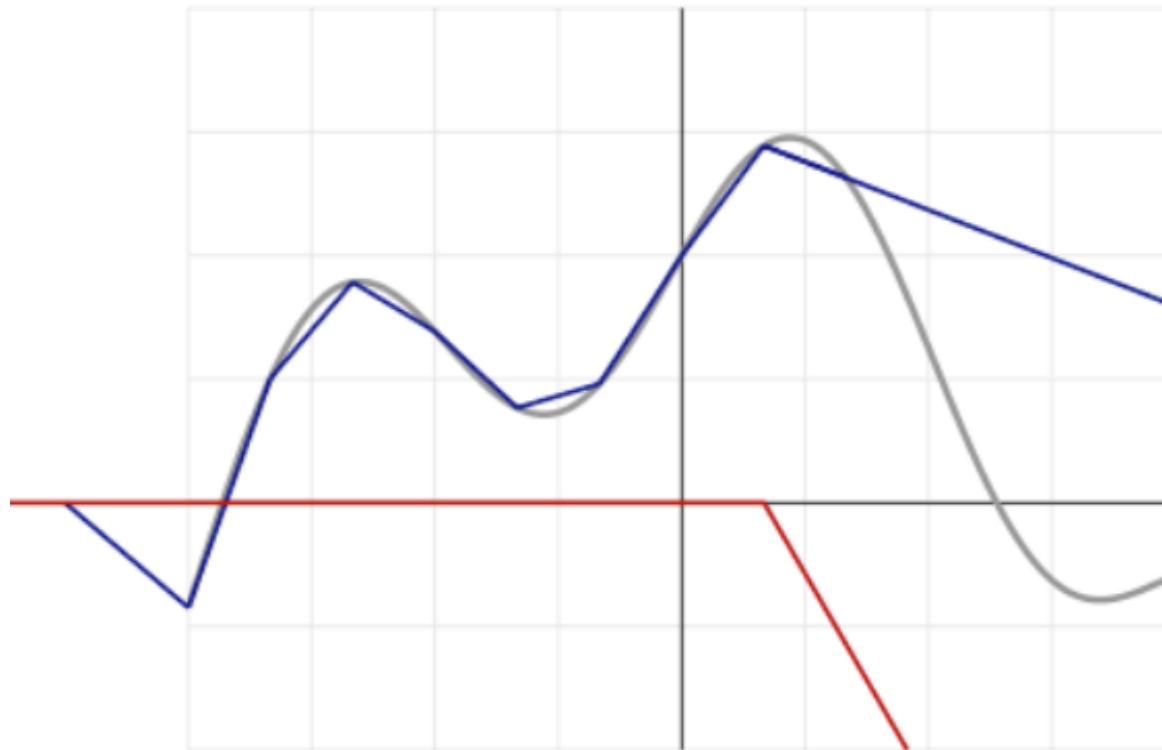
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



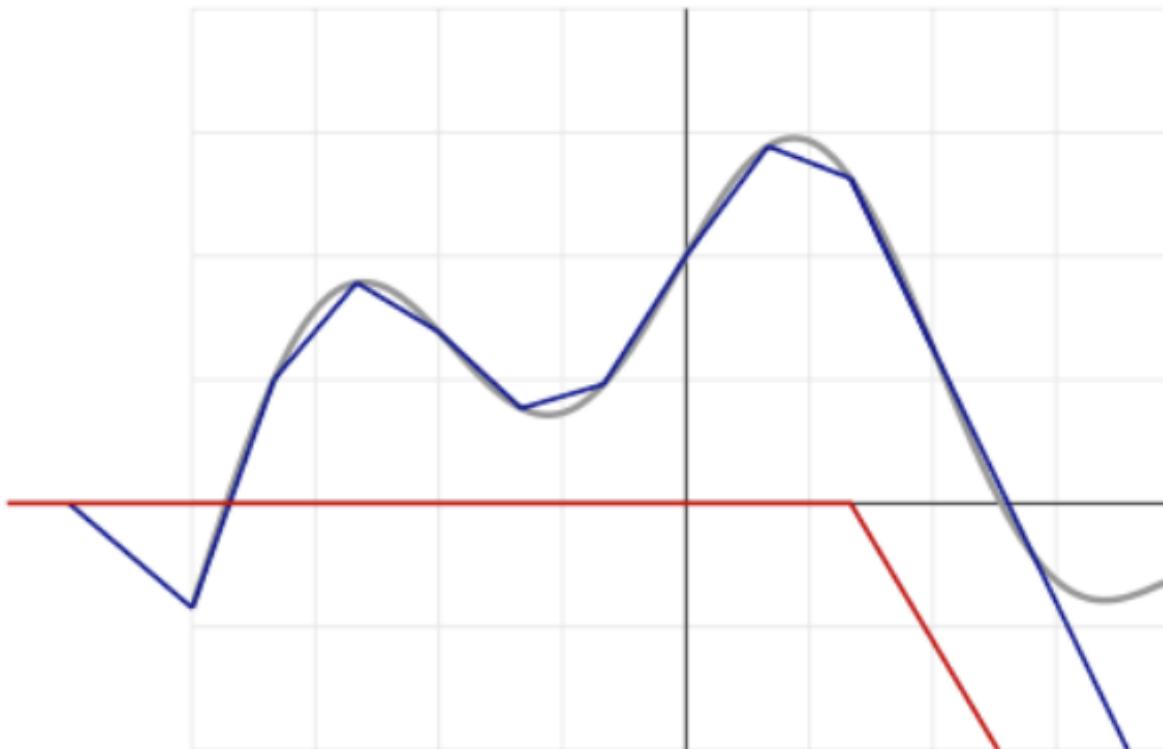
Any real function in a interval (a, b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



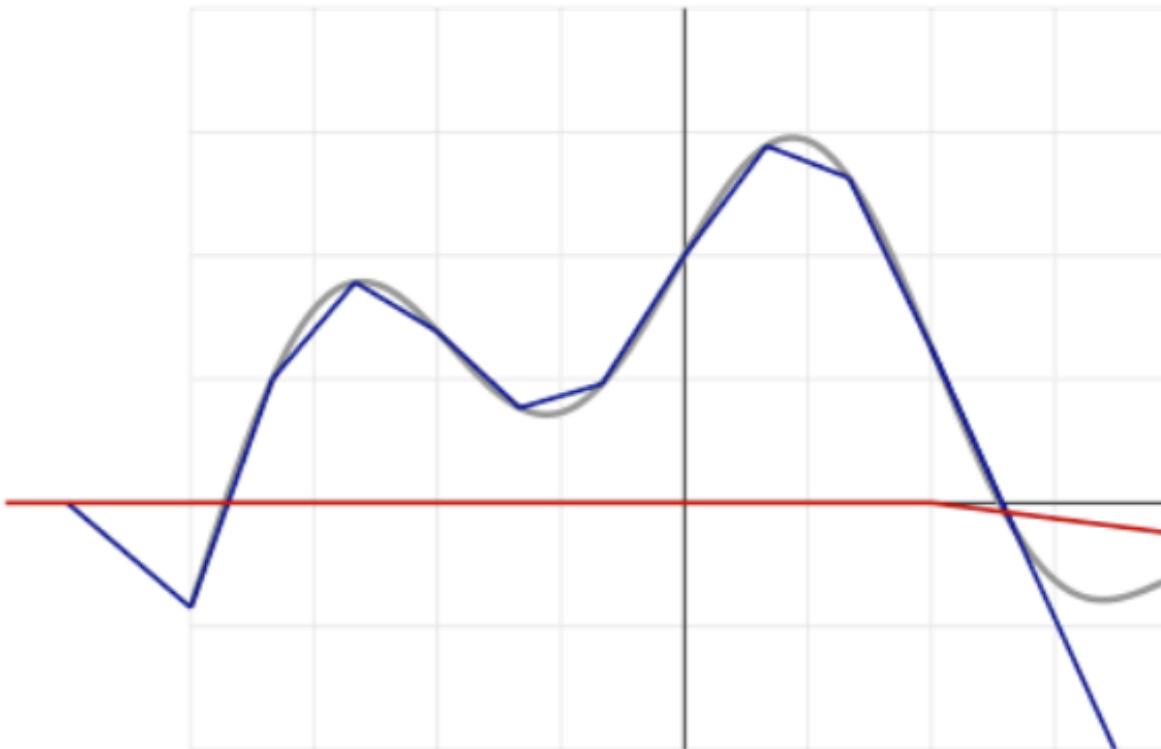
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



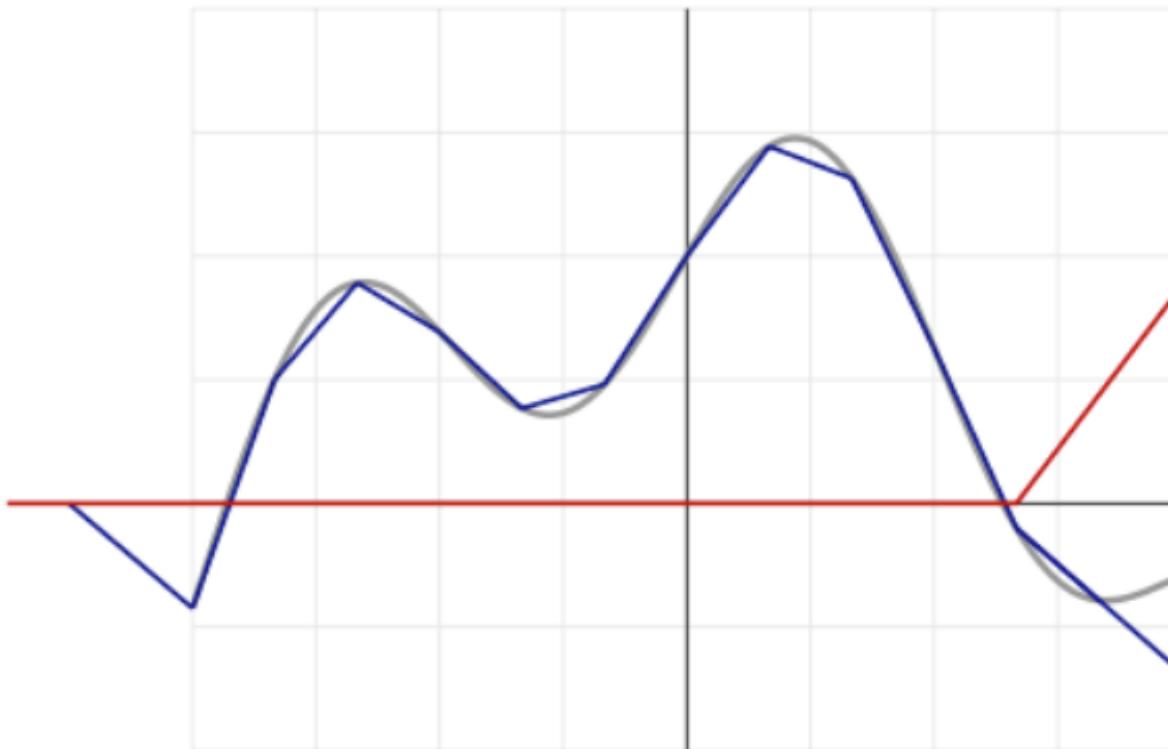
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



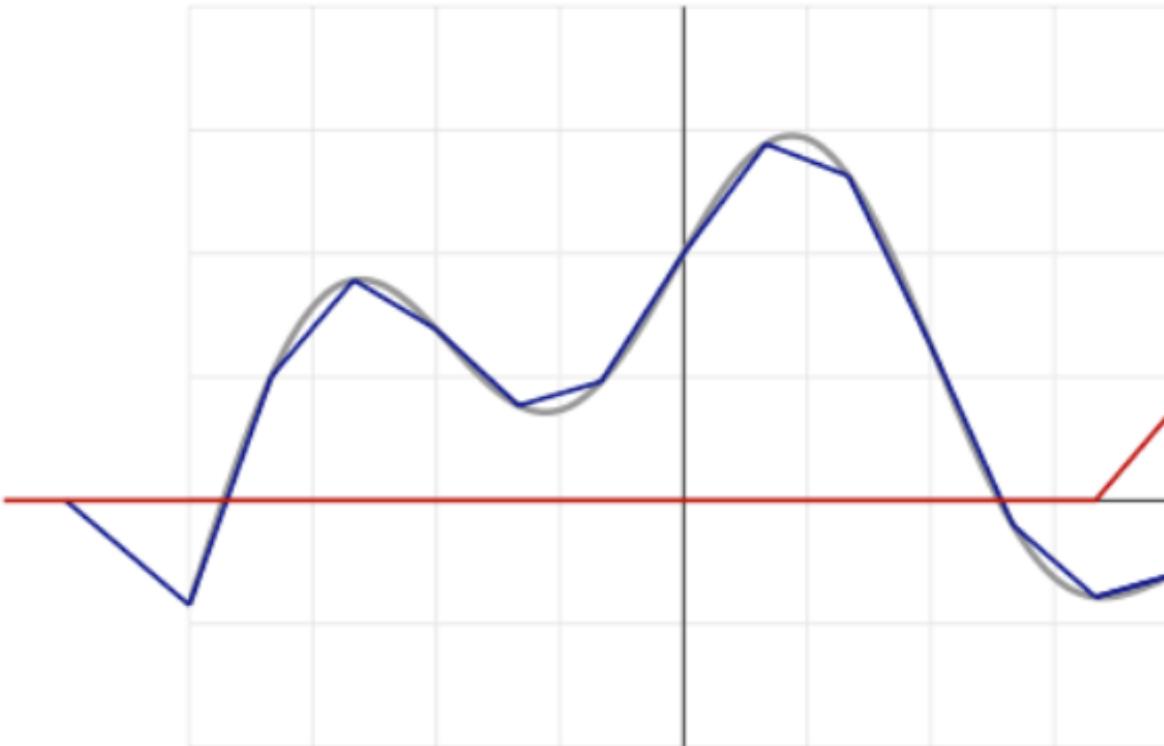
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

WHAT IS THE MEANING OF THE ACTIVATION FUNCTION?



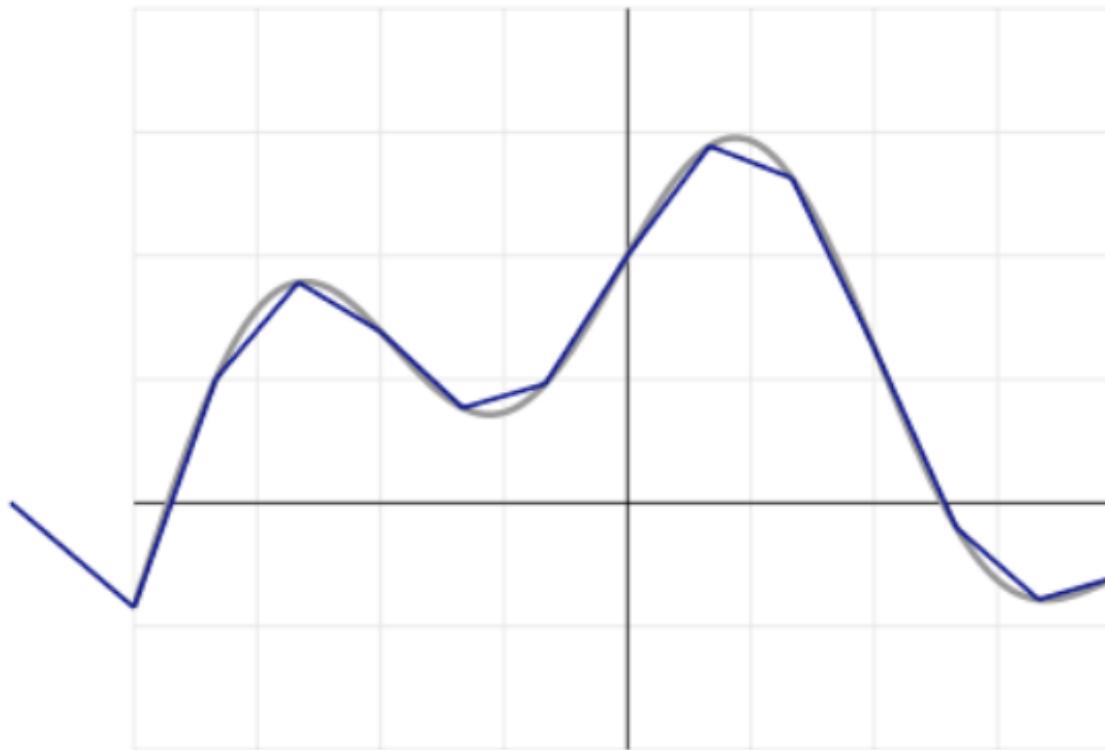
Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

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?

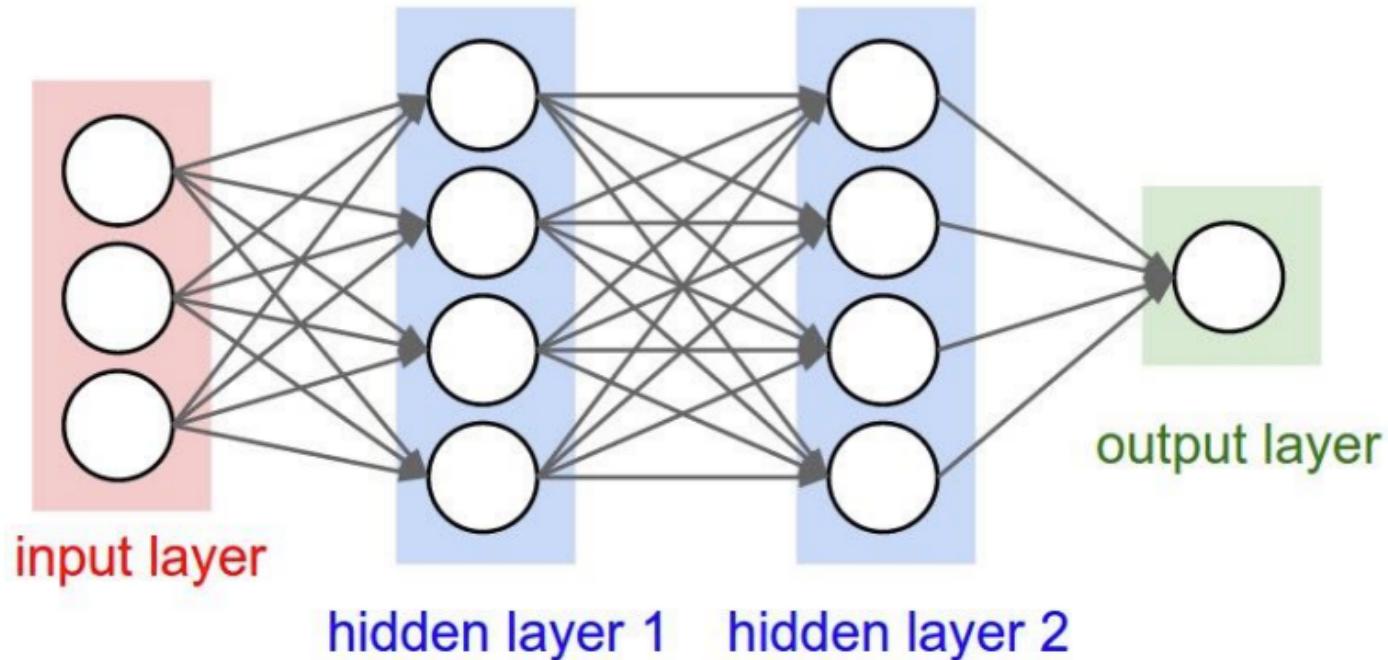


Any real function in an interval (a,b) can be approximated with a linear combination of translated and scaled ReLu functions

OK, SO NOW LET'S FIND
THE WEIGHTS

OPTIMIZATION

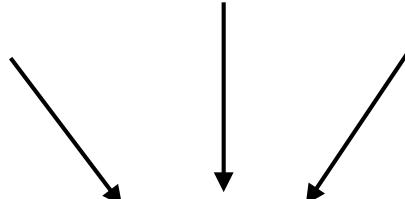
[OR HOW TO FIND THE WEIGHTS?]



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



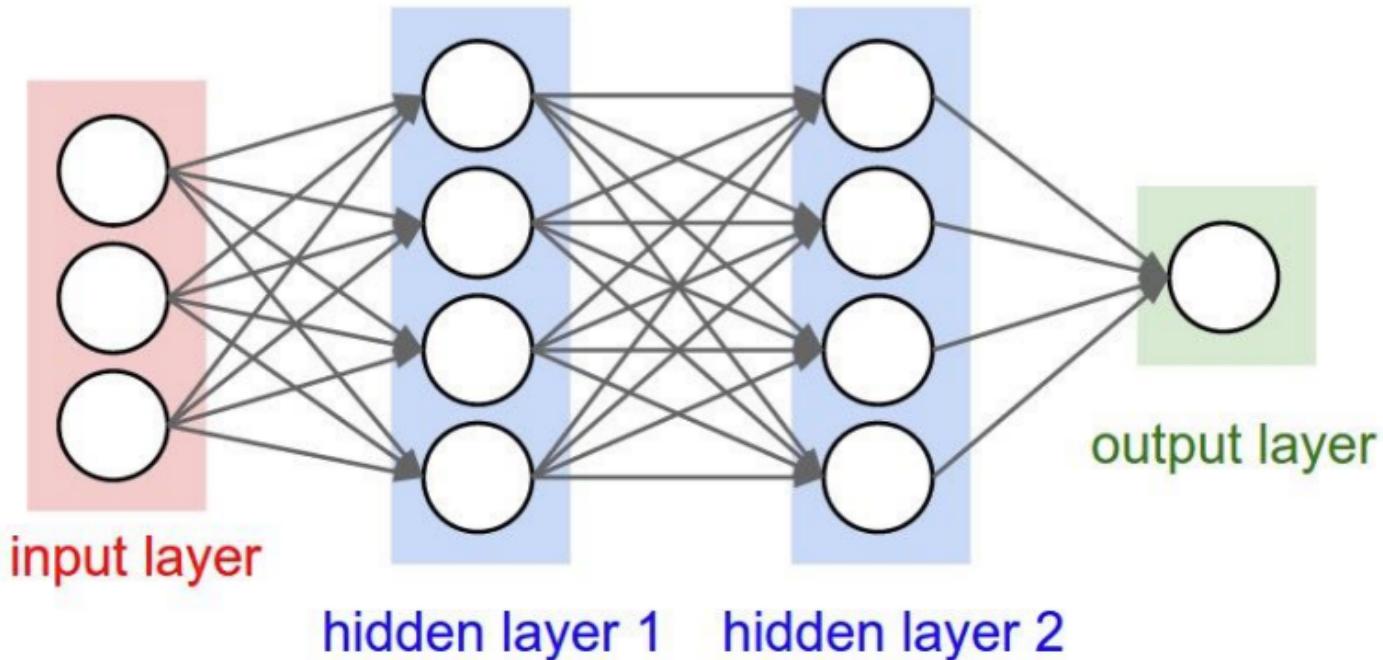
NETWORK
FUNCTION



THIS IS WHAT WE ARE LOOKING FOR

OPTIMIZATION

[OR HOW TO FIND THE WEIGHTS?]



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

WE FIRST ASSUME A
LOSS FUNCTION

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

←
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

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

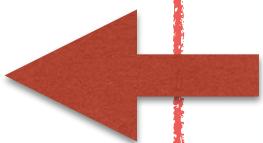


[hessian]

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...

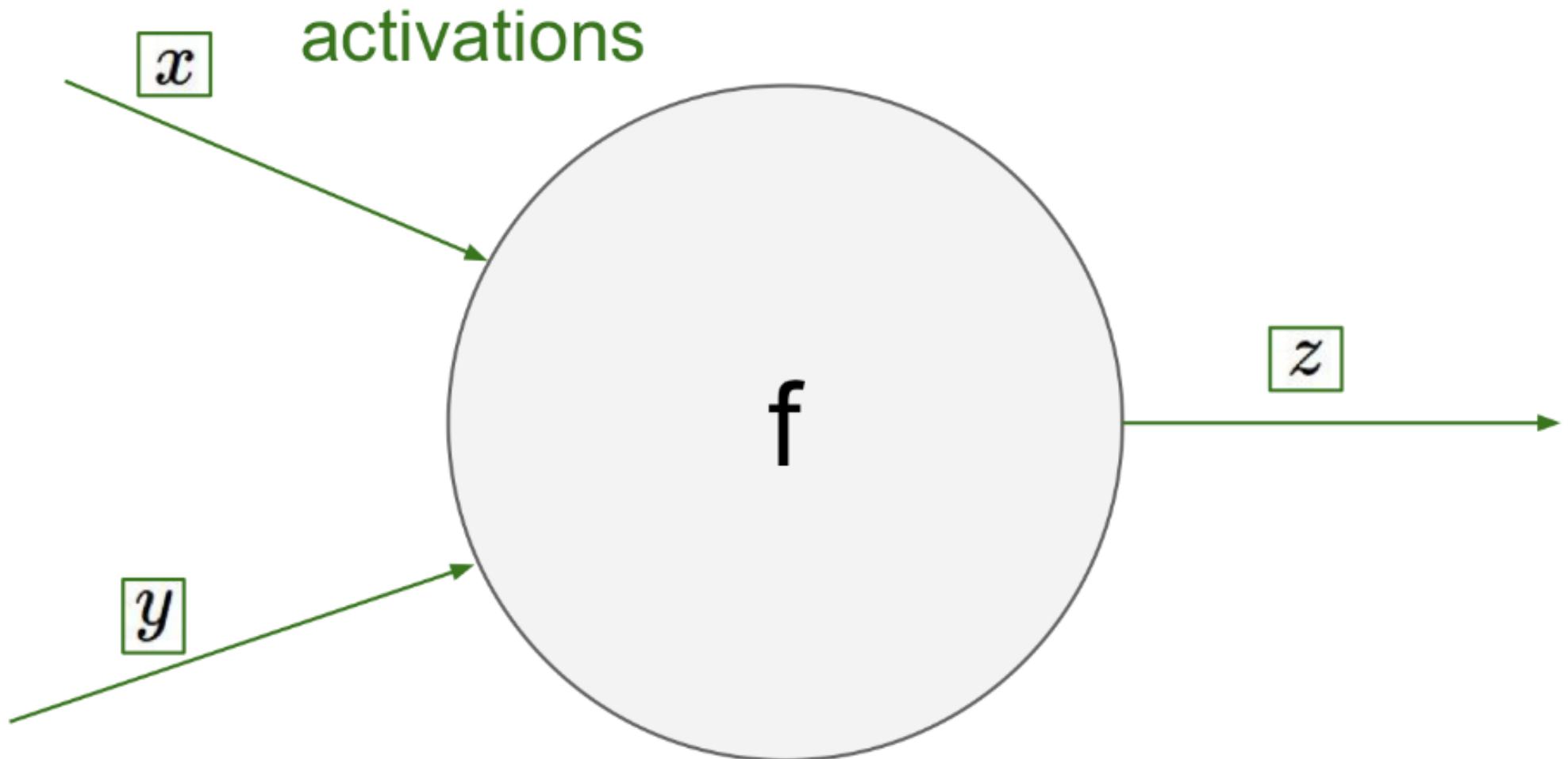
BUT NEEDS THE HESSIAN

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

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

BACKPROPAGATION

[AT THE NEURON LEVEL]



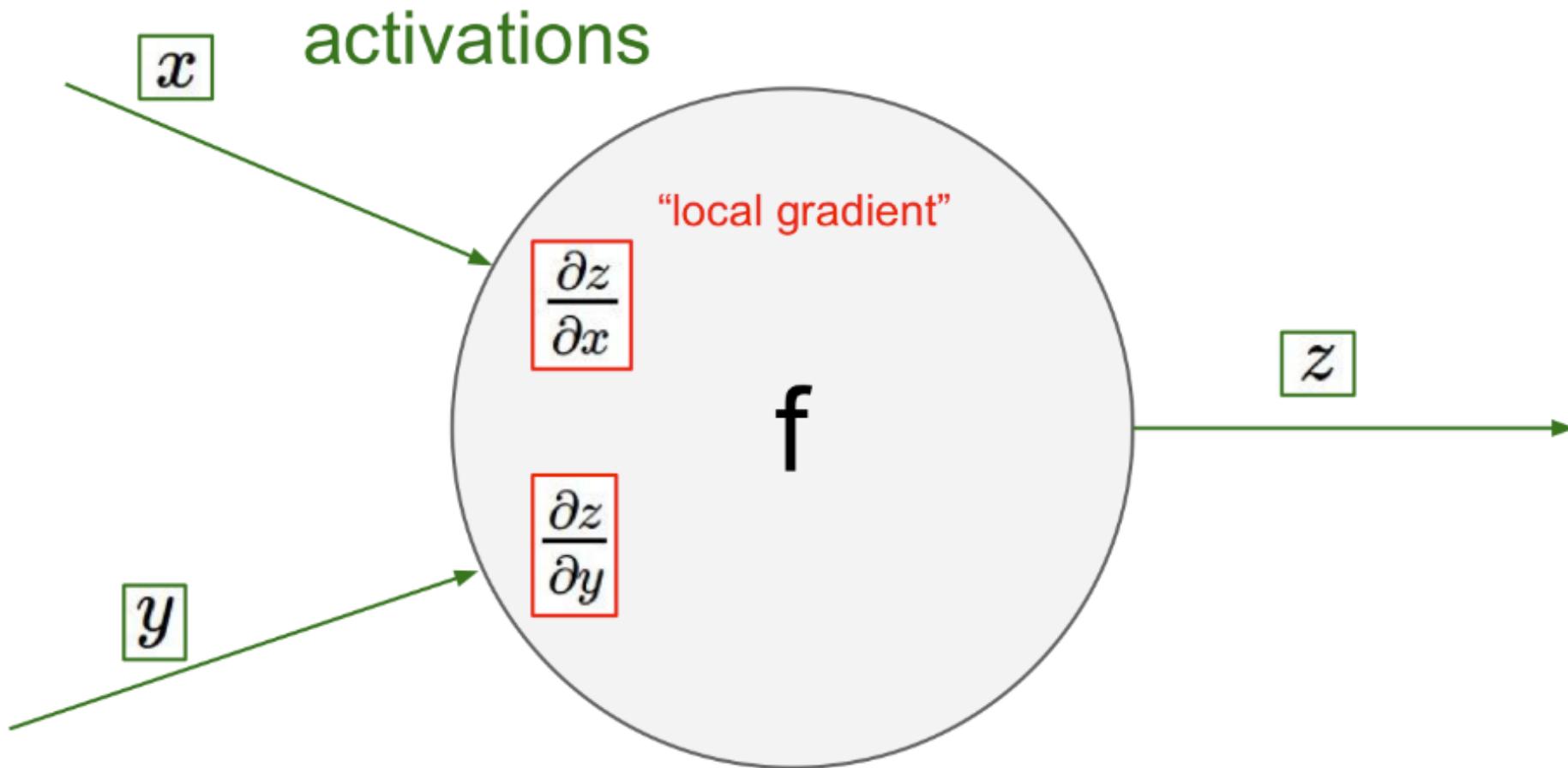
$$z = g[W_1 \times x + W_2 \times y]$$

LOCALLY, I KNOW HOW TO COMPUTE THE DERIVATIVE

Credit: A. Karpathy

BACKPROPAGATION

[AT THE NEURON LEVEL]

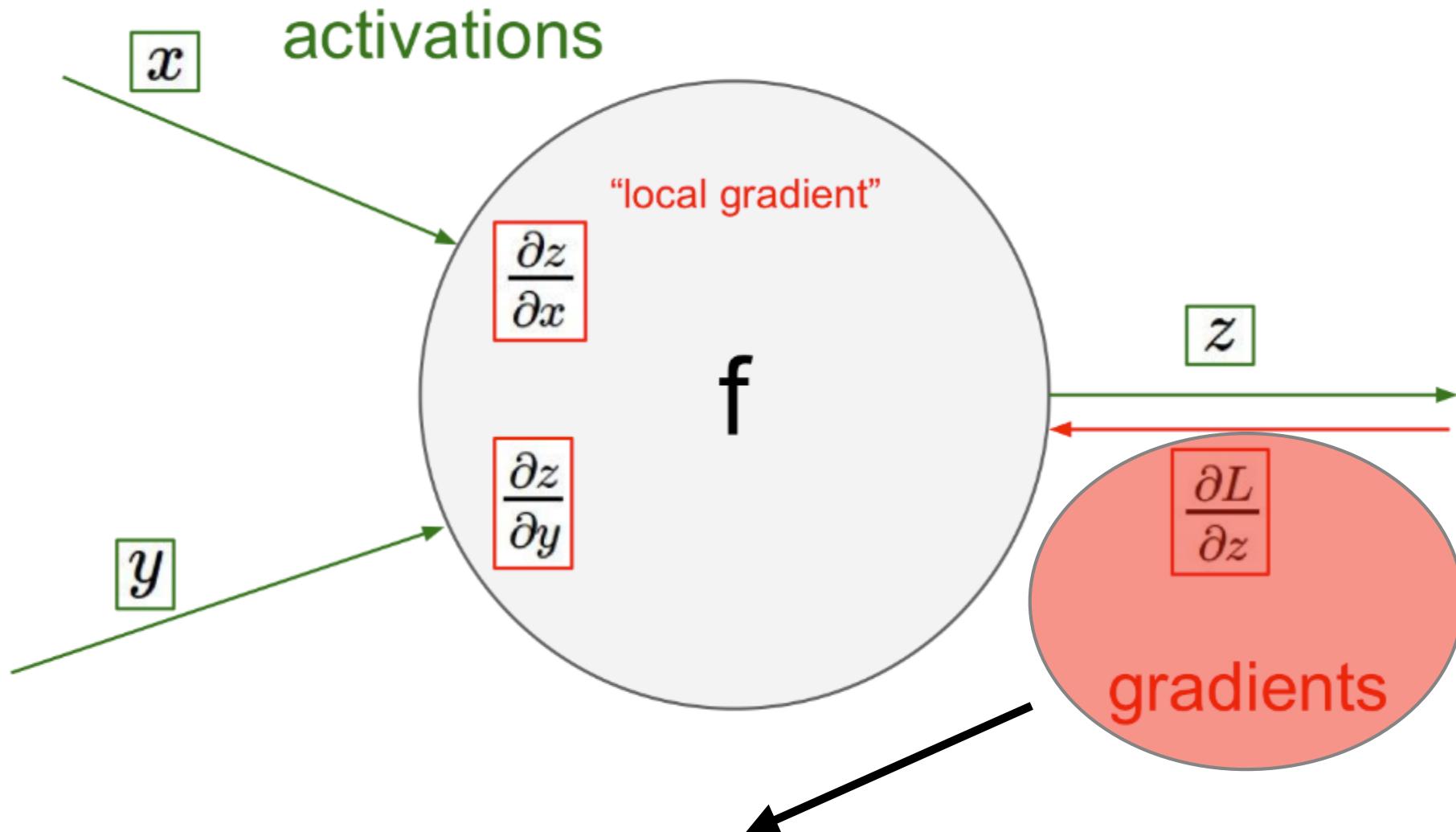


$$z = g[W_1 \times x + W_2 \times y]$$

LOCALLY, I KNOW HOW TO COMPUTE THE DERIVATIVE

BACKPROPAGATION

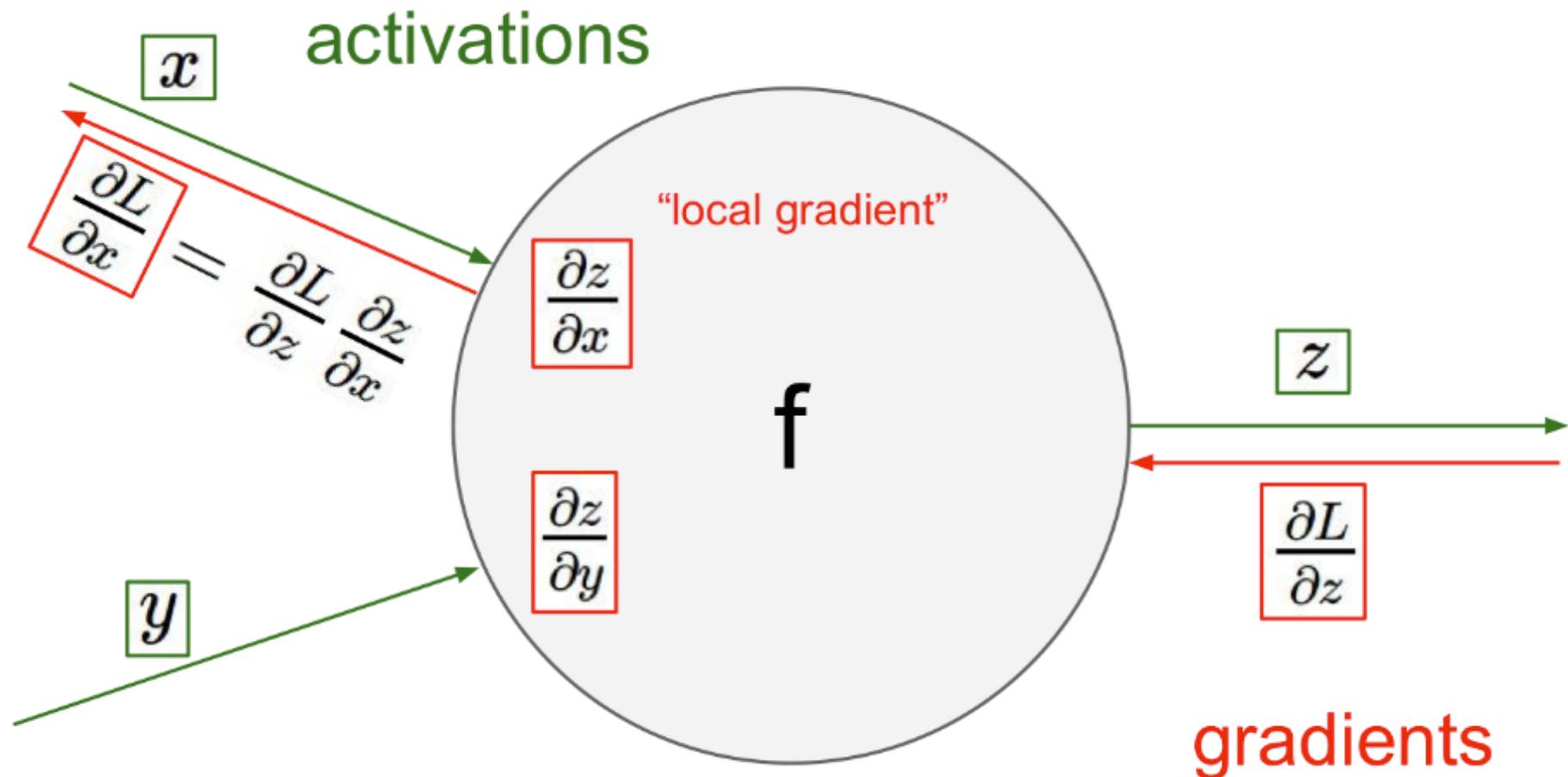
[AT THE NEURON LEVEL]



IF THE GRADIENT FROM THE PREVIOUS PERCEPTRON
IS KNOWN

BACKPROPAGATION

[AT THE NEURON LEVEL]

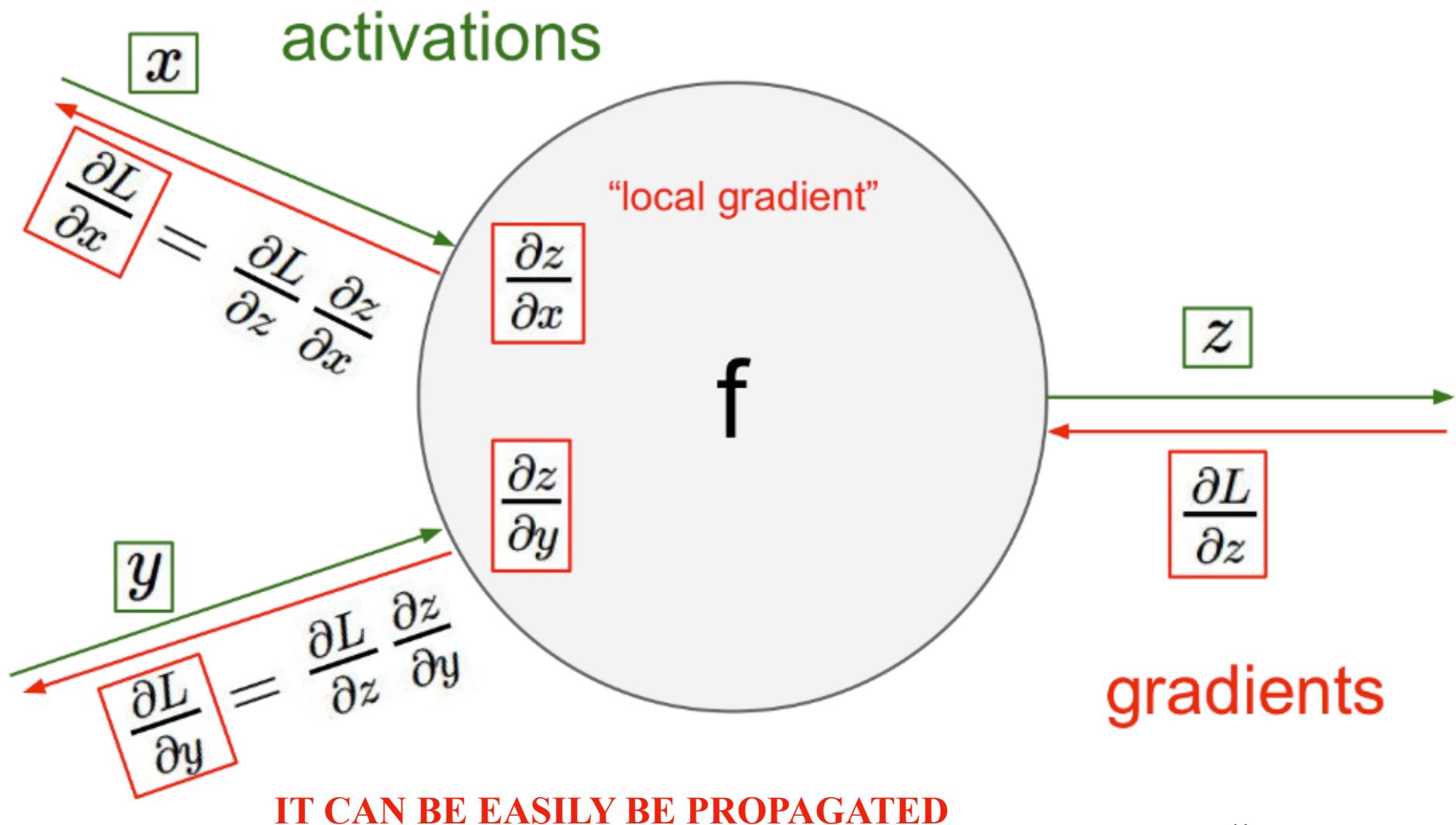


IT CAN BE EASILY BE PROPAGATED

Credit: A. Karpathy

BACKPROPAGATION

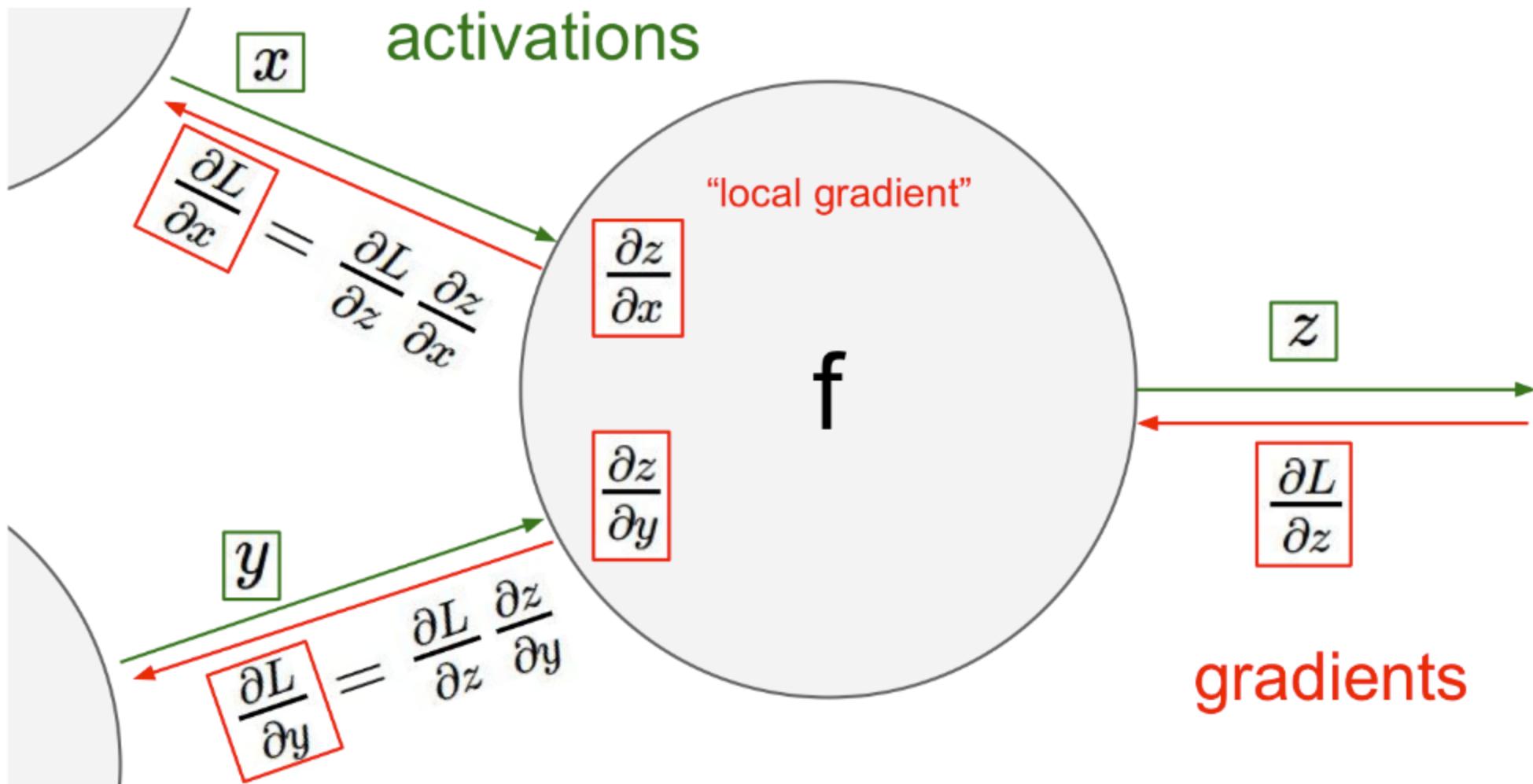
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

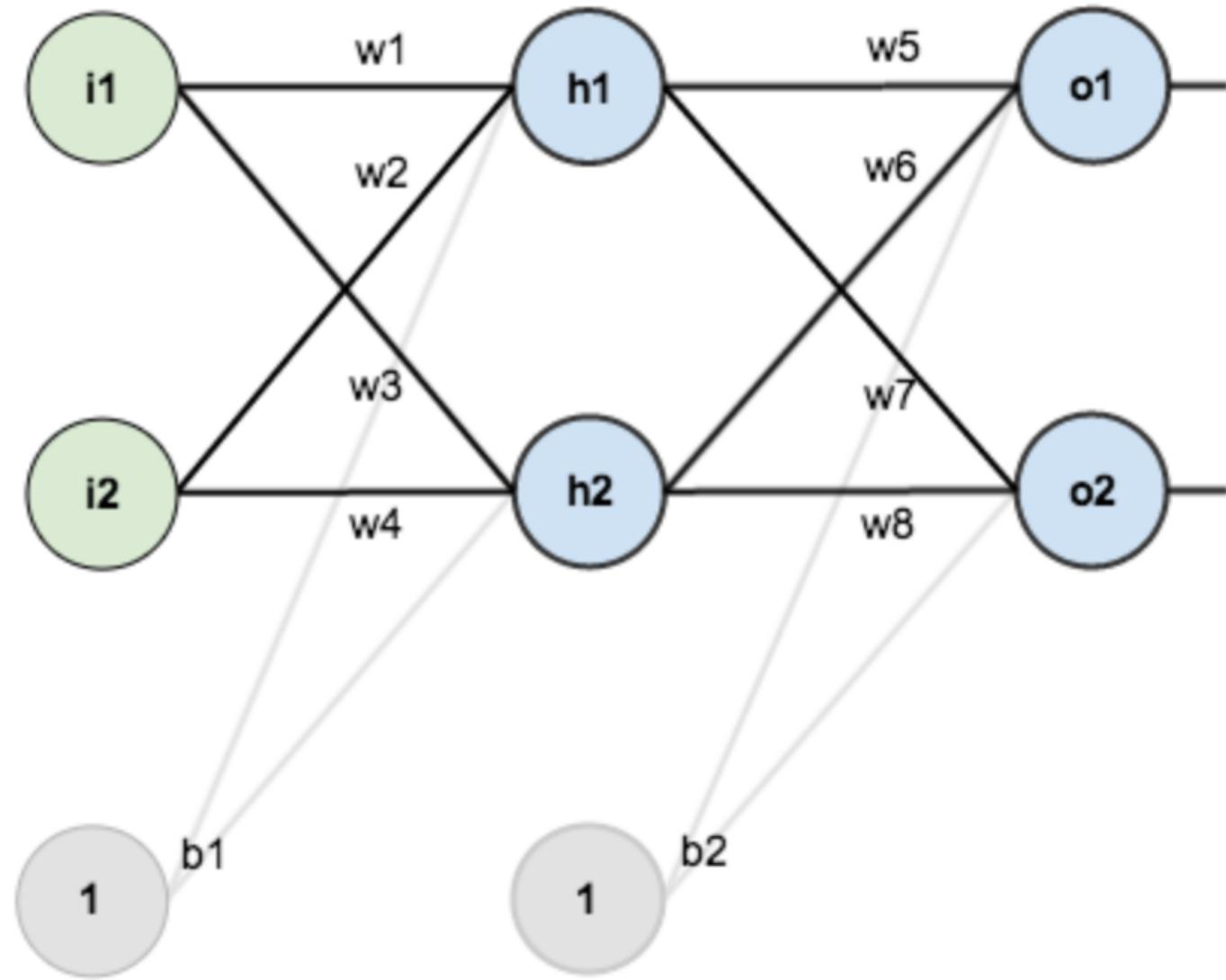
[AT THE NEURON LEVEL]



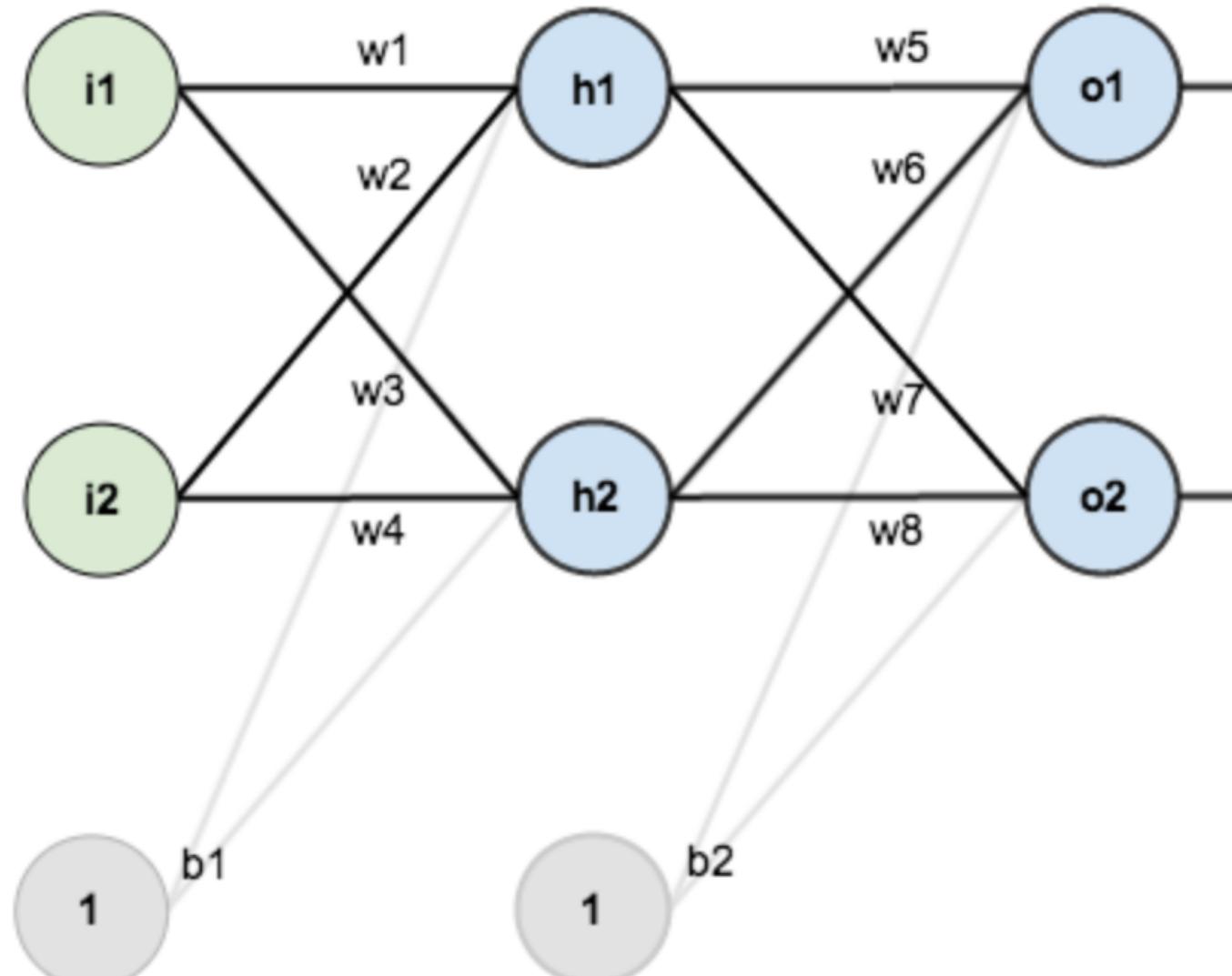
IT CAN BE EASILY BE PROPAGATED

Credit: A. Karpathy

LET'S FOLLOW A NETWORK WHILE IT
LEARNS WITH A PRACTICAL EXAMPLE

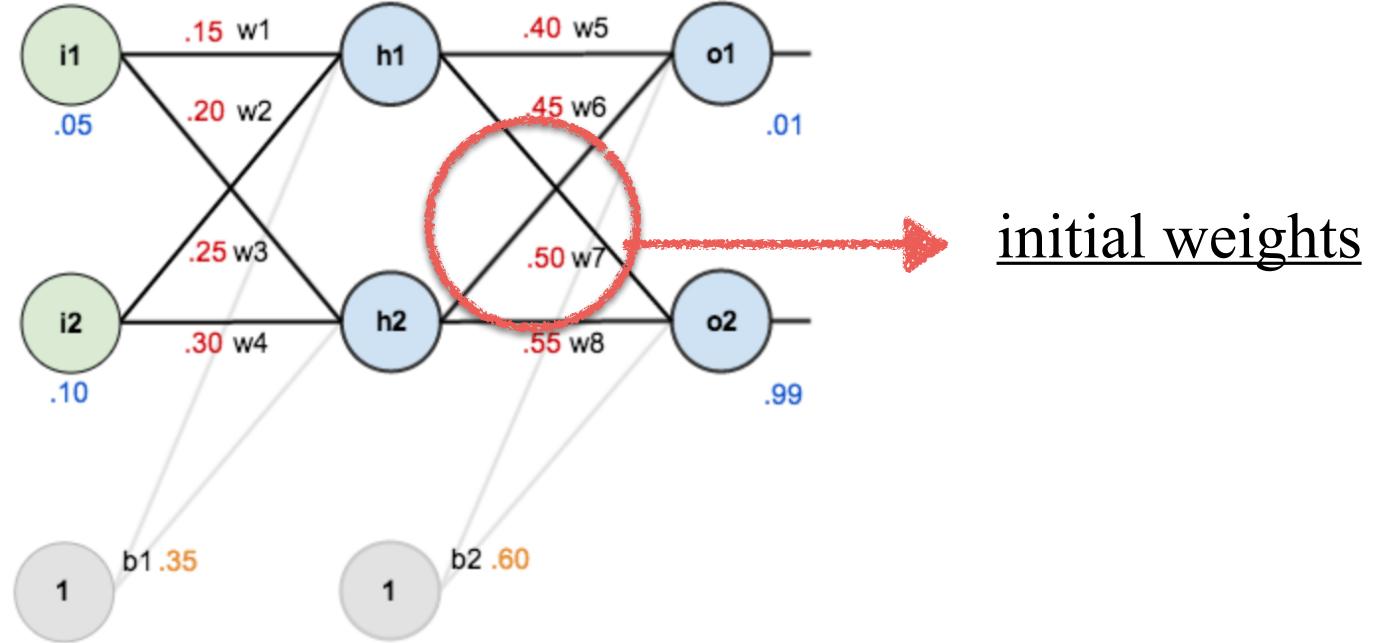


EXAMPLE TAKEN FROM HERE



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

EXAMPLE TAKEN FROM HERE

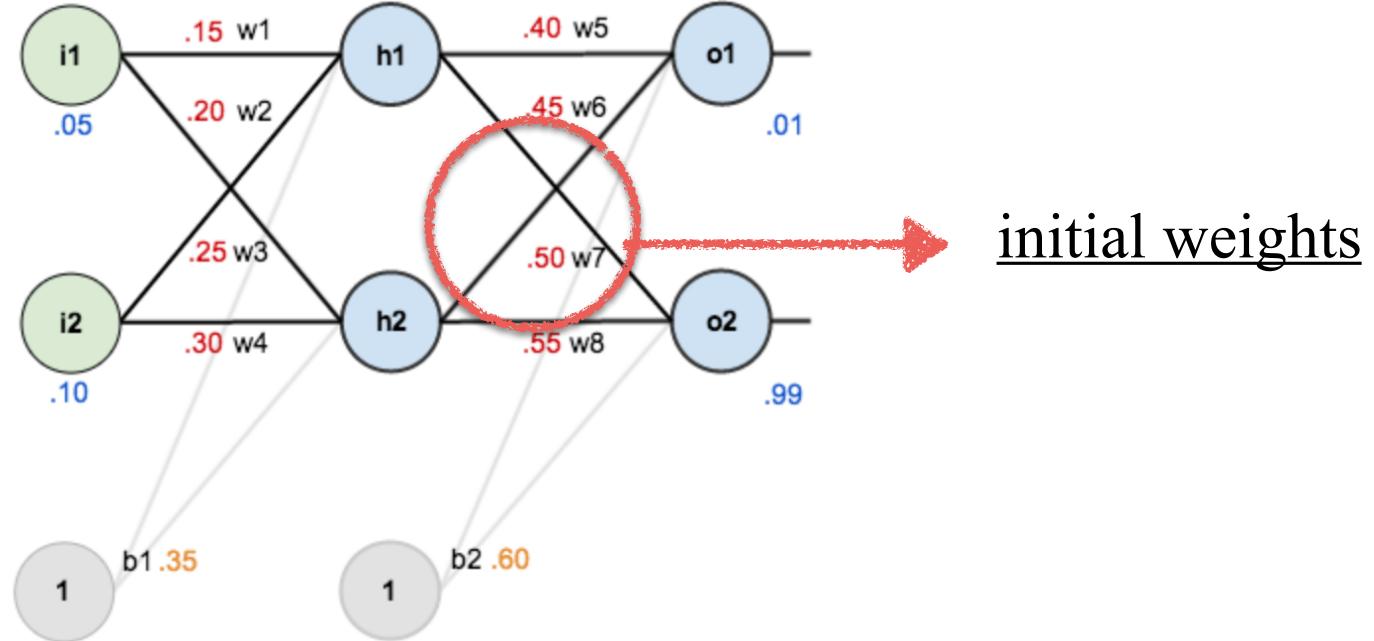


1. THE FORWARD PASS

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

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

[with initial weights]



1. THE FORWARD PASS

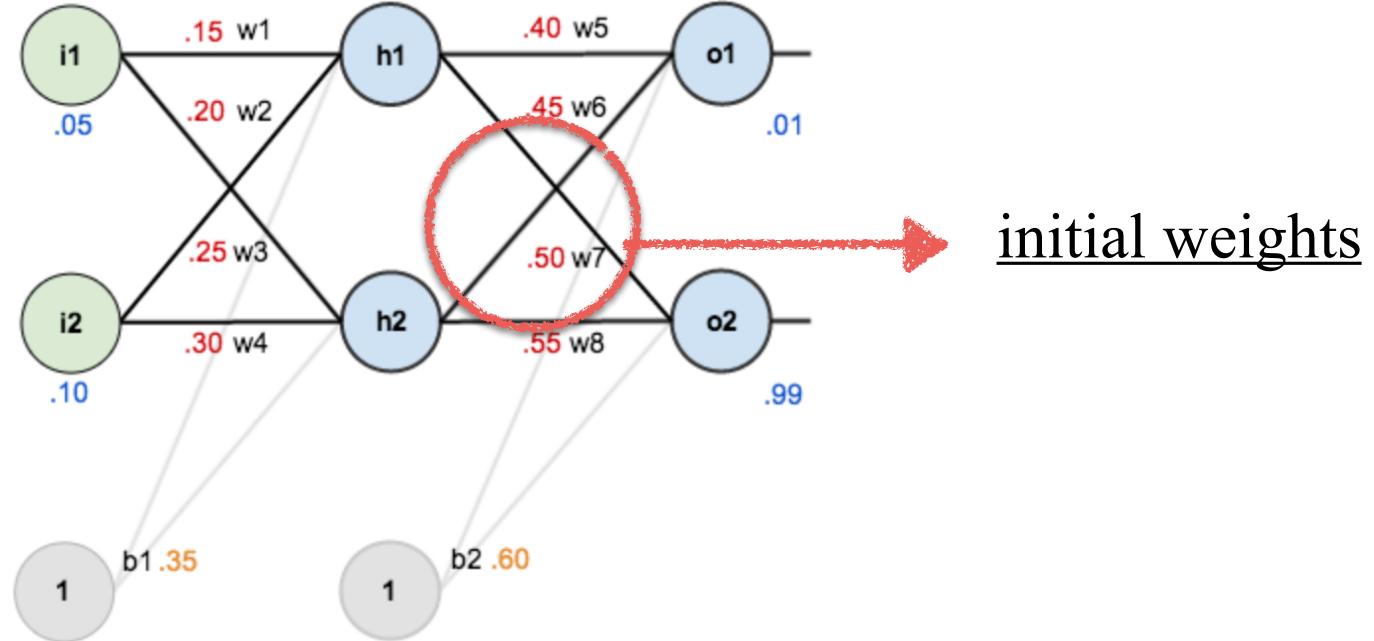
$$in_{h1} = w_1 i_1 + w_2 i_2 + b_1$$

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

[with initial weights]

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

[after the activation function]



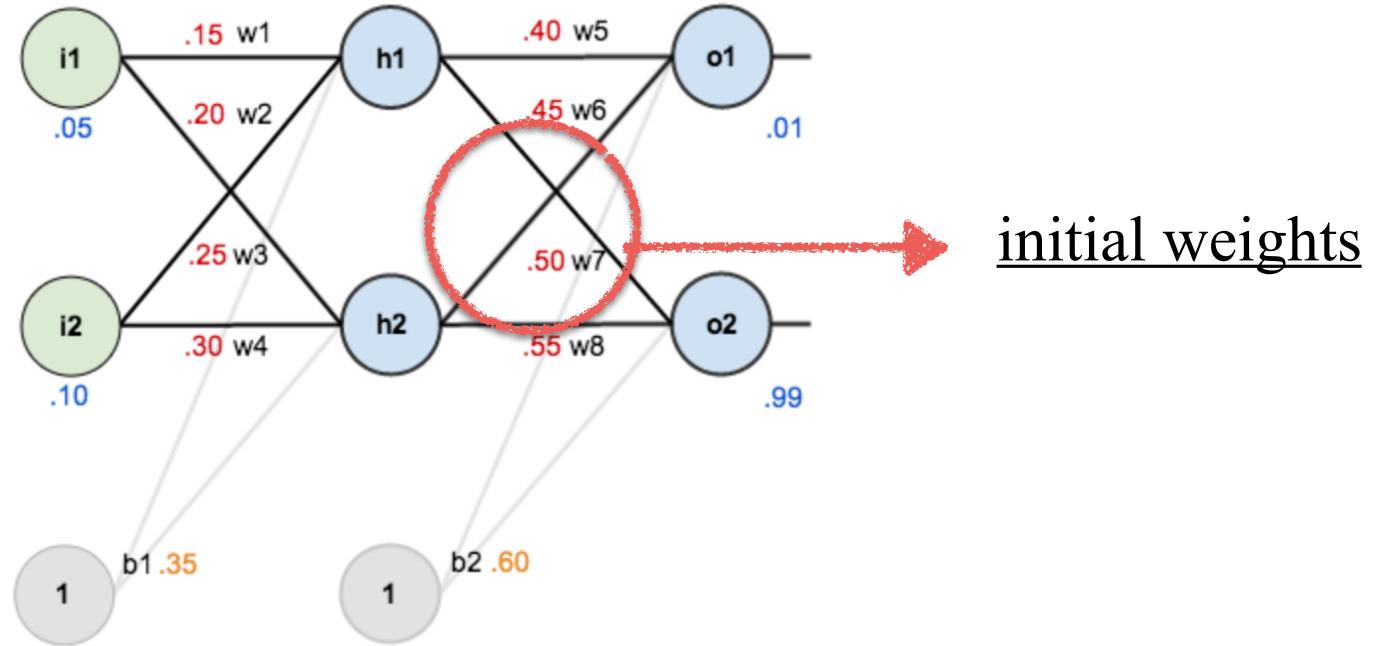
1. THE FORWARD PASS

WE CONTINUE TO o_1

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

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

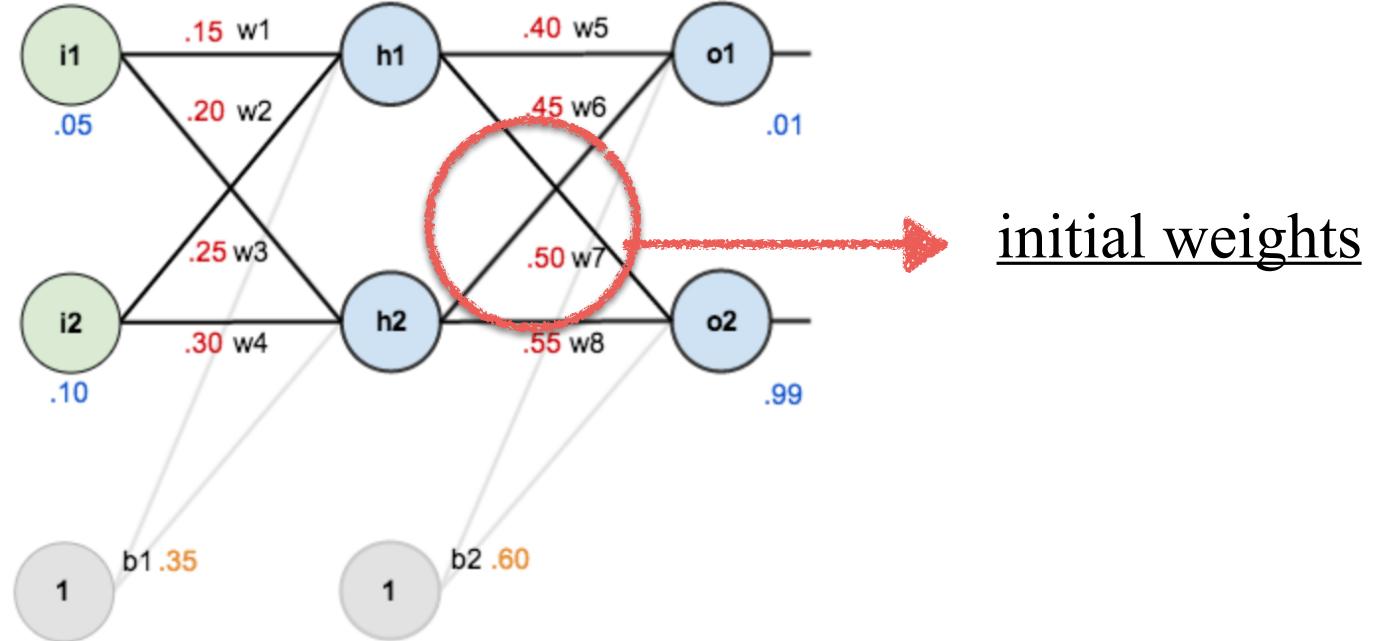
$$out_{o1} = \frac{1}{1 + e^{-1.105}} = 0.751$$



1. THE FORWARD PASS

AND THE SAME FOR o_2

$$out_{o2} = 0.7729$$

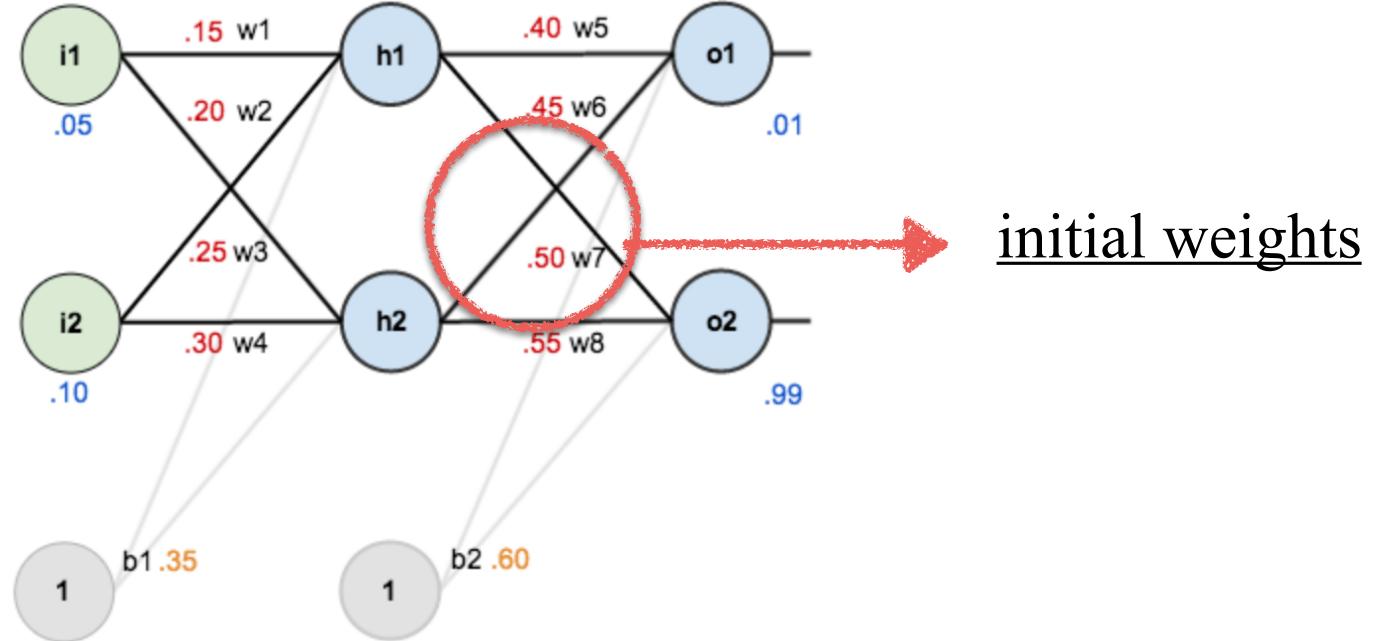


2. THE LOSS FUNCTION

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

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

$$L_{o2} = 0.023$$



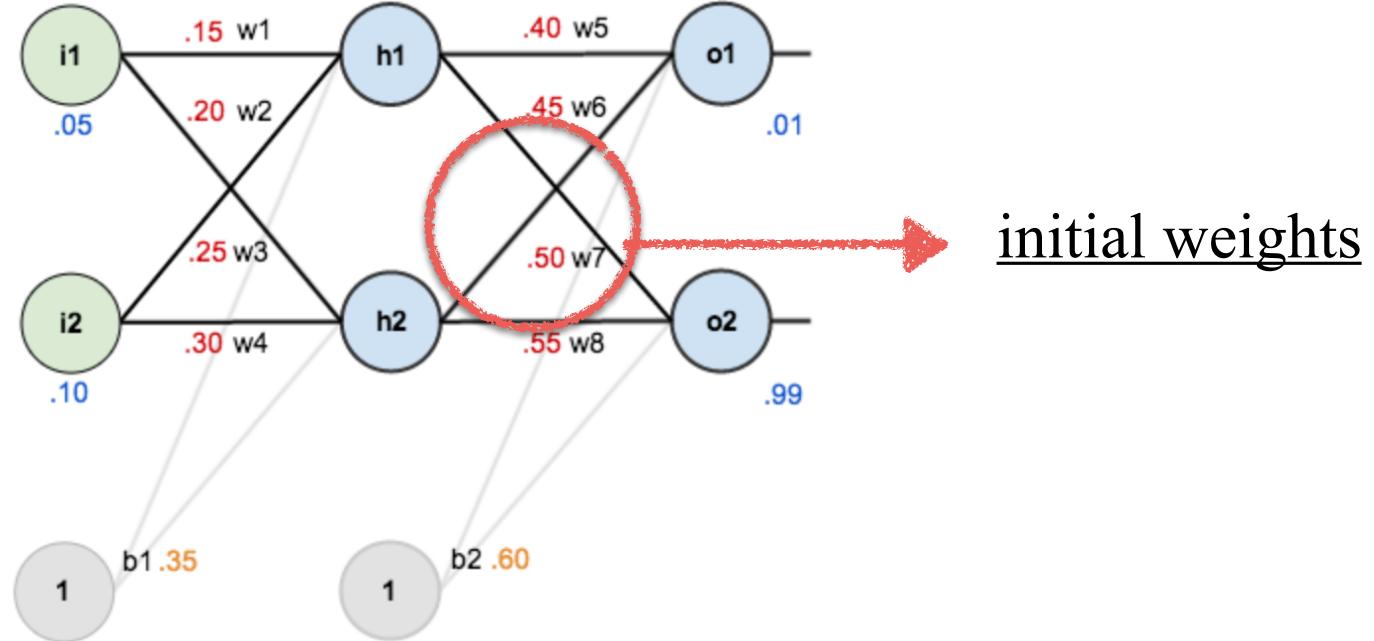
2. THE LOSS FUNCTION

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

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

$$L_{o2} = 0.023$$

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



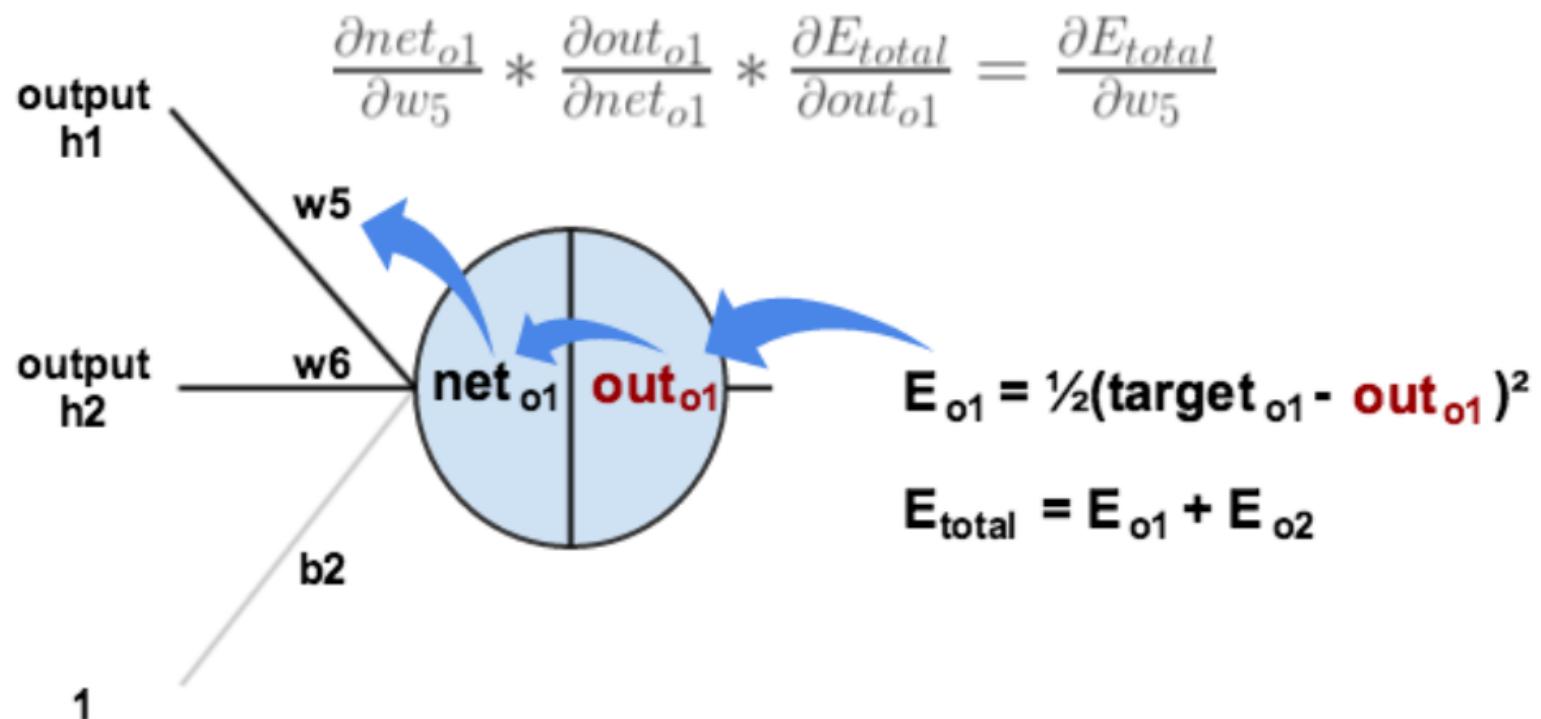
3. THE BACKWARD PASS

FOR w_5

WE WANT:

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

[gradient of loss function]



3. THE BACKWARD PASS

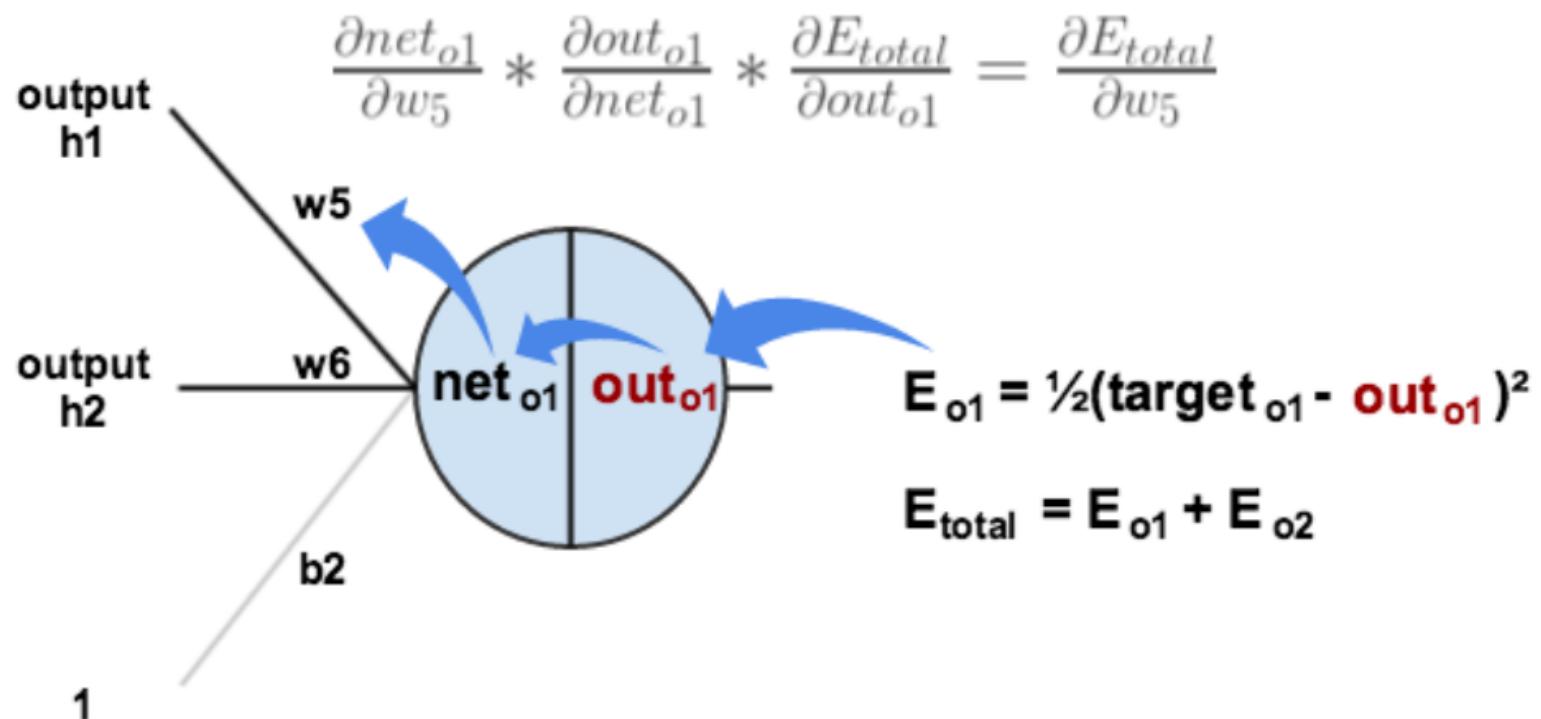
FOR W5

WE WANT:

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

WE APPLY THE CHAIN RULE:

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

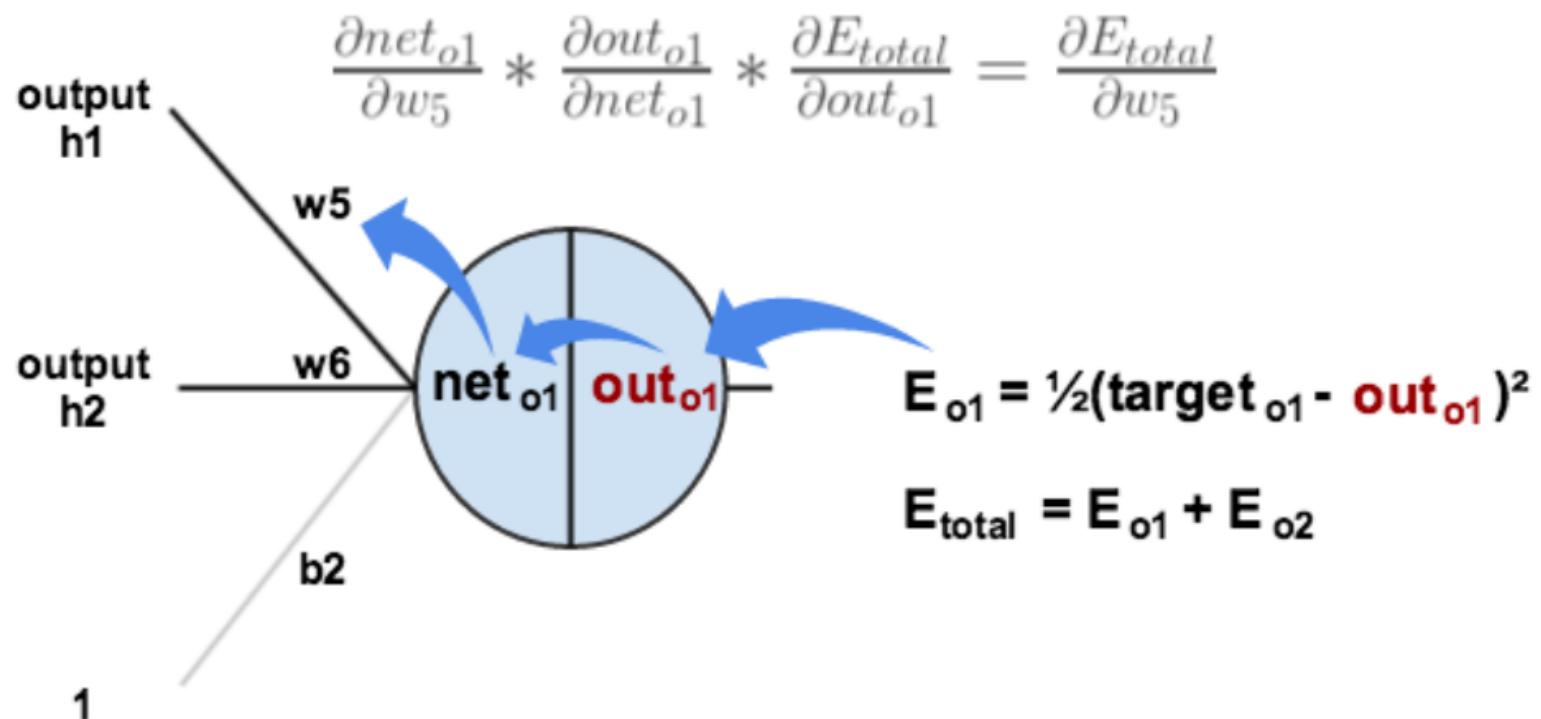


3. THE BACKWARD PASS

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

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

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

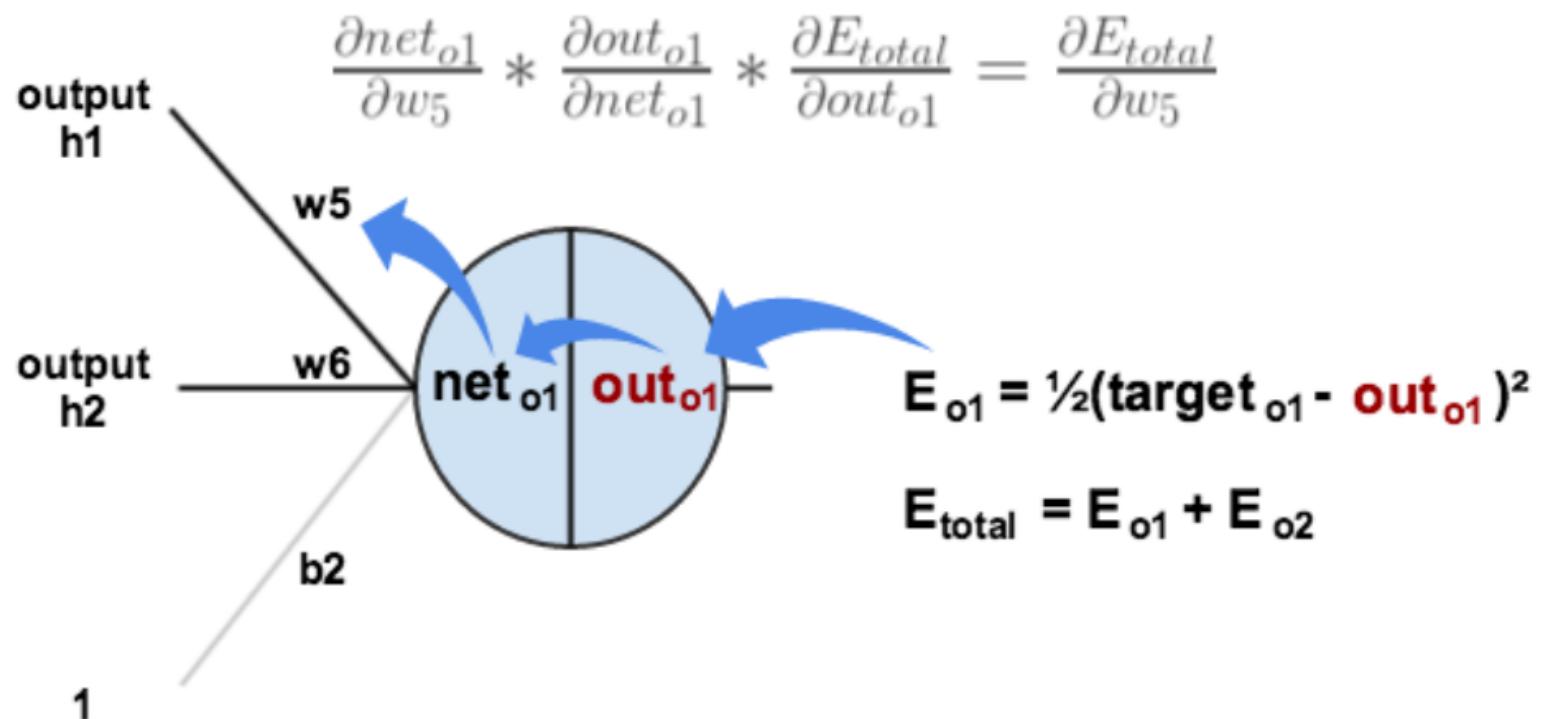


3. THE BACKWARD PASS

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

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

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

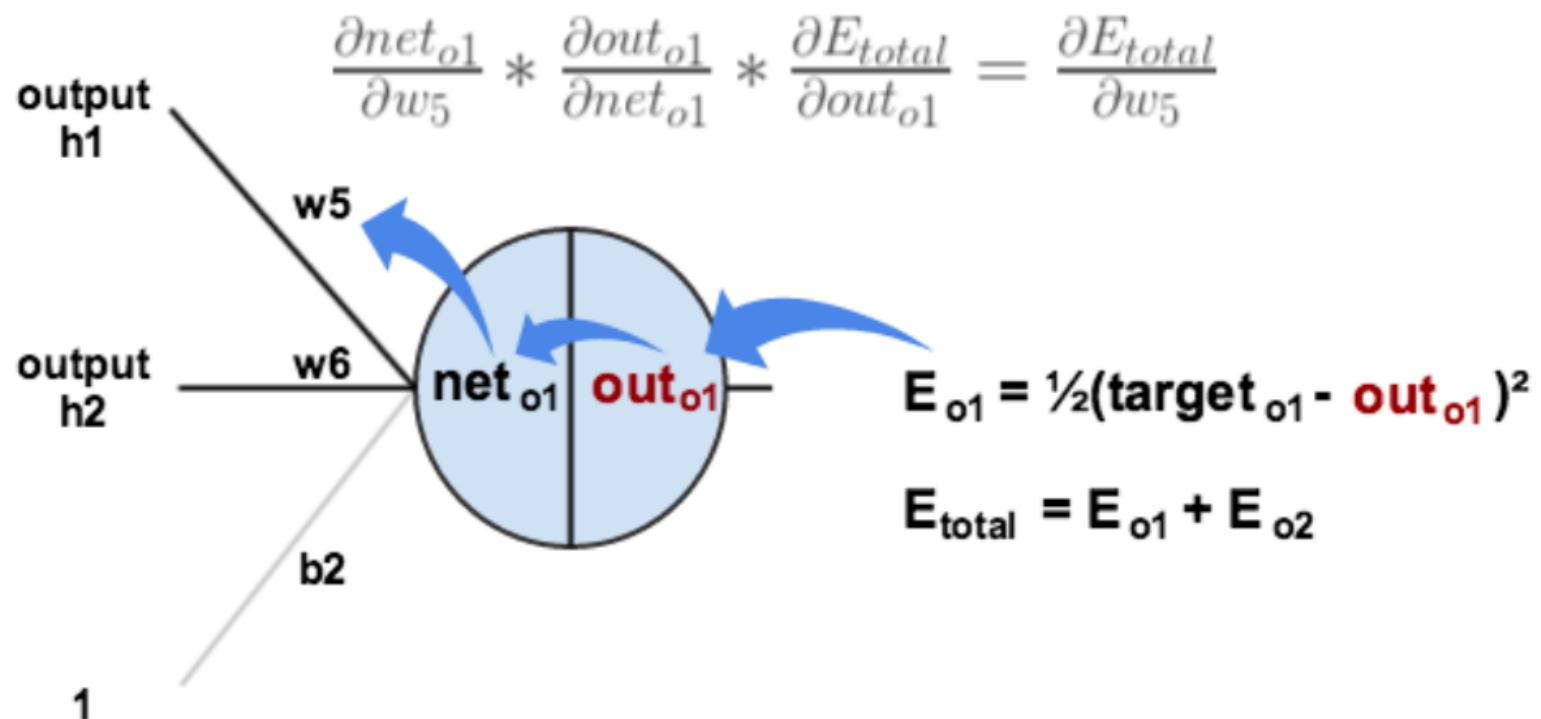


3. THE BACKWARD PASS

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

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

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

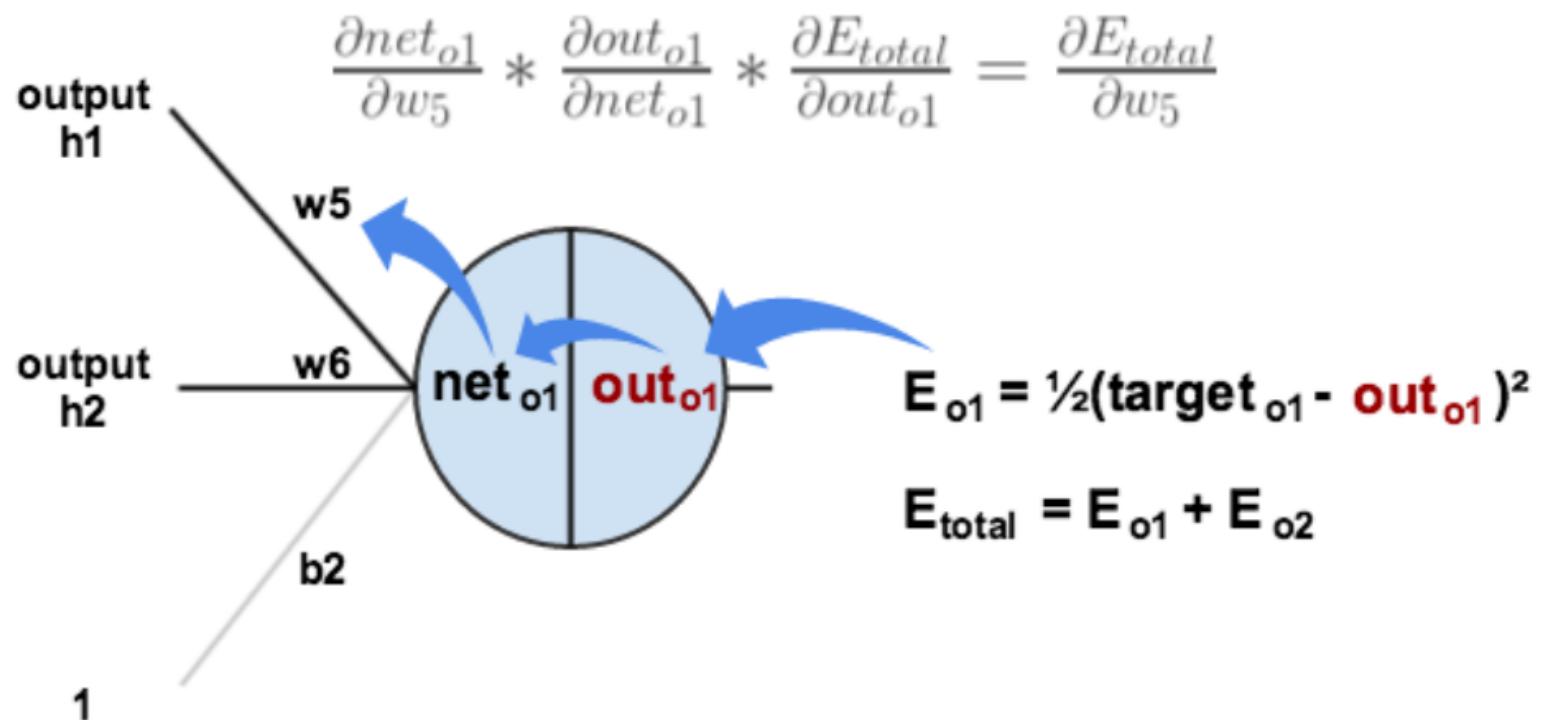


3. THE BACKWARD PASS

ALL TOGETHER:

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

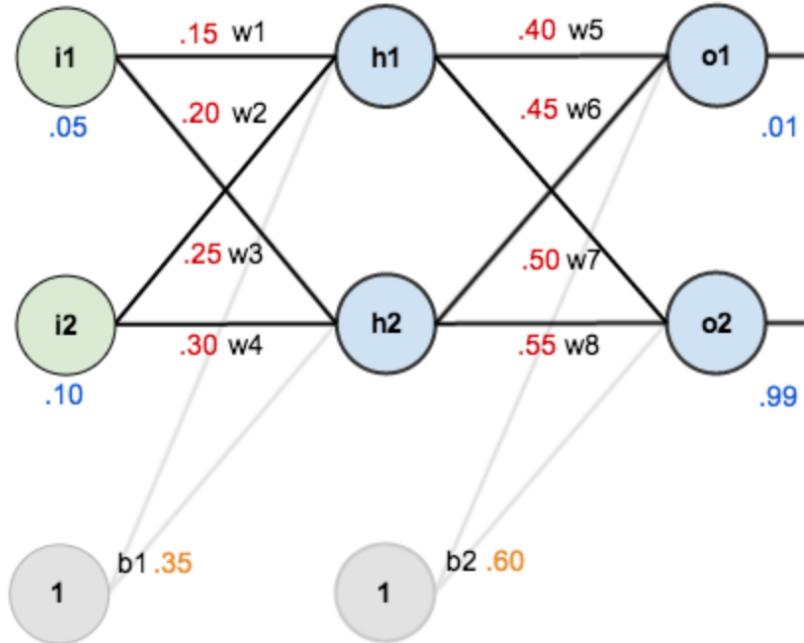
$$\frac{\partial L_{total}}{\partial w_5} = 0.741 \times 0.186 \times 0.593 = 0.082$$



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**

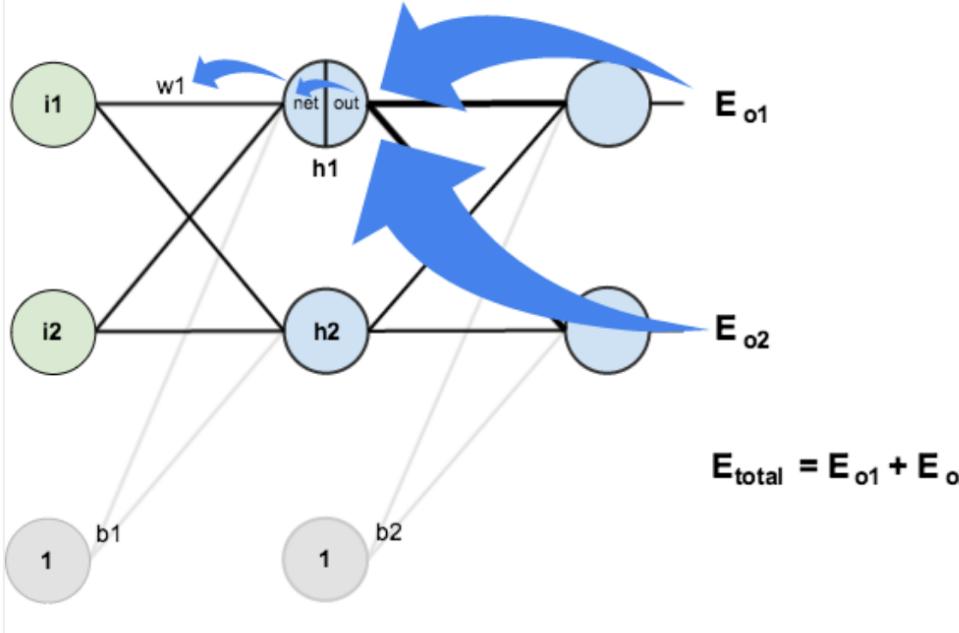
$$w_6^{t+1} = 0.408$$

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

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

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

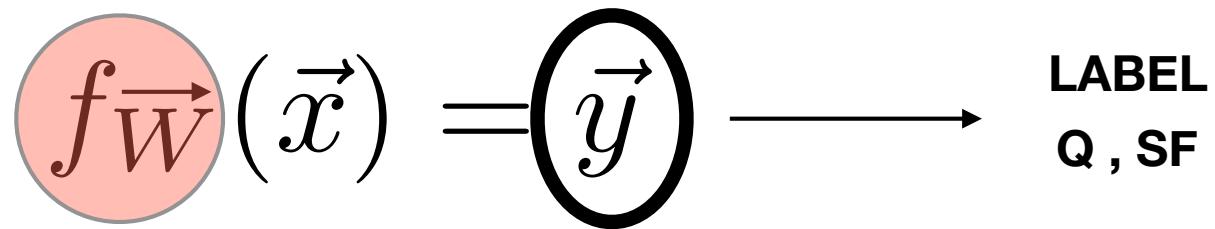
$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



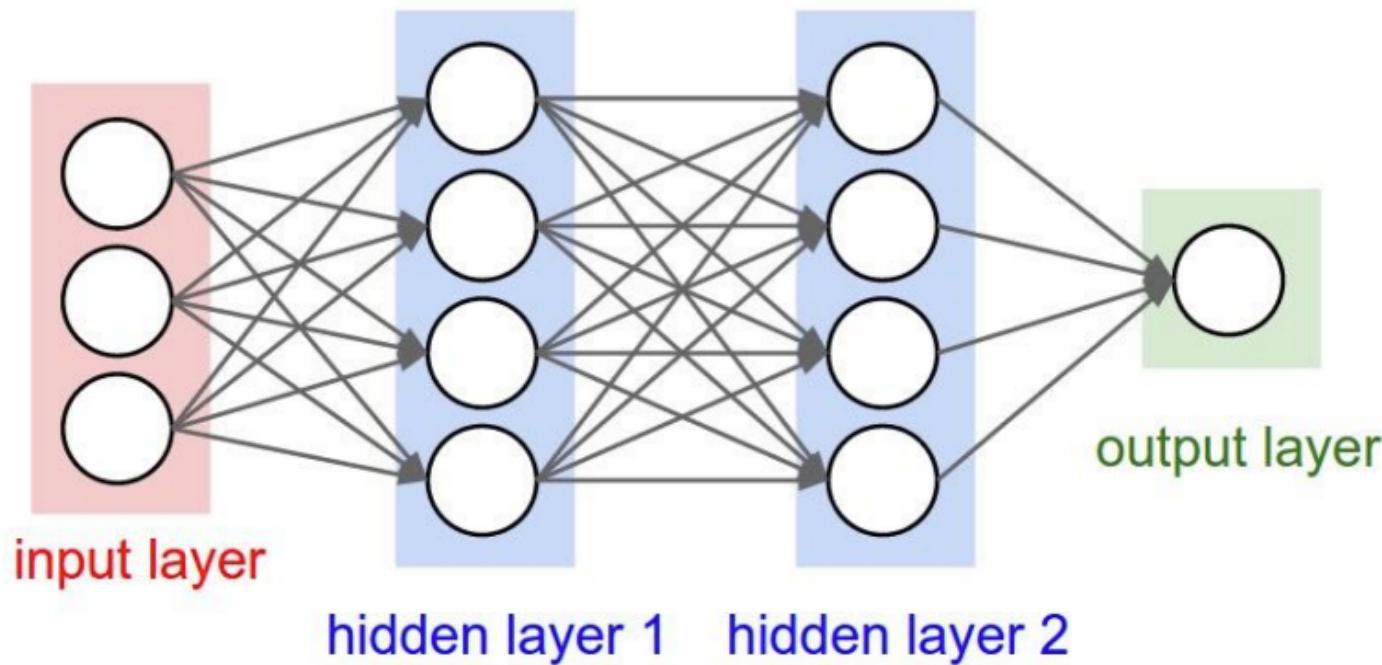
AND BACK-PROPAGATED TO THE HIDDEN LAYERS

VISUALIZE SIMPLE NETWORK LEARNING

**"CLASSICAL"
MACHINE LEARNING**



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



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

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

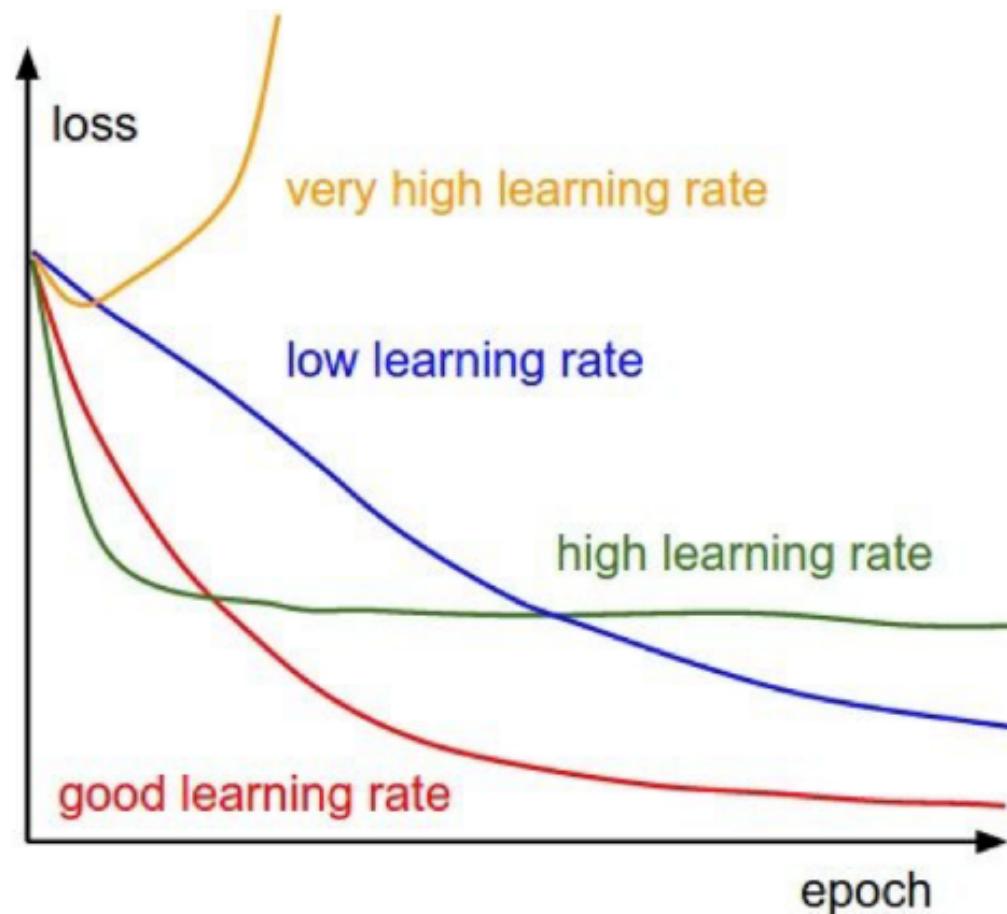
ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT
EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING
THE STEEPEST DESCENT

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

IN KERAS:

Adam

[source]

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

Adam optimizer.

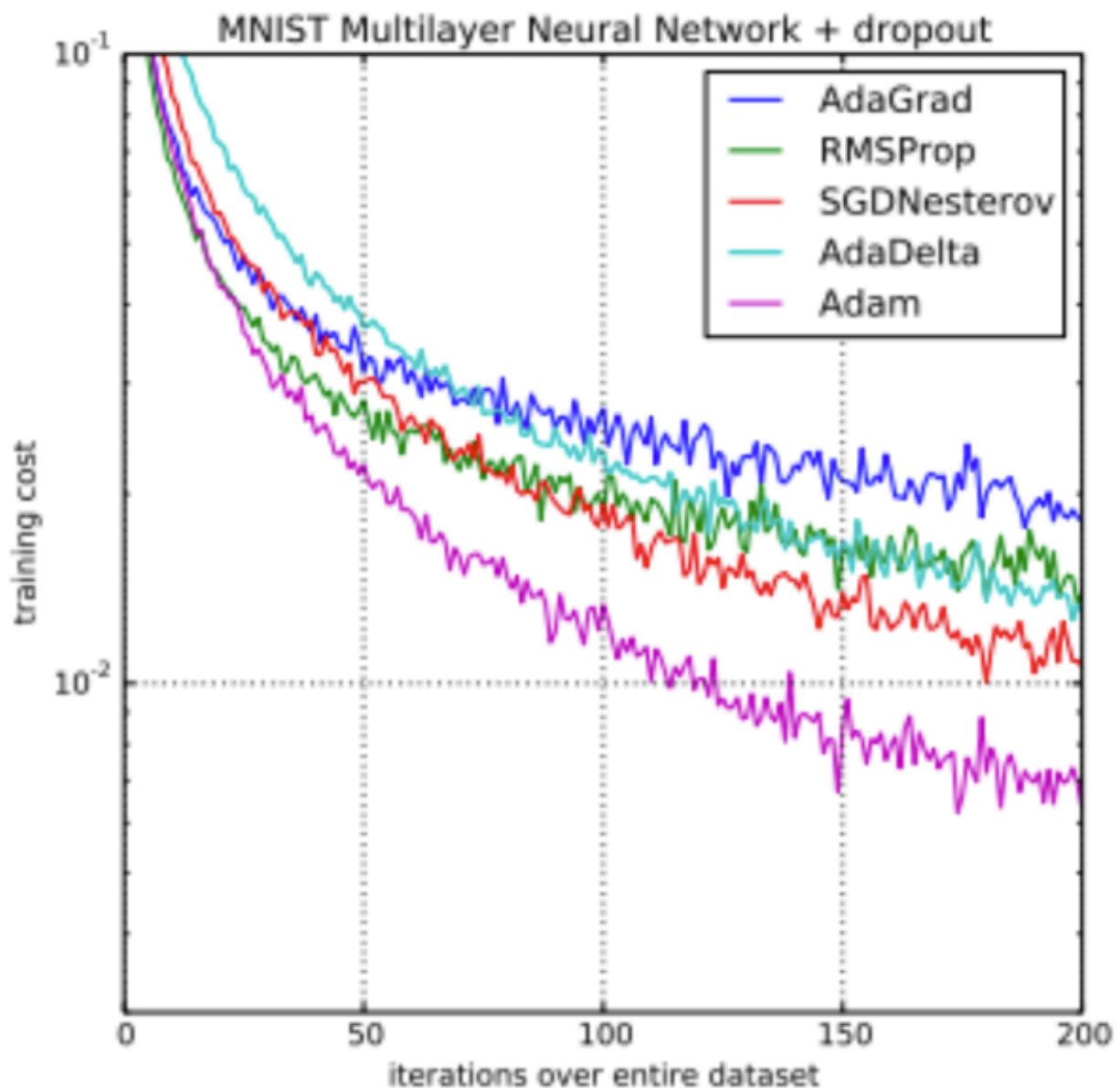
Default parameters follow those provided in the original paper.

Arguments

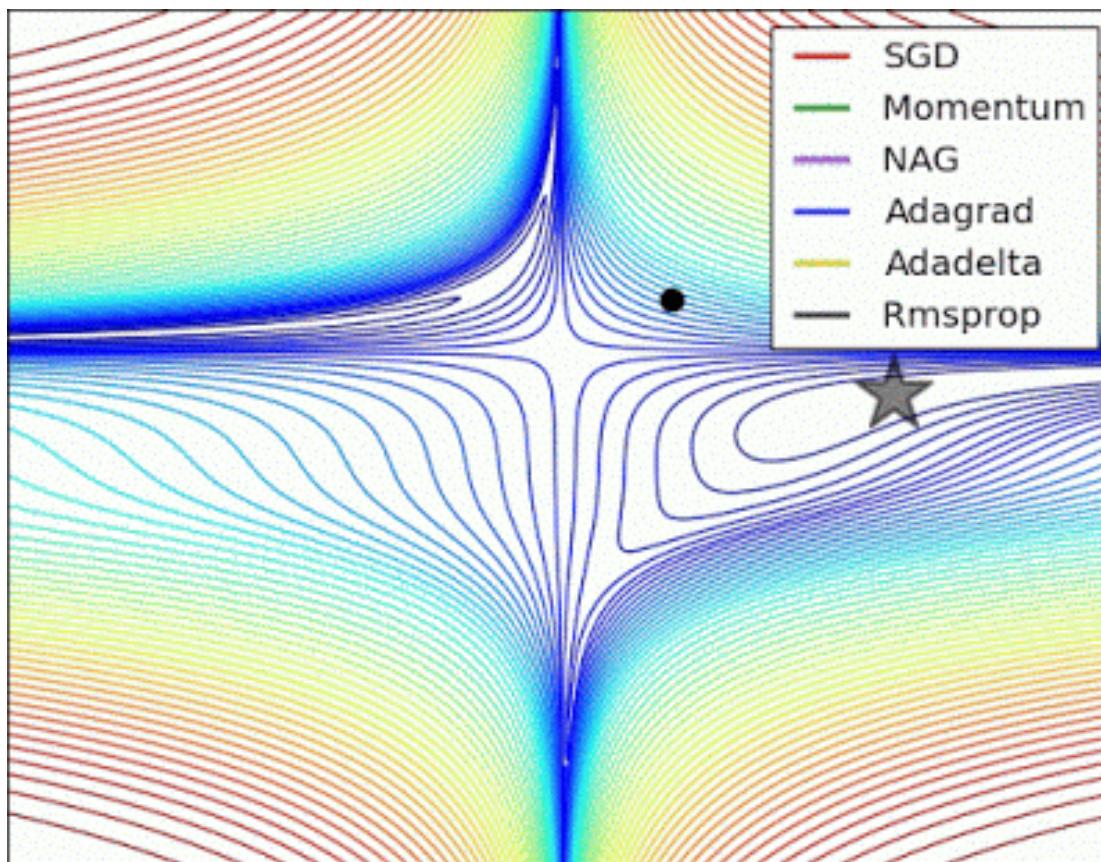
- **lr**: float ≥ 0 . Learning rate.
- **beta_1**: float, $0 < \text{beta} < 1$. Generally close to 1.
- **beta_2**: float, $0 < \text{beta} < 1$. Generally close to 1.
- **epsilon**: float ≥ 0 . Fuzz factor. If `None`, defaults to `K.epsilon()`.
- **decay**: float ≥ 0 . Learning rate decay over each update.
- **amsgrad**: boolean. Whether to apply the AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and Beyond".

References

- [Adam - A Method for Stochastic Optimization](#)
- [On the Convergence of Adam and Beyond](#)



Credit



Credit

BATCH GRADIENT DESCENT

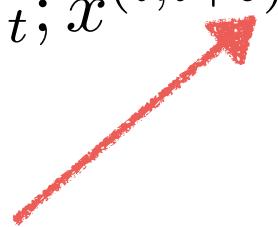
LOCAL MINIMA CAN ALSO BE AVOIDED BY COMPUTING THE
GRADIENT IN SMALL BATCHES INSTEAD OF OVER THE FULL
DATASET

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

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



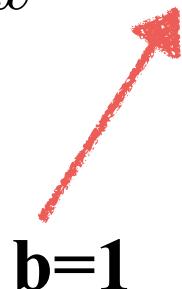
THE GRADIENT IS COMPUTED OVER A BATCH OF SIZE B

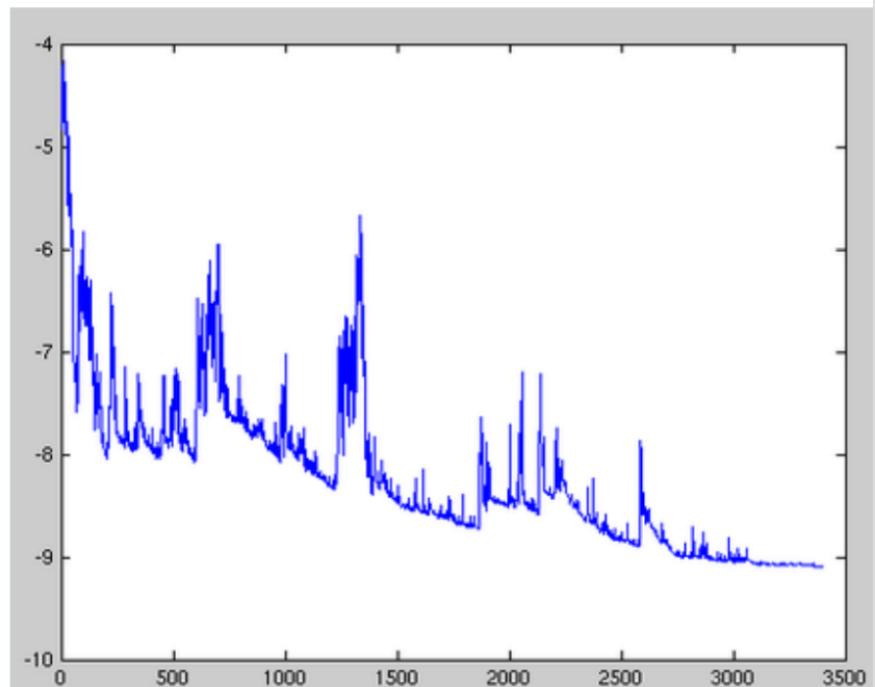
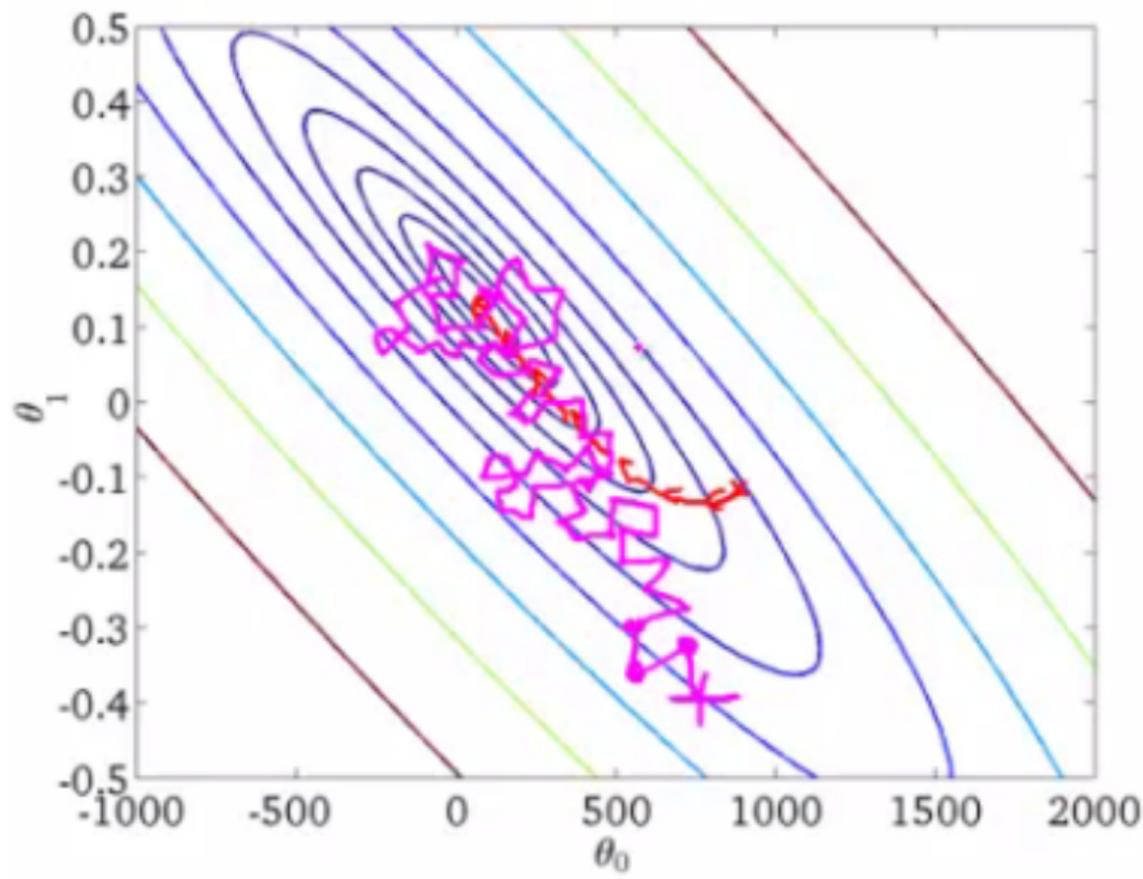
STOCHASTIC GRADIENT DESCENT

THE EXTREME CASE IS TO COMPUTE THE GRADIENT ON EVERY TRAINING EXAMPLE.

STOCHASTIC GRADIENT DESCENT

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





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

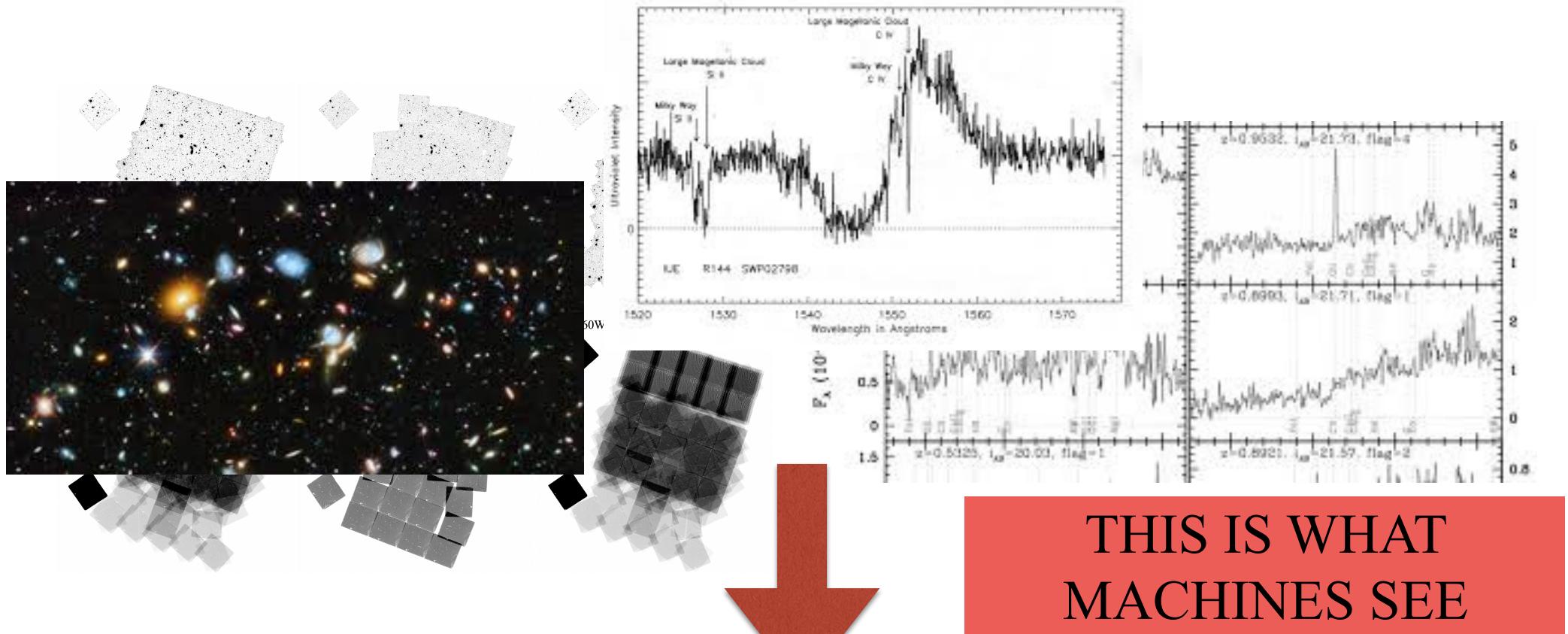
Credit

CAN WE GO DEEP NOW?

CAN WE GO DEEP NOW?

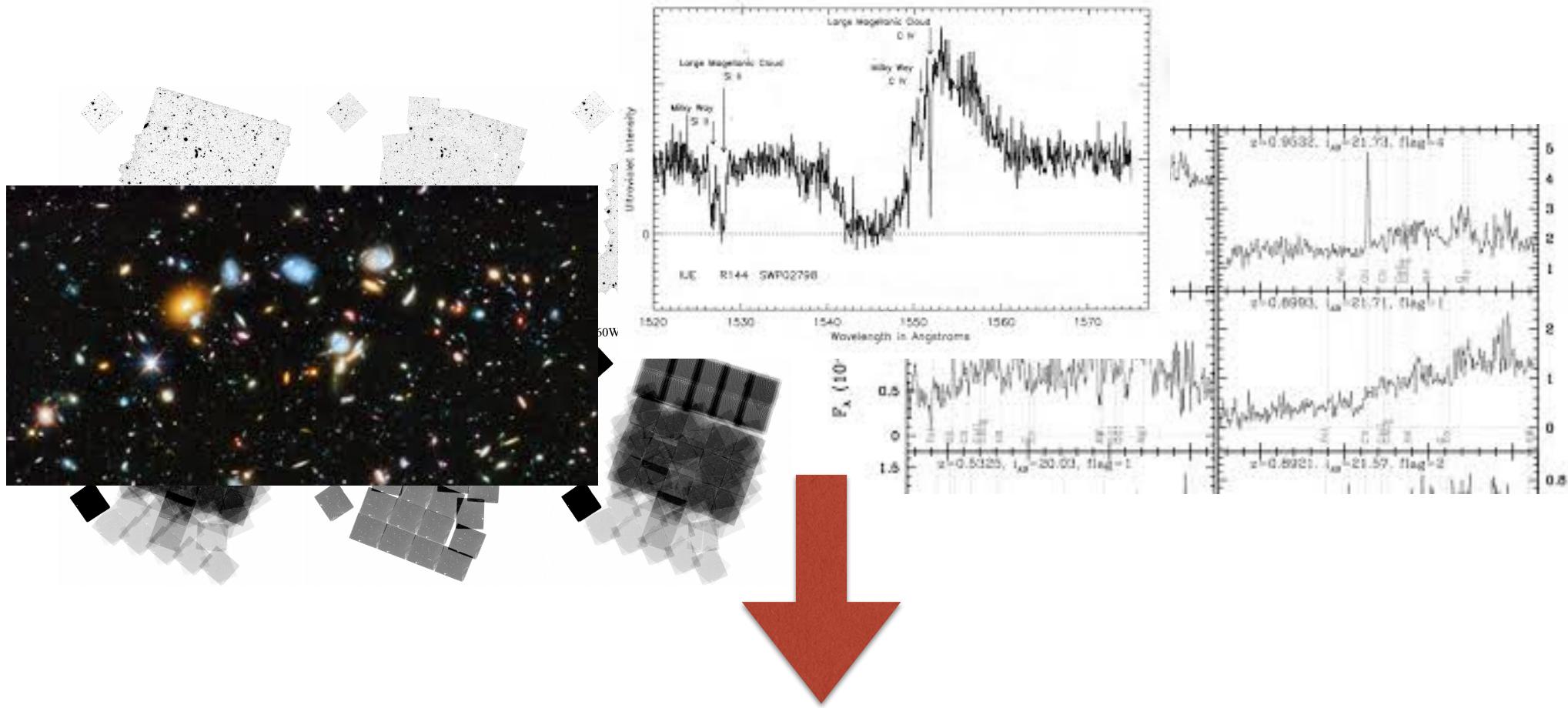
ALMOST THERE...LET'S THINK FOR A
MOMENT ABOUT WHAT WE PUT AS
INPUT...

What do we put as input?



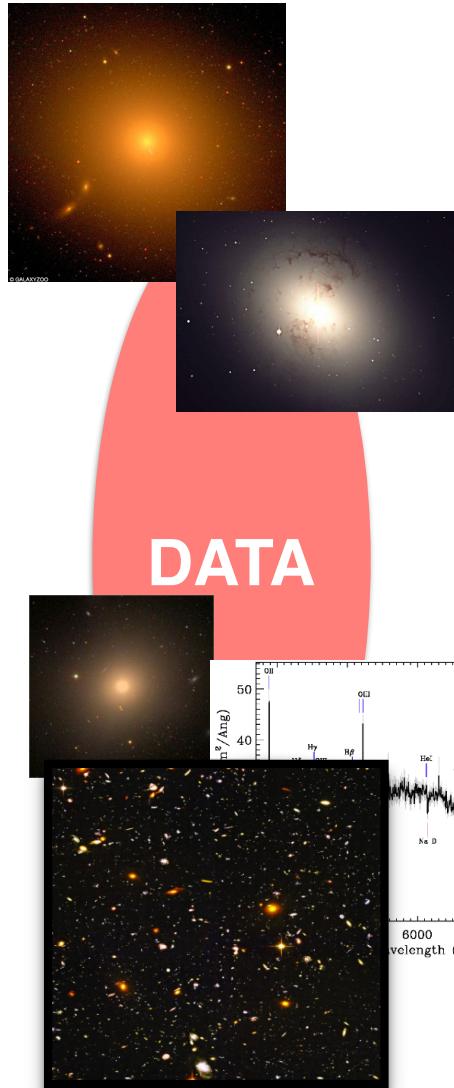
THIS IS WHAT MACHINES SEE

What do we put as input?



PRE-PROCESS DATA TO EXTRACT MEANINGFUL INFORMATION

THIS IS GENERALLY CALLED **FEATURE EXTRACTION**



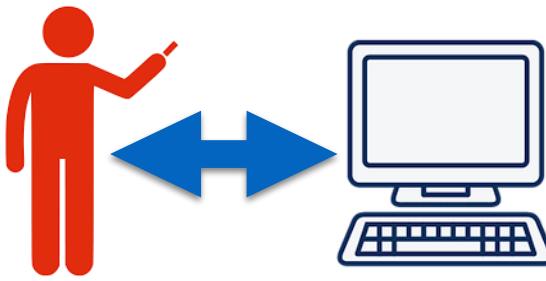
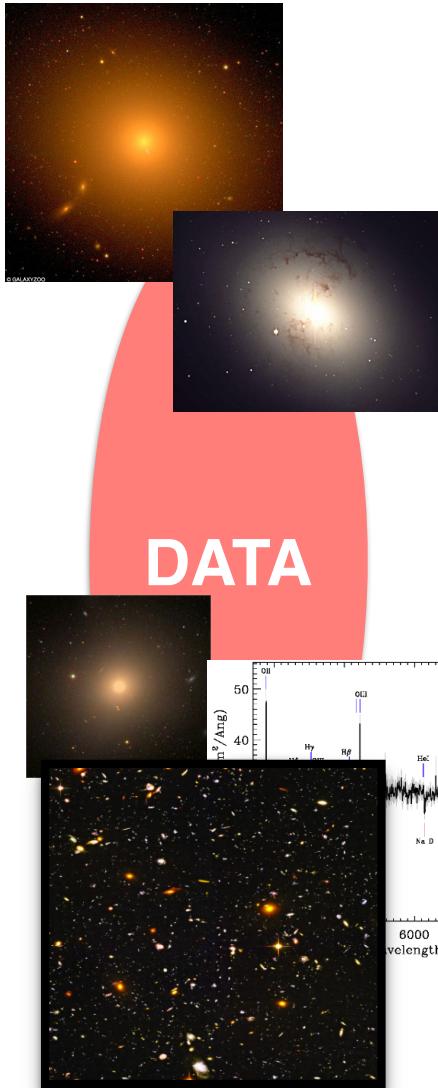
Spiral!

Emission line!

Merger!

Clump!

AGN!



Spiral!

Emission line!

Merger!

Clump!

AGN!

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

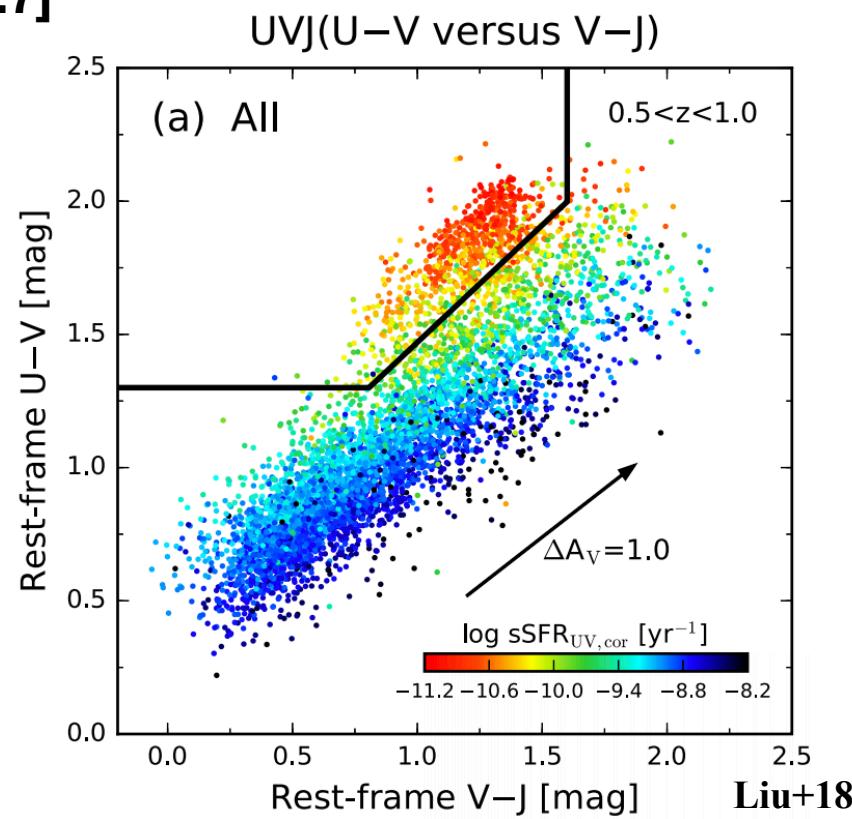
Q(0) , SF(1)

NETWORK FUNCTION

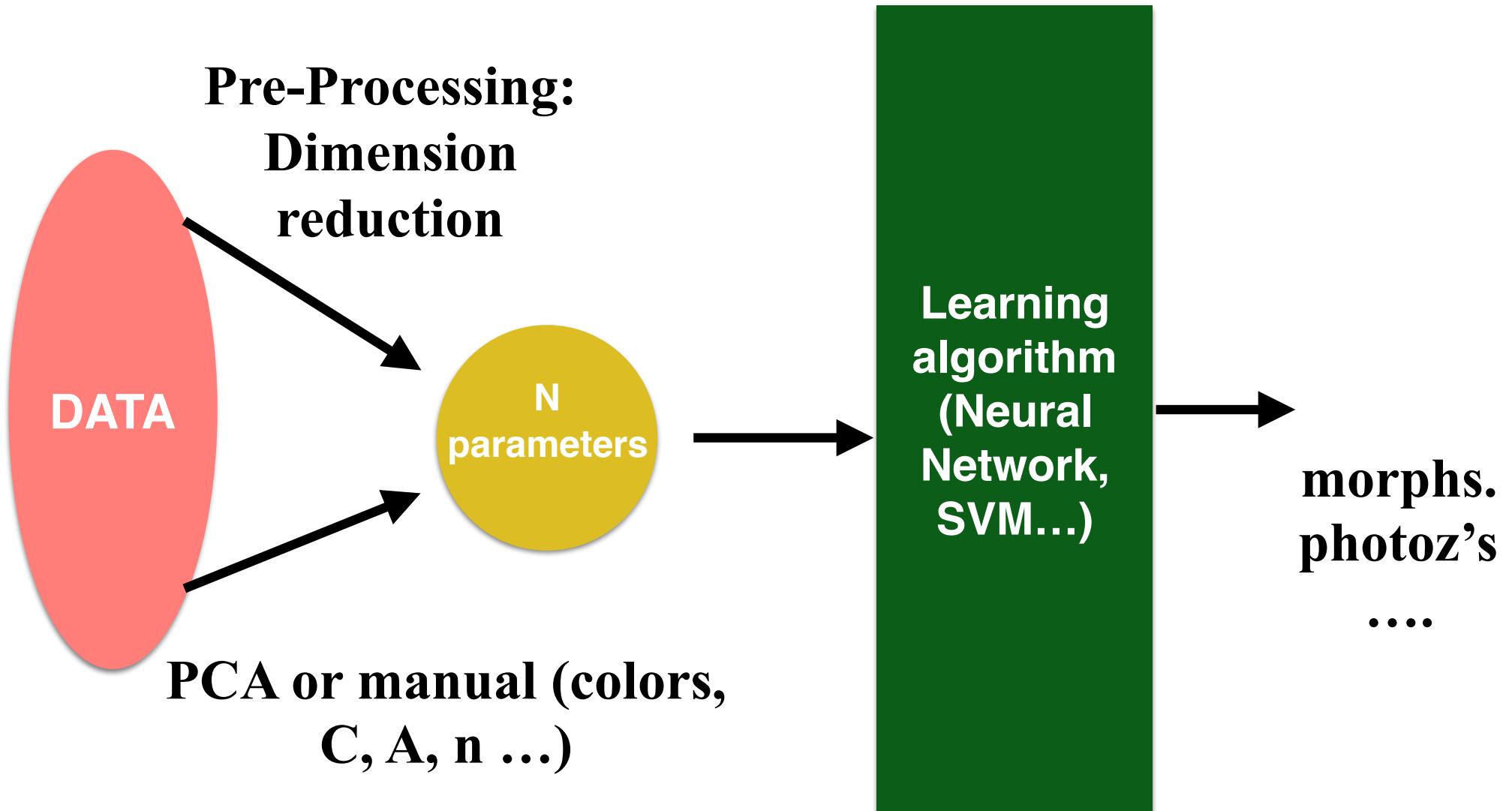
(U-V, V-J) FEATURES

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

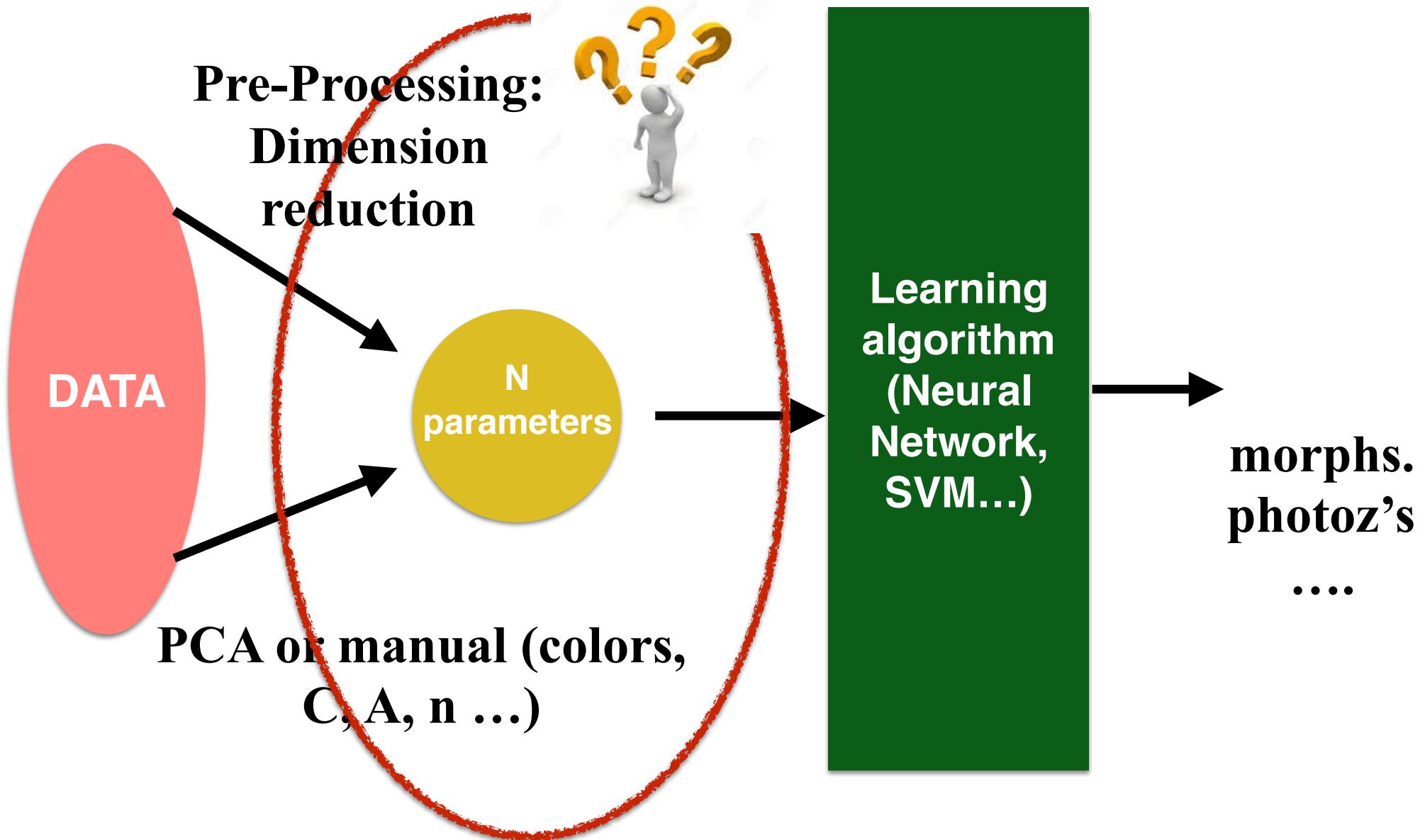
WEIGHTS



THE “CLASSICAL” APPROACH

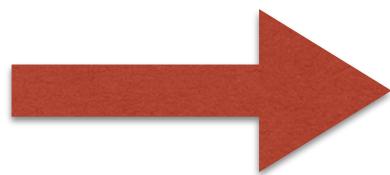


“CLASSICAL” MACHINE LEARNING



In Astronomy

- Colors, Fluxes
- Shape indicators
- Line ratios, spectral features
- Stellar Masses, Velocity Dispersions



Requires specialized software before feeding the machine learning algorithm

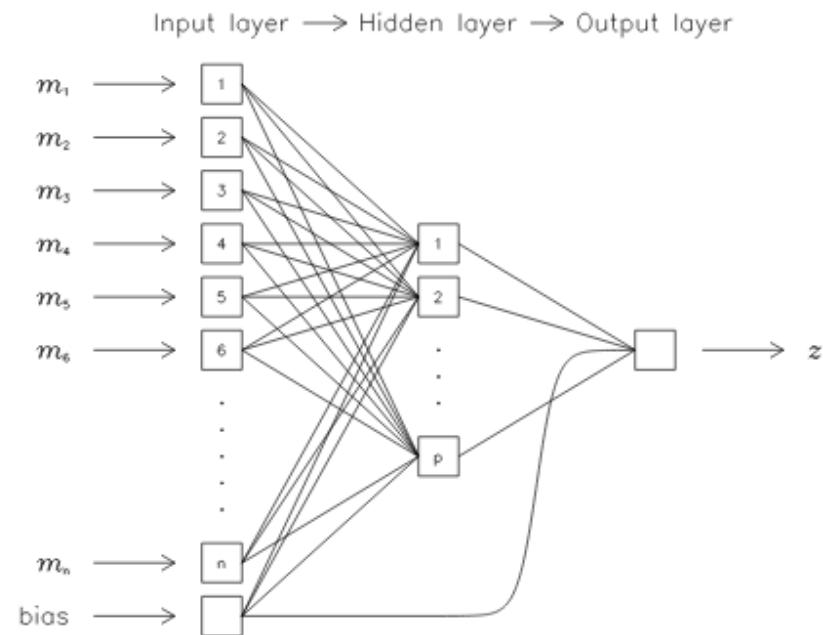
IT IMPLIES A DIMENSIONALITY REDUCTION!

PHOTOMETRIC REDSHIFTS

SDSS



g
r
i
z



Collister+08

**EVERYTHING IS IN THE FEATURES....WHAT IF I
IGNORED SOME IMPORTANT FEATURES?**



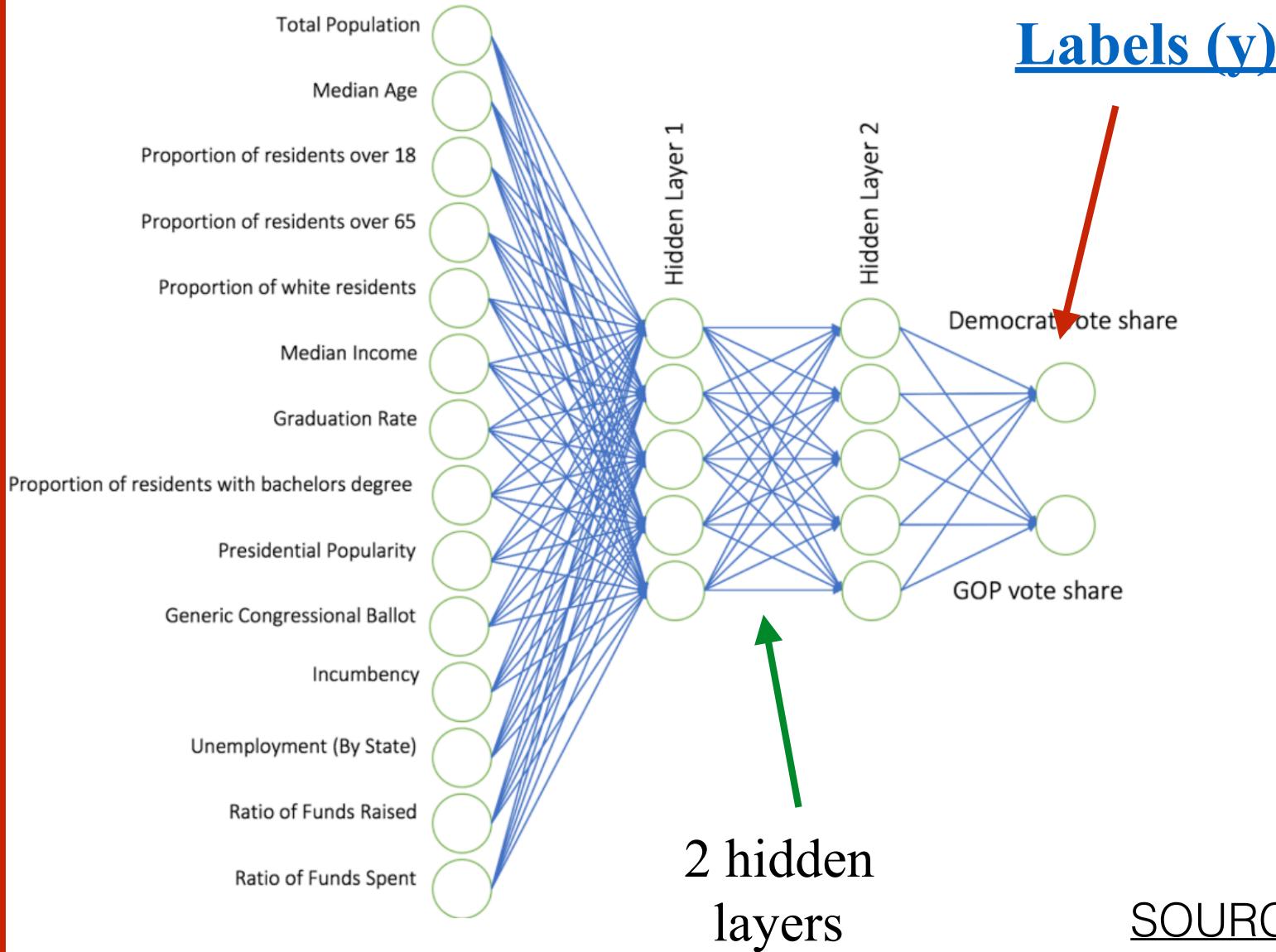
**EVERYTHING IS IN THE FEATURES....WHAT IF I
IGNORED SOME IMPORTANT FEATURES?**



NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

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

Features
(x)



SOURCE

NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

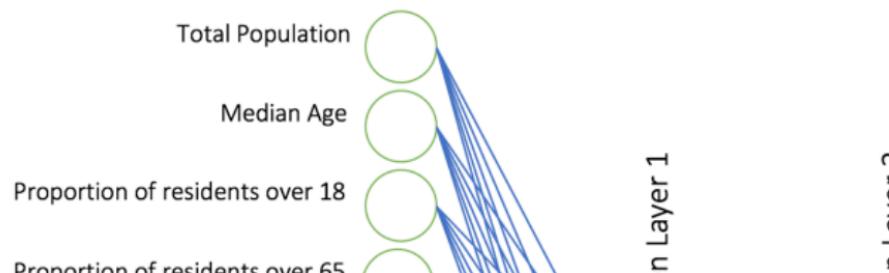
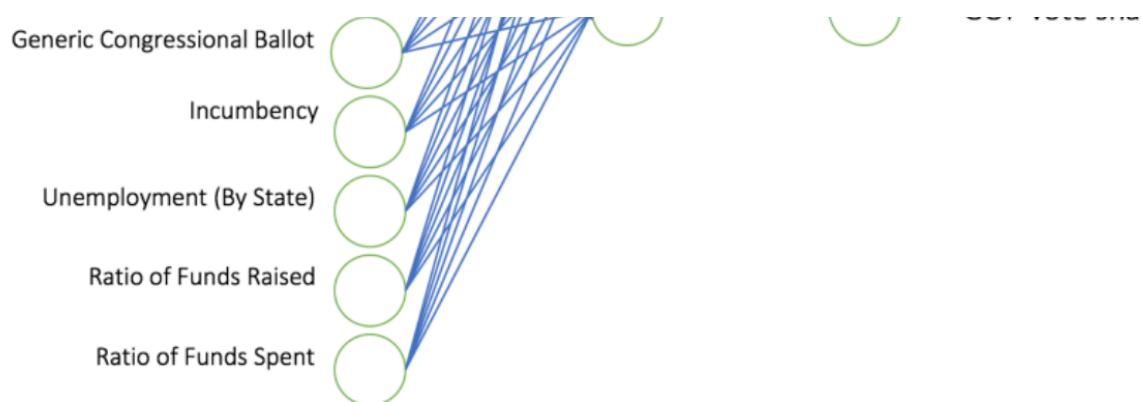


Table 2 – Results of both Models:

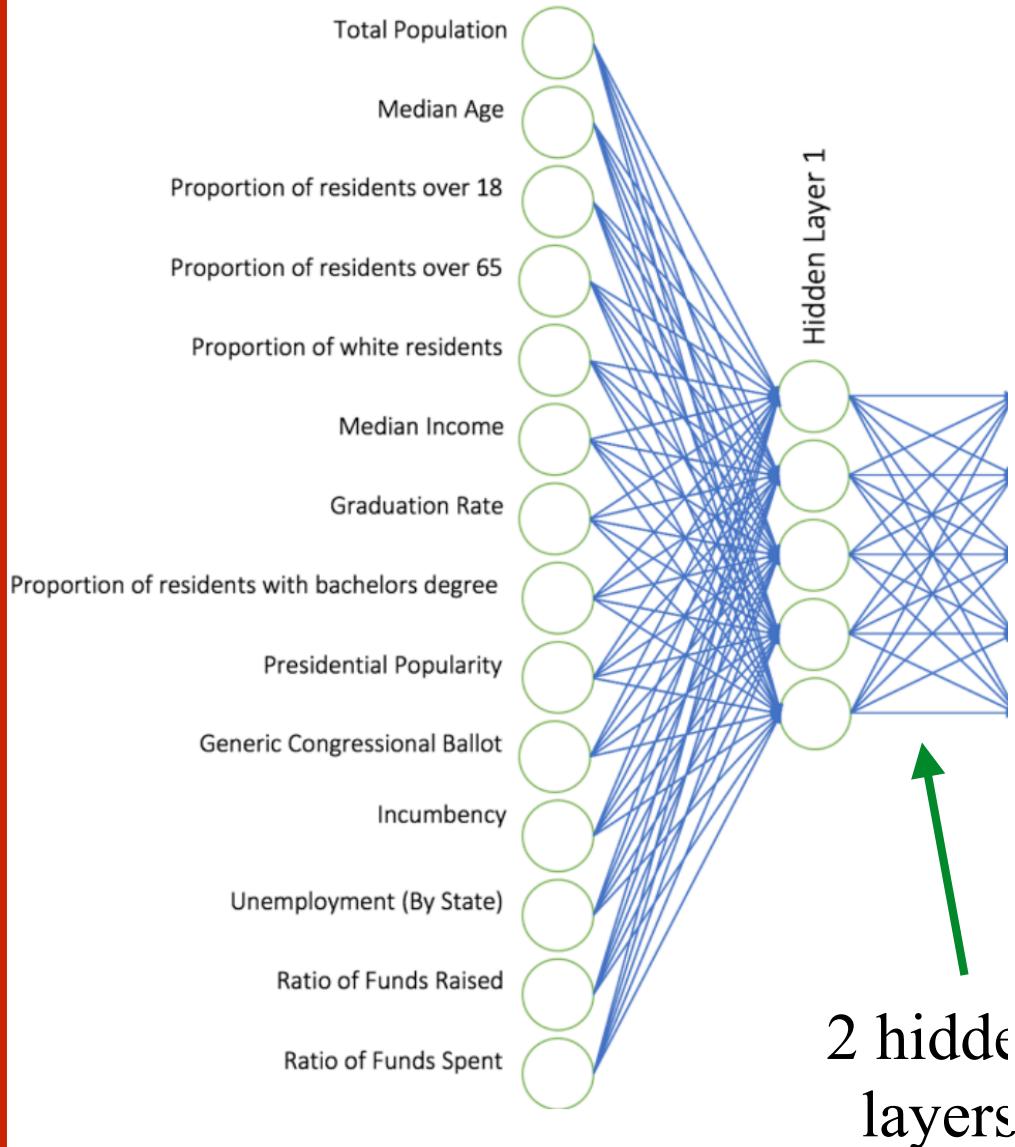
Model not including past elections (Model A)	Model including past elections (Model B)
D +3	D +17
Democrat Seats: 219	Democrat Seats: 226
Republican Seats: 216	Republican Seats: 209
33% chance of Republicans keeping house*	0.3% chance of Republicans keeping house*



232+198

NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

Features (x)



Bad Weather, Known to Lower Turnout, Will Greet Many Voters

Rain can decrease voter numbers, which studies show tends to help Republicans. “I hope it rains hard tomorrow,” one Republican candidate said.

10h ago

Other general computer vision features [for images!]

- Pixel Concatenation
- Color histograms
- Texture Features
- Histogram of Gradients
- SIFT

FOR MANY YEARS COMPUTER VISION RESEARCHERS HAVE BEEN TRYING TO FIND THE MOST GENERAL FEATURES

Other general computer vision features [for images!]

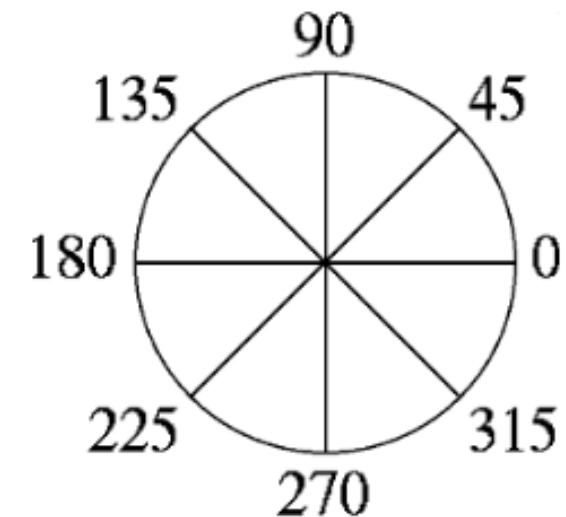
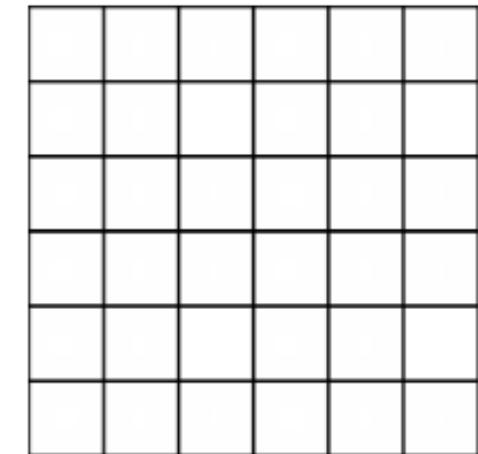
- Pixel Concatenation
- Color histograms
- Texture Features
- Histogram of Gradients
- SIFT

FOR MANY YEARS COMPUTER VISION RESEARCHERS HAVE BEEN TRYING TO FIND THE MOST GENERAL FEATURES

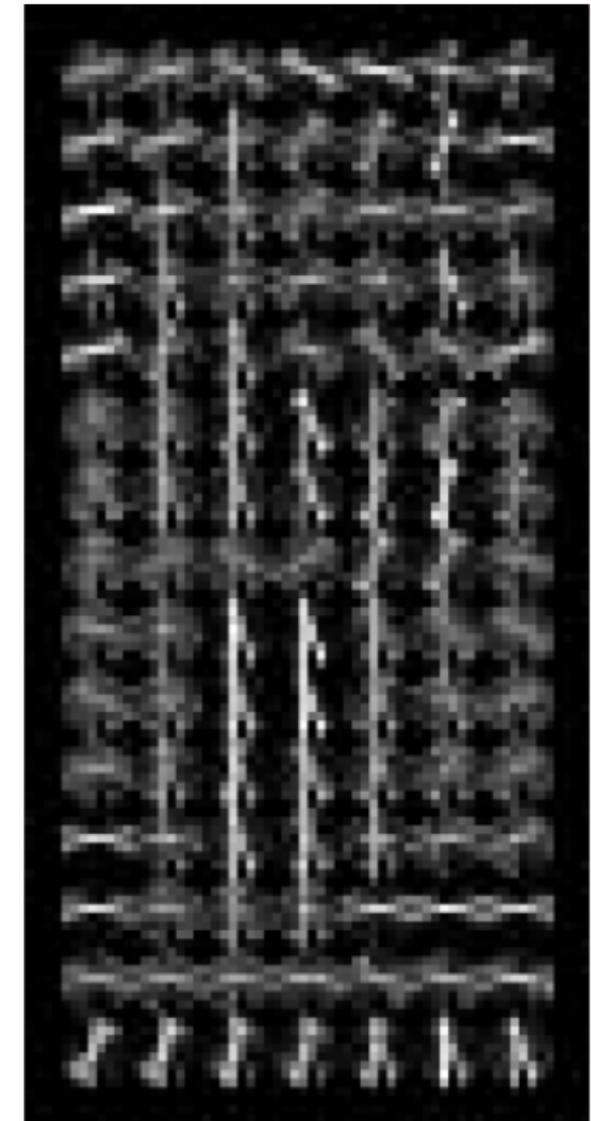
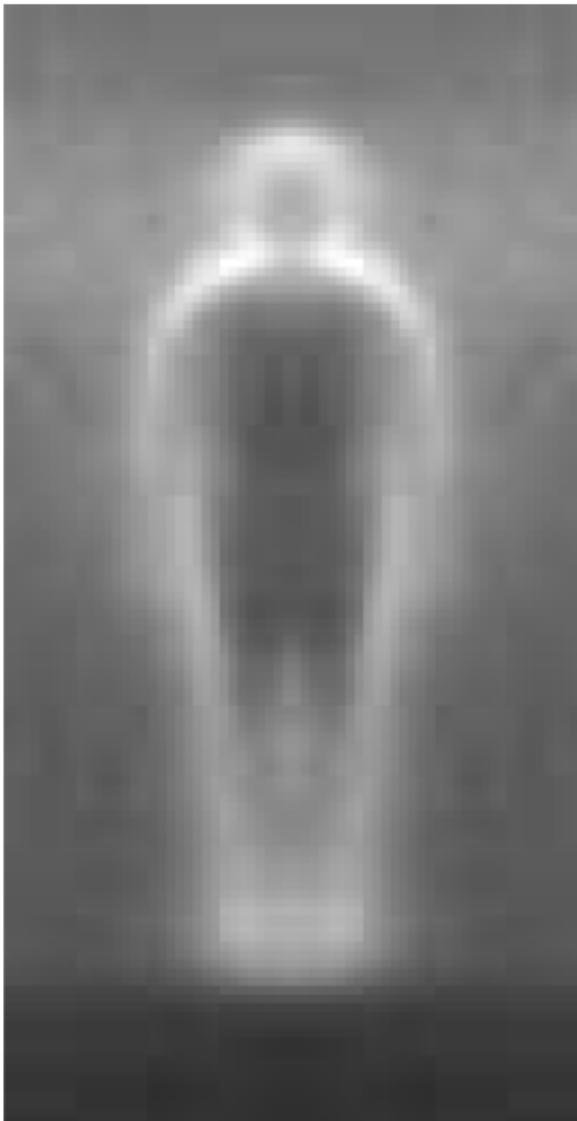
THE BEST CLASSICAL SOLUTION [BEFORE 2012] WHERE BASED ON LOCAL FEATURES

HISTOGRAM OF ORIENTED GRADIENTS (HoG)

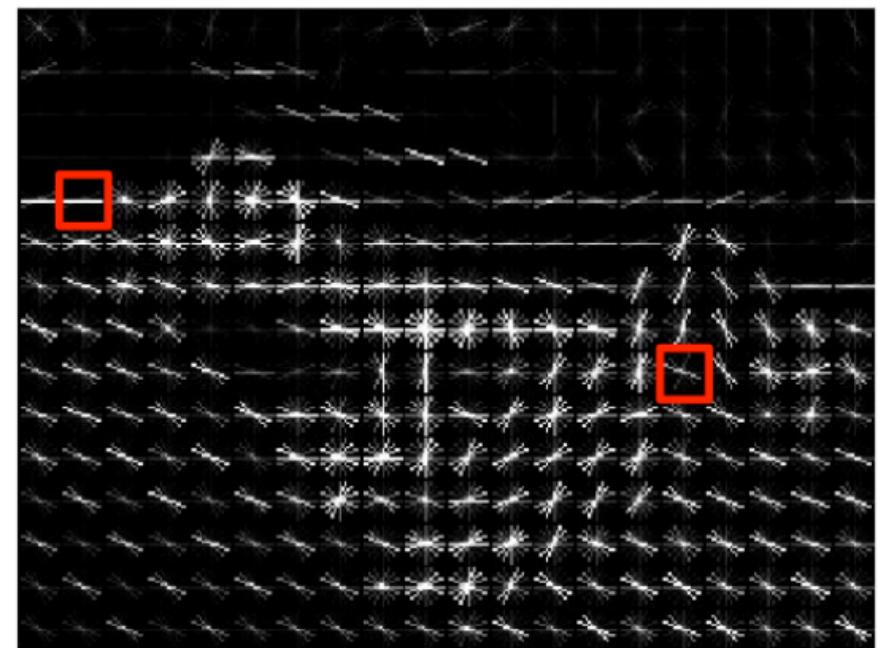
1. DIVIDE IMAGE INTO SMALL SPATIAL REGIONS CALLED CELLS
2. COMPUTE INTENSITY GRADIENTS OVER N DIRECTIONS [TYPICALLY 9 FOR IMAGE]
3. COMPUTE WEIGHTED 1-D HISTOGRAM OF ALL DIRECTIONS. A CELL IS REDUCED TO N NUMBERS



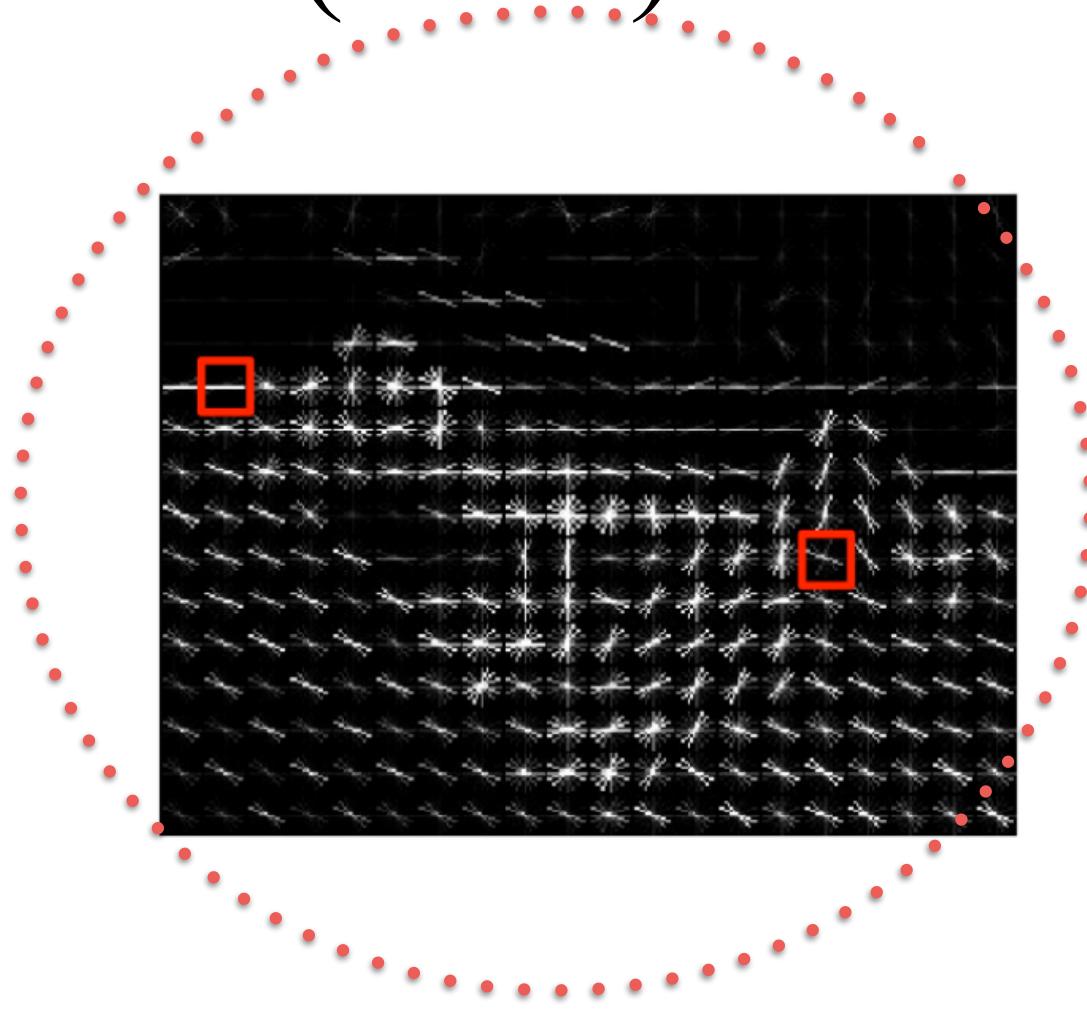
HISTOGRAM OF ORIENTED GRADIENTS (HoG)



HISTOGRAM OF ORIENTED GRADIENTS (HoG)



HISTOGRAM OF ORIENTED GRADIENTS (HoG)



KEEP THIS IMAGE IN MIND FOR LATER...

SIFT FEATURES

[Scale Invariant Feature Transform]

1. FIND LOCAL REGIONS OF INTEREST
WITHIN THE IMAGE

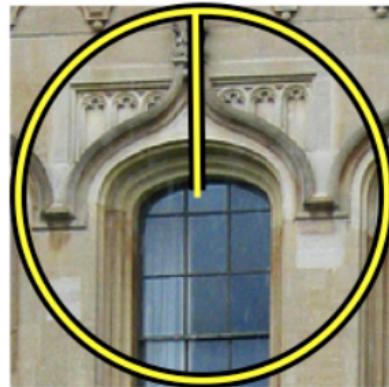
2. REMOVE EFFECTS OF ORIENTATION
AND SCALE

3. COMPUTE HISTOGRAM
OF GRADIENTS IN DEFINED REGIONS

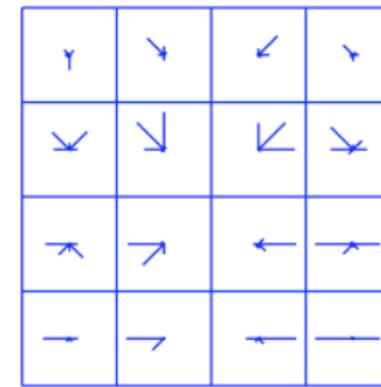
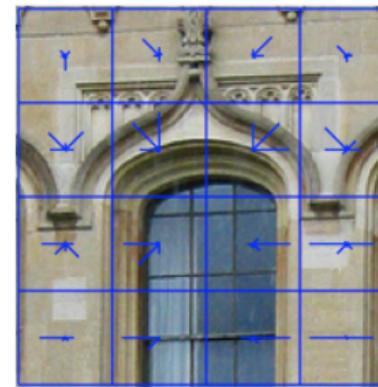
SIFT FEATURES

[Scale Invariant Feature Transform]

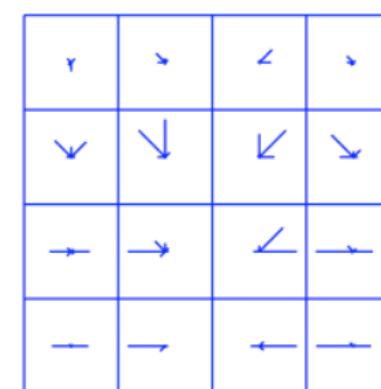
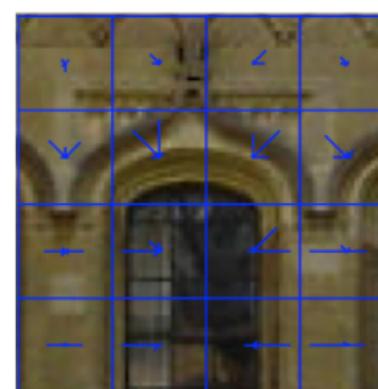
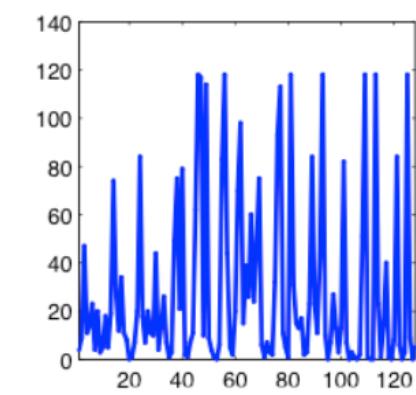
normalised features



spatial histogram of gradients



SIFT descriptor



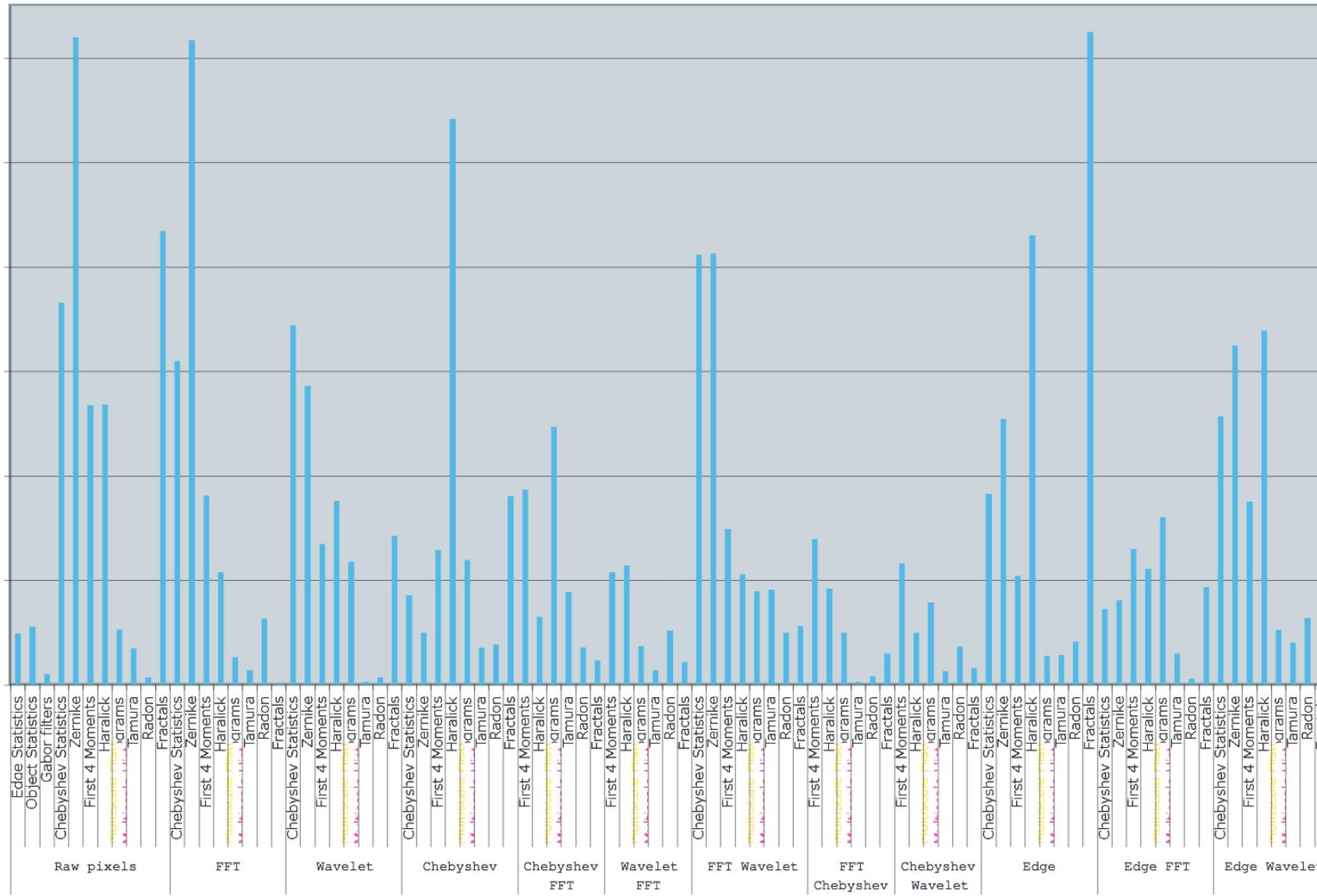
d

d'

TRADITIONAL FEATURES HAVE GENERALLY IN COMMON...

- 1.ACT LOCALLY
- 2.LOOK FOR DIFFERENT ORIENTATIONS
- 3.TRY TO BE SCALE / TRANSLATION / ROTATION
INVARIANT

A LARGE AMOUNT OF FEATURES CAN BE COMBINED



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

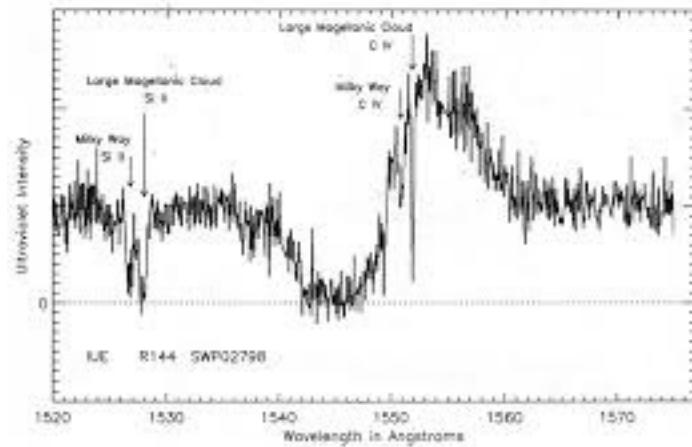
LARGE DIMENSION SIGNALS SUCH AS IMAGES OR SPECTRA WOULD REQUIRE TREMENDOUSLY LARGE MODELS

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

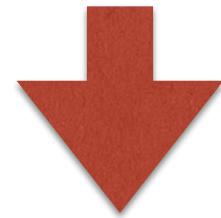
$$(512 \times 512)^2 = 7e10$$

BUT

FEEDING INDIVIDUAL RESOLUTION ELEMENTS IS NOT
VERY EFFICIENT SINCE IT LOOSES ALL INVARIANCE TO
TRANSLATION AND IGNORES CORRELATION IN THE DATA

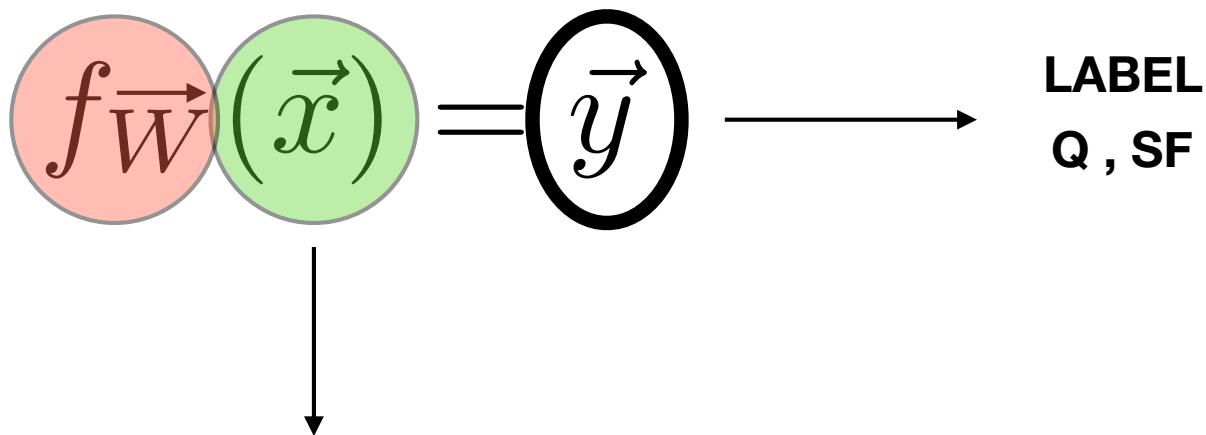


FEEDING INDIVIDUAL RESOLUTION ELEMENTS IS NOT
VERY EFFICIENT SINCE IT LOOSES ALL INVARIANCE TO
TRANSLATION



SO?

DEEP LEARNING



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

LET'S GO A STEP FORWARD INTO LOOSING CONTROL...

QUESTIONS?