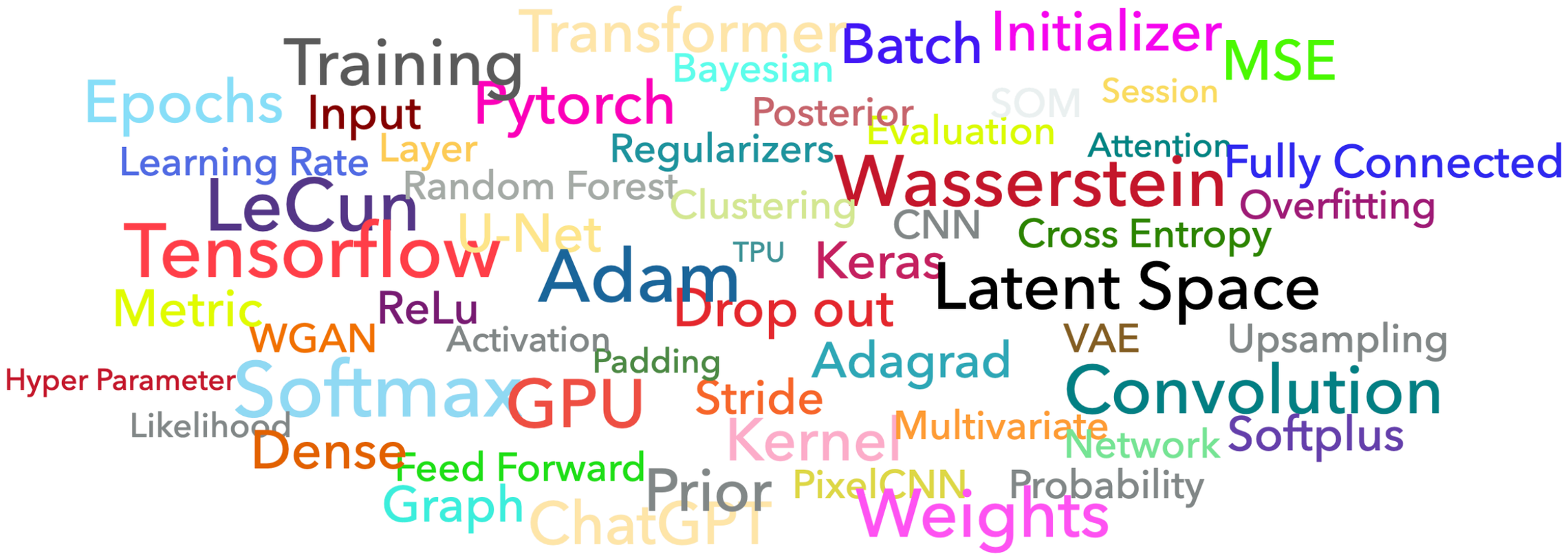
Introduction to deep learning

Marc Huertas-Company - Hubert Bretonnière



Plan (intended)

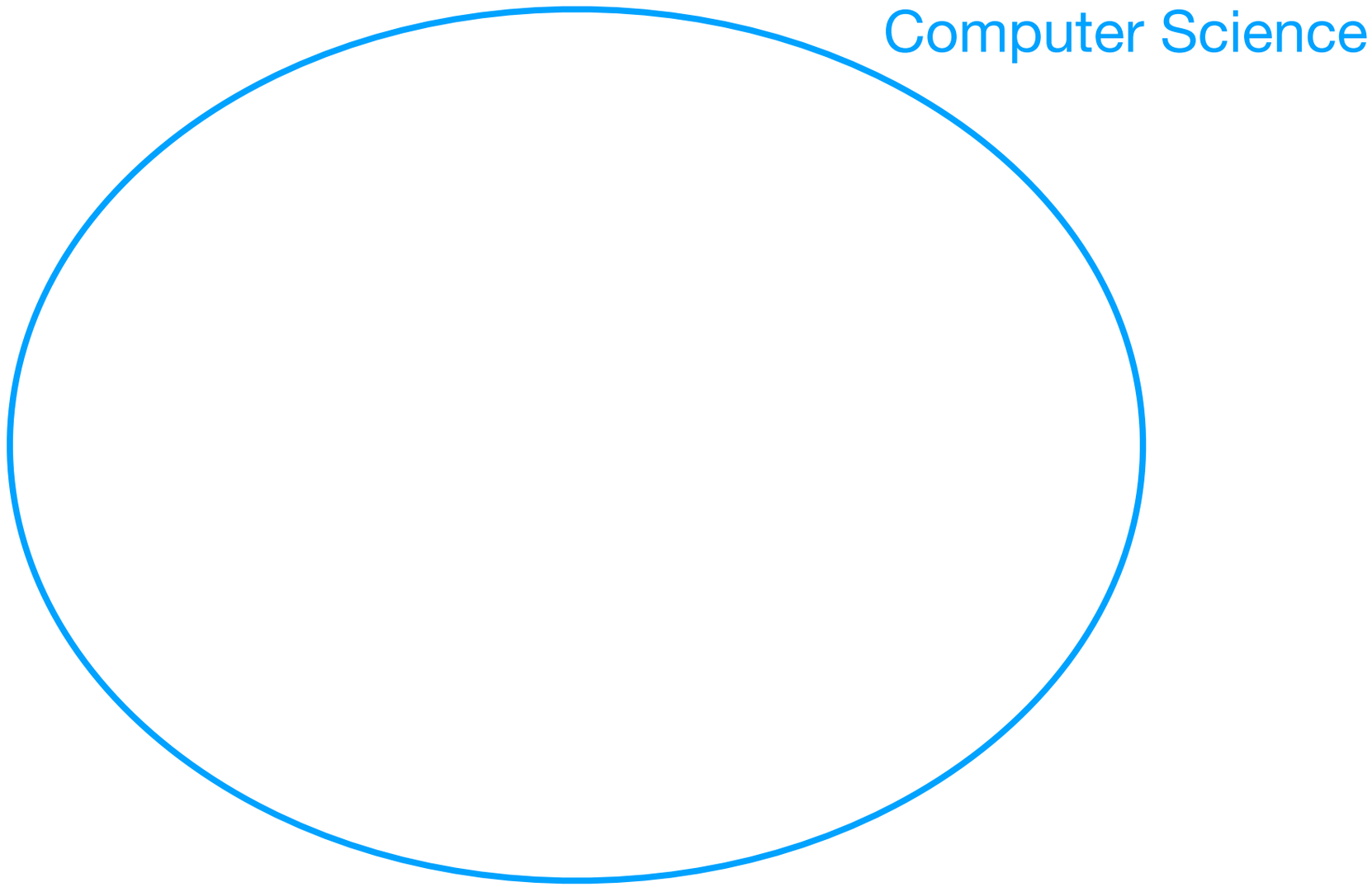
1. Introduction / Contextualisation

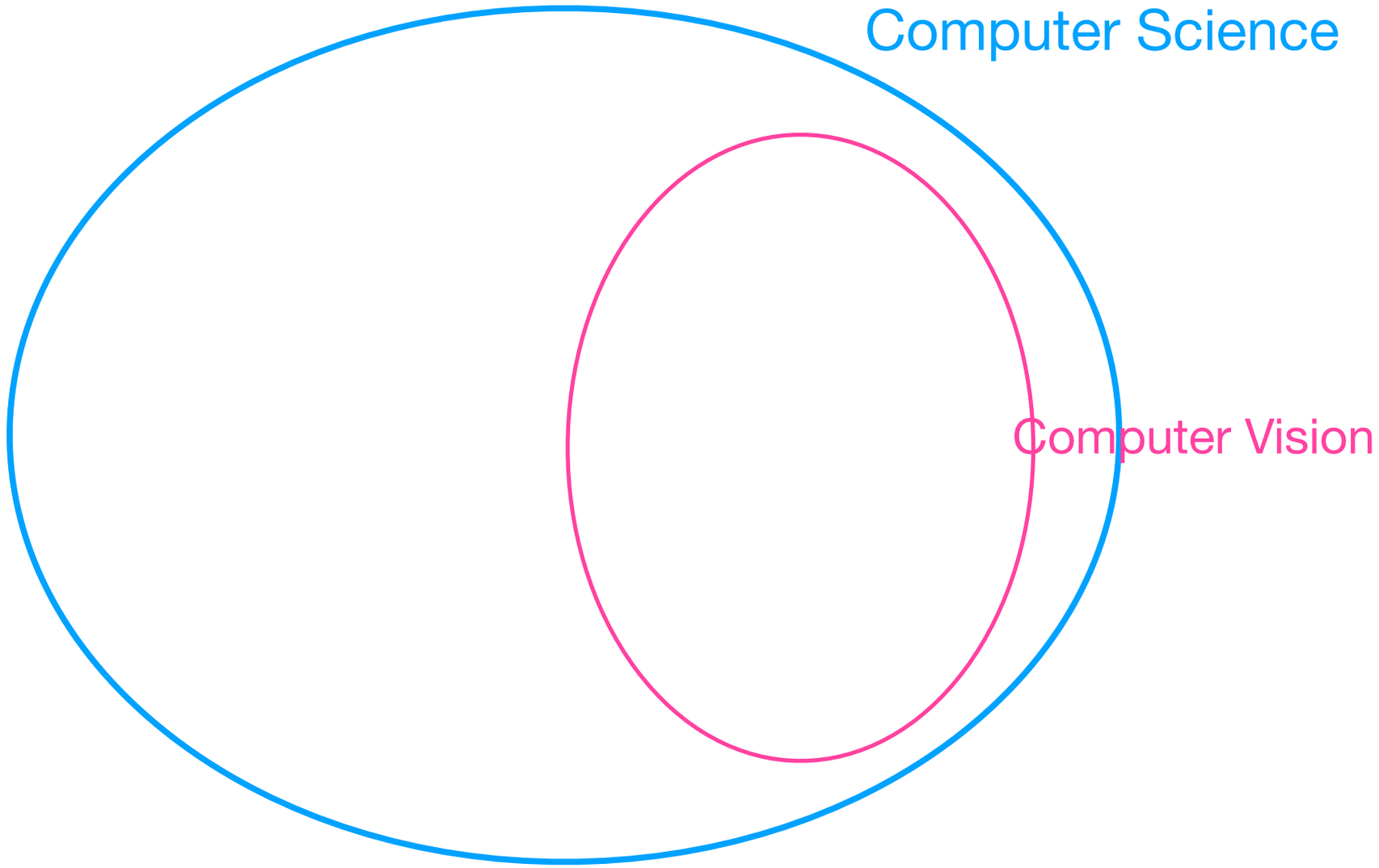
2. The common blocks and principles of DL :

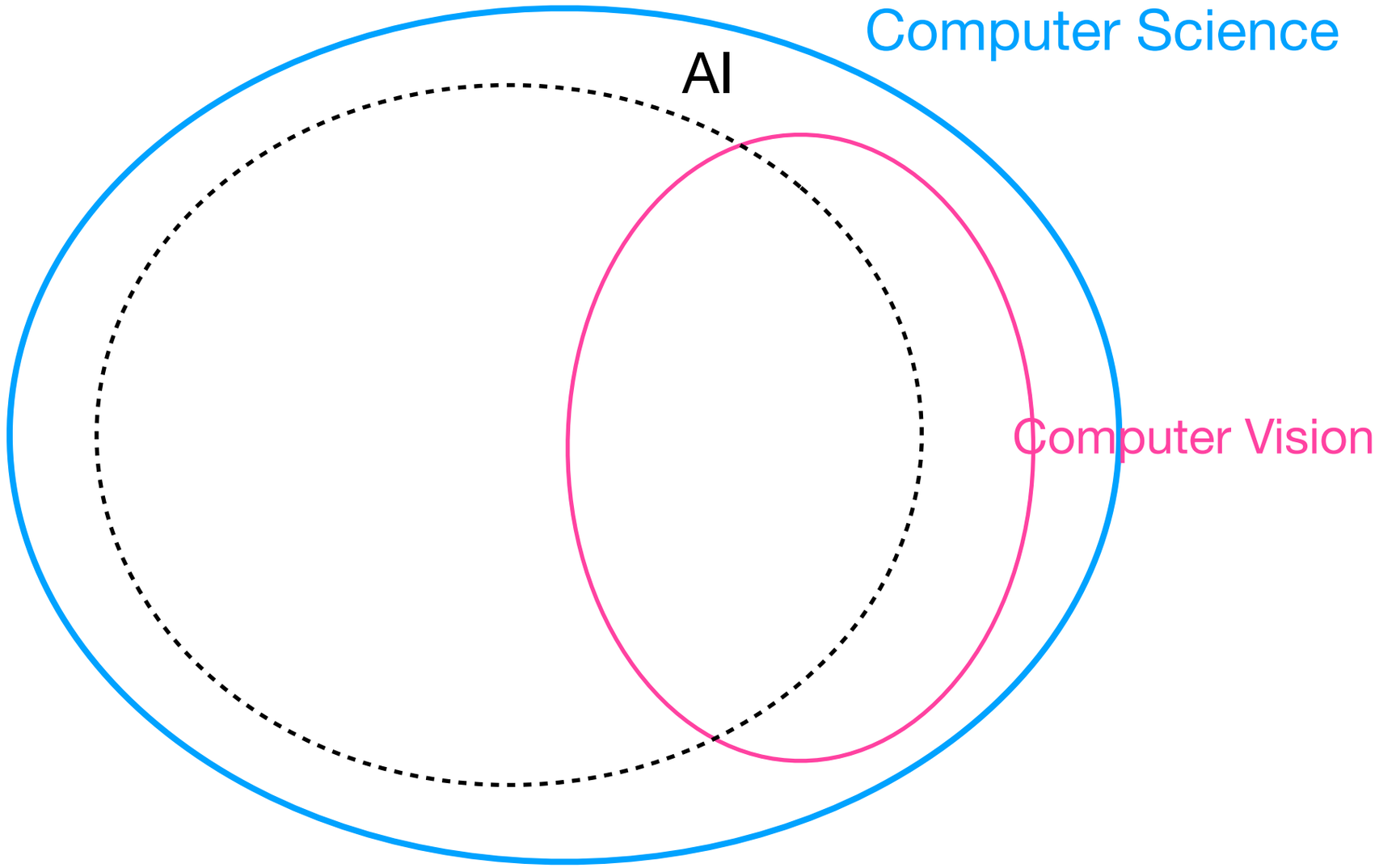
    From easy to not so much + notebooks

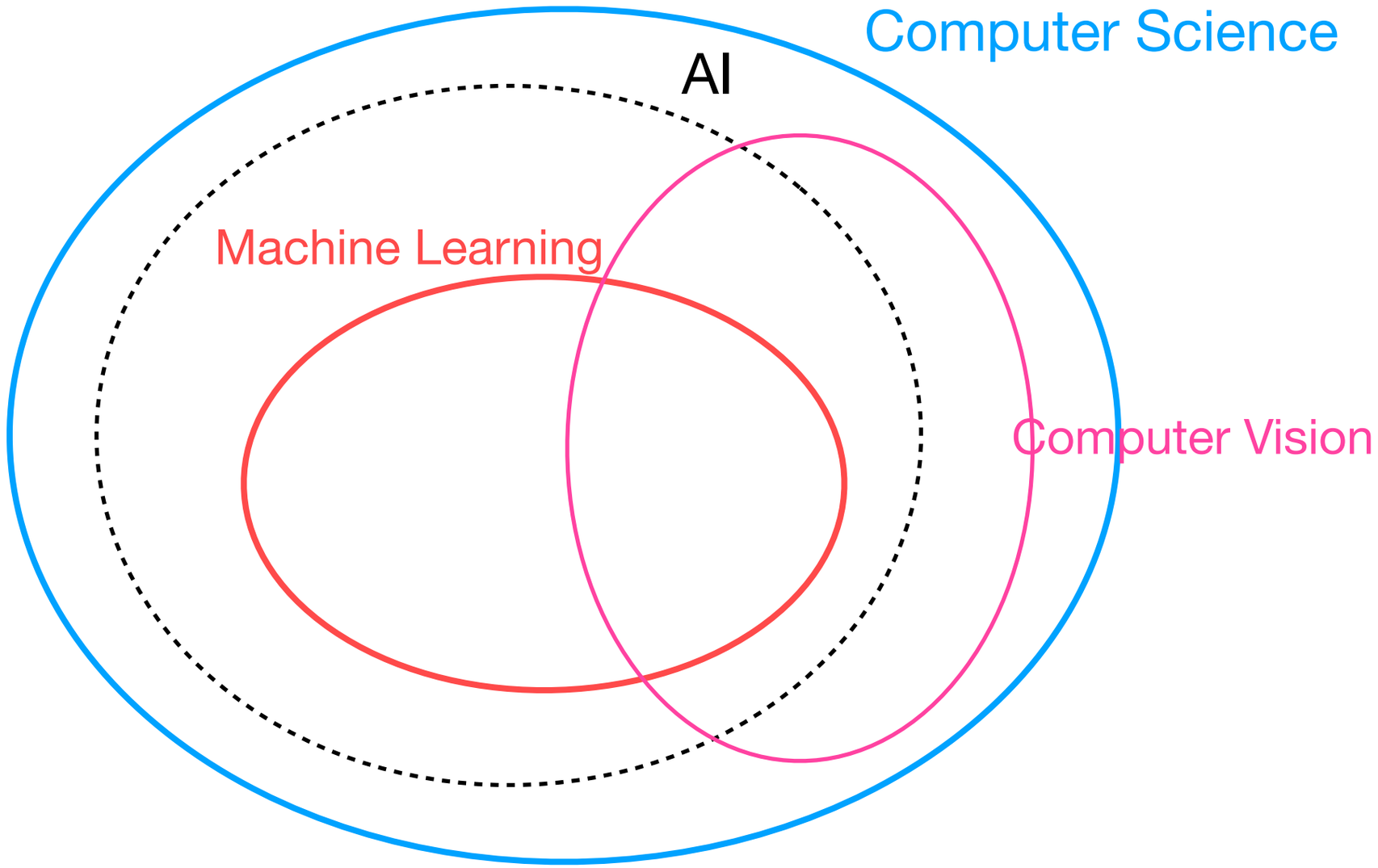
3. Going Convolutional + notebook

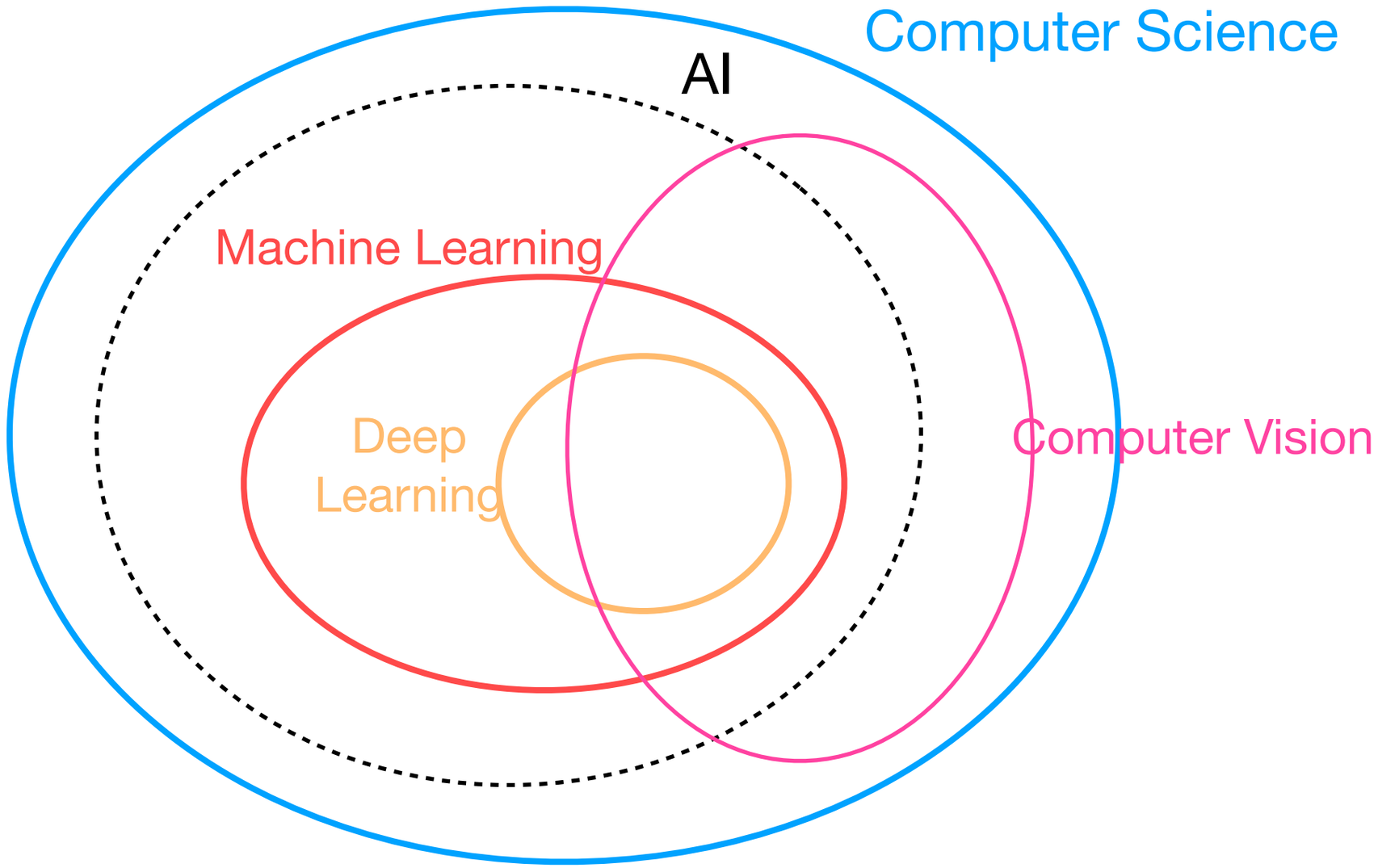
4. Attention is all you need + notebook

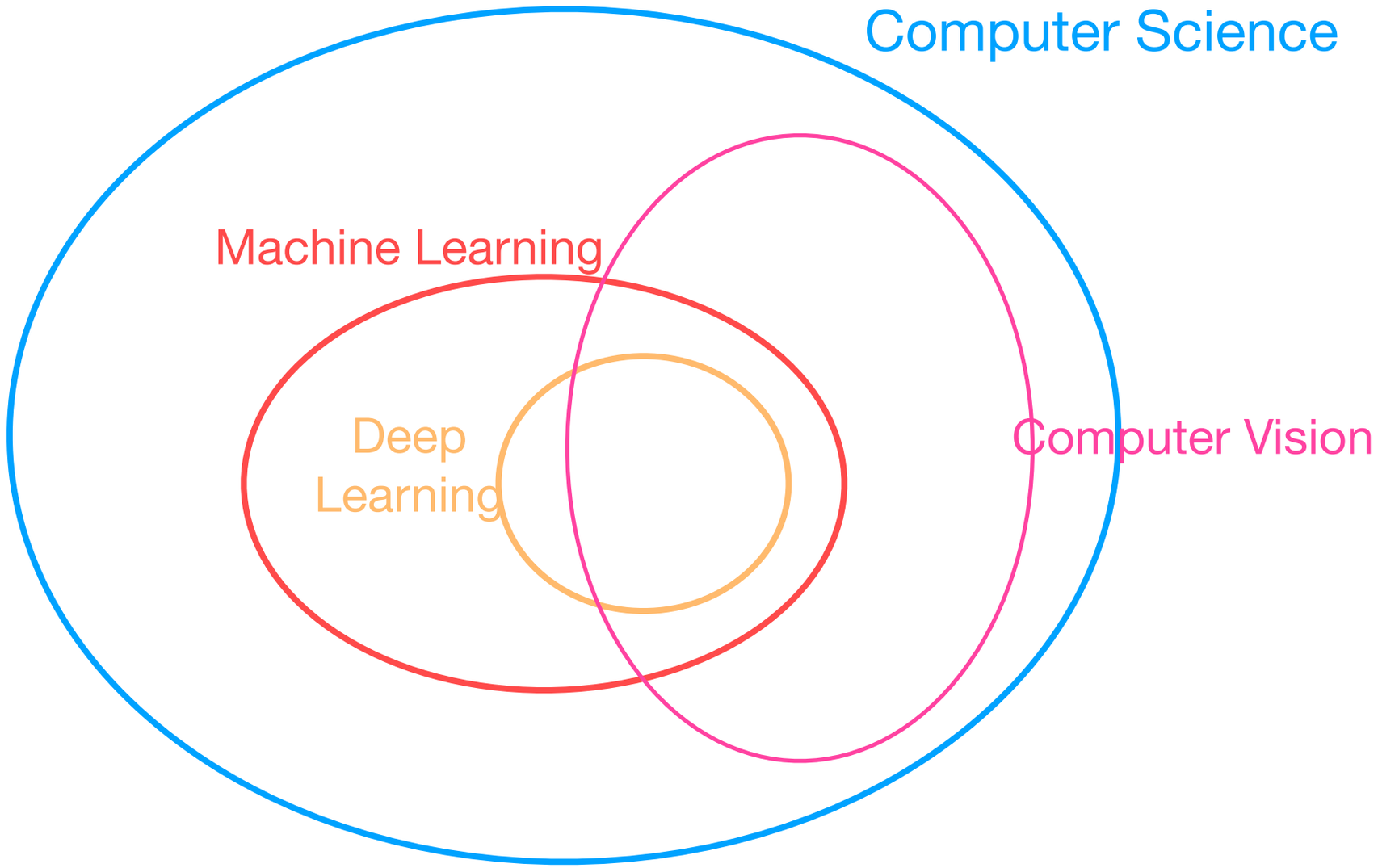

Computer Science


Computer Science
Computer Vision


Computer Science
AI
Computer Vision


Computer Science
AI
Machine Learning
Computer Vision


Computer Science
AI
Machine Learning
Deep
Learning
Computer Vision

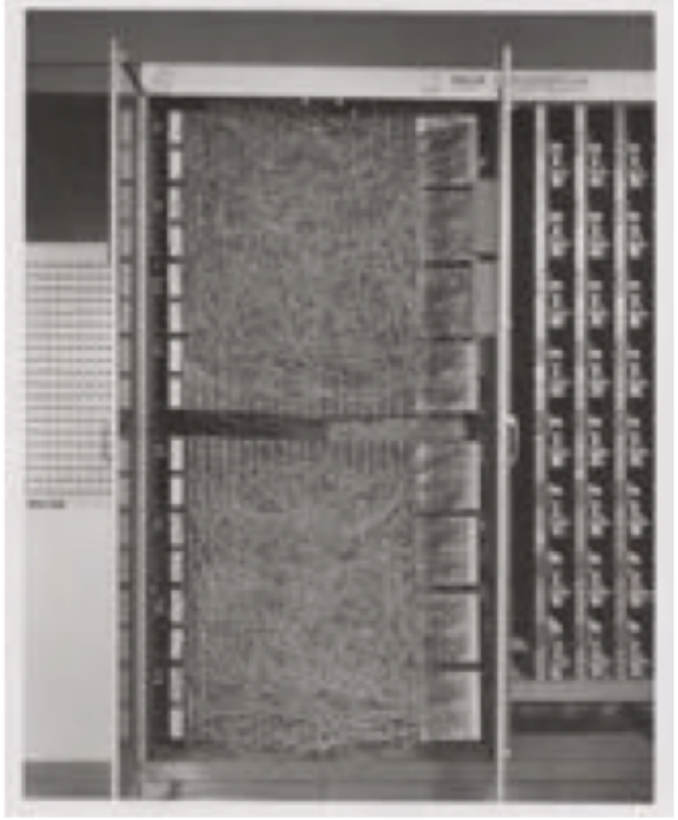

Computer Science
Machine Learning
Deep
Learning
Computer Vision

A bit of History: Rosenblatt Perceptron

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

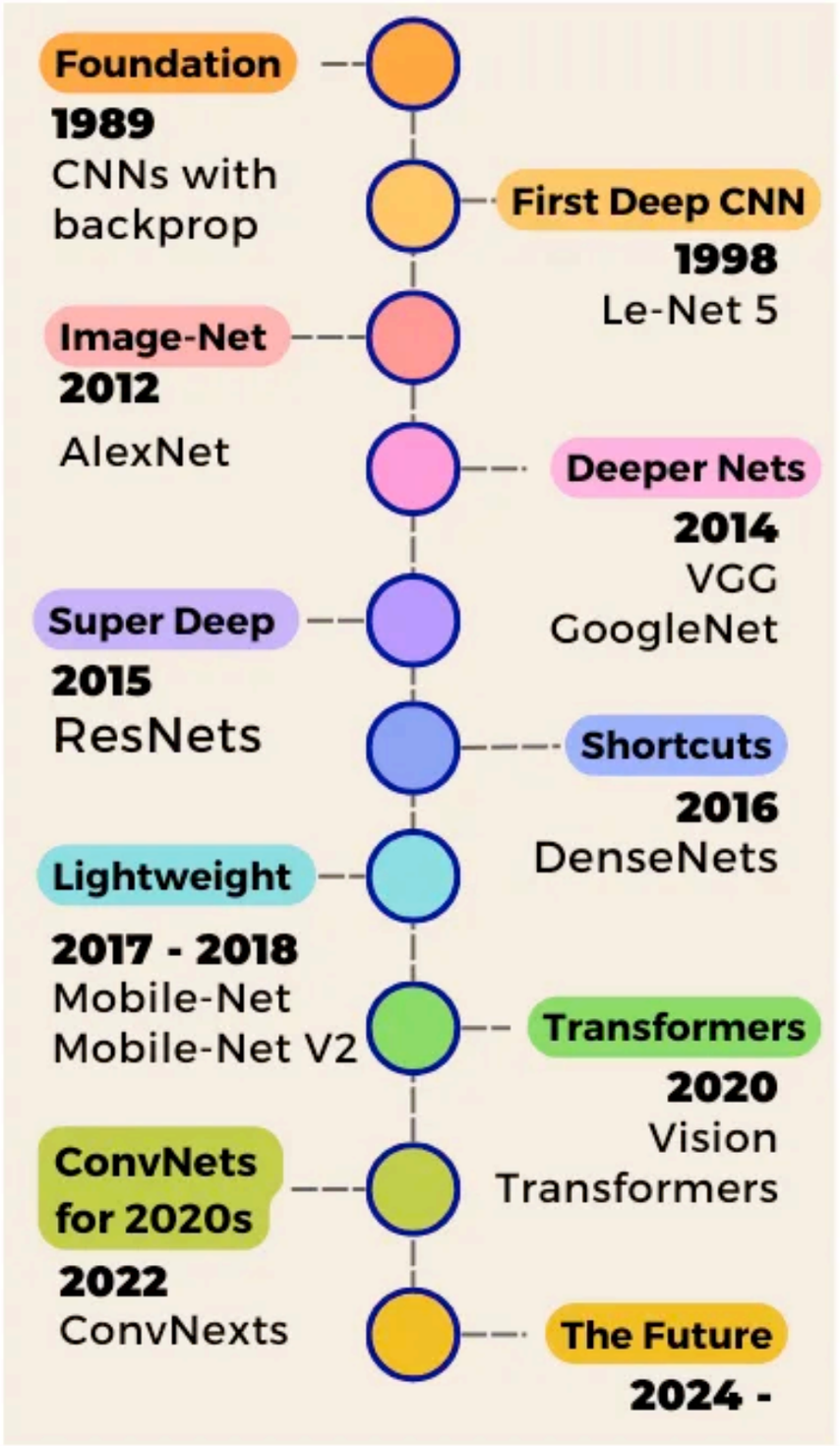
Buulding on the McCulloch-Pitts Neuron (1943) !!!

INTENDED TO BE A MACHINE (NOT AN ALGORITHM)



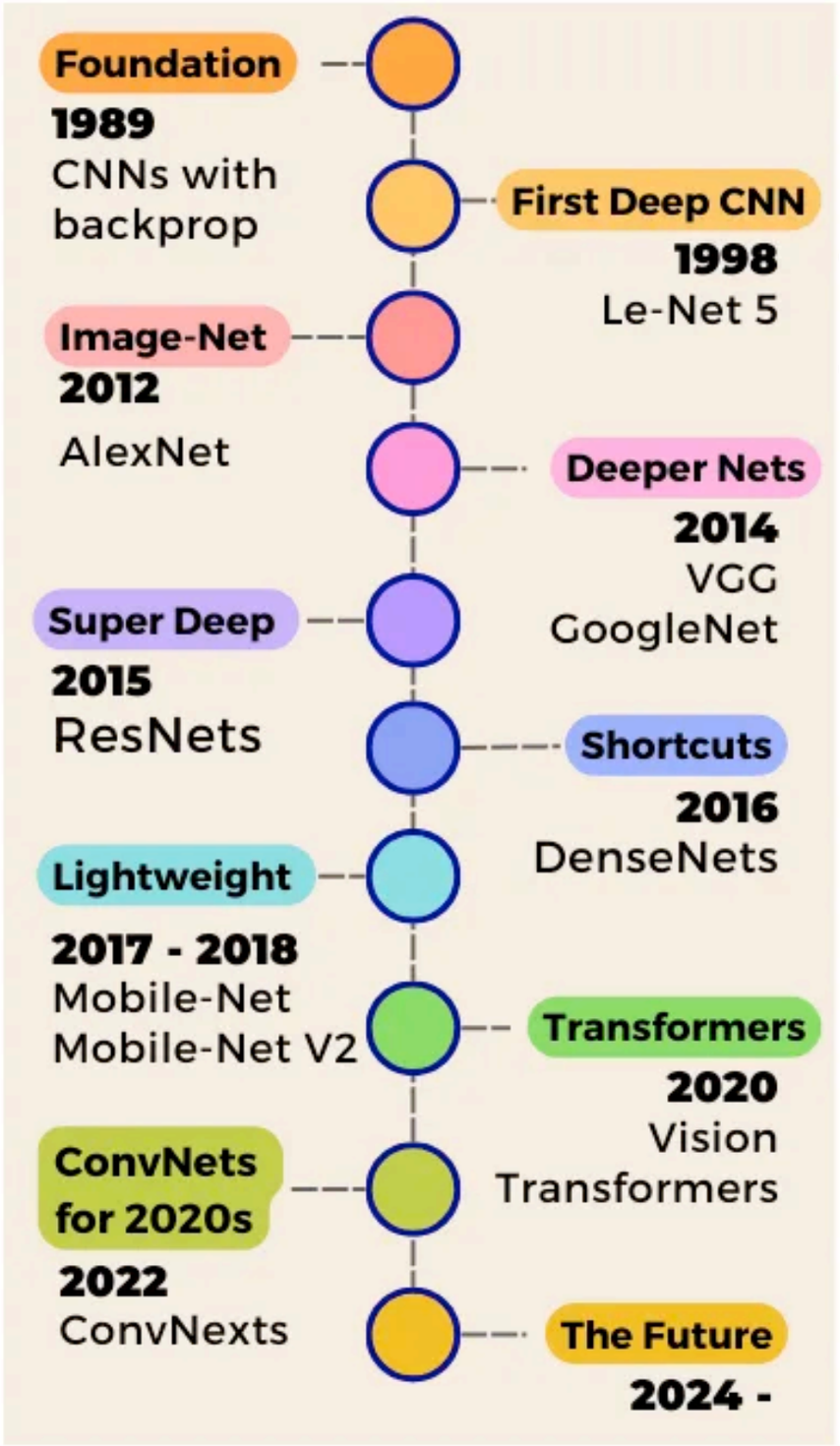
it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors

A bit of History: the evolution



Credit

A bit of History: the evolution



Credit

} Development of GPUs, private sector/companies start to use DL massively

Before 2012

CAT?

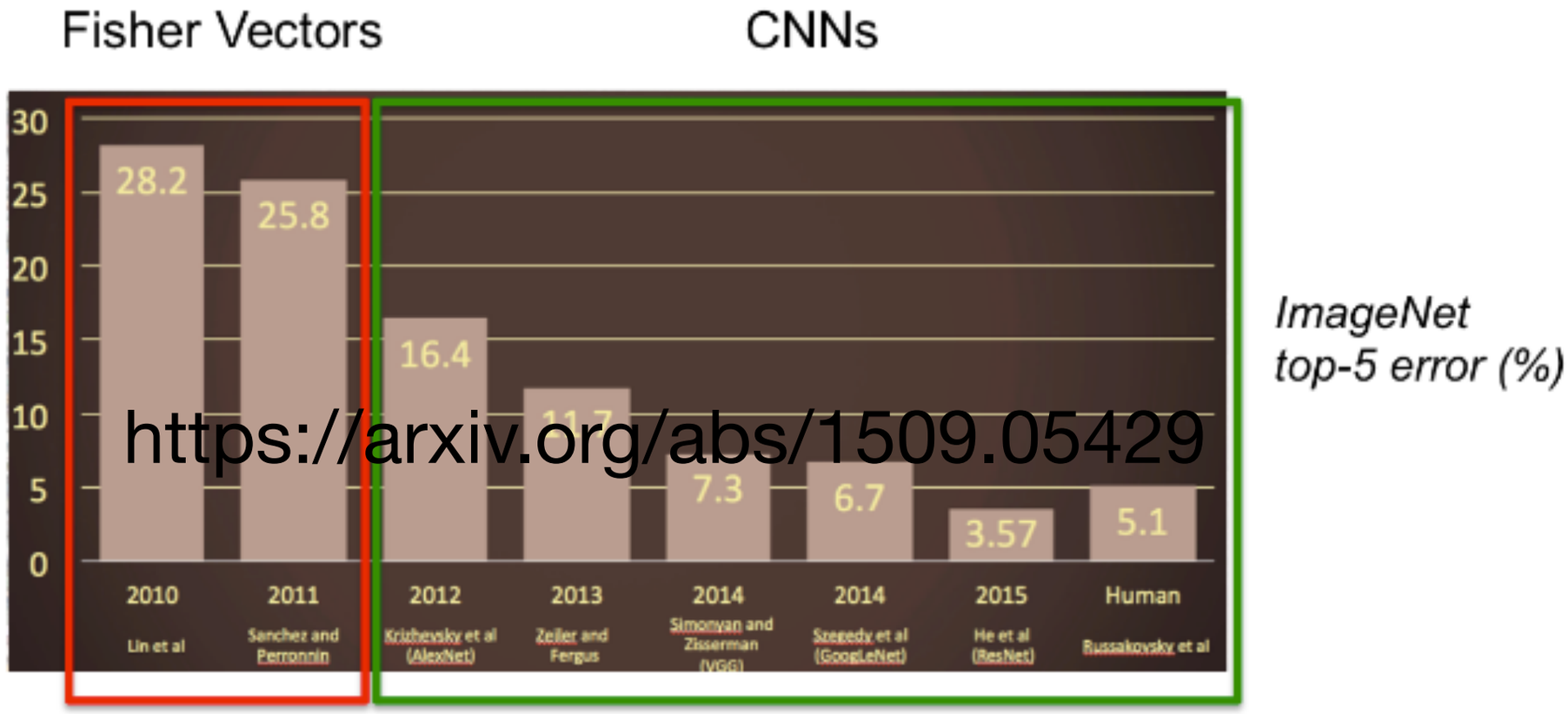
DOG?

TRIVIAL HUMAN TASKS REMAINED CHALLENGING FOR COMPUTERS

After 2012

IT HAS BECOME TRIVIAL….

The end of the ML winter…



A few years later -> academia

2015:

- Rotation-invariant convolutional neural networks for galaxy morphology prediction, Dielman et al.

- A catalog of visual-like morphologies in the 5 CANDELS fields using deep-learning, Huertas-Company et al.

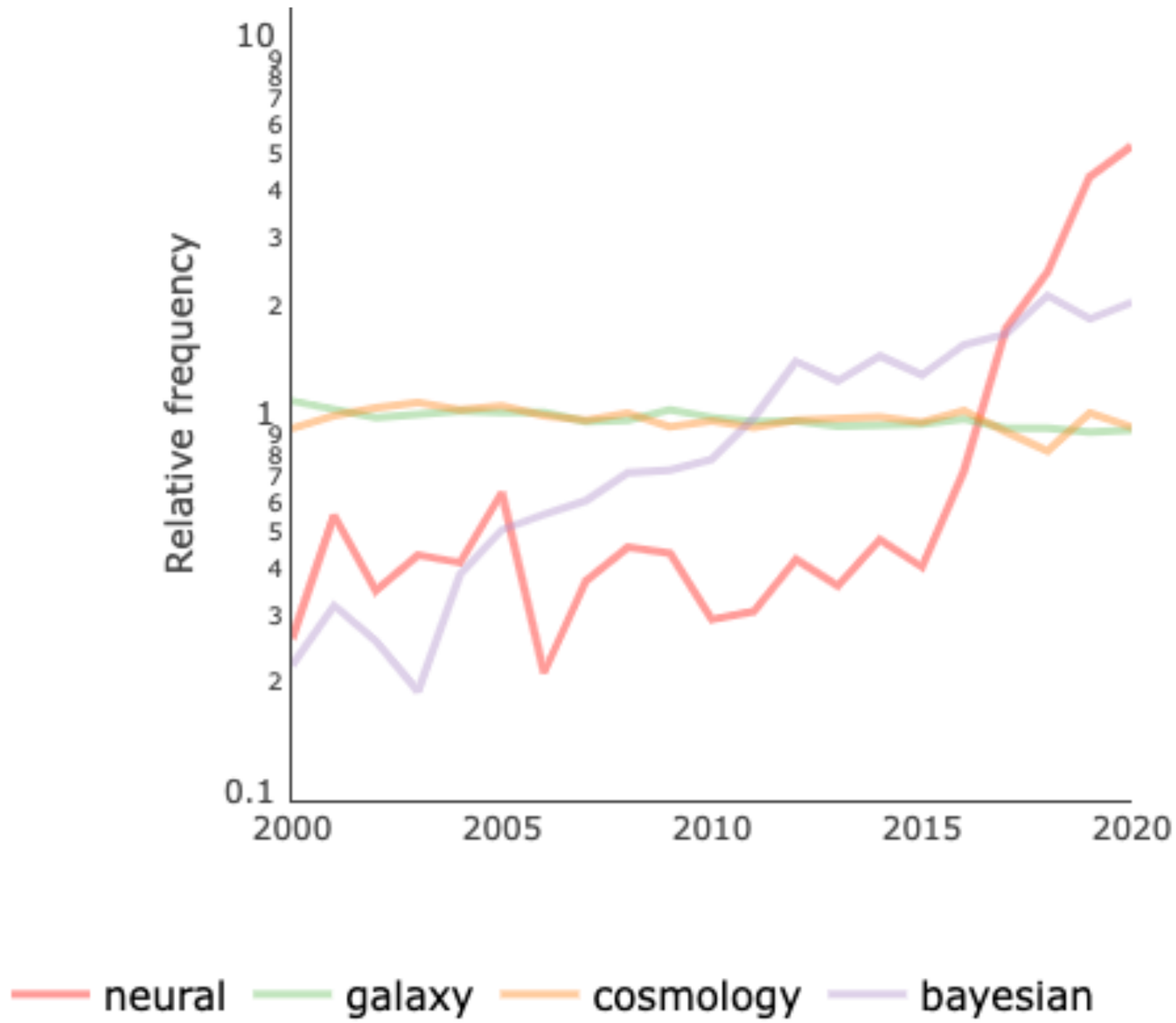
2018:

- Improving galaxy morphologies for SDSS with Deep Learning, Domínguez Sánchez, H

Before that, ML was already present.

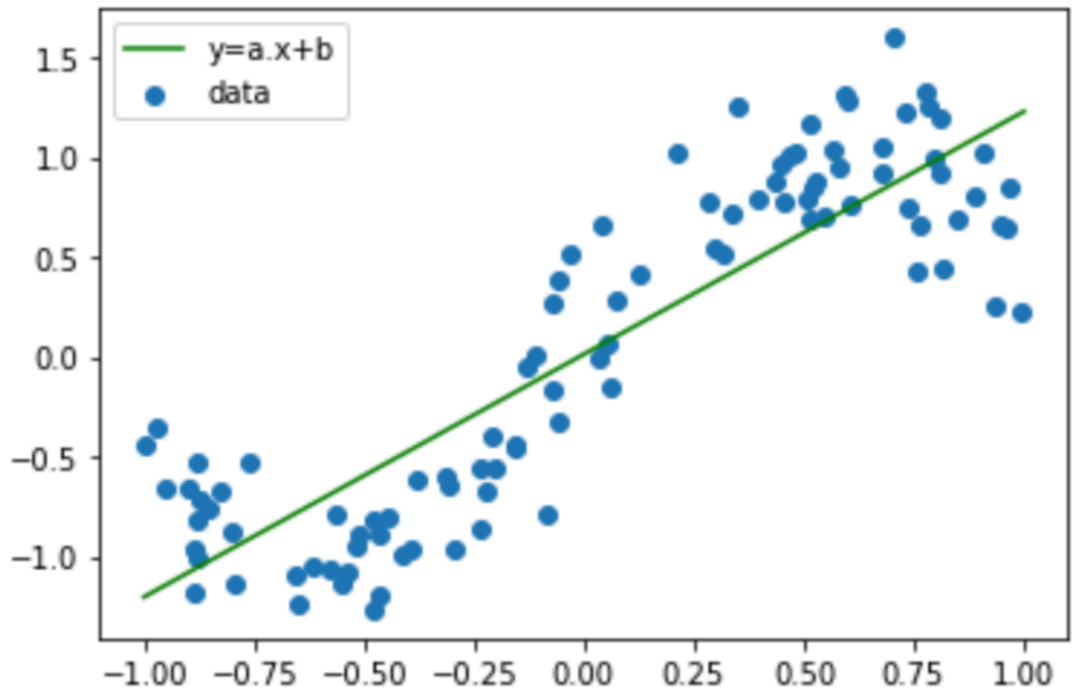
See Huertas-Company, Lanusse for a review: https://arxiv.org/abs/2210.01813

Why are neural networks special?



Huertas-Company+23

Why would we want Deep Learning ?

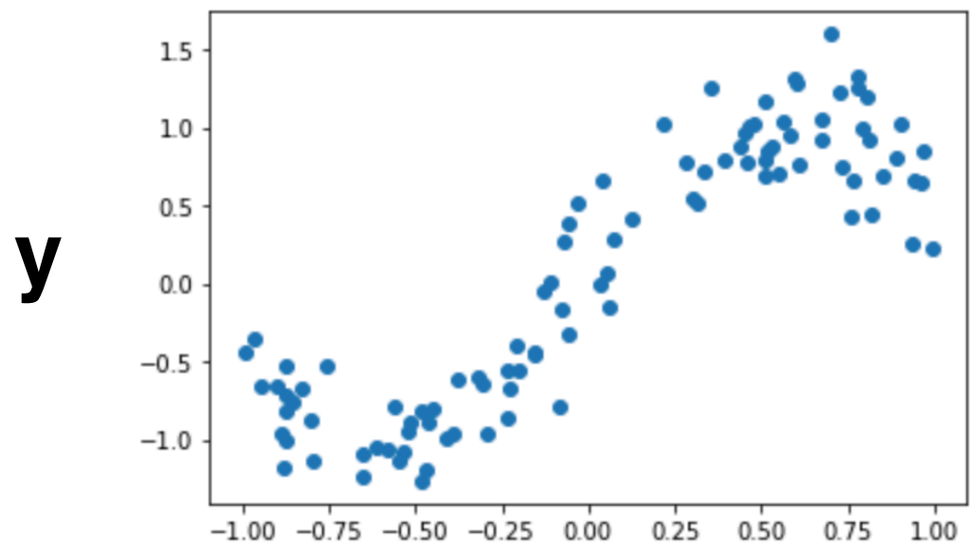


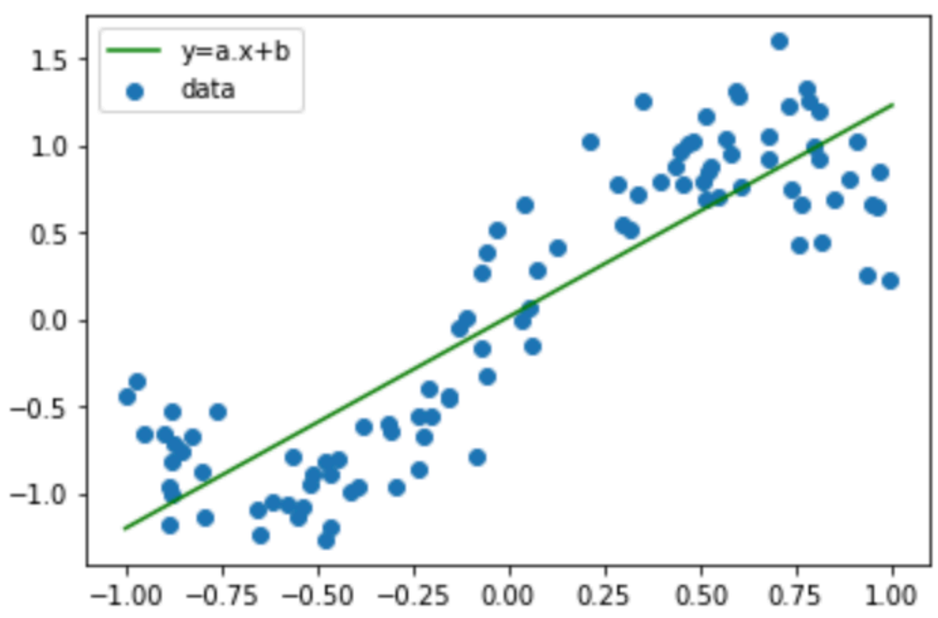
Because it is data-driven

- 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**

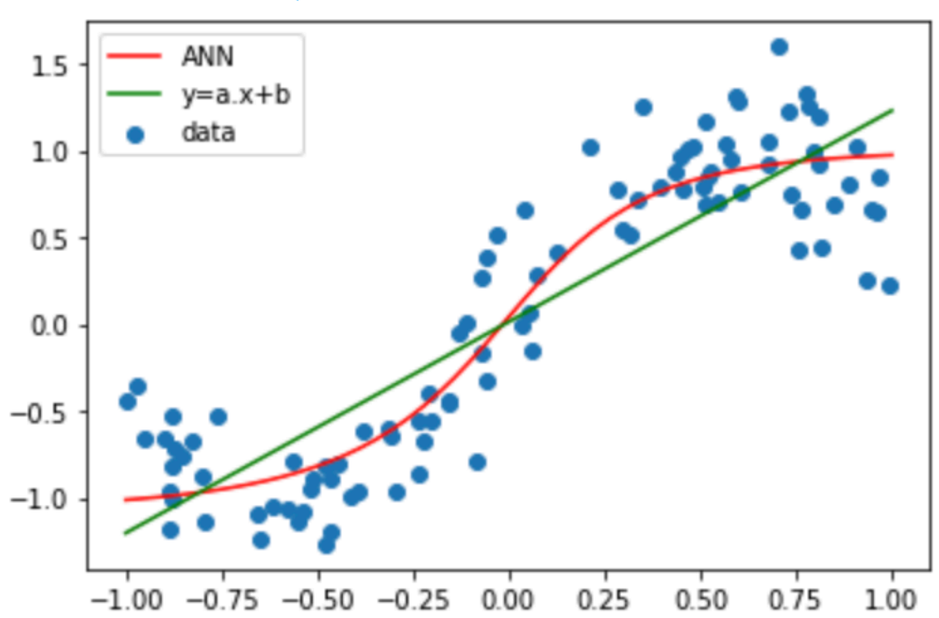
Motivated by large and complex datasets

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



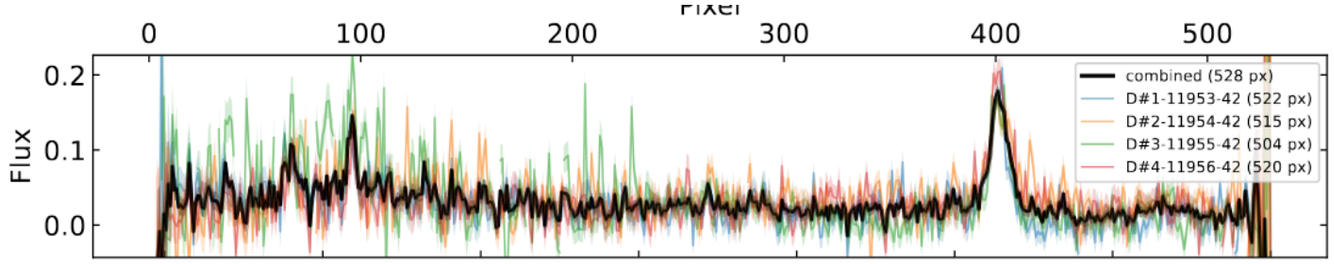


p is derived from physical insight

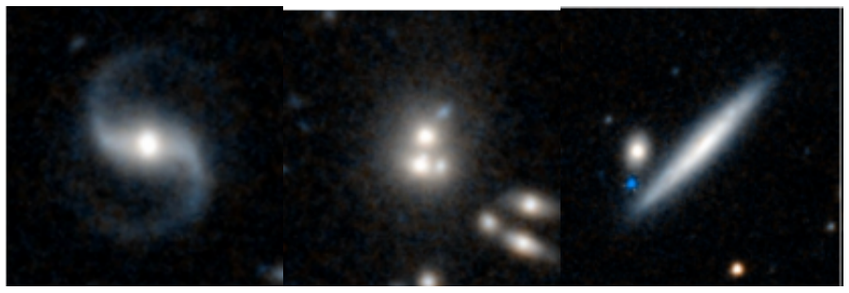


p is data driven, no physics

1. “Classical” DL: supervised Learning

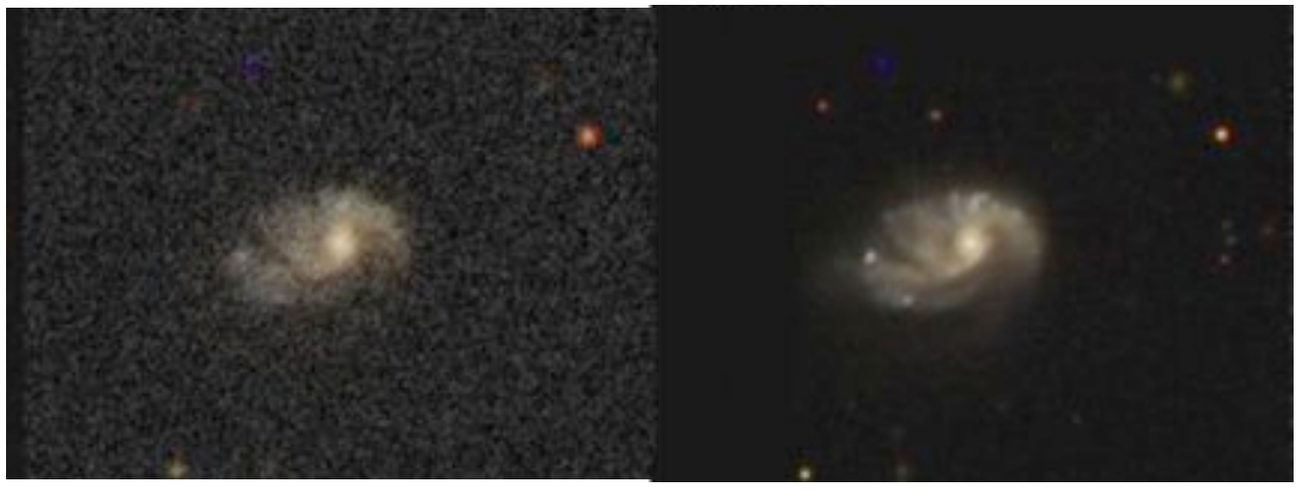


Q1: Euclid Collaboration: Y. Copin+2025



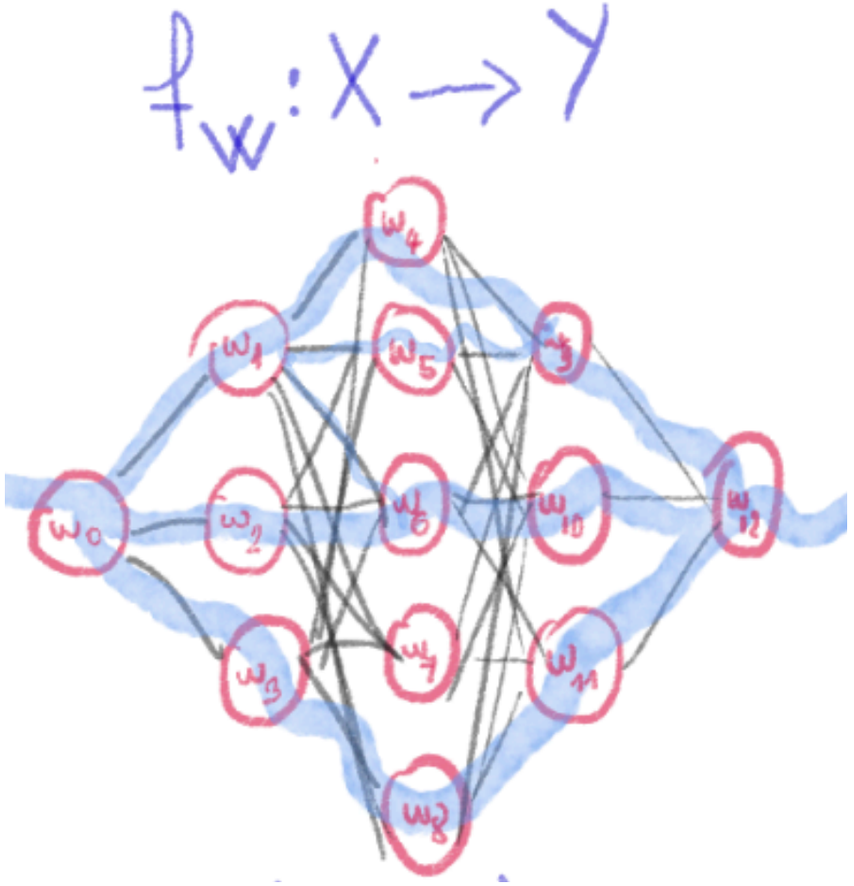
adaped from Q1: Euclid Collaboration: M. Walmsley, M. Huertas-Company+25

adaped from Schawinski et al. (2017)

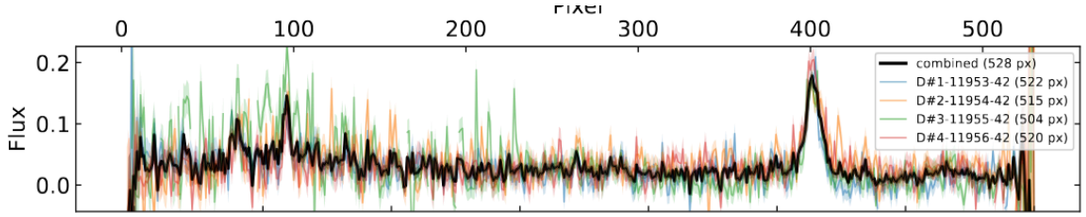


1. “Classical” DL: supervised Learning

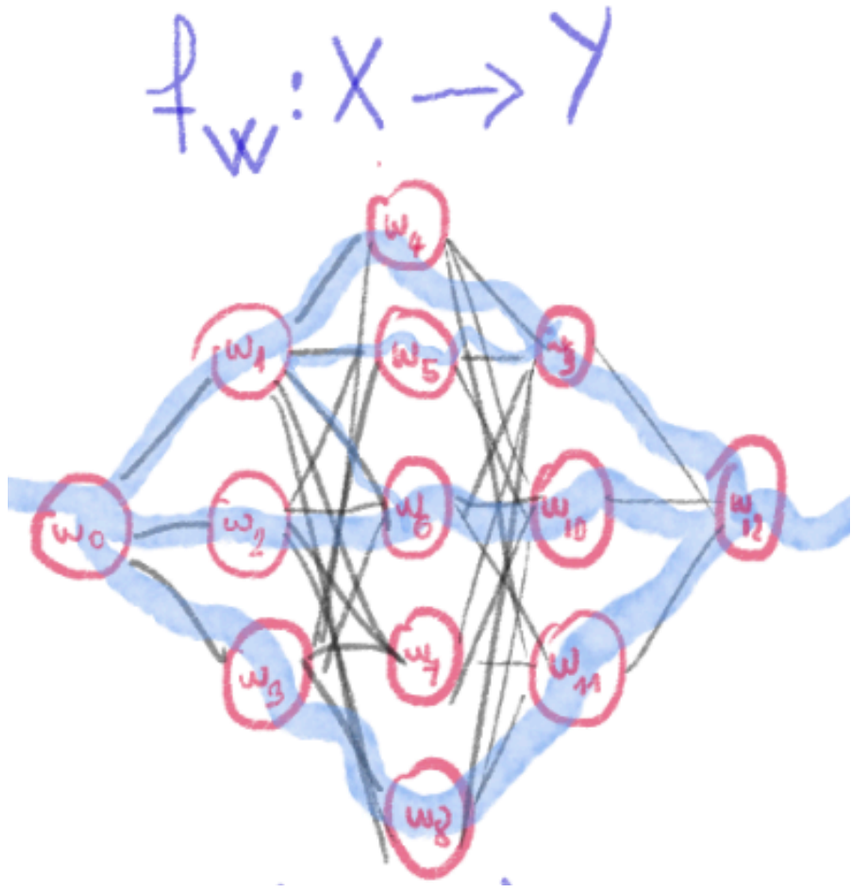
Given an input data (X), and labels (Y), train a parametrical function to map X to Y, validating on data not seen during training



Parameter Estimation



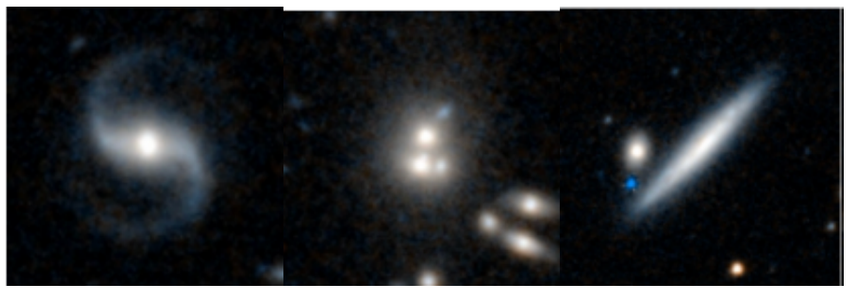
Q1: Euclid Collaboration: Y. Copin+2025



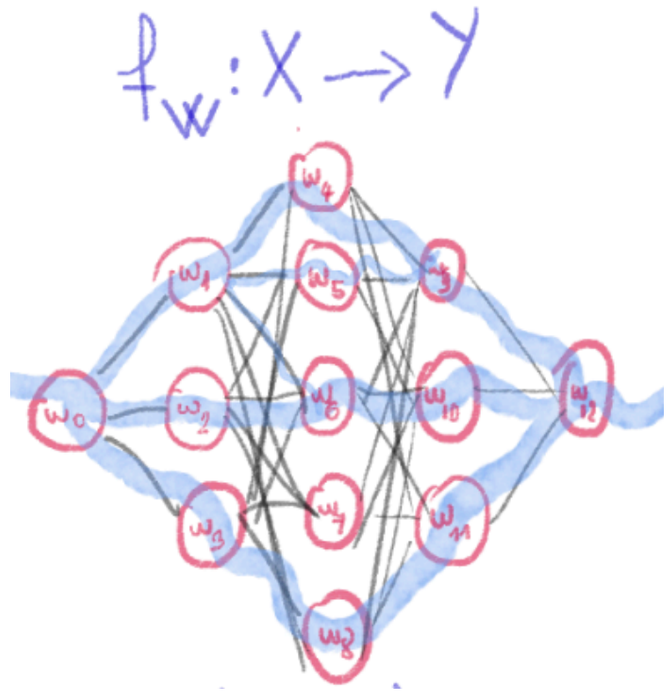
redshift

Classification

1 2 3

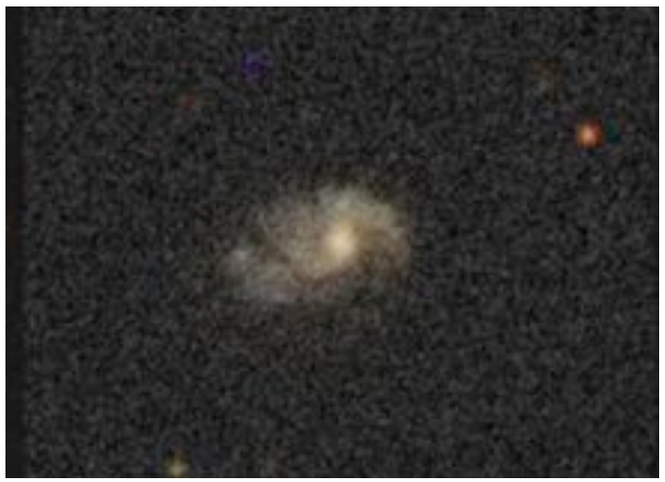


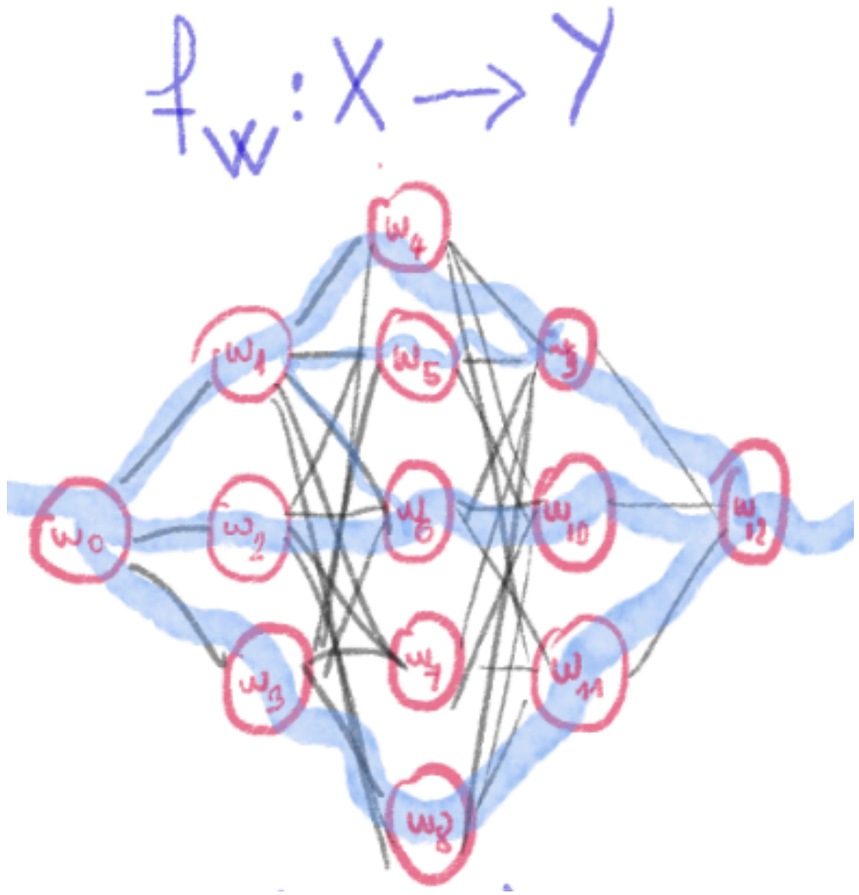
adaped from Q1: Euclid Collaboration: M. Walmsley, M. Huertas-Company+25

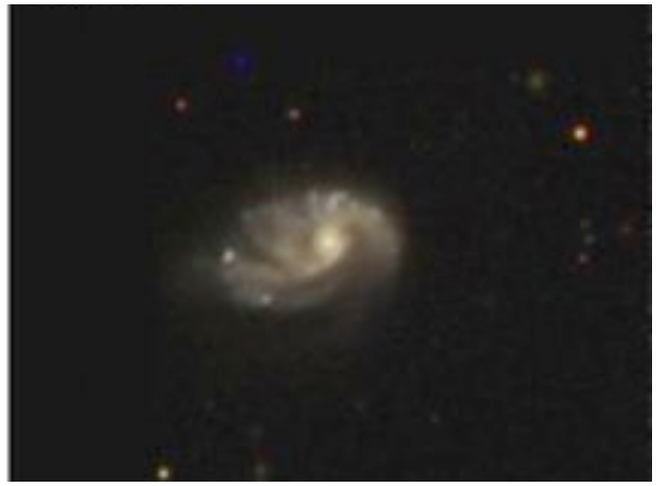


Id	Spiral	Ellipitcal	Irr.
1	0.9	0.05	0.05
2	0.05	0.15	0.8
3	0.25	0.7	0.05

Image enhancement







adaped from Schawinski et al. (2017)

How ?

➡ Define your **dataset**, input X, ground truth/labels Y

➡ Define your **architecture/model**, $f_{\mathscr{W}} : x \rightarrow \hat{y}$

➡ Define the **loss** of your model: $\mathscr{L} = \hat{y} - y$

Hyper-parameters definition

➡ **Predict** a batch of data: $\hat{y} = f(x)$

➡ Compute the **loss** on the batch

➡ Use **stochastic gradient descent** to update your weights: $\mathscr{W}_{t+1} = \mathscr{W}_t + \lambda \frac{\partial \mathscr{L}}{\partial \mathscr{W}_t}$

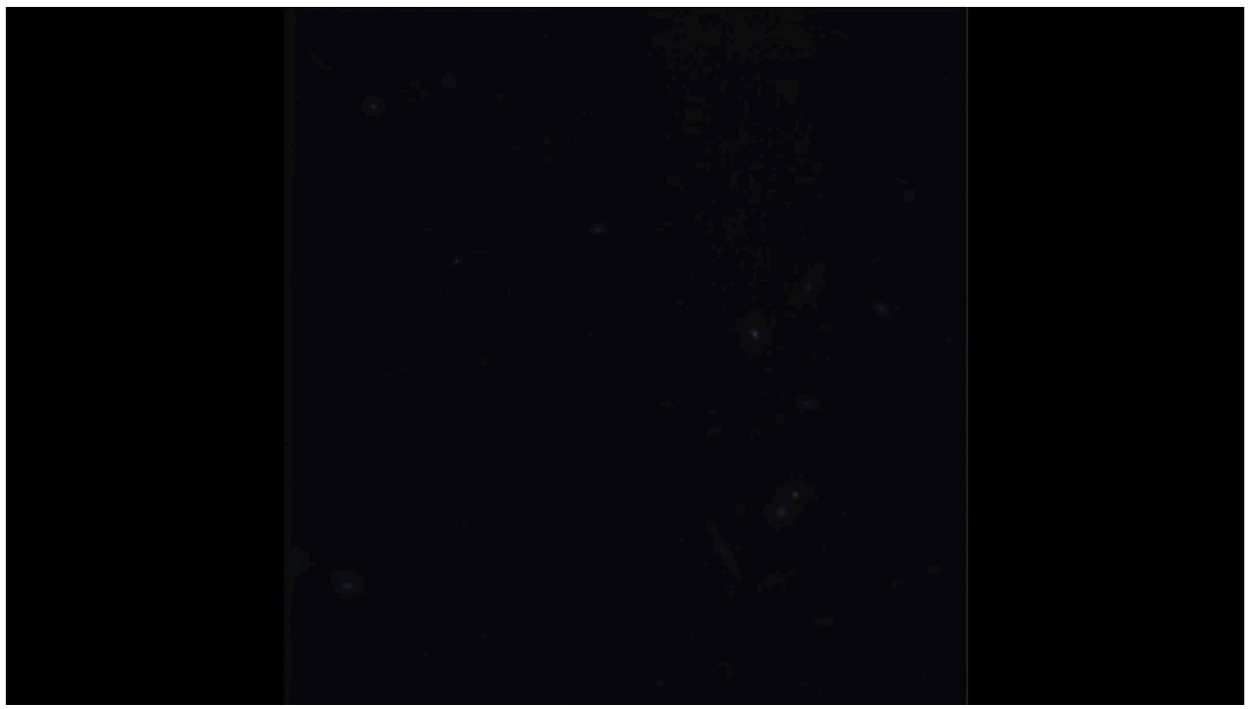
➡ Repeat until **convergence**

Training

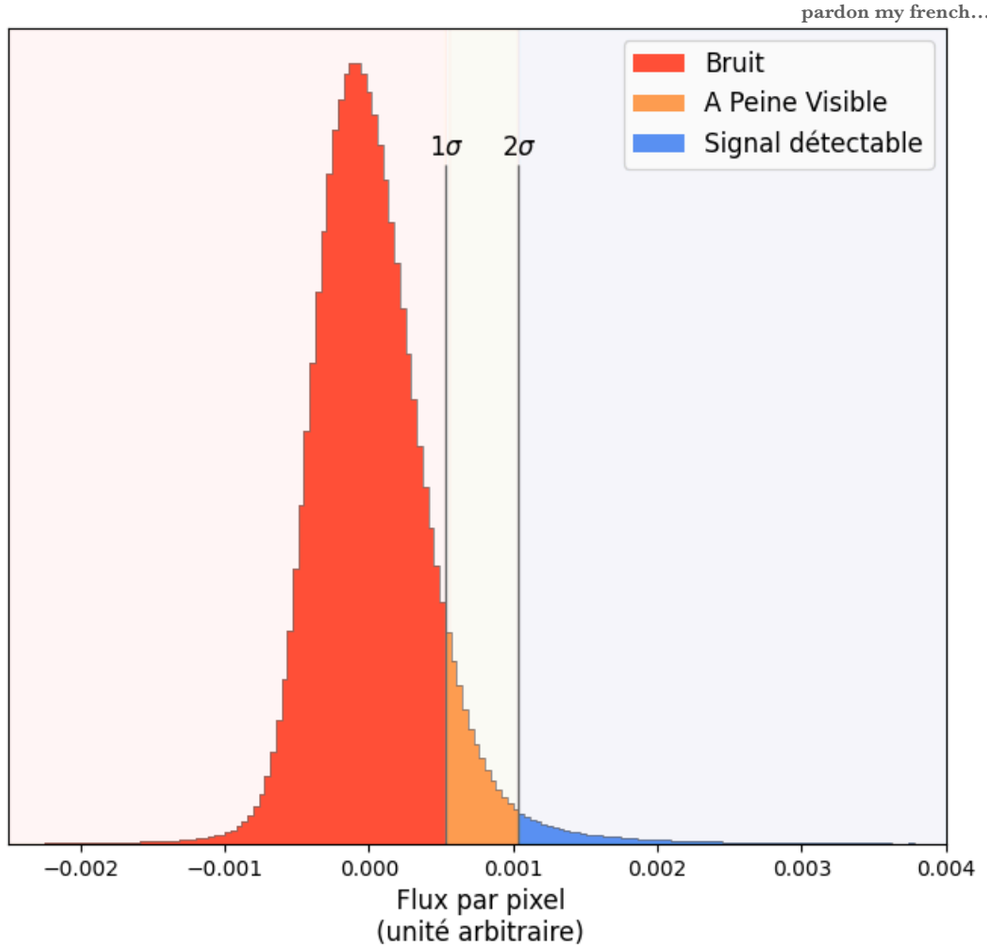
➡ Test the **accuracy** on some test data

Testing

1- Your Data (usually different from “every-day” life data



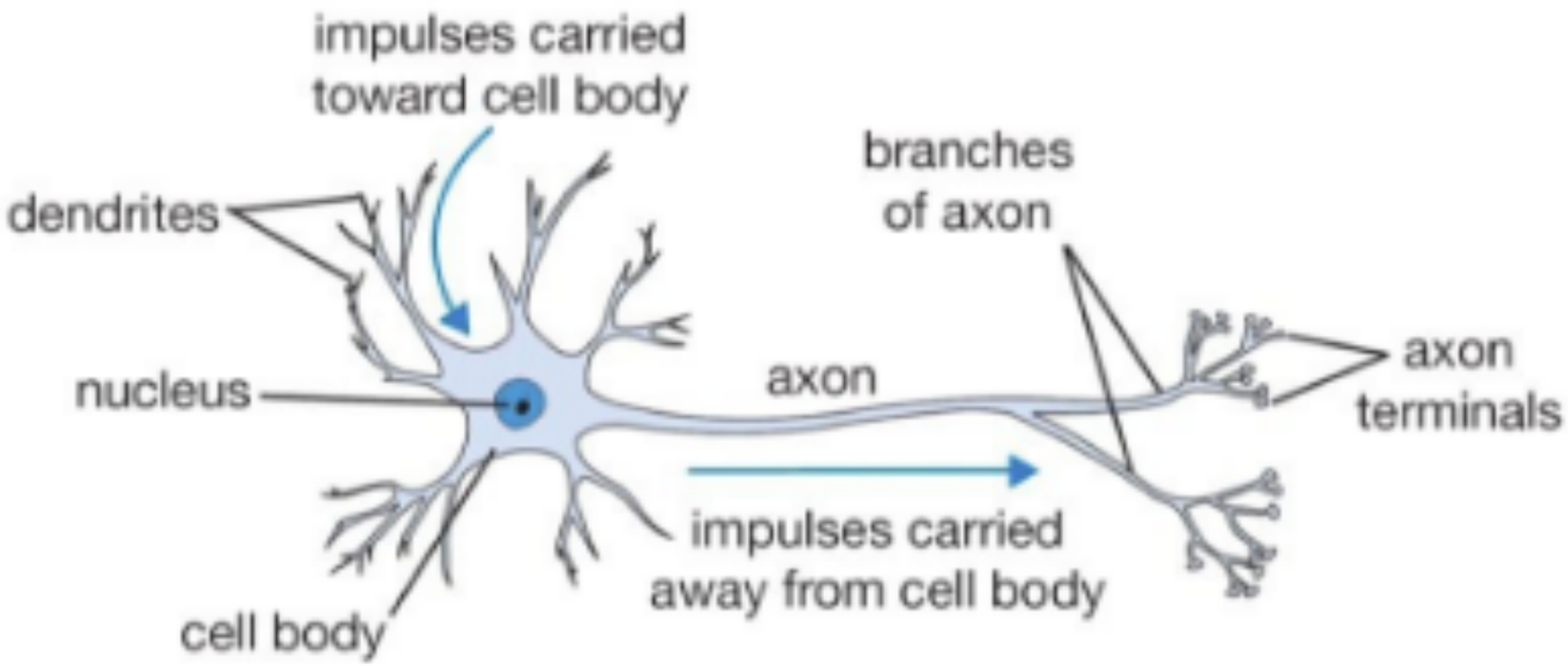
pardon my french…



Normalisation / dynamic range / cleaning / gold sample…

2- Your Model

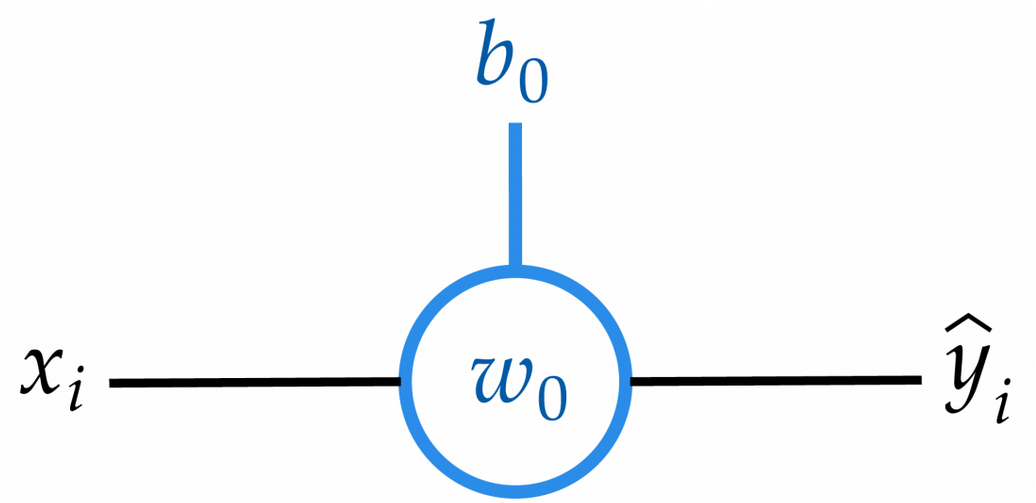
Inspired by neuro-science



2- Your Model

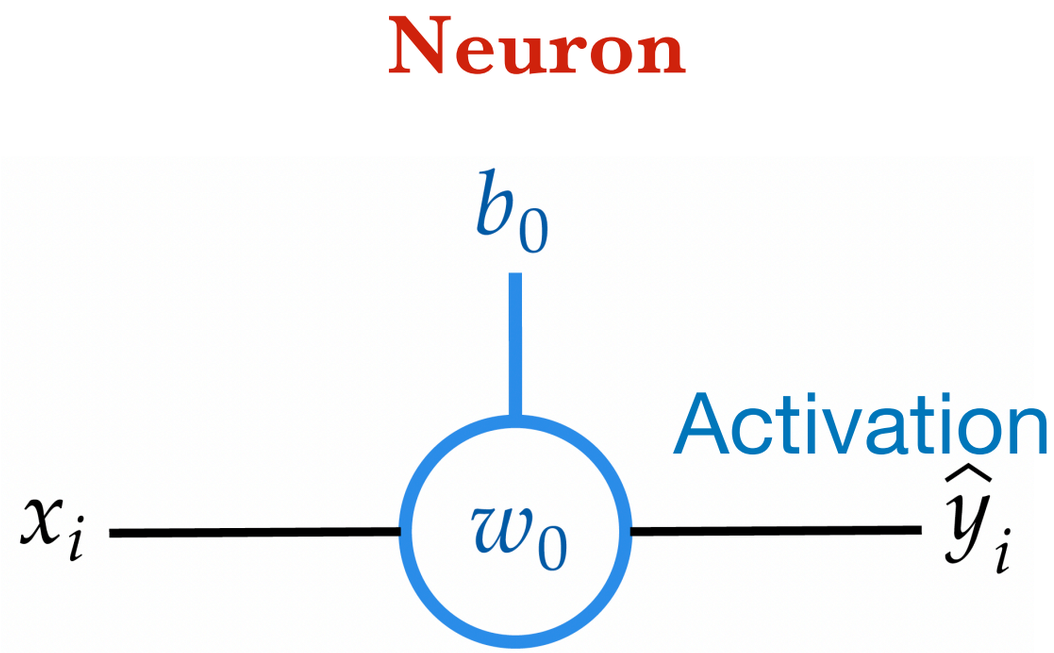
$$f(x) = w_0\, x + b_0$$

Neuron (almost)



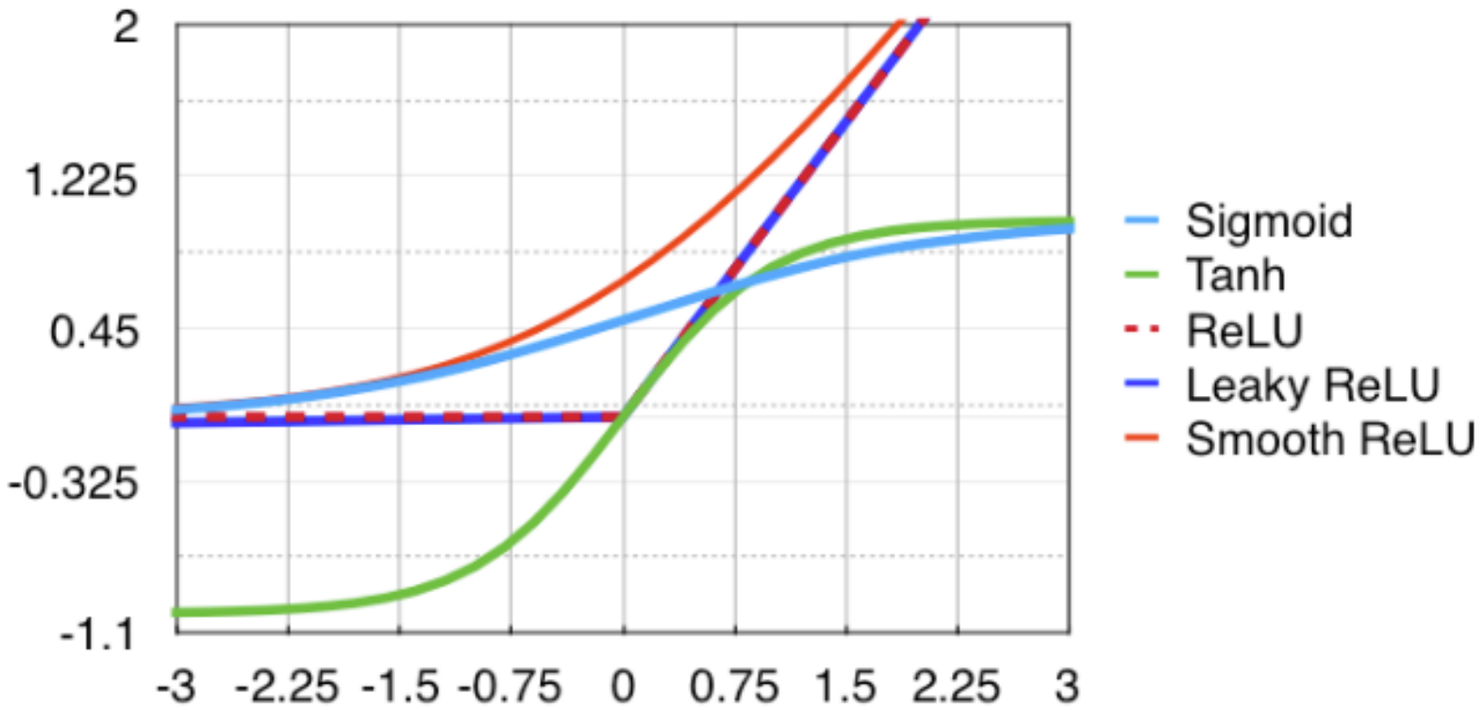
2- Your Model

$$f(x) = \mathscr{A}(w_0\, x + b_0)$$



2- Your Model

$$f(x) = \mathscr{A}(w_0\, x + b_0)$$



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

ReLu: $f(x) = max(0, x)$ Soft ReLu: $f(x) = log(1+e^{x})$

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

3- Your Loss

$$\mathscr{L} = Y - \hat{Y}$$

3- Your Loss

$$\mathscr{L} = Y - \hat{Y}$$

$$\mathscr{L} = \sum_{i}^{N} y_i - \hat{y}_i$$

3- Your Loss

$$\mathscr{L} = Y - \hat{Y}$$

$$\mathscr{L} = \sum_{i}^{N} y_i - \hat{y}_i$$

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

OBSERVED DATASET

3- Your Loss

$$\mathscr{L} = Y - \hat{Y}$$

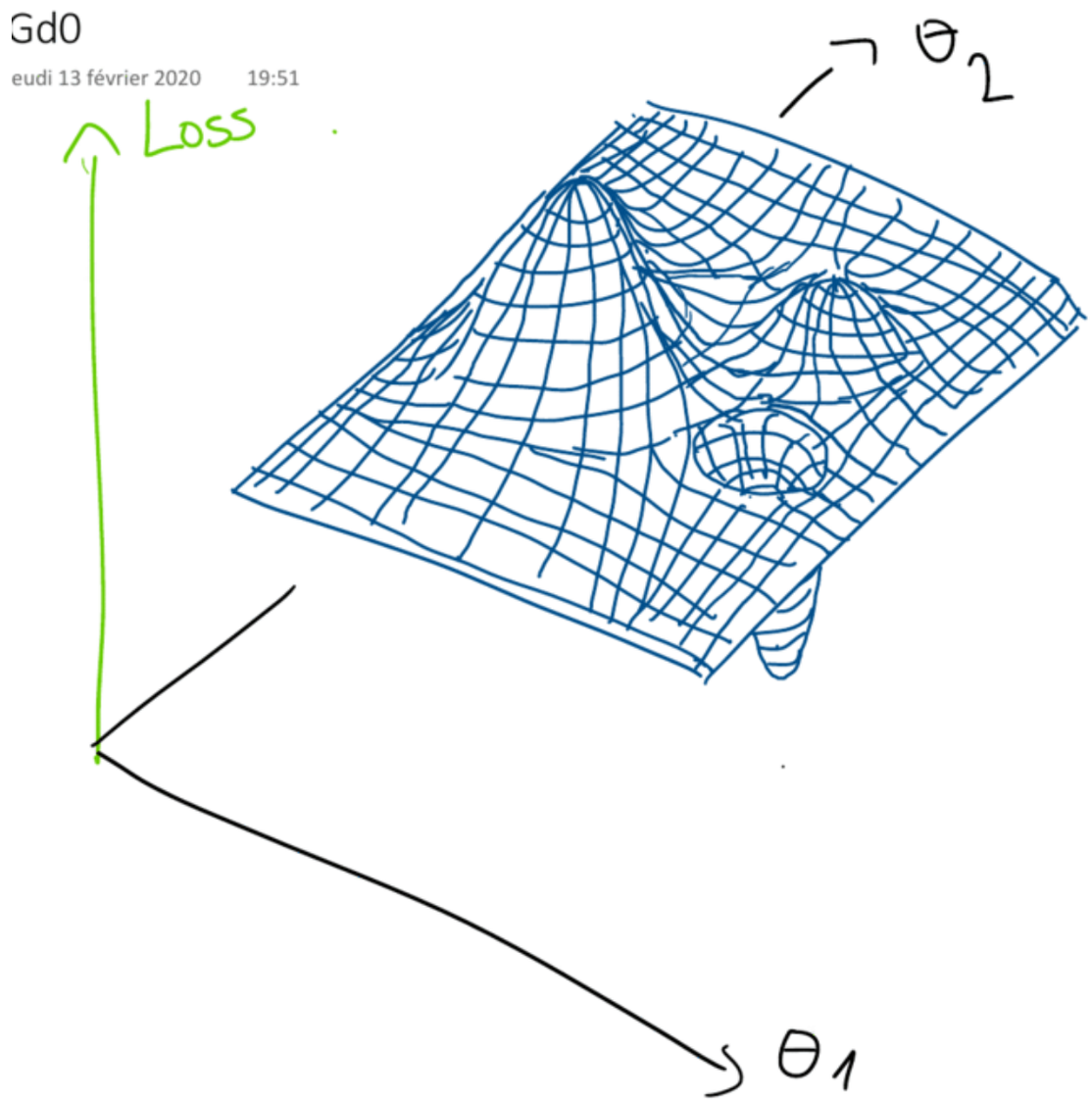
$$\mathscr{L} = \sum_{i}^{N} y_i - \hat{y}_i$$

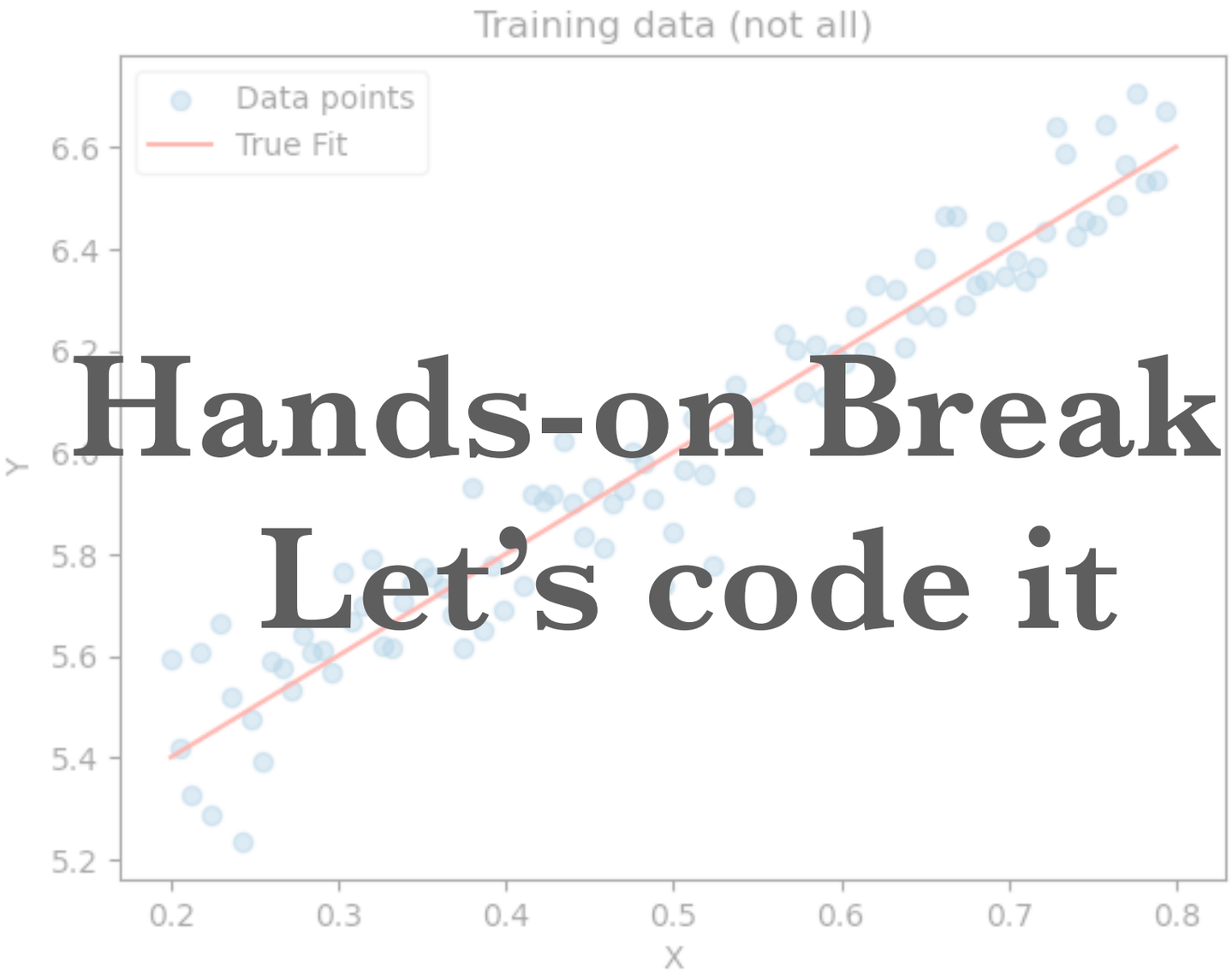
WE ARE MINIMISING WITH RESPECT TO <u>A FINITE NUMBER OF OBSERVED EXAMPLES</u>

ALL “GALAXIES IN THE UNIVERSE”

OBSERVED DATASET

4- Optimisation: Gradient Descent

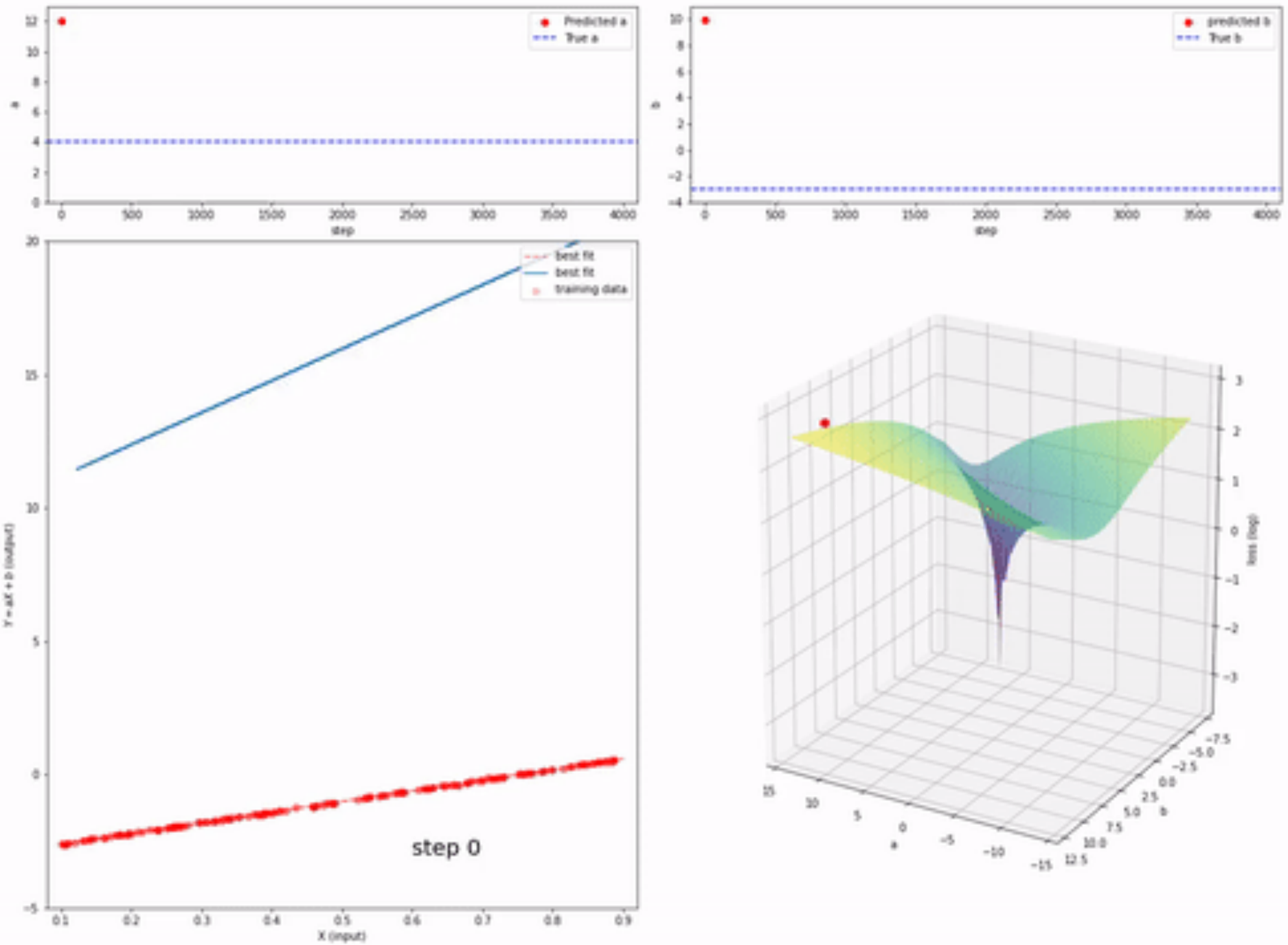




Hands-on Break !

Let’s code it

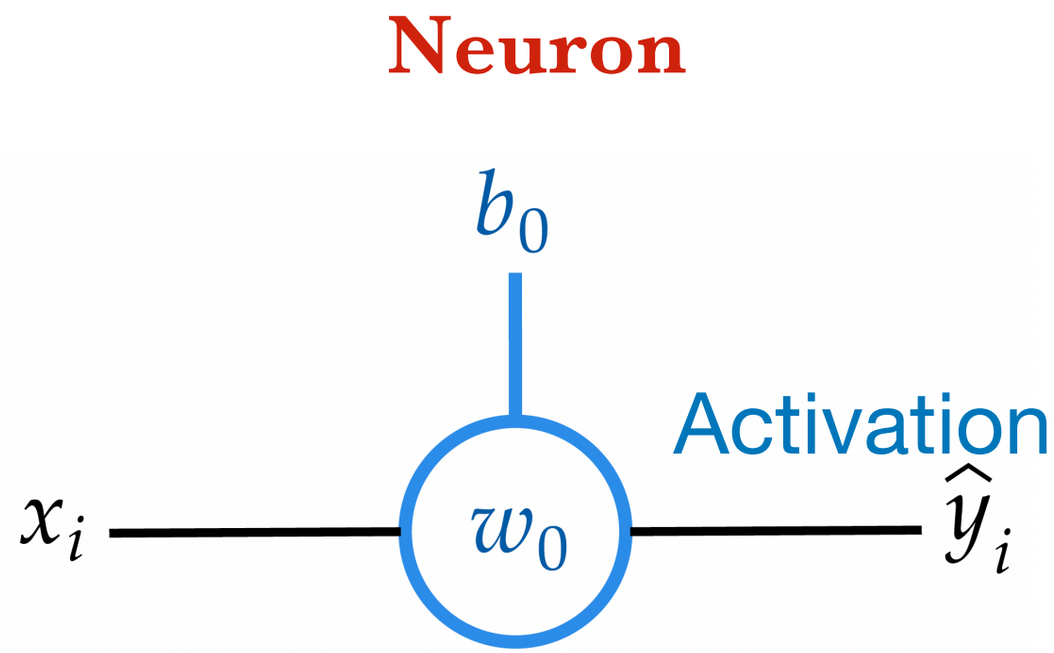
Bonus: visualisation



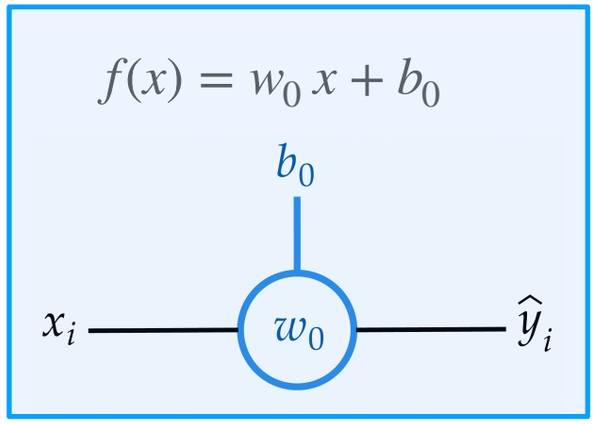
Let’s go a bit deeper

2- Your Model

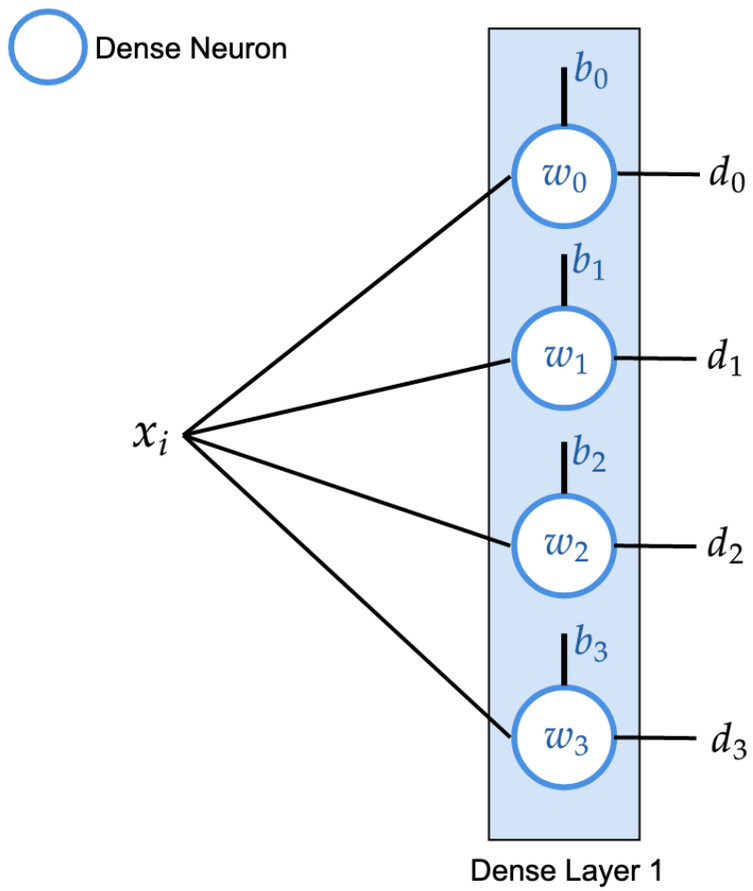
$$f(x) = \mathscr{A}(w_0\, x + b_0)$$



2- Your Model: stack neuron verticaly

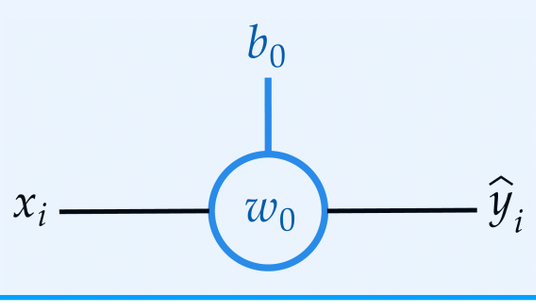


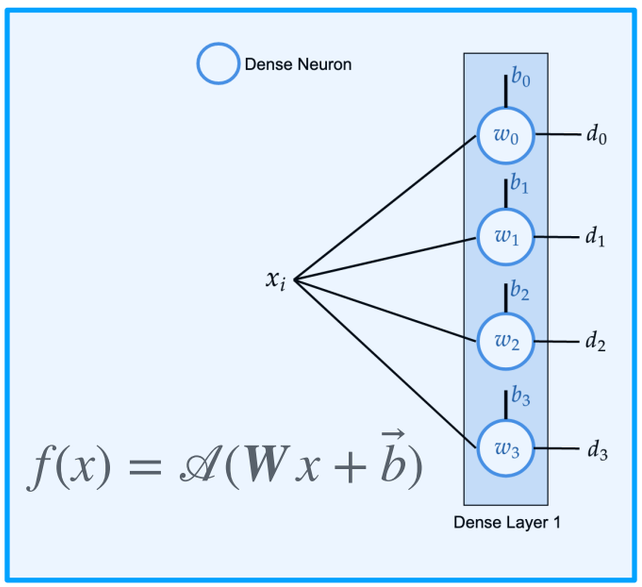
$$f(x) = \mathcal{A}(\boldsymbol{W}\, x + \vec{b})$$

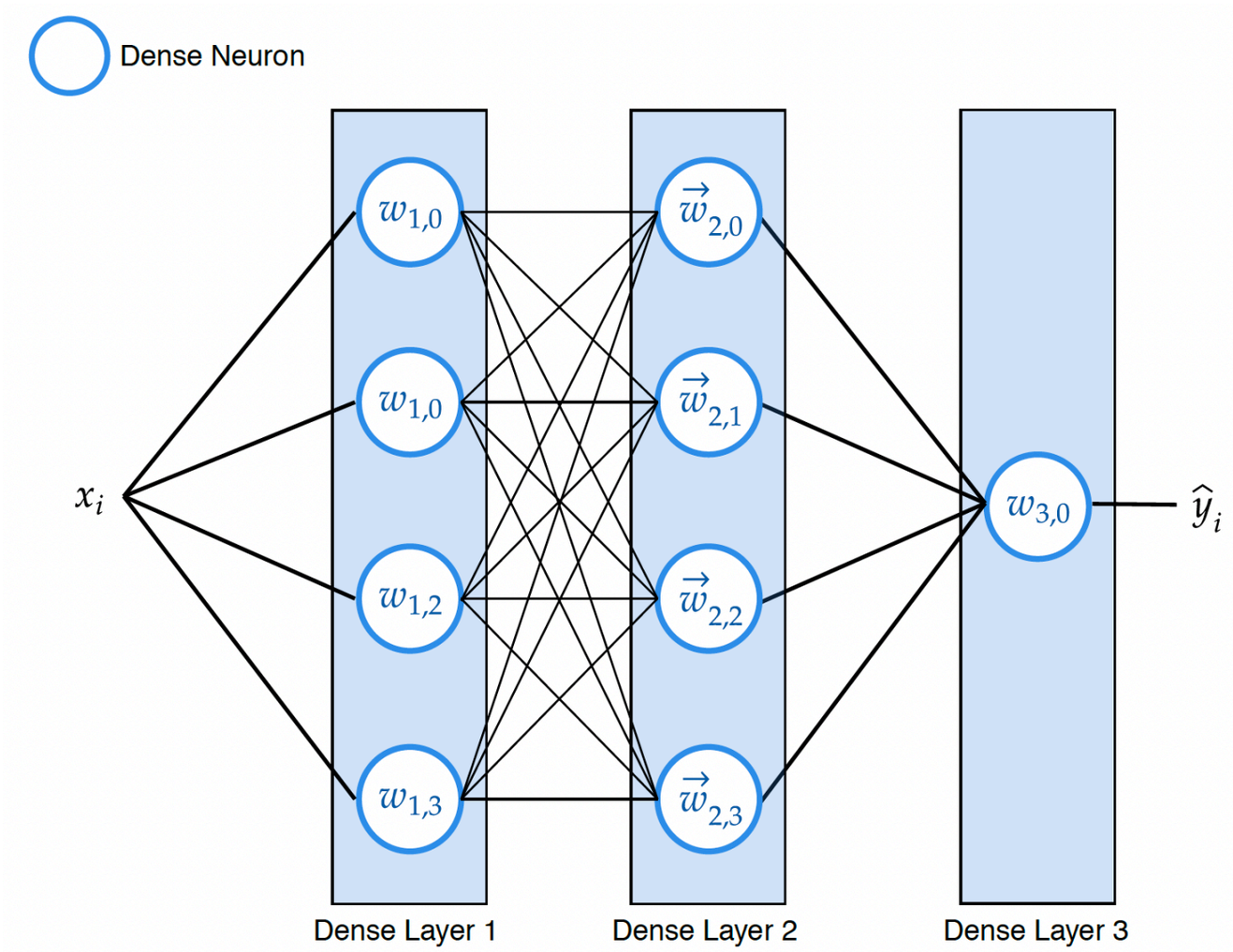


2- Your Model: stack neuron horizontally: layers

$$f(x) = w_0\, x + b_0$$

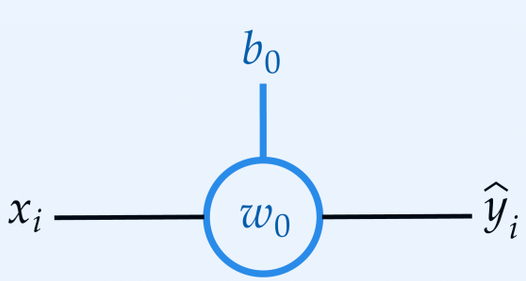


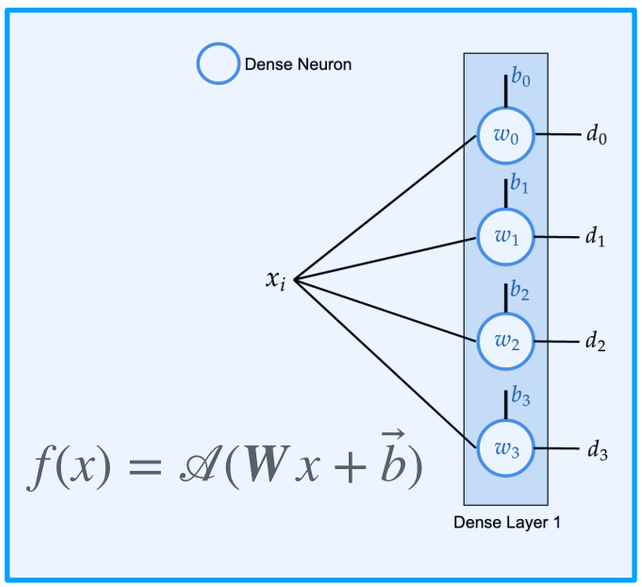


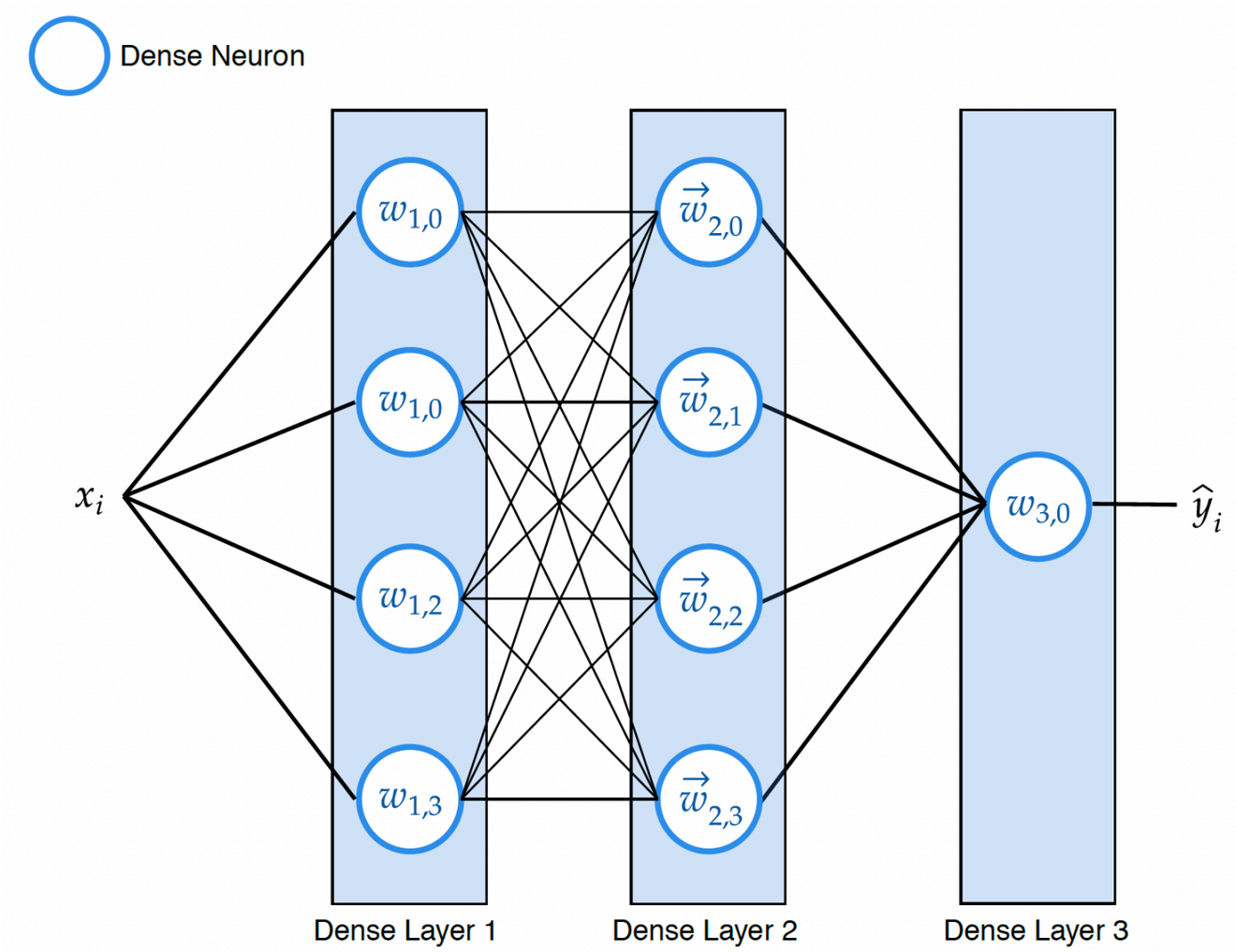


2- Your Model: stack neuron horizontally: layers

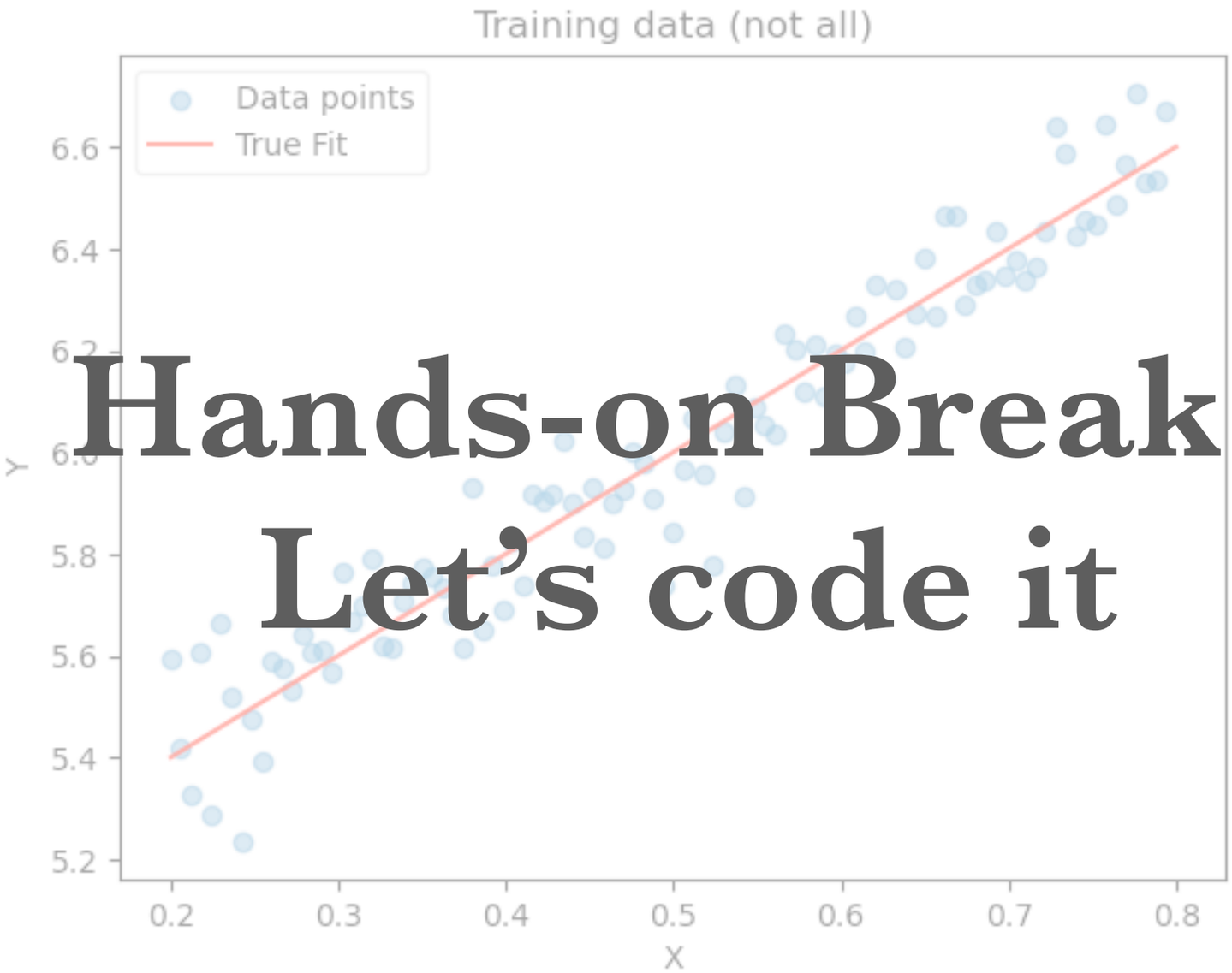
$$f(x) = w_0\, x + b_0$$







$$f(x) = \mathscr{A}\left(\boldsymbol{W_1}\,\mathscr{A}\left(\boldsymbol{W_1}\,\mathscr{A}(\boldsymbol{W_0} + \vec{b_0}) + \vec{b_1}\right) + \vec{b_2}\right)$$



Hands-on Break !

Let’s code it

UNIVERSAL APPROXIMATION THEOREM

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

Cybenko+89

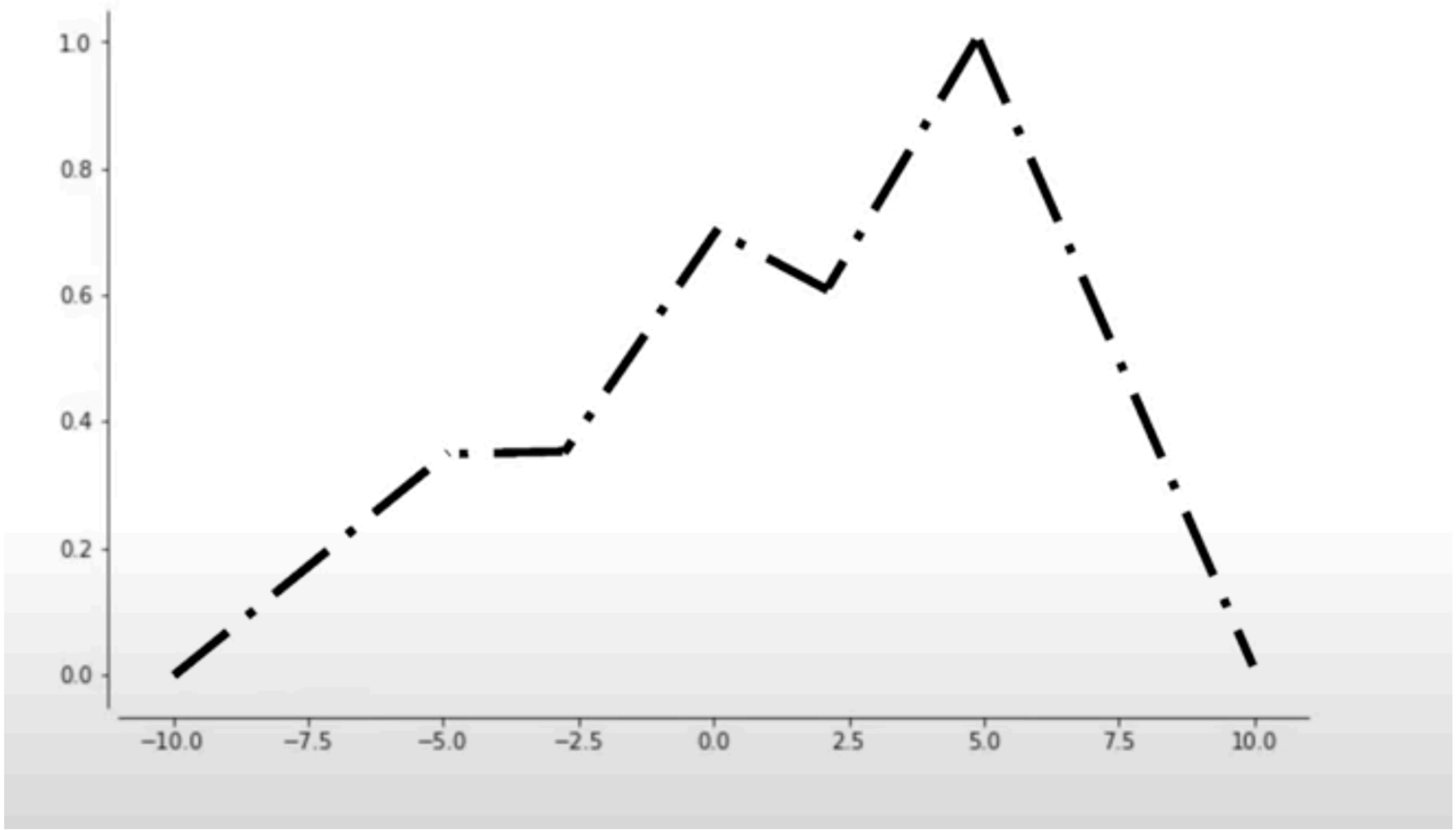
"BIG ENOUGH NETWORK CAN APPROXIMATE, BUT NOT REPRESENT ANY SMOOTH FUNCTION. THE MATH DEMONSTRATION IMPLIES SHOWING THAT NETWORS ARE DENSE IN THE SPACE OF TARGET FUNCTIONS"

UNIVERSAL APPROXIMATION THEOREM

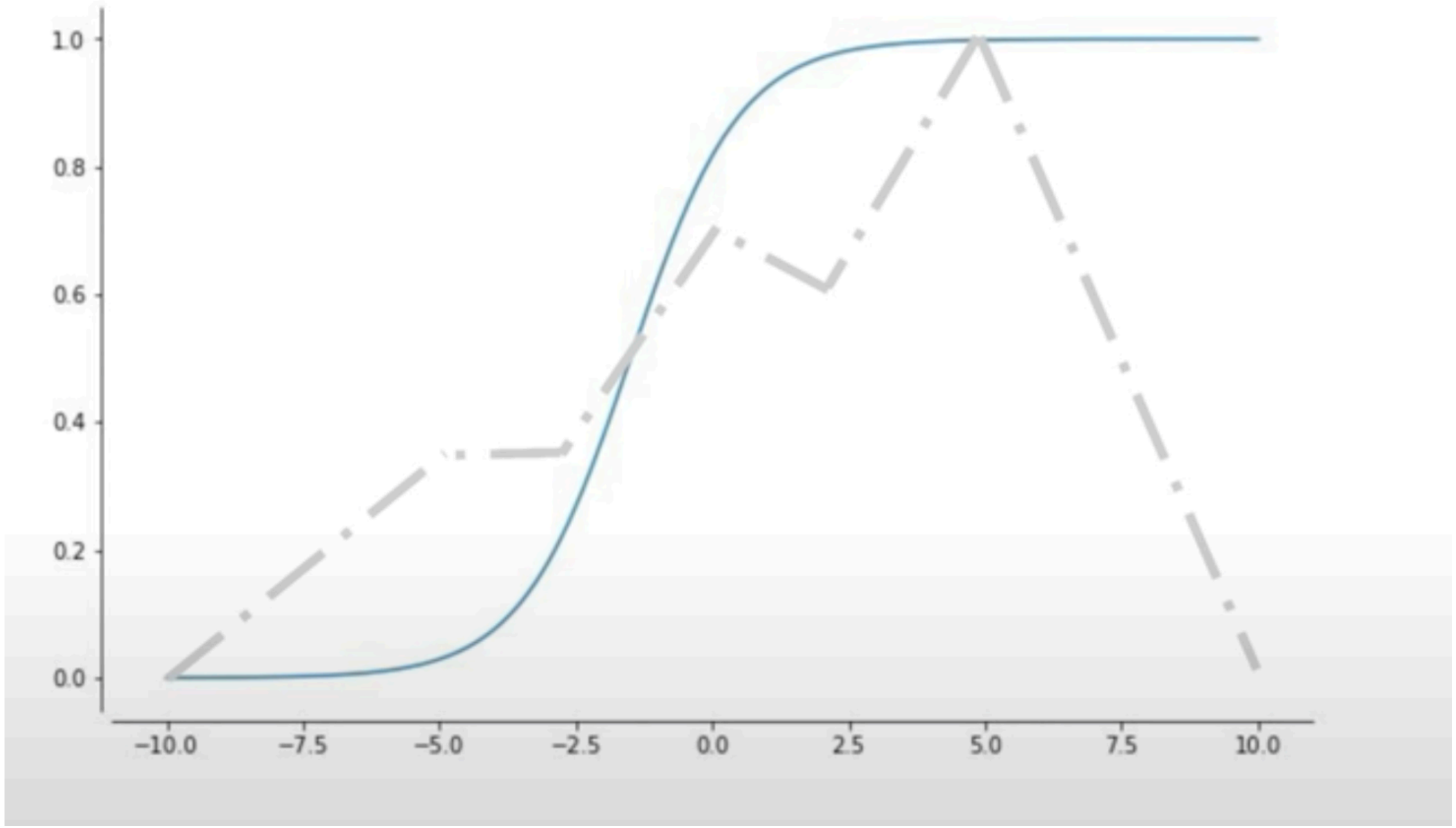
FOR ANY CONTINUOS FUNCTION FOR A HYPERCUBE [0,1]d TO REAL NUMBERS, NON-CONSTANT, BOUNDED AND CONTINUOUS ACTIVATION FUNCTION f, AND EVERY POSITIVE EPSILON, THERE EXISTS A1-HIDDEN LAYER NEURAL NETWORK USING f THAT OBTAINS AT MOST EPSILON ERROR IN FUNCTIONAL SPACE

Horvik+91

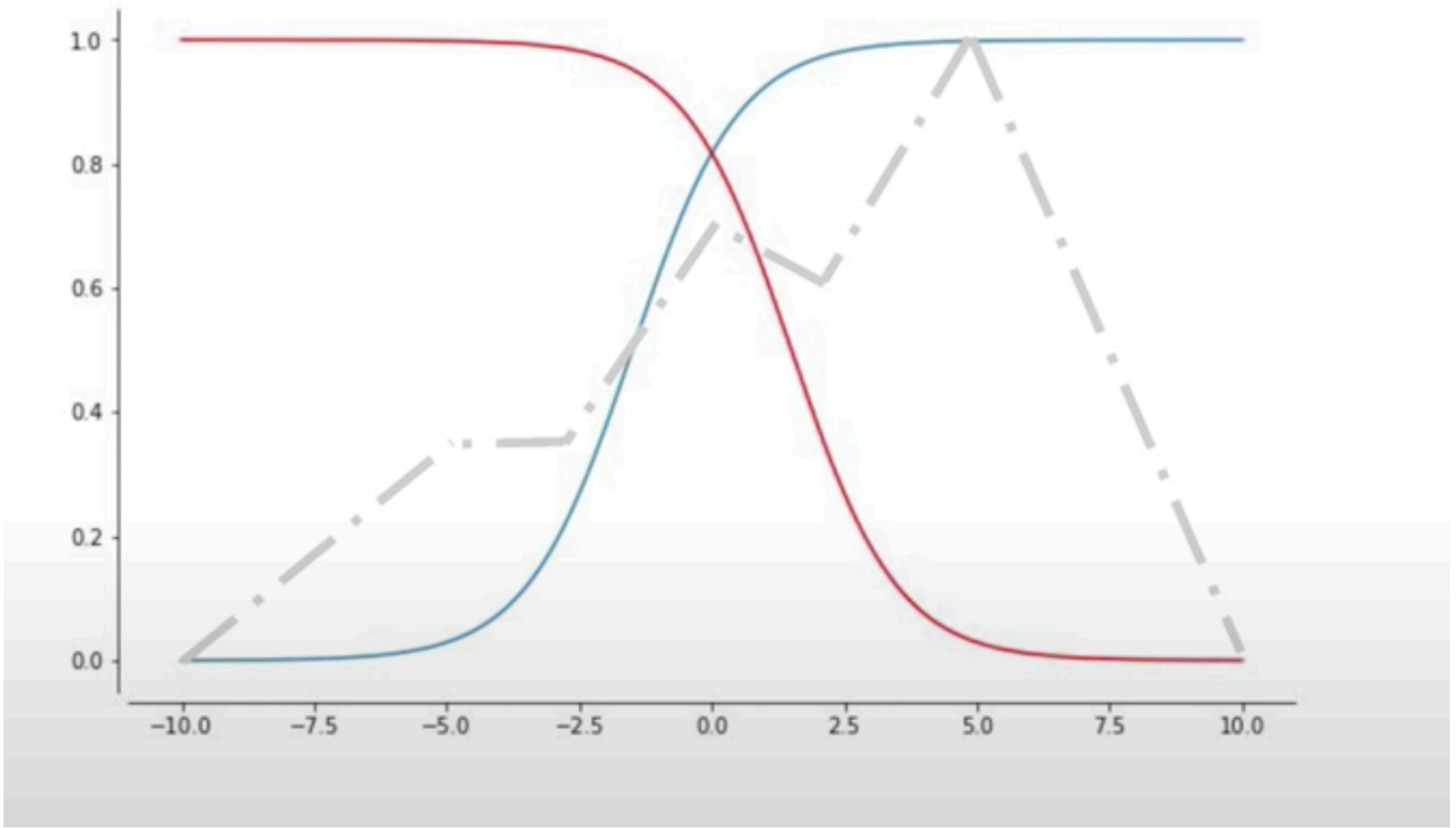
“BIG ENOUGH NETWORK CAN APPROXIMATE, BUT NOT REPRESENT ANY SMOOTH FUNCTION. THE MATH DEMONSTRATION IMPLIES SHOWING THAT NETWORS ARE DENSE IN THE SPACE OF TARGET FUNCTIONS”



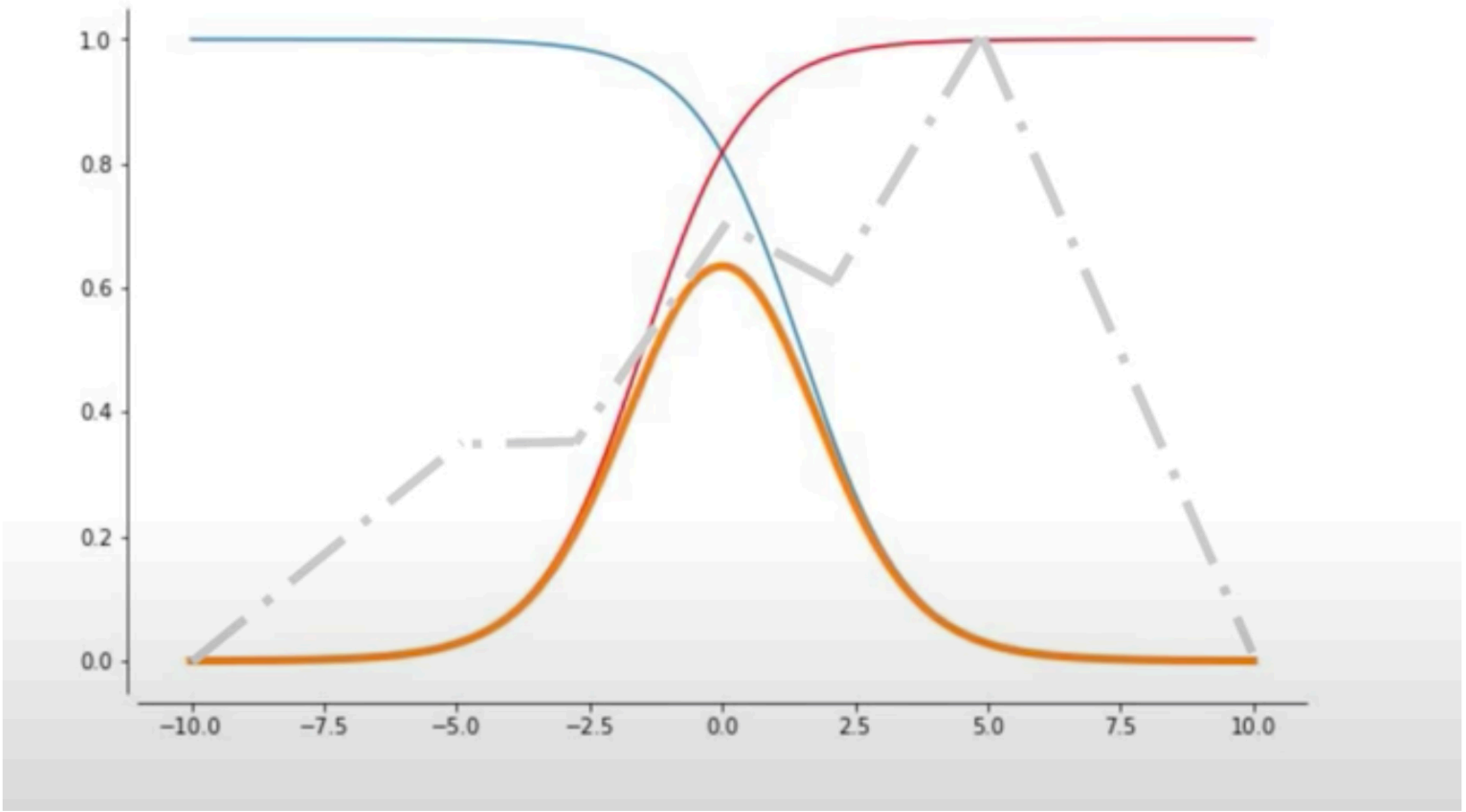
(deepmind @Czarnecki)



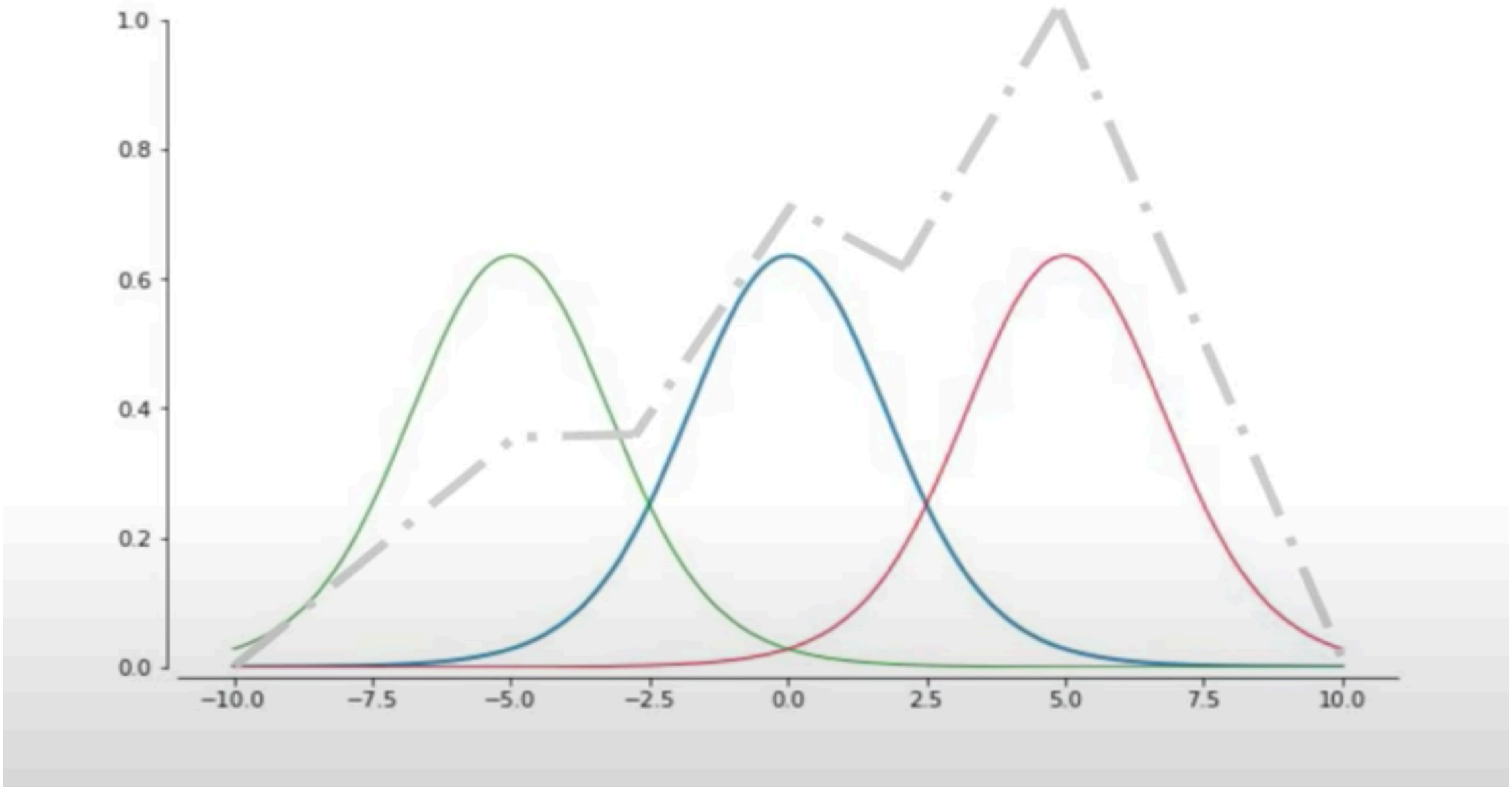
(deepmind @Czarnecki)



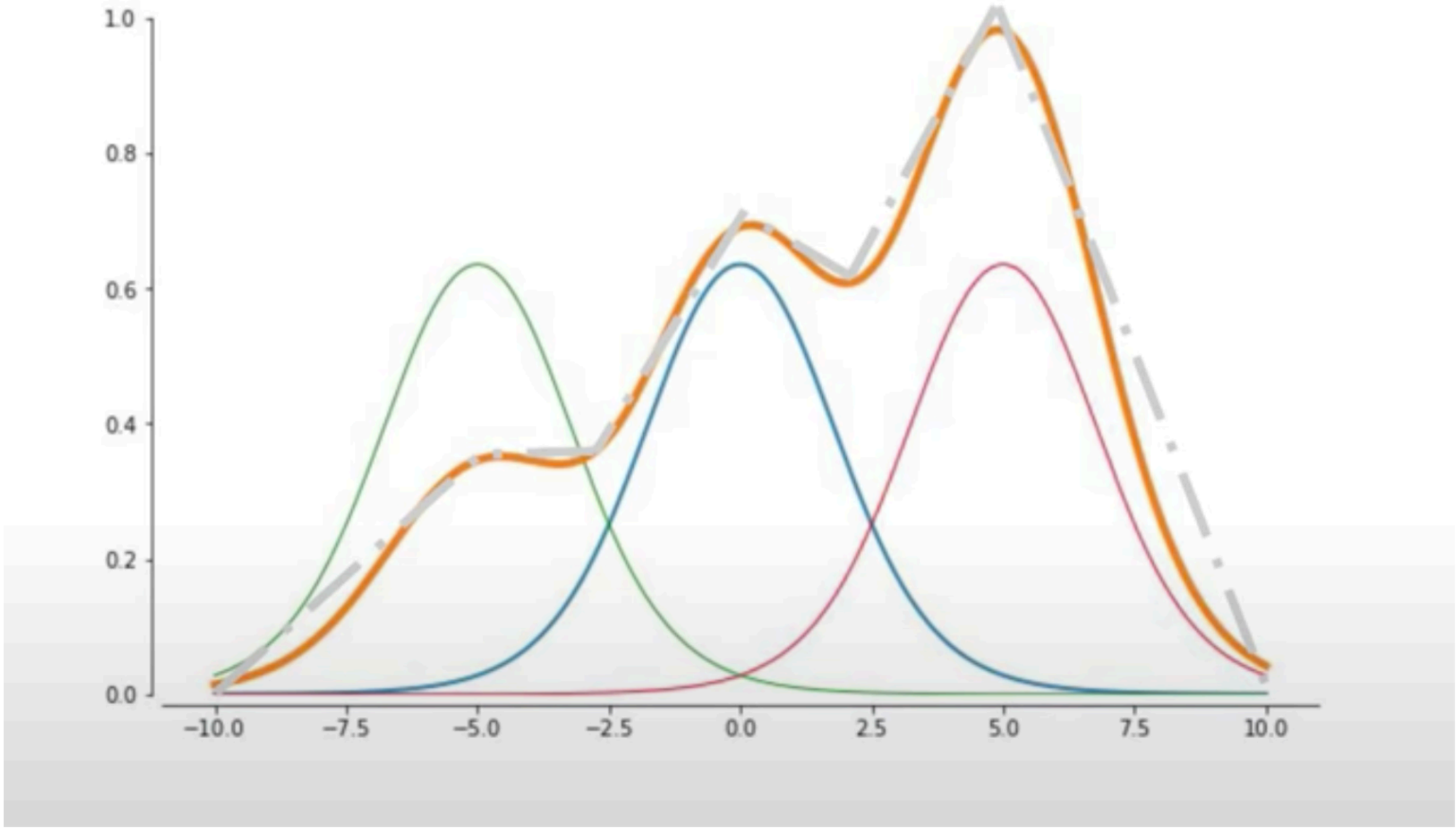
(deepmind @Czarnecki)



(deepmind @Czarnecki)

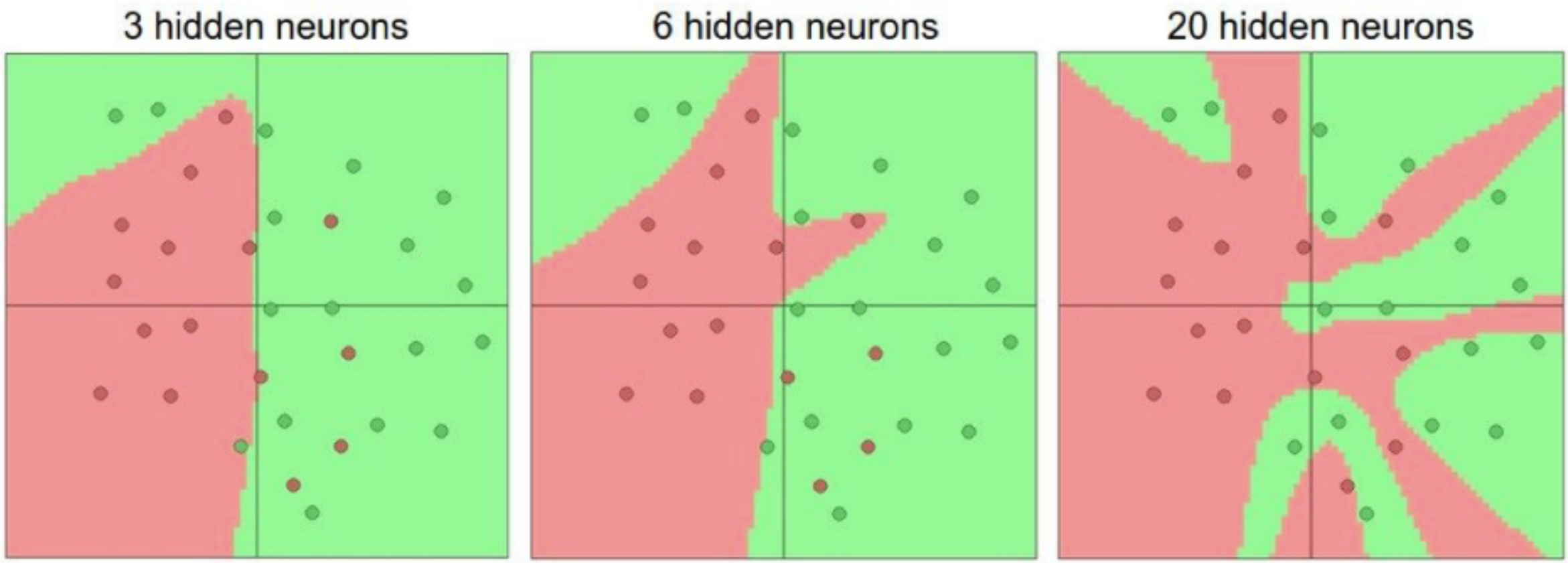


(deepmind @Czarnecki)



(deepmind @Czarnecki)

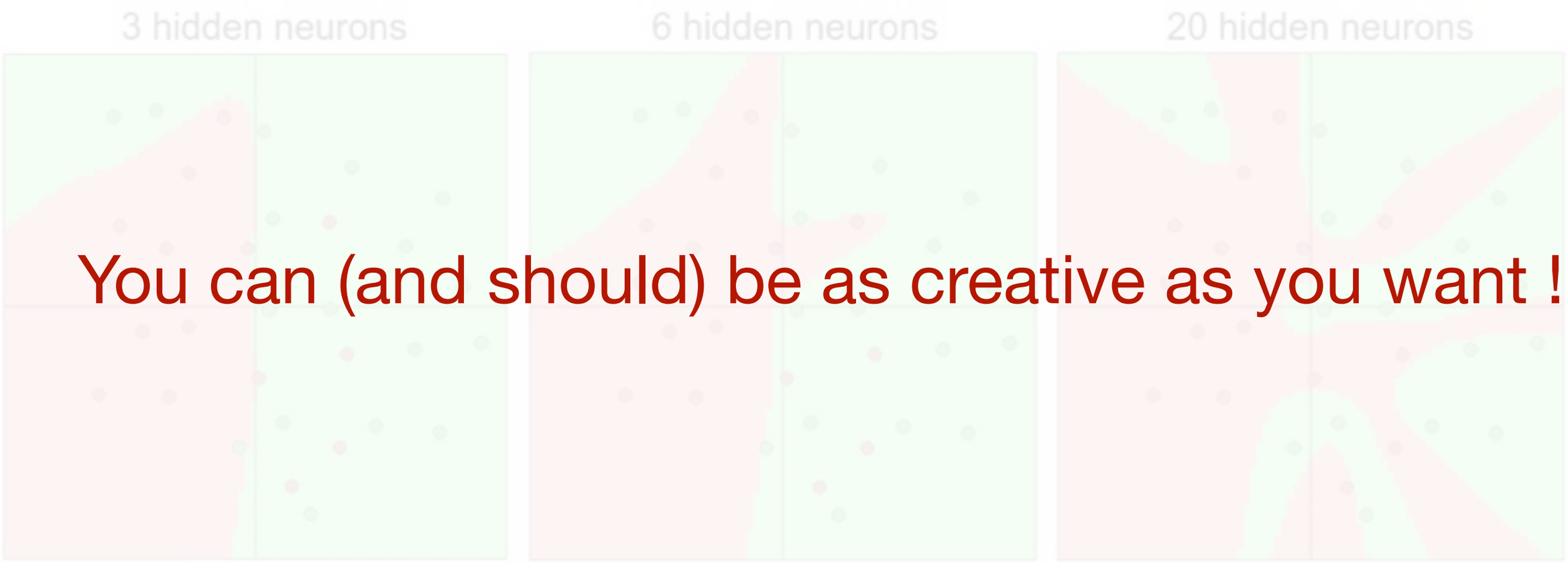
HIDDEN LAYERS ALLOW INCREASING COMPLEXITY



More complex functions allow increasing complexity

Credit: Karpathy

HIDDEN LAYERS ALLOW INCREASING COMPLEXITY

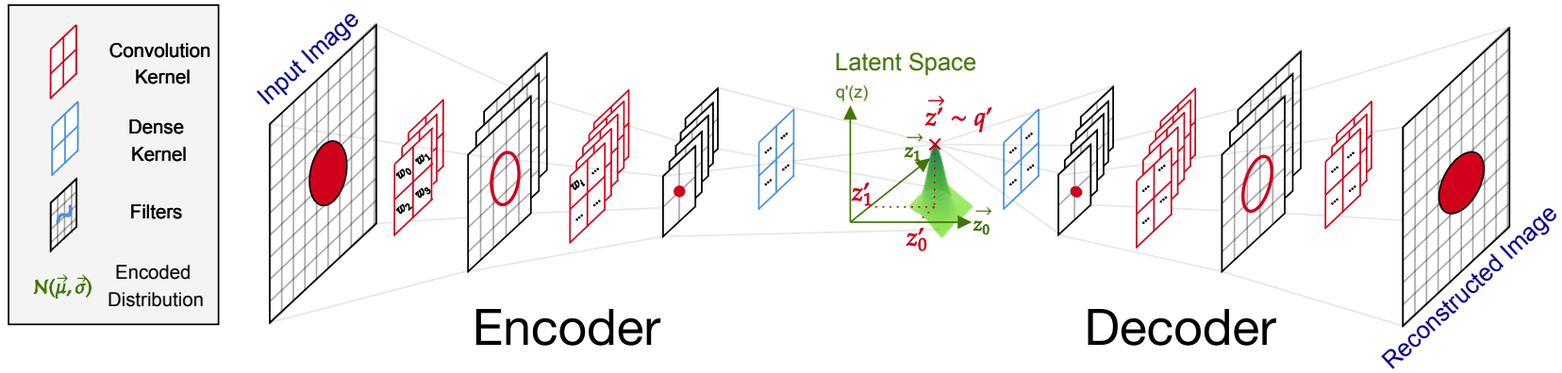


You can (and should) be as creative as you want !

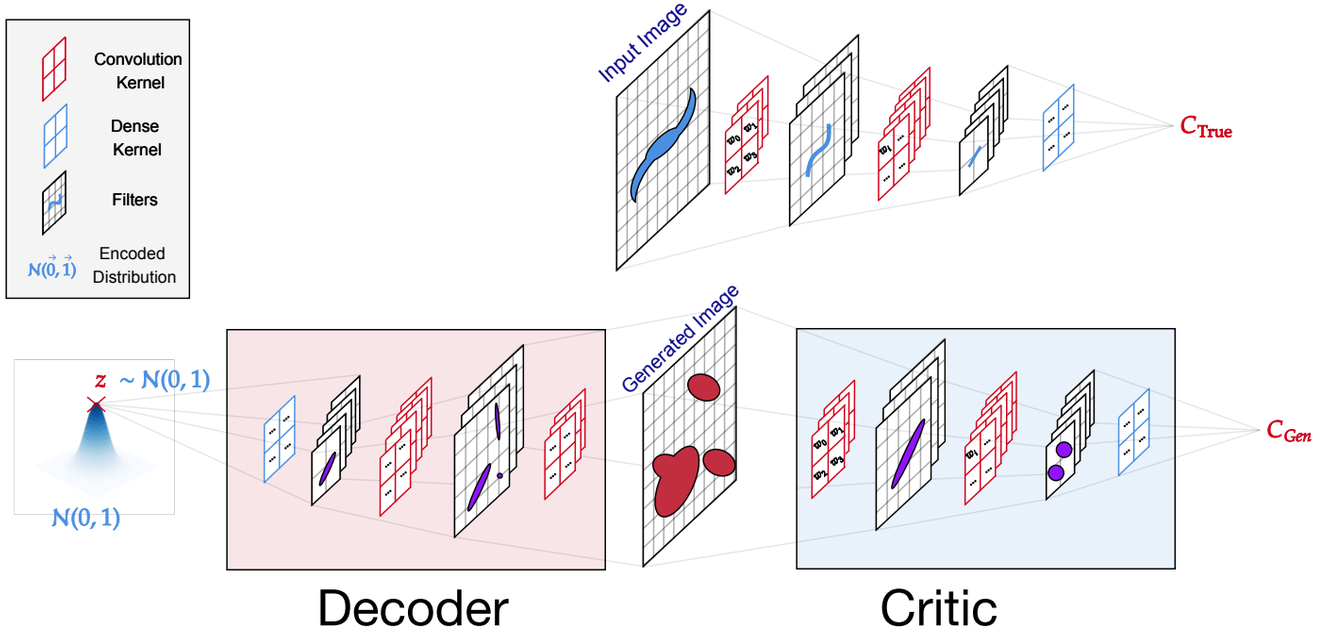
More complex functions allow increasing complexity

Credit: Karpathy

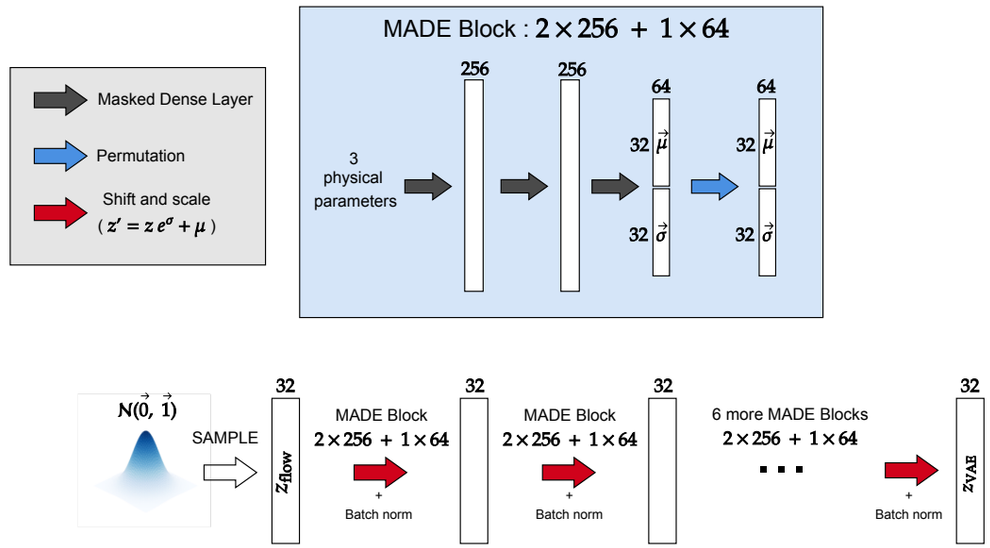
VAE



GAN



Flow (MADE)



3- Your Loss

$$\mathscr{L} = Y - Y$$

3- Your Loss

$$\mathscr{L} = Y - \hat{Y}$$

$$\mathscr{L} = |Y - \hat{Y}|^2$$

3- Your Loss

$$\mathscr{L} = Y - \hat{Y}$$

$$\mathscr{L} = |Y - \hat{Y}|^2$$

$$\mathscr{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}) + \log(1 - \hat{y}_i)$$

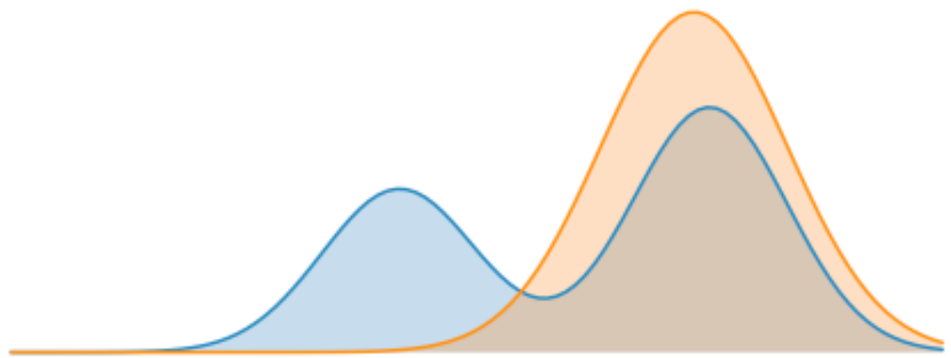
3- Your Loss

$$\mathscr{L} = Y - \hat{Y}$$

$$\mathscr{L} = |Y - \hat{Y}|^2$$

$$\mathscr{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}) + \log(1 - \hat{y}_i)$$

$$\mathscr{L} = \mathscr{P}(\hat{Y})$$



3- Your Loss

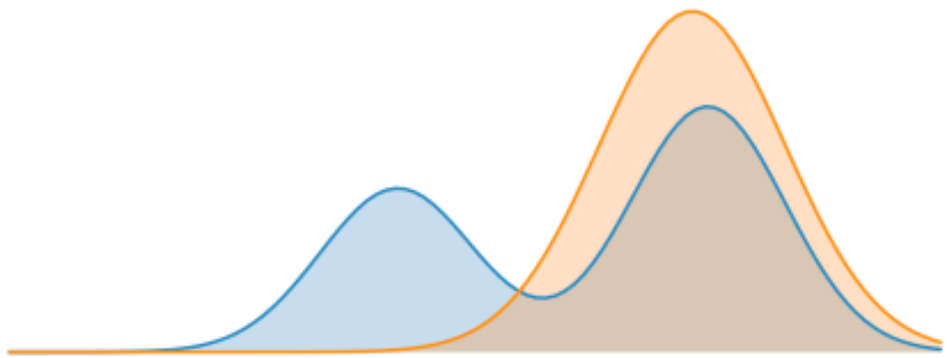
$$\mathscr{L} = Y - \hat{Y}$$

$$\mathscr{L} = |Y - \hat{Y}|^2$$

$$\mathscr{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}) + \log(1 - \hat{y}_i)$$

$$\mathscr{L} = \mathscr{P}(\hat{Y})$$

$$\mathscr{L} = \mathscr{L}_{\text{rec}} + \mathscr{L}_{\text{reg}}$$



• • •

3- Your Loss

$$\mathscr{L} = Y - \hat{Y}$$

$$\mathscr{L} = |Y - \hat{Y}|^2$$

You can (and should) be as creative as you want !

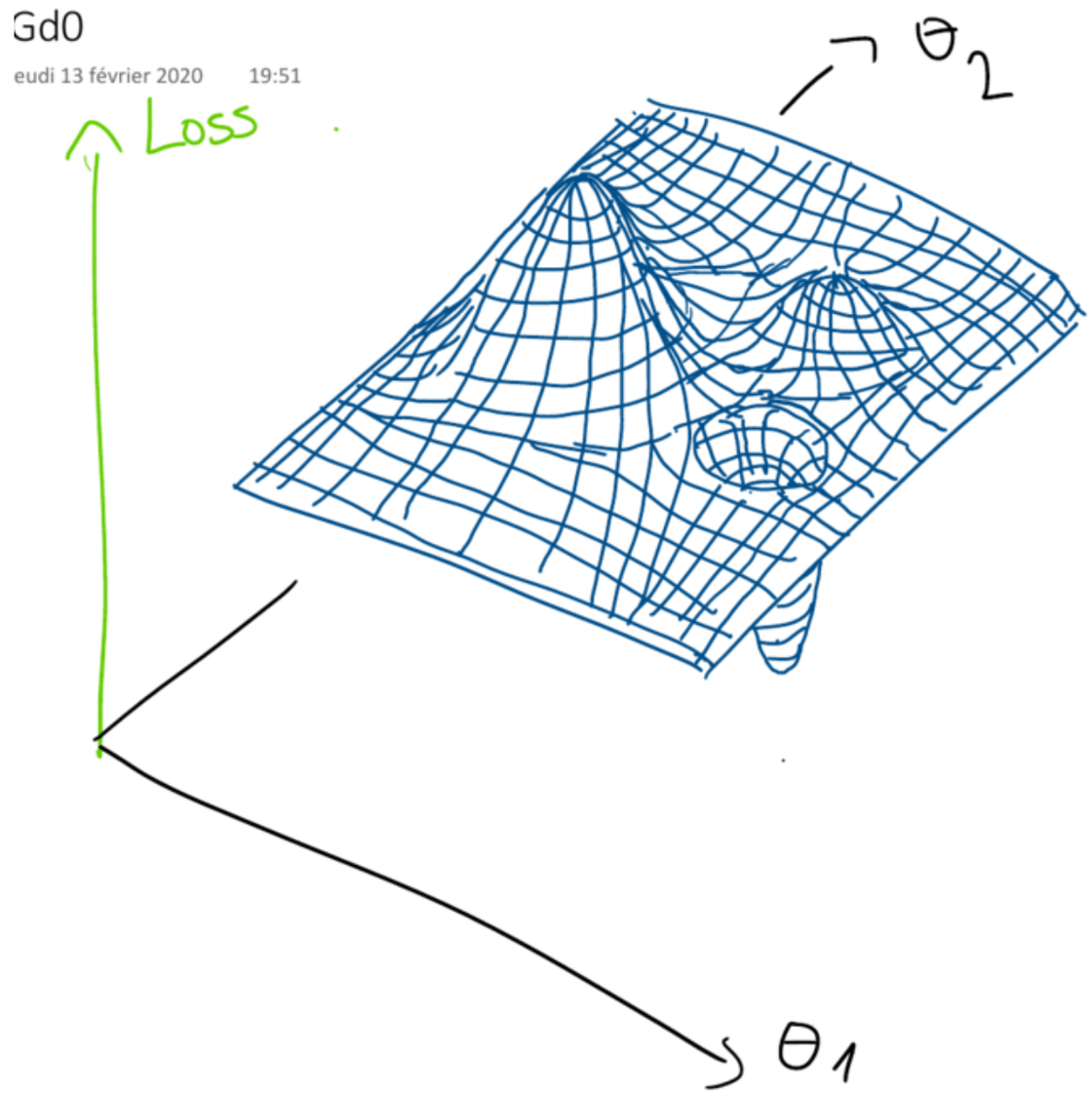
$$\mathscr{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}) + \log(1 - \hat{y}_i)$$

$$\mathscr{L} = \mathscr{P}(\hat{Y})$$

$$\mathscr{L} = \mathscr{L}_{\text{rec}} + \mathscr{L}_{\text{reg}}$$

• • •

4- Optimization



4- Optimization

$$\hat{f} : x \to w\,x + b\,.$$

$$\mathcal{L} = \frac{1}{n}\sum_{i=0}^{n} |\hat{y}_i - y_i|^2$$

$$\begin{pmatrix} w_{i+1} \\ b_{i+1} \end{pmatrix} = \begin{pmatrix} w_i - ds_w \\ b_i - ds_b \end{pmatrix}$$

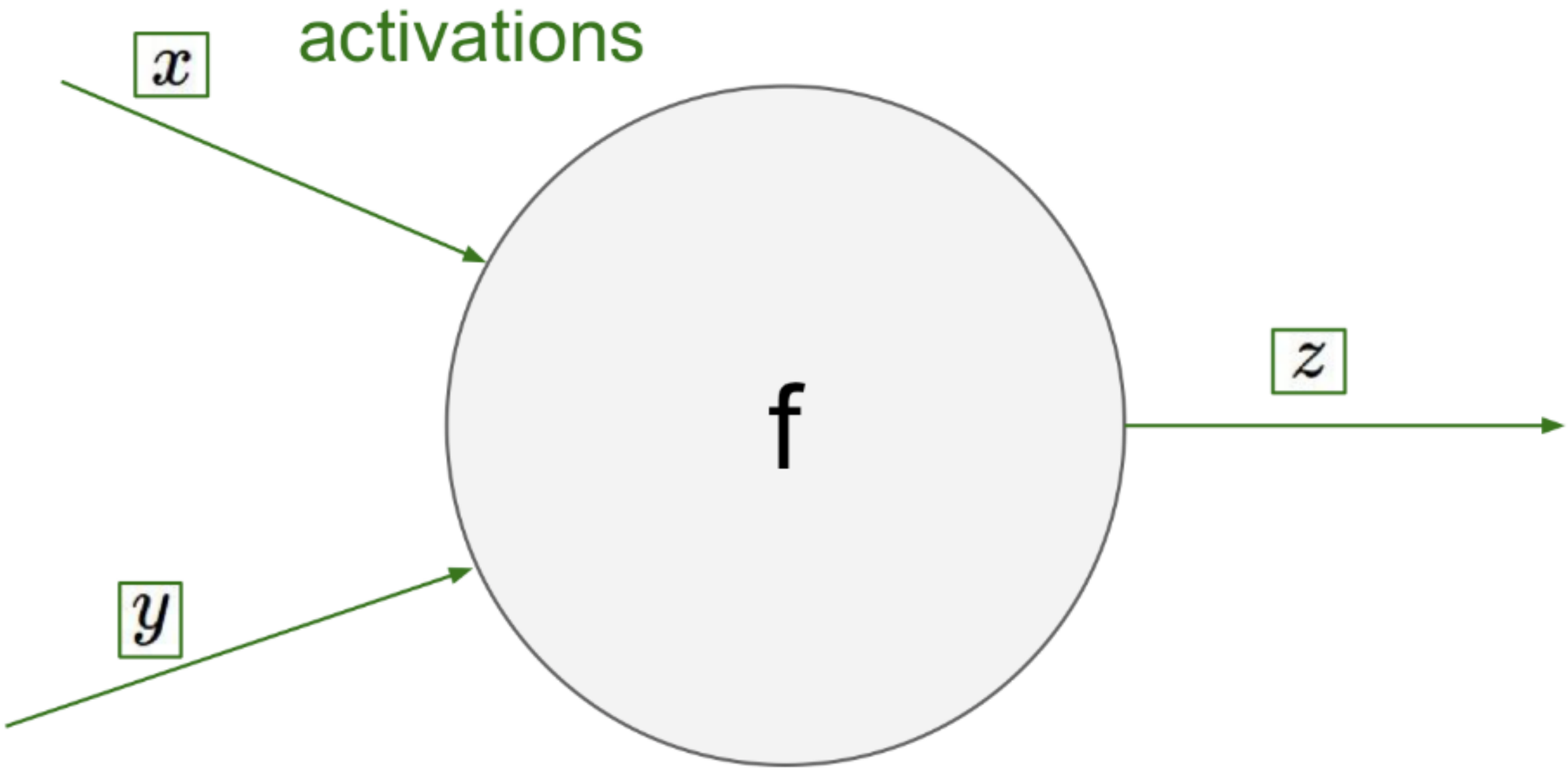
$$\boldsymbol{ds} = \boldsymbol{\nabla}\mathcal{L}(w_i, b_i)\, l_r$$

$$\begin{aligned}
\boldsymbol{ds} &= \boldsymbol{\nabla}\mathcal{L}(w_i, b_i)\, l_r \\
&= \begin{pmatrix} \frac{\partial \mathcal{L}(w_i, b_i)}{\partial w_i} \\ \frac{\partial \mathcal{L}(w_i, b_i)}{\partial b_i} \end{pmatrix} l_r \\
&= \frac{1}{n}\sum_{j=0}^{n} \begin{pmatrix} \frac{\partial\, |\hat{y}_j - y_j|^2}{\partial w_i} \\ \frac{\partial\, |\hat{y}_j - y_j|^2}{\partial b_i} \end{pmatrix} l_r \\
&= \frac{1}{n}\sum_{j=0}^{n} \begin{pmatrix} \frac{\partial\, |w_i x_j + b_i - y_j|^2}{\partial w_i} \\ \frac{\partial\, |w_i x_j + b_i - y_j|^2}{\partial b_i} \end{pmatrix} l_r \\
&= \frac{2}{n}\sum_{j=0}^{n} \begin{pmatrix} x_j\,(w_i x_j + b_i - y_j) \\ w_i x_j + b_i - y_j \end{pmatrix} l_r \\
&= \frac{2}{n}\sum_{j=0}^{n} \begin{pmatrix} x_j\,(\hat{y}_j - y_j) \\ \hat{y}_j - y_j \end{pmatrix} l_r\,,
\end{aligned}$$

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

BACKPROPAGATION

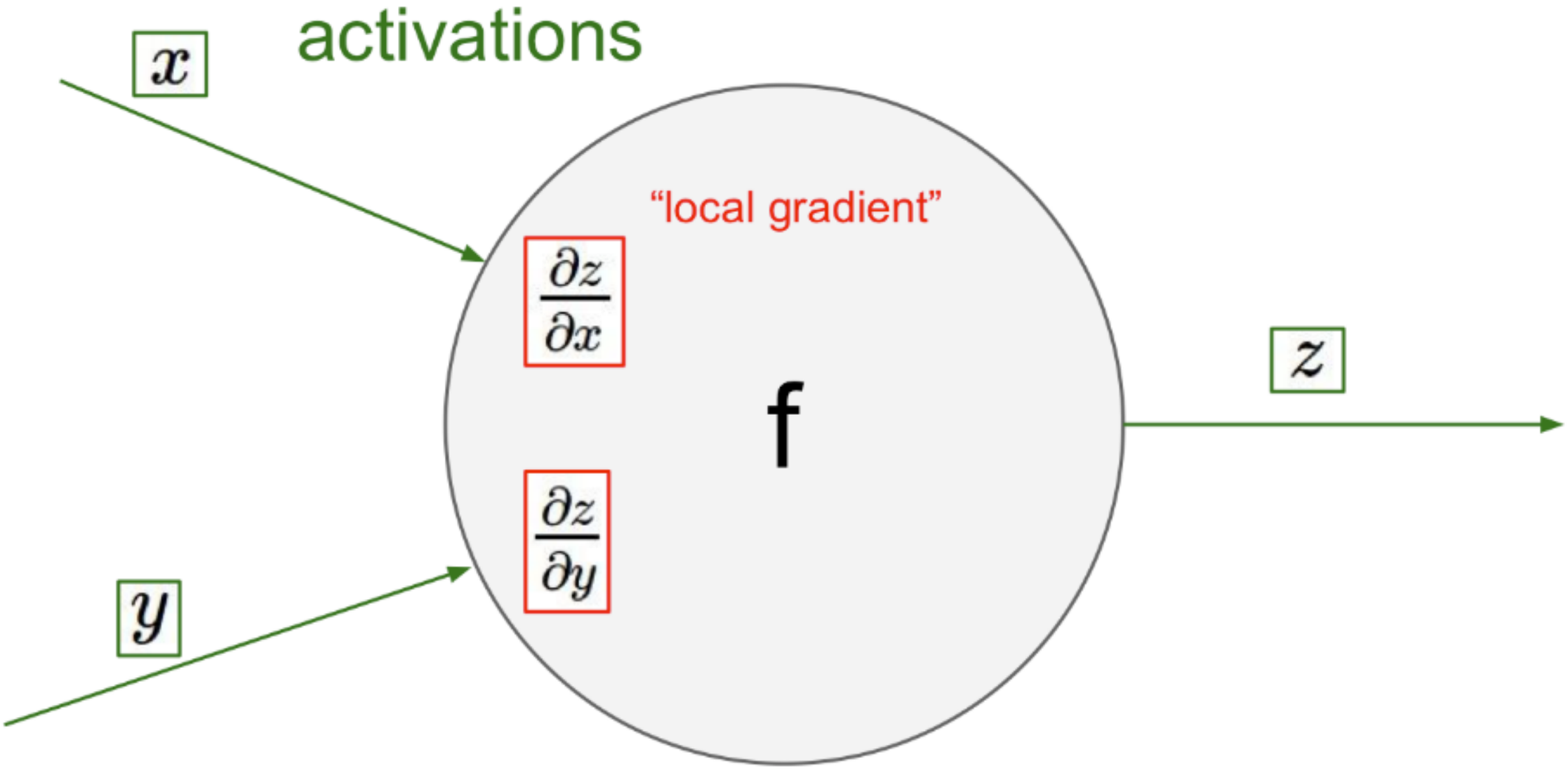
[AT THE NEURON LEVEL]



Credit: A. Karpathy

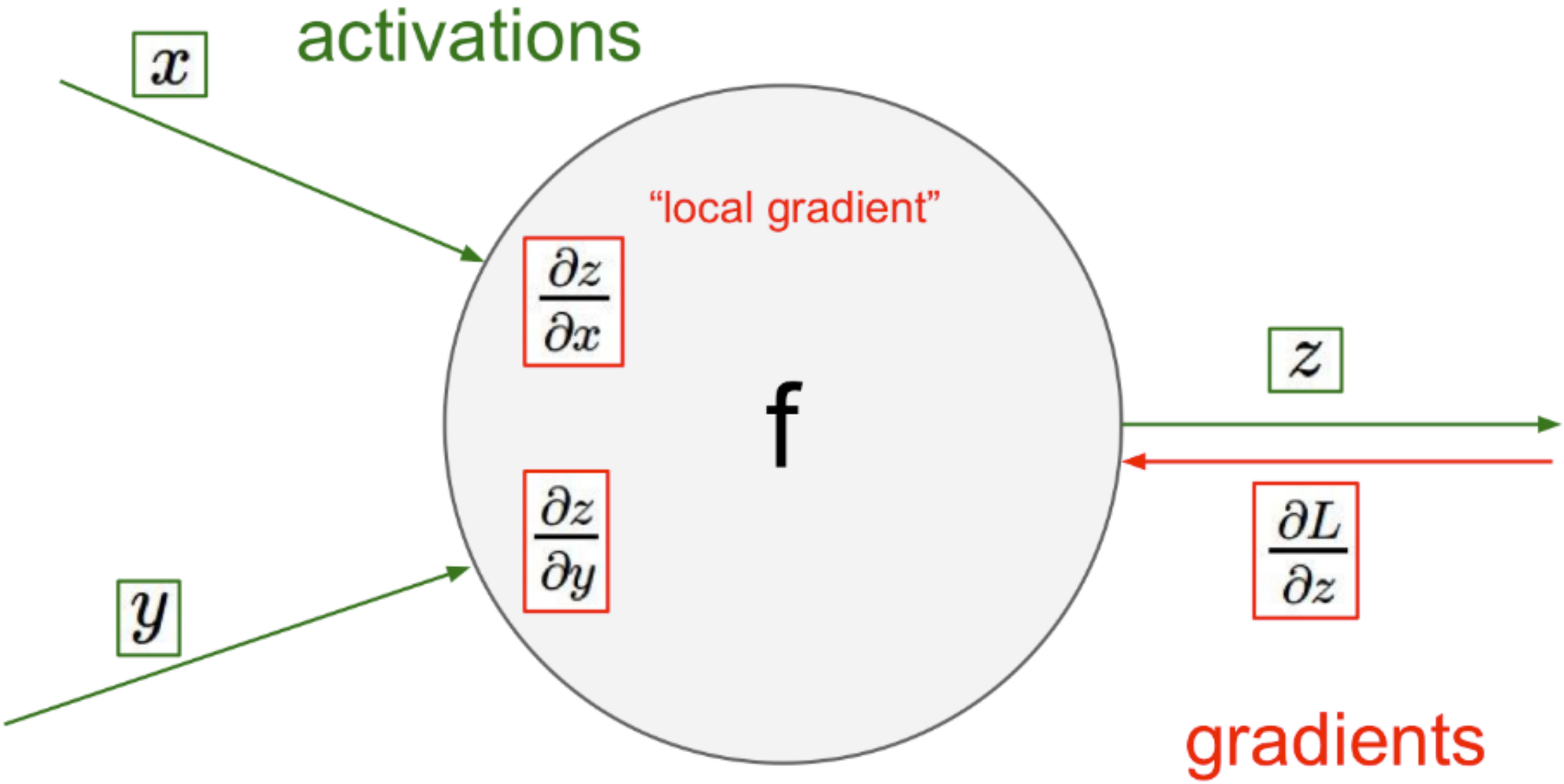
BACKPROPAGATION

[AT THE NEURON LEVEL]



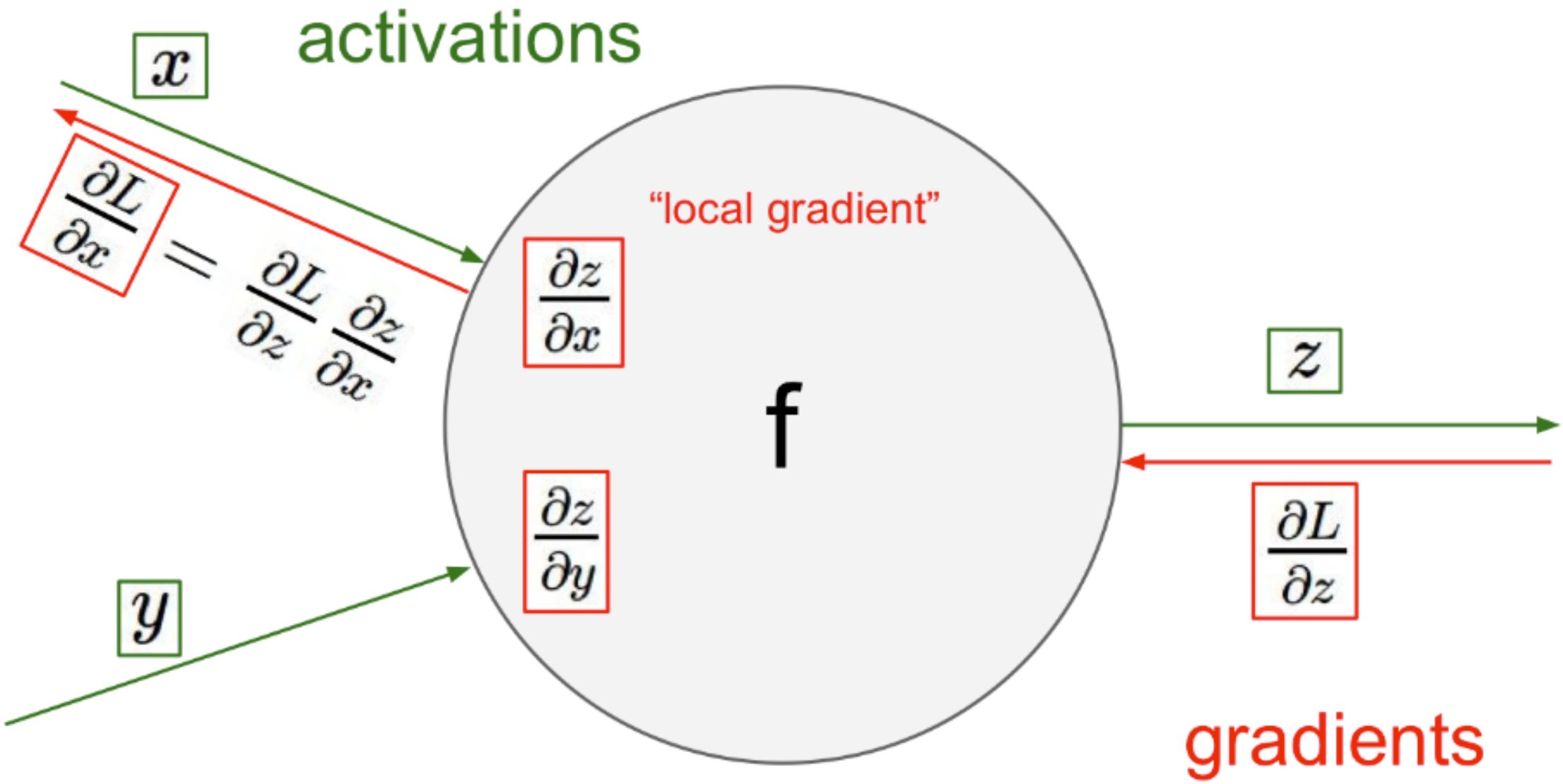
BACKPROPAGATION

[AT THE NEURON LEVEL]



BACKPROPAGATION

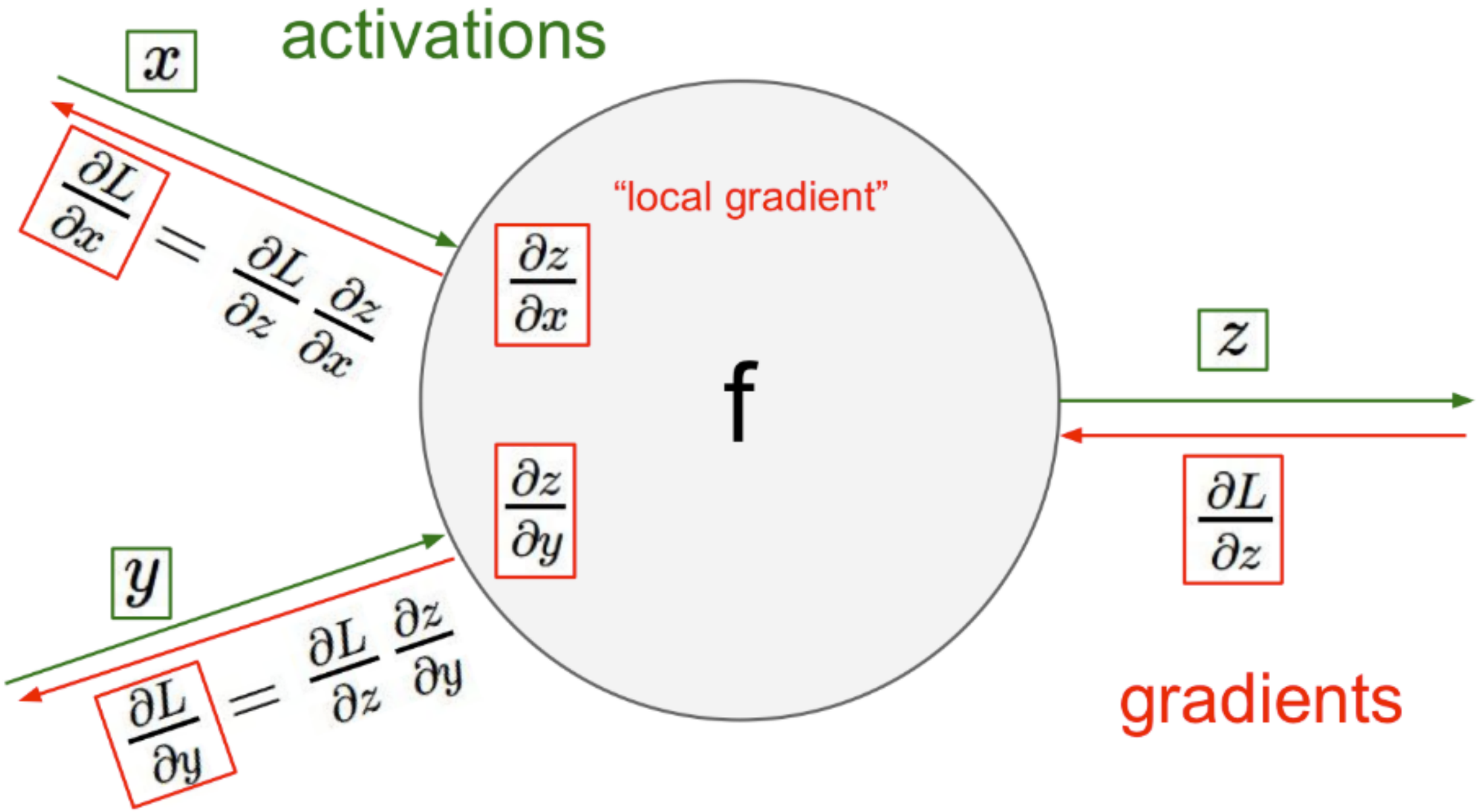
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

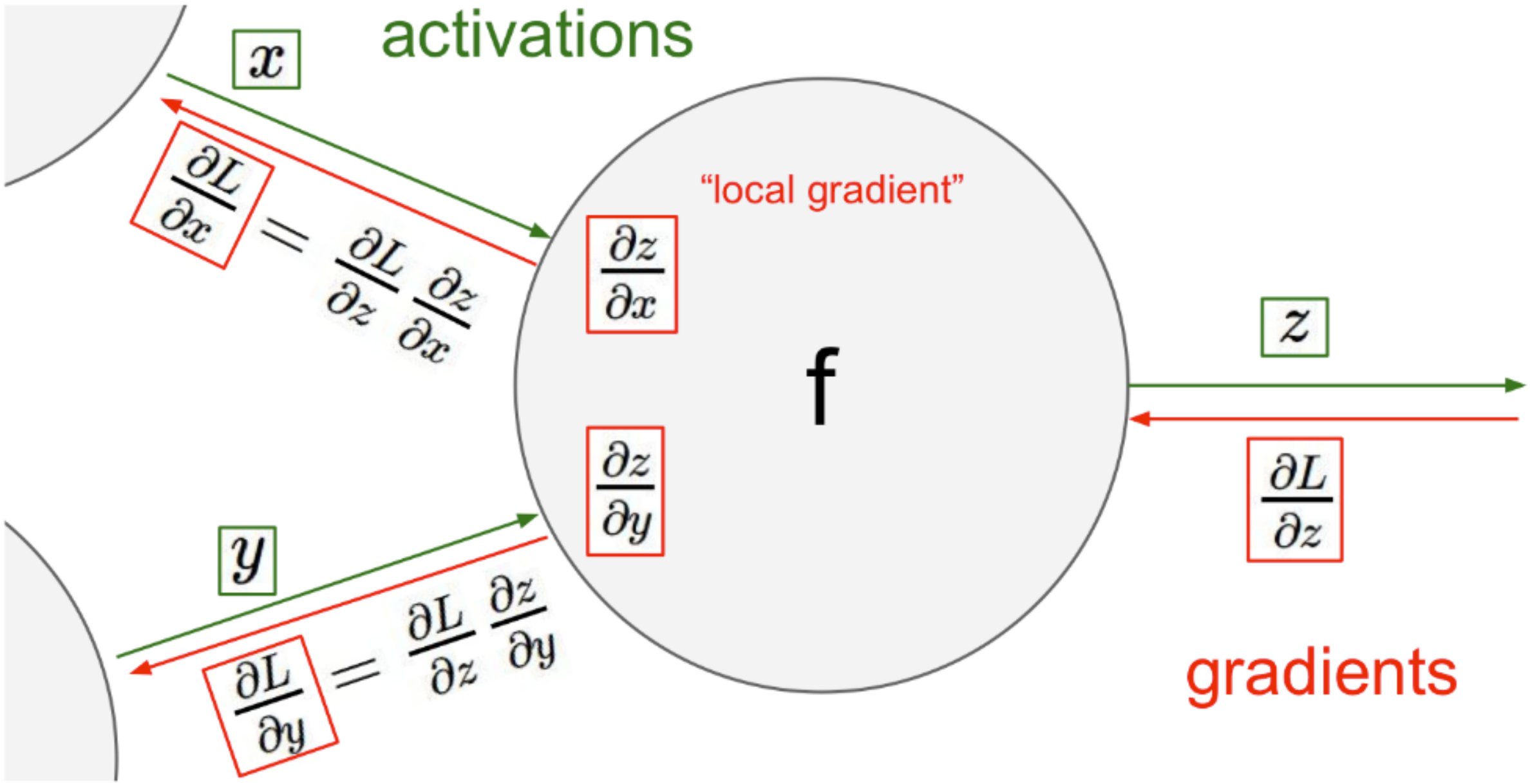
[AT THE NEURON LEVEL]



Credit: A. Karpathy

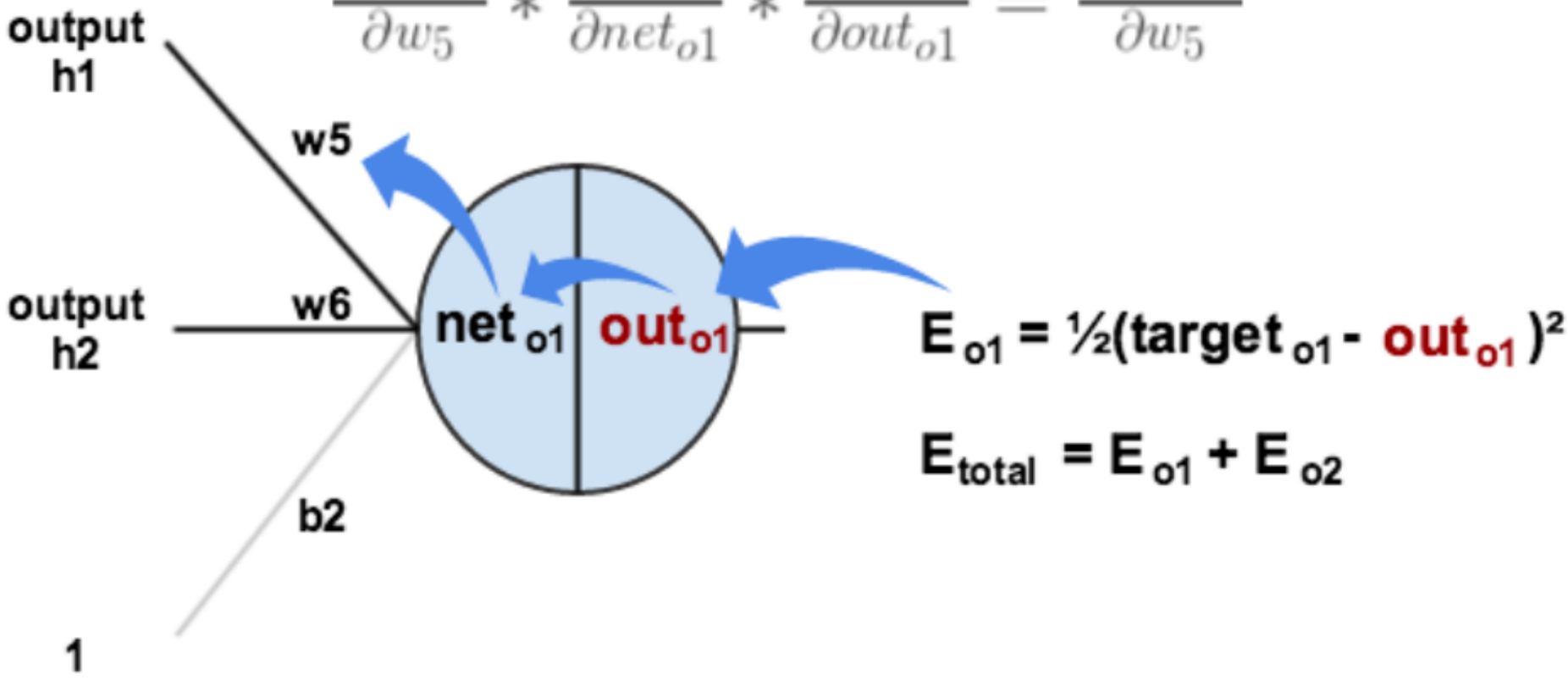
BACKPROPAGATION

[AT THE NEURON LEVEL]



Credit: A. Karpathy

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



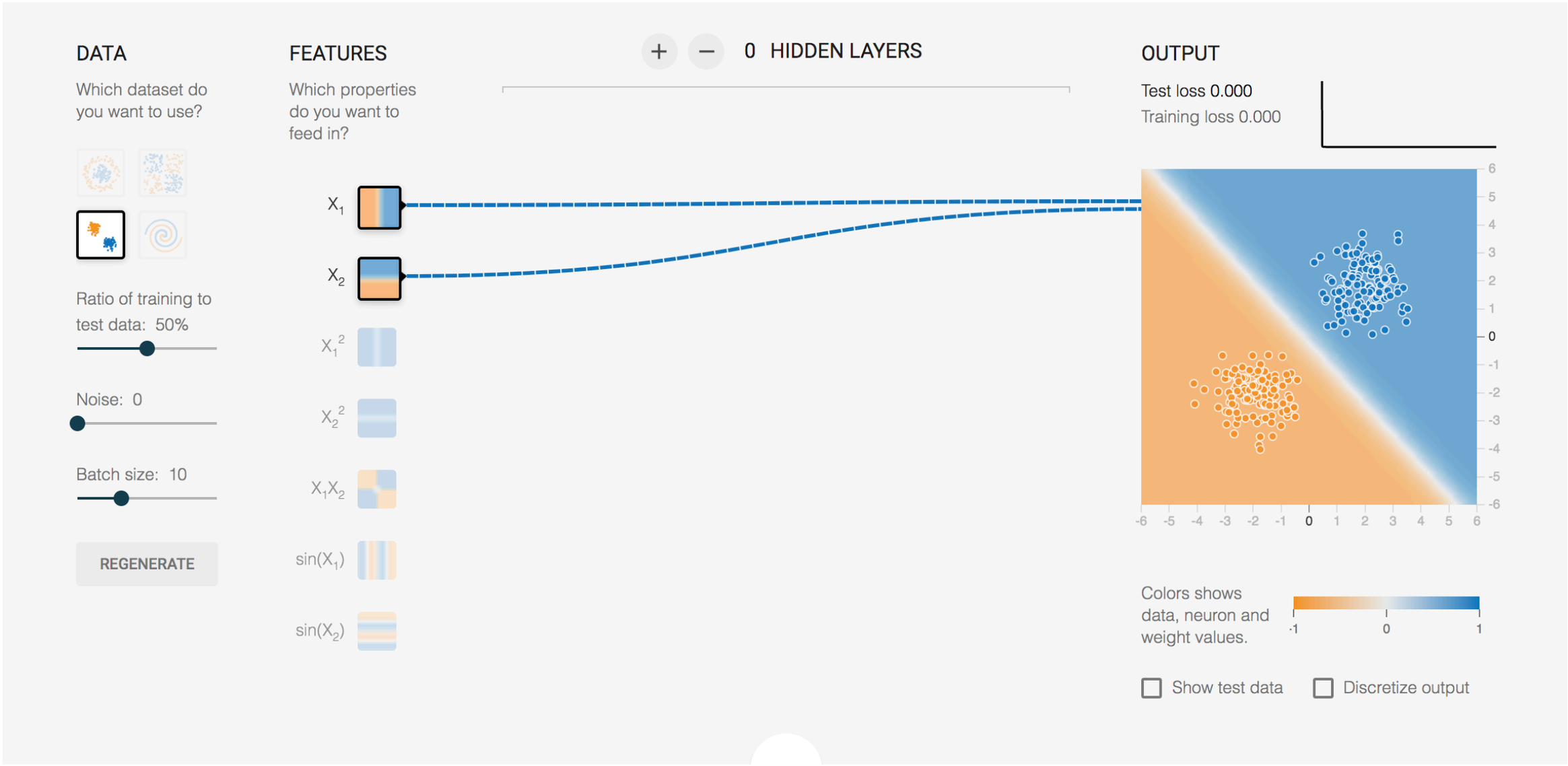
3. THE BACKWARD PASS

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

WE APPLY THE CHAIN RULE:

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

https://
playground.tensorflow.org/



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

THE CHOICES OF THE LEARNING RATES AND THE FREQUENCY OF WEIGTH UPDATES ARE IMPORTANT HYPER PARAMETERS TO MAKE NNs MORE ROBUST

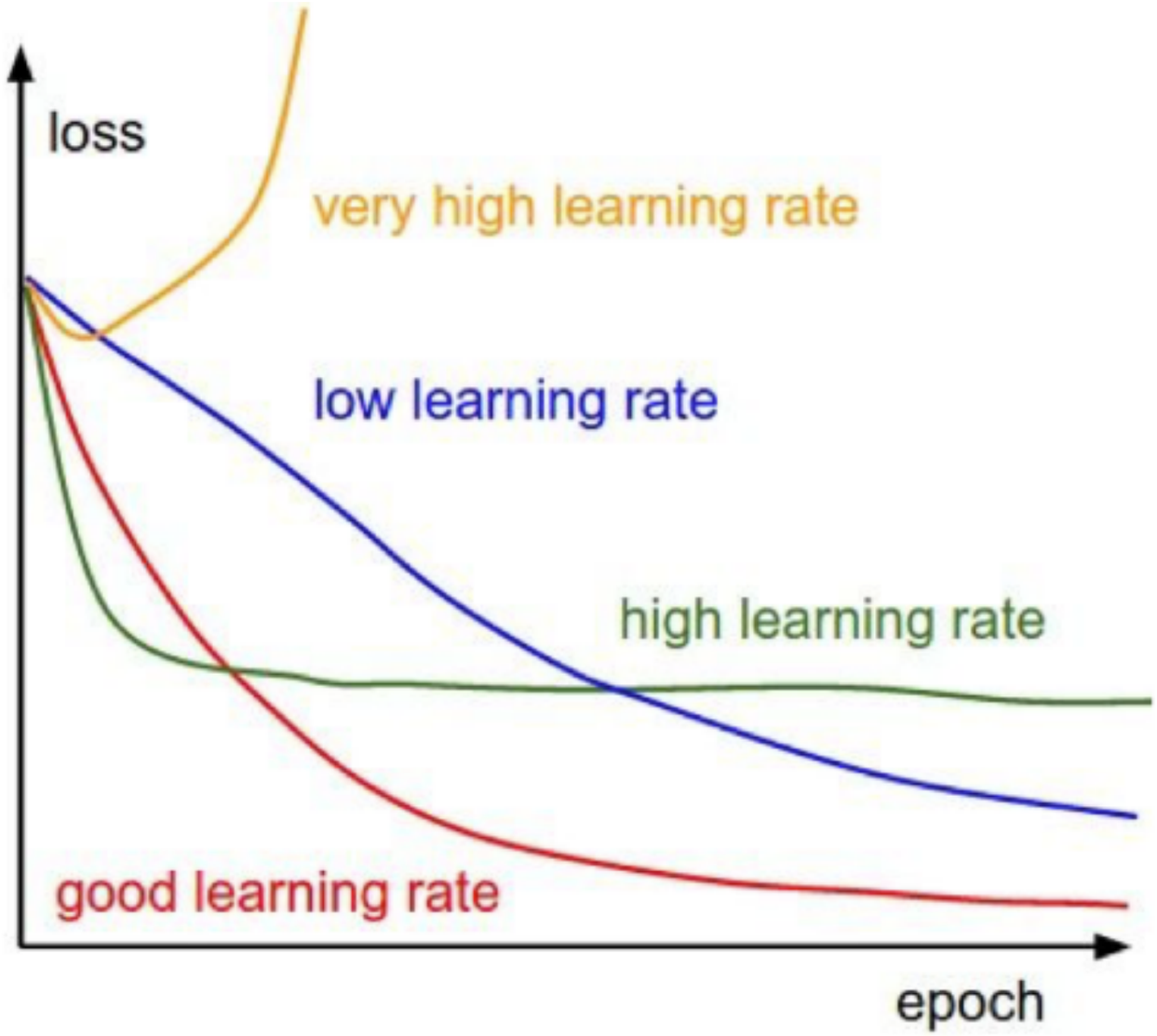
LEARNING RATES

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

THERE ARE DIFFERENT WAYS
TO UPDATE THE LEARNING RATE

Credit:

LEARNING RATES



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.

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

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

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

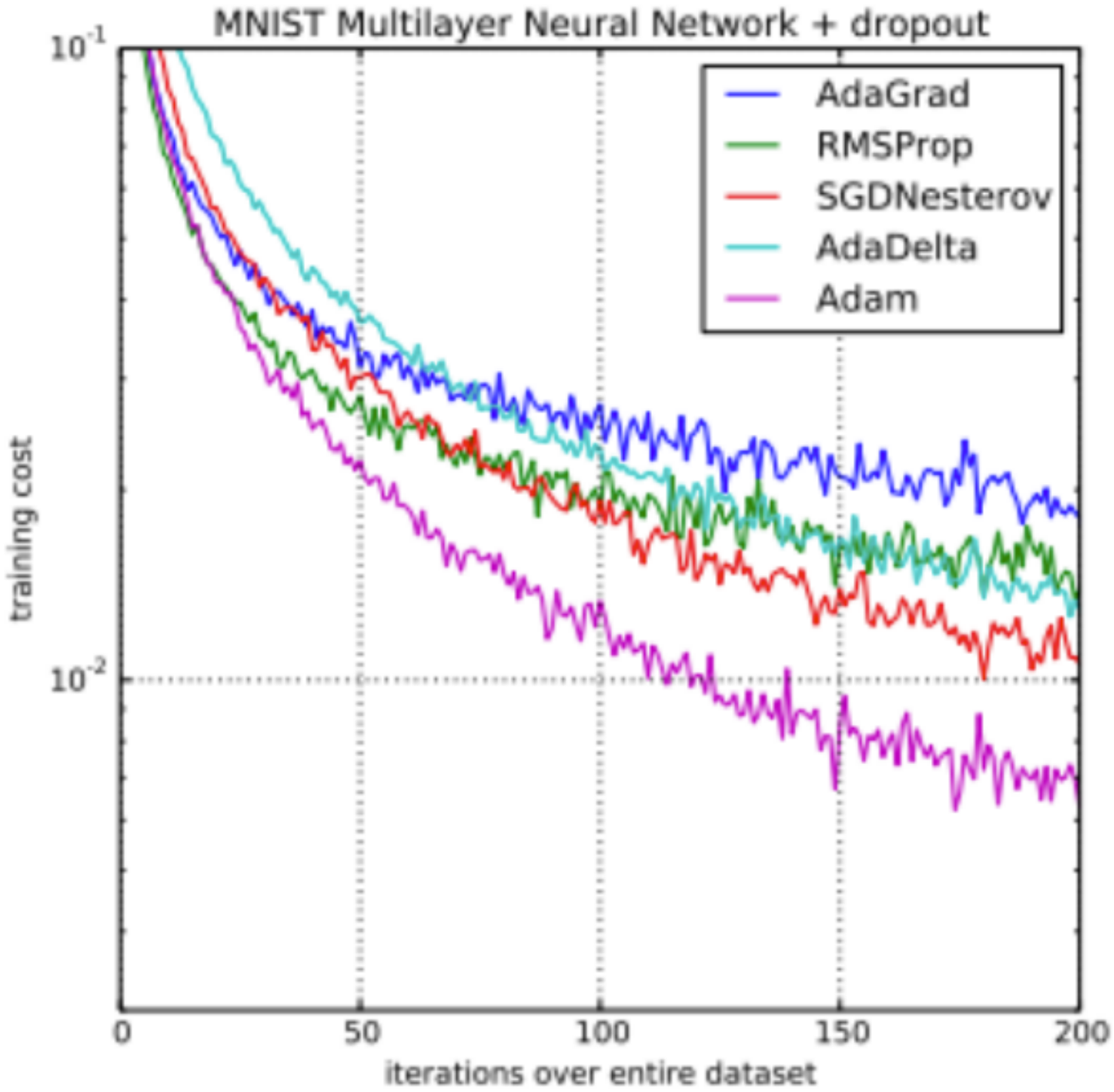
ADAM [Adaptive moment estimator]:

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


MNIST Multilayer Neural Network + dropout
10^{-1}
10^{-2}
training cost
AdaGrad
RMSProp
SGDNesterov
AdaDelta
Adam
0
50
100
150
200
iterations over entire dataset

Credit

BATCH GRADIENT DESCENT

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

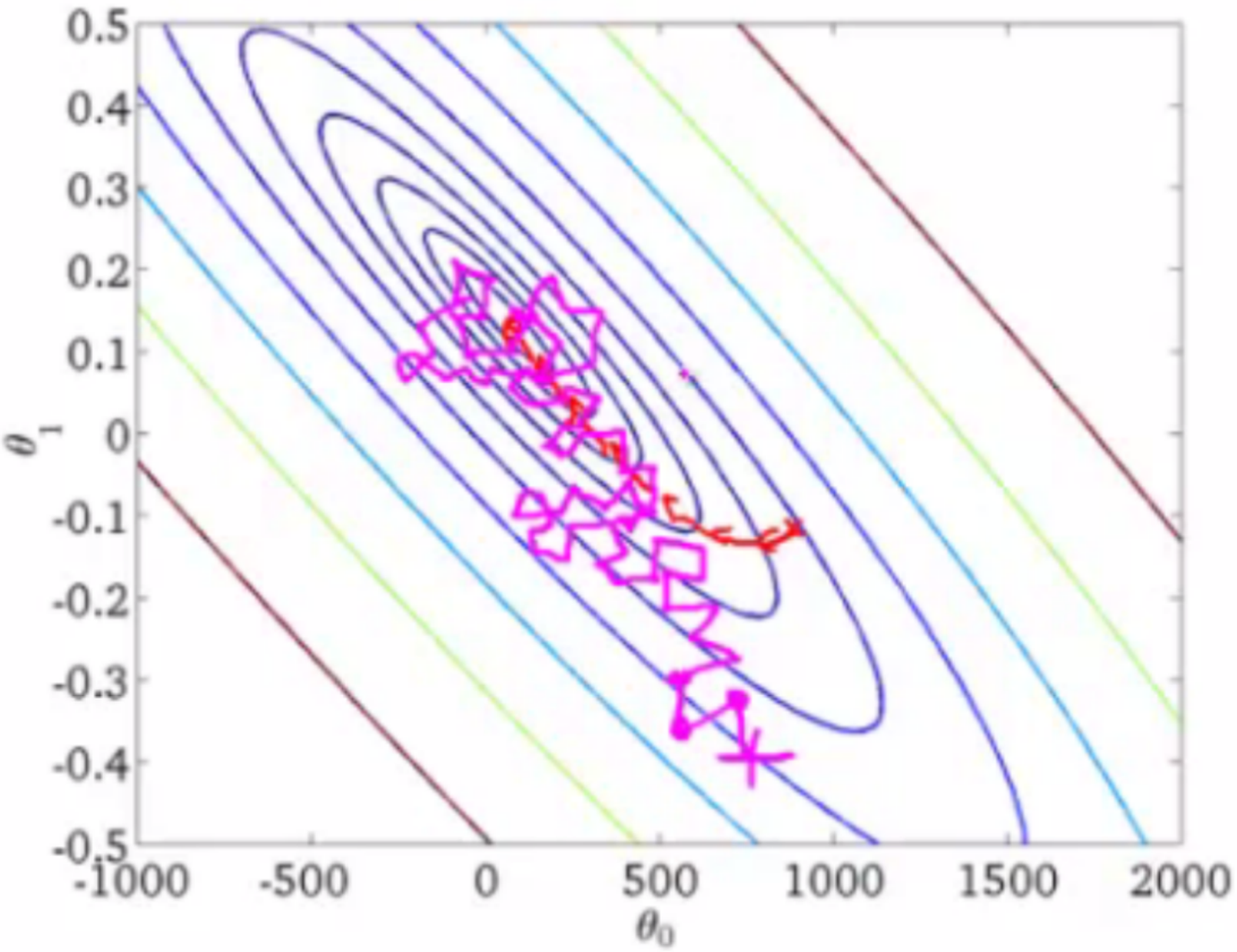
BATCH GRADIENT DESCENT

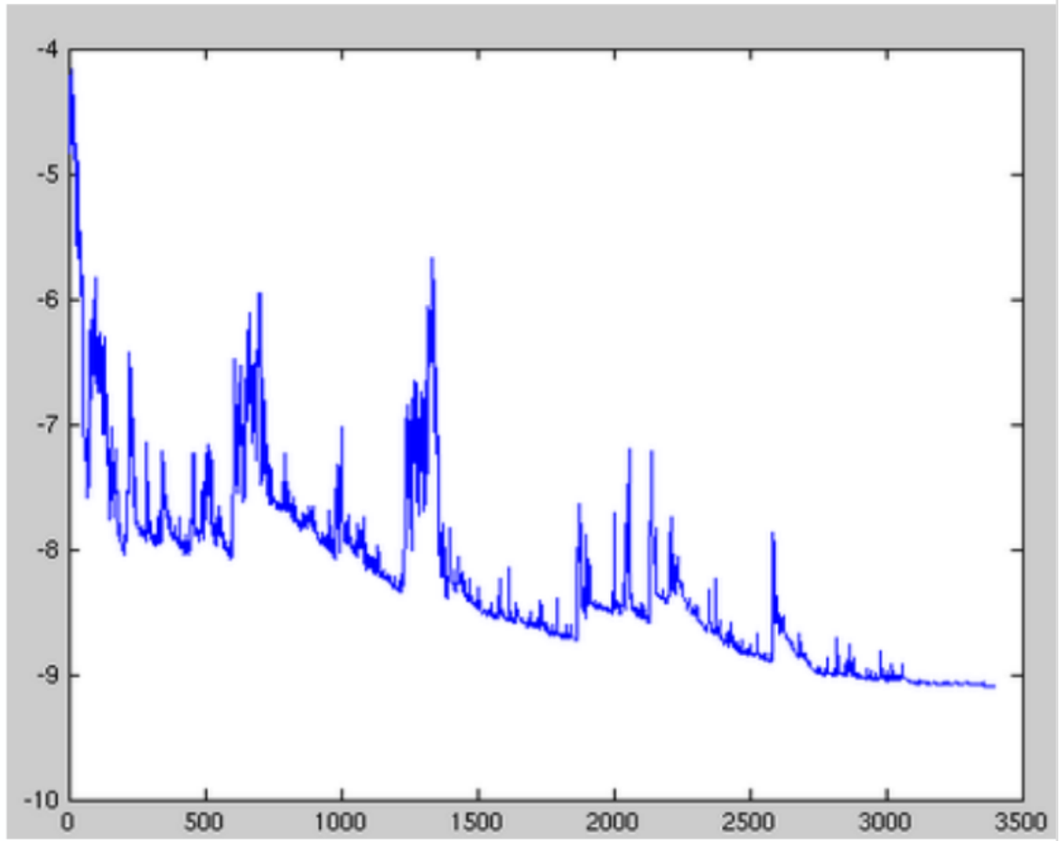
LOCAL MINIMA CAN ALSO BE AVOIDED BY COMPUTING THE GRADIENT IN SMALL BATCHES INSTEAD OF OVER THE FULL DATASET

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



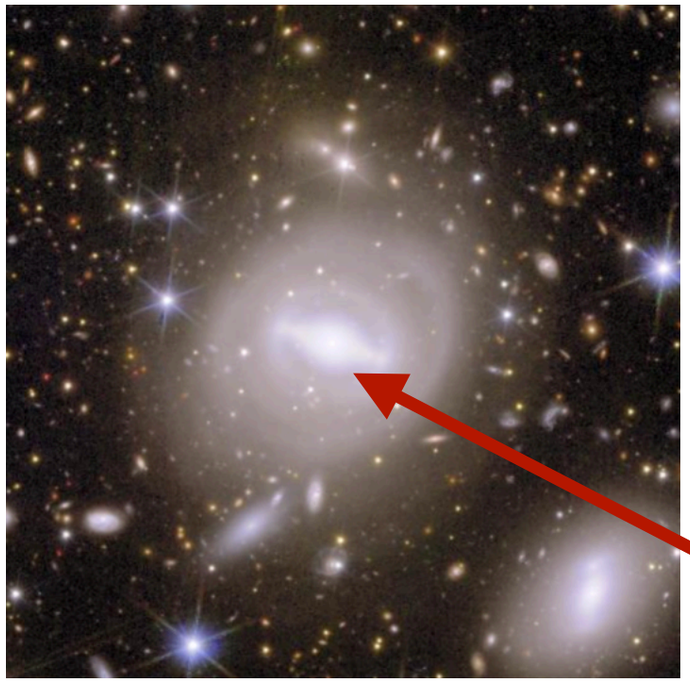


Fluctuations in the total objective function as gradient steps with respect to mini-batches are taken.

Credit

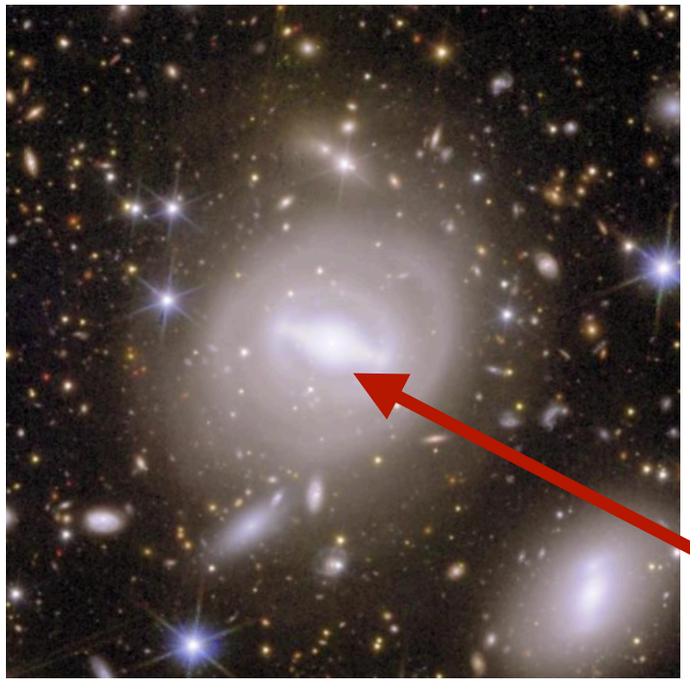
2. Convolutional neural Network

ESA/Euclid/Euclid Consortium/NASA, image processing by J.-C. Cuillandre (CEA Paris-Saclay), G. Anselmi



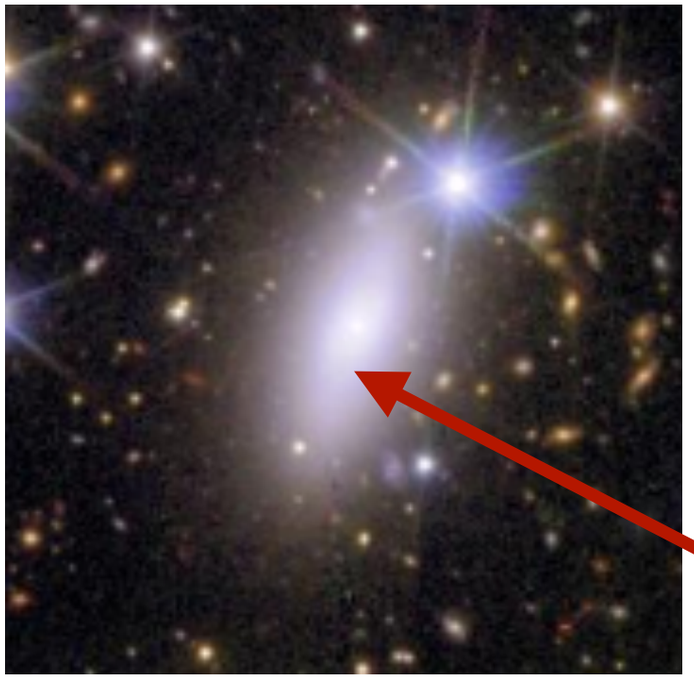
I want to classify this galaxy

ESA/Euclid/Euclid Consortium/NASA, image processing by J.-C. Cuillandre (CEA Paris-Saclay), G. Anselmi



Dense layers: I multiply all the pixels of the image, and learn the correlations

ESA/Euclid/Euclid Consortium/NASA, image processing by J.-C. Cuillandre (CEA Paris-Saclay), G. Anselmi



If i know have this image, I can distiguish one from the other.

ESA/Euclid/Euclid Consortium/NASA, image processing by J.-C. Cuillandre (CEA Paris-Saclay), G. Anselmi

But what if I have this:




The weights I’ve learnt from the first (same) spiral galaxy won’t apply to the same part of the image, and even though it s the same galaxy, it won’t classify it the same.

I would like some operation that is locally translational and rotational invariant: a spiral galaxy is a spiral galaxy independantly of the angle and place we observe it in our CCD.

ESA/Euclid/Euclid Consortium/NASA, image processing by J.-C. Cuillandre (CEA Paris-Saclay), G. Anselmi

But what if I have this:

That’s the convolution operation!

The weights I’ve learnt from the first (same) spiral galaxy won’t apply to the same part of the image, and even though it s the same galaxy, it won’t classify it the same.

I would like some operation that is locally translational and rotational invariant: a spiral galaxy is a spiral galaxy independantly of the angle and place we observe it in our CCD.

Discrete Convolution

1D: [Spectra]

$$f(x) * g(x) = \sum_{k=-\infty}^{k=+\infty} f(k).g(k-x)$$

2D: [Images]

$$f(x,y) * g(x,y) = \sum_{k=-\infty}^{k=+\infty} \sum_{l=-\infty}^{l=+\infty} f(k,l).g(x-k,y-l)$$

DISCRETE CONVOLUTION

1D: [Spectra]

$$f(x) * g(x) = \sum_{k=-\infty}^{k=+\infty} f(k).g(k-x)$$

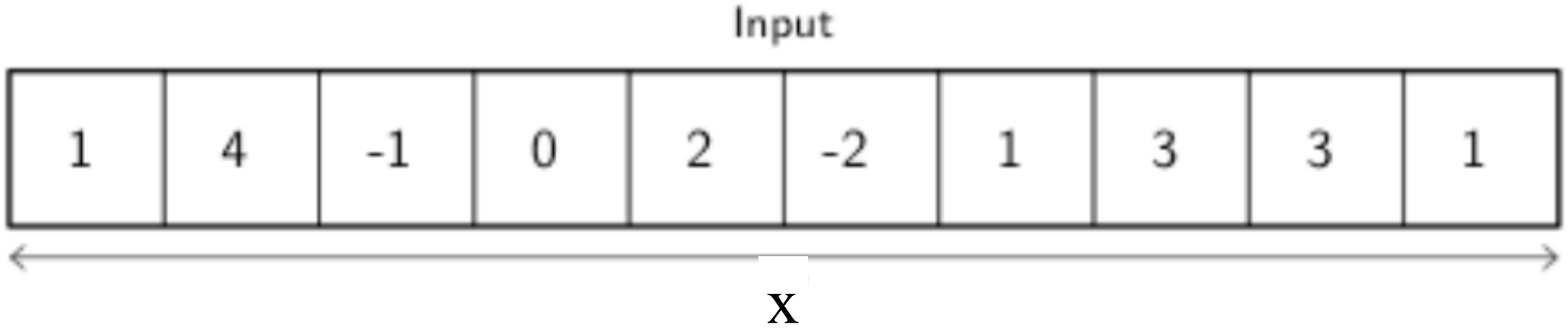
2D: [Images]

$$f(x,y) * g(x,y) = \sum_{k=-\infty}^{k=+\infty} \sum_{l=-\infty}^{l=+\infty} f(k,l).g(x-k,y-l)$$

CONVOLUTION KERNEL

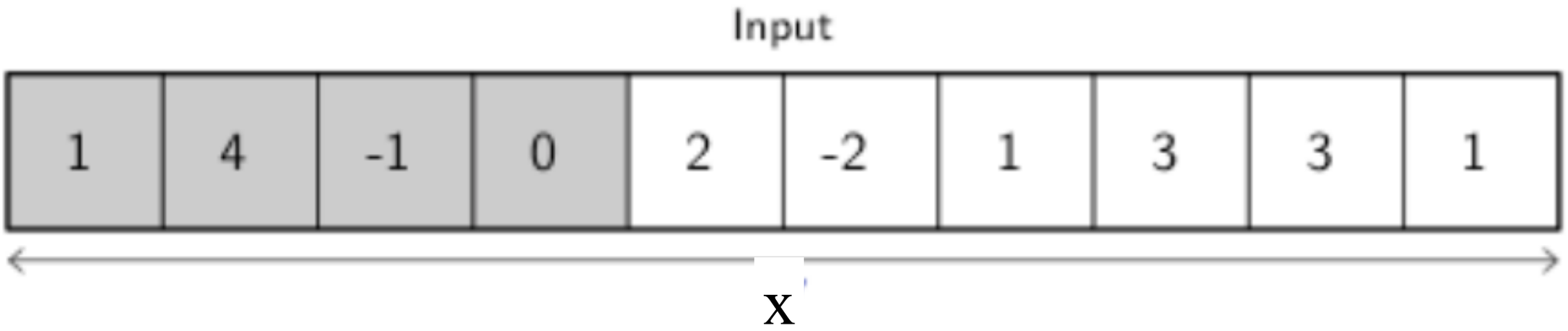
INPUT DATA

1-D CONVOLUTION



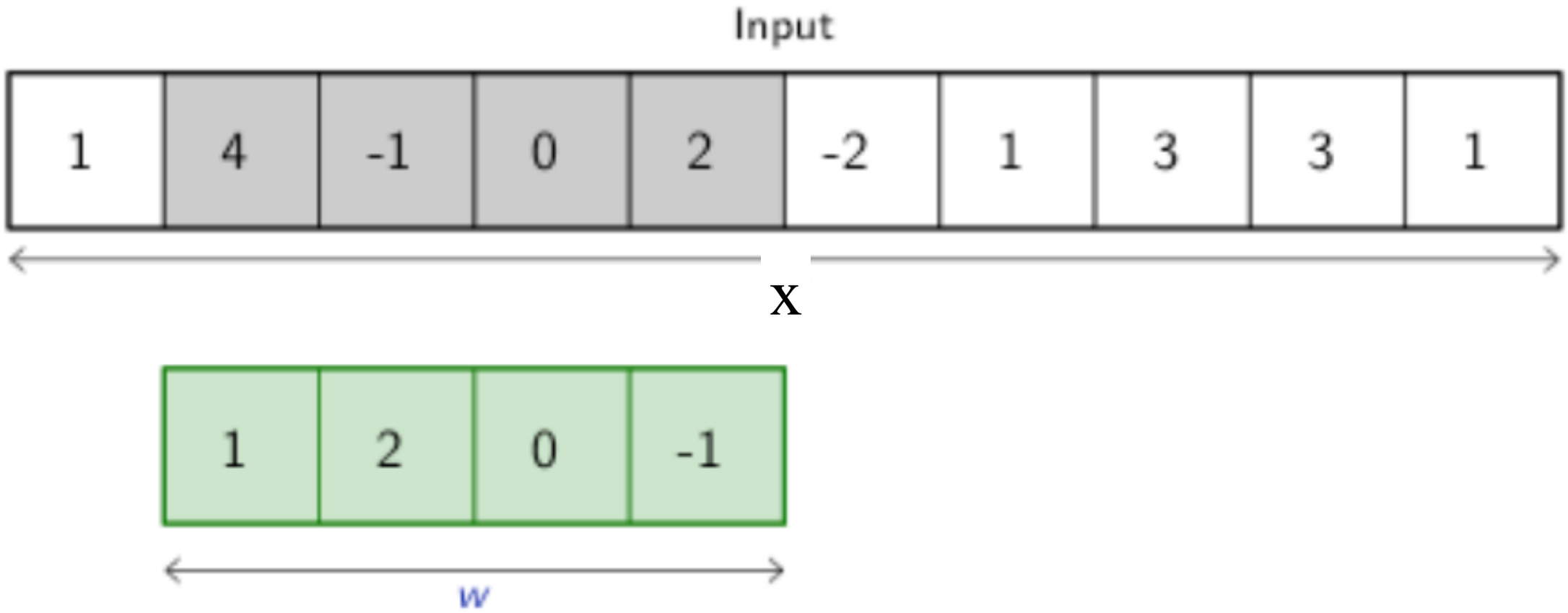
credit

1-D CONVOLUTION



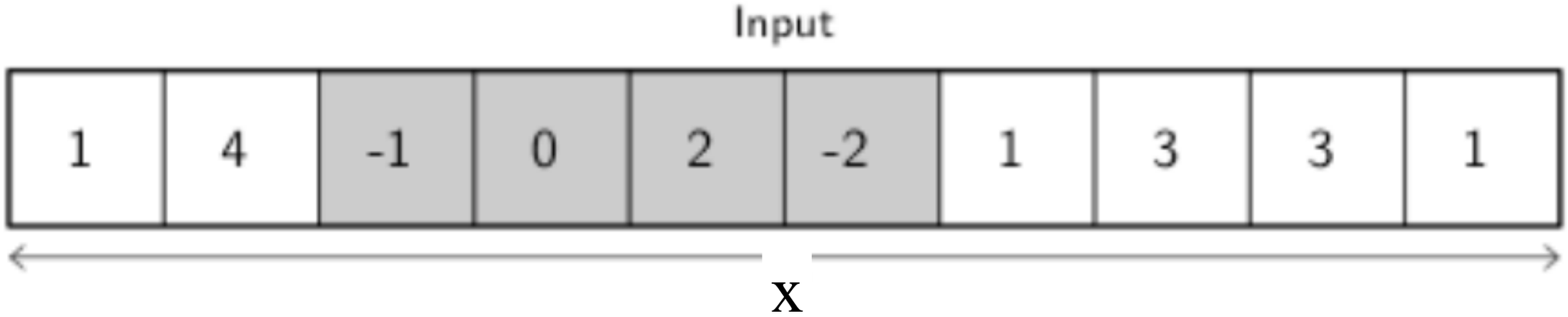
credit

1-D CONVOLUTION



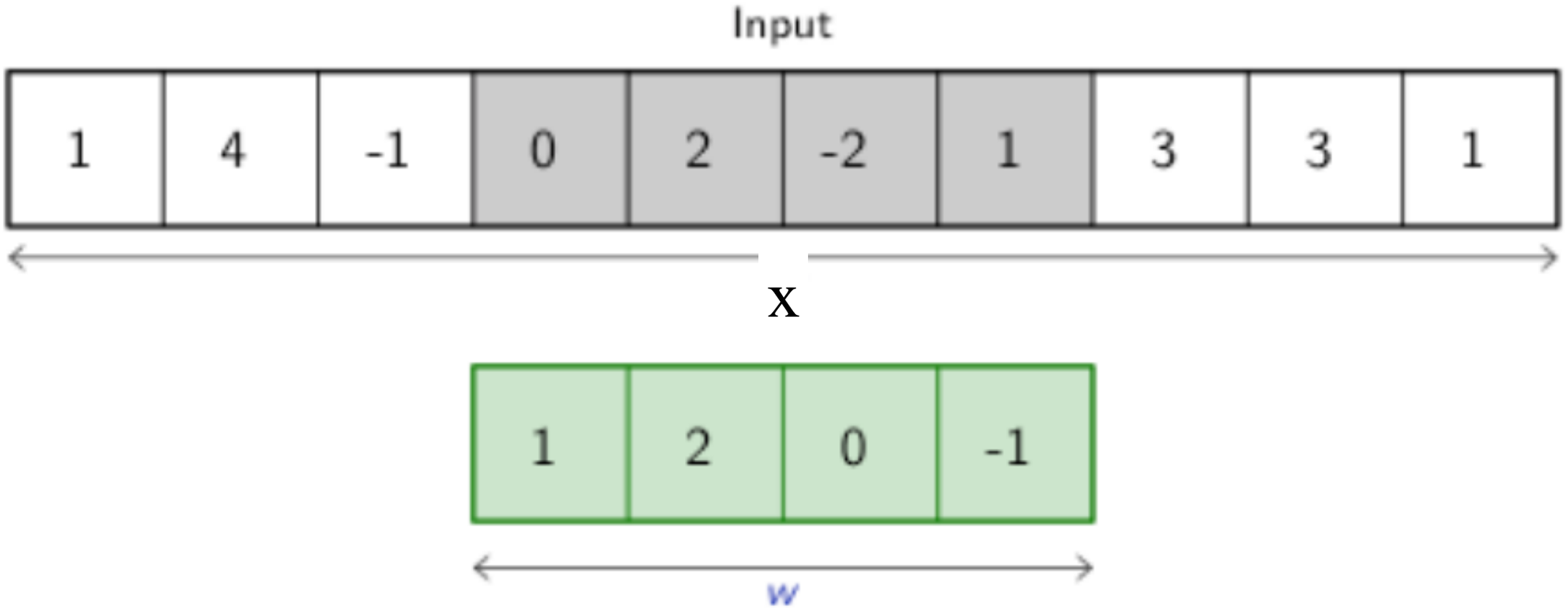
credit

1-D CONVOLUTION



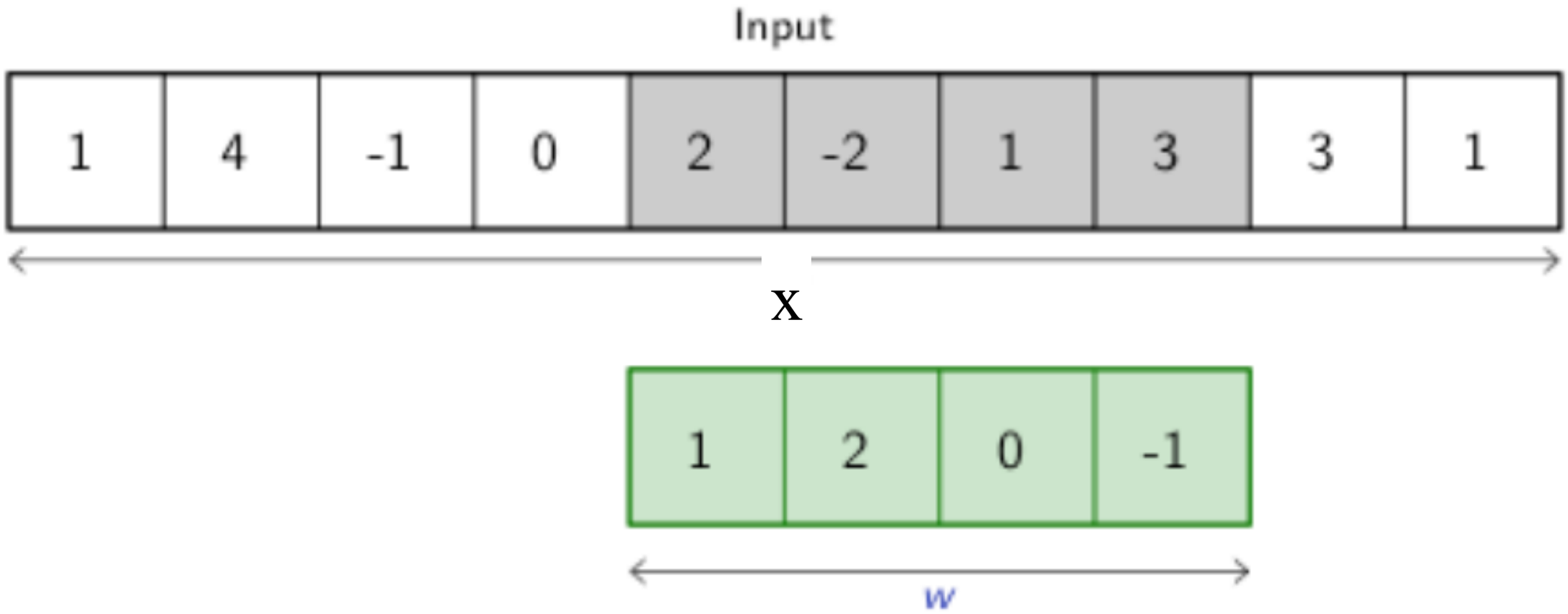
credit

1-D CONVOLUTION



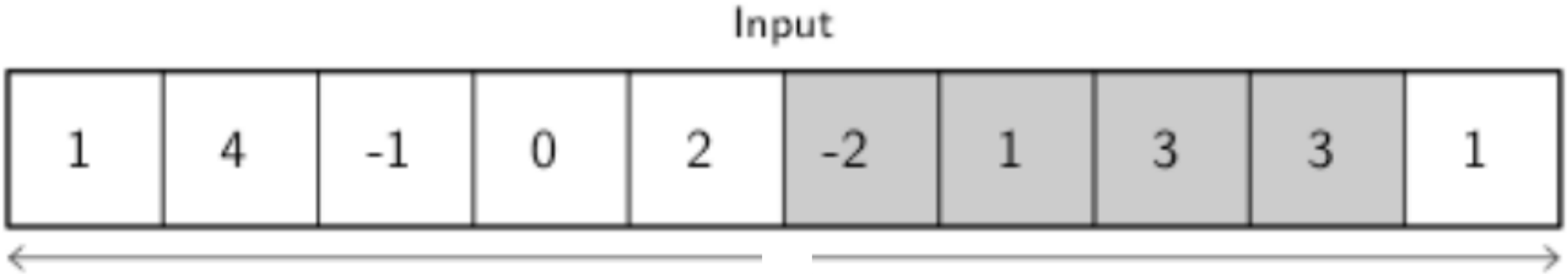
credit

1-D CONVOLUTION



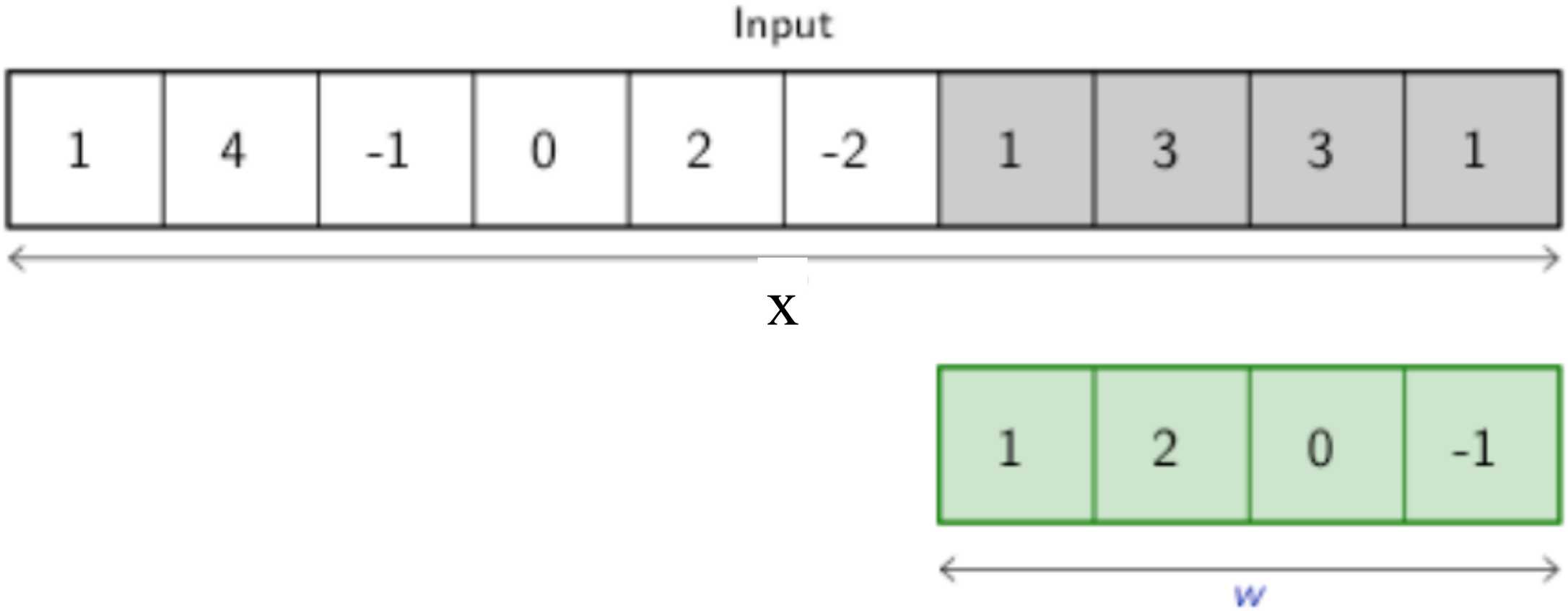
credit

1-D CONVOLUTION

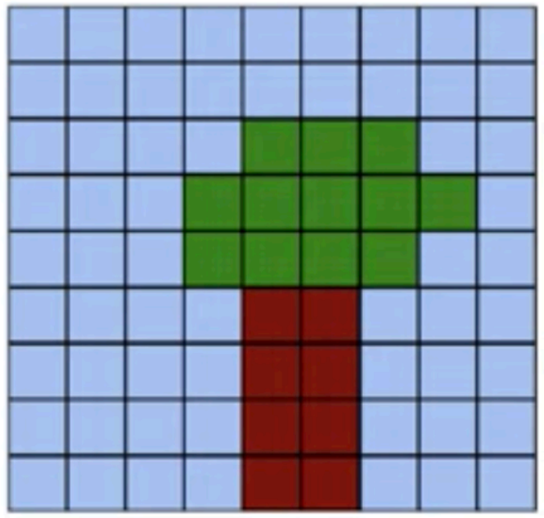


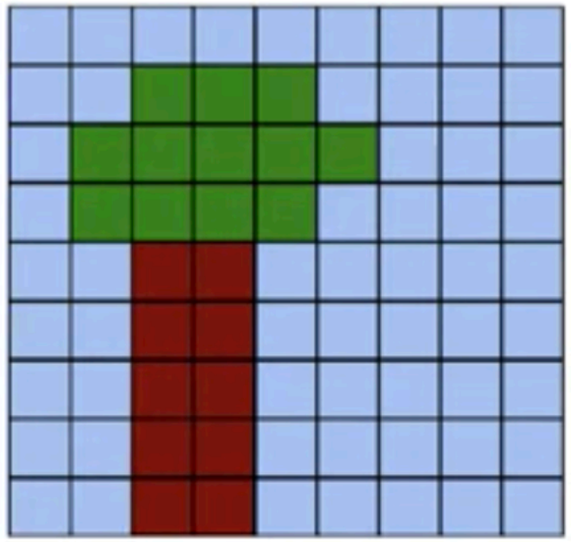
credit

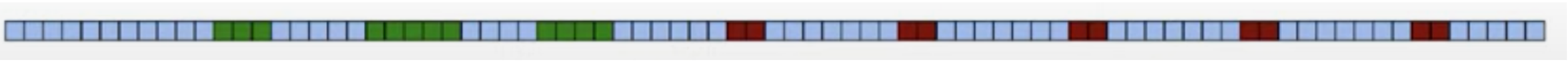
1-D CONVOLUTION



credit

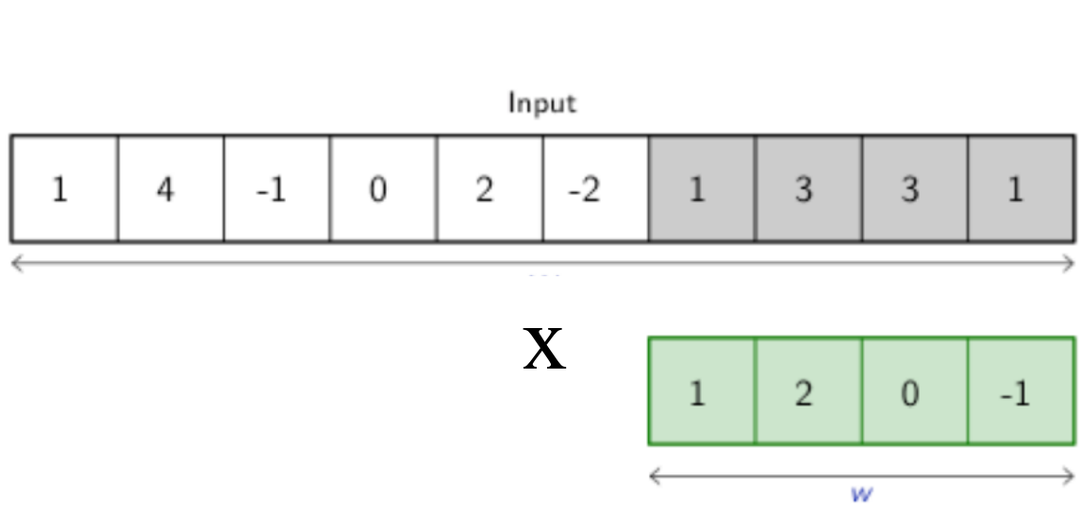


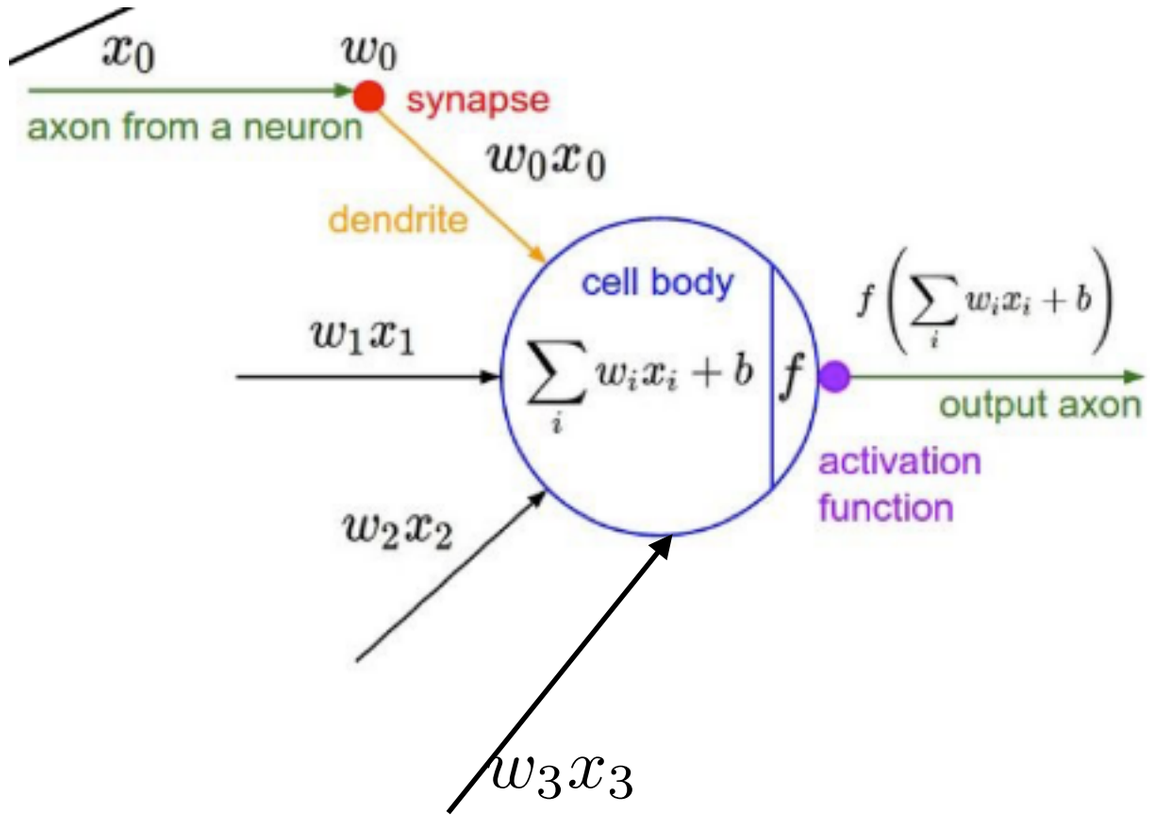




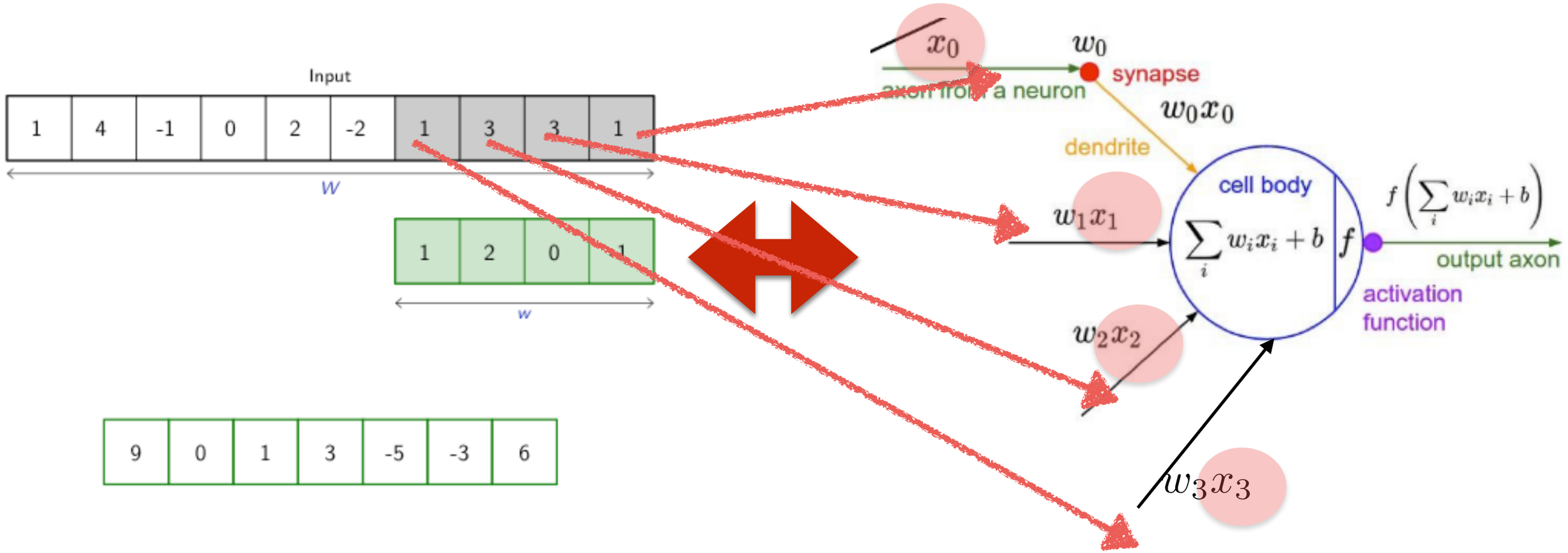
(Dielemann@Deepmind)

THE CONVOLUTION BUILDING BLOCK OPERATION (BEFORE ACTIVATION) IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL

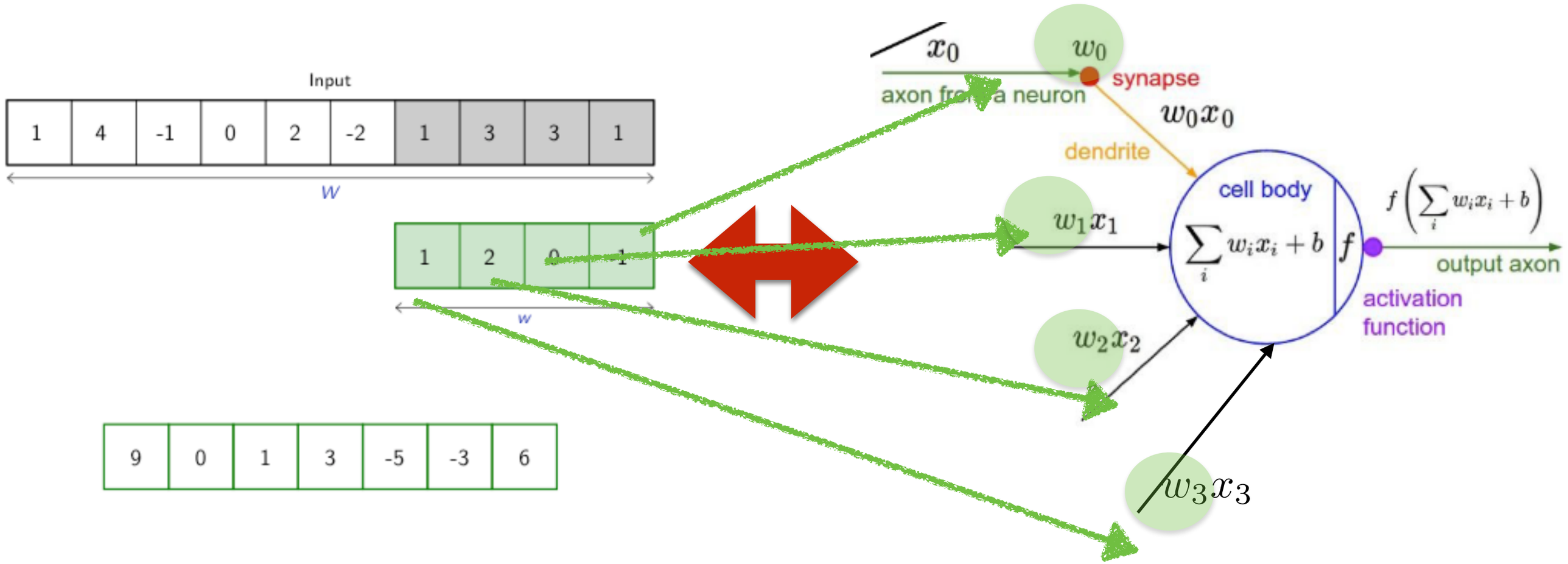




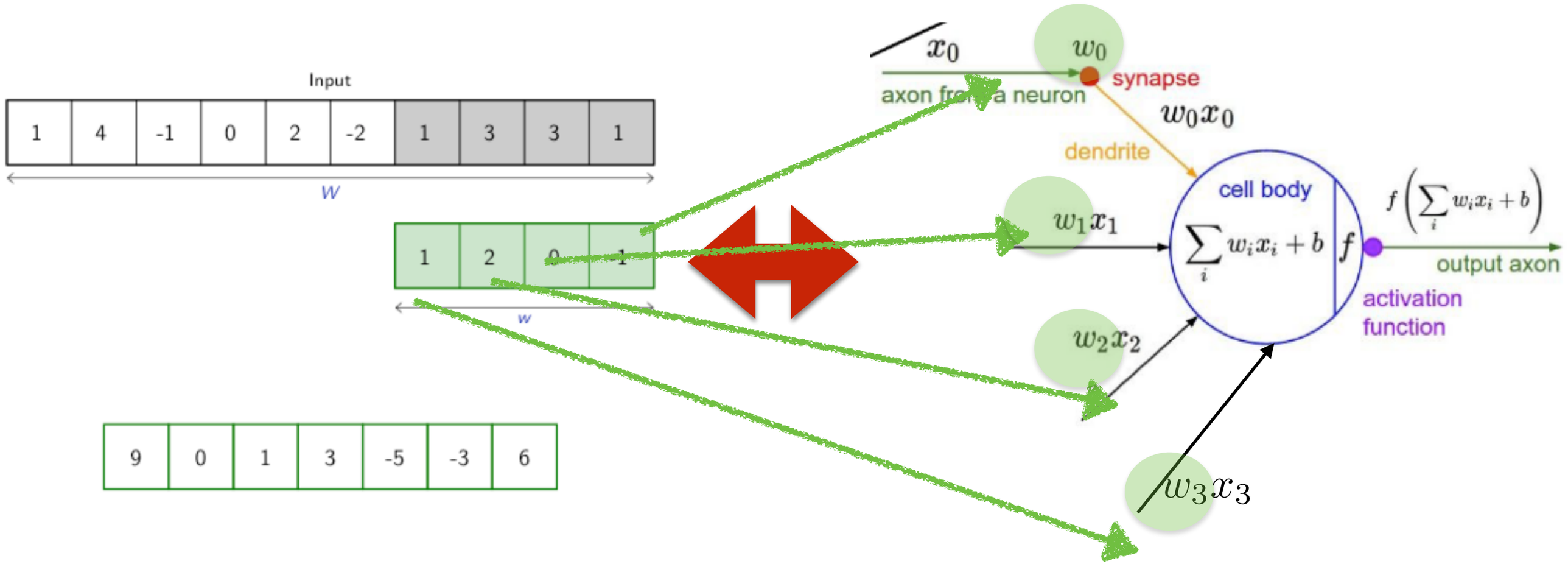
THE CONVOLUTION BUILDING BLOCK OPERATION (BEFORE ACTIVATION) IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL



THE CONVOLUTION BUILDING BLOCK OPERATION (BEFORE ACTIVATION) IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL



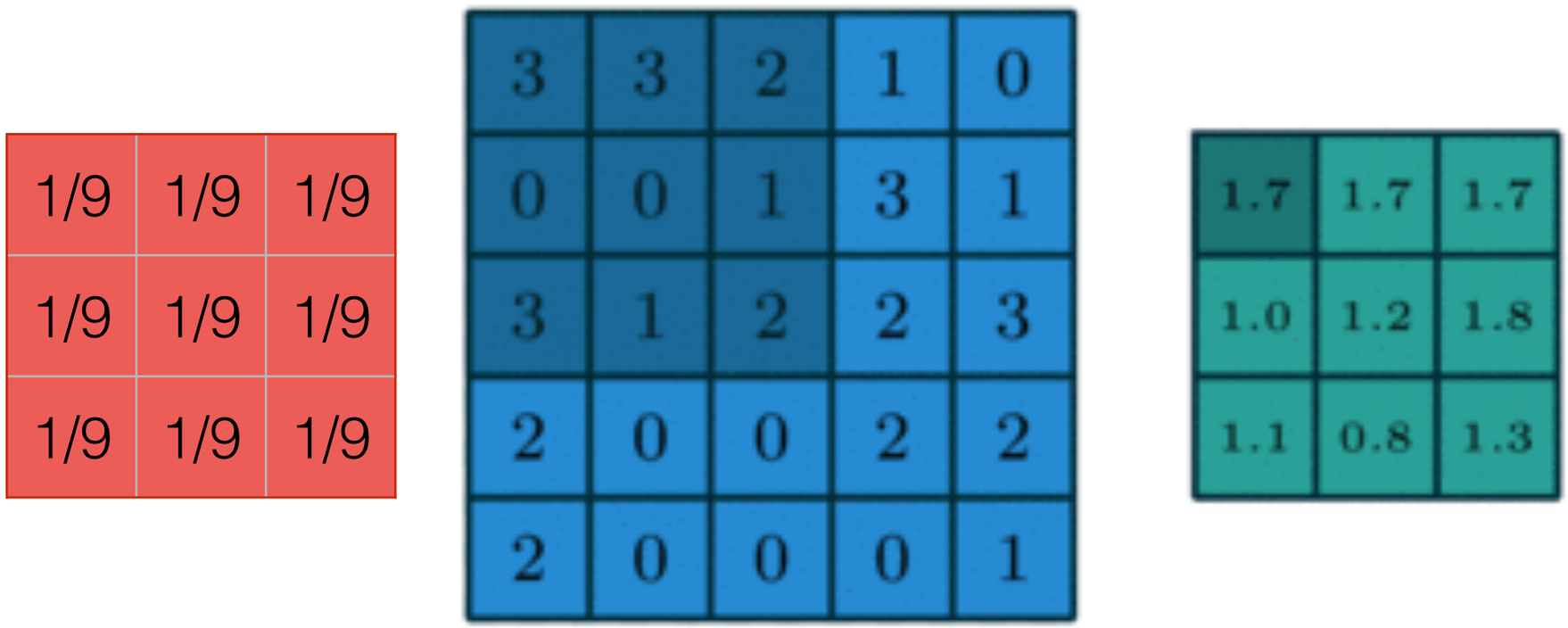
THE CONVOLUTION BUILDING BLOCK OPERATION (BEFORE ACTIVATION) IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL



WITH THE ADVANTAGE THAT THE SAME WEIGHTS ARE APPLIED TO ALL THE SIGNAL: TRANSLATION INVARIANCE

2-D CONVOLUTION

SAME IDEA, BUT THE KERNEL IS NOW 2D



KERNEL **INPUT (IMAGE)** **OUTPUT**

Credit: animations from https://github.com/vdumoulin/conv_arithmetic

2-D CONVOLUTION

SAME IDEA, BUT THE KERNEL IS NOW 2D

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

IN THE EXAMPLE: EACH 3x3 REGION GENERATES AN OUTPUT

$$Size_{output} = Size_{input} - Size_{kernel} + 1$$

Credit: animations from https://github.com/vdumoulin/conv_arithmetic

x_i

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

w_i

[weights]

x_i

w_i

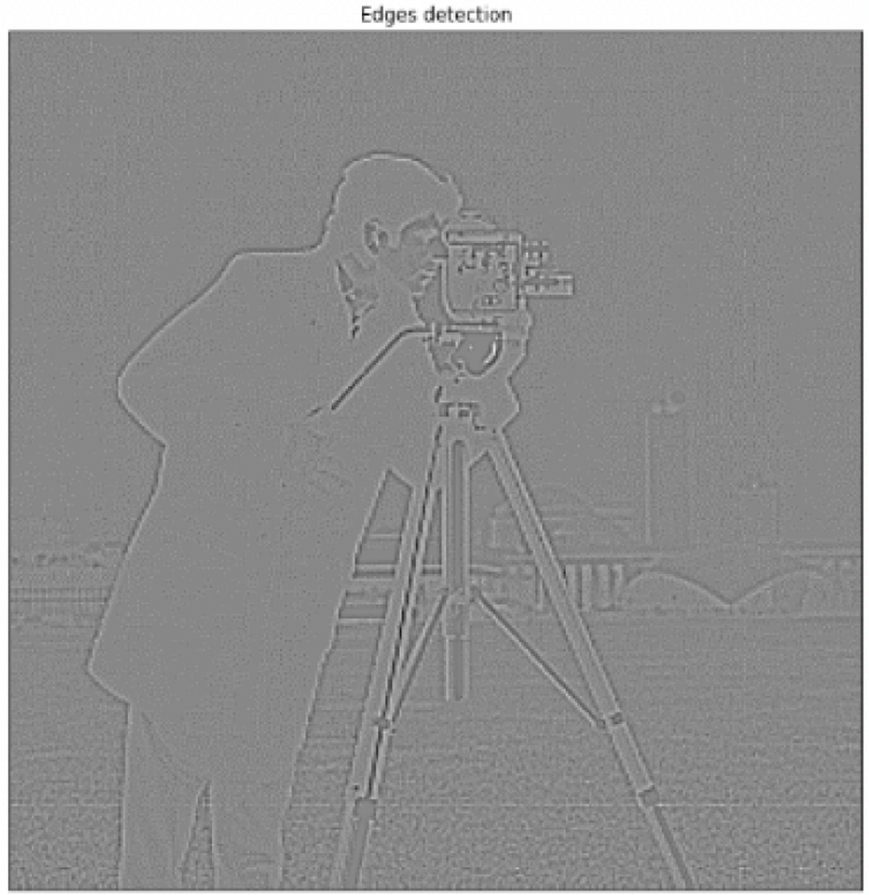
$\sum$

ACTIVATION FUNCTION
AT EVERY KERNEL POSITION

$$z(\mathbf{x}) = relu(w\mathbf{x} + b)$$

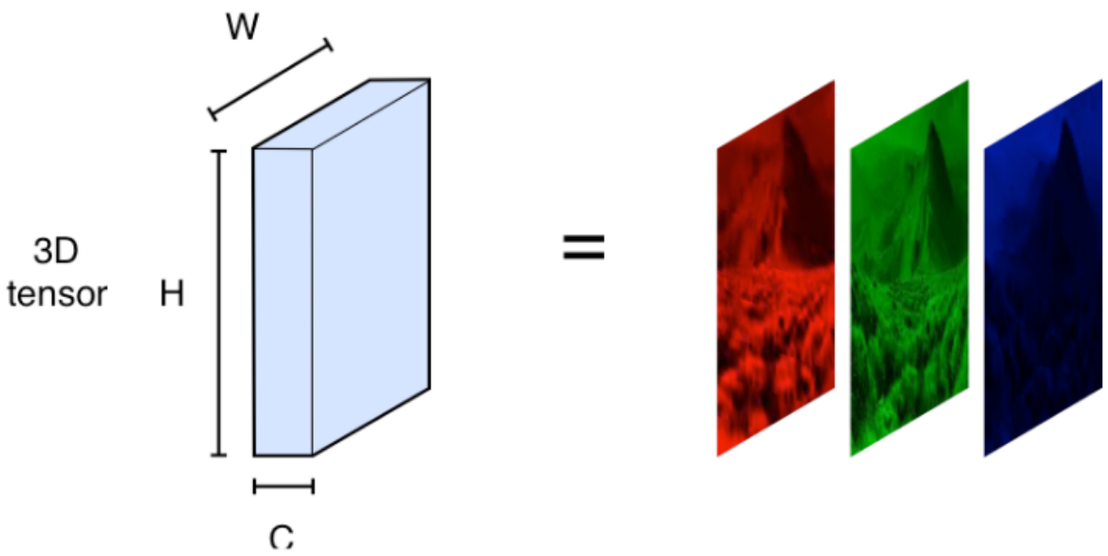
Conv Kernels are a long known method to identify patterns in images

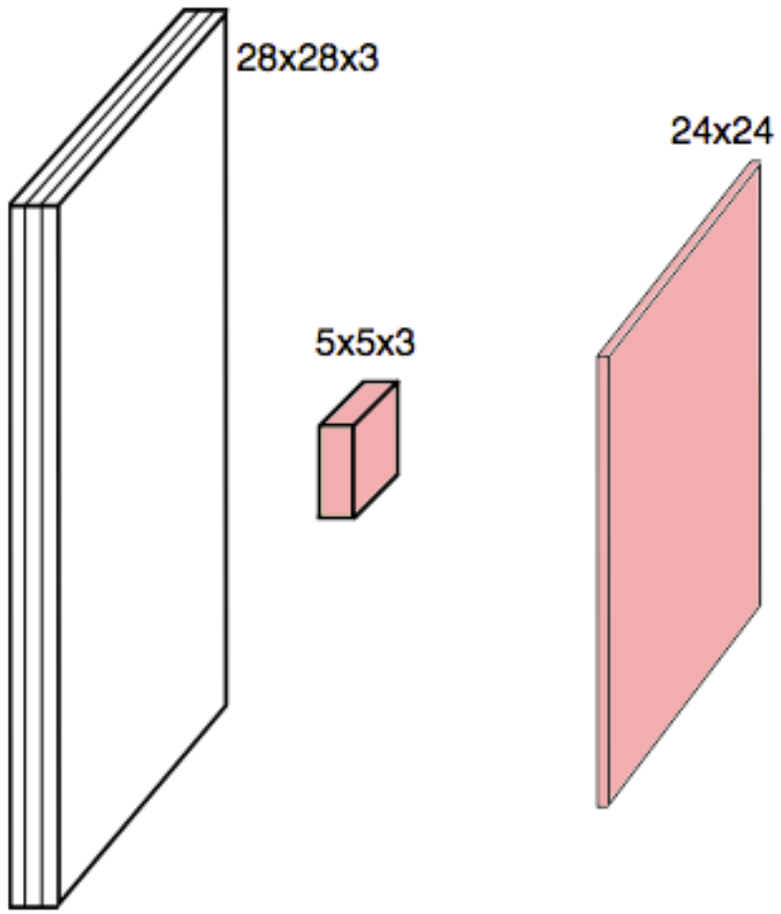
1	1	1
1	-8	1
1	1	1



CONVOLUTIONS CAN ALSO BE COMPUTED ACROSS CHANNELS (OR COLORS)

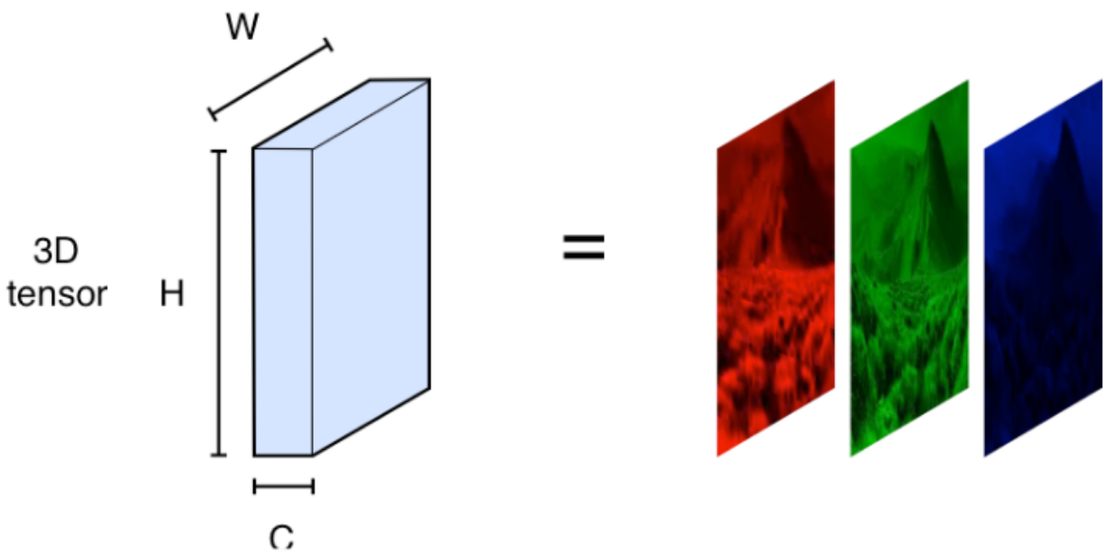
A COLOR IMAGE IS A TENSOR
OF SIZE height x width x channels

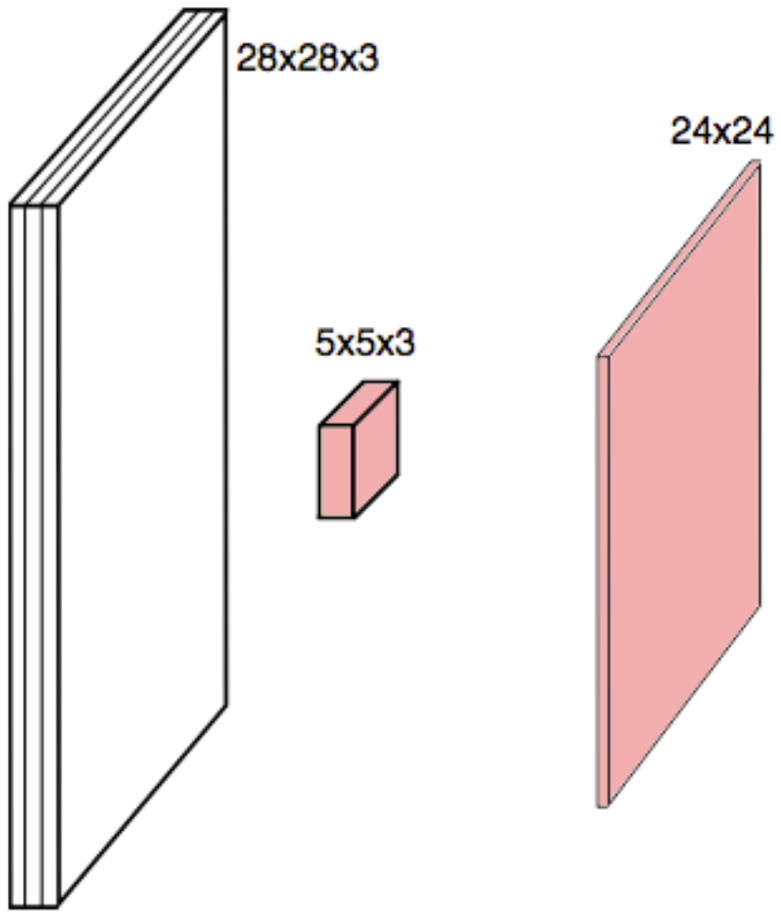




CONVOLUTIONS CAN ALSO BE COMPUTED ACROSS CHANNELS (OR COLORS)

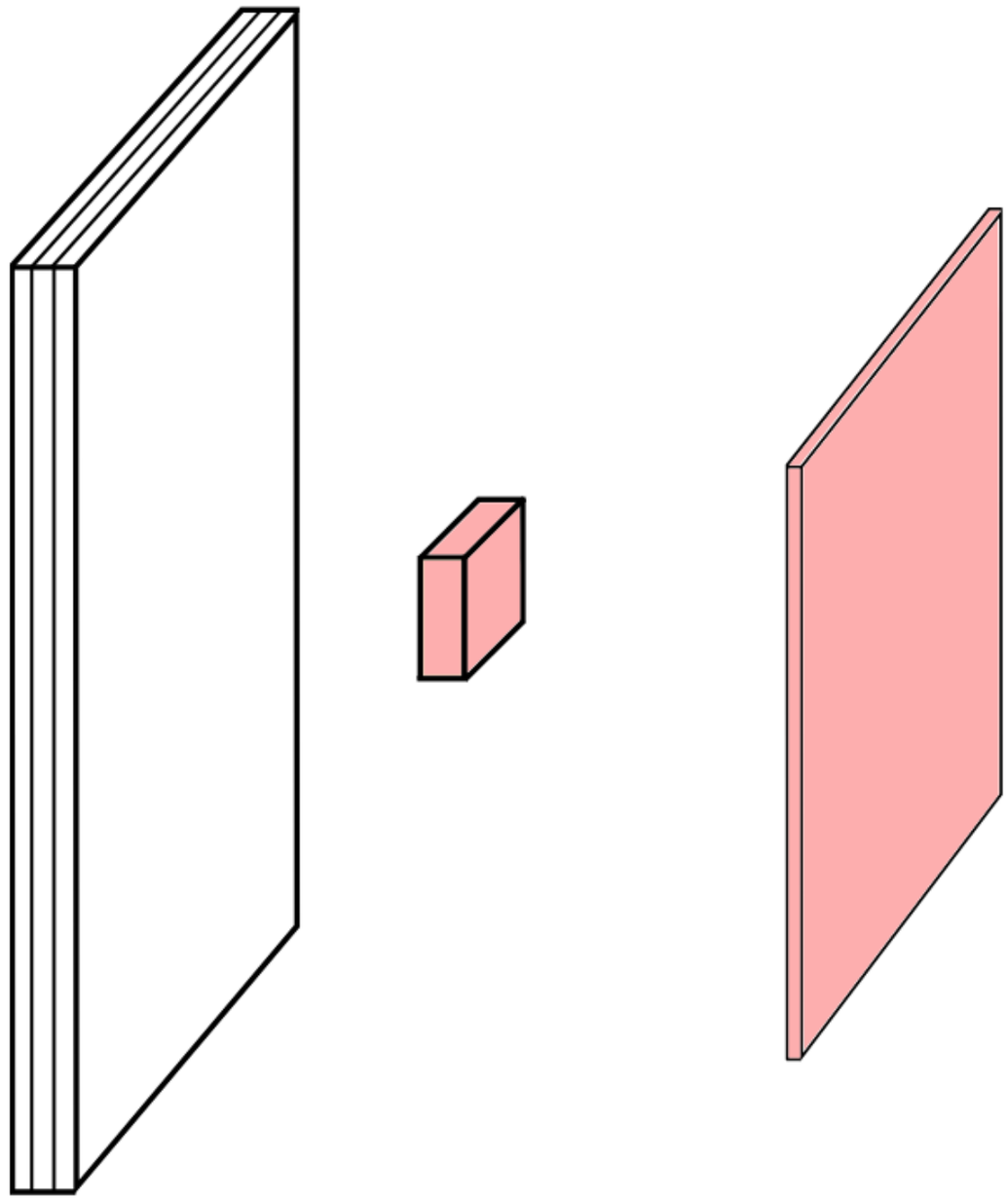
A COLOR IMAGE IS A TENSOR
OF SIZE height x width x channels





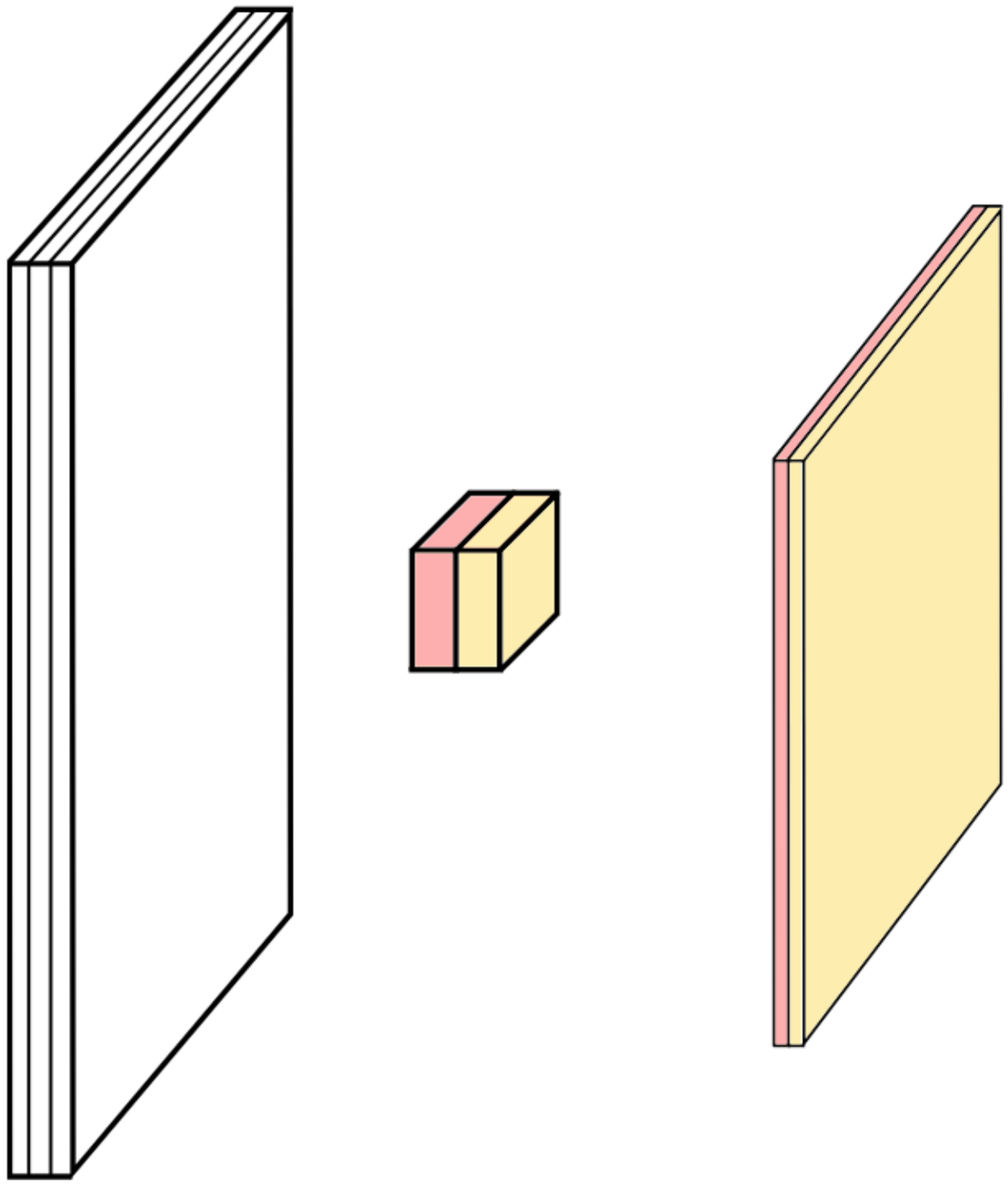
THEN THE KERNEL HAS ALSO 3 CHANNELS

MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED



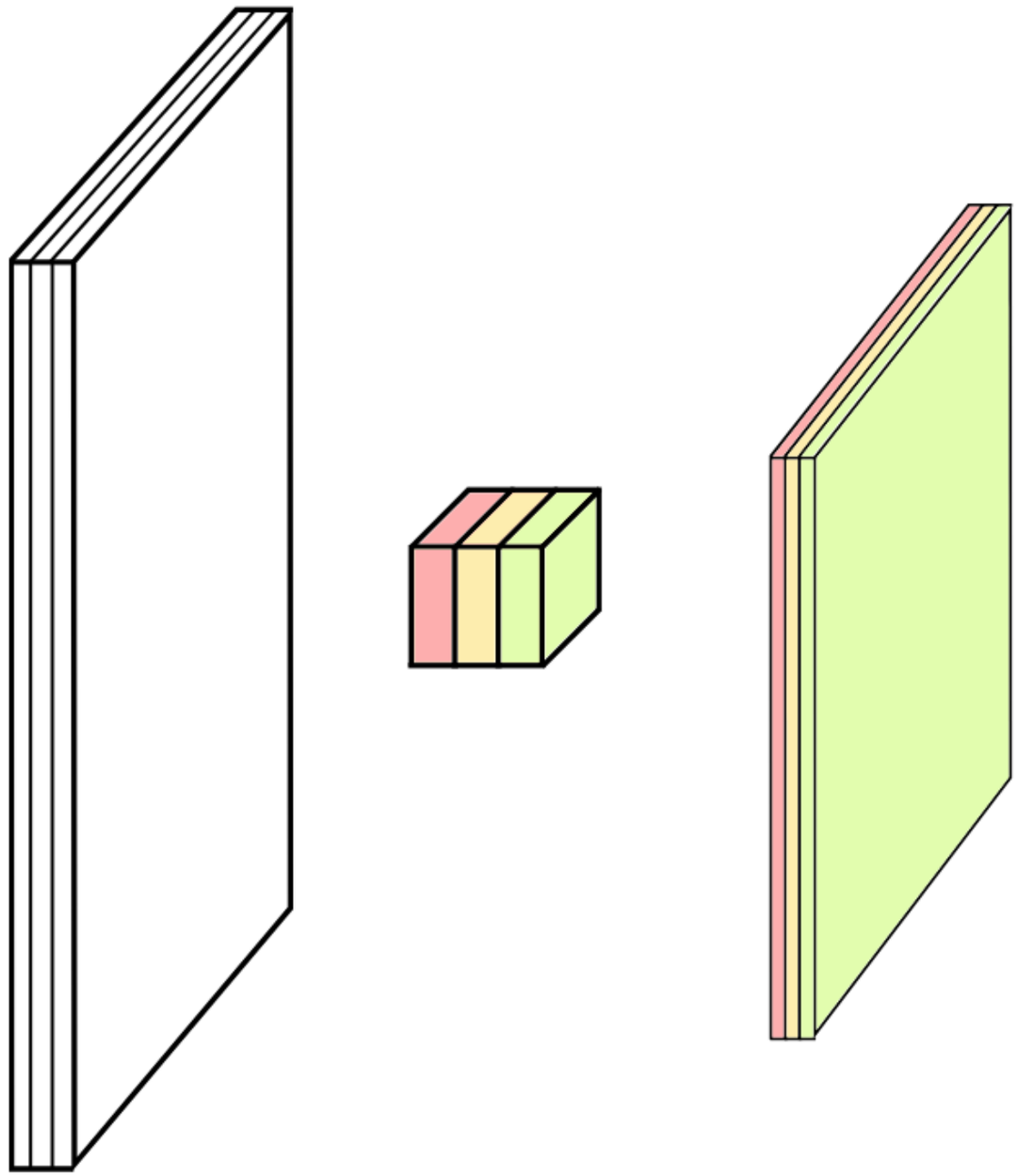
credit

MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED



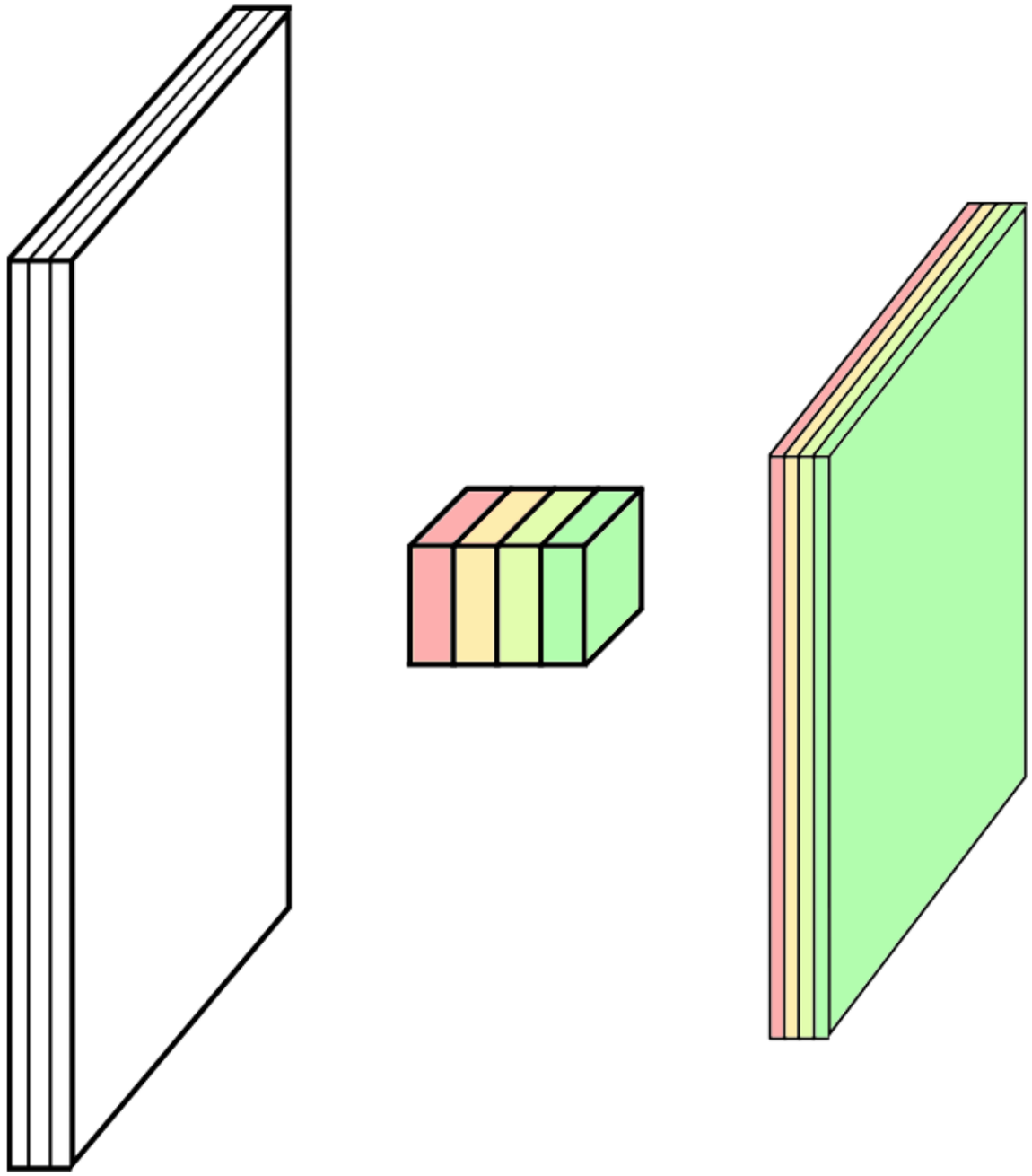
credit

MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED



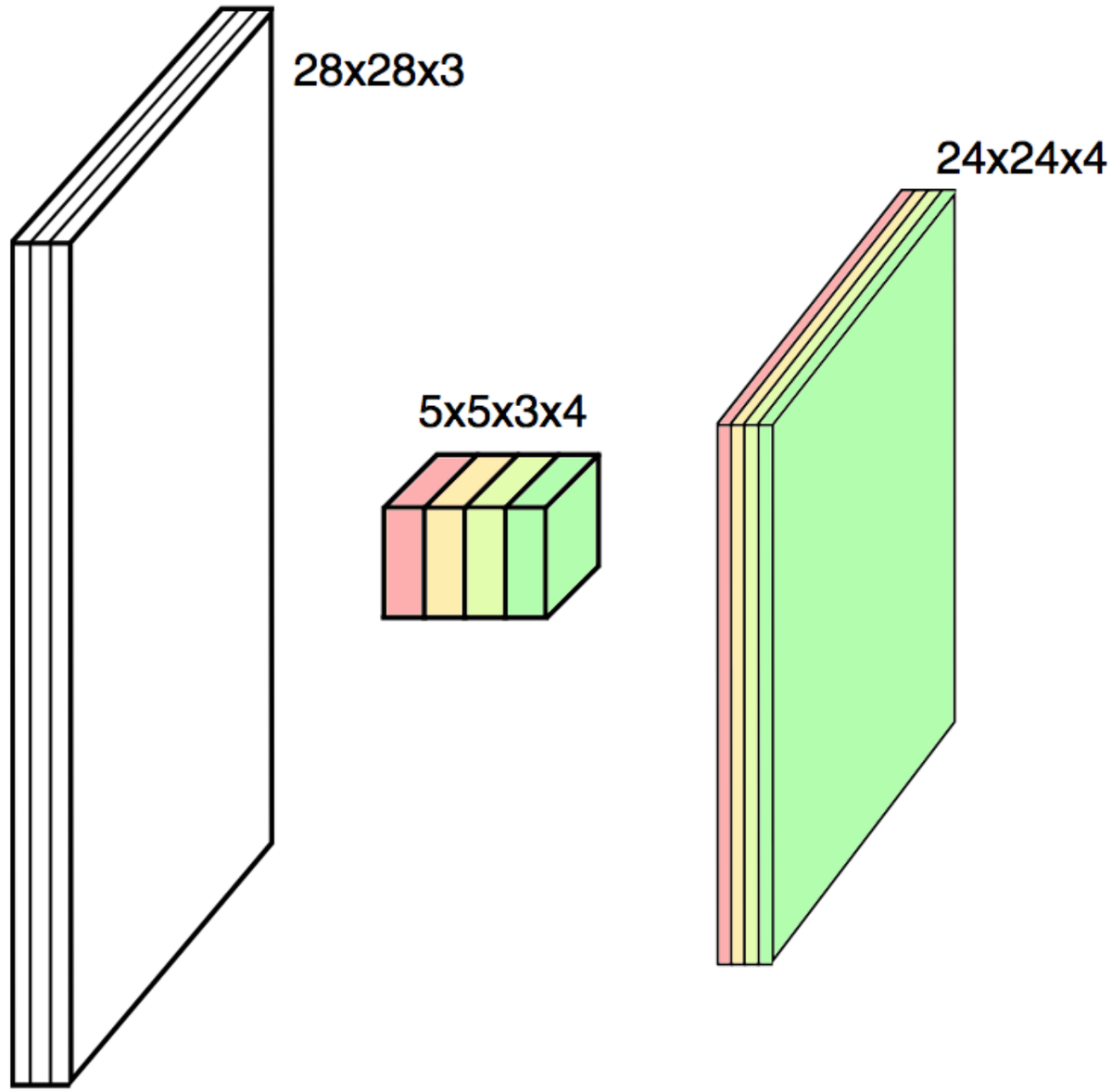
credit

MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED



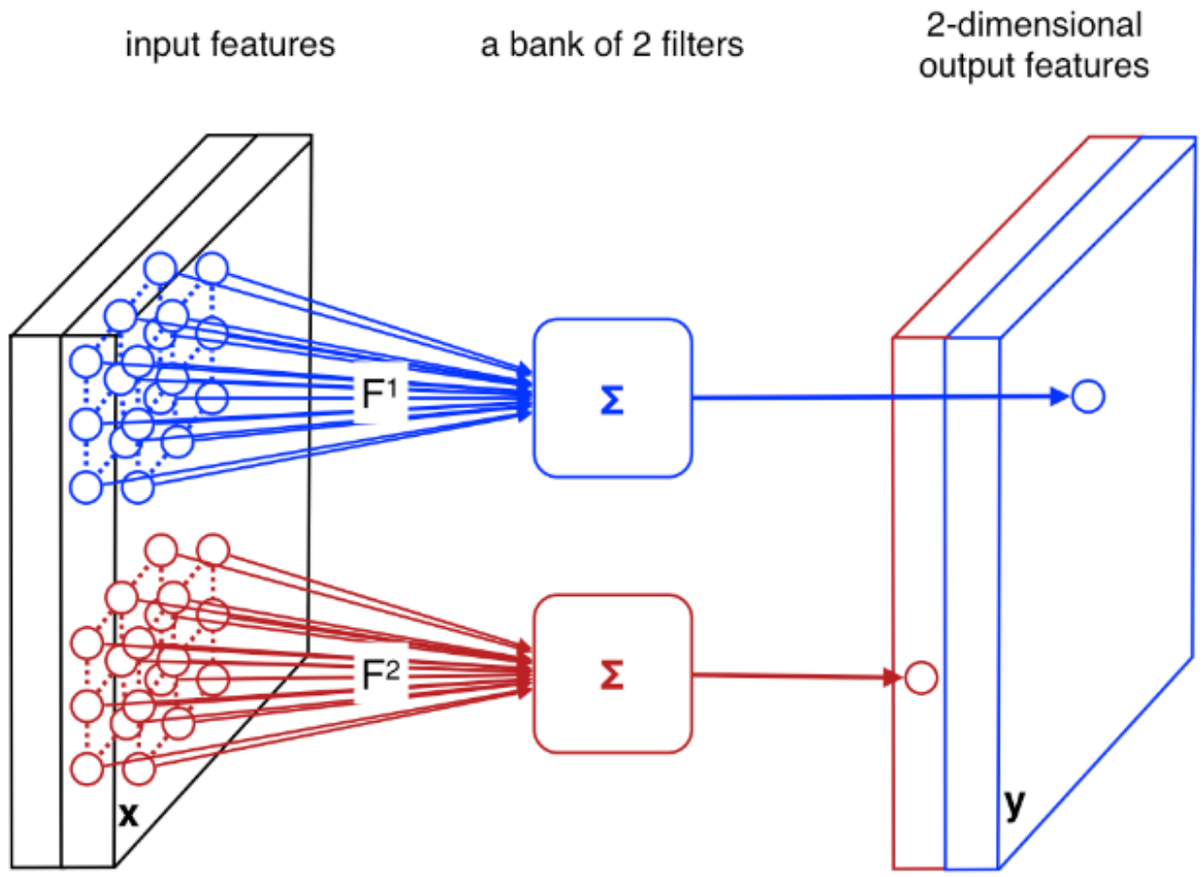
credit

MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED



credit

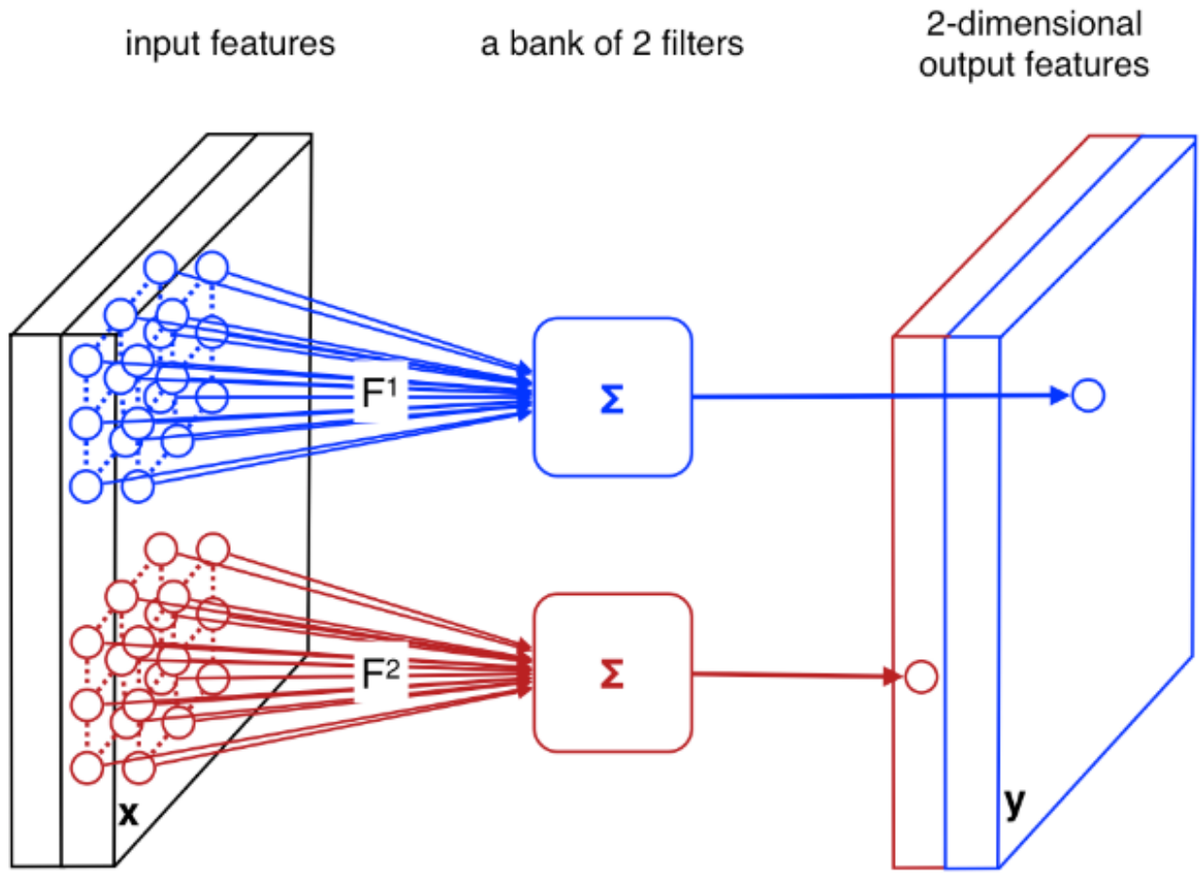
SINCE CONVOLUTIONS OUTPUT ONE SCALAR, THEY CAN BE SEEN AS AN INDIVIDUAL NEURON WITH A RECEPTIVE FIELD LIMITED TO THE KERNEL DIMENSIONS



Credit

SINCE CONVOLUTIONS OUTPUT ONE SCALAR, THEY CAN BE SEEN AS AN INDIVIDUAL NEURON WITH A RECEPTIVE FIELD LIMITED TO THE KERNEL DIMENSIONS

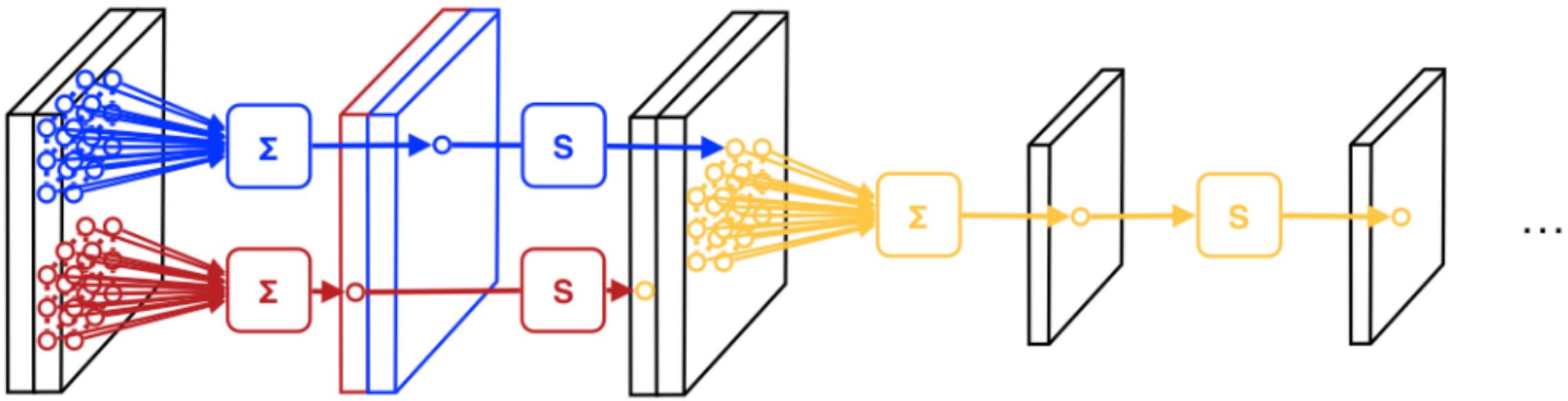
THE SAME NEURON IS FIRED WITH DIFFERENT AREAS FROM THE INPUT



Credit

DOWNSAMPLING

DOWNSAMPLING IS APPLIED TO REDUCE THE OVERALL SIZE OF TENSORS



POOLING

CONVOLUTIONS ARE OFTEN FOLLOWED BY AN OPERATION OF DOWNSAMPLING [POOLING]

VERY SIMPLE OPERATION - ONLY ONE OUT OF EVERY N PIXELS ARE KEPT

OFTEN MATCHED WITH AN INCREASE OF THE FEATURE CHANNELS

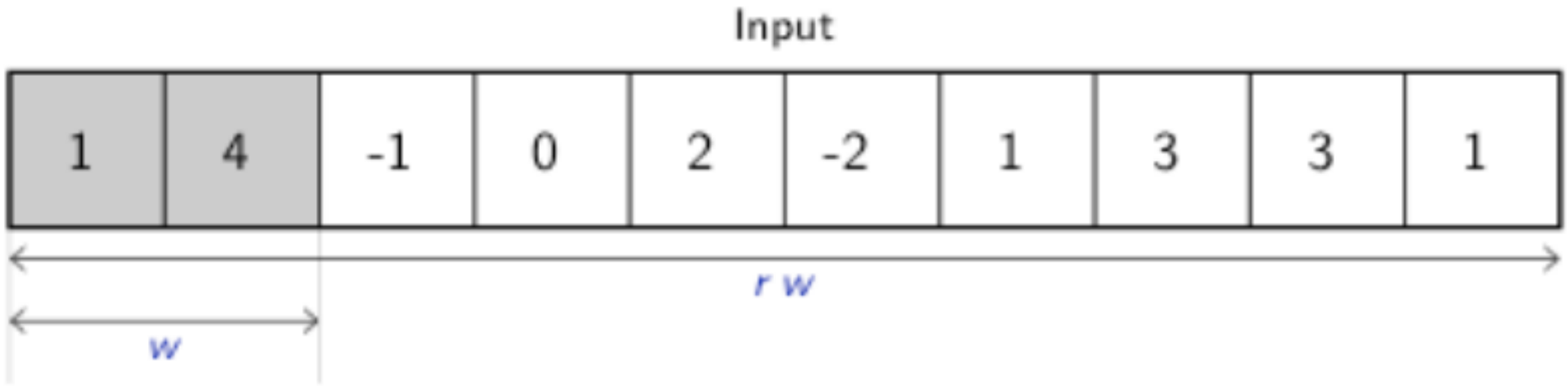
TYPES OF POOLING

SUM POOLING $$y = \sum x_{uv}$$

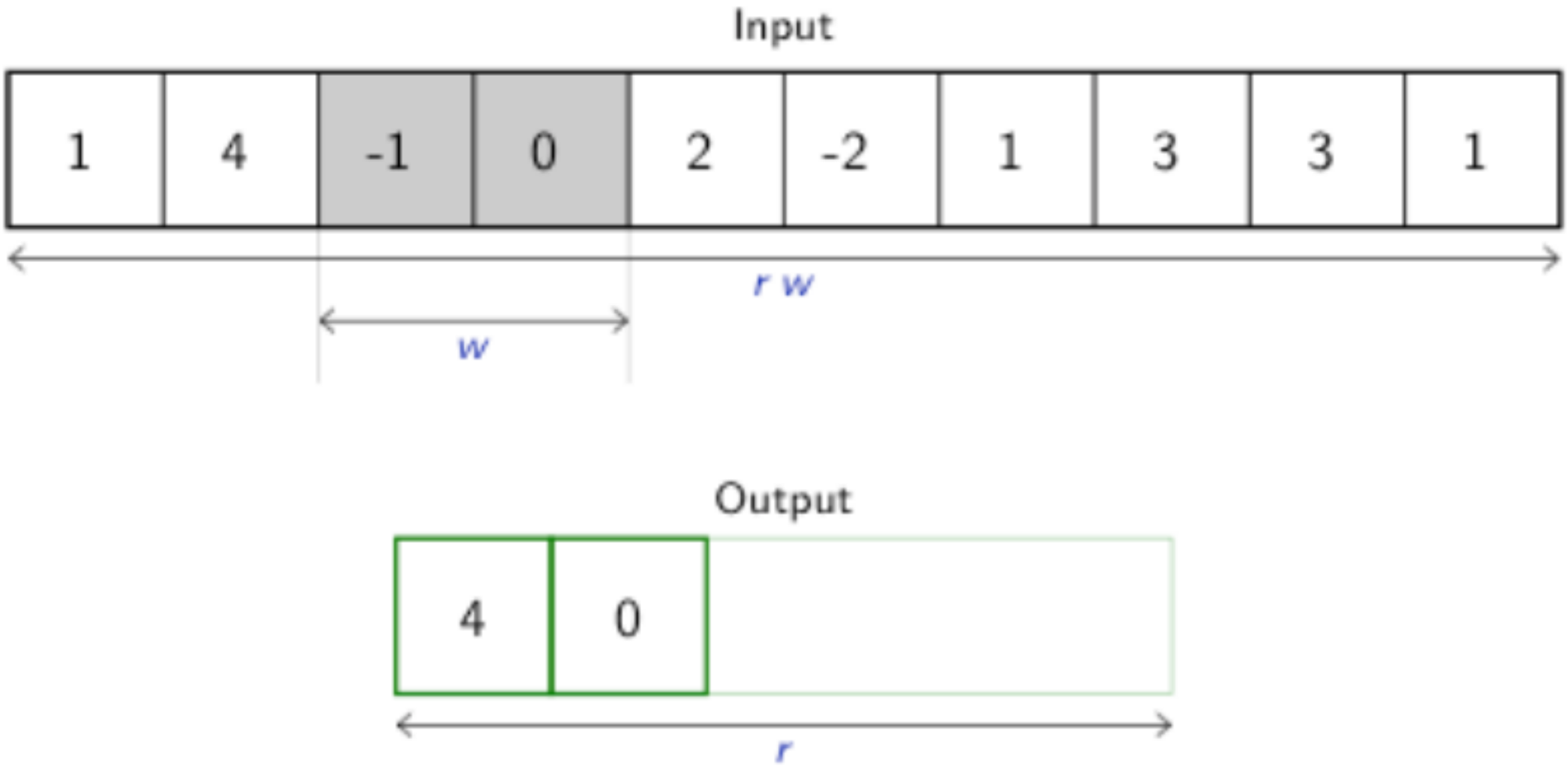
SQUARE SUM POOLING $$y = \sqrt{\sum x_{uv}^2}$$

MAX POOLING $$y = max(x_{uv})$$

MAX POOLING 1D

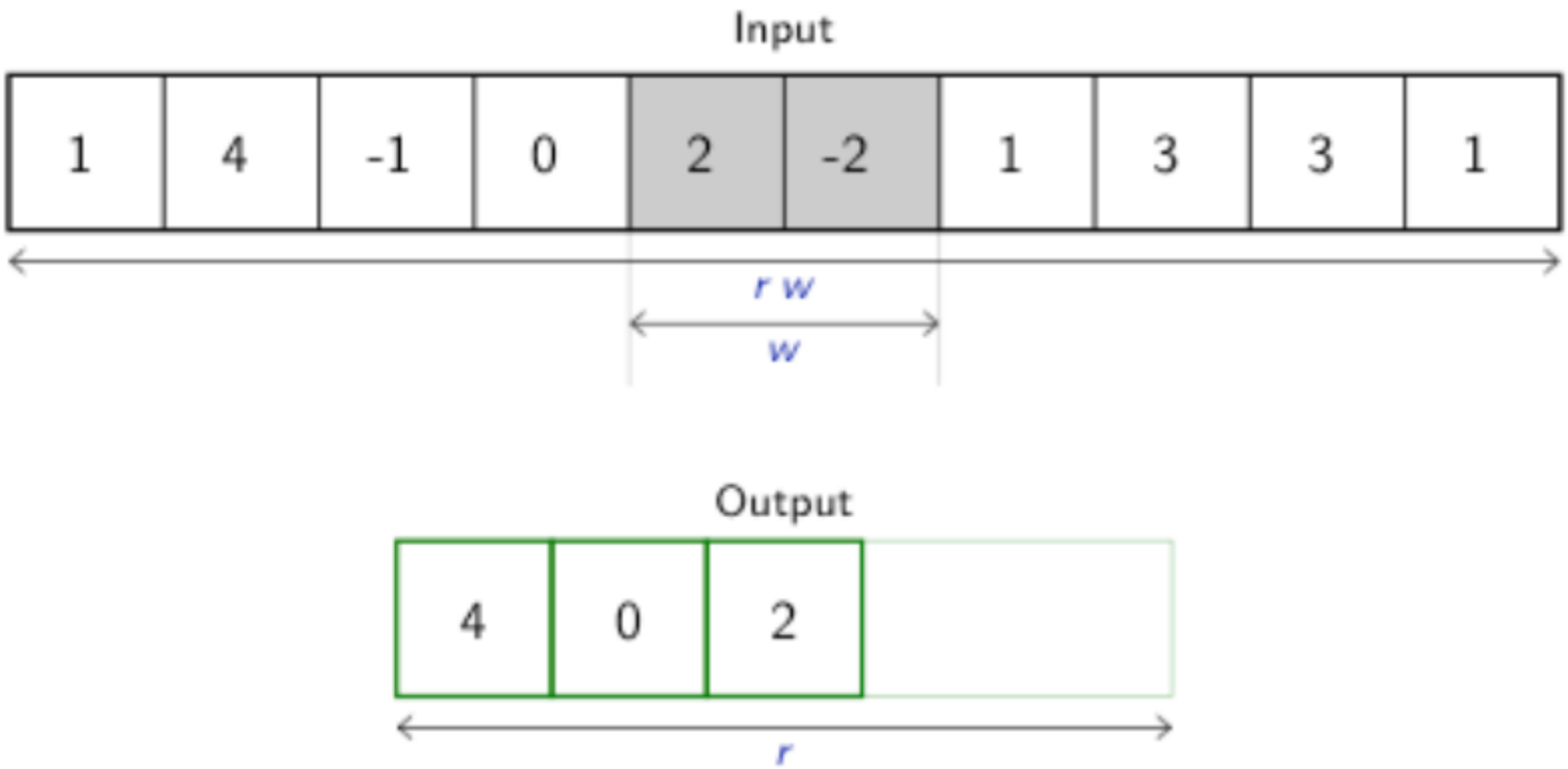


MAX POOLING 1D



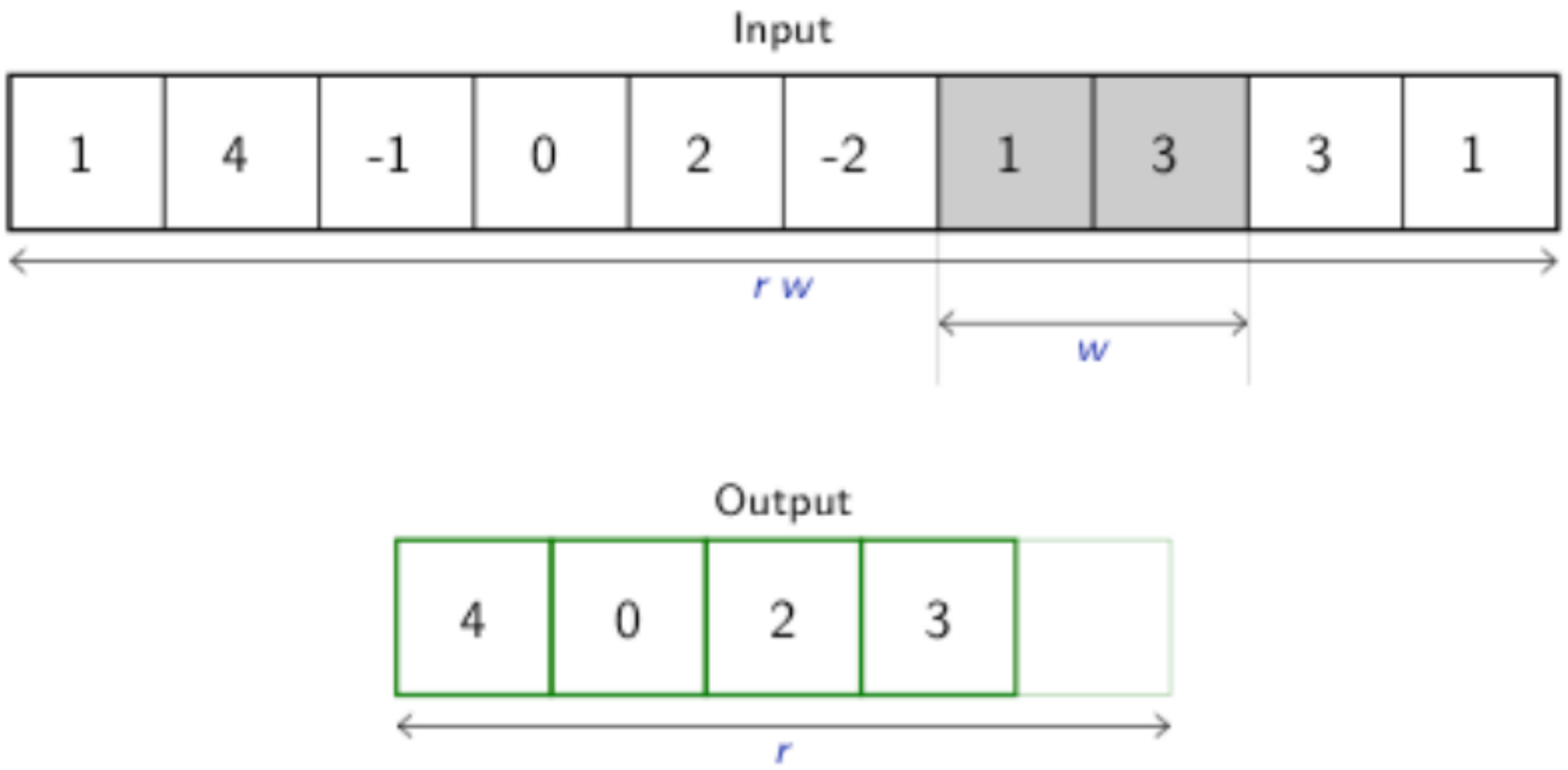
Credit: F. Fleuret

MAX POOLING 1D



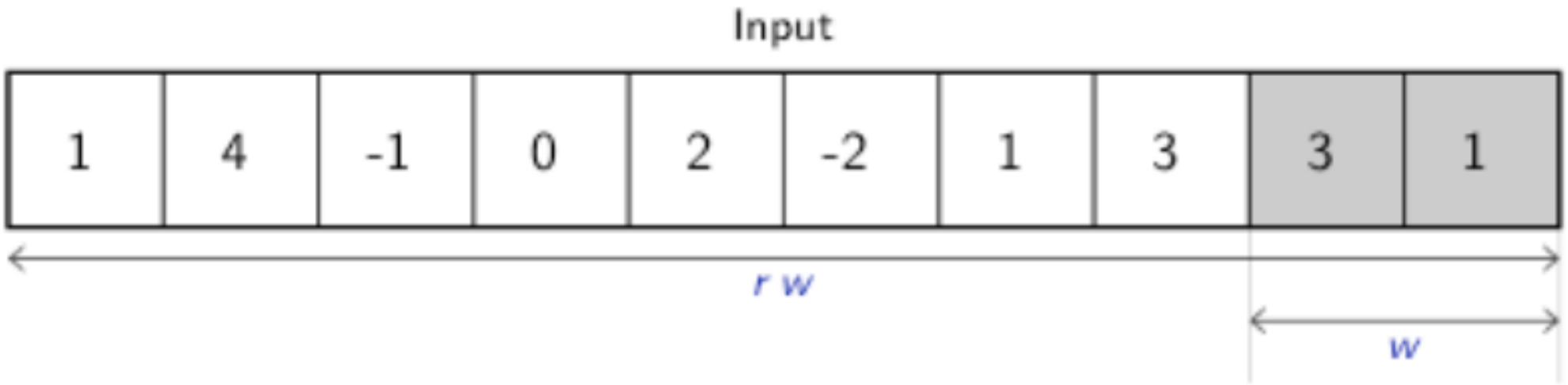
Credit: F. Fleuret

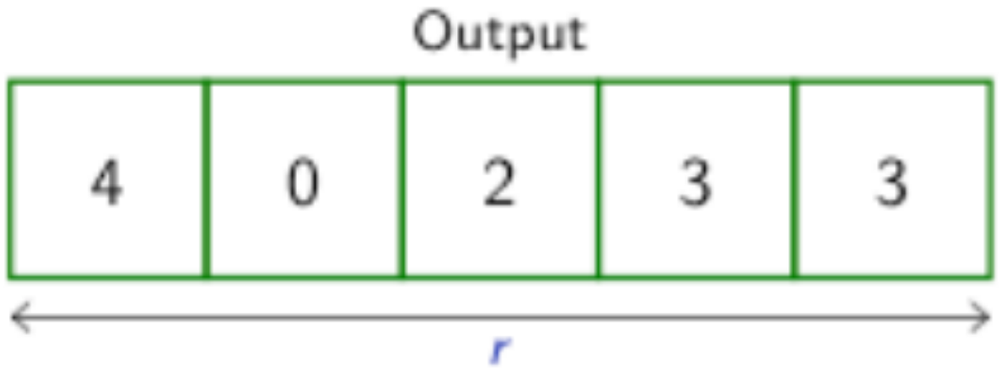
MAX POOLING 1D



Credit: F. Fleuret

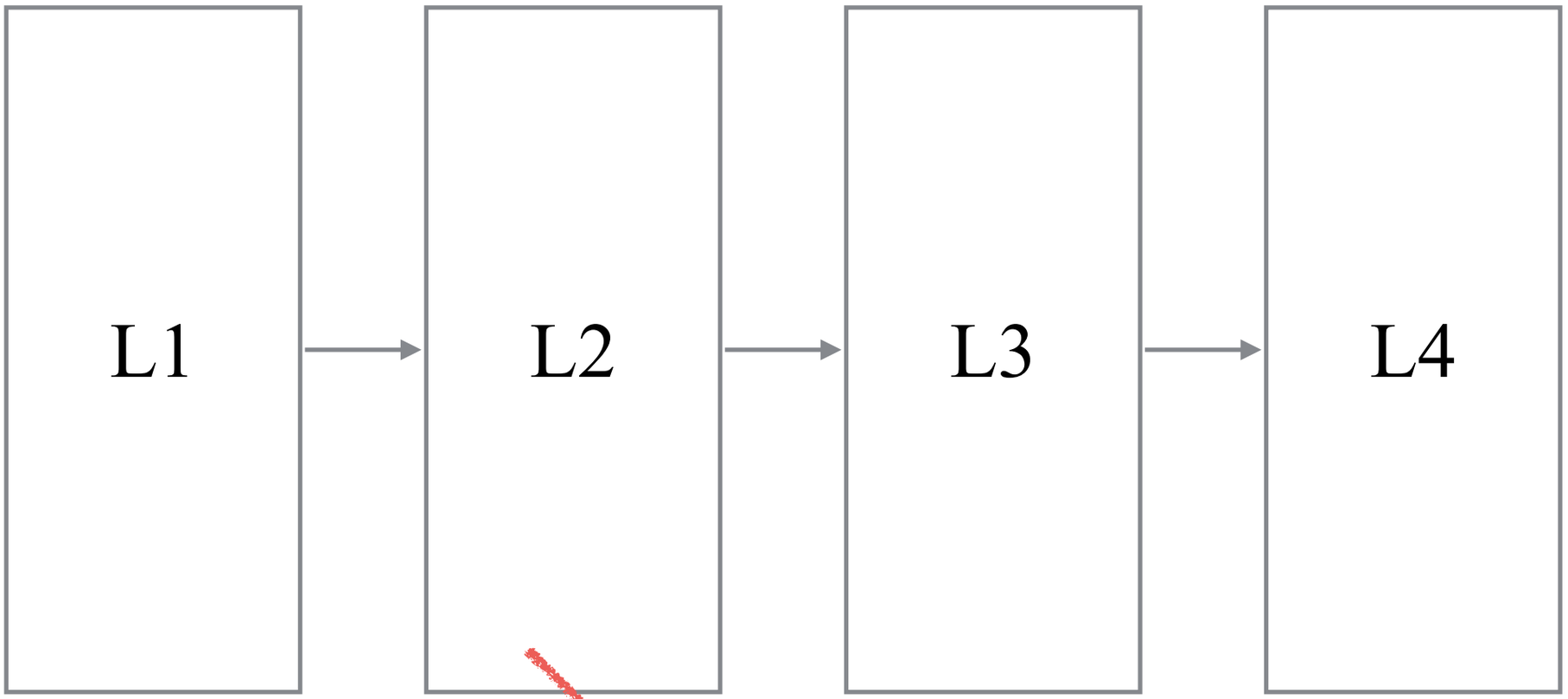
MAX POOLING 1D





Credit: F. Fleuret

CONVNET OR CNN



EACH BLOCK TYPICALLY MADE OF:

CONV

ACTIVATION

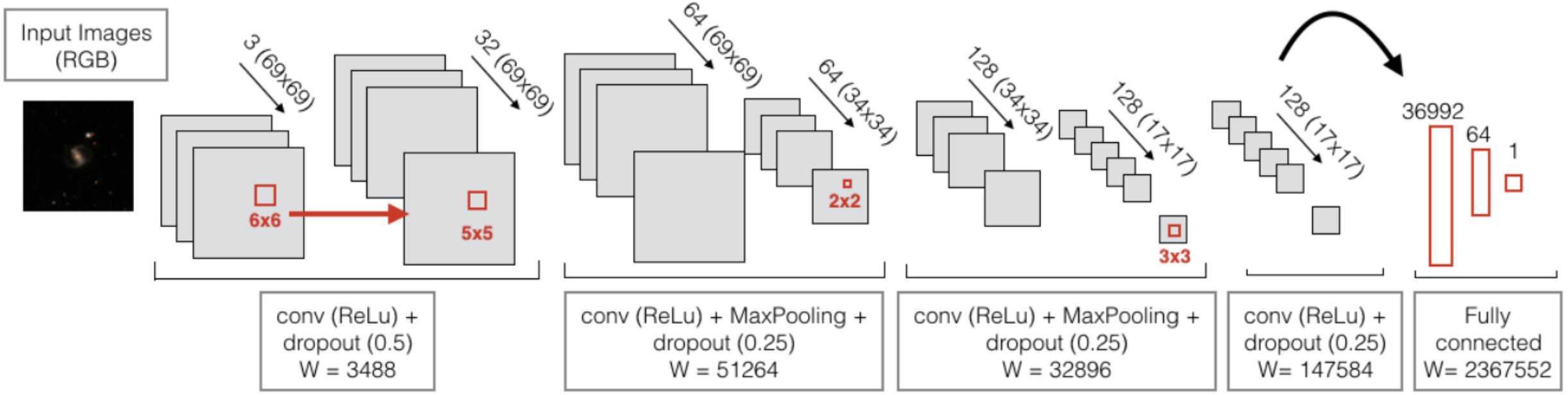
POOLING

(+dropout for training)

EXAMPLE OF VERY SIMPLE CNN

OVERALL:

- decrease of tensor size
- increase of depth

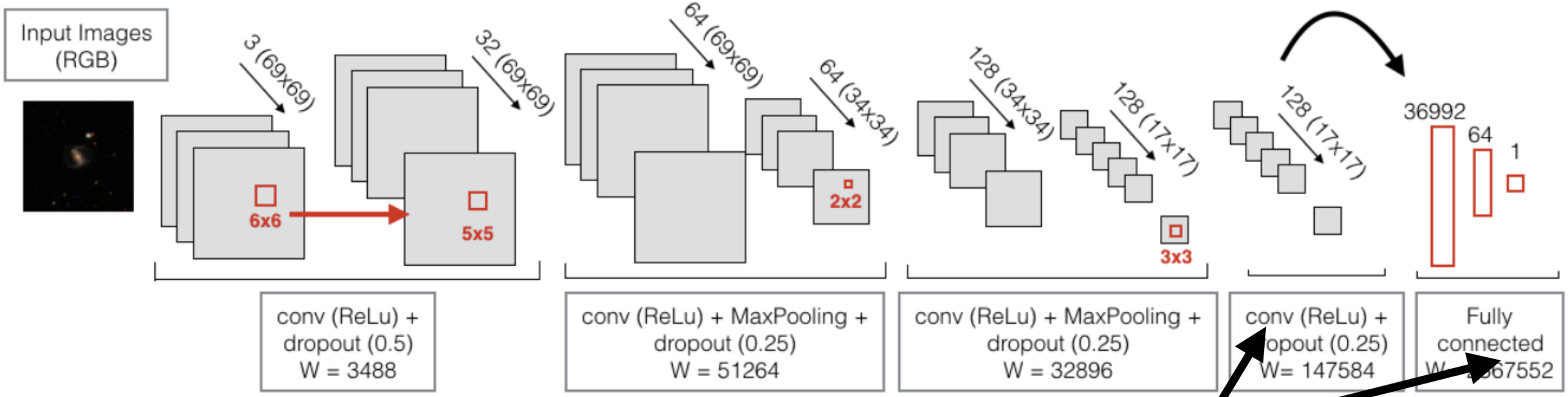


Dominguez-Sanchez+18

EXAMPLE OF VERY SIMPLE CNN

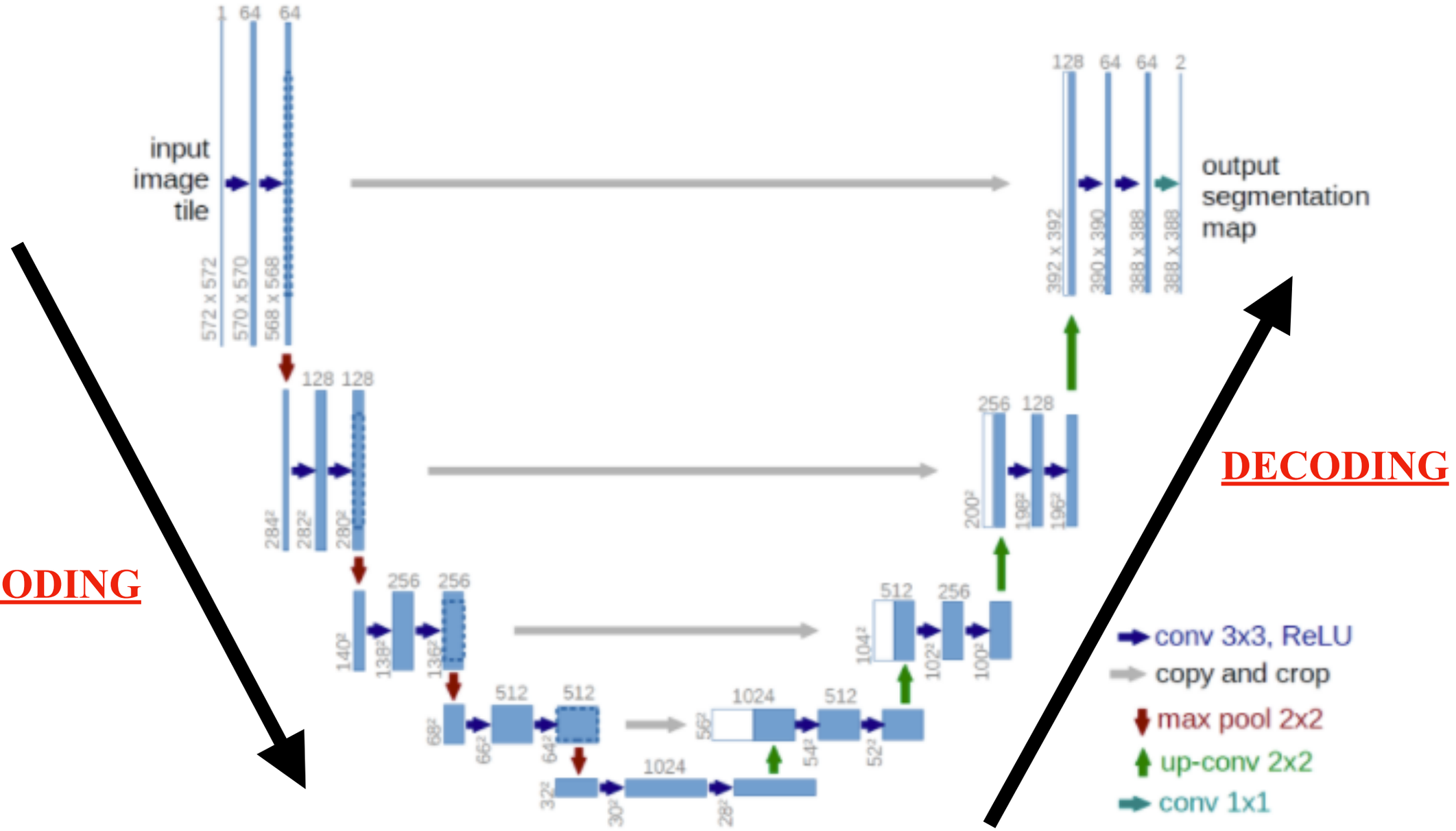
OVERALL:

- decrease of tensor size
- increase of depth



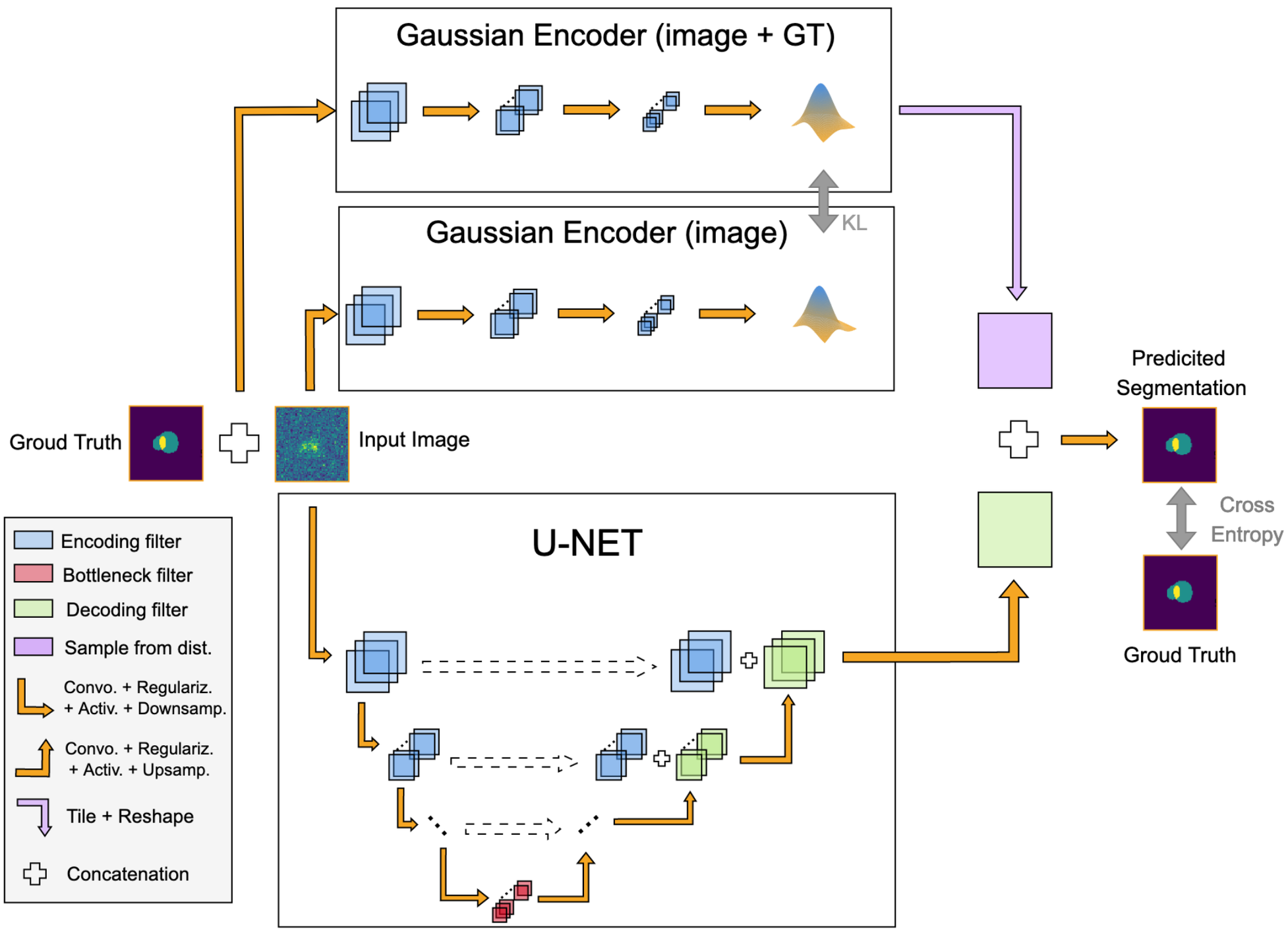
And you can mix the type of layers

Encoder-decoder: a powerfull architecture

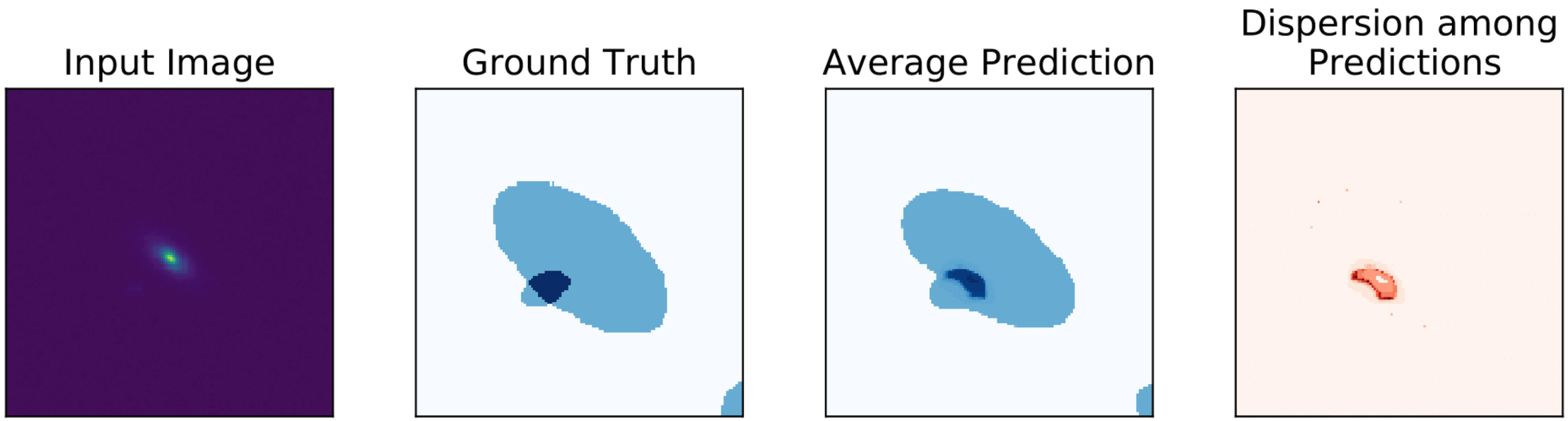


Ronnerberger+15

Encoder-decoder: a powerfull architecture

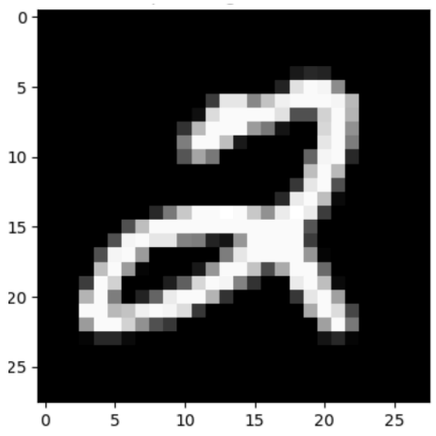


Encoder-decoder: a powerfull architecture



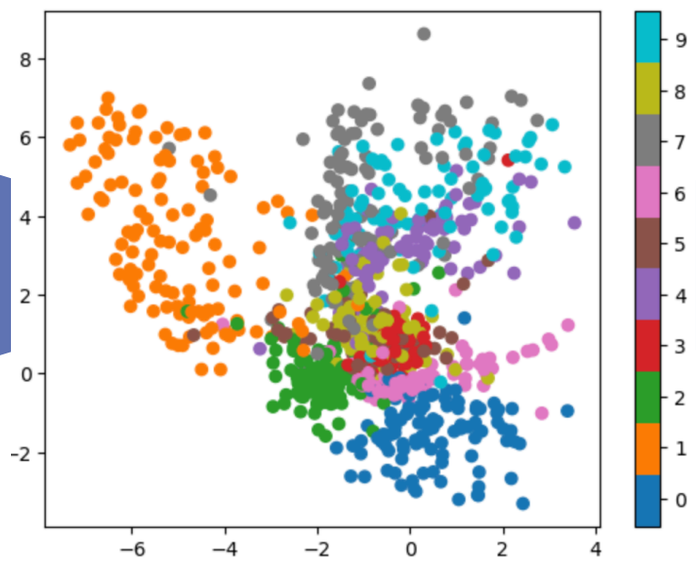
Encoder-decoder: a powerfull architecture

Input



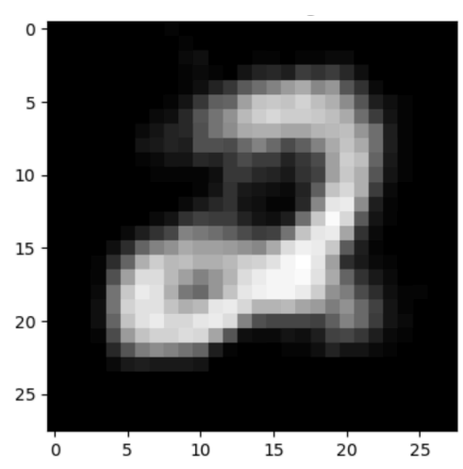
Encoder

Latent Space



Decoder

Reconstruction



Interactive interaction if time

A quick word about classic training problems and their possible solutions

OVER-FITTING

THE TEST STAYS CONSTANT
OR INCREASES

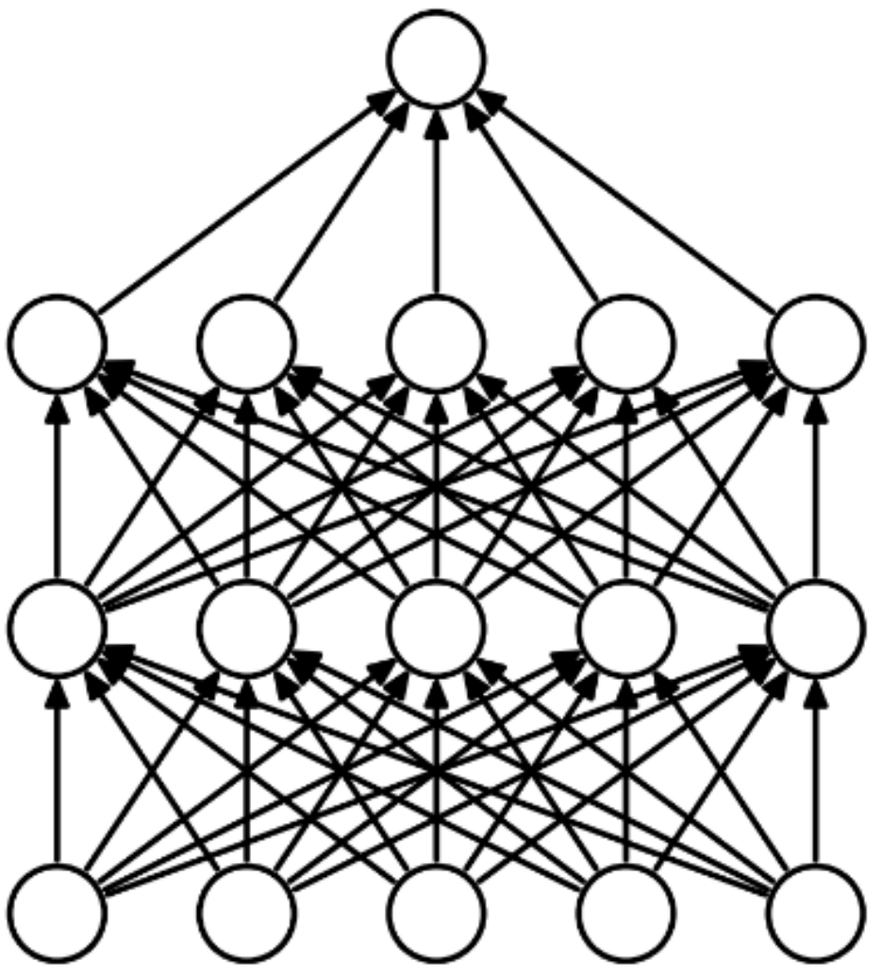
THE TRAINING
LOSS DECREASES

DROPOUT

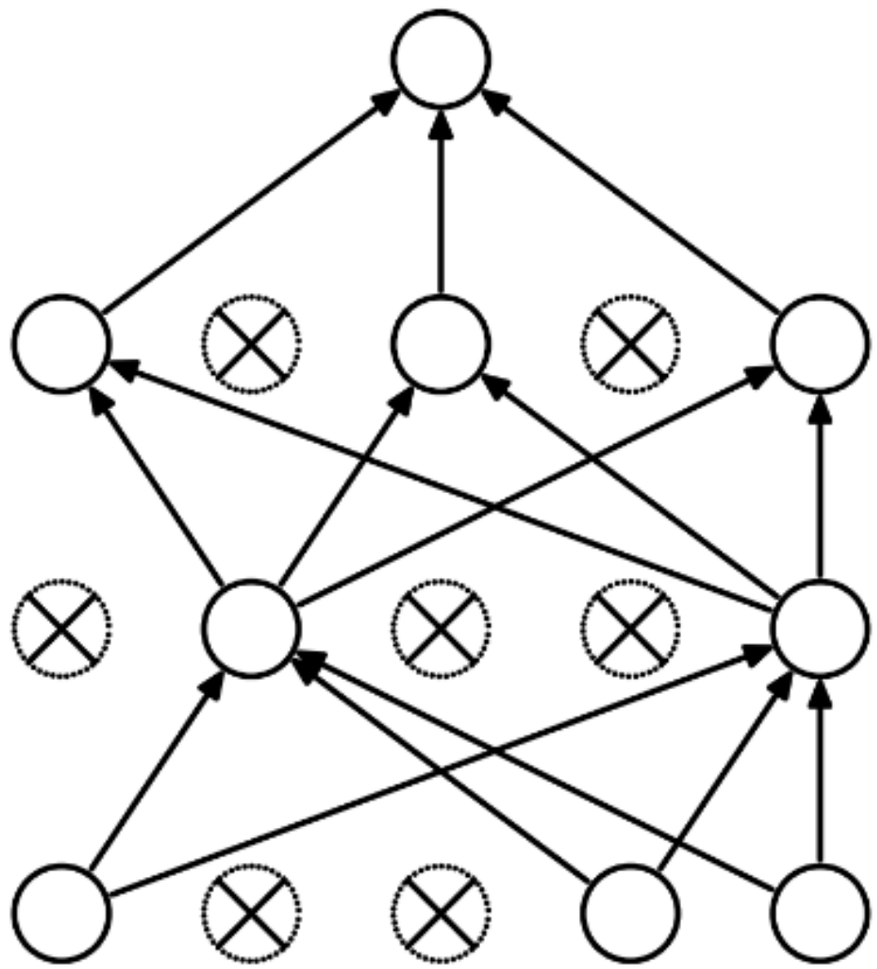
[Hinton+12]

- THE IDEA IS TO REMOVE NEURONS RANDOMLY DURING THE TRAINING

- ALL NEURONS ARE PUT BACK DURING THE TEST PHASE



(a) Standard Neural Net



(b) After applying dropout.

DROPOUT

WHY DOES IT WORK?

1. SINCE NEURONS ARE REMOVED RANDOMLY, IT AVOIDS CO-ADAPTATION AMONG THEMSELVES

2. DIFFERENT SETS OF NEURONS WHICH ARE SWITCHED OFF, REPRESENT A DIFFERENT ARCHITECTURE AND ALL THESE DIFFERENT ARCHITECTURES ARE TRAINED IN PARALLEL. FOR N NEURONS ATTACHED TO DROPOUT, THE NUMBER OF SUBSET ARCHITECTURES FORMED IS 2^N. SO IT AMOUNTS TO PREDICTION BEING AVERAGED OVER THESE ENSEMBLES OF MODELS.

VANISHING / EXPLODING GRADIENT PROBLEM

<u>REMEMBER THAT:</u>

$$y_{i+1} = \sigma[\sum w_i y_i]$$

output layer i+1

activation function

weights

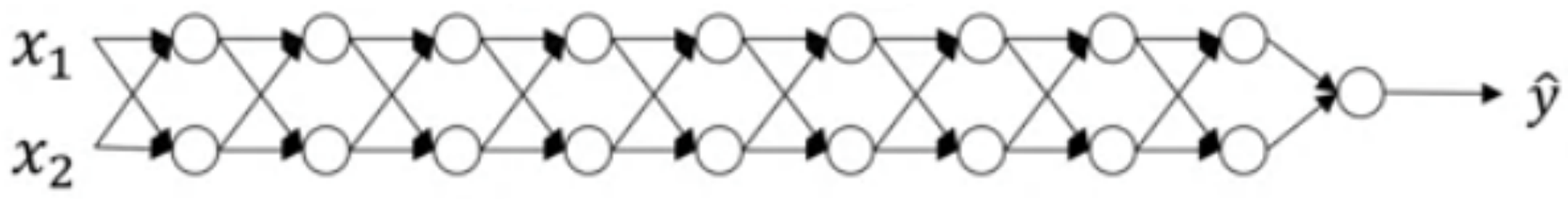
output layer i

VANISHING / EXPLODING GRADIENT PROBLEM

<u>WITH MANY LAYERS:</u>

$$y_n = \sigma\left(...\sigma\left(...\sigma\left(\sum w_0 x\right)\right)\right)$$

VANISHING/EXPLODING GRADIENT PROBLEM



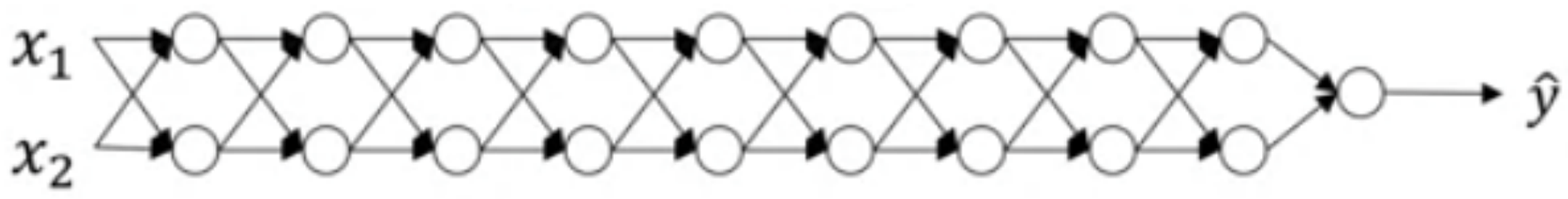
$$w_i = \begin{pmatrix} w_i^0 & 0 \\ 0 & w_i^1 \end{pmatrix} \qquad \hat{y} = x \prod_n w_i$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

IF WEIGHTS ARE ALL INITIALIZED TO VALUES <<1:

$$\hat{y} \to 0$$

VANISHING/EXPLODING GRADIENT PROBLEM



$$w_i = \begin{pmatrix} w_i^0 & 0 \\ 0 & w_i^1 \end{pmatrix}$$

$$\hat{y} = x \prod_n w_i$$

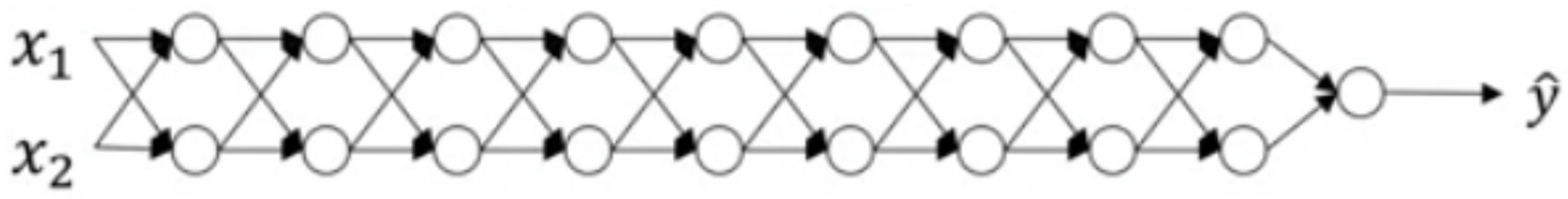
$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

IF WEIGHTS ARE ALL INITIALIZED TO VALUES >1:

$$\hat{y} \to \infty$$

EXPLODING GRADIENT

VANISHING/EXPLODING GRADIENT PROBLEM



$$w_i = \begin{pmatrix} w_i^0 & 0 \\ 0 & w_i^1 \end{pmatrix}$$

$$\hat{y} = x \prod_n w_i$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

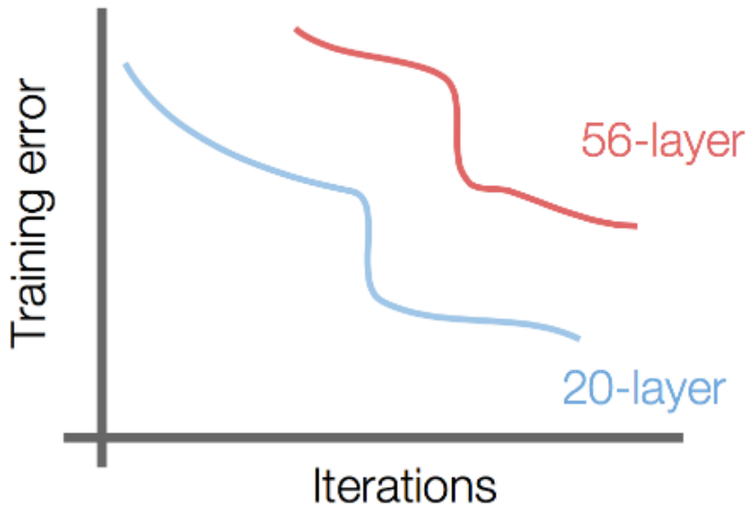
IF WEIGHTS ARE ALL INITIALIZED TO VALUES >1:

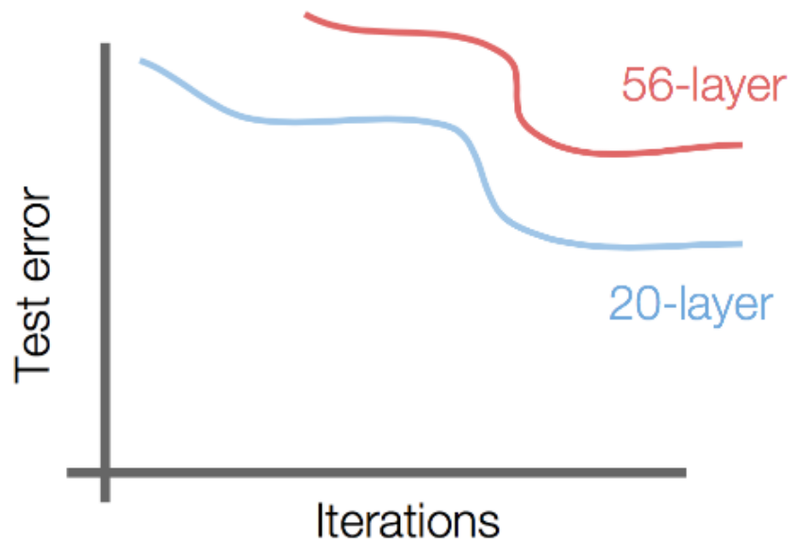
$$w_i^L \rightarrow \infty$$

EXPLODING GRADIENT

VANISHING/EXPLODING GRADIENT PROBLEM

TRAINING BECOMES UNSTABLE
VERY SLOW OR NO CONVERGENCE





BATCH NORMALIZATION

[SZEGEDY+15]

A SOLUTION TO KEEP REASONABLE VALUES OF THE ACTIVATIONS IN DEEP NETWORKS

BATCH NORMALIZATION PREVENTS LOW OR LARGE VALUES BY RE-NORMALIZING THE VALUES BEFORE ACTIVATION FOR EVERY BATCH

INPUT

$$\hat{y}_i = \gamma \frac{y_i - E(y_i)}{\sigma(y_i)} + \beta$$

NORMALIZED INPUT

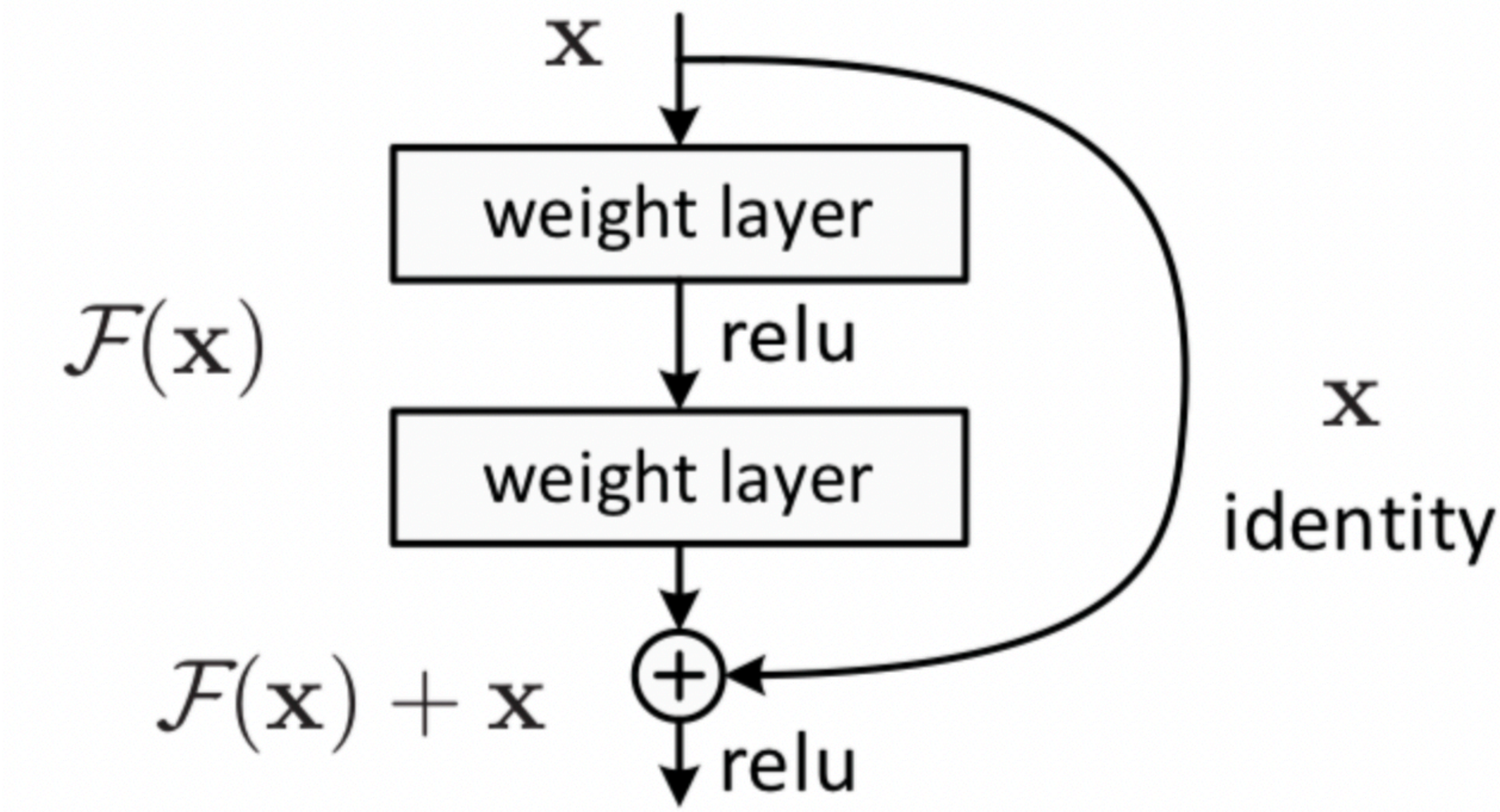
SCATTER

BATCH NORMALIZATION

[SZEGEDY+15]

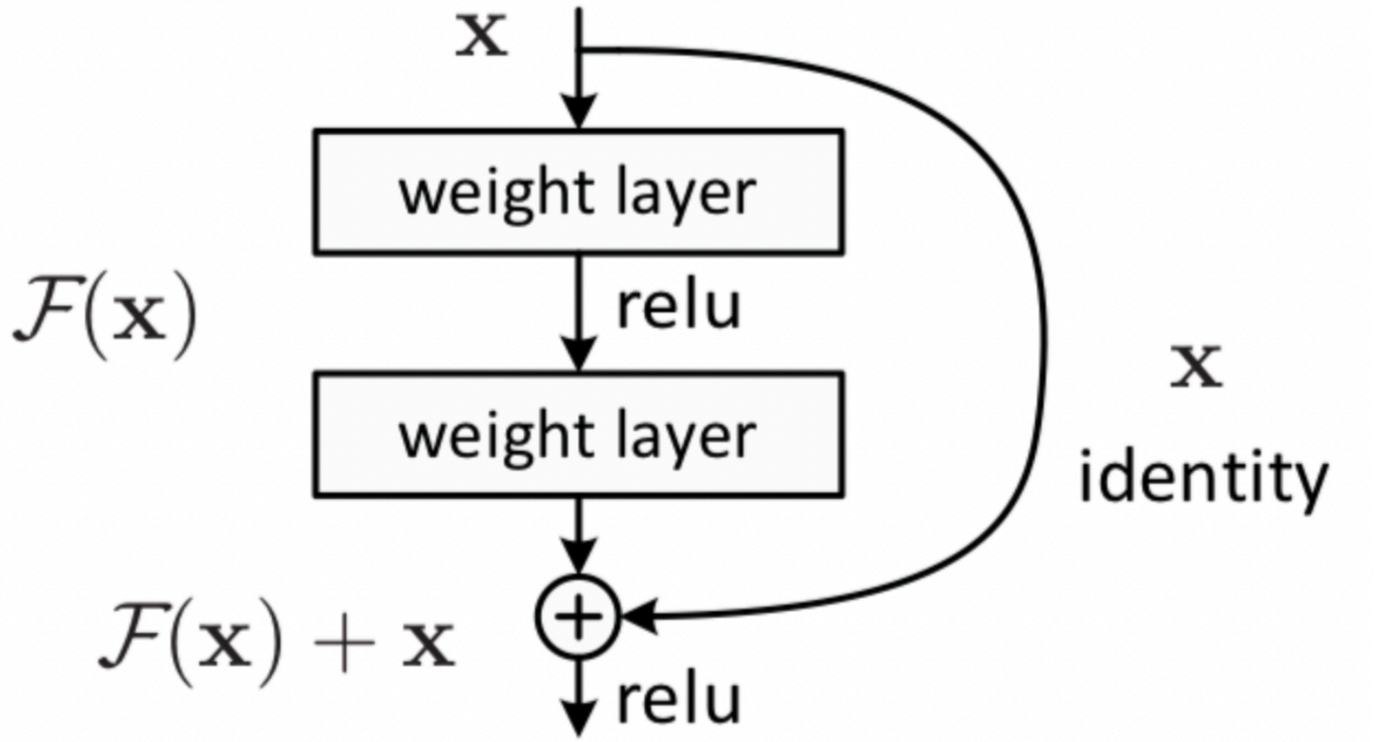
BATCH NORMALIZATION SPEEDS UP AND STABILIZES TRAINING

RESIDUAL NETWORKS



Kaiming et al. 2015

RESIDUAL NETWORKS



- Adding additional / new layers would not hurt the model's performance as regularisation will skip over them if those layers were not useful.

- If the additional / new layers were useful, even with the presence of regularisation, the weights or kernels of the layers will be non-zero and model performance could increase slightly.

Kaiming et al. 2015

Attention (VIT and Transformers)

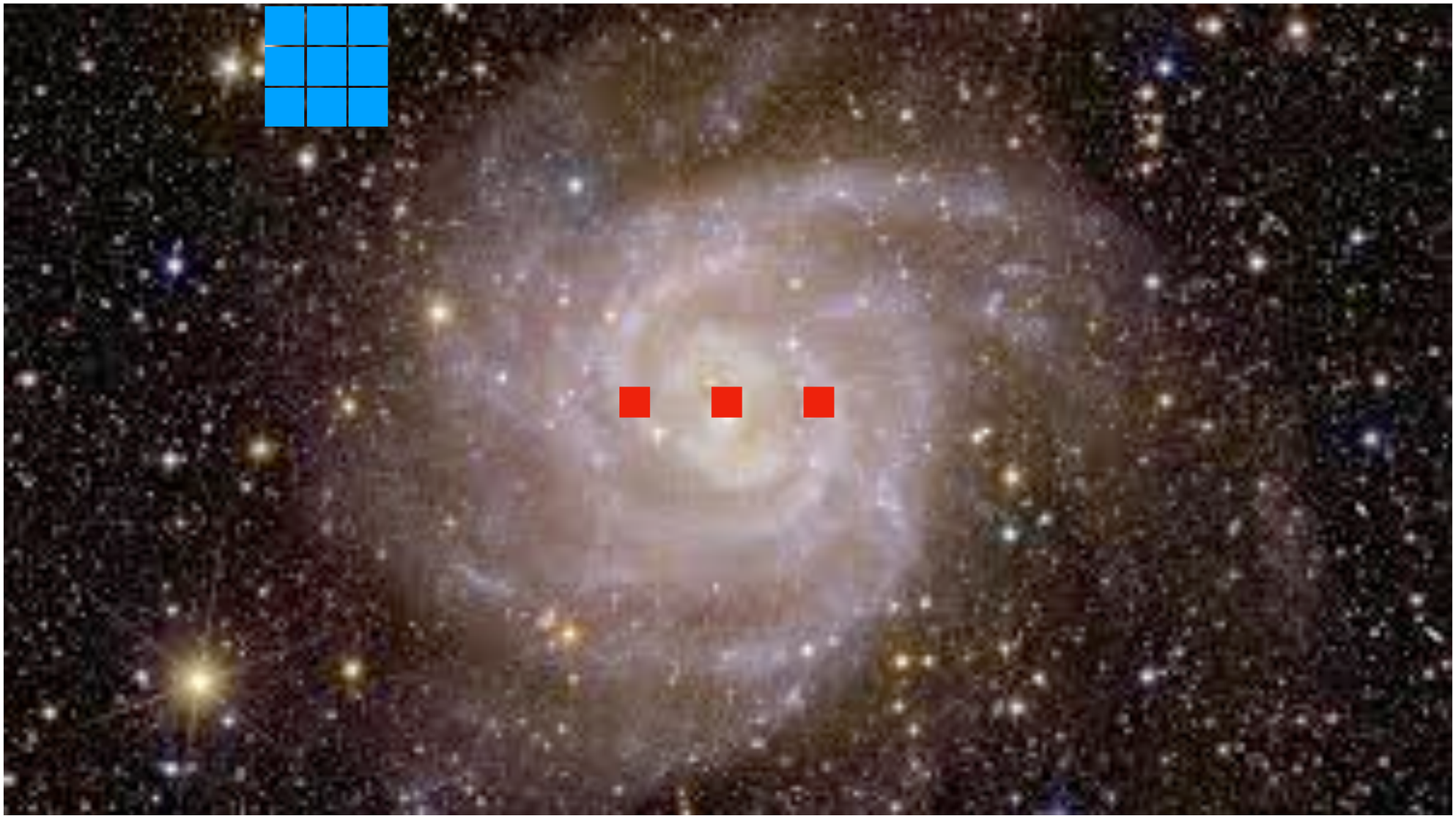
The problem of receptive field

credit: Esa Euclid

The problem of receptive field

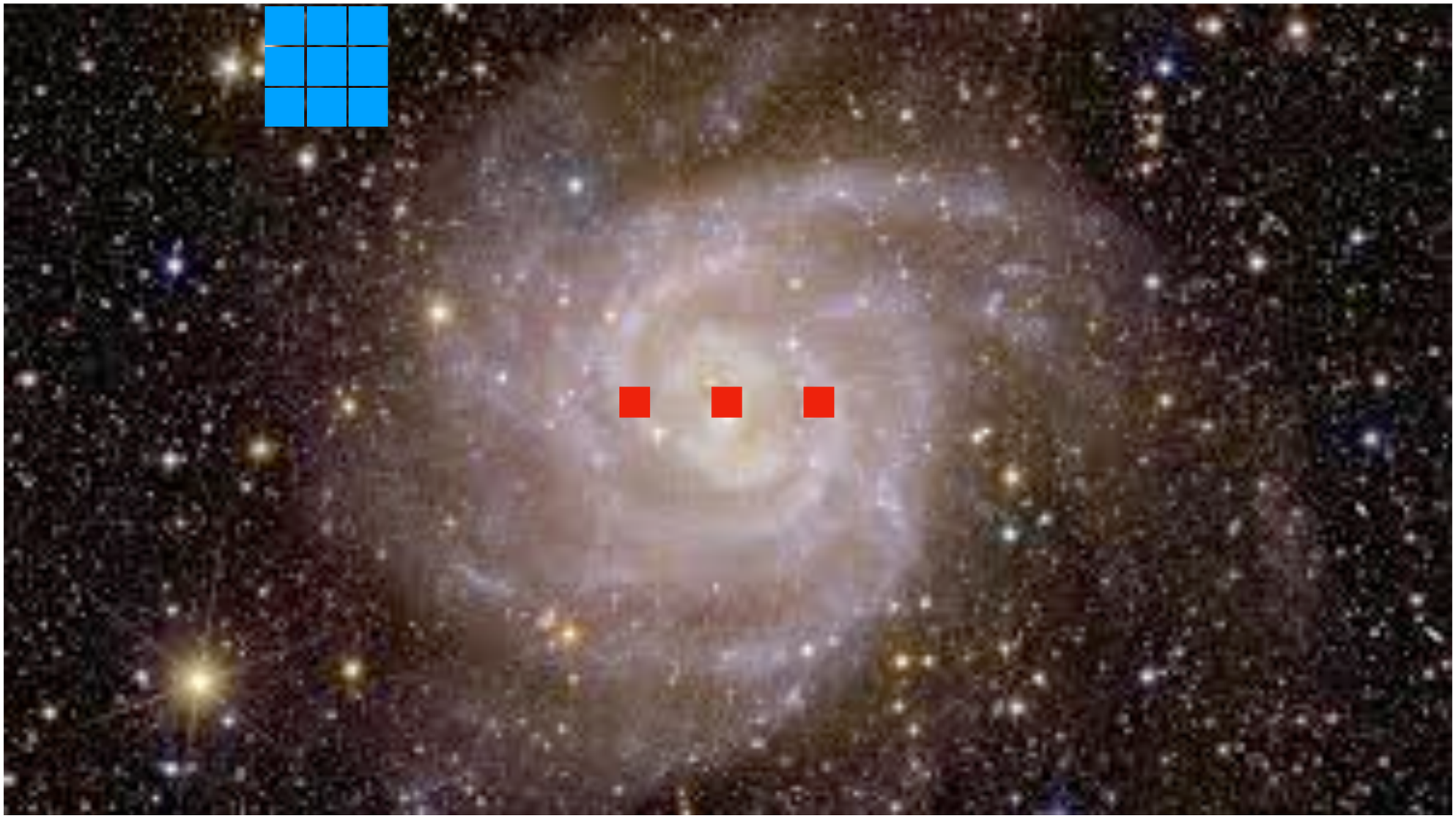
credit: Esa Euclid

The problem of receptive field



credit: Esa Euclid

The problem of receptive field



credit: Esa Euclid

The problem of receptive field

credit: Esa Euclid

The problem of receptive field

We usually reduce this problem with compression:
By reducing the “physical” size of the image, and keeping the convolution kernel the same size, we sort of increase the perception field.

But can we do better?

credit: Esa Euclid

Yes, with Transformers !

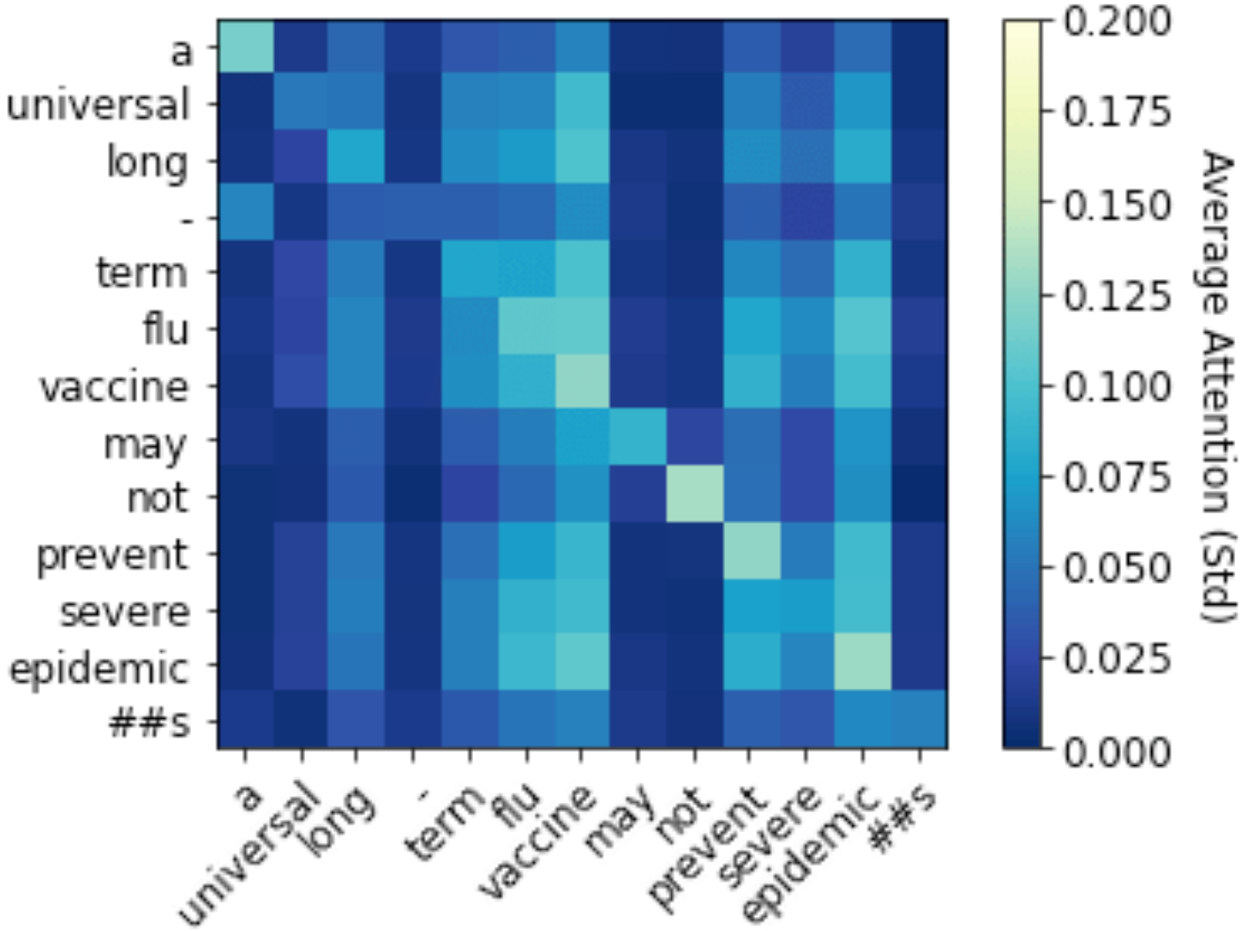
ATTENTION IS ALL YOU NEED

TRANSFORMER MODEL

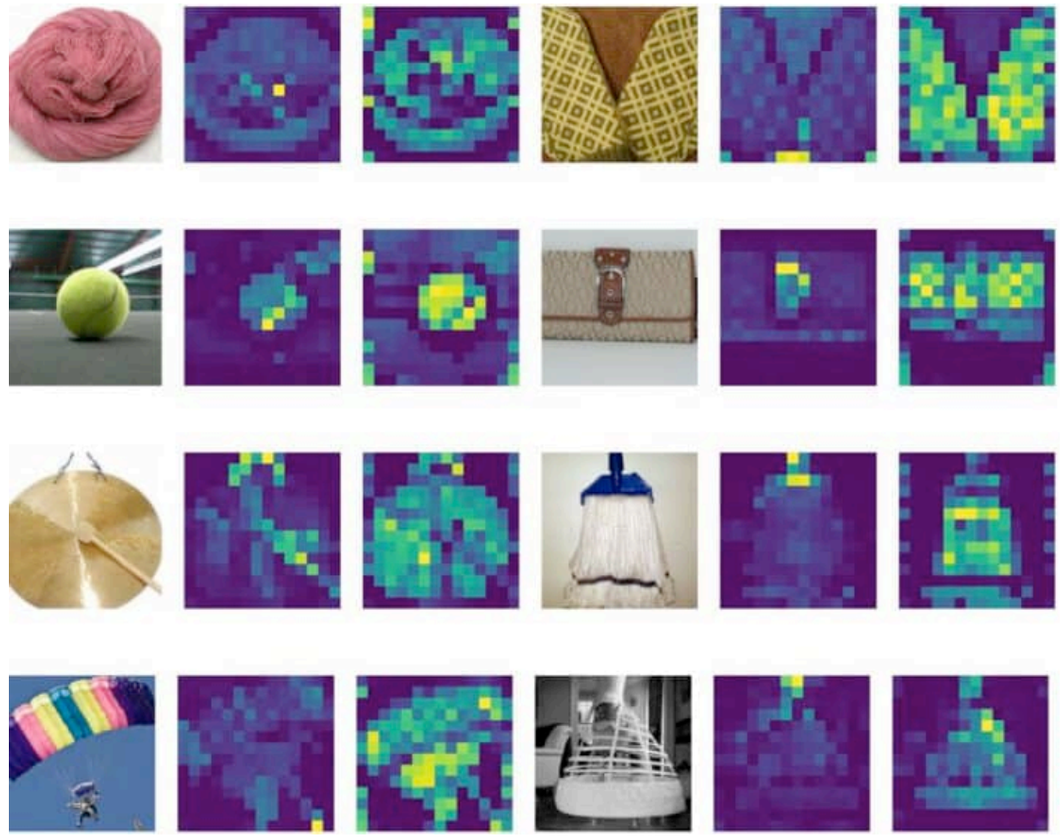
credit: robertocrespo : https://www.robertocrespo.net/transformer-attention-is-all-you-need/

Idea: learn from the context, from the “global picture”

First developped for language processing problems (translation, chatbot…)



Garcia-Silva +21 Classifying Scientific Publications with BERT - Is Self-attention a Feature Selection Method?



When Vision Transformers Outperform ResNets without Pre-training or Strong Data Augmentations

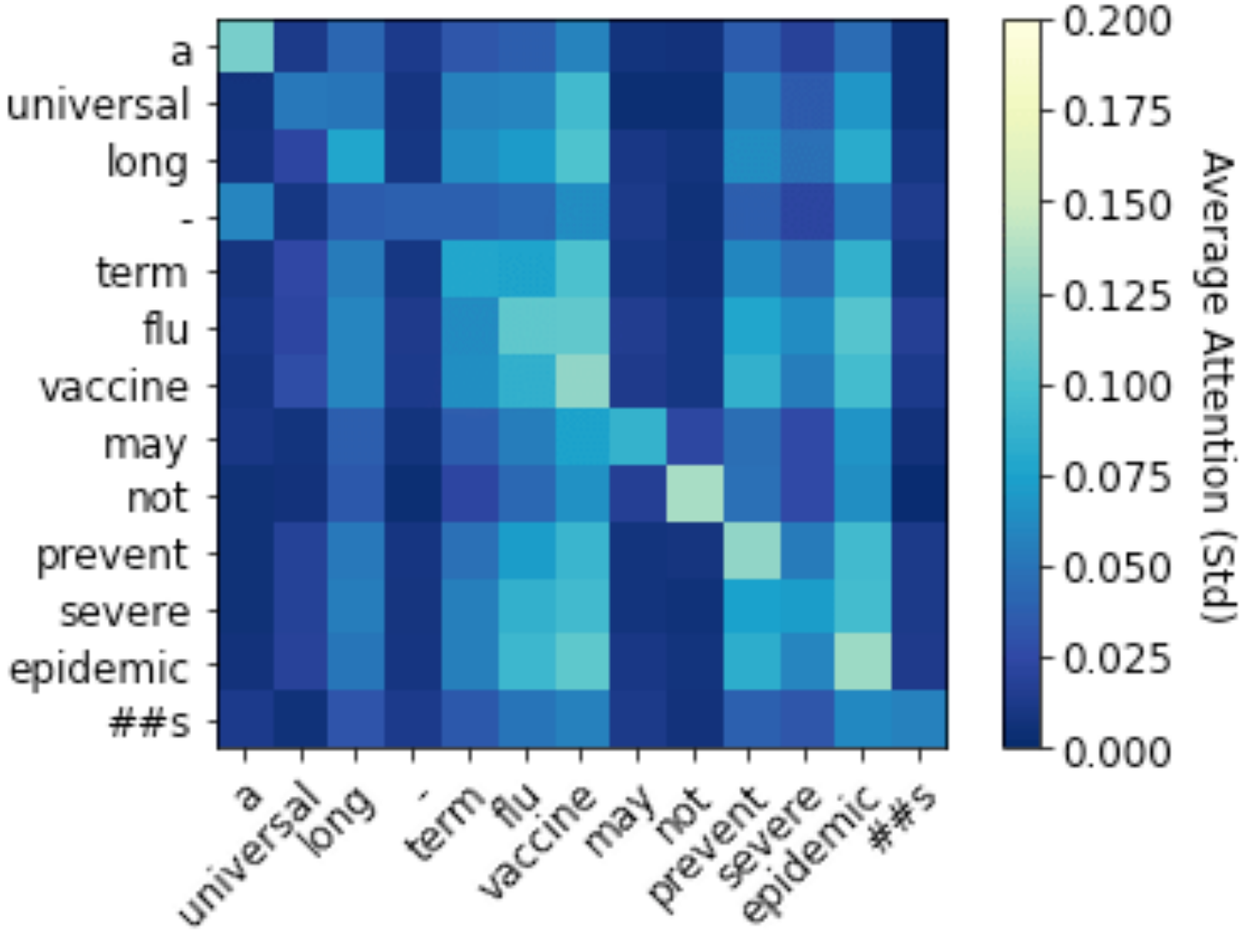
Xiangning Chen + 21

Usefull tutorials / blog / papers

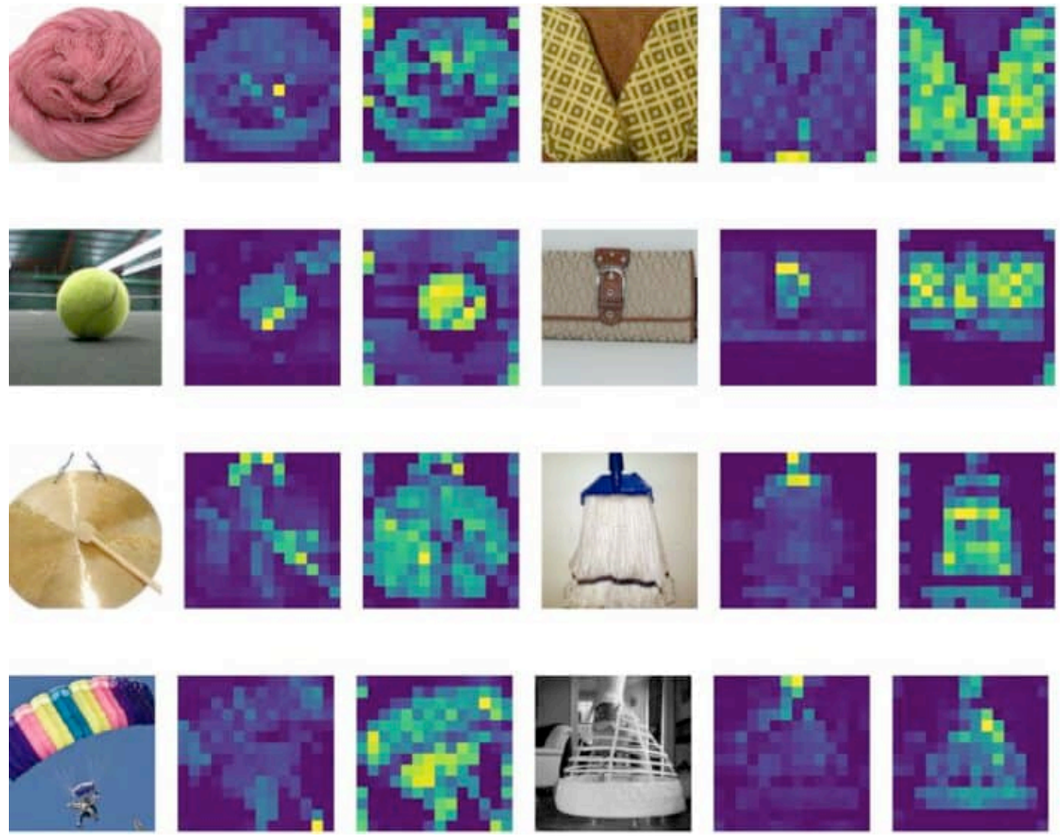
https://www.datacamp.com/tutorial/how-transformers-work

https://nlp.seas.harvard.edu/2018/04/03/attention.html

https://medium.com/data-science/vision-transformers-explained-a9d07147e4c8



Garcia-Silva +21 Classifying Scientific Publications with BERT - Is Self-attention a Feature Selection Method?



When Vision Transformers Outperform ResNets without Pre-training or Strong Data Augmentations

Xiangning Chen + 21


How are you?
INPUT
TRANSFORMER
¿Cómo estás?
OUTPUT

OUTPUT

¿Cómo estás?

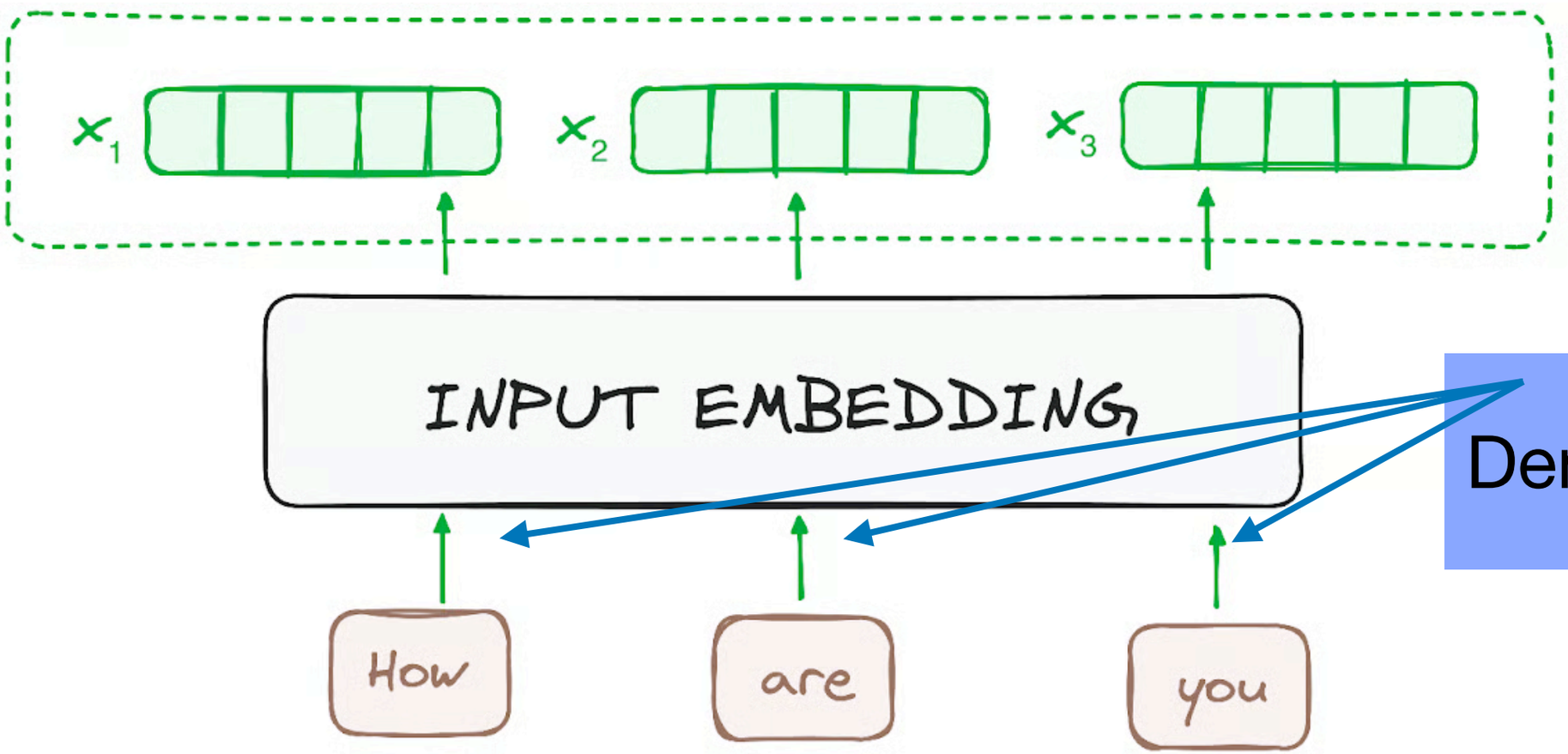
ENCODER

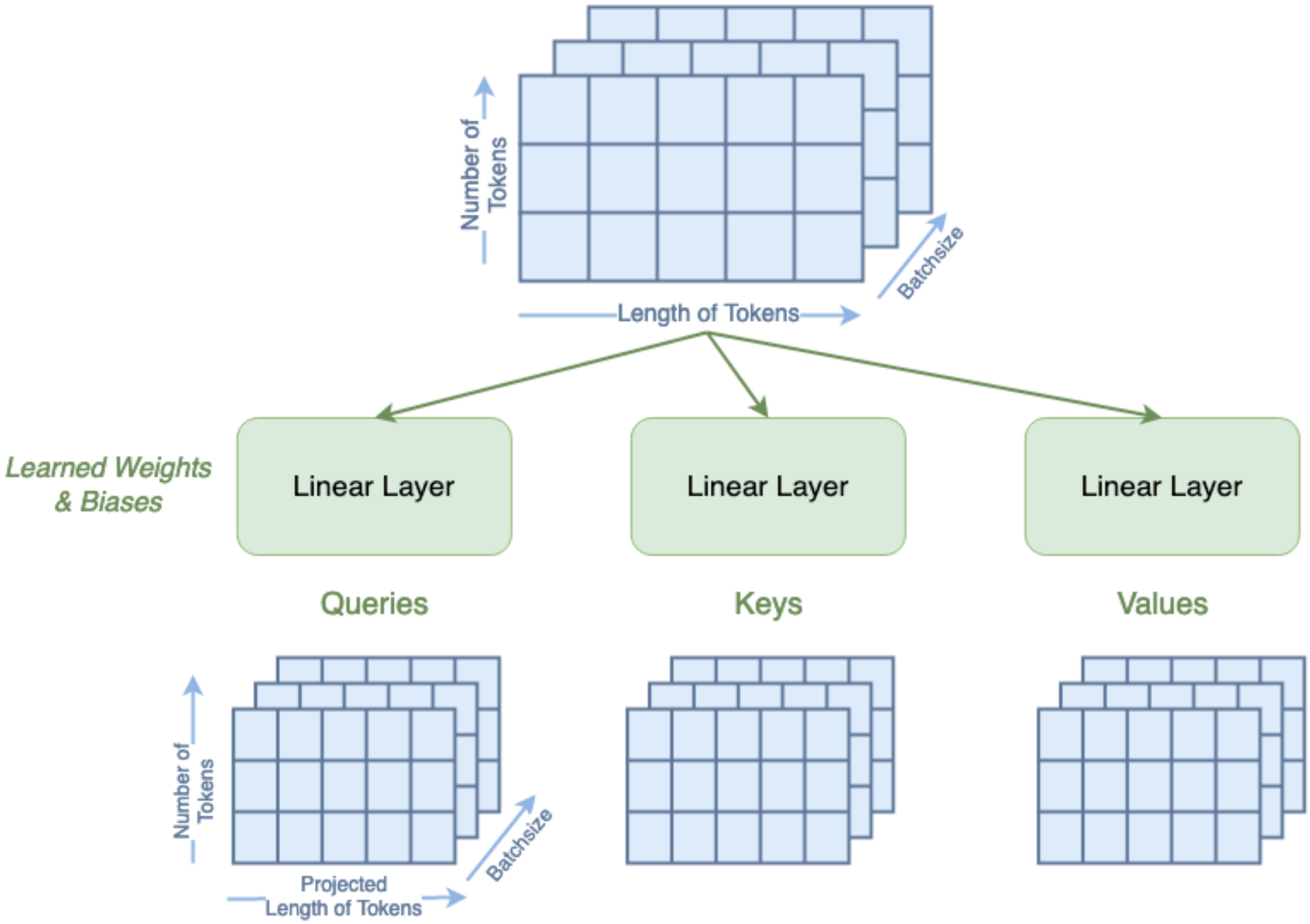
DECODER

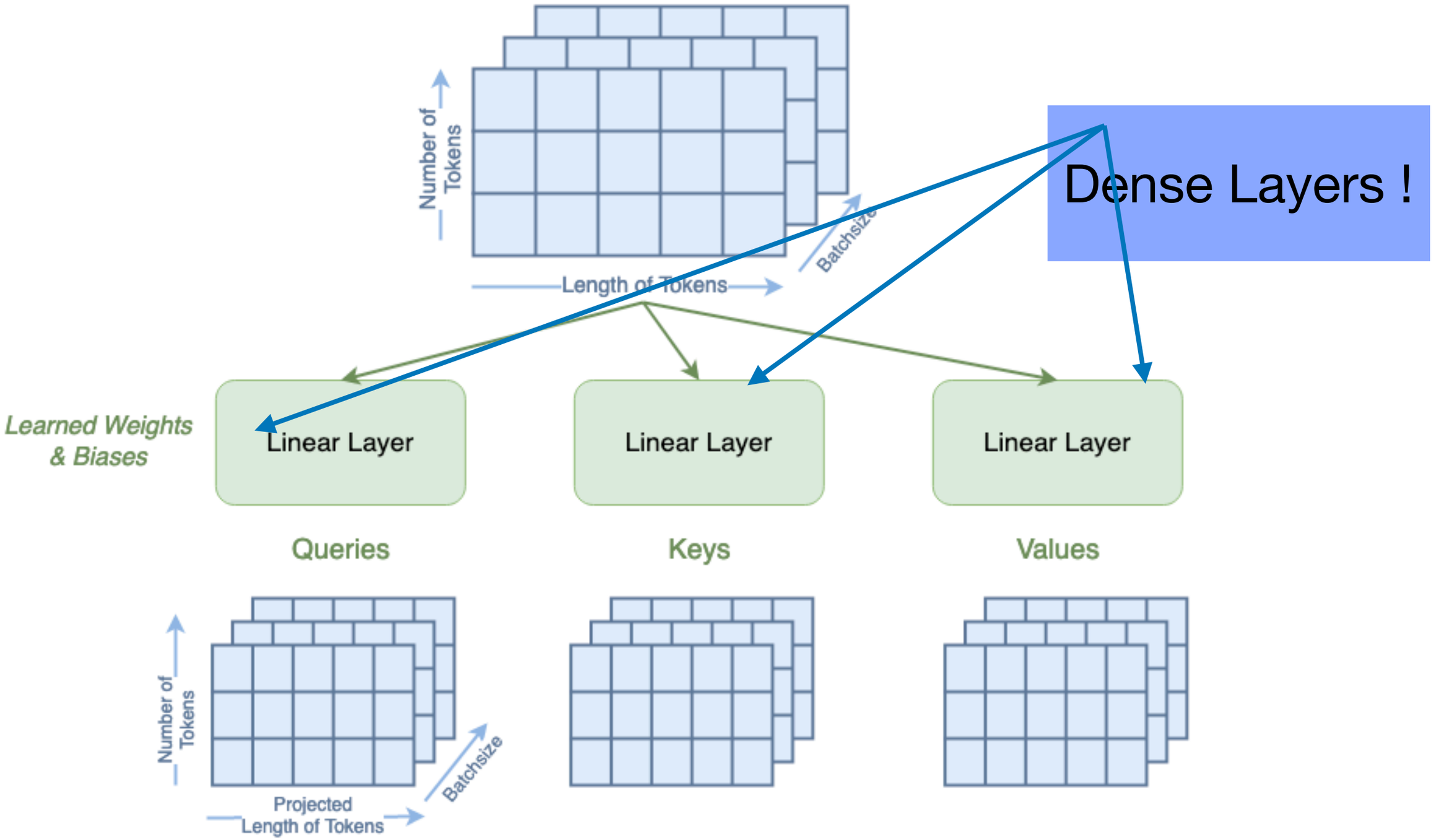
TRANSFORMER

How are you?

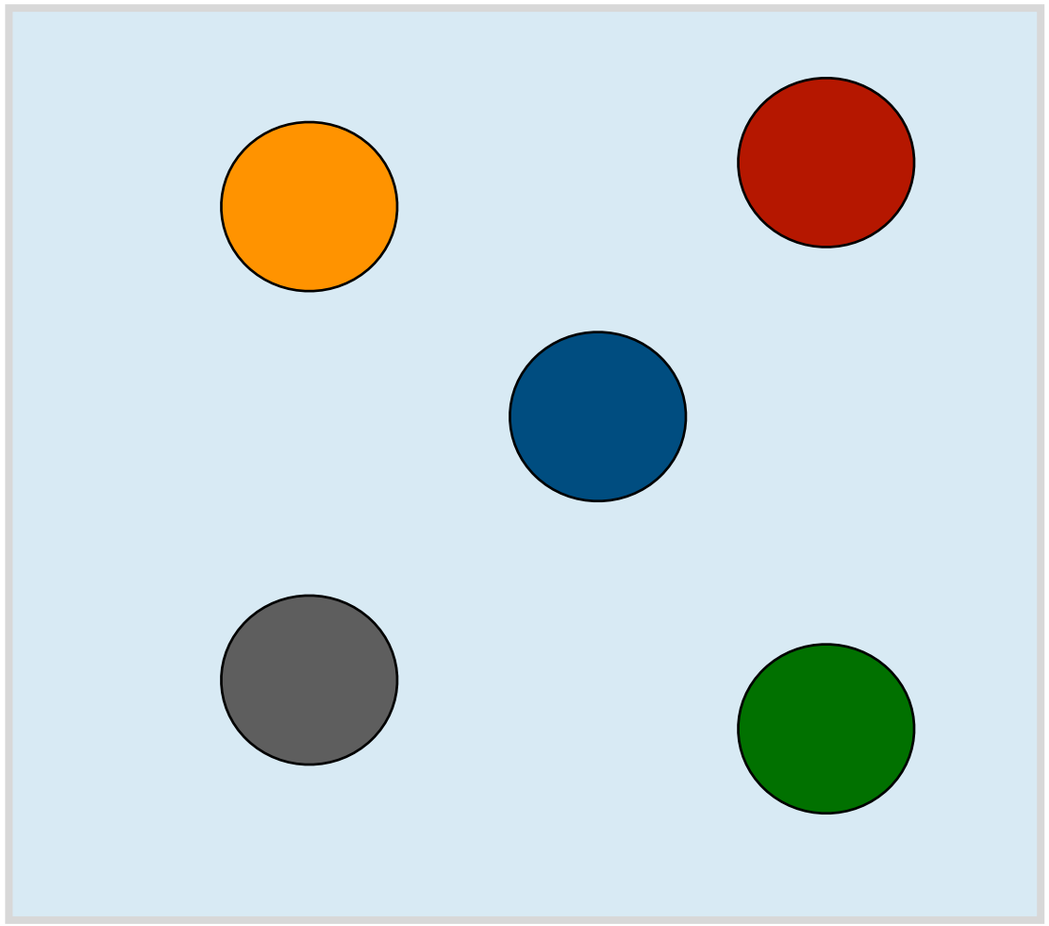
INPUT


token 1
token 2
token 3
x_1
x_2
x_3
Each word is embedded into a vector of size 512
INPUT EMBEDDING
Dense Layers !
How
are
you


Number of Tokens
Length of Tokens
Batchsize
Learned Weights & Biases
Linear Layer
Linear Layer
Linear Layer
Queries
Keys
Values
Number of Tokens
Projected Length of Tokens
Batchsize


Number of Tokens
Length of Tokens
Batchsize
Dense Layers !
Learned Weights & Biases
Linear Layer
Linear Layer
Linear Layer
Queries
Keys
Values
Number of Tokens
Projected Length of Tokens
Batchsize

Why Key, Query and Values?

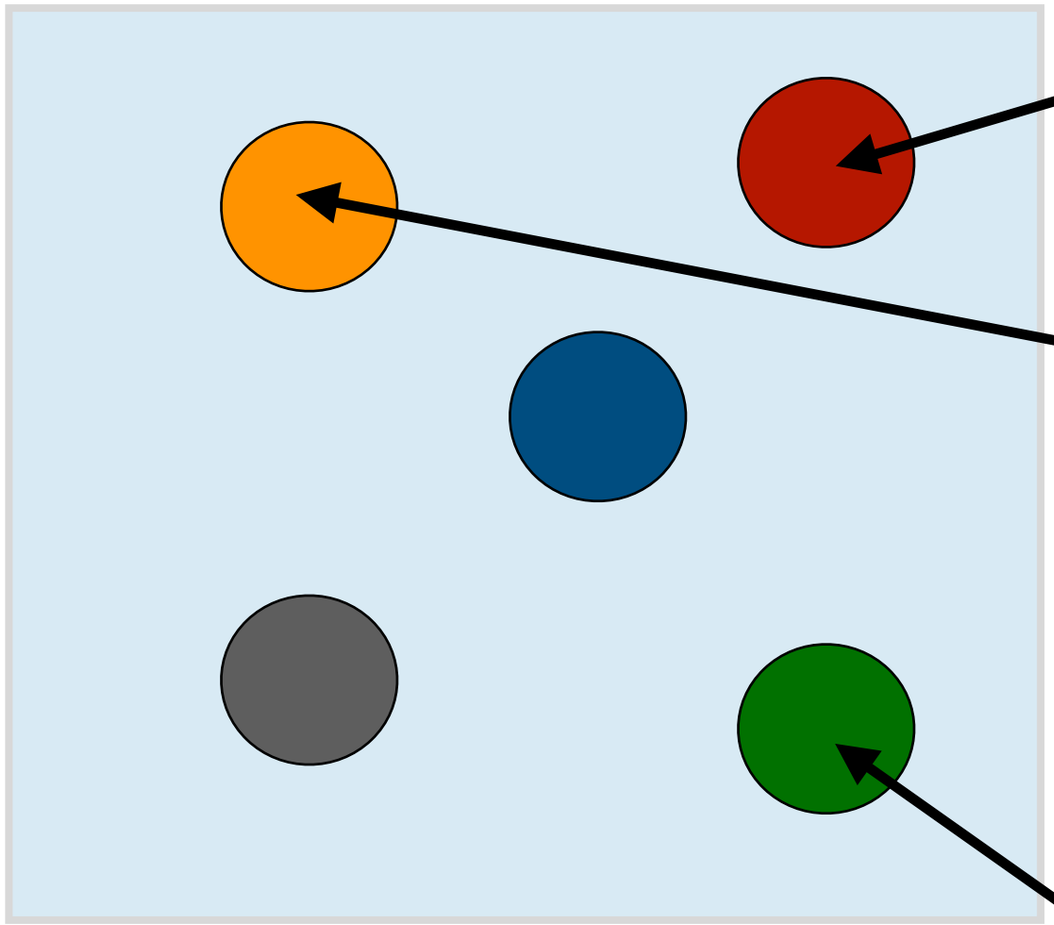


Value: a note with what you know

Query: a question you have

Key: a way to advertise what you know

Why Key, Query and Values?



Token 1

Value_1
Key_1
Query_1

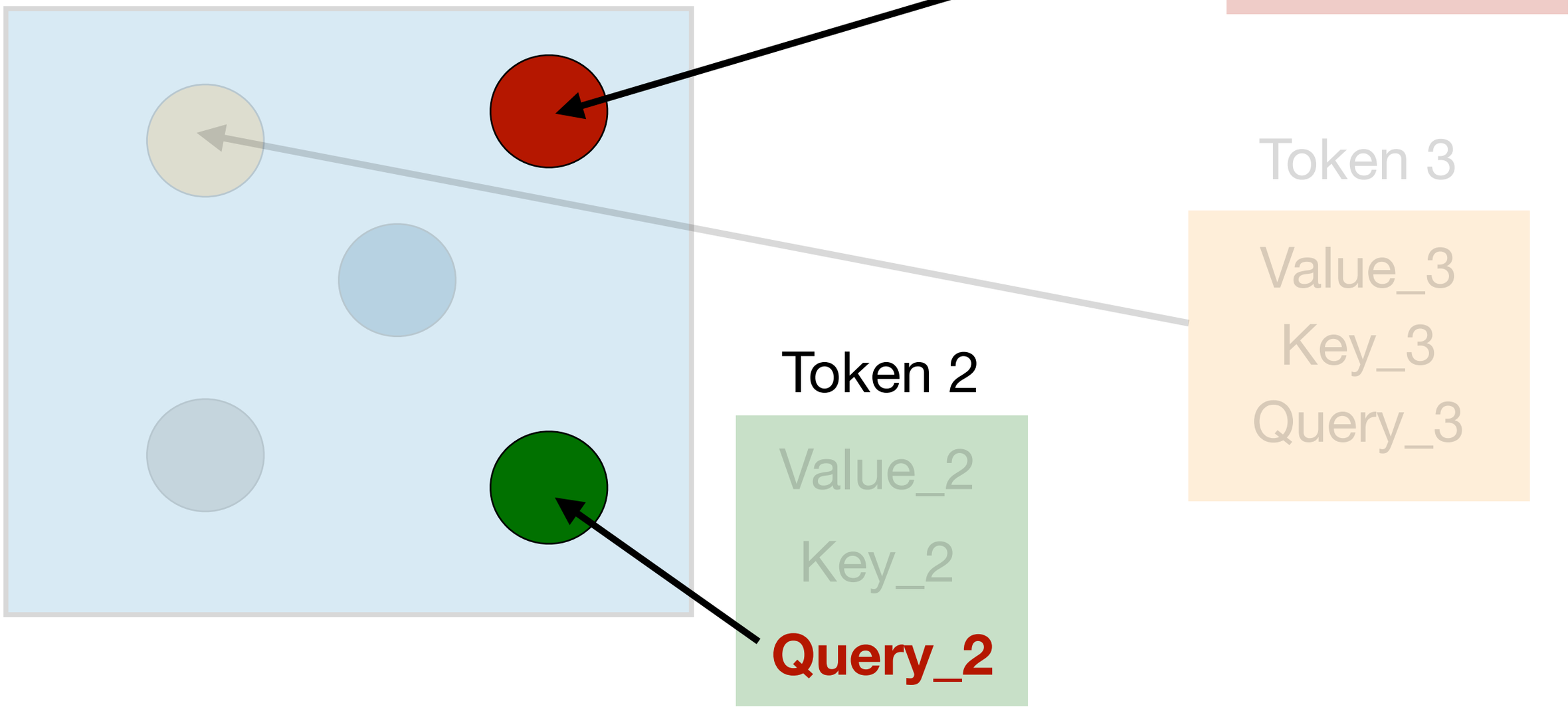
Token 3

Value_3
Key_3
Query_3

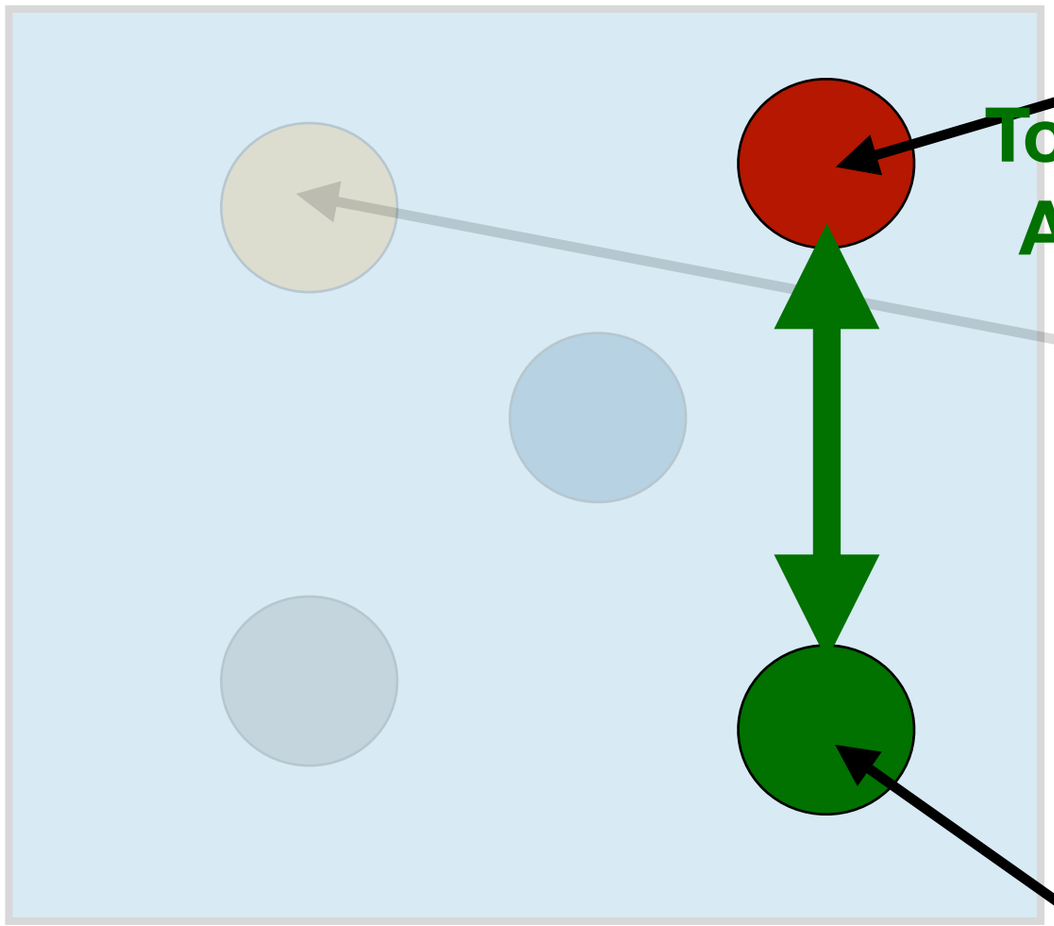
Token 2

Value_2
Key_2
Query_2

Why Key, Query and Values?



Why Key, Query and Values?



Token 1

Value_1

Key_1

Query_1

Token 1 & token 2 Are related, and should share knowledge

Token 3

Value_3

Key_3

Query_3

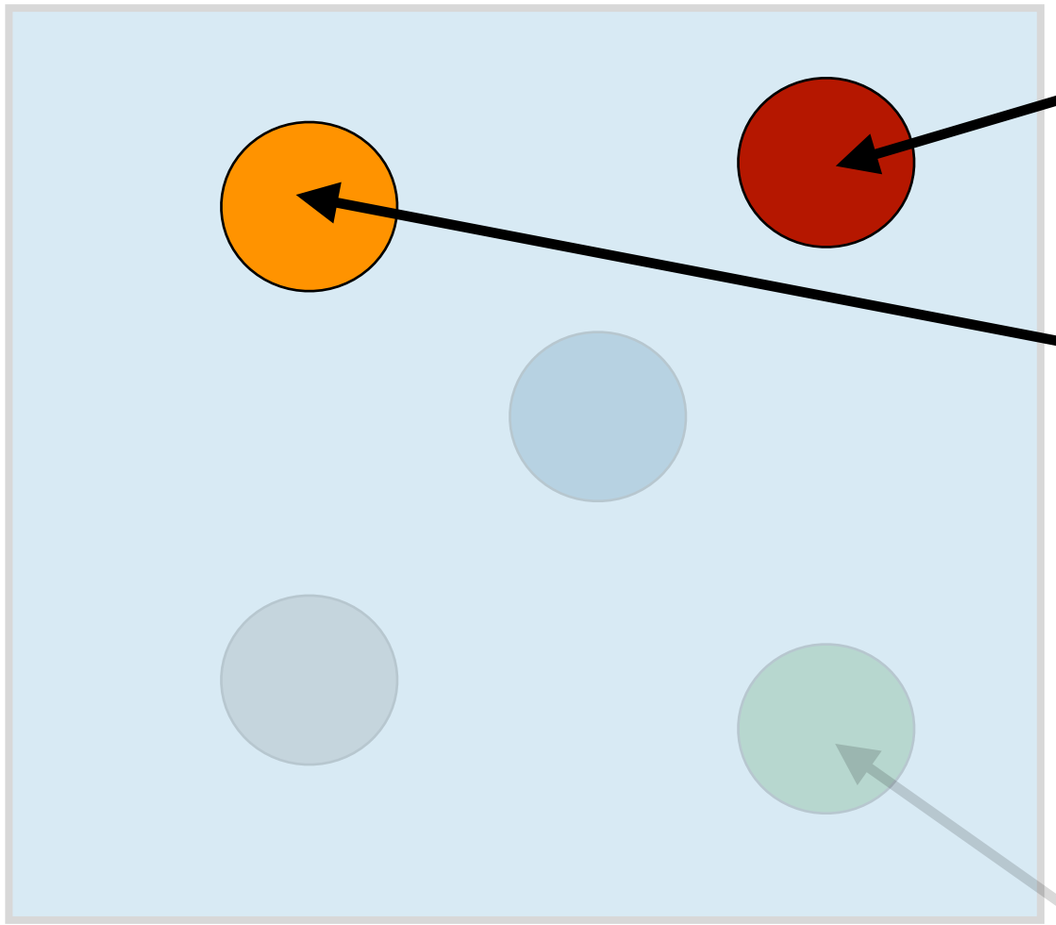
Token 2

Value_2

Key_2

Query_2

Why Key, Query and Values?



Token 1

Value_1

Key_1

Query_1

Token 3

Value_3

Key_3

Query_3

Token 2

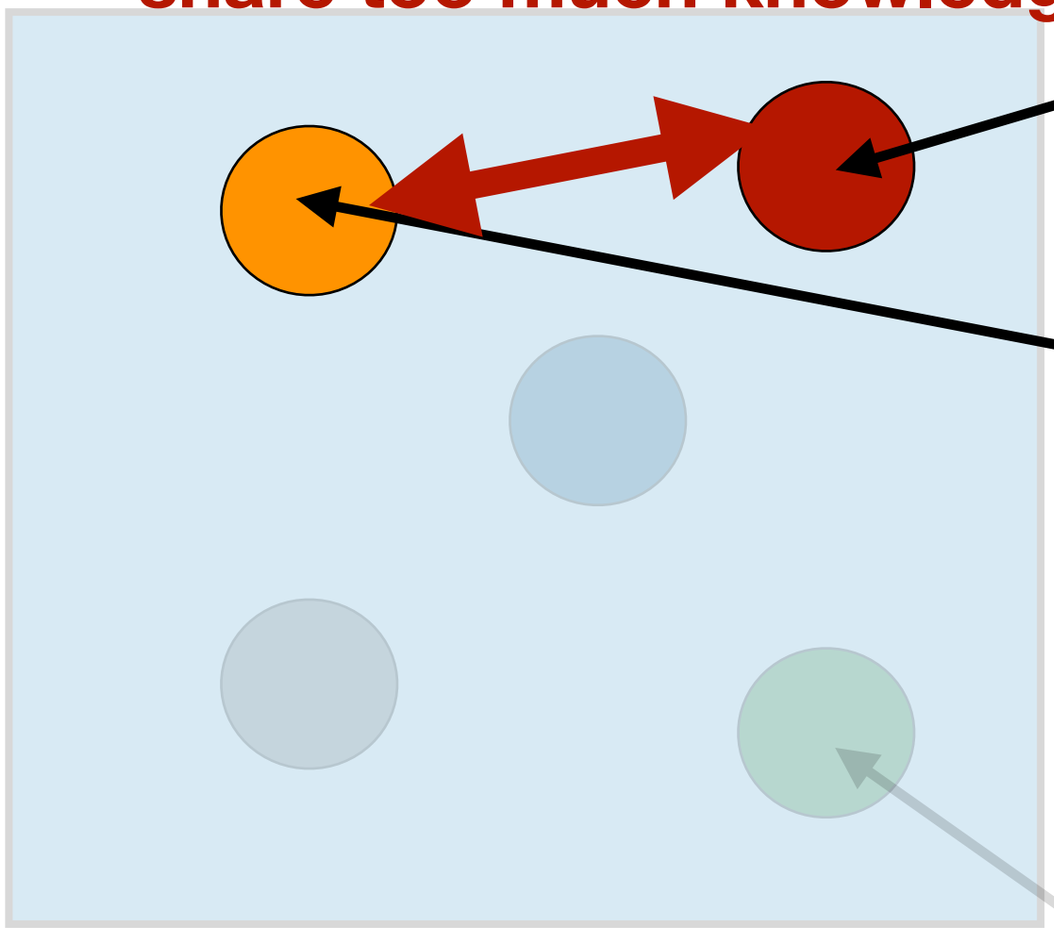
Value_2

Key_2

Query_2

Why Key, Query and Values?

Token 1 & token 3
Are not so related, and should not share too much knowledge



Token 1

Value_1

Key_1

Query_1

Token 3

Value_3

Key_3

Query_3

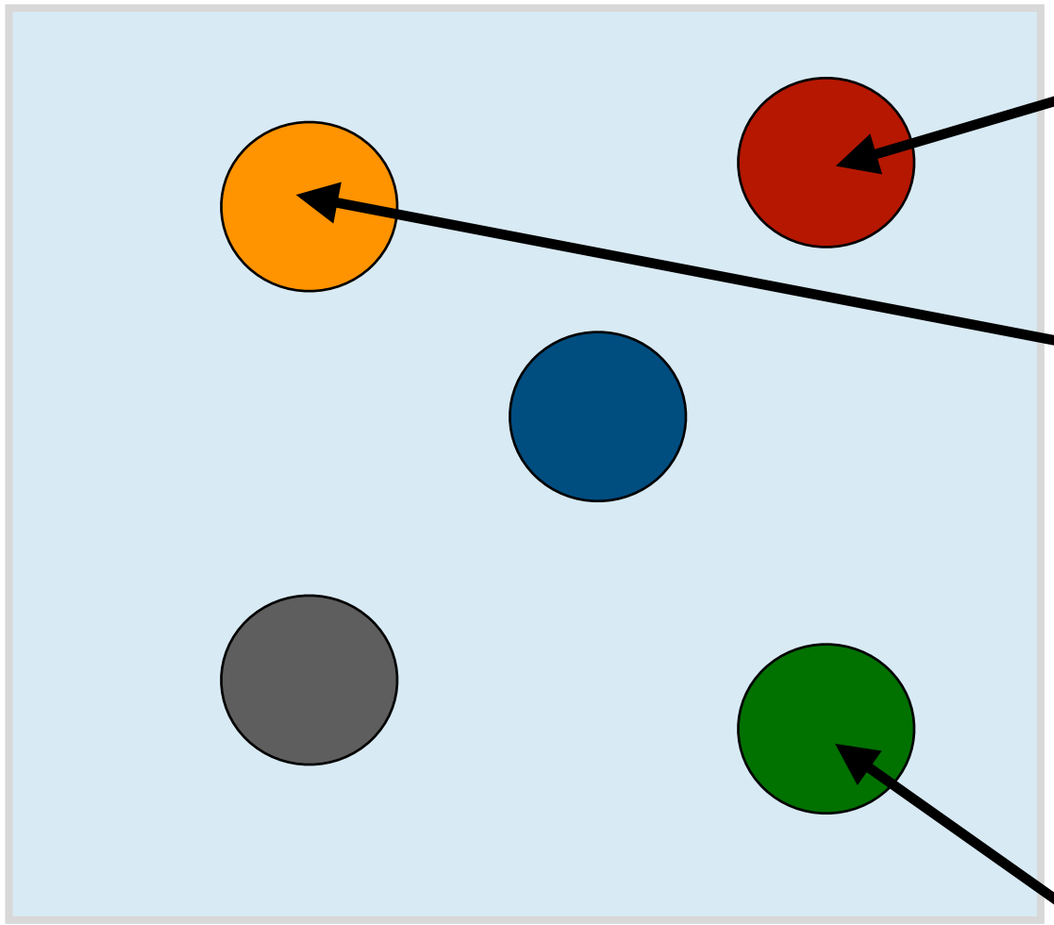
Token 2

Value_2

Key_2

Query_2

Why Key, Query and Values?



Token 1

Value_1
Key_1
Query_1

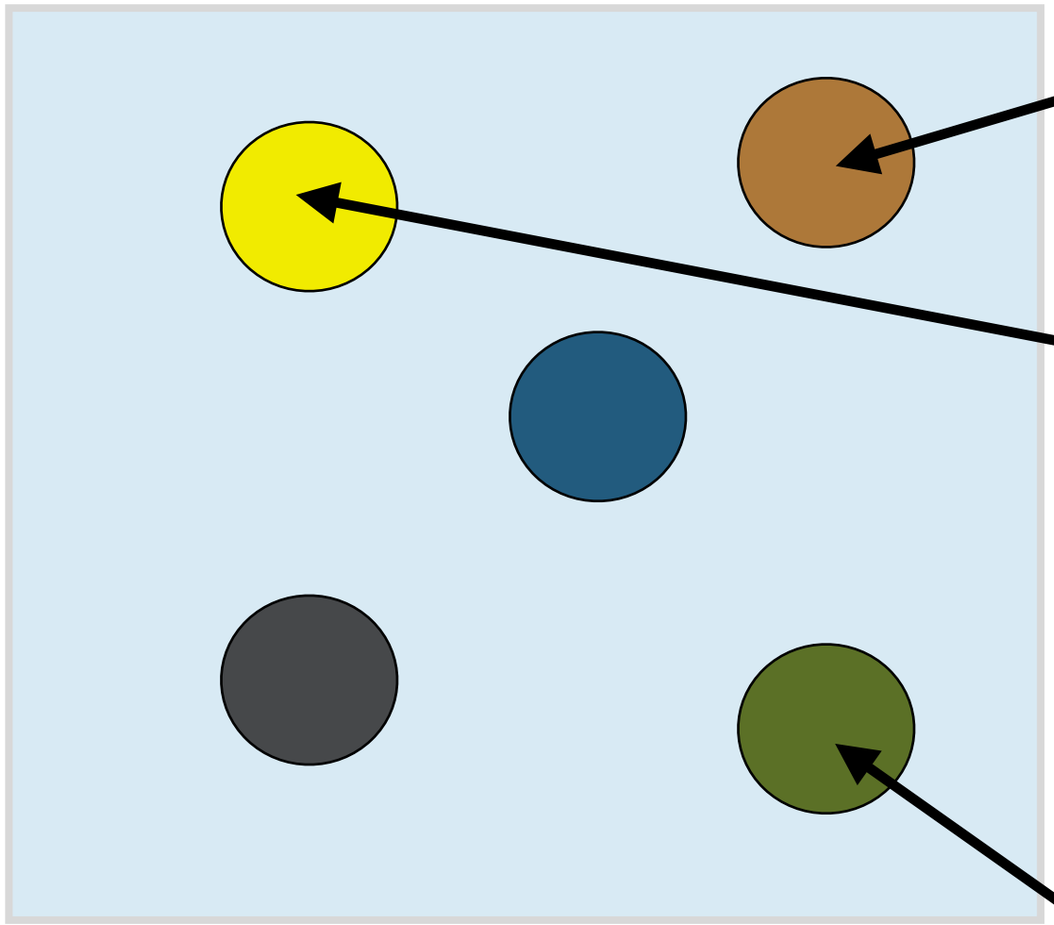
Token 3

Value_3
Key_3
Query_3

Token 2

Value_2
Key_2
Query_2

Why Key, Query and Values?



Token 1

Value_1-2

Key_1

Query_1

Token 3

Value_3-2

Key_1

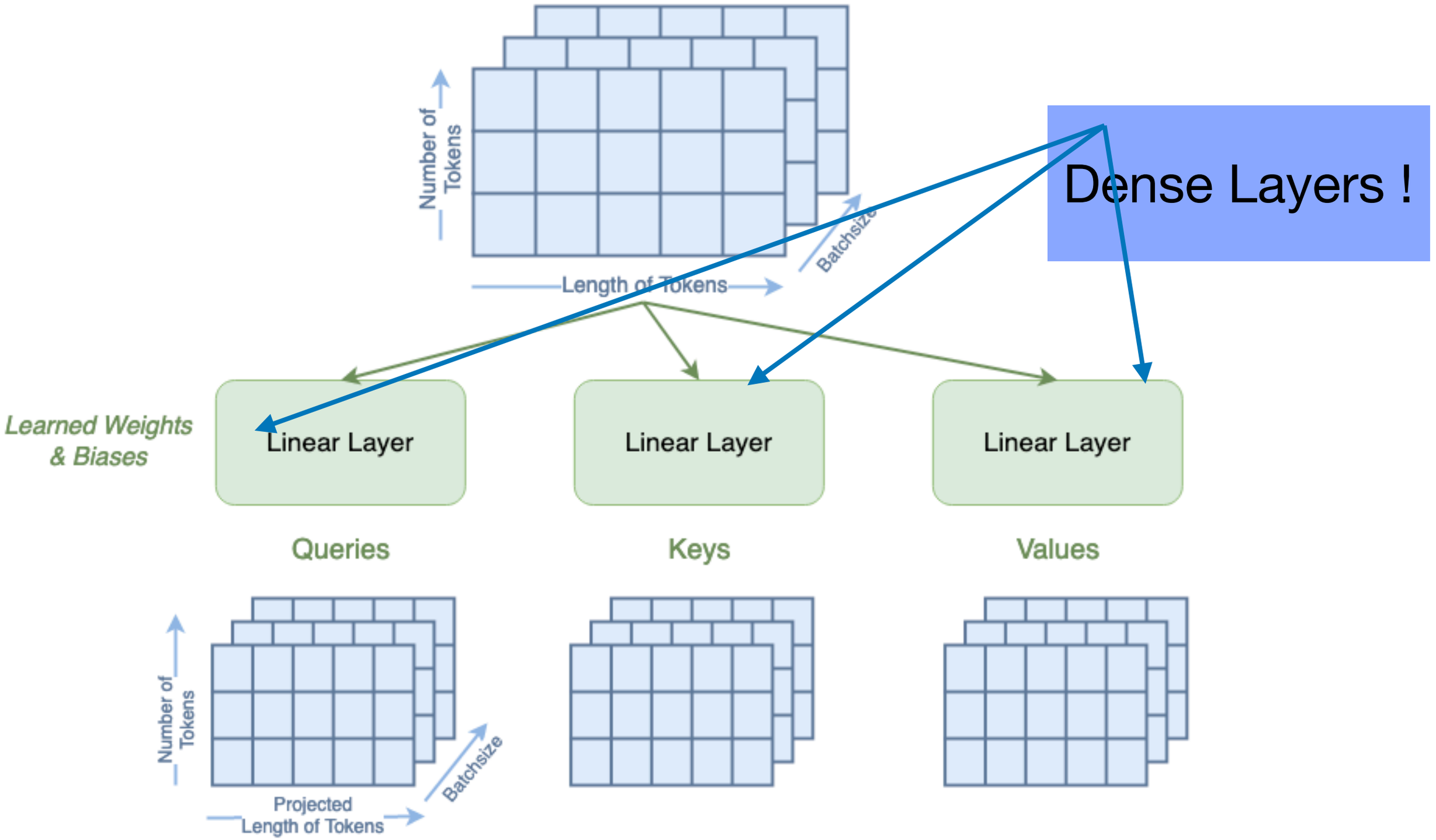
Query_1

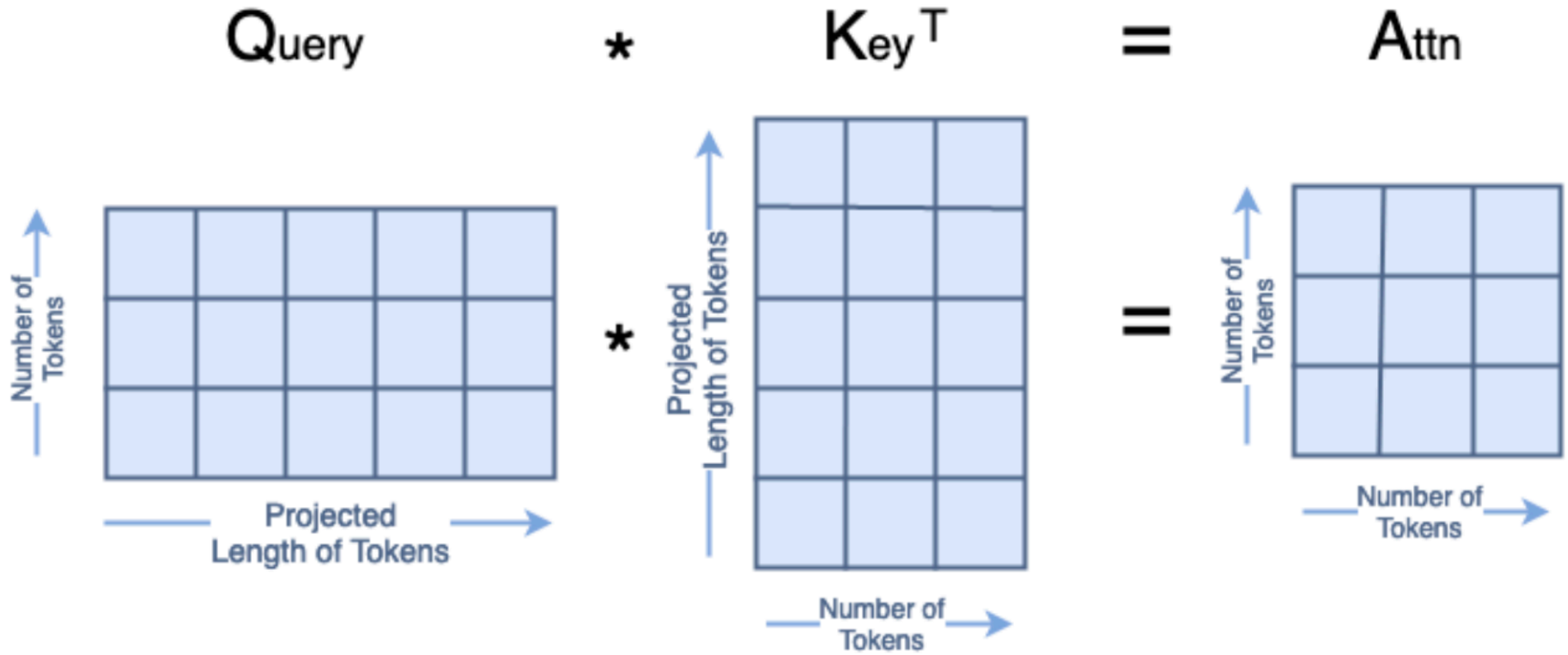
Token 2

Value_2-2

Key_2

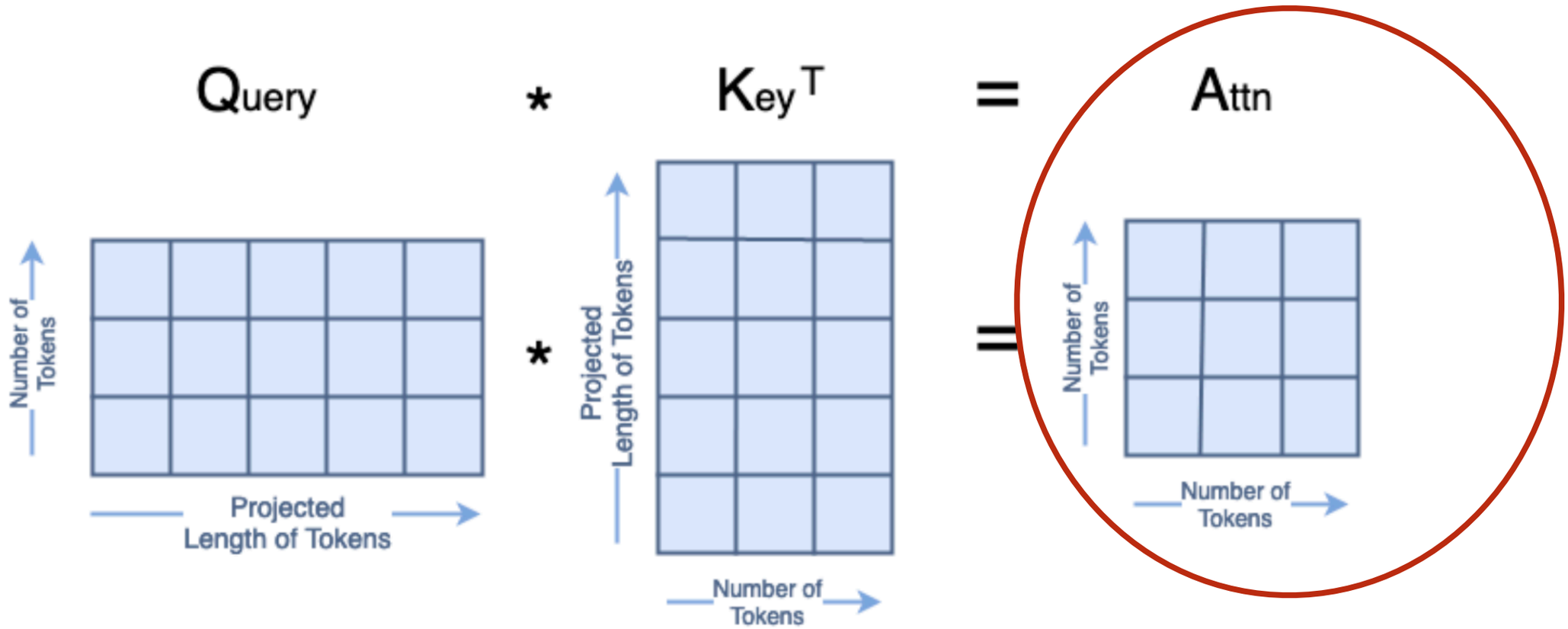
Query_2


Number of Tokens
Length of Tokens
Batchsize
Dense Layers !
Learned Weights & Biases
Linear Layer
Linear Layer
Linear Layer
Queries
Keys
Values
Number of Tokens
Projected Length of Tokens
Batchsize

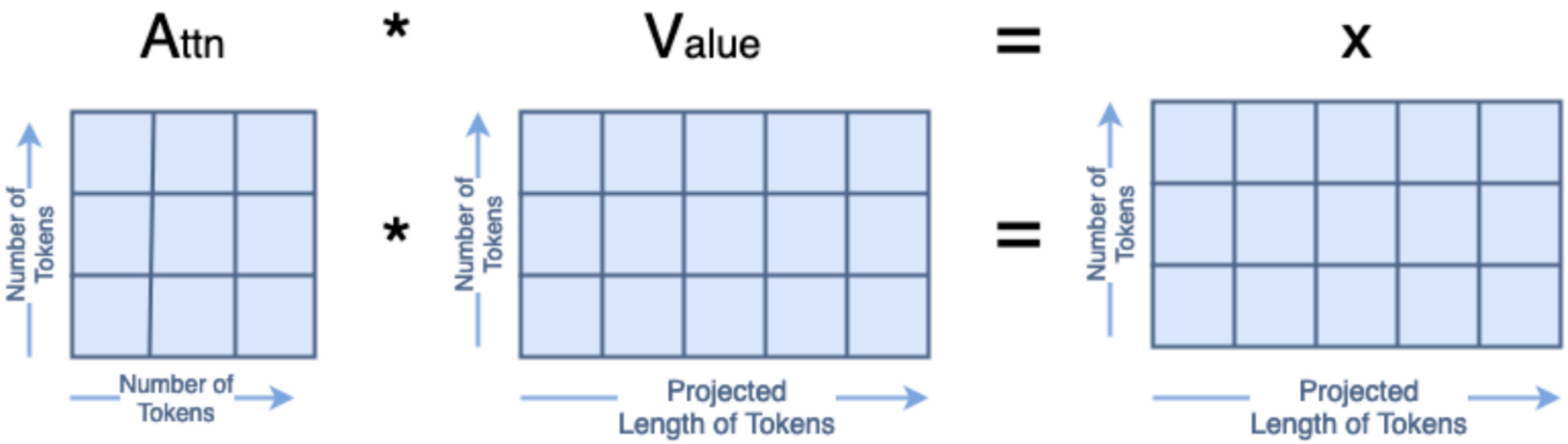


Q·K^T Matrix Multiplication (image by author)

Super important and full of interesting information!!

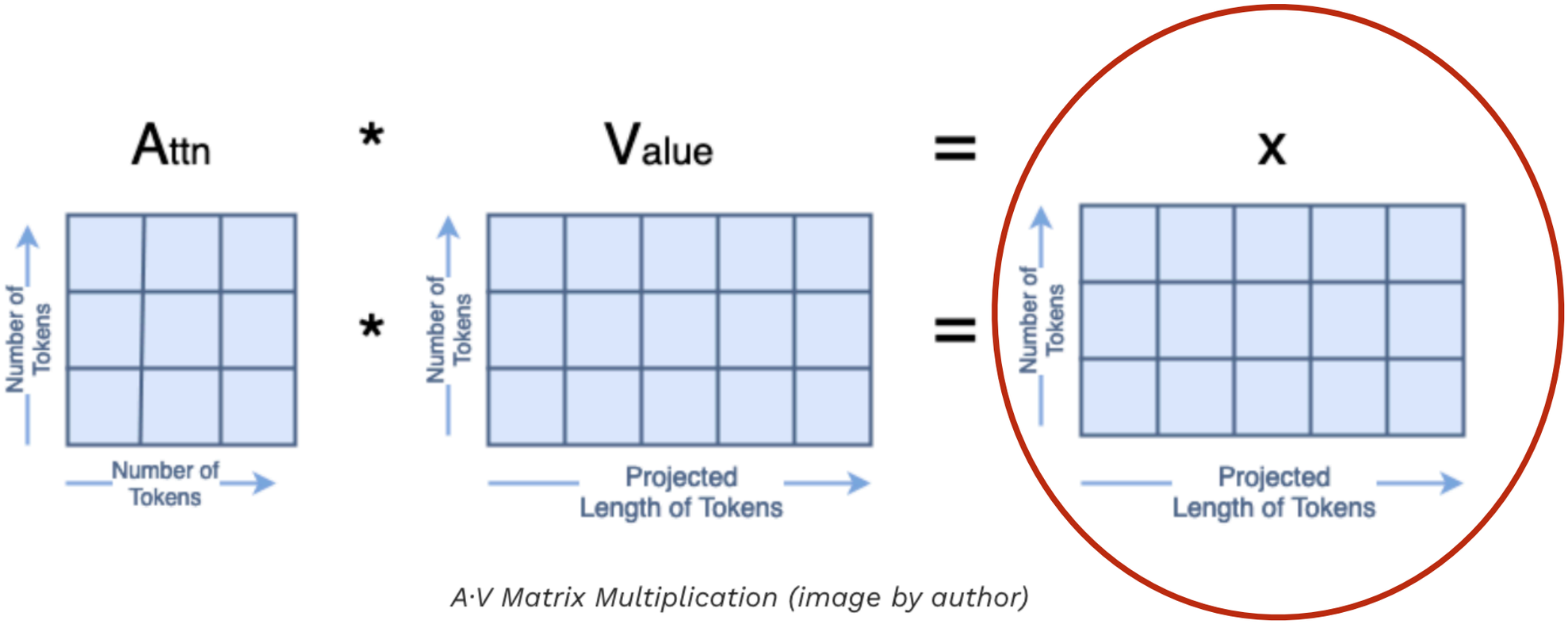


Q·K^T Matrix Multiplication (image by author)

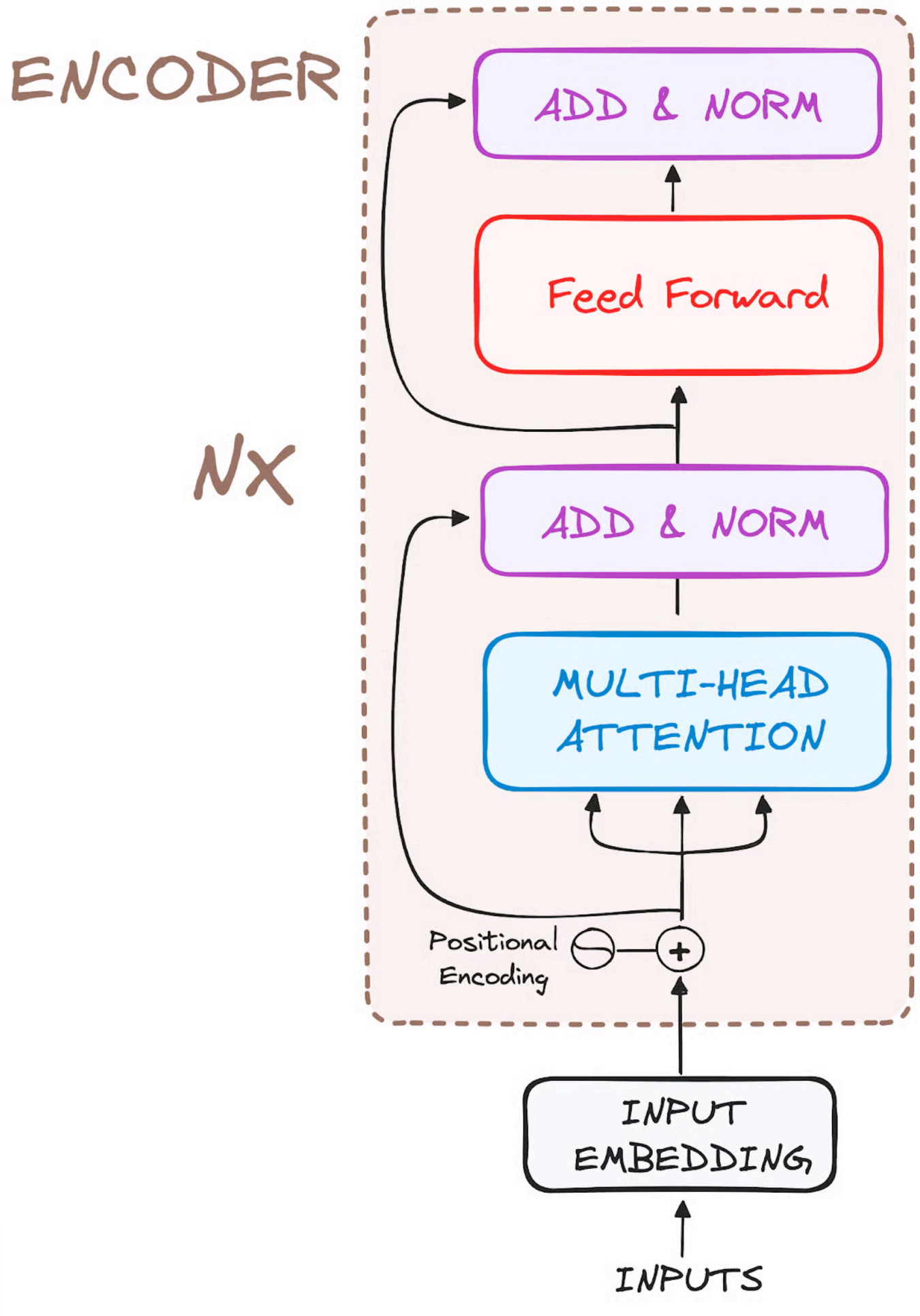


A·V Matrix Multiplication (image by author)

Our new representation, augmented by the global context



A·V Matrix Multiplication (image by author)


ENCODER
Nx
ADD & NORM
Feed Forward
ADD & NORM
MULTI-HEAD ATTENTION
Positional Encoding
INPUT EMBEDDING
INPUTS

For a classifier task, we can stop here: we have our new super informative representation, and we can plug any model we want here to perform the task. (see foundation models later)

Published as a conference paper at ICLR 2021

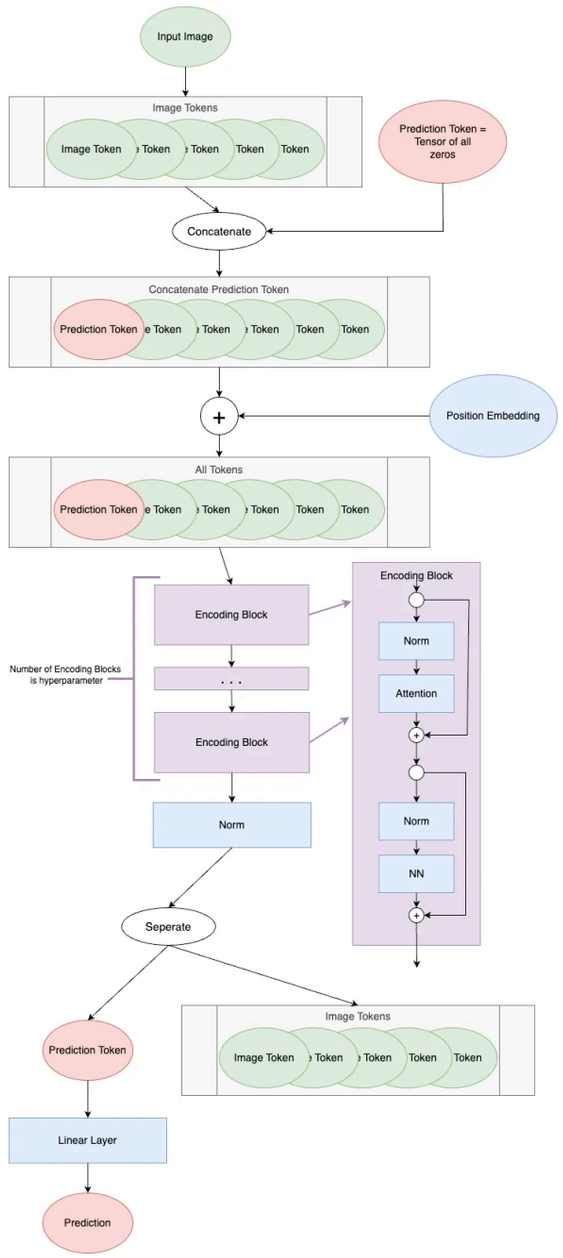
AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*], **Xiaohua Zhai**[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby**[*,†]

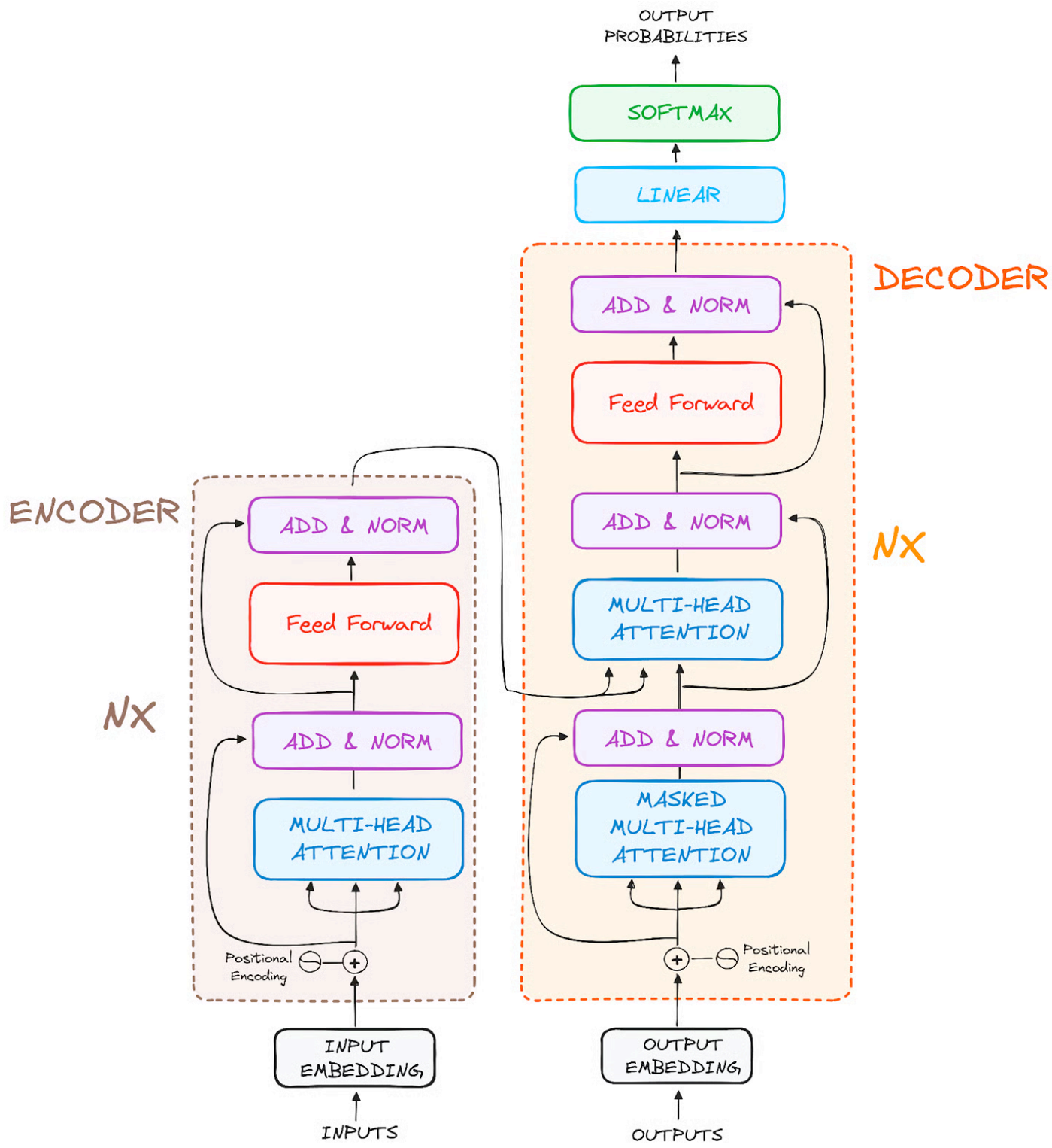
[*]equal technical contribution, [†]equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

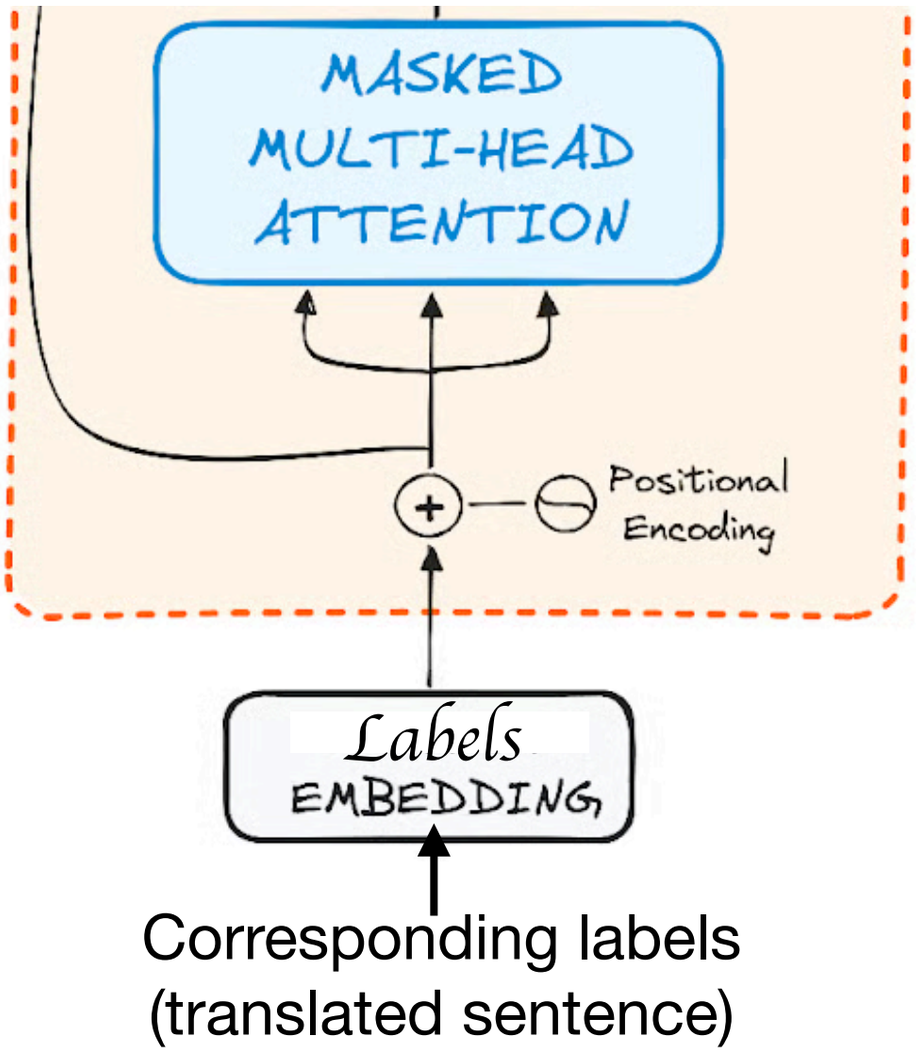


For a sequence to sequence task, we need to “decode”



We use a mixing of self & context / input attention to **sequentially** generate new tokens, hiding some parts to not “see the future”

For a sequence to sequence task, we need to “decode”

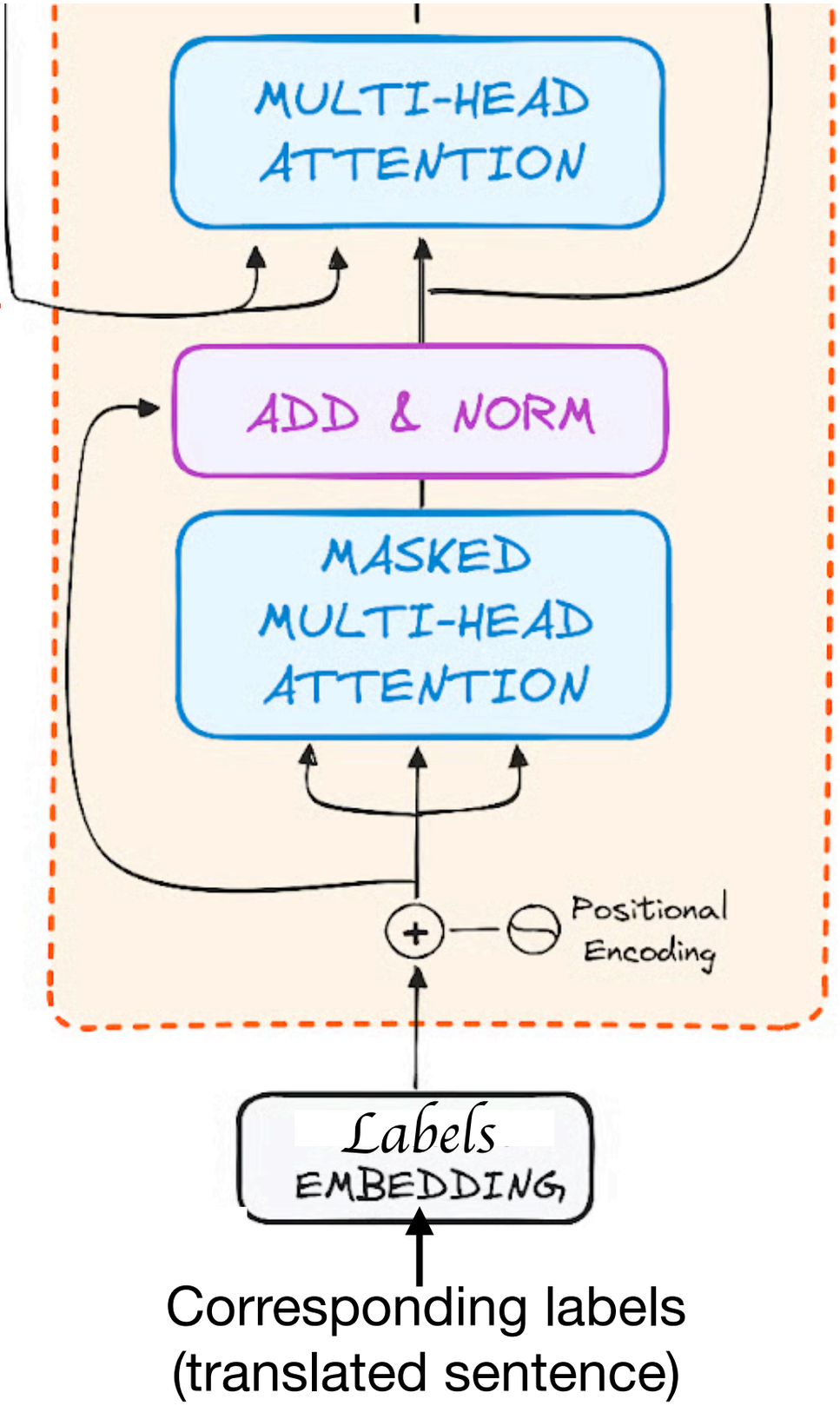


Masked Self attention

Corresponding labels
(translated sentence)

For a sequence to sequence task, we need to “decode”

Context embeddings from encoder (not masked)

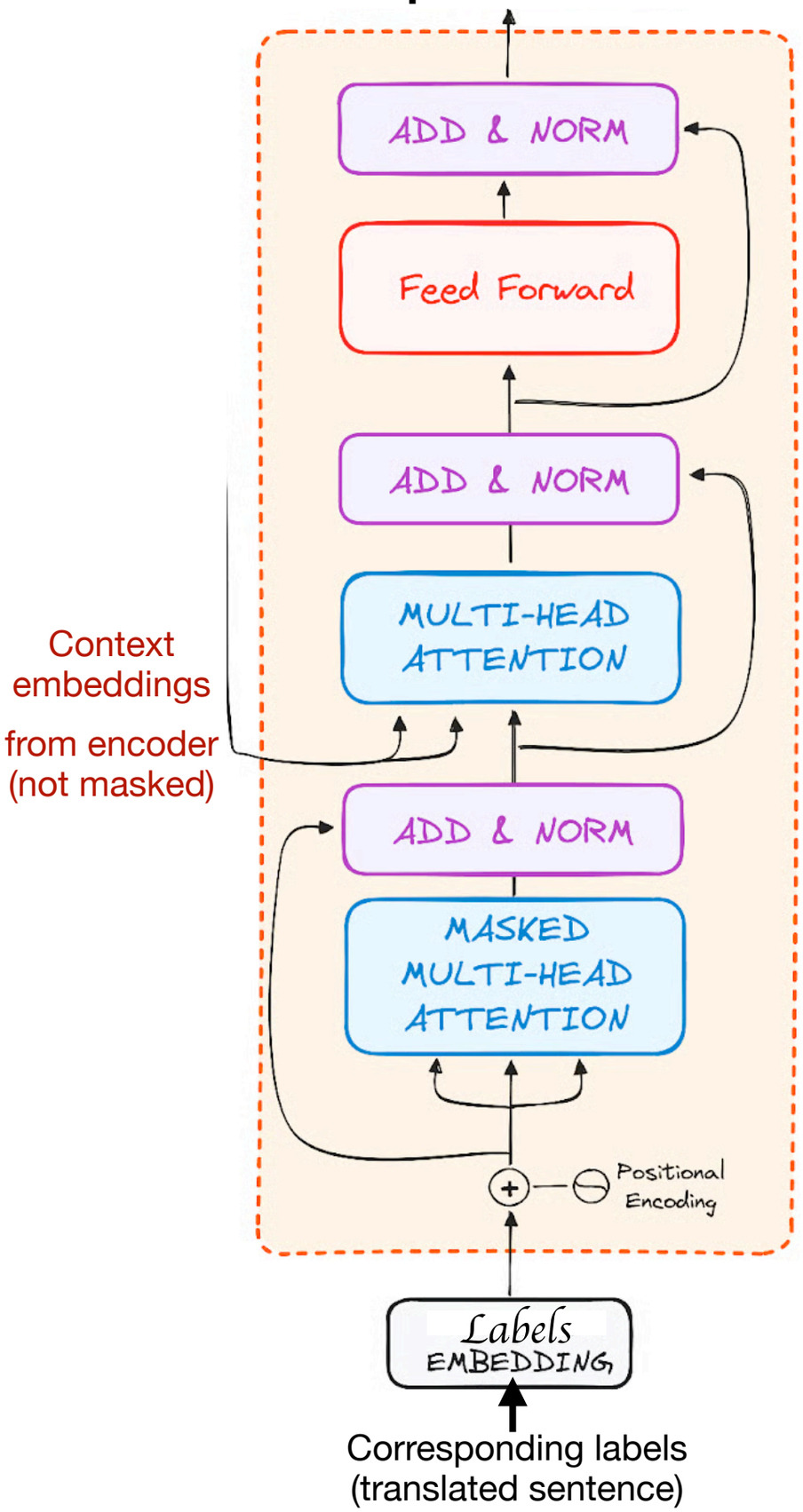


Corresponding labels (translated sentence)

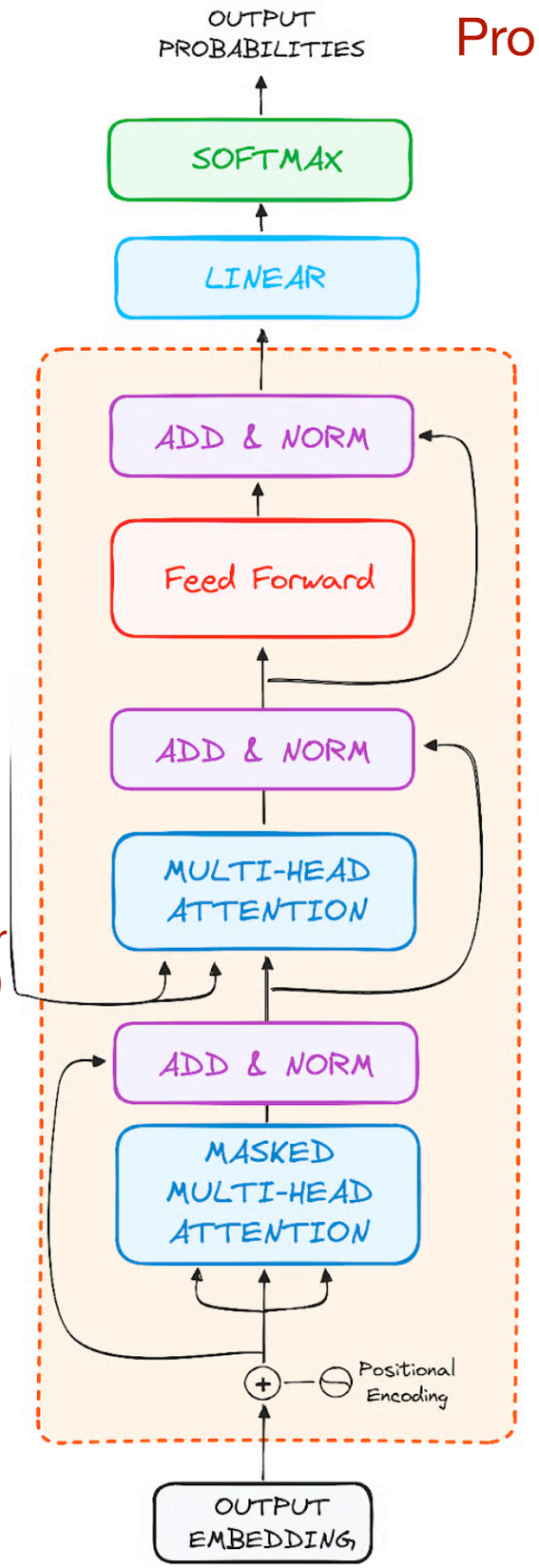
Attention between already generated and context

Masked Self attention

For a sequence to sequence task, we need to “decode”



For a sequence to sequence task, we need to “decode”



Probability over all possible words (or letters) of your dictionnary

Nx

Attention between already generated and context

Context embeddings from encoder (not masked)

Masked Self attention