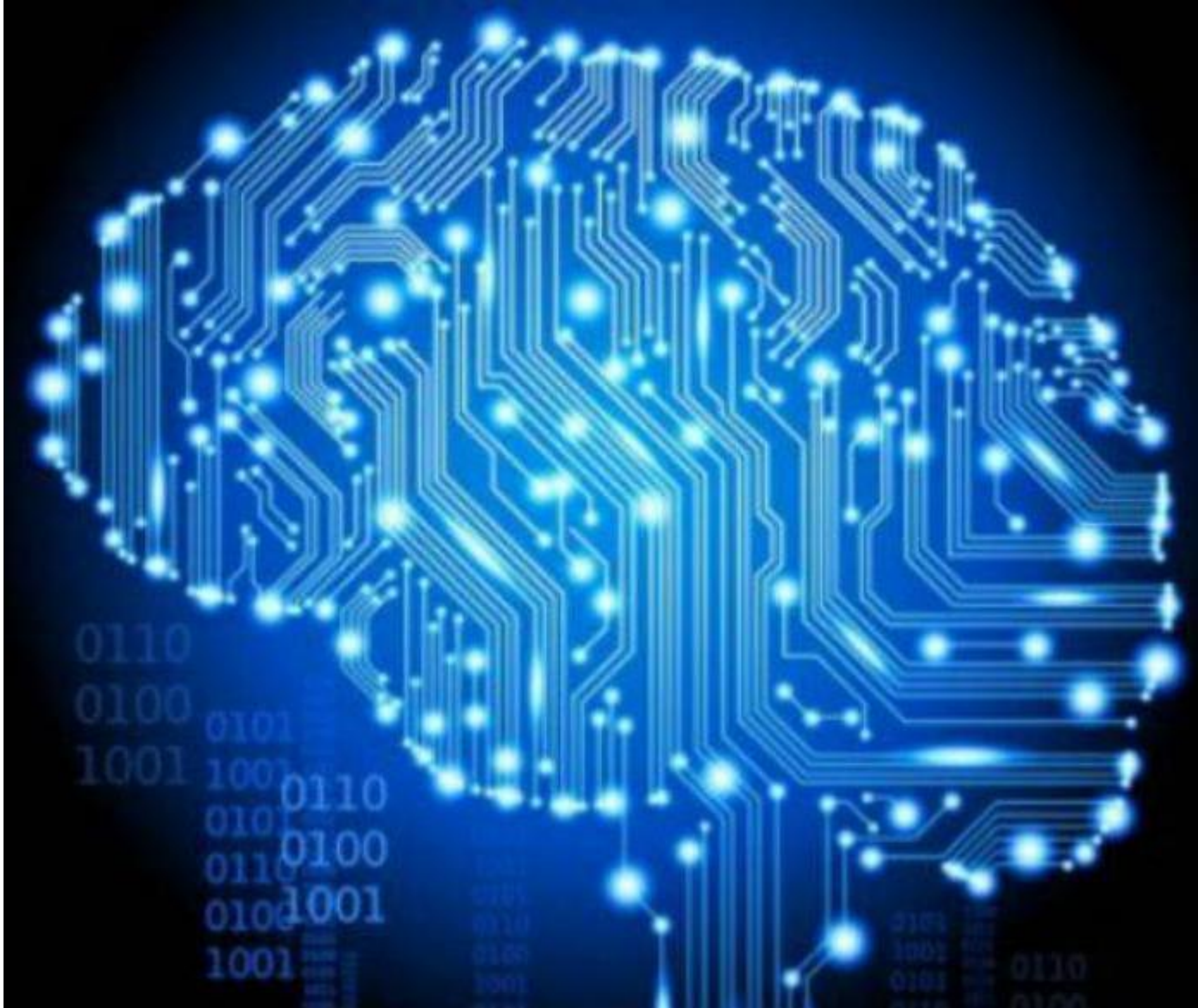
A bit of History



A bit of History

McCullock & Pitts / Hebb
Rosenblatt's Perceptron
Minsky & Papert - XOR
Backpropagation Algorithm

- 1943 Warren McCulloch & Walter Pitts:
  - How To: From neurons to complex thought
  - Binary threshold activations
- 1949 Howard Hebb:
  - Neurons that fire together wire together
  - Weights: Learning and memory

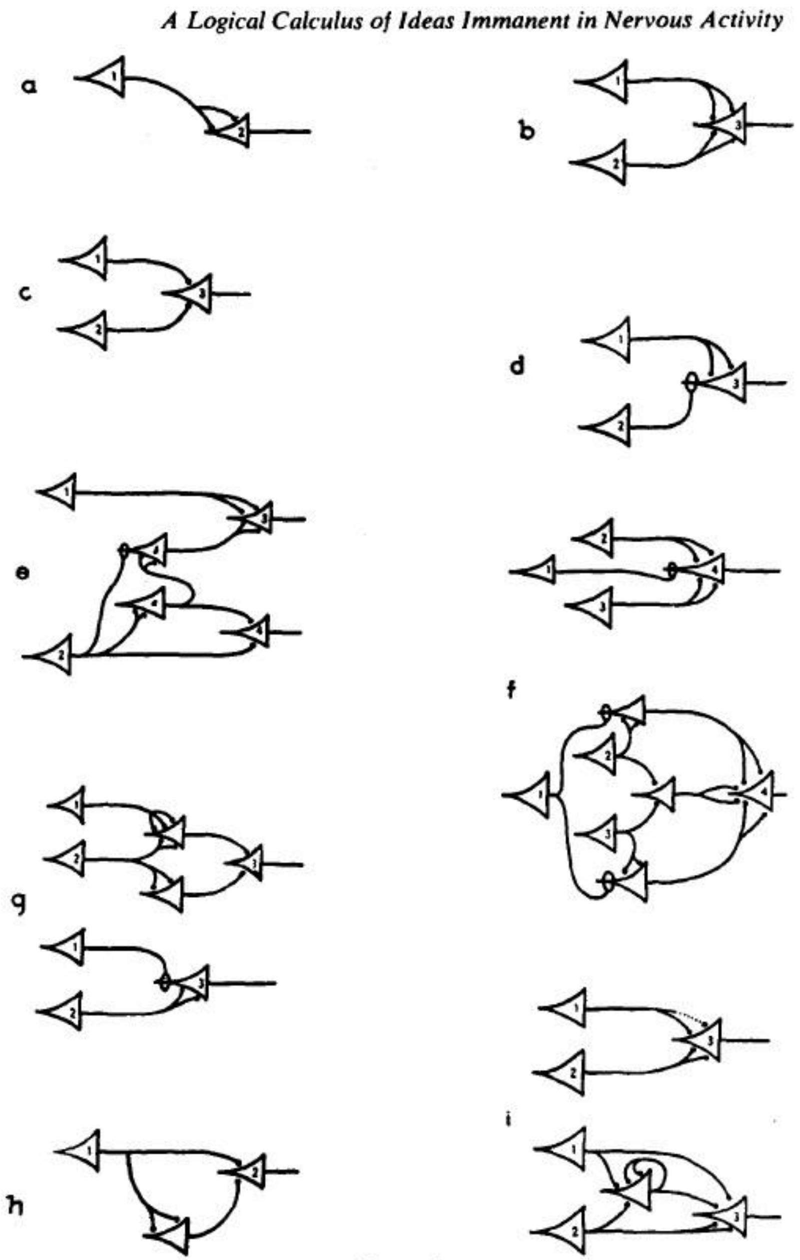


Figure 1

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

[McCullock, 43]

A bit of History

McCullock & Pitts / Hebb

Rosenblatt's Perceptron

Minsky & Papert - XOR

Backpropagation Algorithm

[Rosenblatt, 58]
[Mark I Perceptron]
[Perceptrons]
[1]

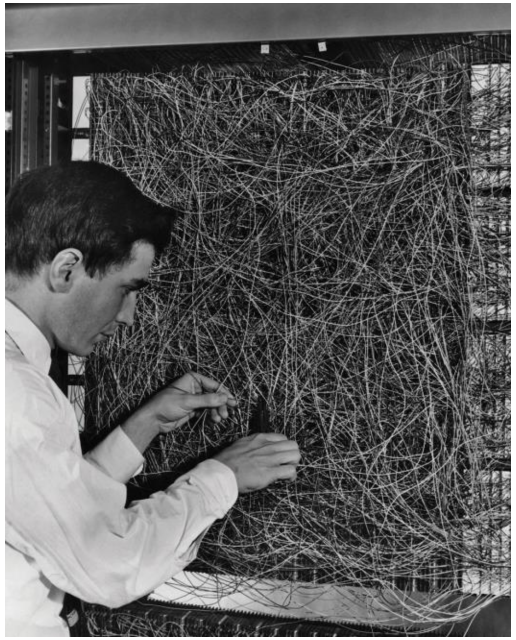
1948, Rosenblatt applied *Hebb's* learning to *McCulloch & Pitts* design

$$f(x) = \begin{cases} 1 \text{ if } w \cdot x + b > 0 \\ 0 \text{ otherwise} \end{cases}$$

w real-valued weights
· dot product
b real scalar constant

The Mark I Perceptron. A visual classifier with:

- 400 photosensitive receptors (sensory units)
- 512 stepping motors (association units, trainable)
- 8 output neurons (response units)



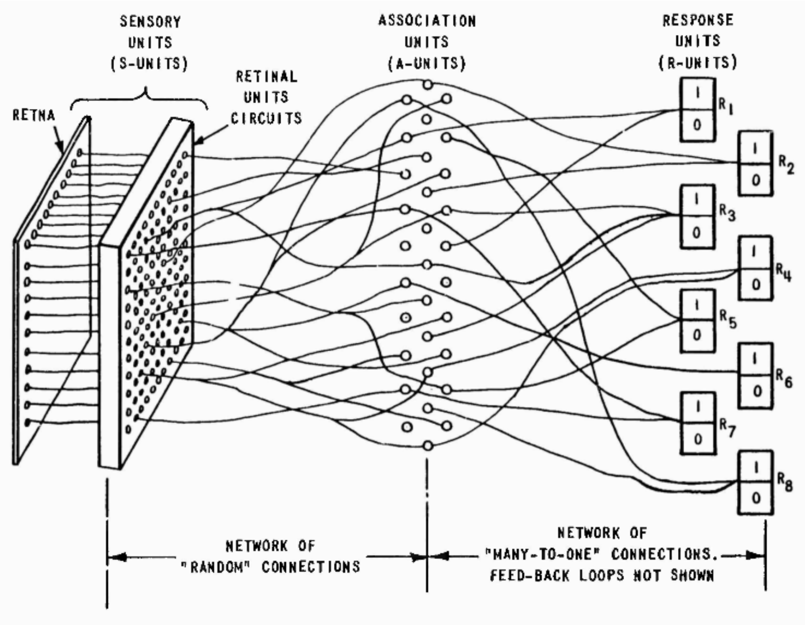


Figure I ORGANIZATION OF THE MARK I PERCEPTRON

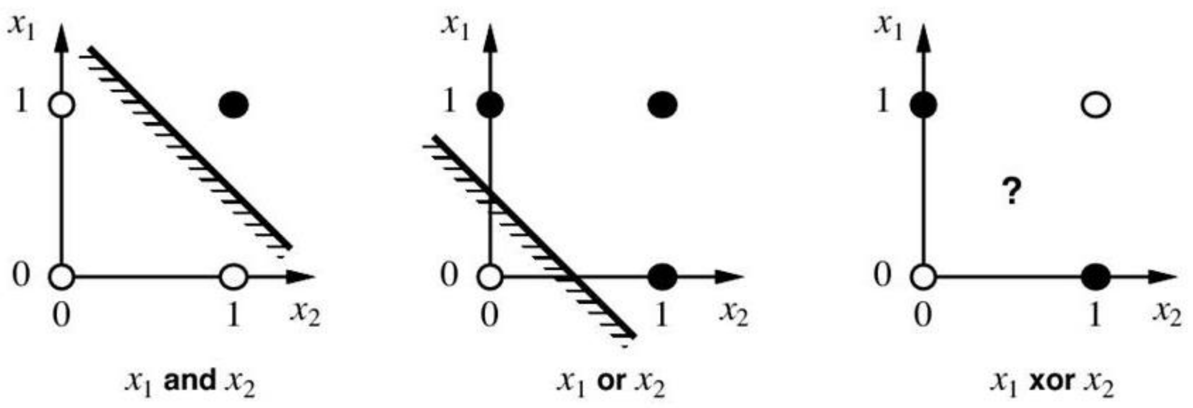
A bit of History

McCullock & Pitts / Hebb
Rosenblatt's Perceptron
Minsky & Papert - XOR
Backpropagation Algorithm

Rosenblatt acknowledged a set of limitations in the Perceptron machine.

Minsky & Papert did too in “*Perceptrons: an introduction to computational geometry*”, including:

- A multilayer perceptron (MLP) is needed for learning basic functions like XOR
- MLP cannot be trained.



This had a huge impact on the public, resulting in a drastic cut in funding of NNs until the mid 80s

1st AI WINTER

[Minsky, 69]

A bit of History

McCullock & Pitts / Hebb
Rosenblatt's Perceptron
Minsky & Papert - XOR
Backpropagation Algorithm

[Werbos,74]
[Rumelhard,85]

How can we optimize neuron weights which are not directly connected to the error measure?

Backpropagation algorithm:
Use the chain rule to find the derivative of cost with respect to any variable.

In other words, find the contribution of each weight to the overall error.

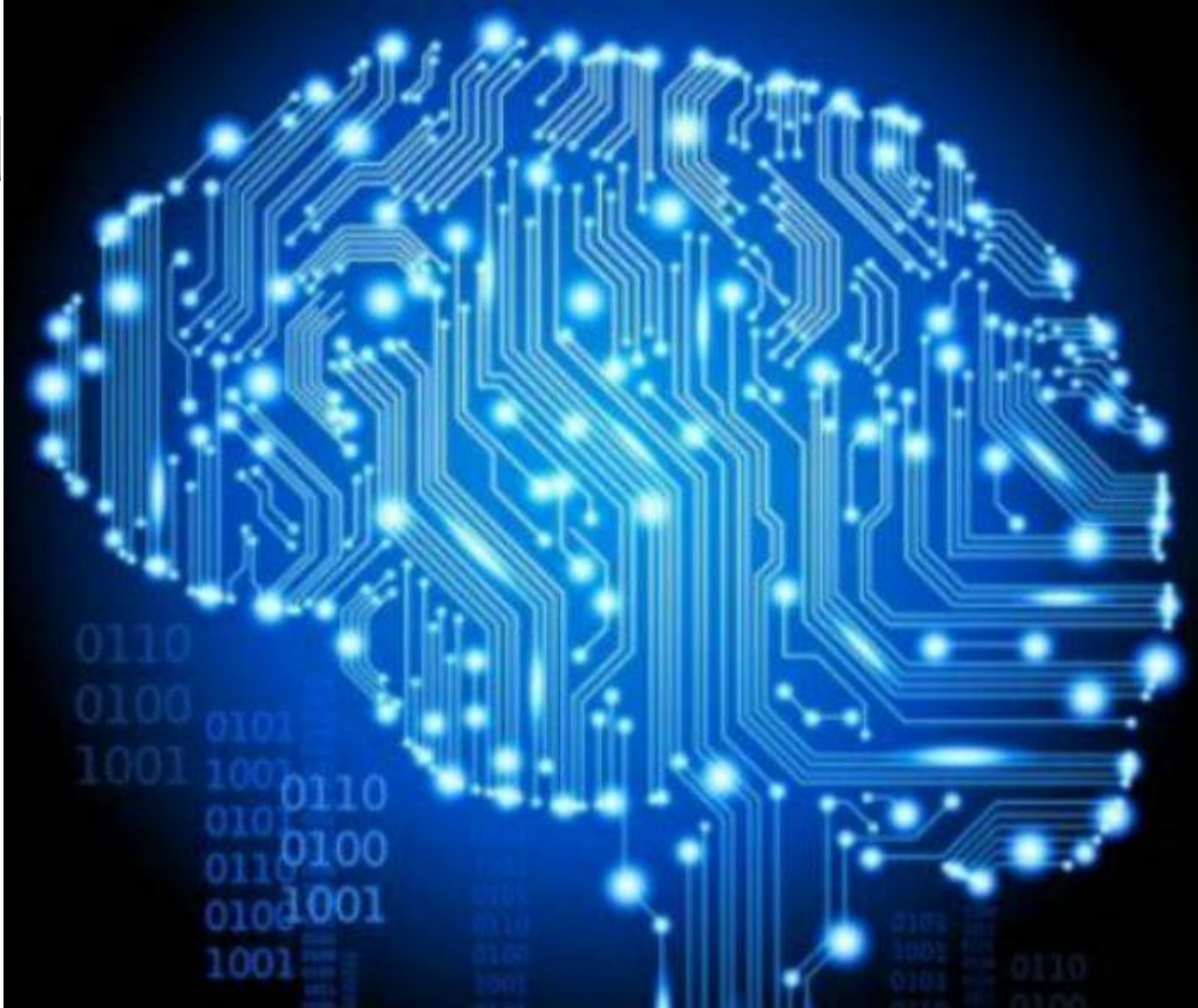
First proposed for training MLPs by *Werbos* in '74.
Rediscovered by *Rumelhart, Hinton and Williams* in '85.

End of NNs Winter

Training with backprop

1. Forward pass from input to output
2. Error measurement (loss function)
3. Find gradients towards minimizing error layer by layer (backward pass)

Feedforward Neural Networks



BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Computing the gradients using all available training data would require huge amounts of memory.

Stochastic Gradient Descent: Iteratively update weights using random samples (hence, *stochastic*)

Each feedforward/backward cycle (a **step**) processes a random **batch** of images.

- Typical batch sizes: Powers of 2.
- Batch size = 1 --> Full stochastic (slower)
- Batch size = dataset_size --> Deterministic (bad generalization)

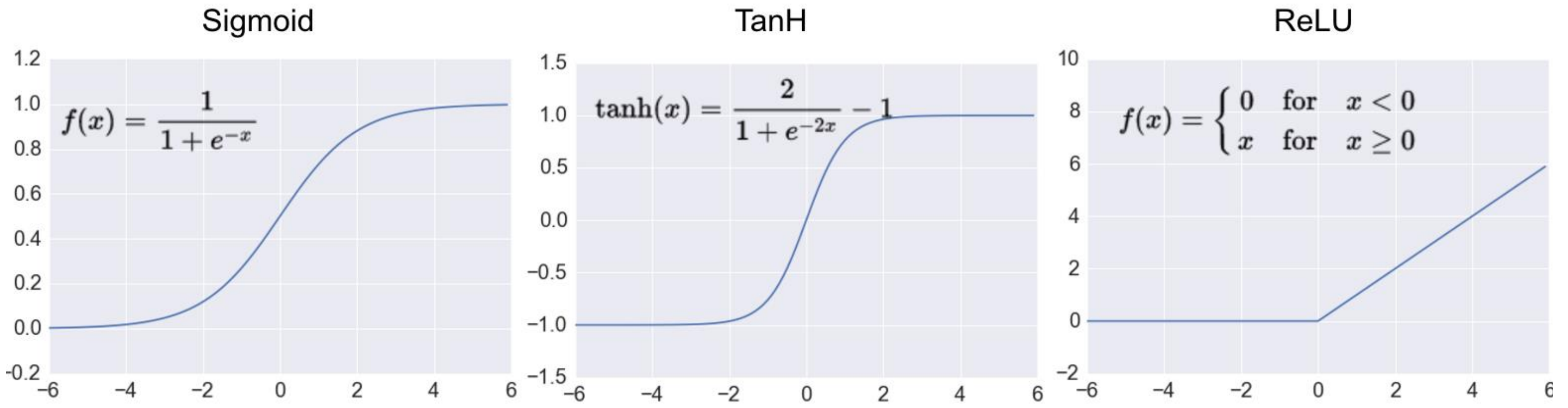
An **epoch** is the processing of the whole dataset once. It corresponds to processing as many batches as:

`dataset_size / batch_size`

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Activation functions transform the output of a layer to a given range. If the function is non-linear, the net can learn non-linear patterns (e.g., XOR).



- Zero gradient in most of f(x). Saturates!
- Max gradient is 0.25 or 1. Vanishing!

- Does not saturate
- Does not vanish
- Faster
- May die

ReLU is a safe choice in most cases
Undying alternatives: Leaky ReLU, ELU, ...

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

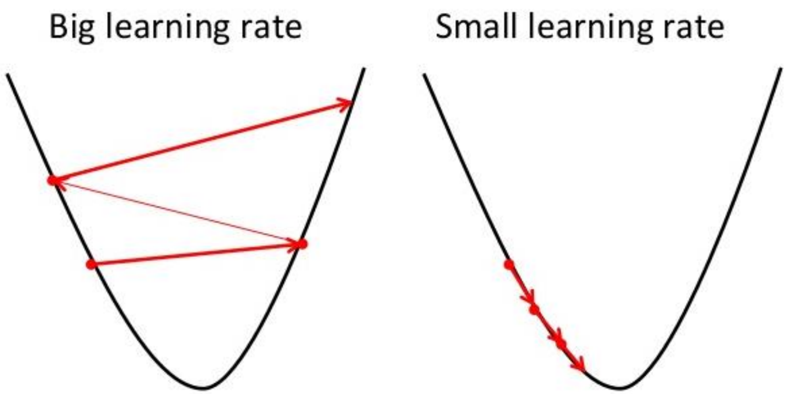
Feedforward Neural Networks

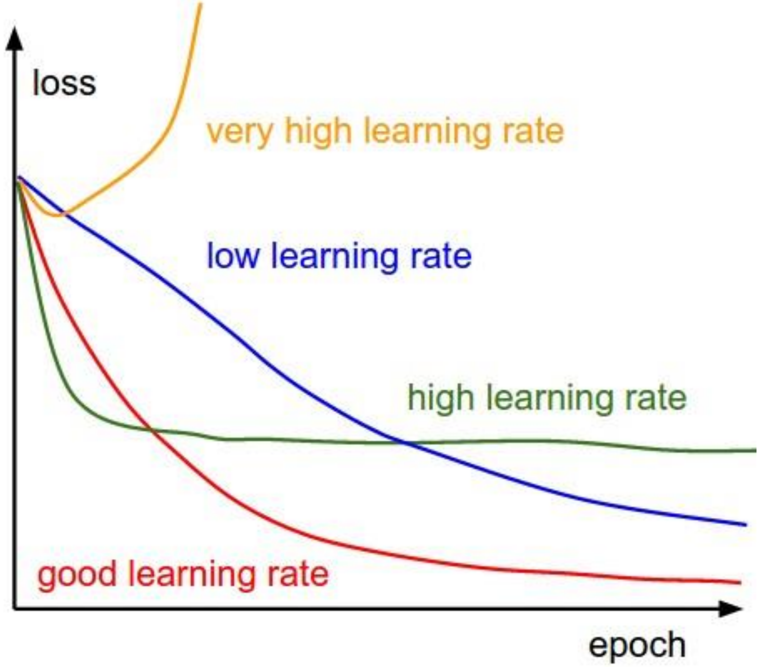
SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Gradient descent is a simple and straight-forward optimization algorithm to update weights towards a min.

Learning rate determines how much we move in that direction. With the wrong LR you may end up in local minima or saddle points, or be too slow.

SGD will overshoot unless we keep decreasing the LR.





Barcelona Supercomputing Center
Centro Nacional de Supercomputación

[Dauphin, 14]

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Momentum: Include a fraction of the previous gradient. Keeps the general direction so far.

Nesterov: Compute current gradient considering where the previous gradient took you. (RNNs?)

Adagrad: Parameter-wise LR considering past updates. Good for infrequent patterns (GloVe). Vanishing LR due to growing history.

Adadelta: Adagrad with a decaying average over history. Typically set around 0.9.

Adam: Adadelta + Momentum

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

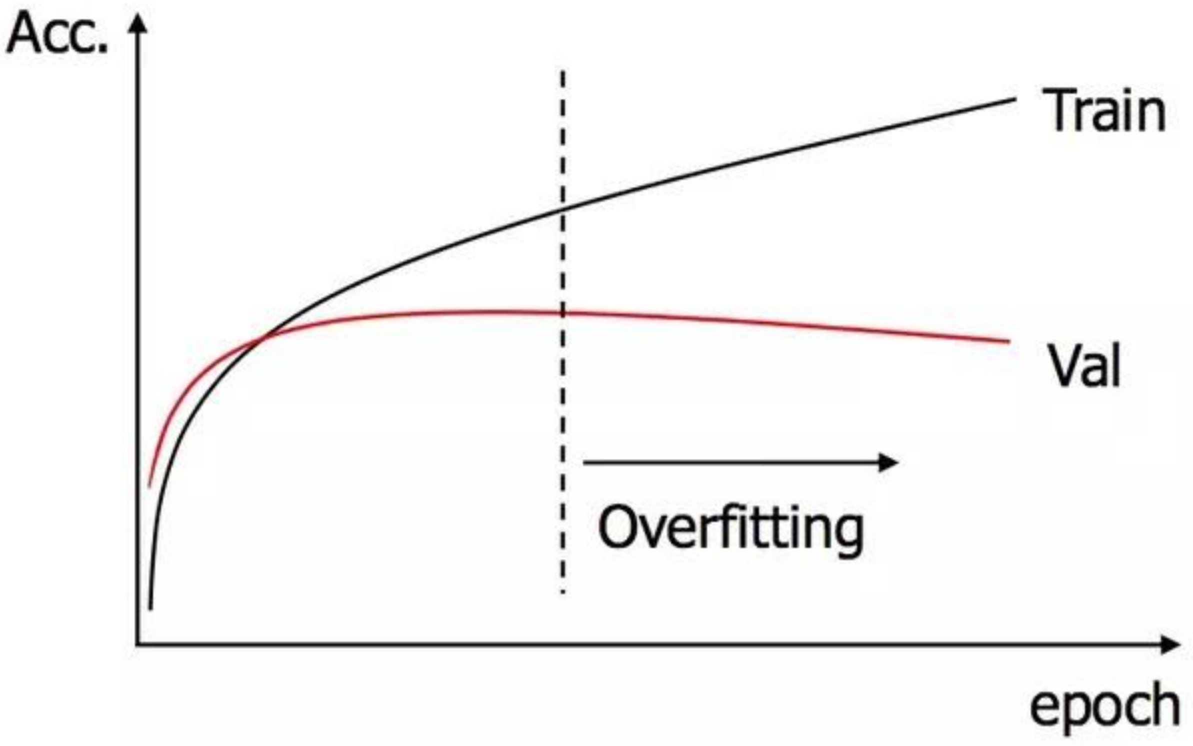
[Dauphin, 14]
[Ruder,www]

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Why do we need regularization?

Because the difference between **Machine Learning** and **Optimization** is called **Generalization**



Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

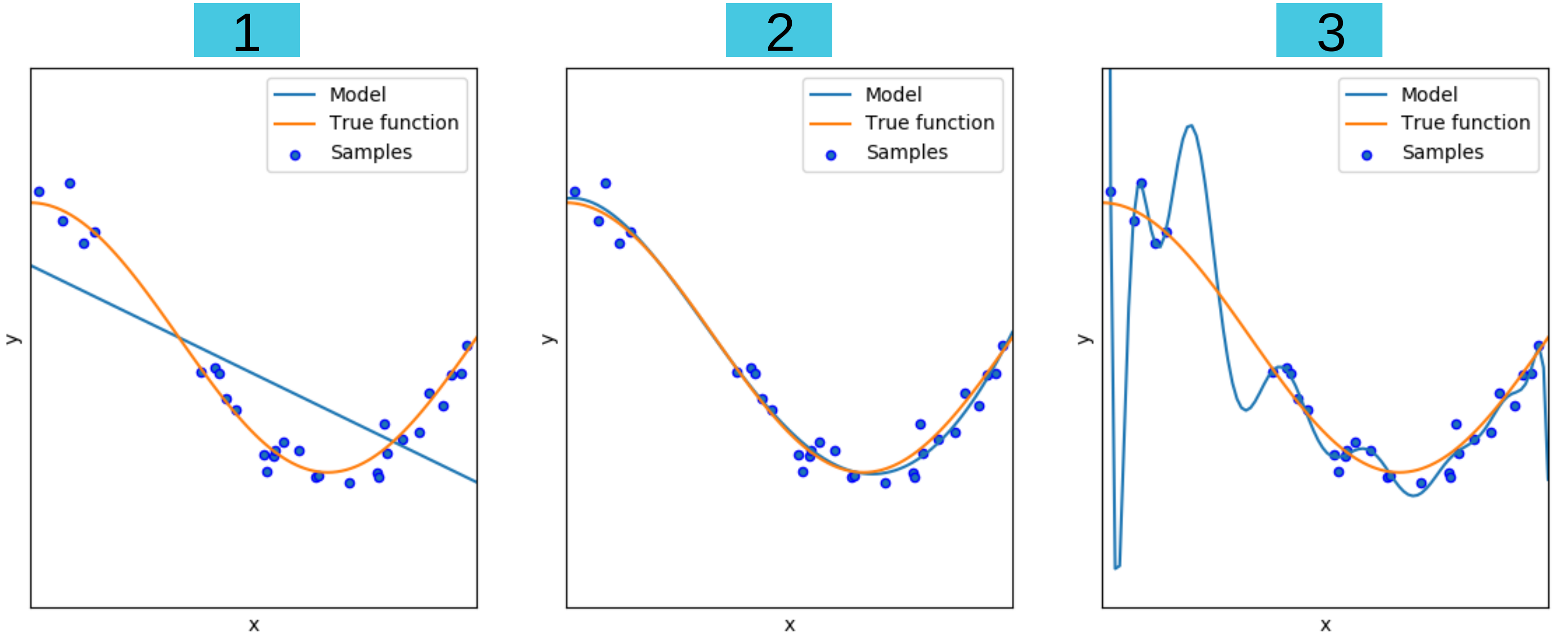
Generalization

Polynomial regression

1 $h(x) = w_1 x + b$

2 $h(x) = w_3 x^3 + w_2 x^2 + w_1 x + b$

3 $h(x) = w_{14} x^{14} + w_{13} x^{13} + \cdots + w_1 x + b$



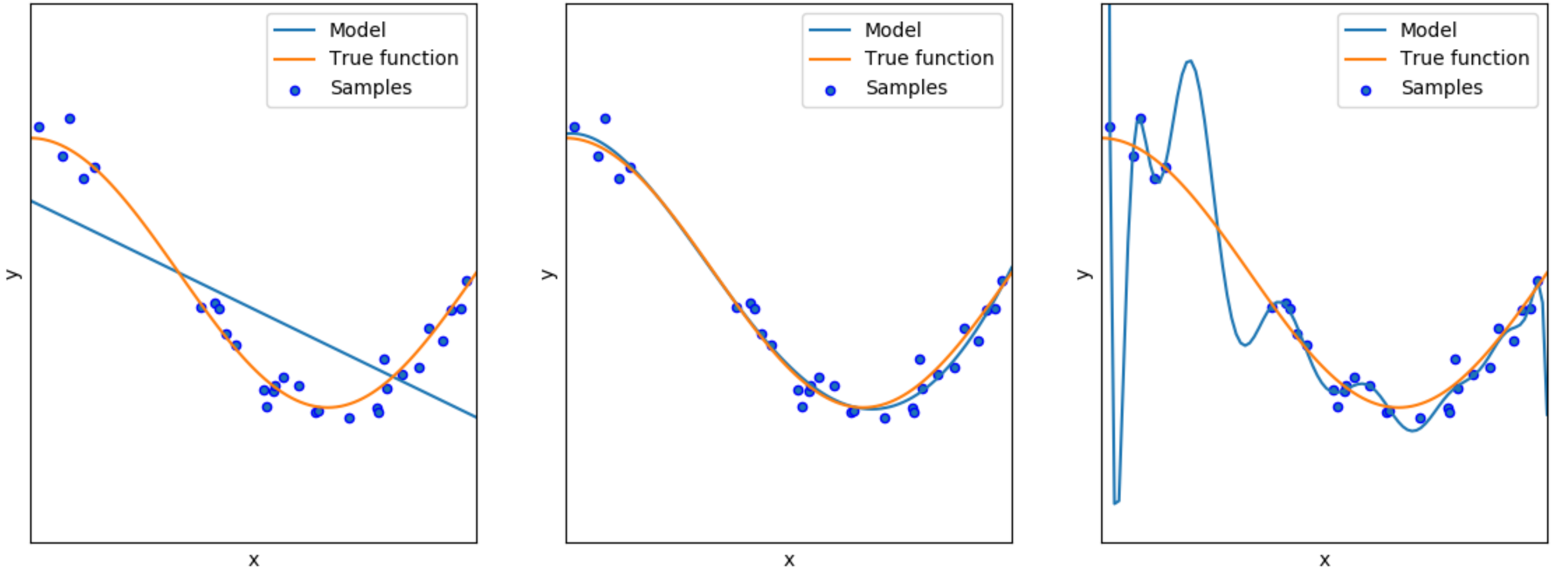
Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Generalization

Polynomial regression

	Huge	Small	Tiny
Training Error	Huge	Small	Tiny
Model Generalization	Bad	Good	Horrible



Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Generalization

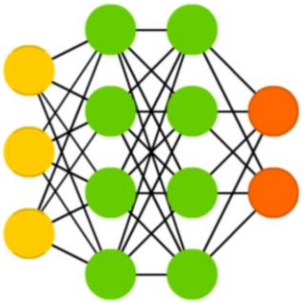
What **policy** can we use to **improve model generalization**?

Occam's Razor

when you have **two competing hypotheses**

that make the **same predictions**,

the **simpler one is the better**



Machine Learning

given **two models**

that have a **similar performance**,

It's better to **choose the simpler one**

Feedforward Neural Networks

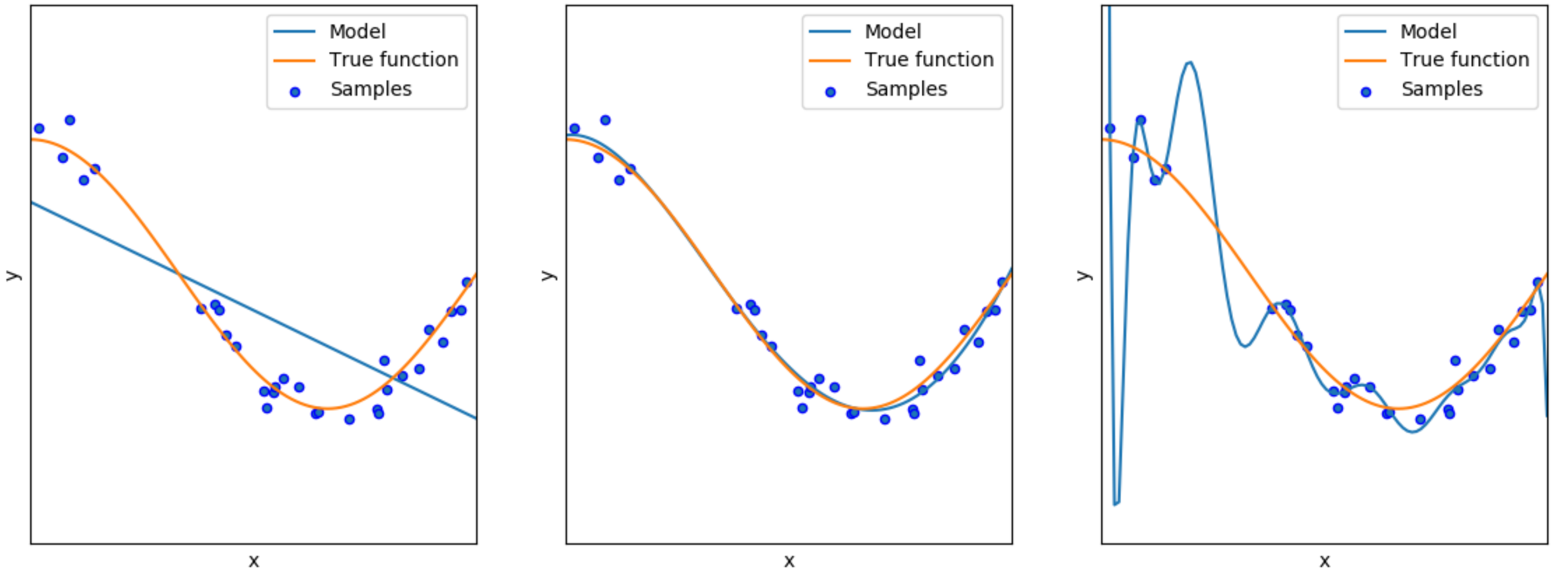
SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Model Complexity

What **policy** can we use to **improve model generalization**?

$$\text{Cost function} = \text{Training Error} + \textbf{Model Complexity}$$



Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Model Complexity

$$h(x) = w_3x^3 + w_2x^2 + w_1x + w_0 \quad \textbf{VS} \quad h(x) = 0x^3 + 0x^2 + w_1x + w_0$$

Feedforward Neural Networks

- SGD, Epochs, Batches and Steps
- Activation functions
- SGD learning rate
- Other optimization methods
- **Regularization**
- Normalizing inputs
- Vanishing/Exploding Gradients
- Weights initialization

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Model Complexity

$$h(x) = w_3x^3 + w_2x^2 + w_1x + w_0 \quad \text{VS} \quad h(x) = 0x^3 + 0x^2 + w_1x + w_0$$

$$h(x) = w_3x^3 + w_2x^2 + w_1x + w_0 \quad \text{VS} \quad h(x) = 0x^3 + w_2x^2 + 0x + w_0$$

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Model Complexity

$$h(x) = w_3x^3 + w_2x^2 + w_1x + w_0 \quad \text{VS} \quad h(x) = 0x^3 + 0x^2 + w_1x + w_0$$

$$h(x) = w_3x^3 + w_2x^2 + w_1x + w_0 \quad \text{VS} \quad h(x) = 0x^3 + w_2x^2 + 0x + w_0$$

$$h(x) = 0x^3 + 0x^2 + w_1x + w_0 \quad \text{VS} \quad h(x) = 0x^3 + w_2x^2 + 0x + w_0$$

Feedforward Neural Networks

SGD, Epochs, Batches and Steps

Activation functions

SGD learning rate

Other optimization methods

Regularization

Normalizing inputs

Vanishing/Exploding Gradients

Weights initialization

Model Complexity

$h(x) = w_3x^3 + w_2x^2 + w_1x + w_0$ VS $h(x) = 0x^3 + 0x^2 + w_1x + w_0$

$h(x) = w_3x^3 + w_2x^2 + w_1x + w_0$ VS $h(x) = 0x^3 + w_2x^2 + 0x + w_0$

$h(x) = 0x^3 + 0x^2 + w_1x + w_0$ VS $h(x) = 0x^3 + w_2x^2 + 0x + w_0$

$h(x) = 0x^3 + 0x^2 + w_1x + w_0$ VS $h(x) = 0x^3 + w_2x^2 + 0x + 0$

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Model Complexity

$h(x) = w_3x^3 + w_2x^2 + w_1x + w_0$ VS $h(x) = 0x^3 + 0x^2 + w_1x + w_0$

$h(x) = w_3x^3 + w_2x^2 + w_1x + w_0$ VS $h(x) = 0x^3 + w_2x^2 + 0x + w_0$

$h(x) = 0x^3 + 0x^2 + w_1x + w_0$ VS $h(x) = 0x^3 + w_2x^2 + 0x + w_0$

$h(x) = 0x^3 + 0x^2 + w_1x + w_0$ VS $h(x) = 0x^3 + w_2x^2 + 0x + 0$

? ? ?

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Model Complexity

$h(x) = 0x^3 + 0x^2 + w_1 x + w_0$ VS $h(x) = 0x^3 + w_2 x^2 + 0x + 0$

ℓ_0 complexity: Number of non-zero coefficients

ℓ_1 "lasso" complexity: $\sum_{i=0}^{d} |w_i|$, for coefficients $w_0, ..., w_d$

ℓ_2 "ridge" complexity: $\sum_{i=0}^{d} w_i^2$, for coefficients $w_0, ..., w_d$

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Model Complexity

$$h(x) = 0x^3 + 0x^2 + w_1 x + w_0 \quad \text{VS} \quad h(x) = 0x^3 + w_2 x^2 + 0x + 0$$

$w_0 = 1.3 \quad w_1 = -1.2$ $\qquad$ $w_2 = 2.2$

ℓ_0 complexity

$$|\{w_1, w_0\}| = 2 \quad \text{VS} \quad |\{w_2\}| = 1$$

ℓ_1 complexity

$$|1.3| + |-1.2| = 2.5 \quad \text{VS} \quad |2.2| = 2.2$$

ℓ_2 complexity

$$1.3^2 + (-1.2)^2 = 3.13 \quad \text{VS} \quad 2.2^2 = 4.84$$

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

L1 / L2 Regularization

$$\text{Cost function} = \text{Loss} + \boxed{\frac{\lambda}{m}\sum_{i=0}^{m}|w_i|}$$

$$\text{Cost function} = \text{Loss} + \boxed{\frac{\lambda}{2m}\sum_{i=0}^{m}w_i^2}$$

$$\text{Regularization parameter} \rightarrow \lambda$$

What **complexities** do these methods use?

ℓ_1 "lasso" complexity: $\sum_{i=0}^{d}|w_i|$, for coefficients $w_0, ..., w_d$

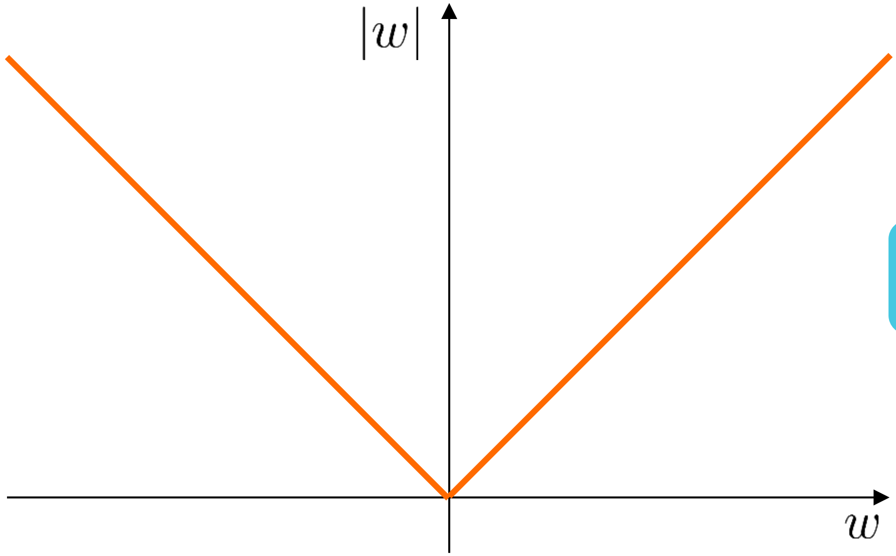
ℓ_2 "ridge" complexity: $\sum_{i=0}^{d}w_i^2$, for coefficients $w_0, ..., w_d$

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

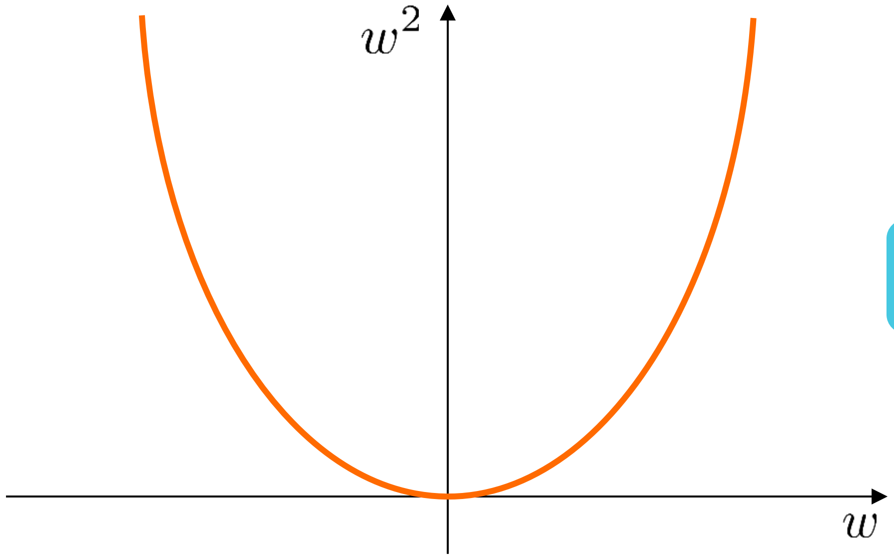
Feedforward Neural Networks

- SGD, Epochs, Batches and Steps
- Activation functions
- SGD learning rate
- Other optimization methods
- Regularization
- Normalizing inputs
- Vanishing/Exploding Gradients
- Weights initialization

L1 / L2 Regularization



L1 Penalty

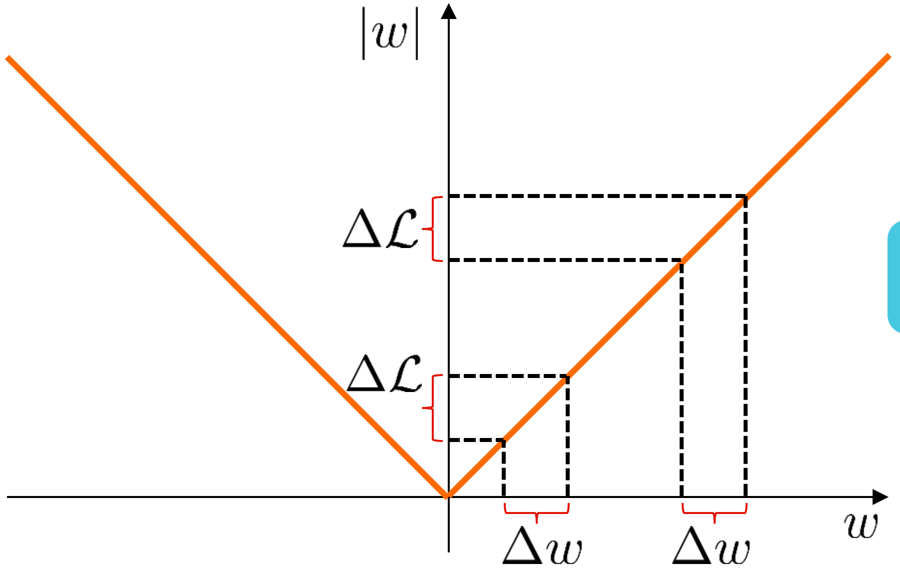


L2 Penalty

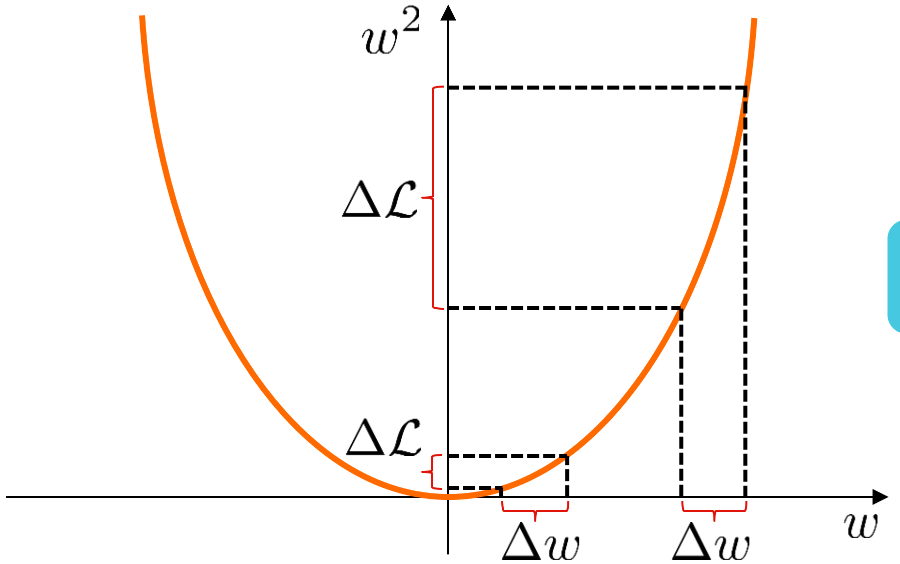
Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

L1 / L2 Regularization



L1 Penalty



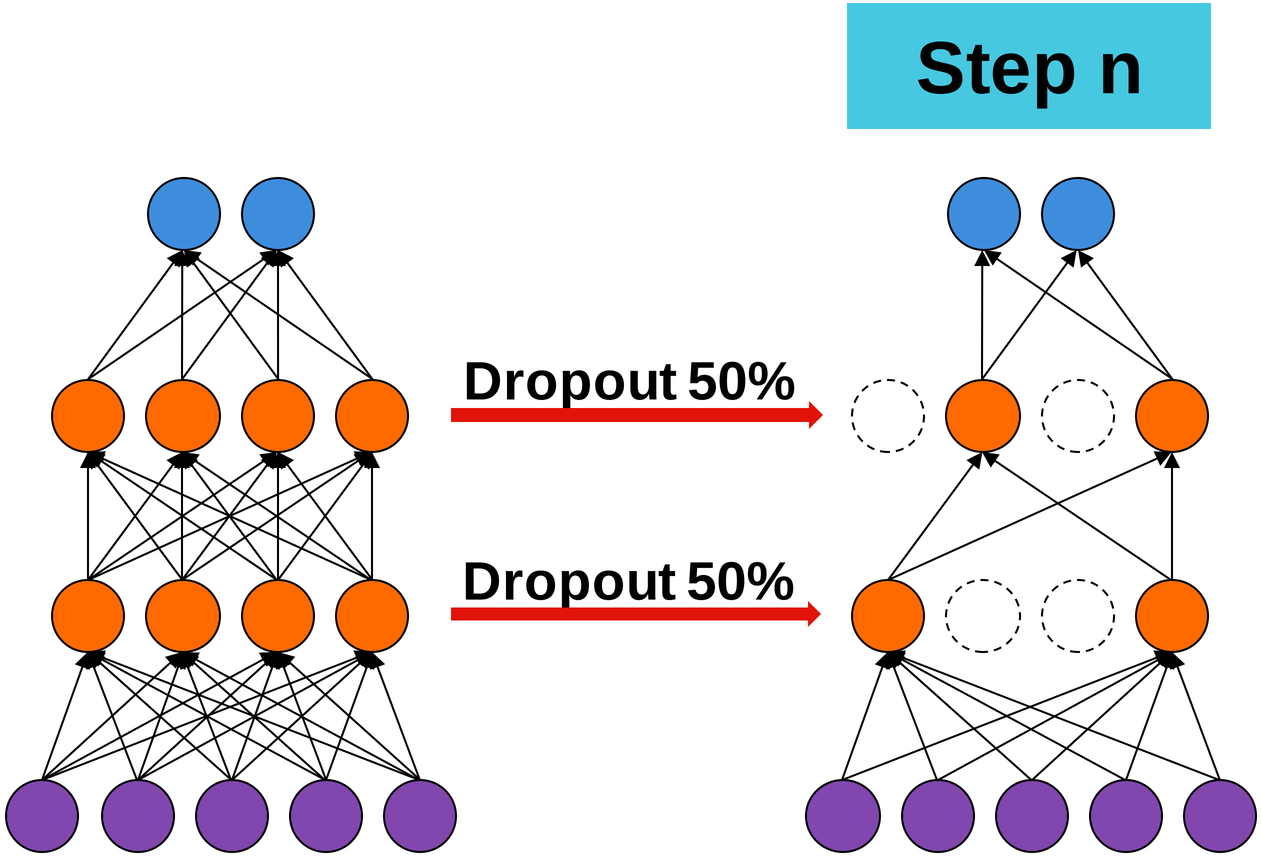
L2 Penalty

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Feedforward Neural Networks

- SGD, Epochs, Batches and Steps
- Activation functions
- SGD learning rate
- Other optimization methods
- Regularization
- Normalizing inputs
- Vanishing/Exploding Gradients
- Weights initialization

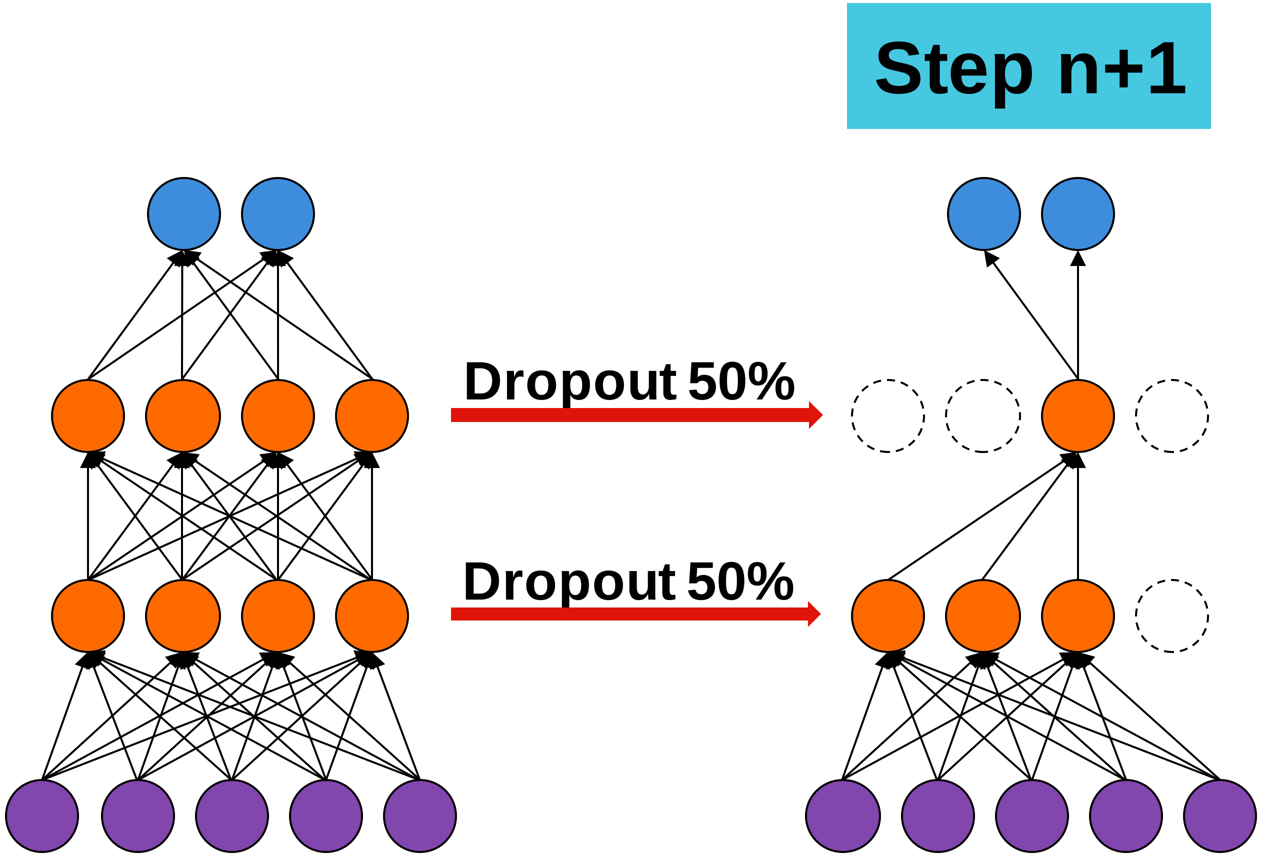
Dropout



Feedforward Neural Networks

- SGD, Epochs, Batches and Steps
- Activation functions
- SGD learning rate
- Other optimization methods
- Regularization
- Normalizing inputs
- Vanishing/Exploding Gradients
- Weights initialization

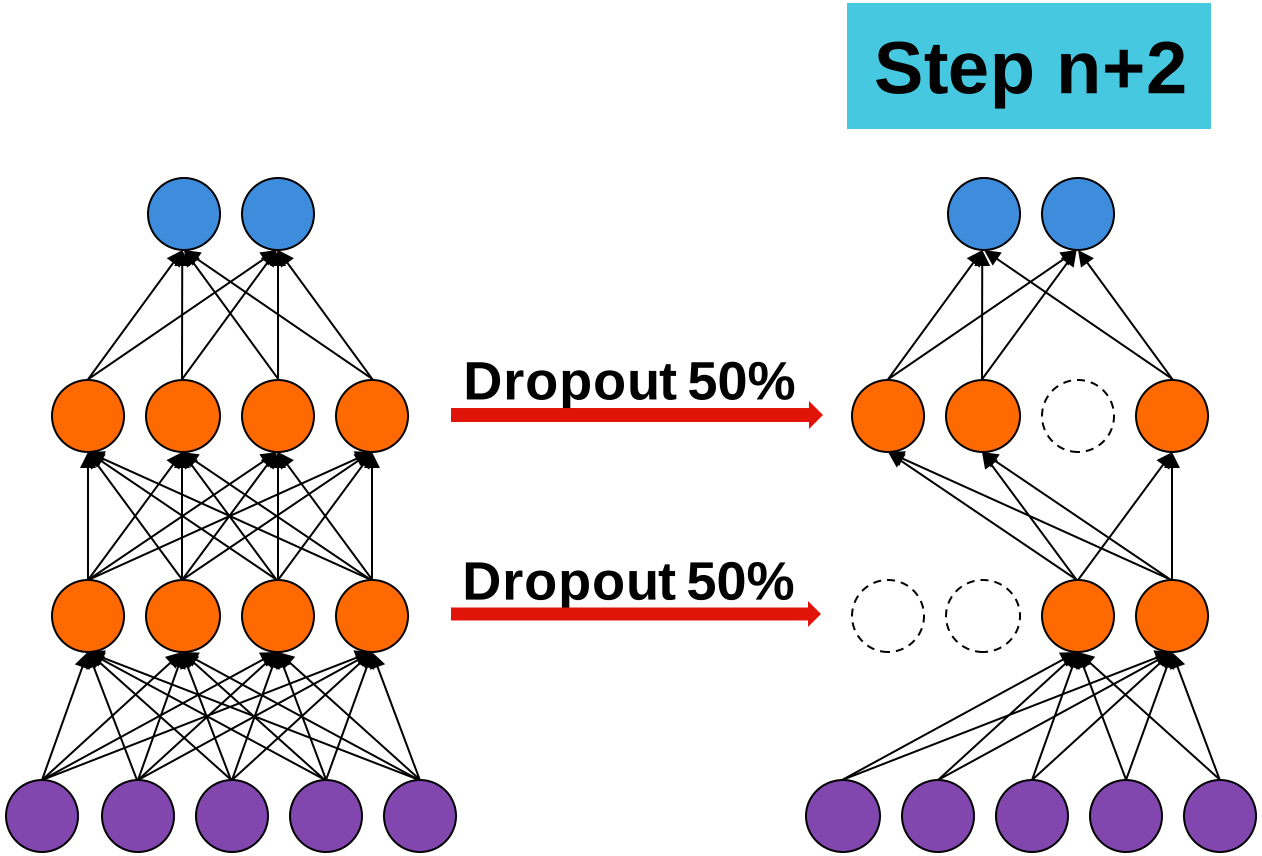
Dropout



Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Dropout

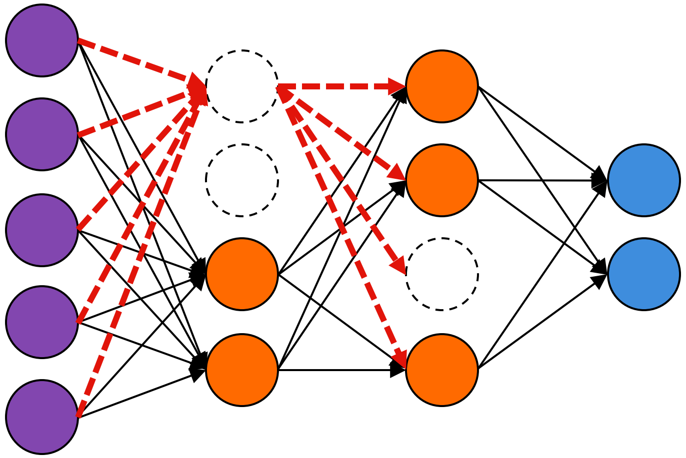


Feedforward Neural Networks

- SGD, Epochs, Batches and Steps
- Activation functions
- SGD learning rate
- Other optimization methods
- **Regularization**
- Normalizing inputs
- Vanishing/Exploding Gradients
- Weights initialization

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Dropout



Before drop-out:

$$a_0^{[0]} = g\left(w_{00}^{[0]}x_0 + w_{10}^{[0]}x_1 + w_{20}^{[0]}x_2 + w_{30}^{[0]}x_3 + w_{40}^{[0]}x_4 + b_0^{[0]}\right)$$

After drop-out: $a_0^{[0]} = 0$

What **complexity** does this method use?

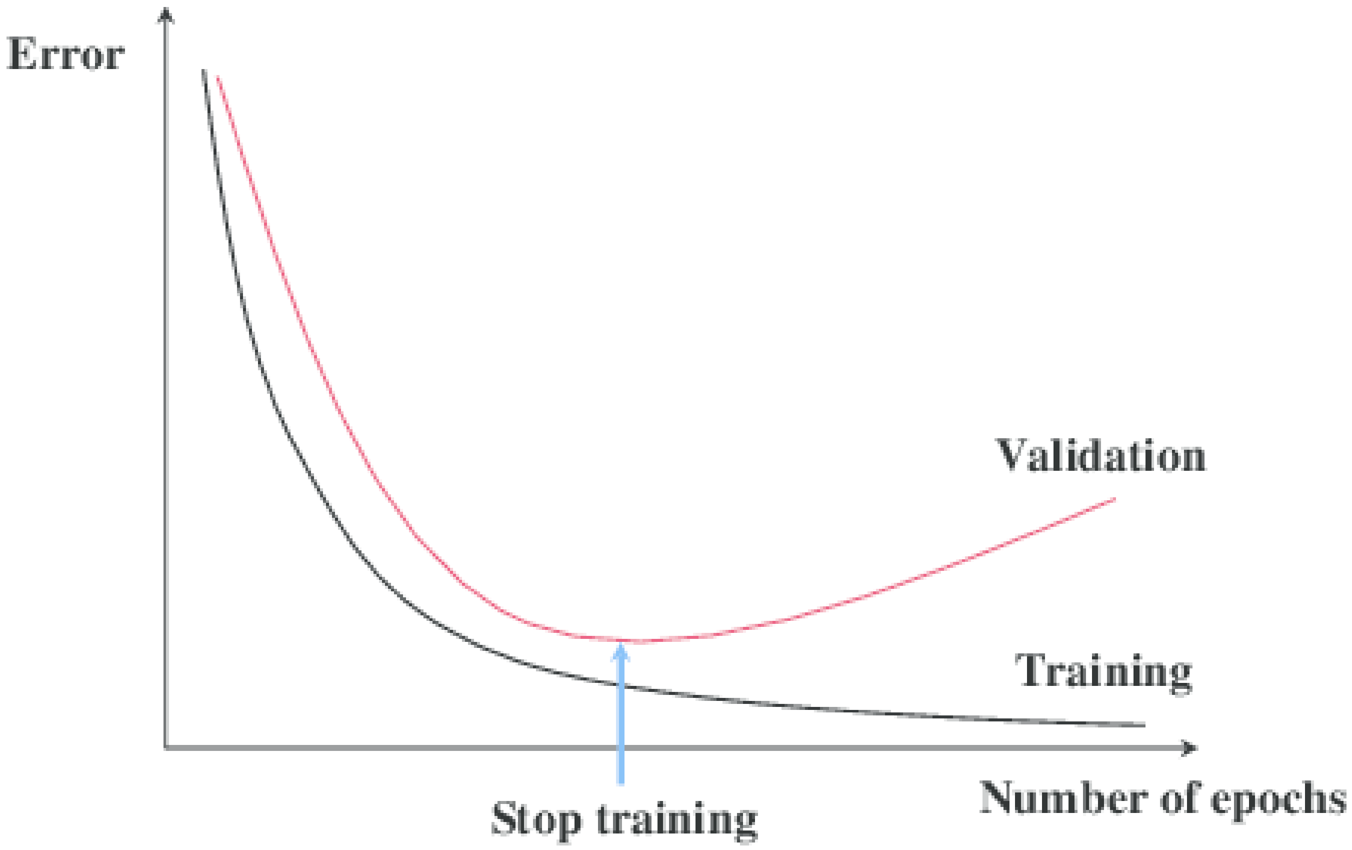
ℓ_0 complexity: Number of non-zero coefficients

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Early Stopping

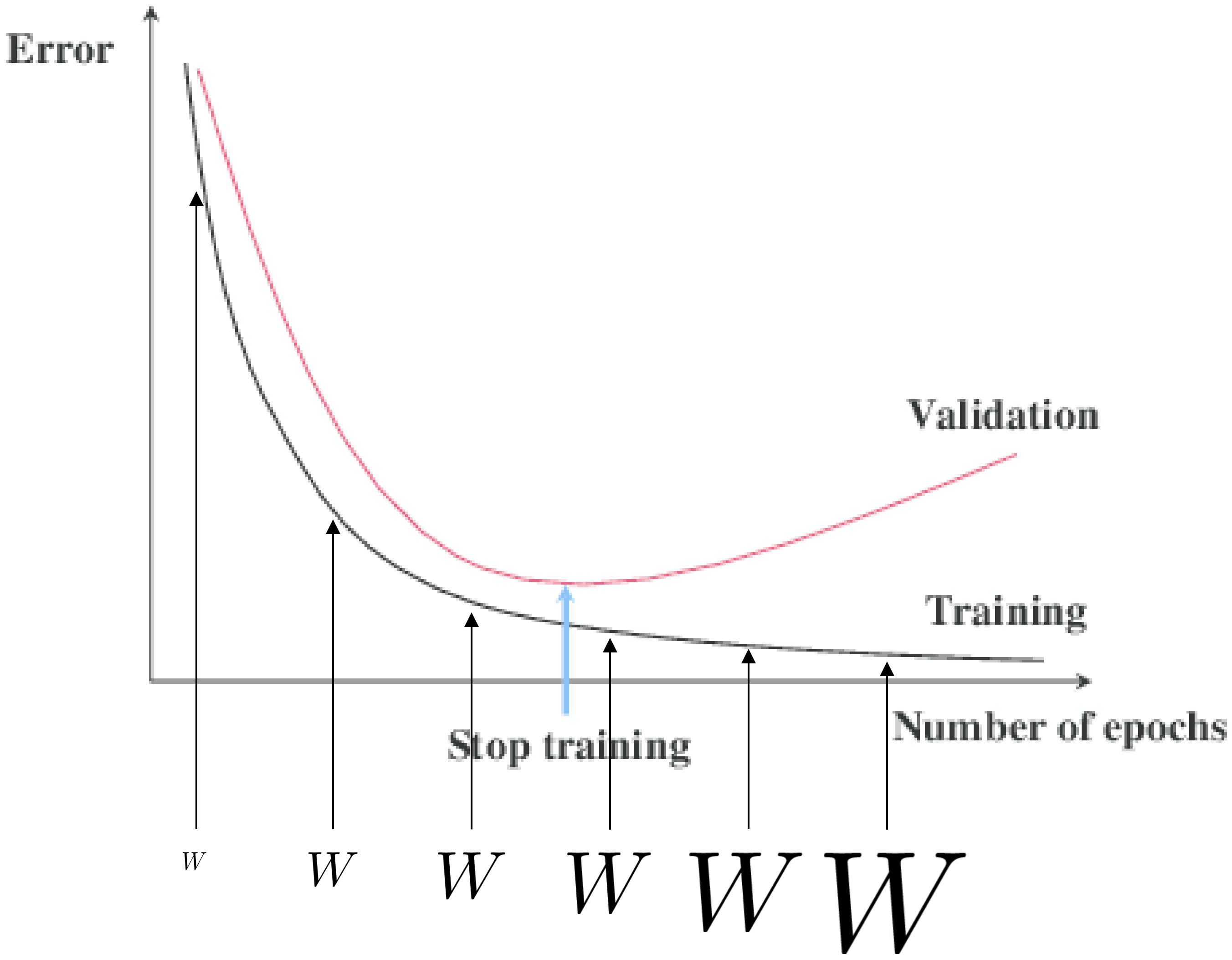


Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

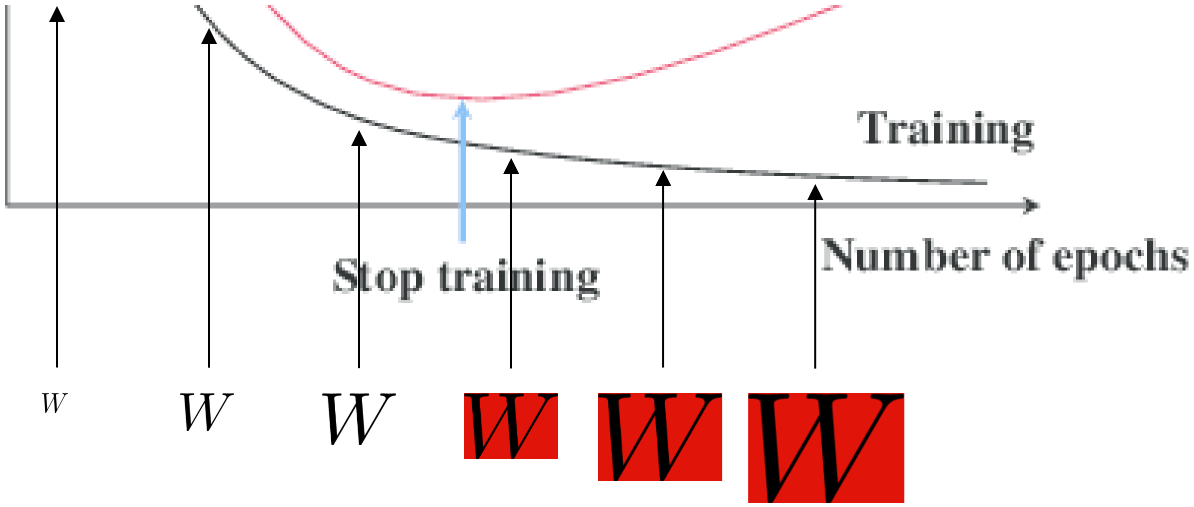
Early Stopping



Feedforward Neural Networks

- SGD, Epochs, Batches and Steps
- Activation functions
- SGD learning rate
- Other optimization methods
- **Regularization**
- Normalizing inputs
- Vanishing/Exploding Gradients
- Weights initialization

Early Stopping



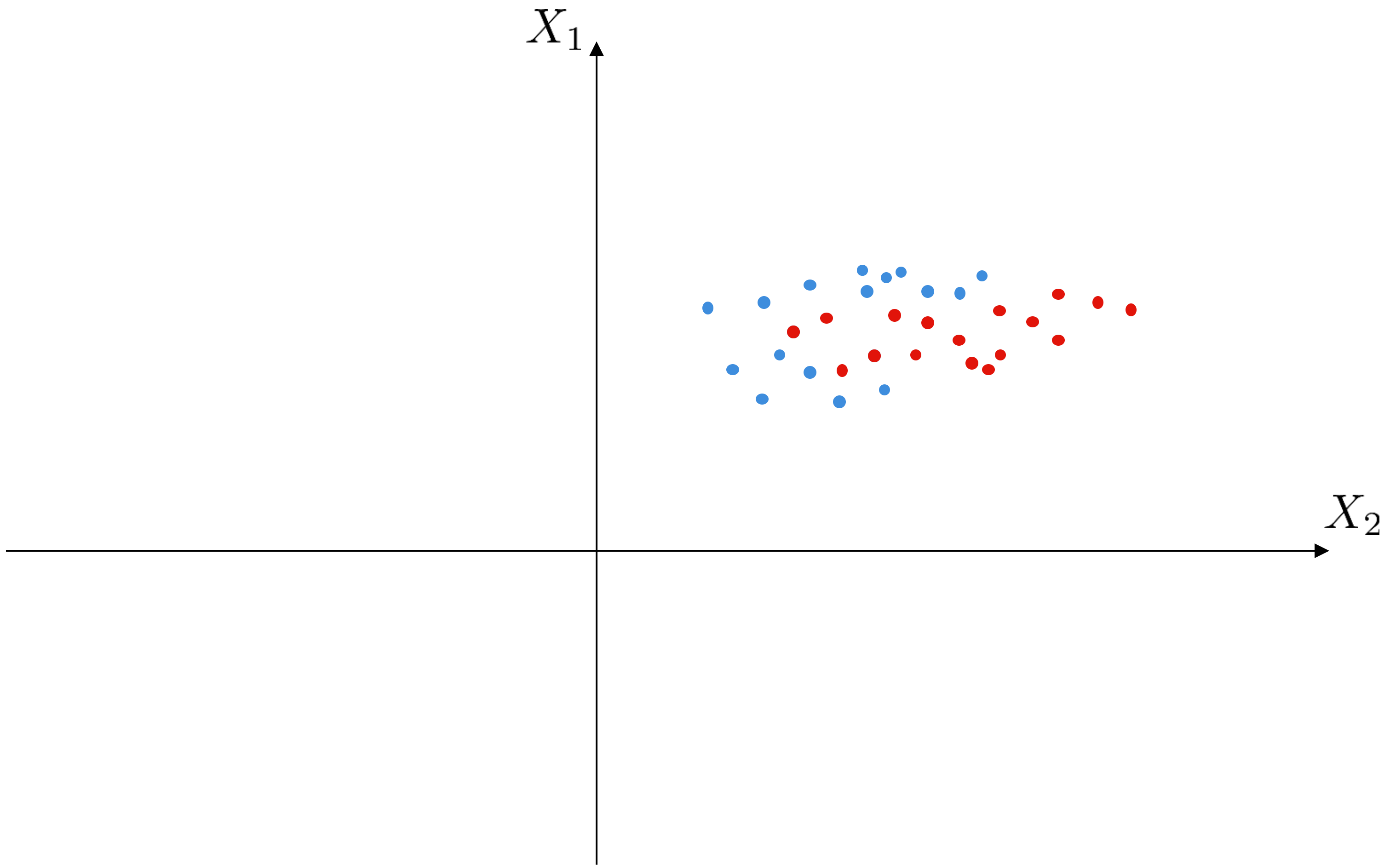
What **complexity** does this method use?

ℓ_1 "lasso" complexity: $\sum_{i=0}^{d} |w_i|$, for coefficients $w_0, \ldots, w_d$

ℓ_2 "ridge" complexity: $\sum_{i=0}^{d} w_i^2$, for coefficients $w_0, \ldots, w_d$

Feedforward Neural Networks

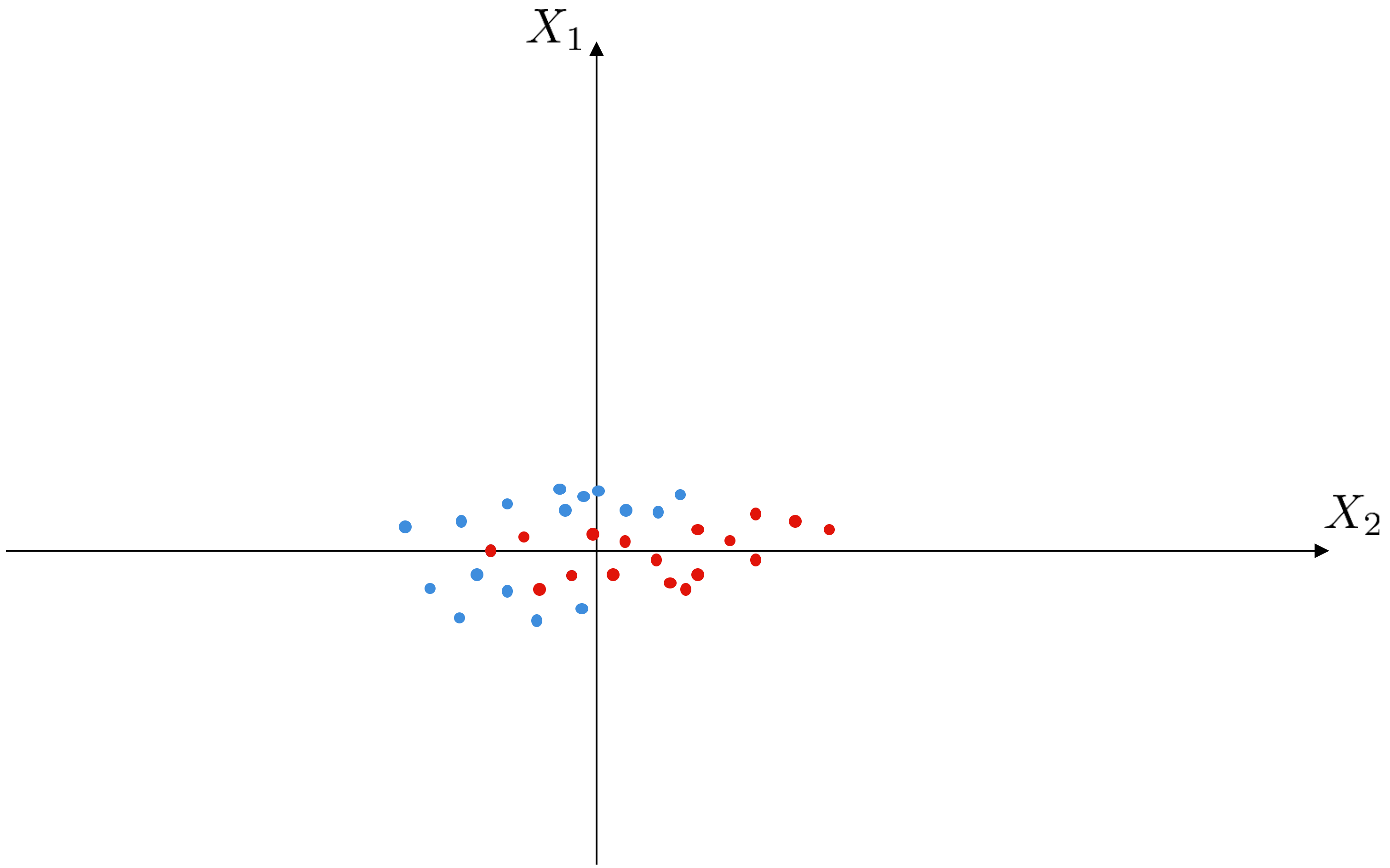
SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



$$x = \frac{x - \mu}{\sigma^2}$$

Feedforward Neural Networks

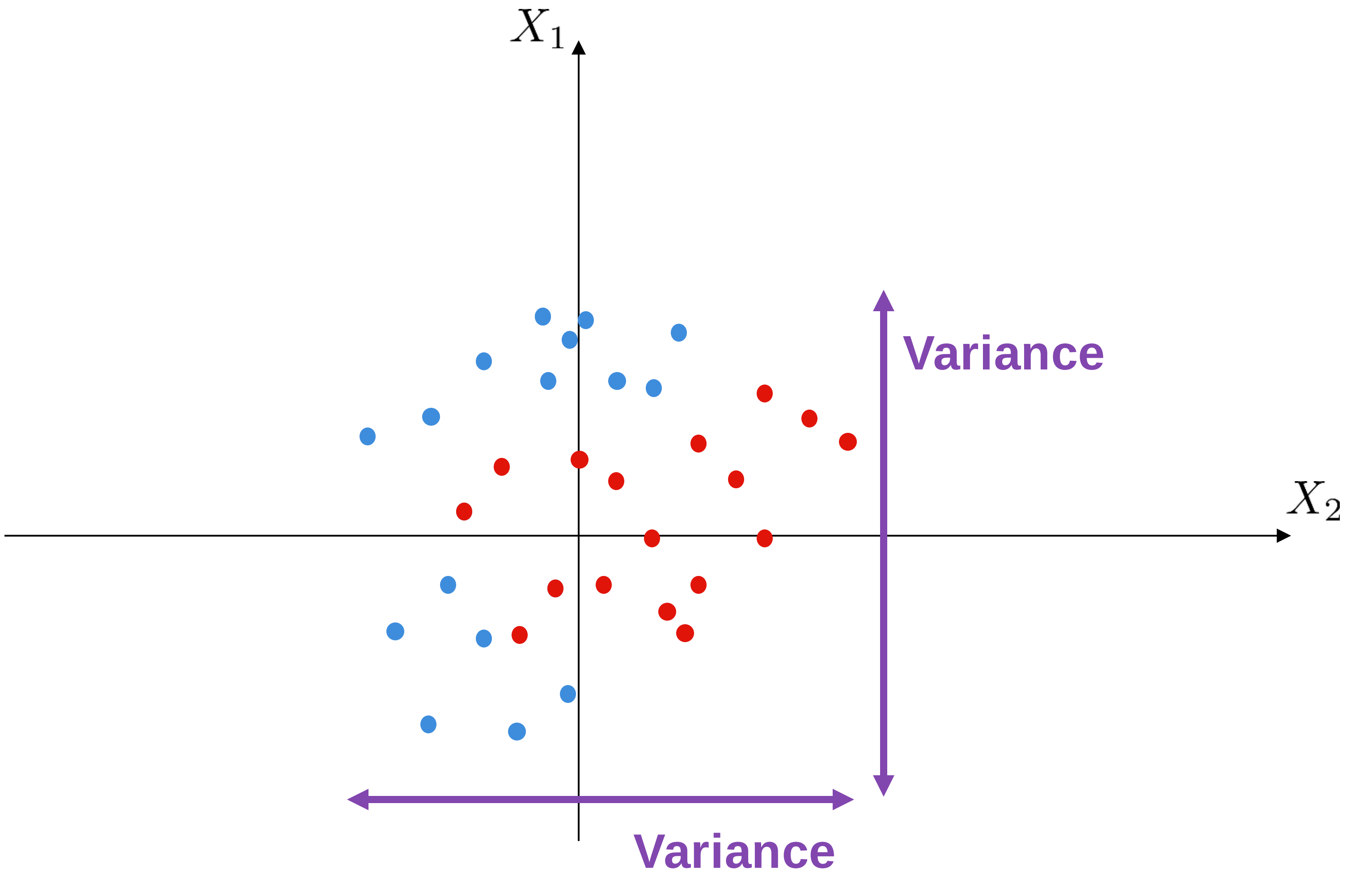
SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



$$x = \frac{x - \mu}{\sigma^2}$$

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



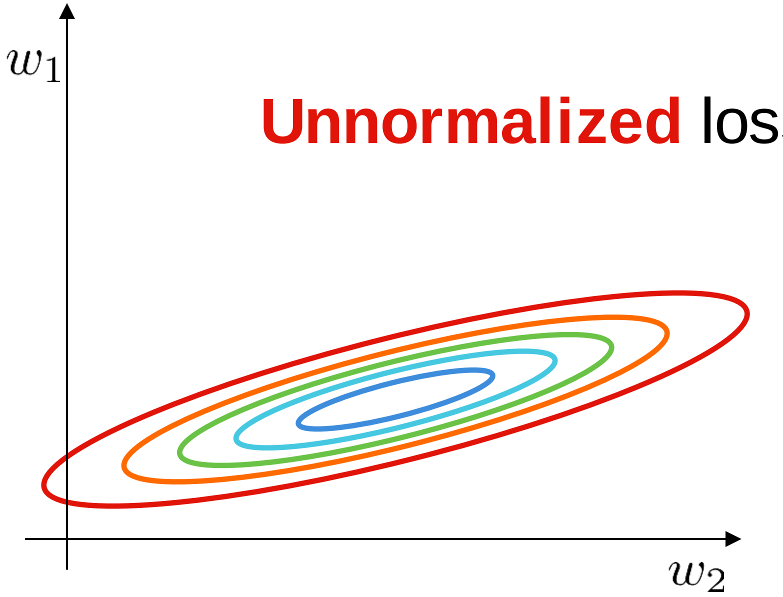
$$x = \frac{x - \mu}{\sigma^2}$$

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

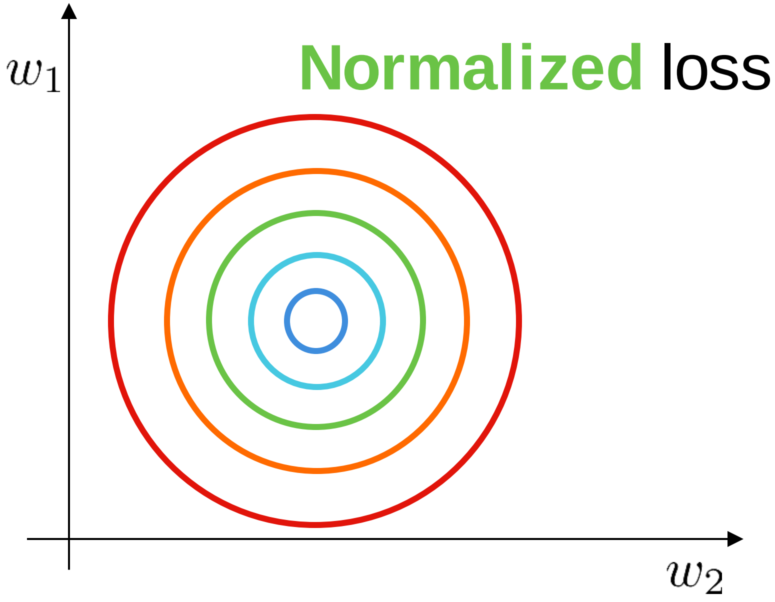
Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Why **input normalization** matters?



$f_1 \in [1, 1000]$
$f_2 \in [0, 1]$

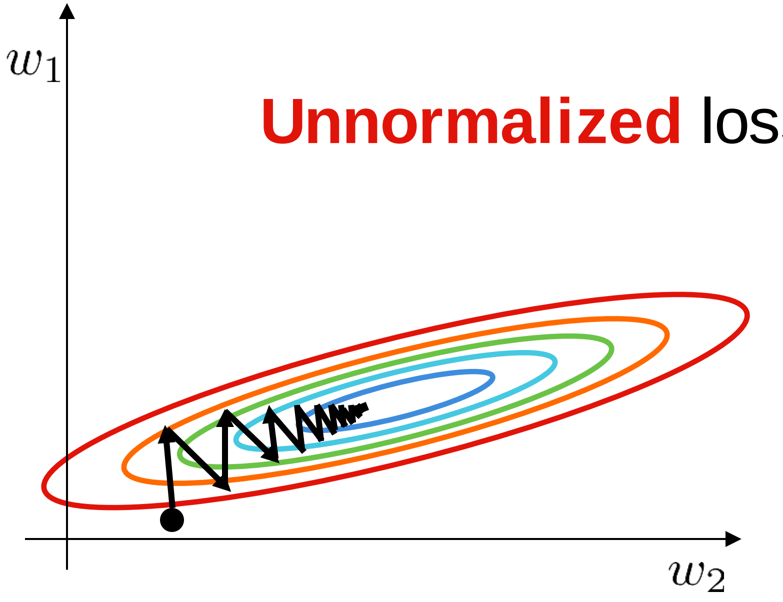


$f_1 \in [-0.5, 0.5]$
$f_2 \in [-0.5, 0.5]$

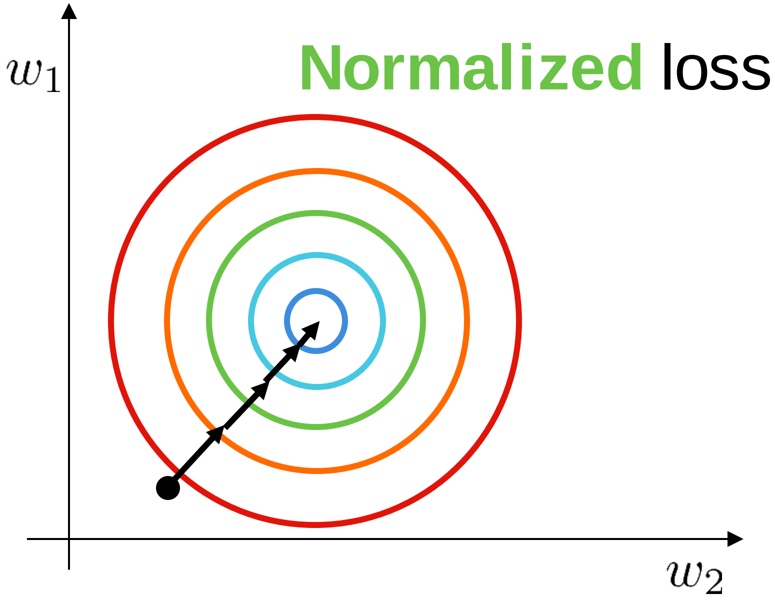
Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Why **input normalization** matters?



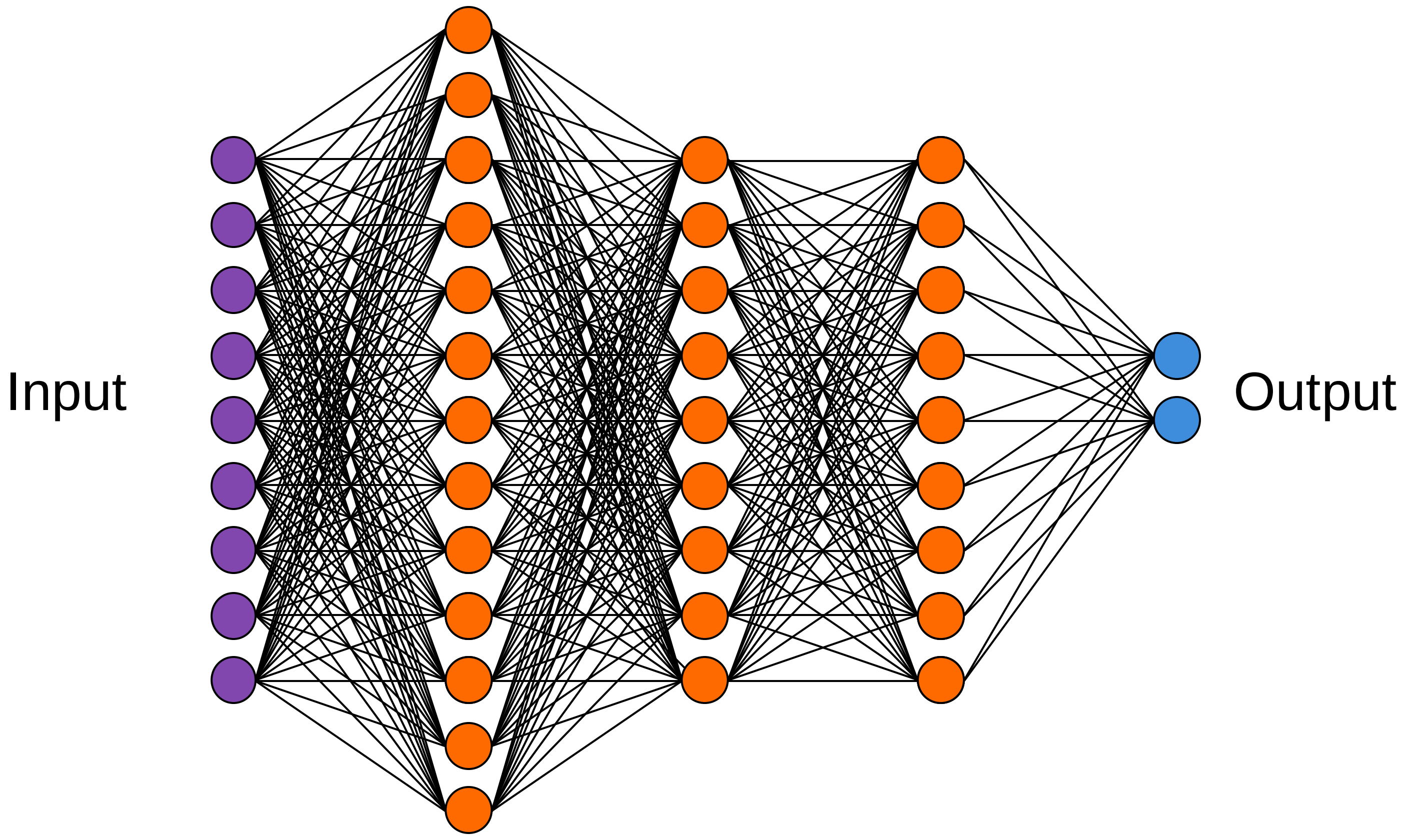
$f_1 \in [1, 1000]$
$f_2 \in [0, 1]$



$f_1 \in [-0.5, 0.5]$
$f_2 \in [-0.5, 0.5]$

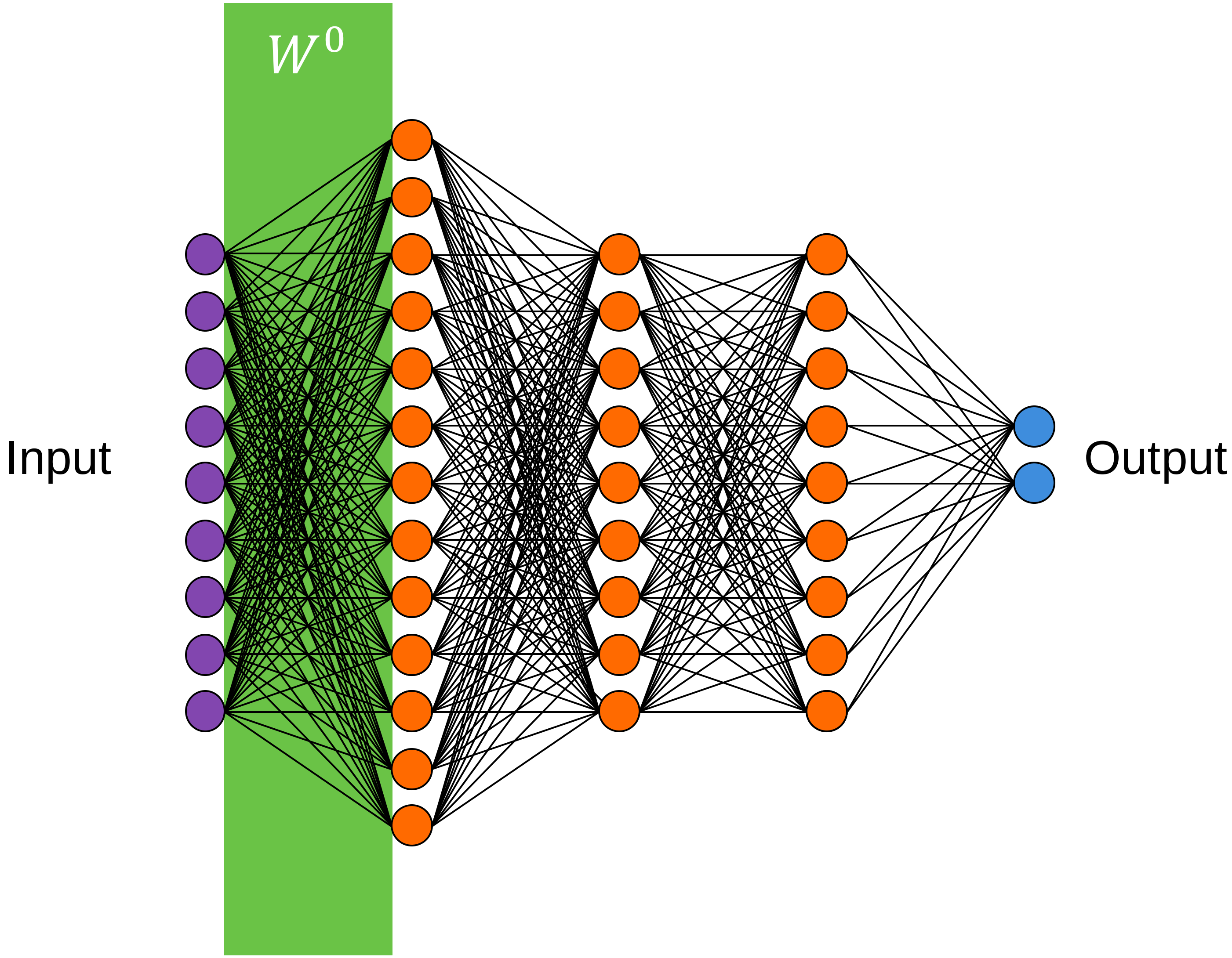
Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



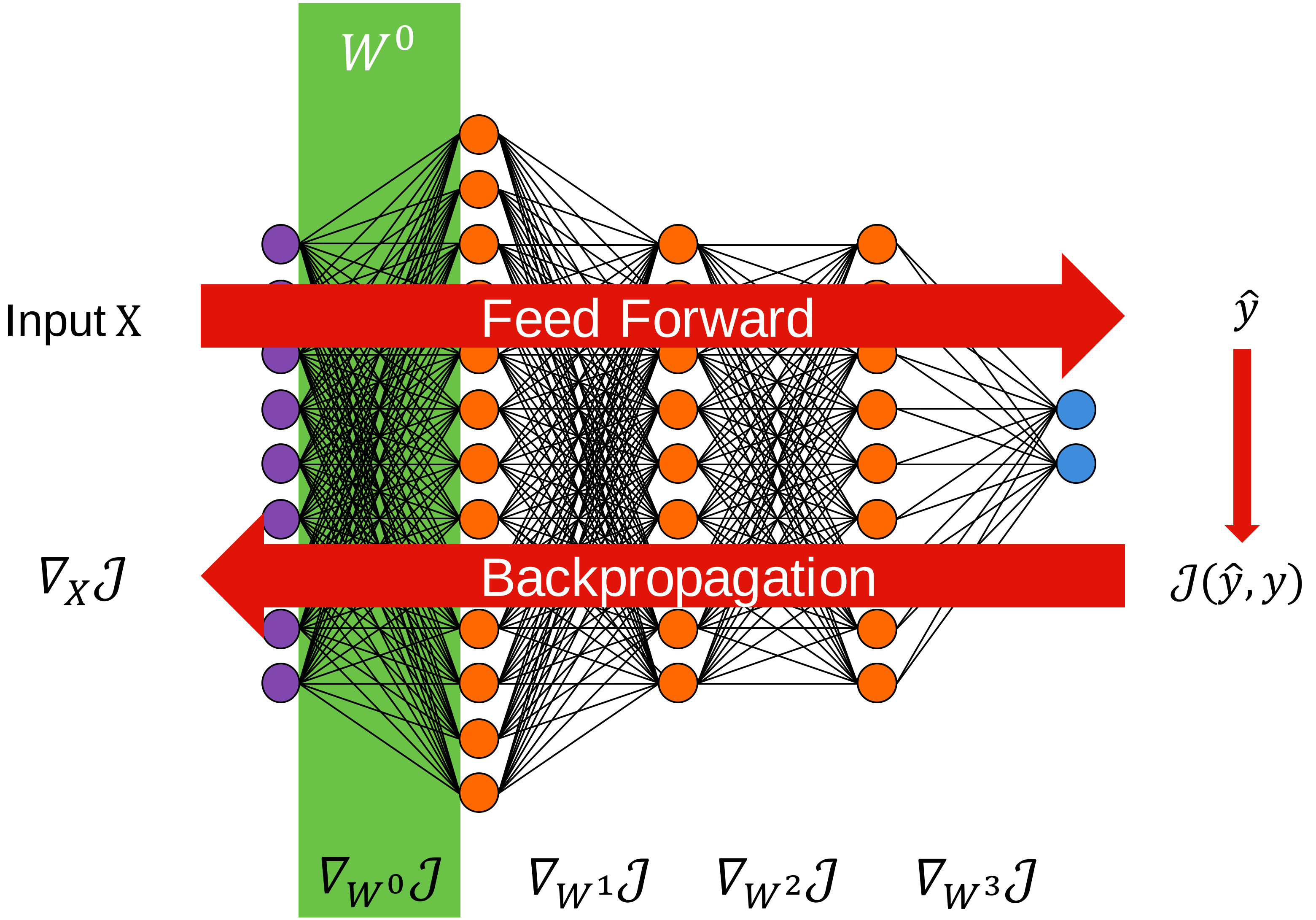
Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



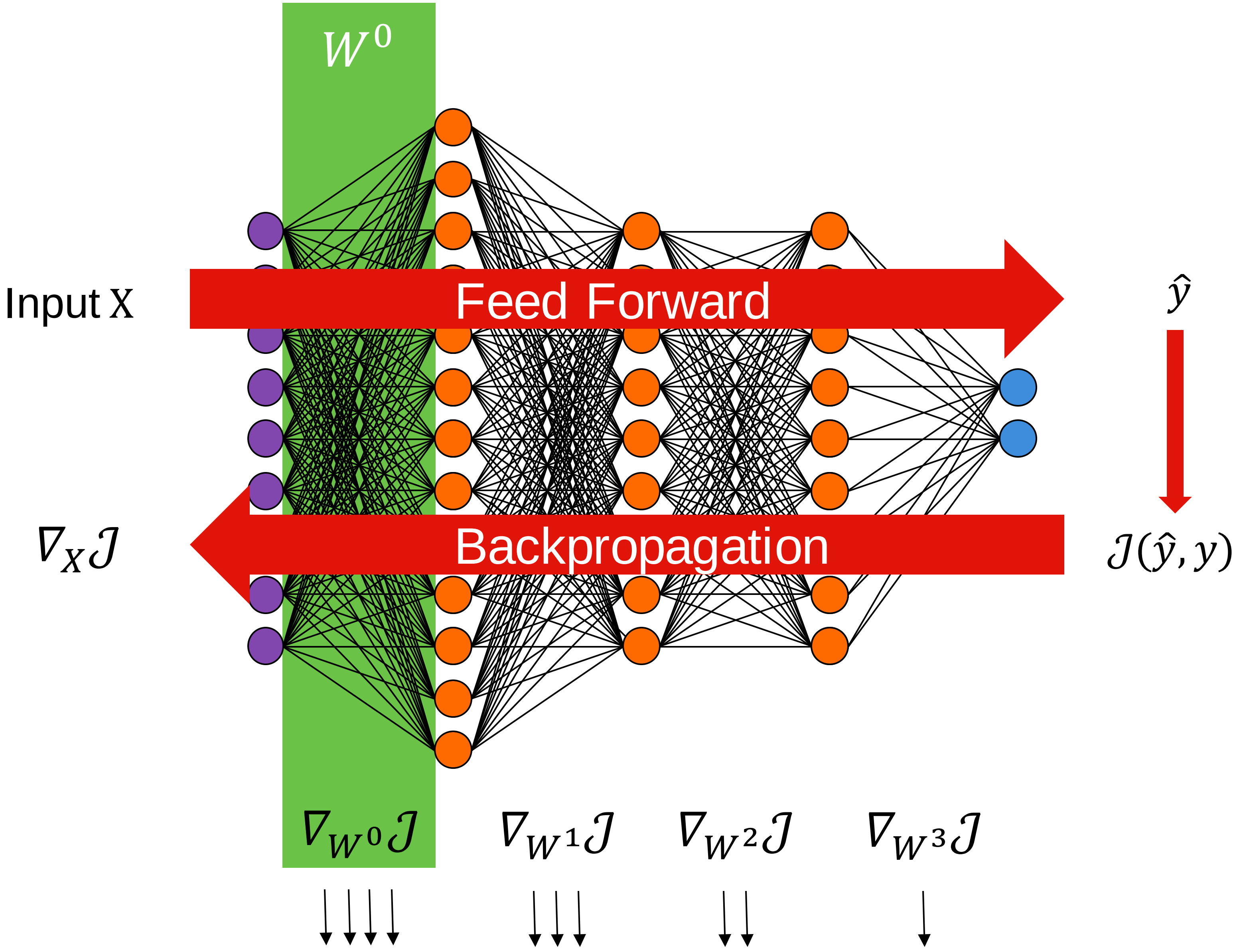
Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



Feedforward Neural Networks

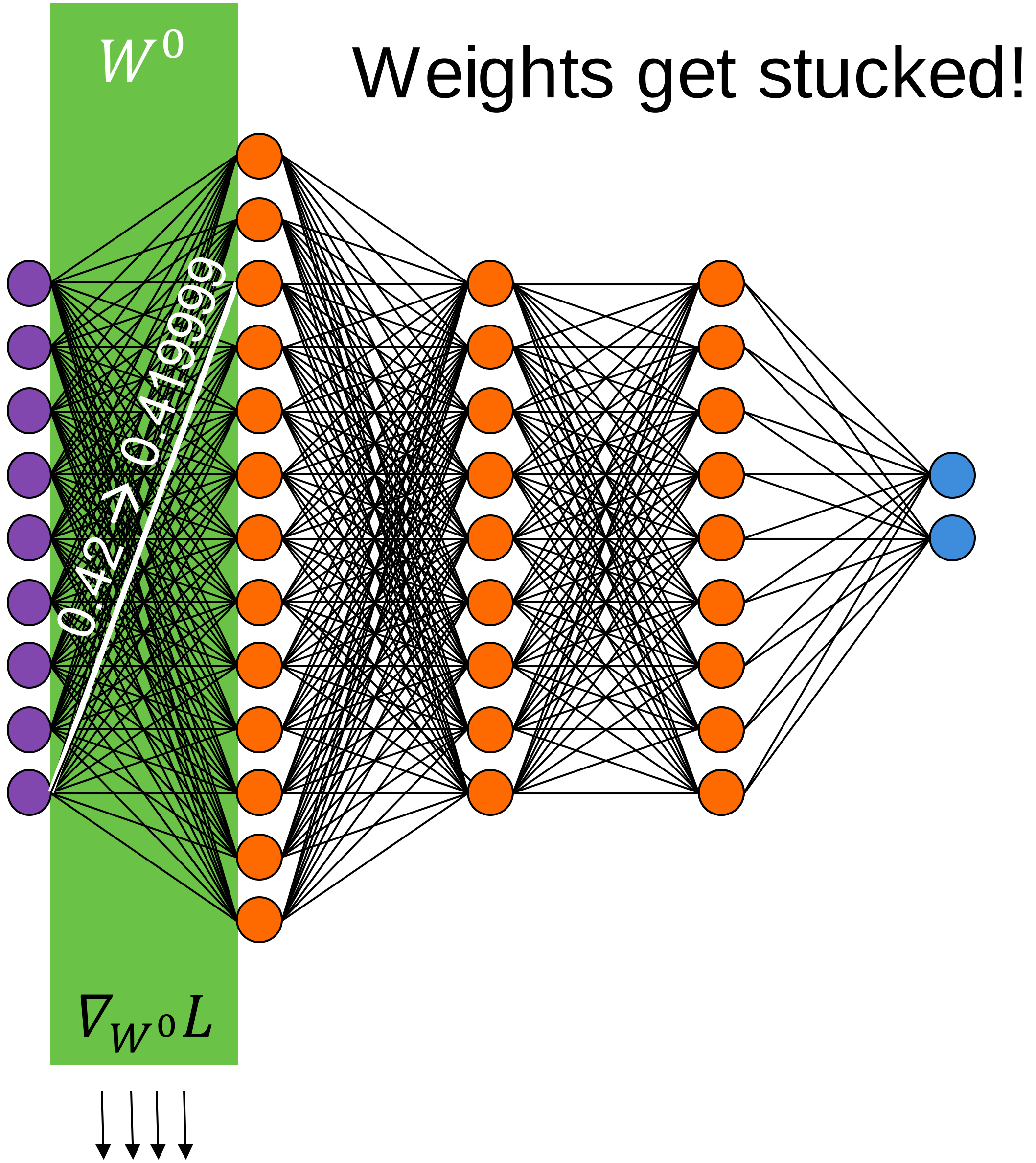
SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Feedforward Neural Networks

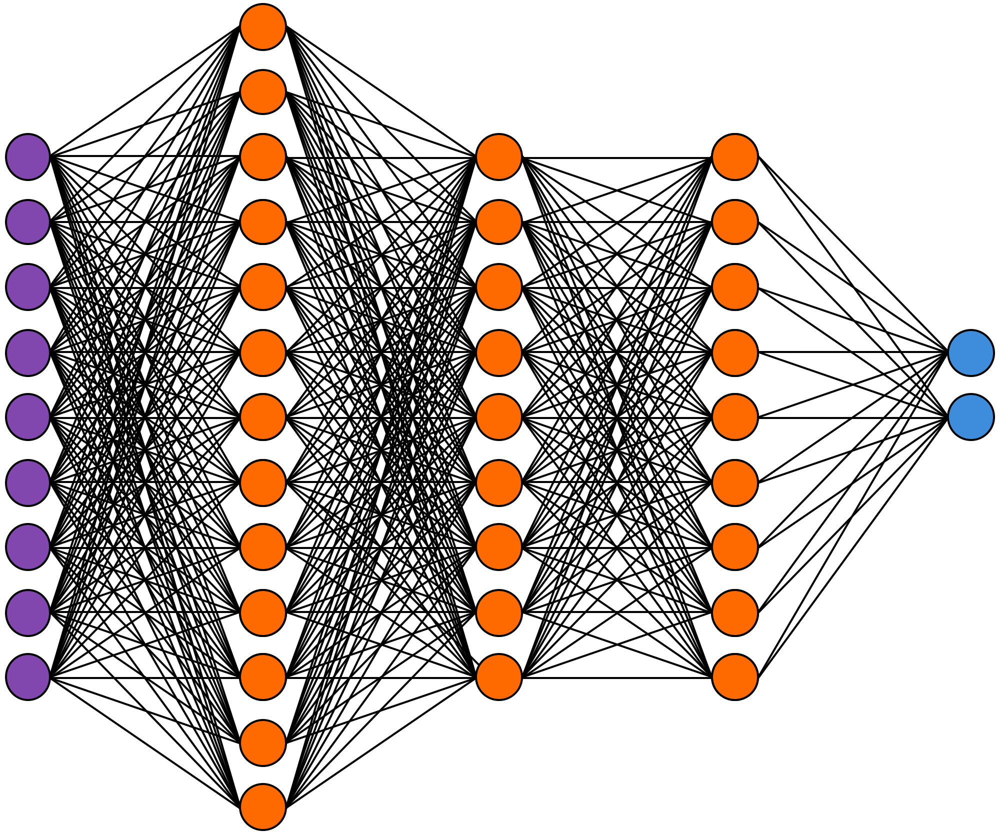
SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Why **gradients** get **smaller and smaller** at each layer on backpropagation?



$\nabla_{W^0}\mathcal{J}$ $\quad\nabla_{W^1}\mathcal{J}$ $\quad\nabla_{W^2}\mathcal{J}$ $\quad\nabla_{W^3}\mathcal{J}$

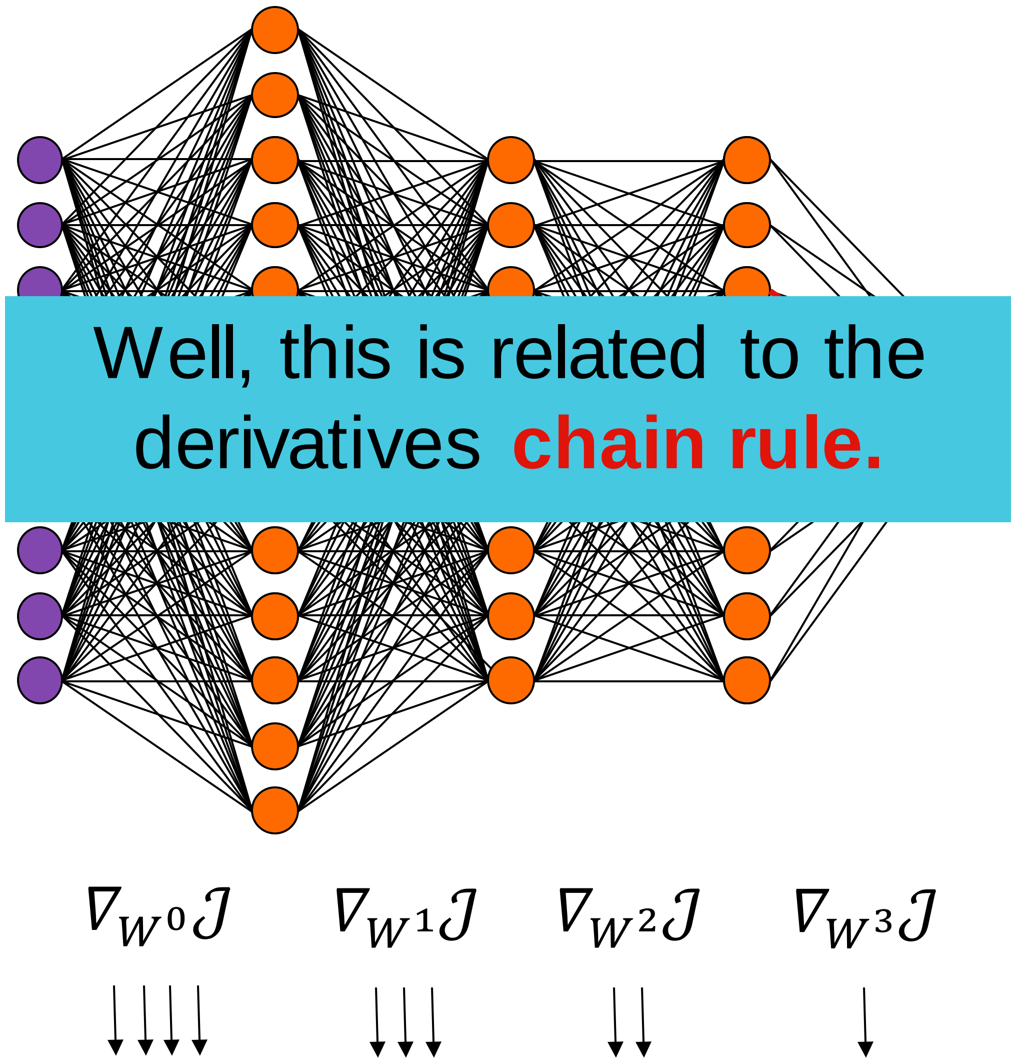
$\downarrow\downarrow\downarrow\downarrow$ $\quad\downarrow\downarrow\downarrow$ $\quad\downarrow\downarrow$ $\quad\downarrow$

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Why **gradients** get **smaller and smaller** at each layer on backpropagation?

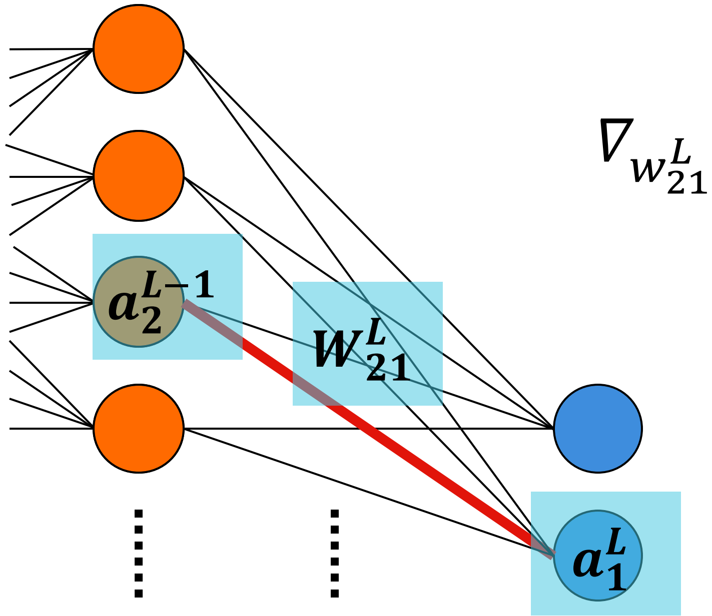


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Why **gradients** get **smaller and smaller** at each layer on backpropagation?



$$\nabla_{w_{21}^{L}} \mathcal{J} = \frac{\partial \mathcal{J}}{\partial w_{21}^{L}} = \frac{\partial \mathcal{J}}{\partial a_1^{L}} \times \frac{\partial a_1^{L}}{\partial z_1^{L}} \times \frac{\partial z_1^{L}}{\partial w_{21}^{L}}$$

$$\nabla_{w_{ij}^{L}} \mathcal{J} = a \times b \times c$$

$$\nabla_{w_{ij}^{L-1}} \mathcal{J} = a \times b \times c \times d \times e \times f$$

$$\nabla_{w_{ij}^{L-2}} \mathcal{J} = a \times b \times c \times d \times e \times f \times g \cdots$$

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Why **gradients** get **smaller and smaller** at each layer on backpropagation?

$$\nabla_{w_{ij}^{L}} \mathcal{J} = a \times b \times c$$

$$\nabla_{w_{ij}^{L-1}} \mathcal{J} = a \times b \times c \times d \times e \times f$$

$$\nabla_{w_{ij}^{L-2}} \mathcal{J} = a \times b \times c \times d \times e \times f \times g \cdots$$

Terms values < 1.0

+

#Multypling terms $\uparrow\uparrow\uparrow$

=

$\nabla_{W^0} \mathcal{J}$ $\downarrow\downarrow\downarrow\downarrow$ $\nabla_{W^1} \mathcal{J}$ $\downarrow\downarrow\downarrow$ $\nabla_{W^2} \mathcal{J}$ $\downarrow\downarrow$ $\nabla_{W^3} \mathcal{J}$ $\downarrow$

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

What about **exploding gradients**?

$$\nabla_{w_{ij}^{L}} \mathcal{J} = a \times b \times c$$

$$\nabla_{w_{ij}^{L-1}} \mathcal{J} = a \times b \times c \times d \times e \times f$$

$$\nabla_{w_{ij}^{L-2}} \mathcal{J} = a \times b \times c \times d \times e \times f \times g \cdots$$

Terms values > 1.0

+

#Multypling terms $\uparrow\uparrow\uparrow$

=

$\nabla_{W^0} \mathcal{J}$ $\uparrow\uparrow\uparrow\uparrow$ $\quad$ $\nabla_{W^1} \mathcal{J}$ $\uparrow\uparrow\uparrow$ $\quad$ $\nabla_{W^2} \mathcal{J}$ $\uparrow\uparrow$ $\quad$ $\nabla_{W^3} \mathcal{J}$ $\uparrow$

Feedforward Neural Networks

SGD, Epochs, Batches and Steps

Activation functions

SGD learning rate

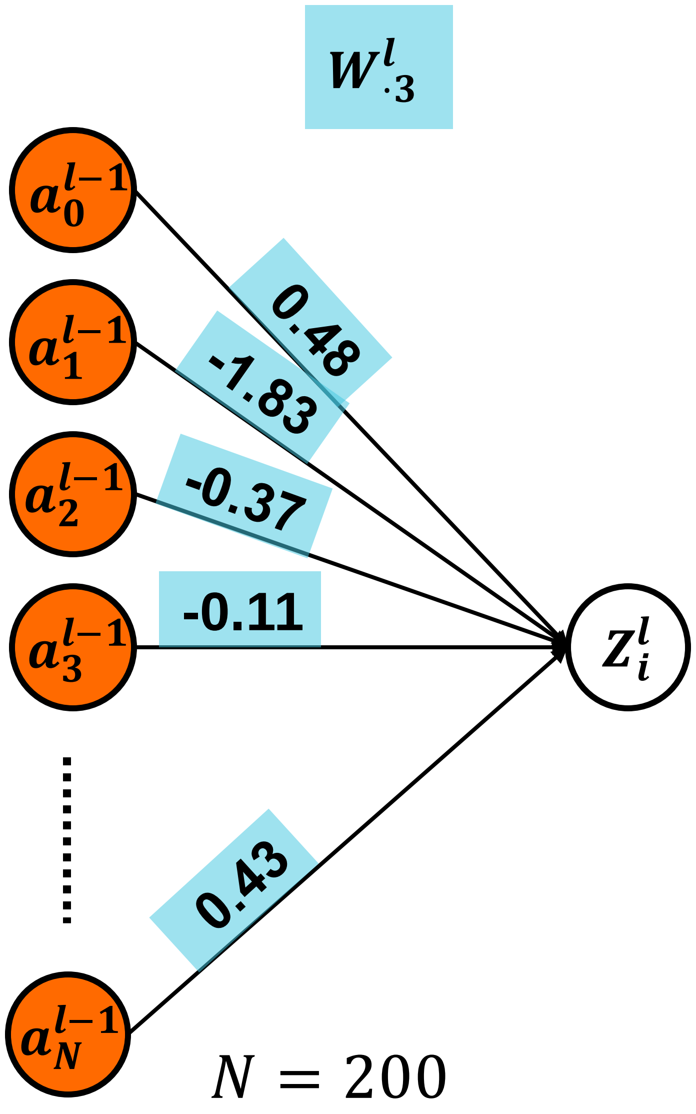
Other optimization methods

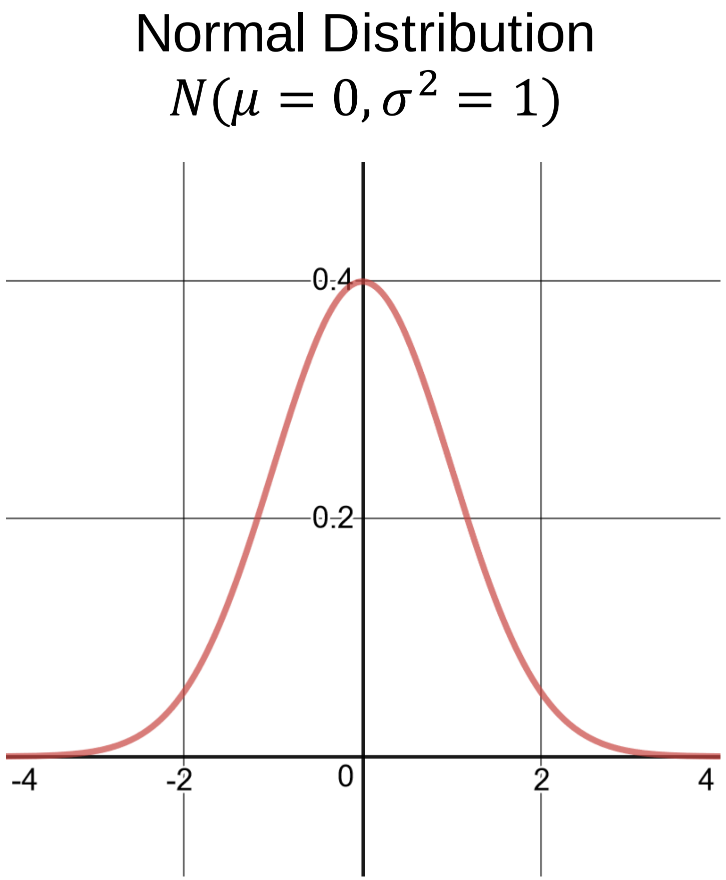
Regularization

Normalizing inputs

Vanishing/Exploding Gradients

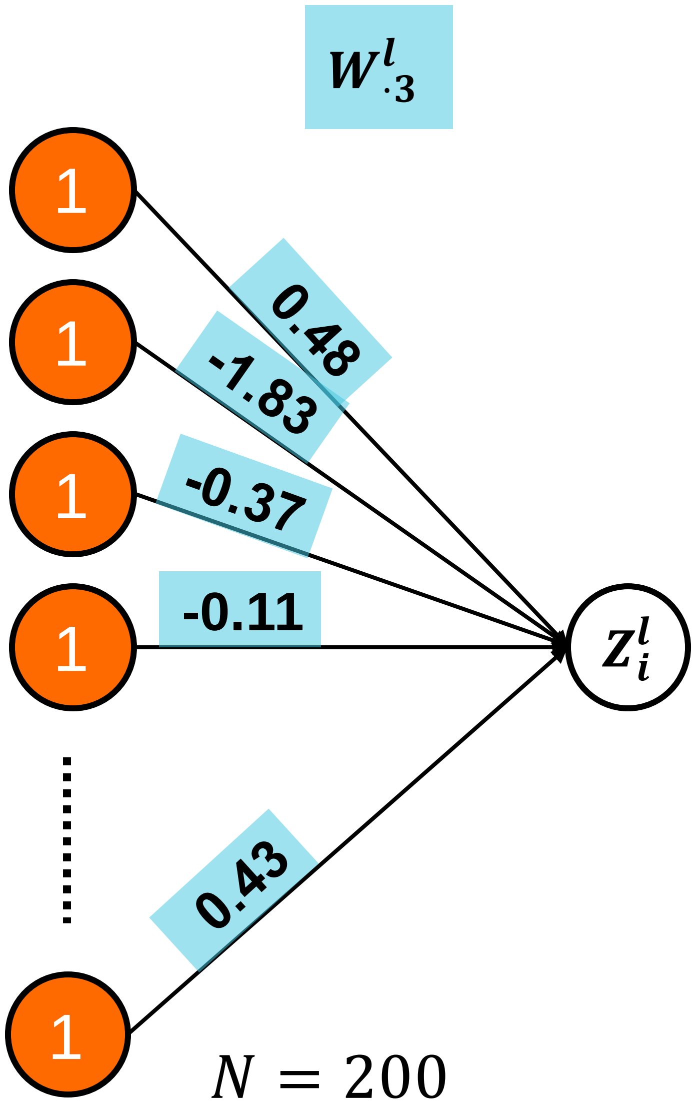
Weights initialization



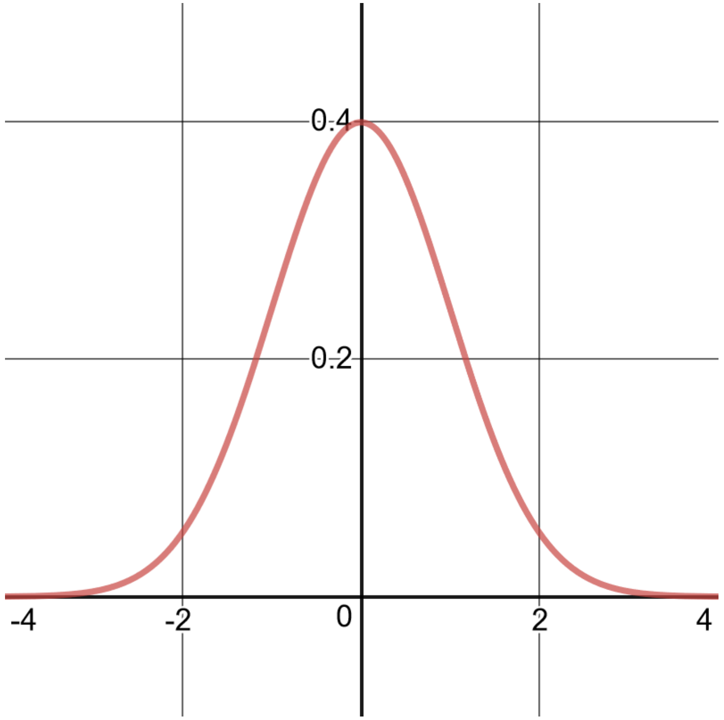


Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

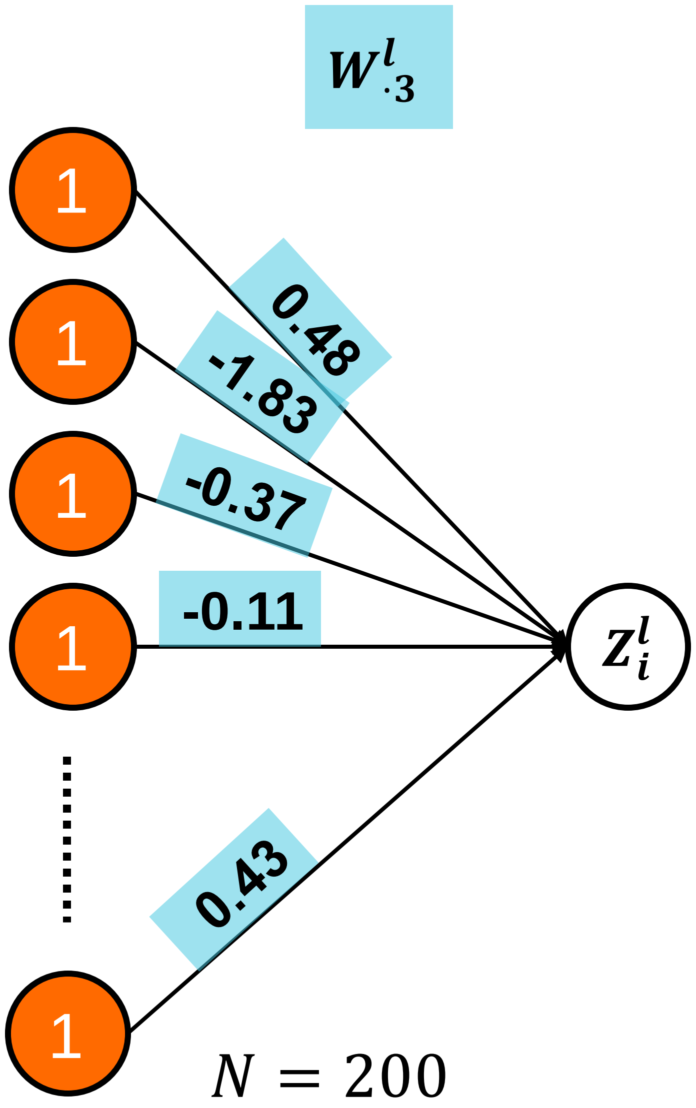


Normal Distribution
$N(\mu = 0, \sigma^2 = 1)$



Feedforward Neural Networks

- SGD, Epochs, Batches and Steps
- Activation functions
- SGD learning rate
- Other optimization methods
- Regularization
- Normalizing inputs
- Vanishing/Exploding Gradients
- **Weights initialization**



Sum of **independent random variables** that are **normally distributed**:

$$X \sim N(\mu_X, {\sigma^2}_X)$$

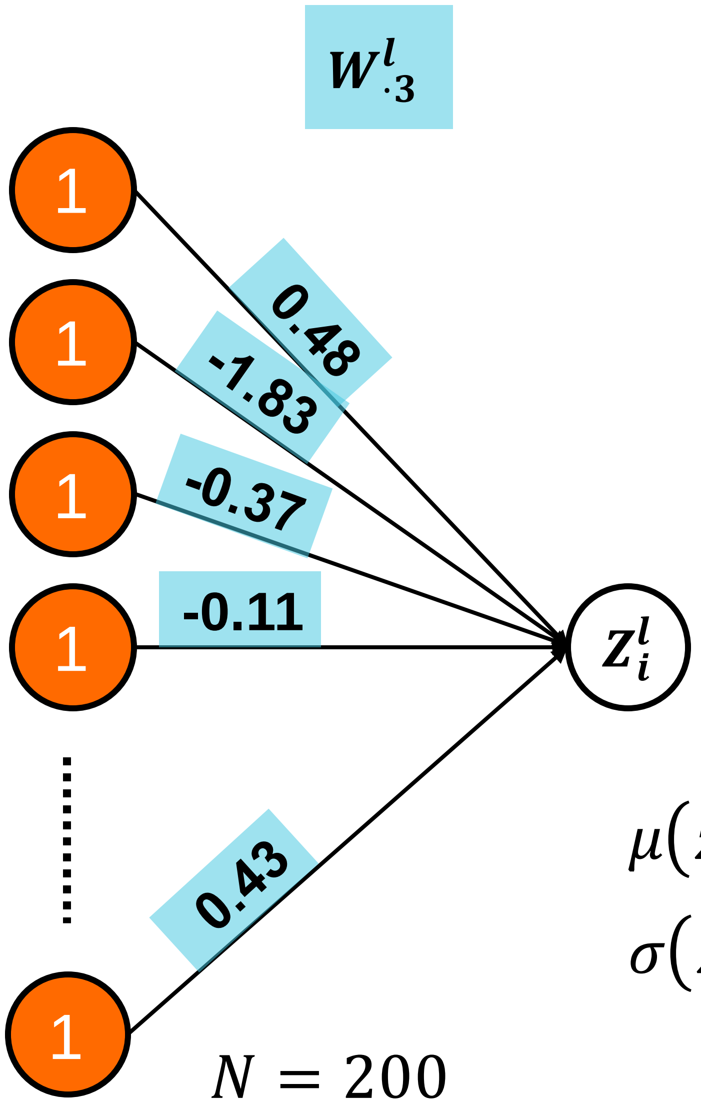
$$Y \sim N(\mu_Y, {\sigma^2}_Y)$$

$$Z = X + Y$$

$$Z \sim N(\mu_X + \mu_Y, {\sigma^2}_X + {\sigma^2}_Y)$$

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



Sum of **independent random variables** that are **normally distributed**:

$$X \sim N(\mu_X, \sigma^2{}_X)$$
$$Y \sim N(\mu_Y, \sigma^2{}_Y)$$
$$Z = X + Y$$

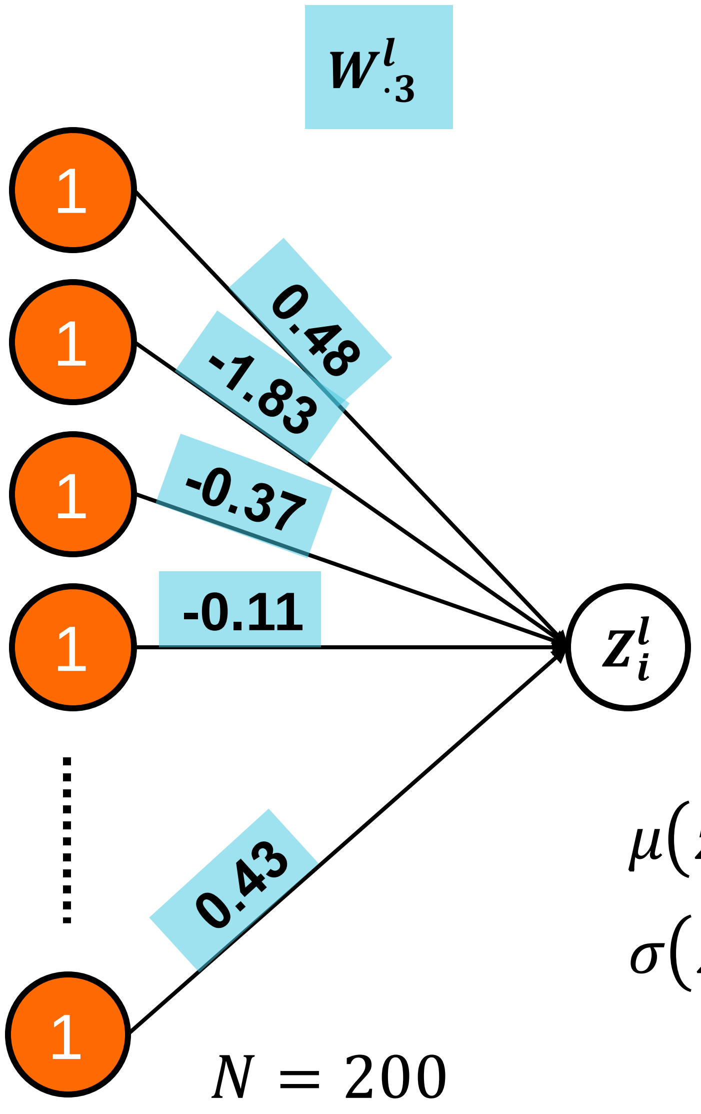
$$Z \sim N(\mu_X + \mu_Y, \sigma^2{}_X + \sigma^2{}_Y)$$

$$\mu(z_i^l) = 0.0$$
$$\sigma(z_i^l) = \sqrt{200} \approx 14.14$$

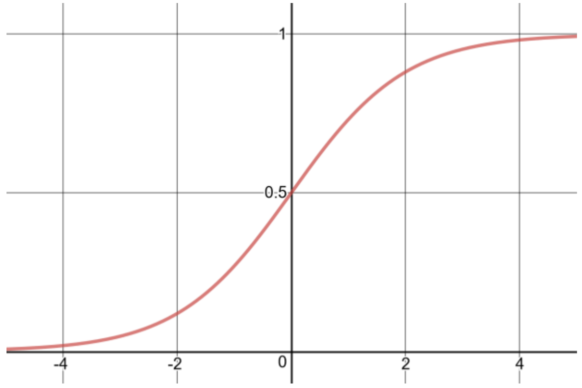
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



$$sigmoid(x) = \frac{e^x}{e^x + 1}$$



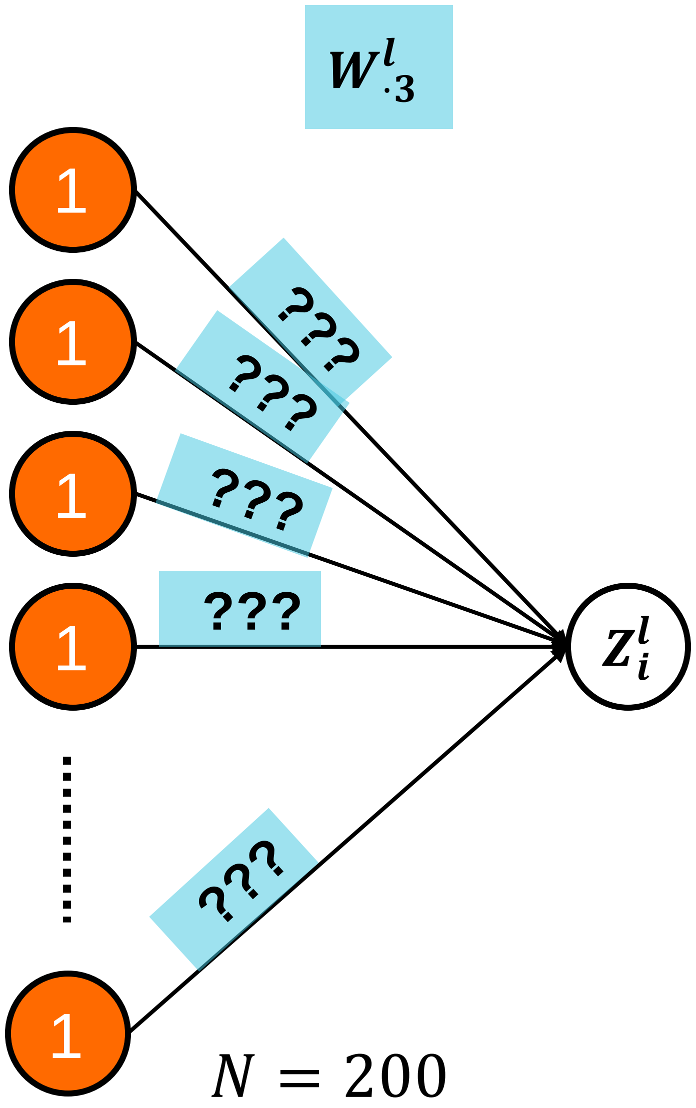
$$\mu(z_i^l) = 0.0$$

$$\sigma(z_i^l) = \sqrt{200} \approx 14.14$$

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Feedforward Neural Networks

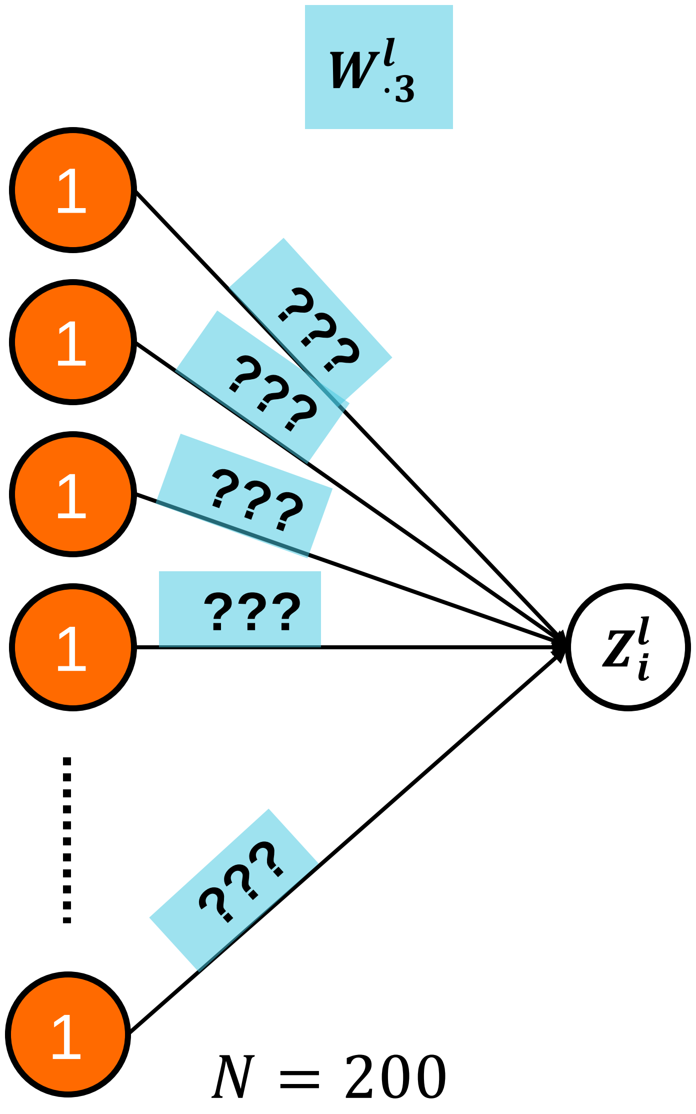
SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



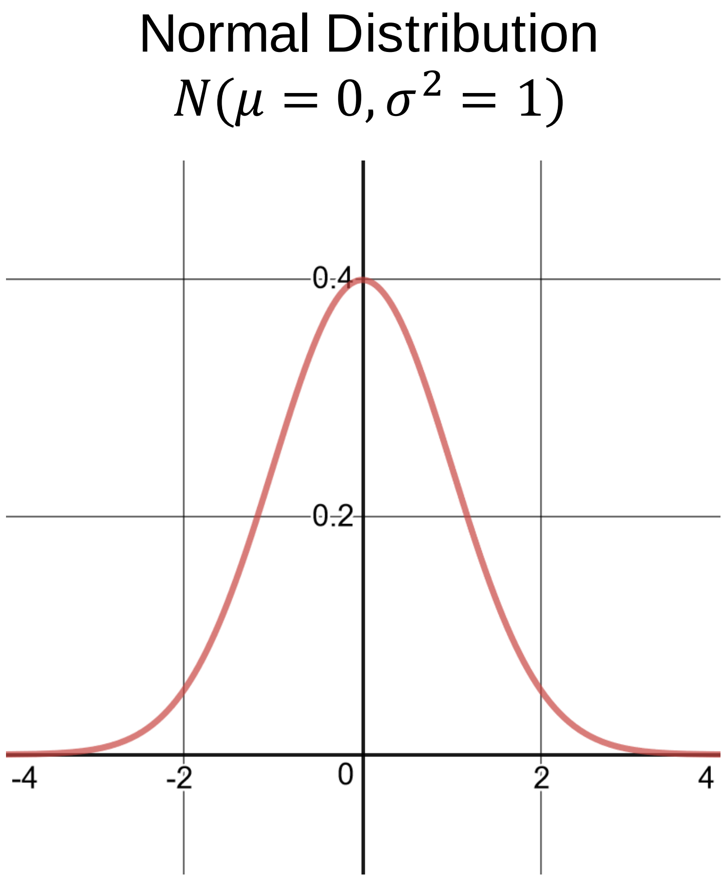
Alternative weights initializations?

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

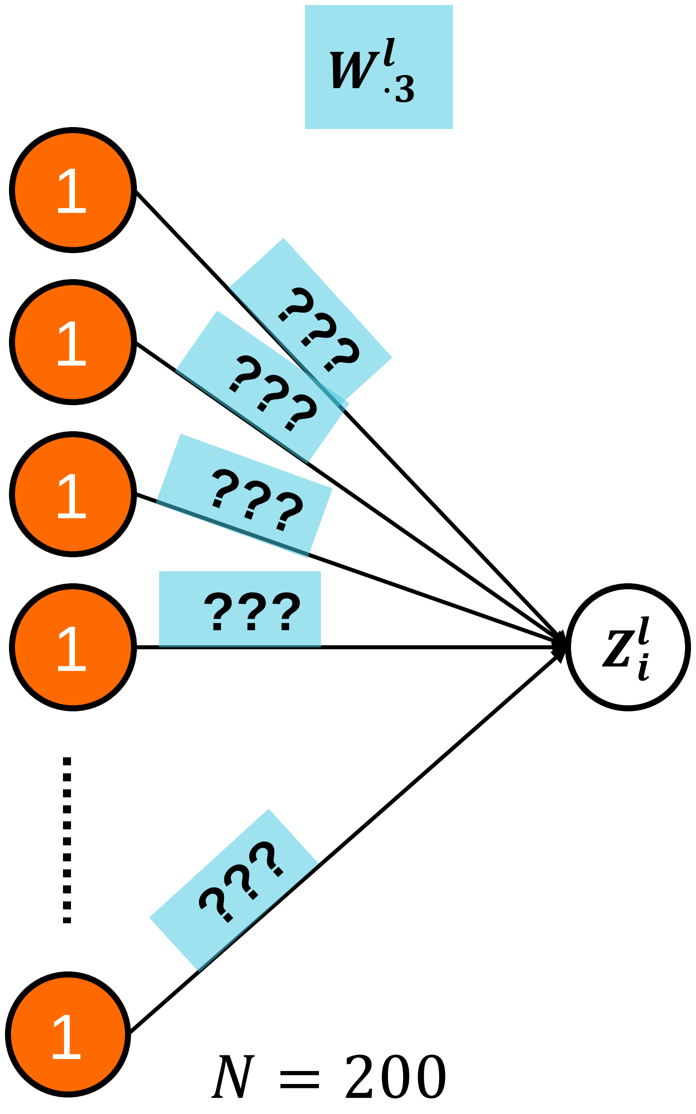


Truncated Normal Initialization



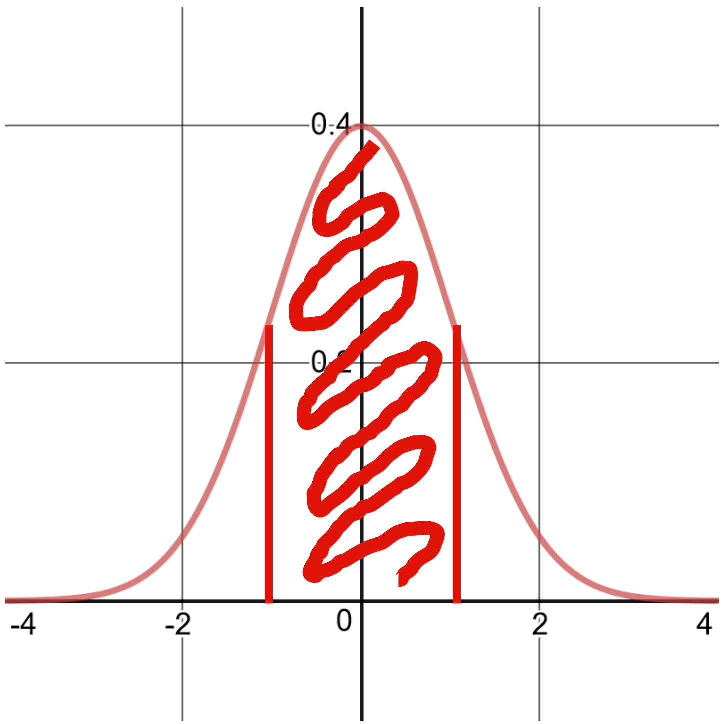
Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



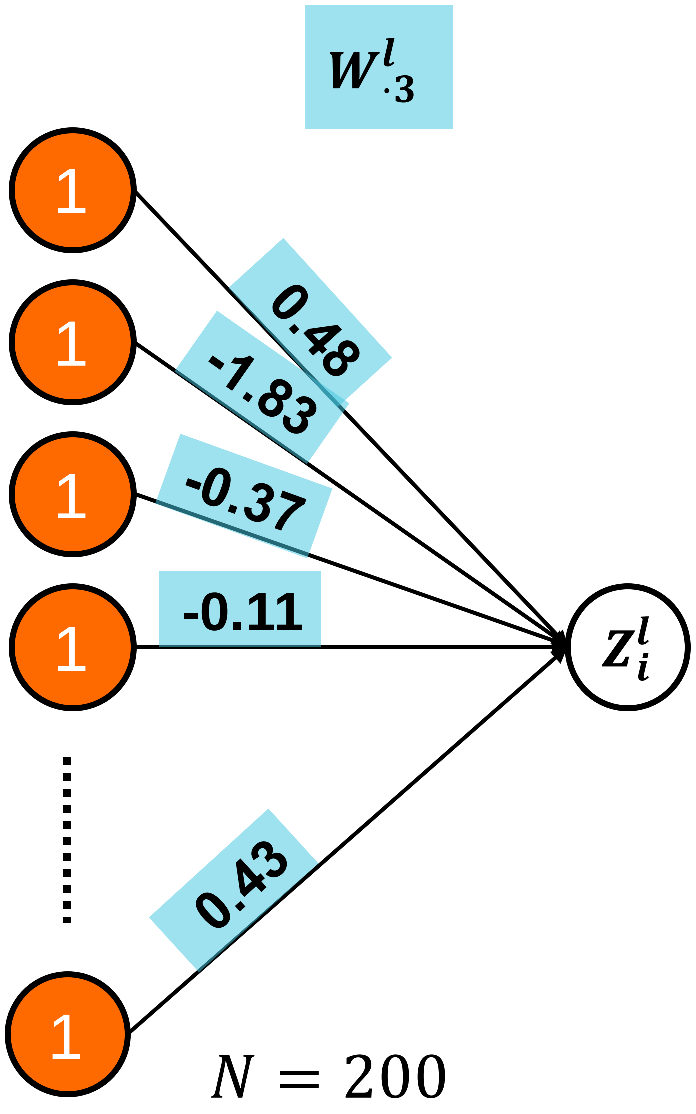
Truncated Normal Initialization

Normal Distribution
$N(\mu = 0, \sigma^2 = 1)$

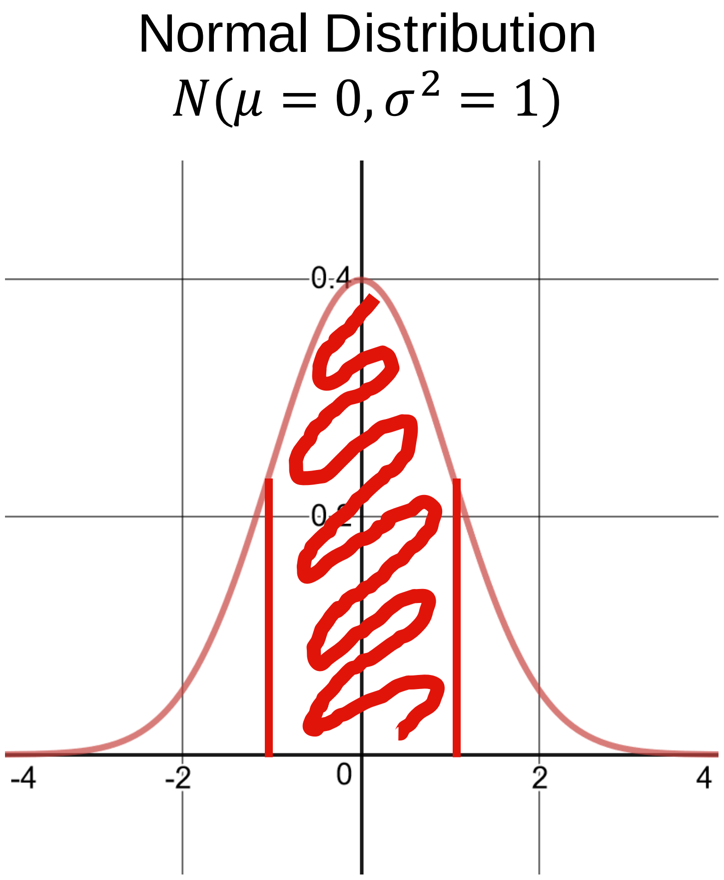


Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

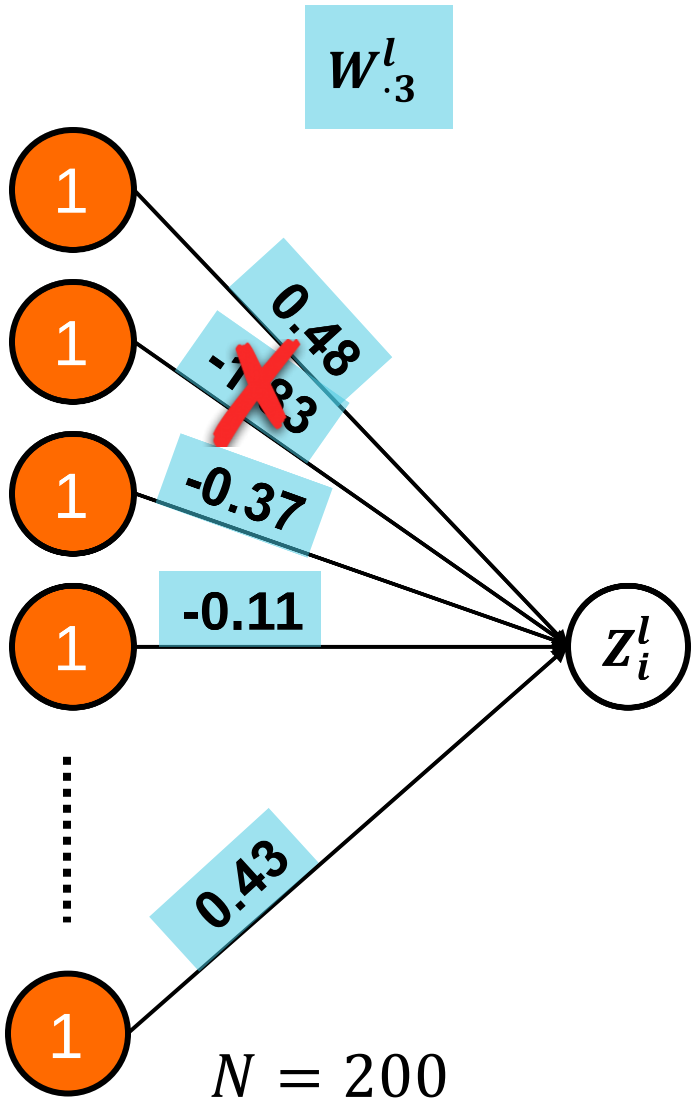


Truncated Normal Initialization



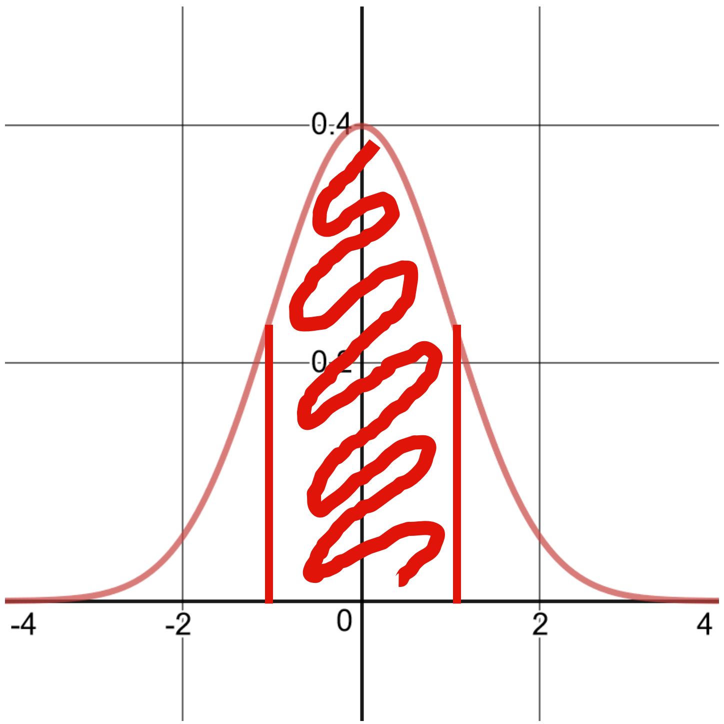
Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



Truncated Normal Initialization

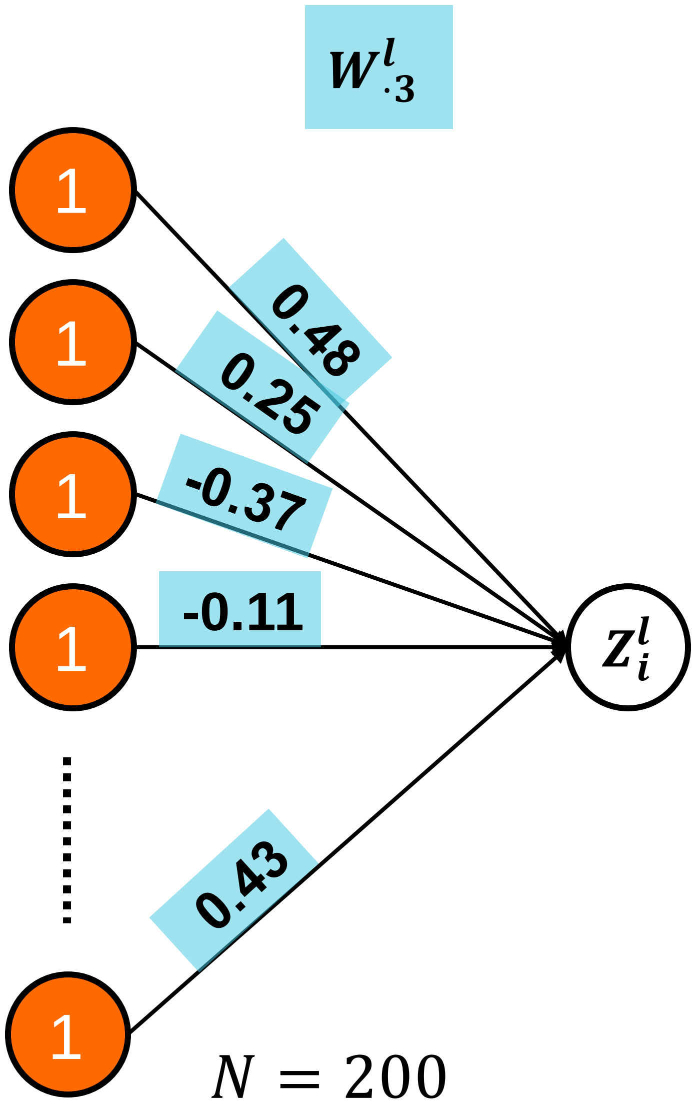
Normal Distribution
$N(\mu = 0, \sigma^2 = 1)$



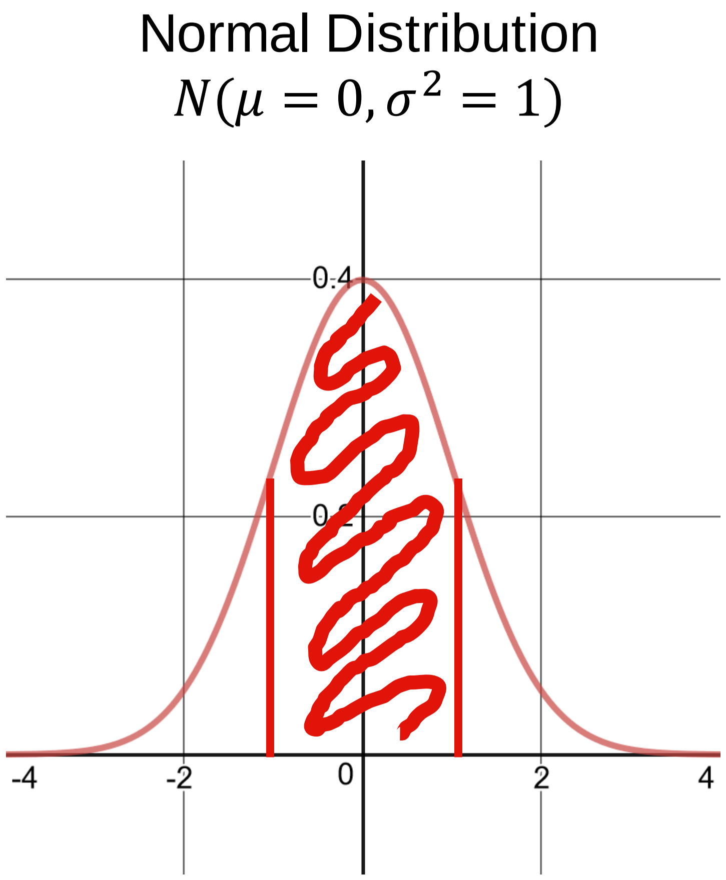
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Feedforward Neural Networks

SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization



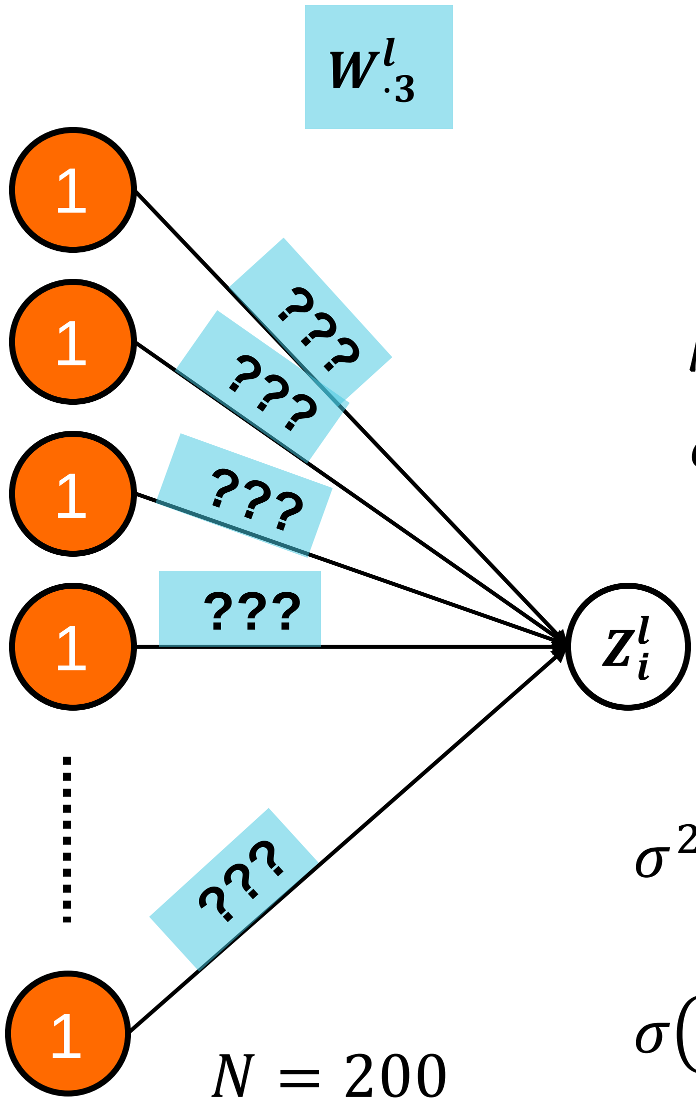
Truncated Normal Initialization



Feedforward Neural Networks

- SGD, Epochs, Batches and Steps
- Activation functions
- SGD learning rate
- Other optimization methods
- Regularization
- Normalizing inputs
- Vanishing/Exploding Gradients
- **Weights initialization**

Xavier / Glorot Initialization



$$\mu(z^l_i) = 0.0$$

$$\sigma(z^l_i) = 1.0$$

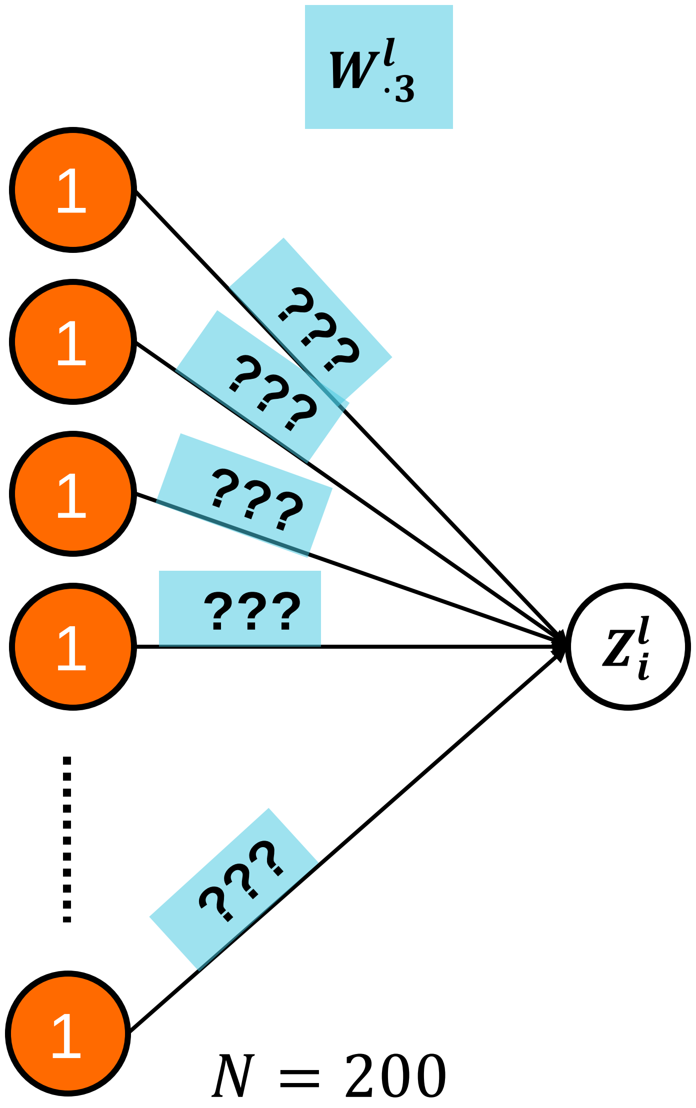
$$\sigma^2(W^l_{\cdot 3}) = \frac{1}{N} = 0.005$$

$$\sigma(W^l_{\cdot 3}) = \sqrt{1/N} \approx 0.0707$$

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

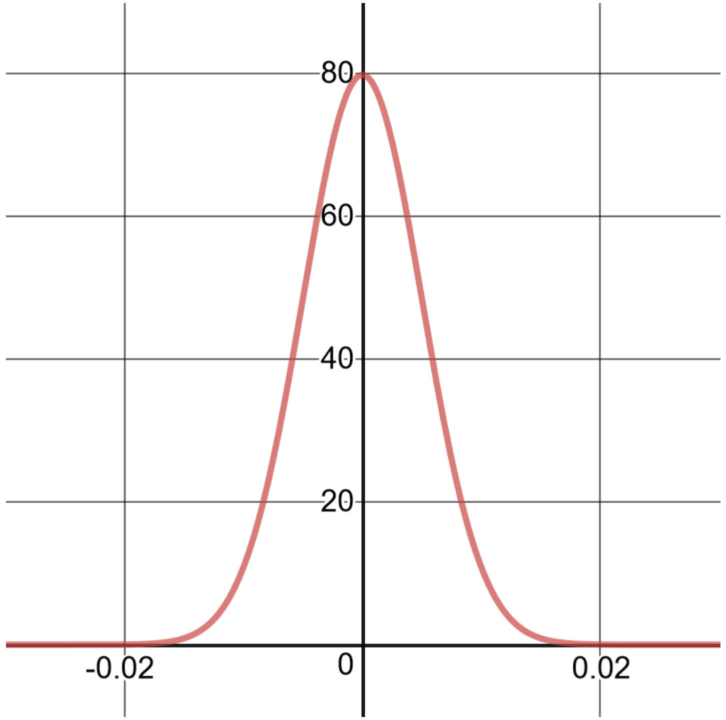
Feedforward Neural Networks

- SGD, Epochs, Batches and Steps
- Activation functions
- SGD learning rate
- Other optimization methods
- Regularization
- Normalizing inputs
- Vanishing/Exploding Gradients
- **Weights initialization**

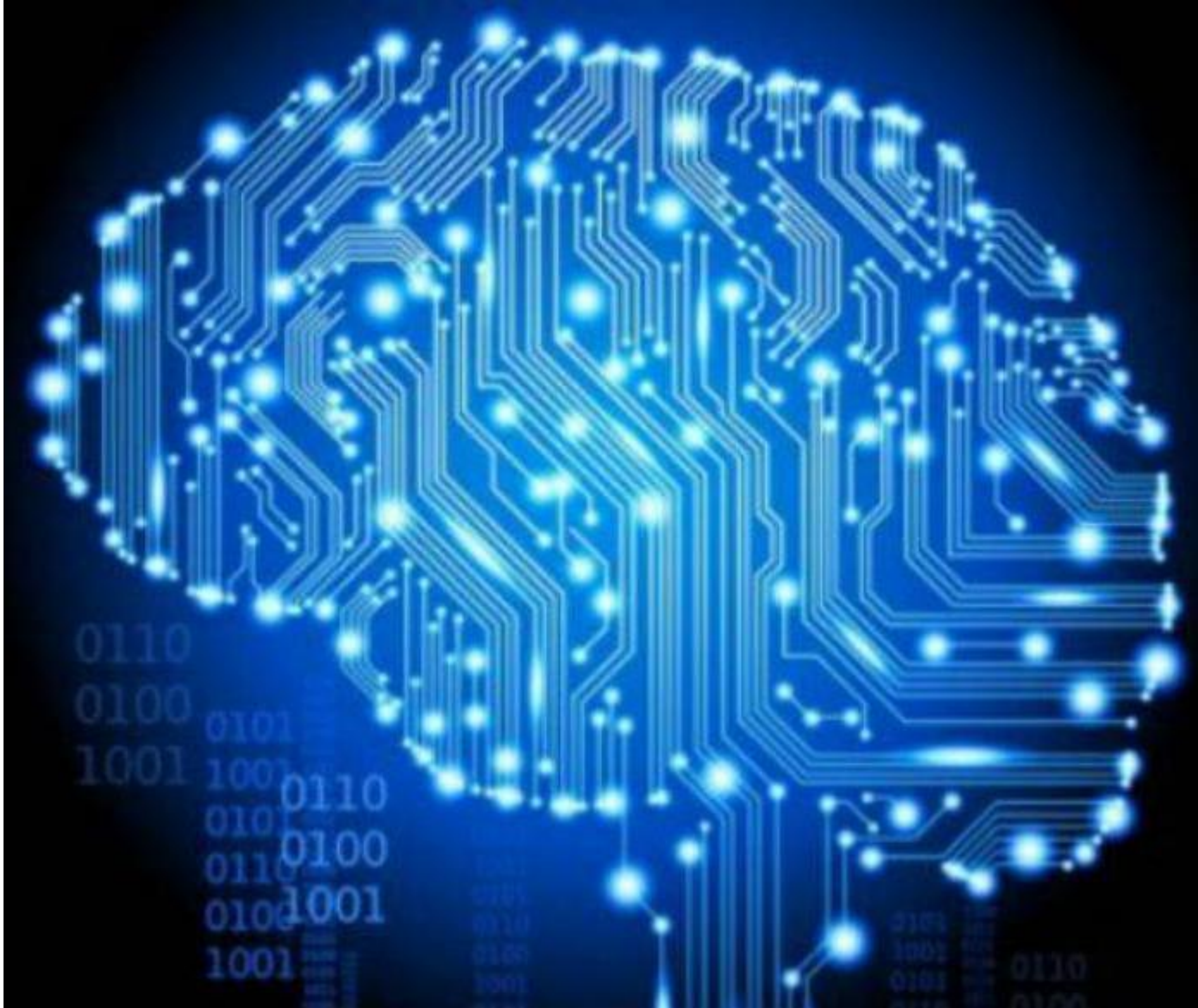


Xavier / Glorot Initialization

Normal Distribution
$N(\mu = 0, \sigma^2 = 0.005)$



Practical Aspects

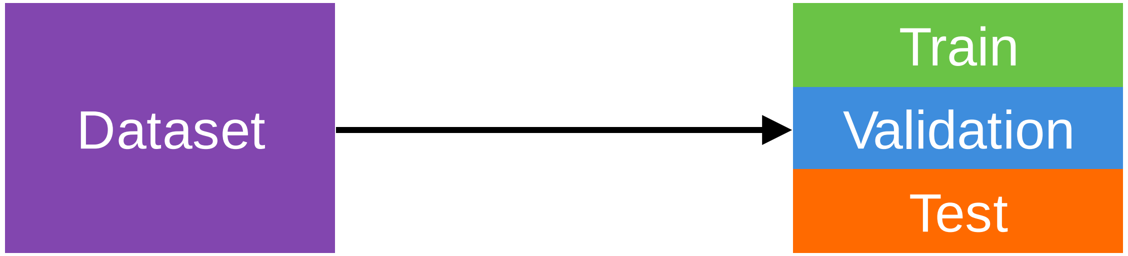


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Practical Aspects

Train / Validation / Test

Workflow



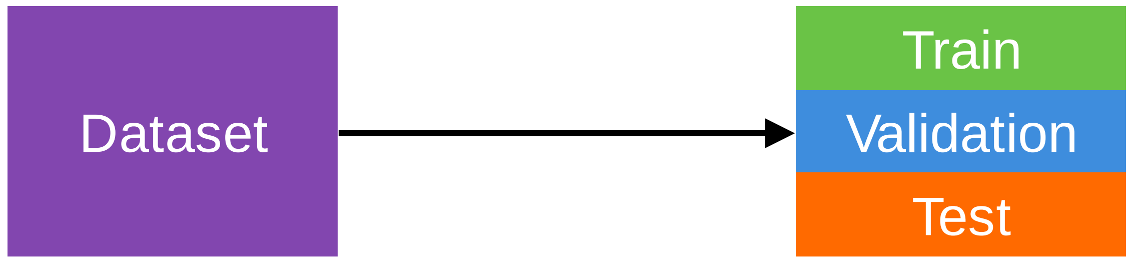
Train	Set used to **train & control bias**
Validation	Set used to **control variance**
Test	Set used to **estimate** the **generalization error** of the final model

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Practical Aspects

Train / Validation / Test

Workflow



Train	Set used to **train & control bias**
Validation	Set used to **control variance**
Test	Set used to **estimate** the **generalization error** of the final model

OPTIONAL

Practical Aspects

Train / Validation / Test

Workflow

Dataset → Train / Test

Train — Set used to **train & control bias**

Test — Set used to **control variance**

OPTIONAL

Test — Set used to **estimate** the **generalization error** of the final model

Practical Aspects

Train / Validation / Test

Workflow

Machine Learning

80%	10%	10%

How many **images** do we **need** for each set?

Train	**As many as possible**
Validation	The **minimum** amount to appropriately **represent each class**
Test	The **minimum** amount to appropriately **represent each class**

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Practical Aspects

Train / Validation / Test

Workflow

Machine Learning

Typical dataset size: 1.000 - 30.000

Deep Learning

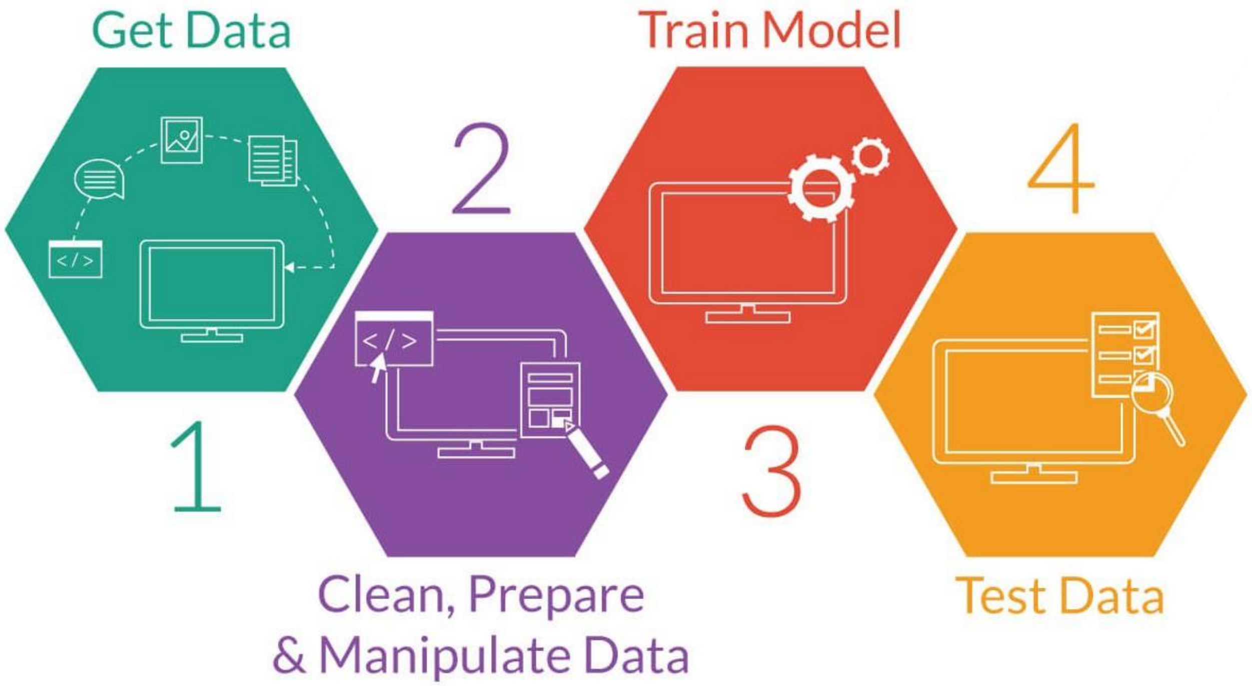
Typical dataset size: 30.000 - 10.000.000

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Practical Aspects

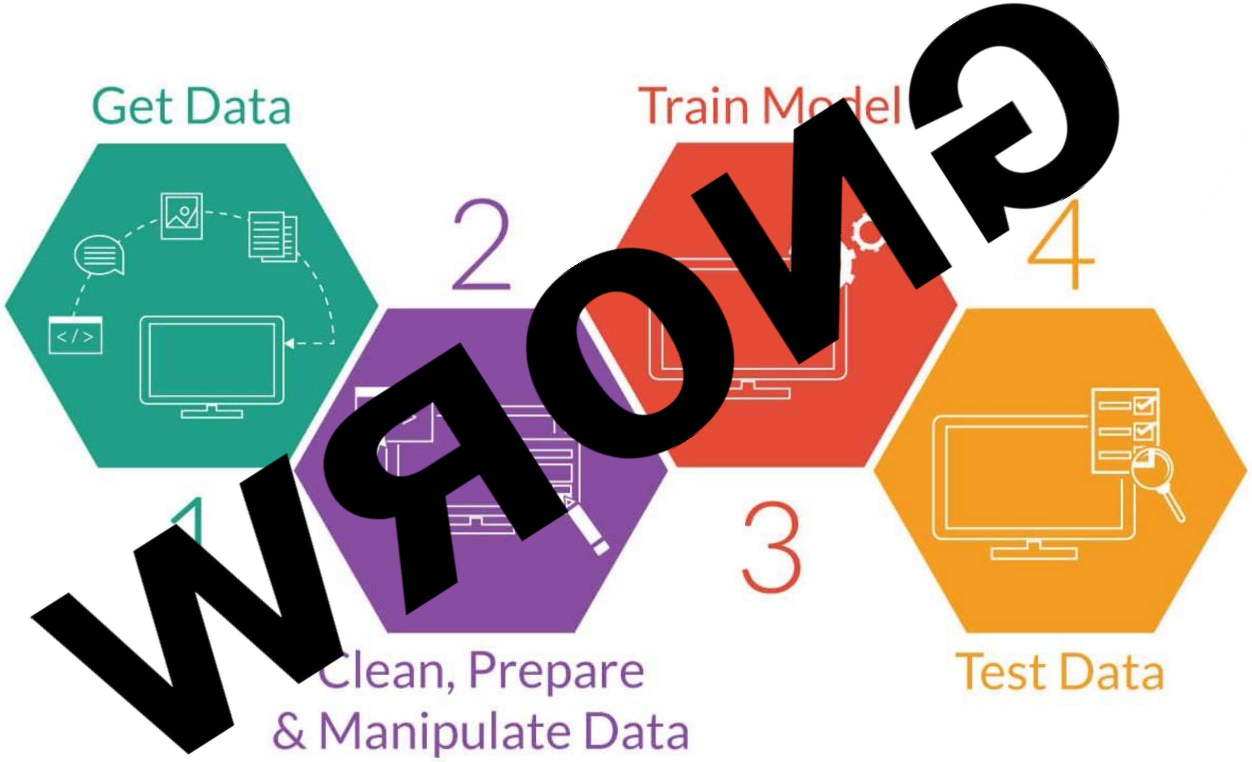
Train / Validation / Test

Workflow



Practical Aspects

Train / Validation / Test
Workflow



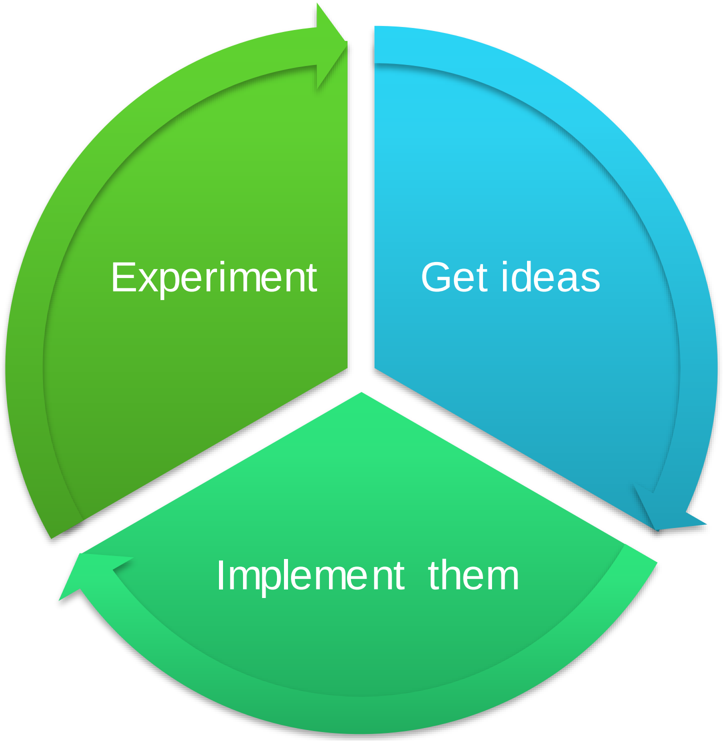
Practical Aspects

Train / Validation / Test

Workflow

Lots of hyper-parameters:

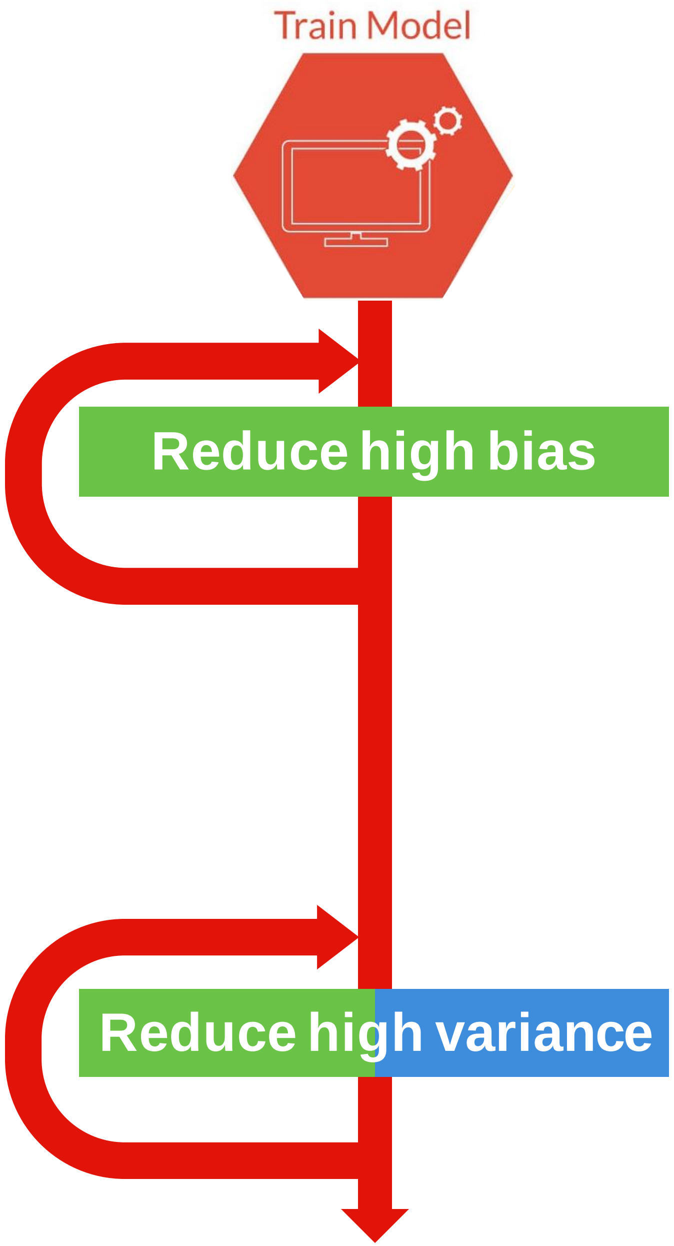
- Network architecture:
    - # layers
    - # neurons
    - Activation function
- Learning rate
- Optimization algorithm
- ...



Practical Aspects

Train / Validation / Test

Workflow

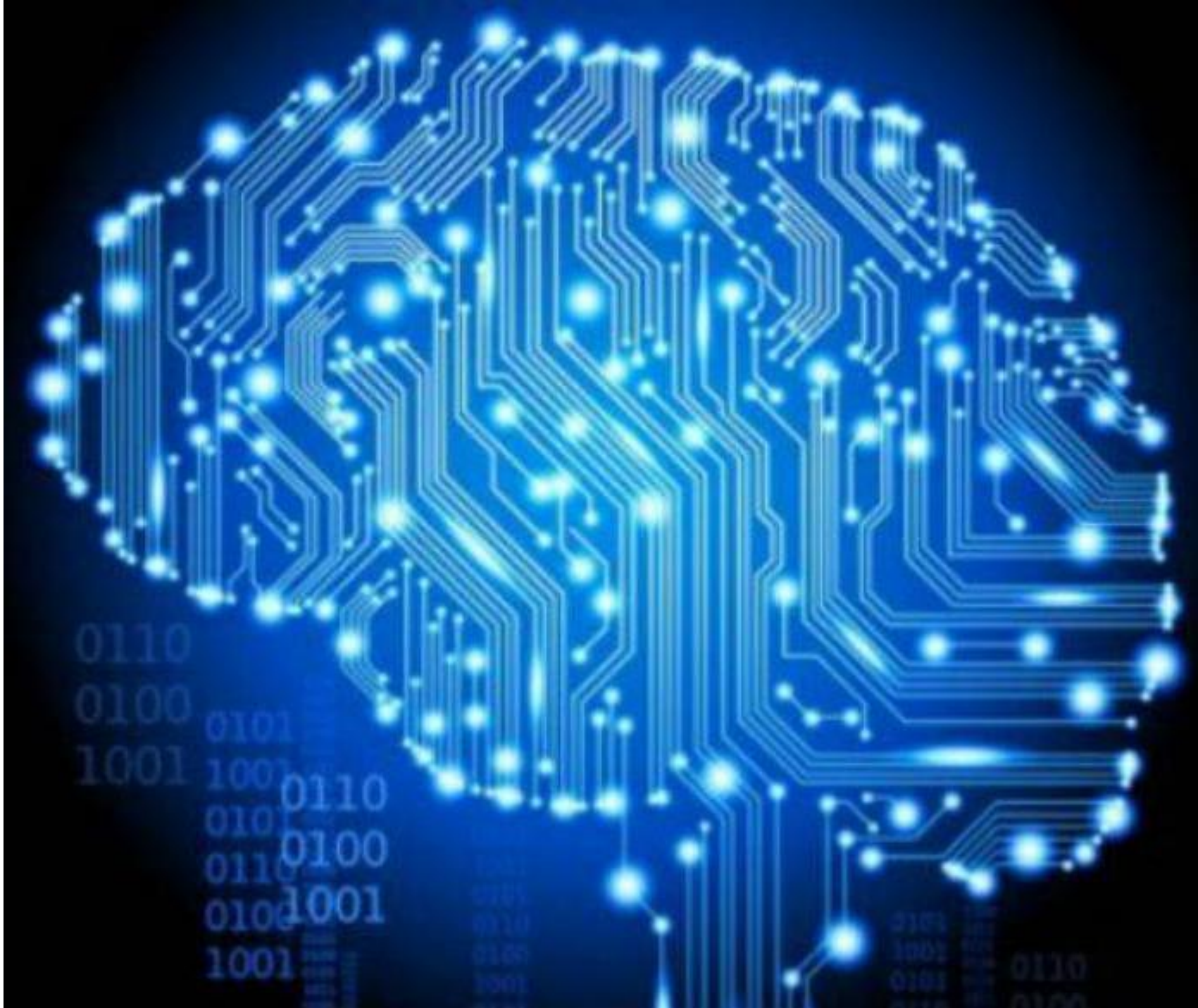


- Train longer
- Increase model's capacity:
  - More layers
  - More hidden neurons
- Search for architecture suitable for your data structure
- Preprocessing data to simplify the task

- More data
- Data Augmentation
- Regularization
- Search for architecture suitable for your data structure

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Convolutional Neural Networks



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

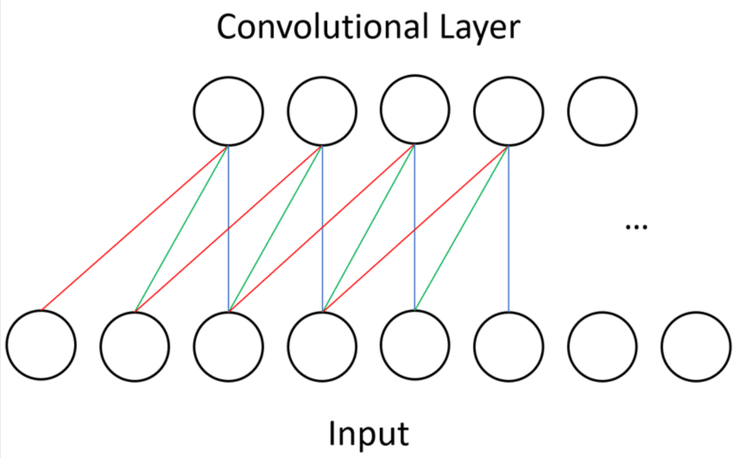
Convolutional Neural Networks

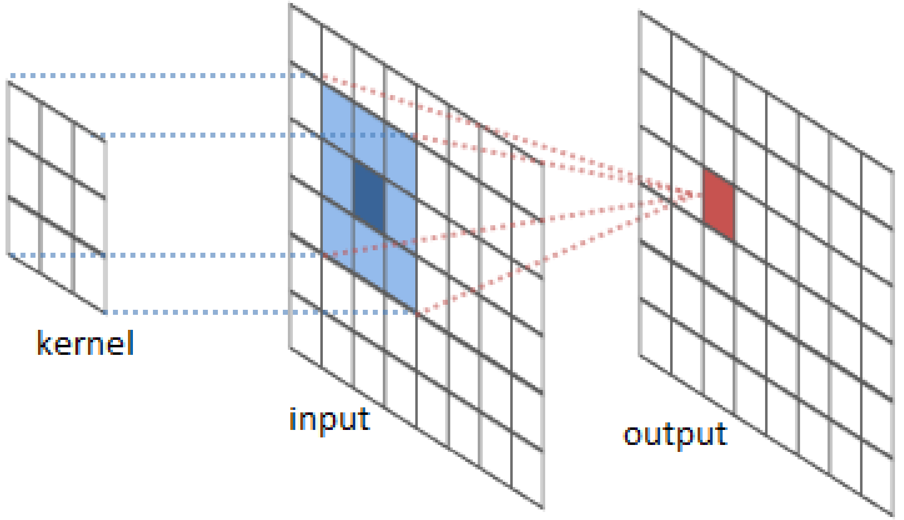
Limited connectivity
Convolution & weight sharing
Filters
Kernel size, stride and padding
Convolutional volumes
Pooling layers
Convolutional architectures
CNNs from the inside
CNN Applications

Some data has spatial correlations that could be exploited (in 1D, 2D, 3D, ...):

- *Near-by* data points are more relevant than *far-away.*

If we sparsify connectivity with a consistent purpose, we may **reduce complexity** and ease the learning of **more coherent patterns**





Convolutional Neural Networks

Limited connectivity

Convolution & weight sharing

Filters

Kernel size, stride and padding

Convolutional volumes

Pooling layers

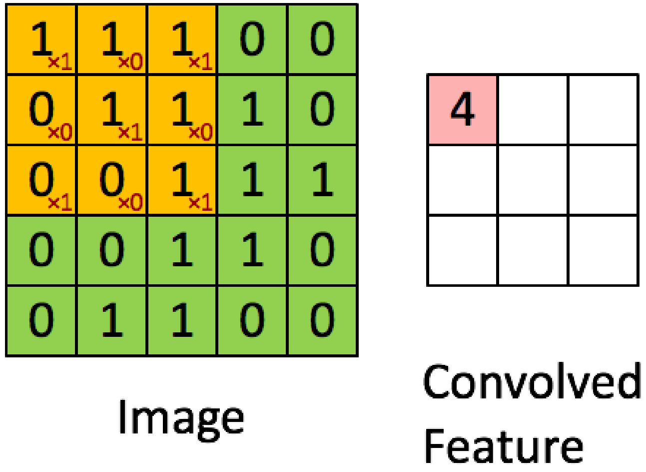
Convolutional architectures

CNNs from the inside

CNN Applications

Sparse connectivity is nice, but we still want to apply filters everywhere.

Each limited connectivity pattern (a **kernel**) will get **convolved** all over the image, generating a number of values.
Notice each kernel generates a 2D matrix of values.



In practice we have sets of neurons **sharing weights**

Convolutional Neural Networks

- **Limited connectivity**
- **Convolution & weight sharing**
- **Filters**
- **Kernel size, stride and padding**
- **Convolutional volumes**
- **Pooling layers**
- **Convolutional architectures**
- **CNNs from the inside**
- **CNN Applications**

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Convolution kernels can do all sorts of things on an image:

Input image

Edge detection	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Gaussian blur 3 × 3	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

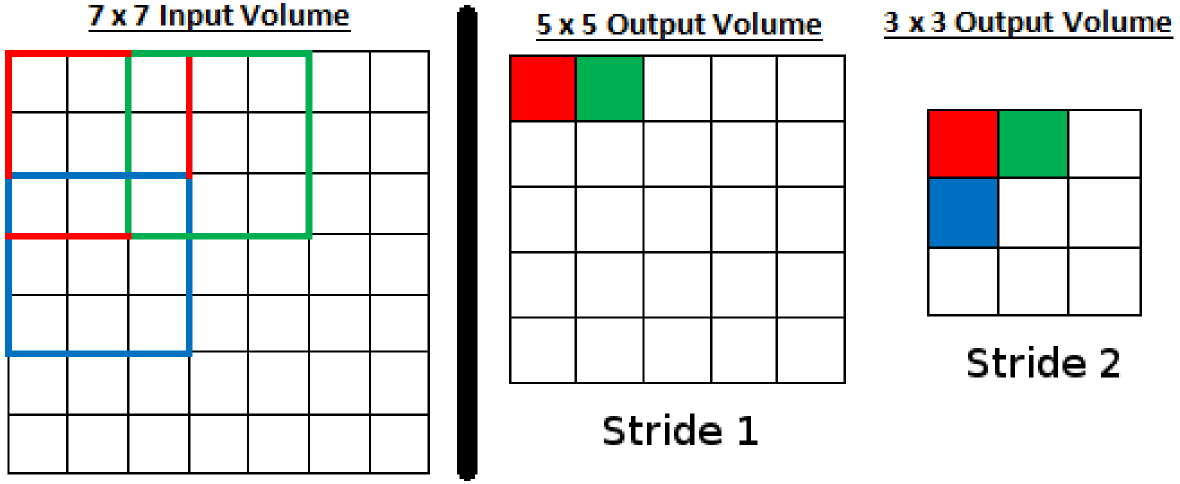
Let's let the model learn them

Convolutional Neural Networks

Limited connectivity
Convolution & weight sharing
Filters
Kernel size, stride and padding
Convolutional volumes
Pooling layers
Convolutional architectures
CNNs from the inside
CNN Applications

Kernel size: Size of the receptive field of convolutional neurons. Typically 3x3, 5x5, 7x7

Stride: Number of steps while convolving filter.



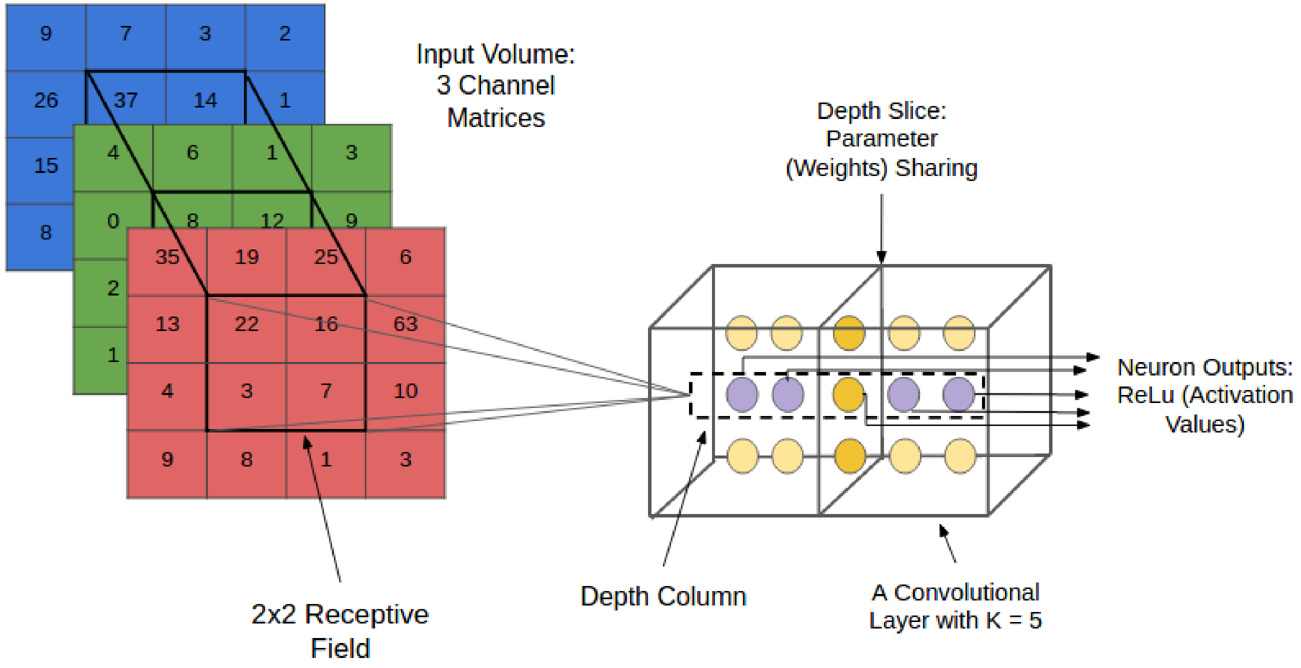
Stride 1 the most common. Larger strides can replace pooling.

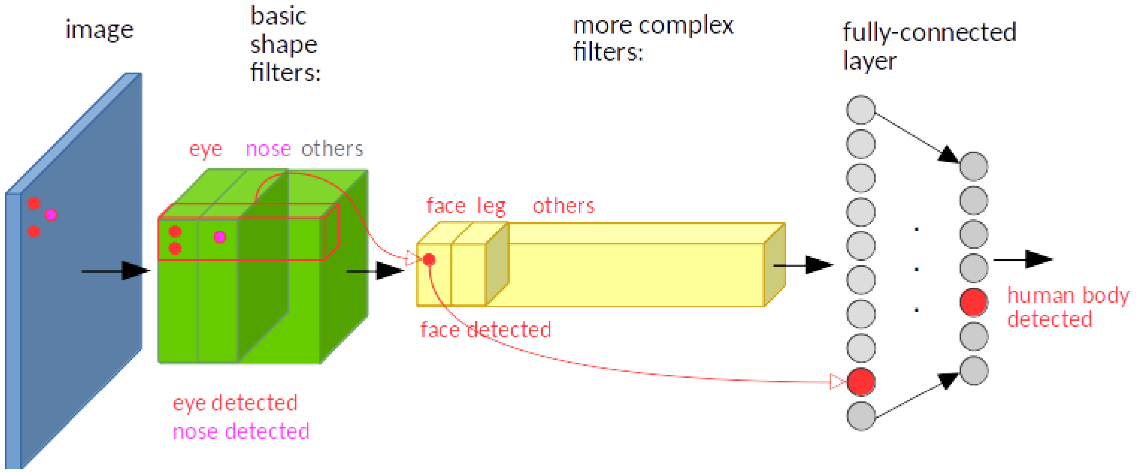
Padding: Border added to center conv. everywhere
- No padding: Dimensionality reduced
- Most common, zero equal/same padding

$$OutputSize = \frac{InputSize - KernelSize + 2 * Padding}{Stride} + 1$$

Convolutional Neural Networks

- In a typical 2D CNN, conv filters are 3D (full depth).
- Each filter convolved generates a 2D plane of data.
- Depth provides all the neural views on a part of data





Limited connectivity

Convolution & weight sharing

Filters

Kernel size, stride and padding

Convolutional volumes

Pooling layers

Convolutional architectures

CNNs from the inside

CNN Applications

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

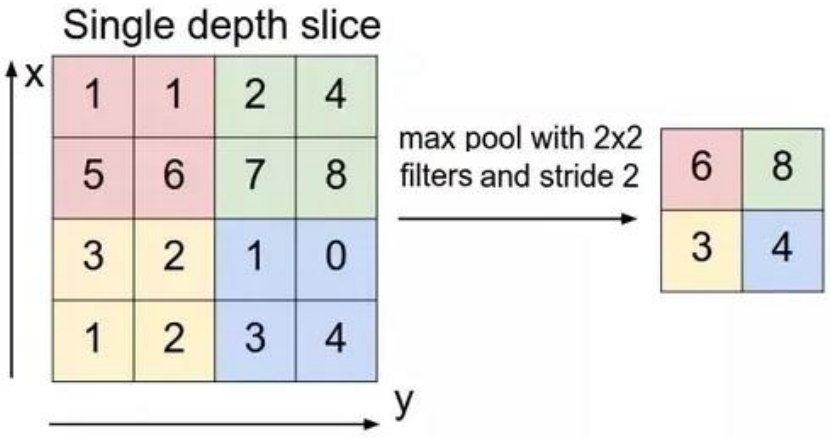
Convolutional Neural Networks

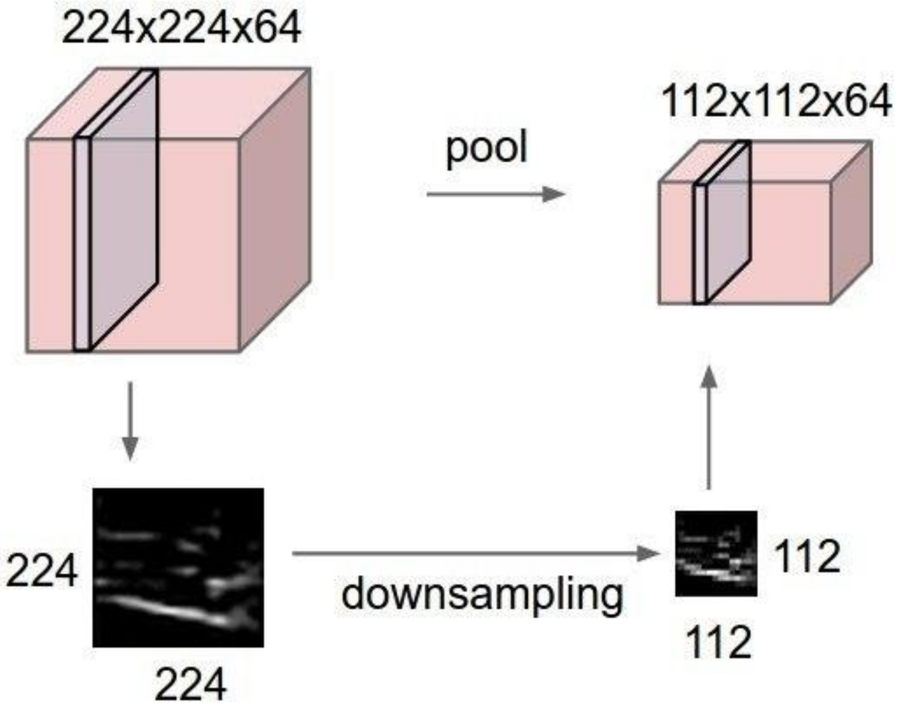
Limited connectivity
Convolution & weight sharing
Filters
Kernel size, stride and padding
Convolutional volumes
Pooling layers
Convolutional architectures
CNNs from the inside
CNN Applications

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Pooling:

- Small spatial invariance
- Dimensionality reduction (along x and y only)
- Never applied full depth!
- Parameter free layer
- Hyperparams:
  - Size & Stride
- Loss in precision
- Max >> Avg





Convolutional Neural Networks

- **Limited connectivity**
- **Convolution & weight sharing**
- **Filters**
- **Kernel size, stride and padding**
- **Convolutional volumes**
- **Pooling layers**
- **Convolutional architectures**
- **CNNs from the inside**
- **CNN Applications**

[AlexNet,12] [VGG,14]

The first influential architecture was **AlexNet**:

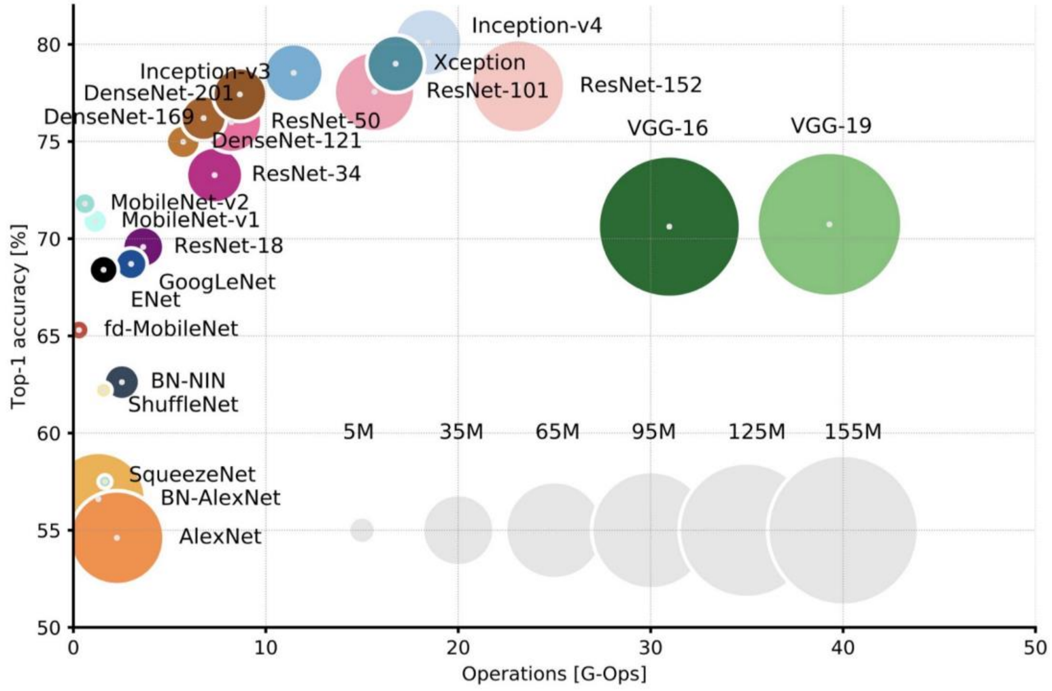
- 5 layers using convs, pools, *ReLU*, 2 dense, and *dropout*.
- 62M parameters

VGG16/19 extends the (conv-pool)*dense design:

- Smaller, 3x3 filters, but more
- 138M parameters

Some design principles: KISS, be repetitive & pyramidal

Bigger is not better!



Convolutional Neural Networks

- Limited connectivity
- Convolution & weight sharing
- Filters
- Kernel size, stride and padding
- Convolutional volumes
- Pooling layers
- **Convolutional architectures**
- CNNs from the inside
- CNN Applications

[Inception,15]
[ResNet,16]
[Huang,16]
[Xu,WWW]

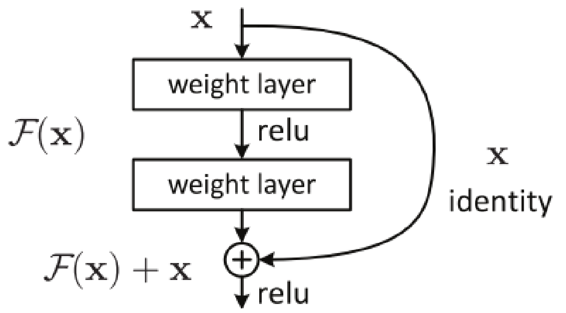
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

But deeper should never be worse!

- In theory, yes. In practice, identity is hard to learn

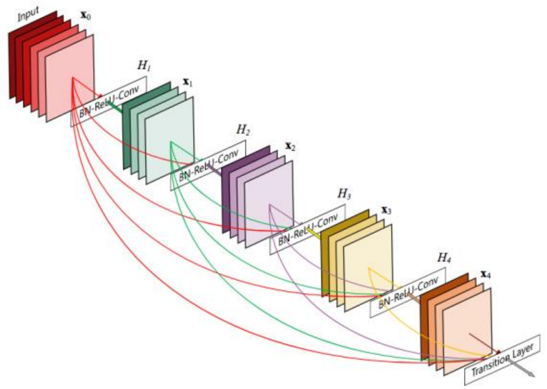
ResNet: Learning zero is easier than learning id.

- We can now train a 1K layer net



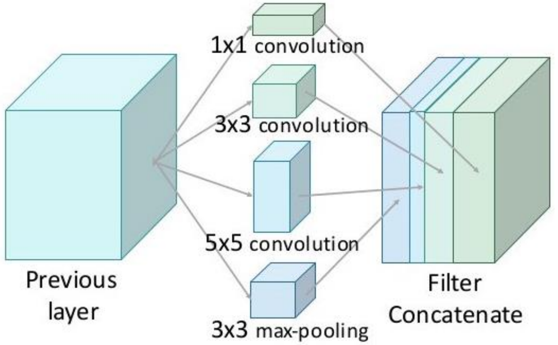
DenseNet: link all to all

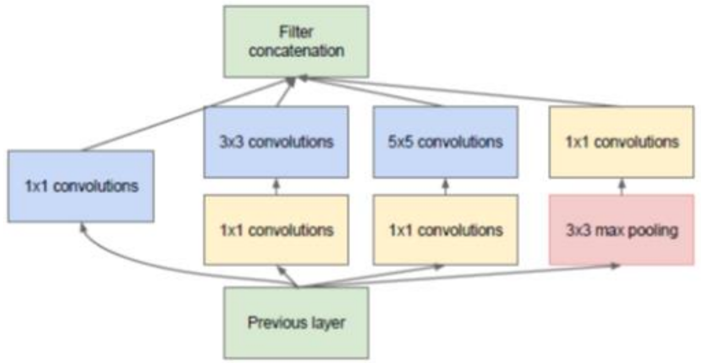
- Use depth concats
- 1x1 convs to make it feasible



Inception: how to fix filter size?

- Let the net decide which is best
- Avg. Pooling instead of dense





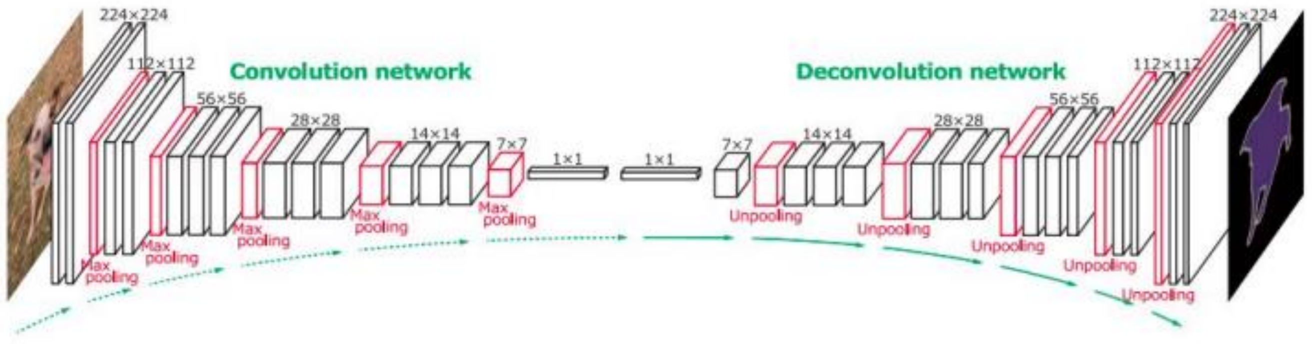
Convolutional Neural Networks

Limited connectivity
Convolution & weight sharing
Filters
Kernel size, stride and padding
Convolutional volumes
Pooling layers
Convolutional architectures
CNNs from the inside
CNN Applications

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Different architectures that can be done...

- Convolution – Transposed convolution (pixel-wise)

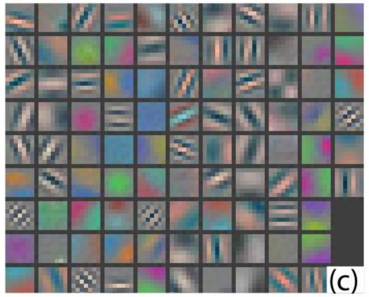


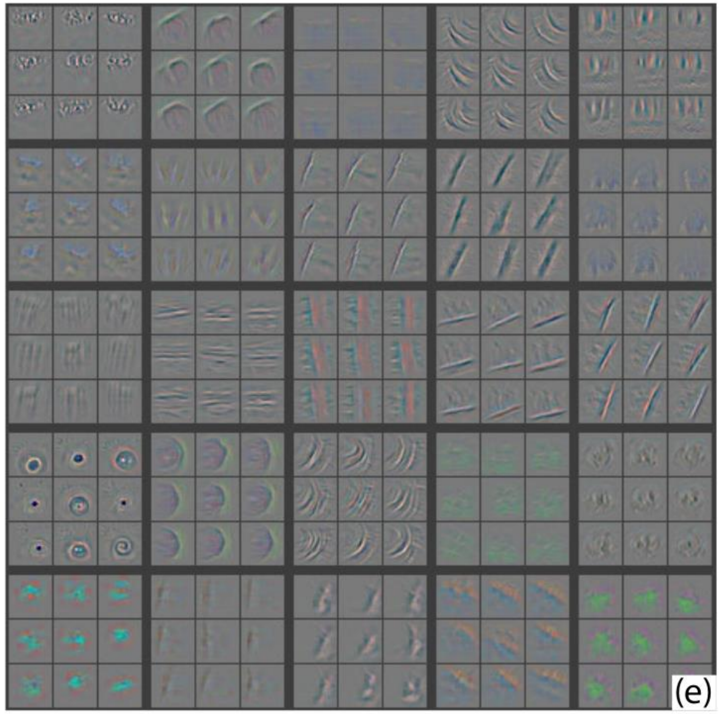
Convolutional Neural Networks

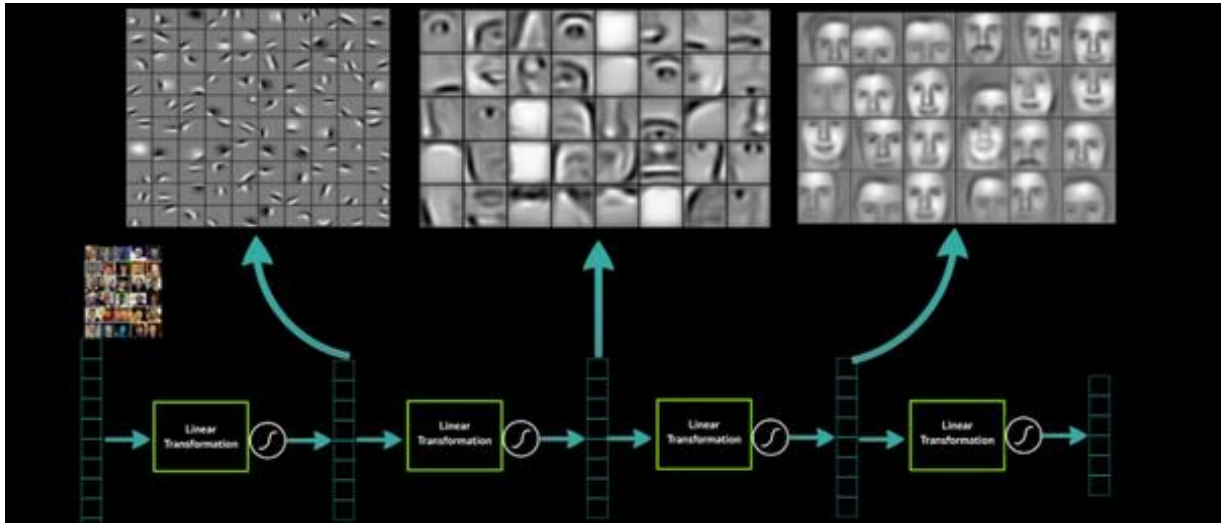
- Limited connectivity
- Convolution & weight sharing
- Filters
- Kernel size, stride and padding
- Convolutional volumes
- Pooling layers
- Convolutional architectures
- **CNNs from the inside**
- CNN Applications

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

What do filters learn?



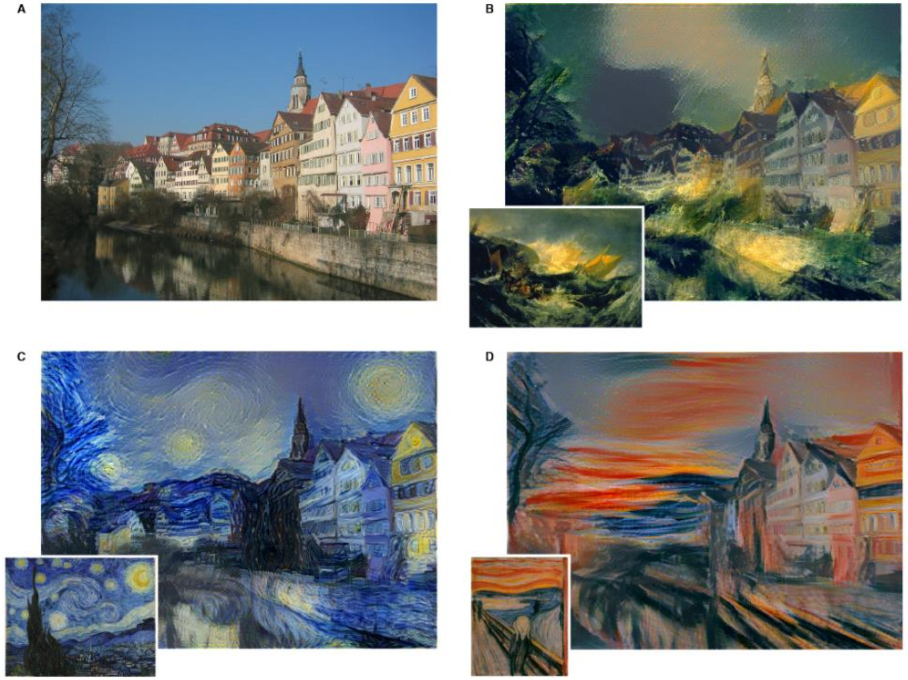




Convolutional Neural Networks

Limited connectivity
Convolution & weight sharing
Filters
Kernel size, stride and padding
Convolutional volumes
Pooling layers
Convolutional architectures
CNNs from the inside
CNN Applications

Style transfer



Multimodal pipelines

[Gatys,15]
[Kiros,14]

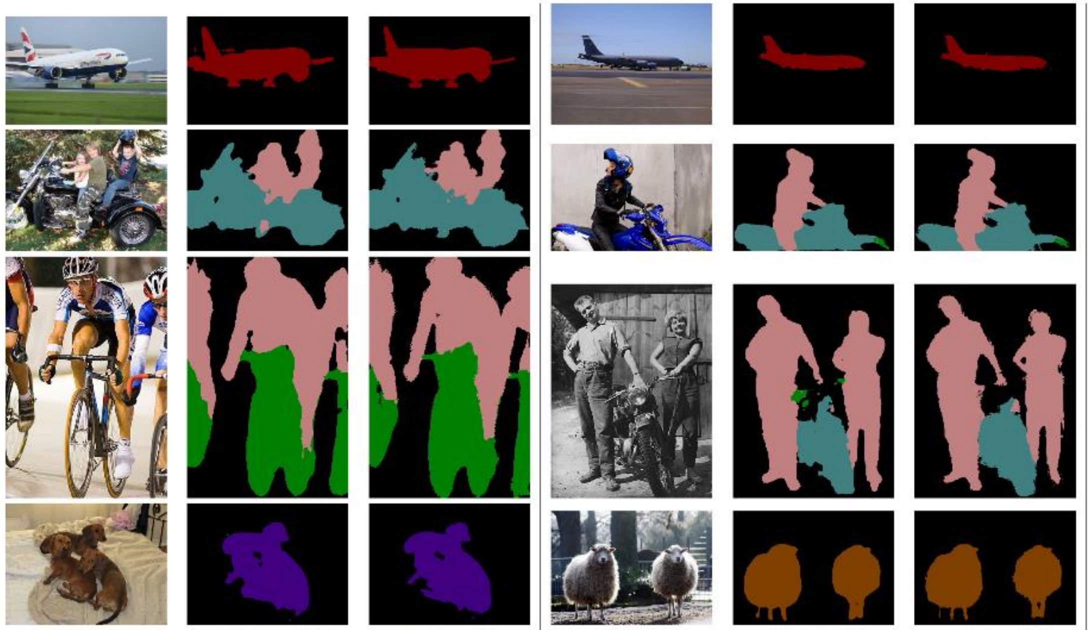
Convolutional Neural Networks

- Limited connectivity
- Convolution & weight sharing
- Filters
- Kernel size, stride and padding
- Convolutional volumes
- Pooling layers
- Convolutional architectures
- CNNs from the inside
- **CNN Applications**

Image colorization

(a) Colorado National Park, 1941 (b) Textile Mill, June 1937 (c) Berry Field, June 1909 (d) Hamilton, 1936

Image segmentation



(a) Image (b) Before CRF (c) After CRF (a) Image (b) Before CRF (c) After CRF

[Zhang,16]
[Iizuka,16]
[Chen,16]

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Thanks

mail@domain.com