MAI

Deep Learning

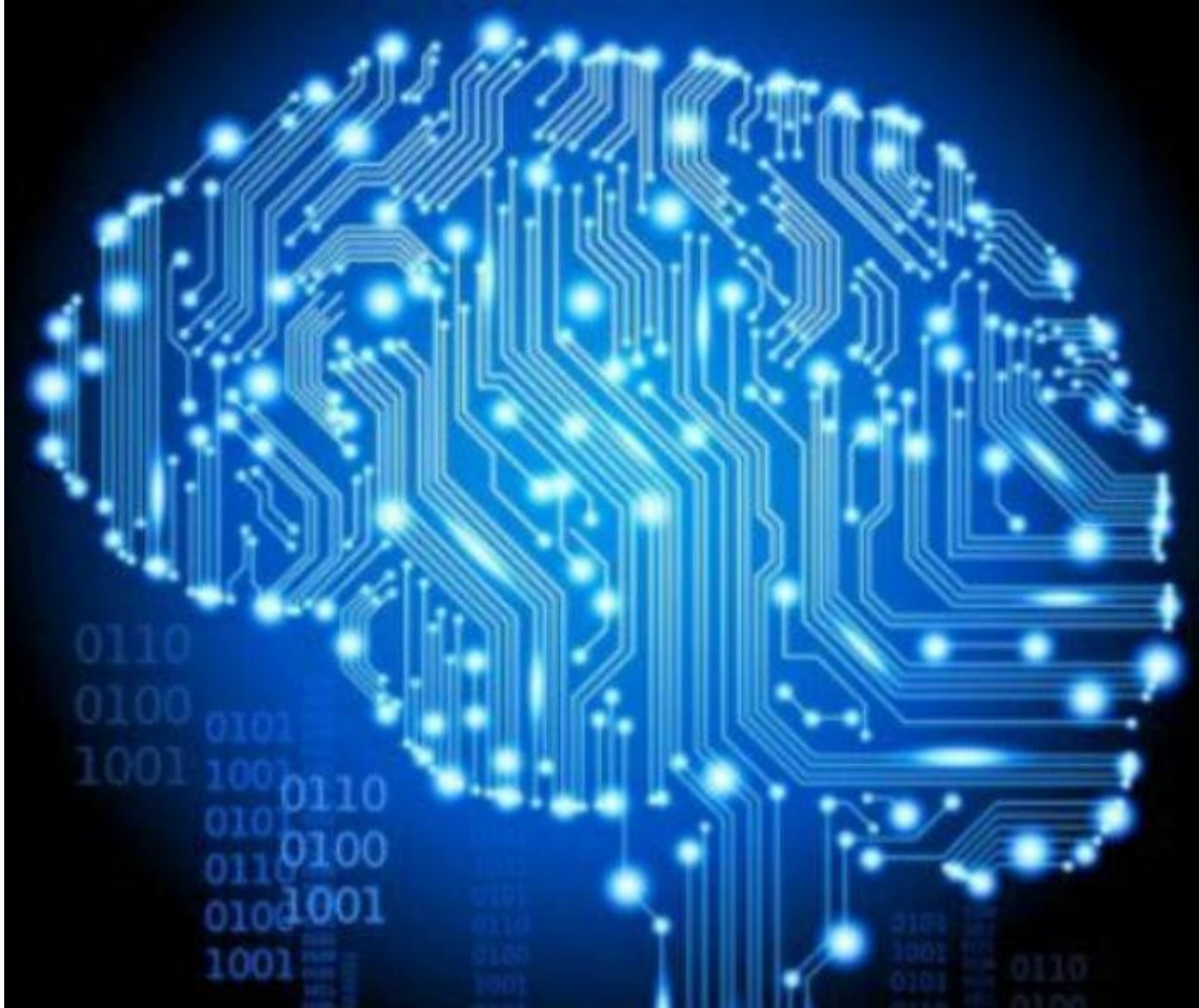
THEORY

Feed-forward and convolutional neural networks

Barcelona Supercomputing Center
Centro Nacional de Supercomputación
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Dario Garcia Gasulla
dario.garcia@bsc.es

A bit of History



HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

A bit of History

McCullock & Pitts / Hebb

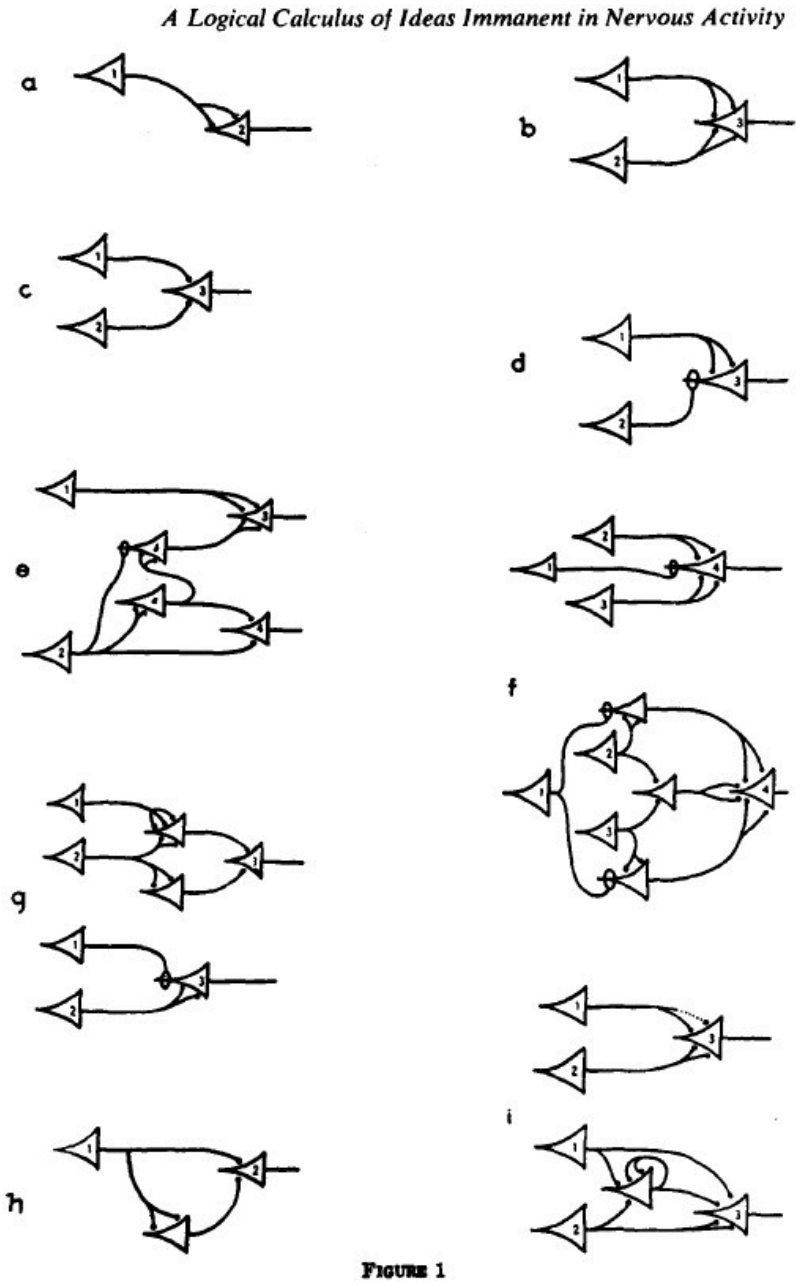
Rosenblatt's Perceptron

Minsky & Papert - XOR

Backpropagation Algorithm

[McCullock, 43]

- 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



A bit of History

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

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

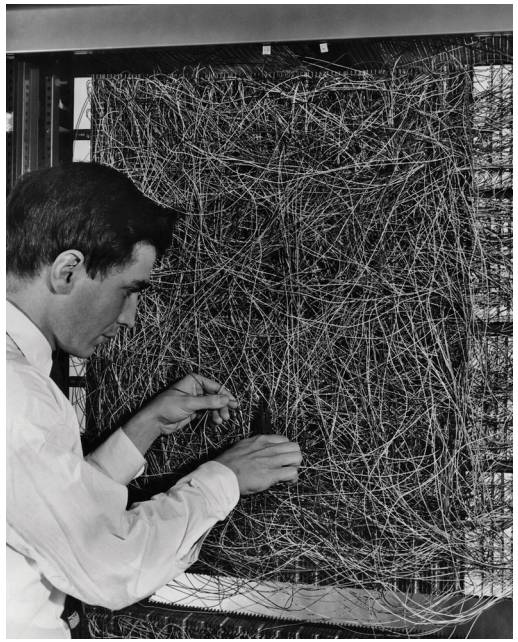
w real-valued weights
· dot product
b real scalar constant

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

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

The Mark I Perceptron. A visual classifier with:

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



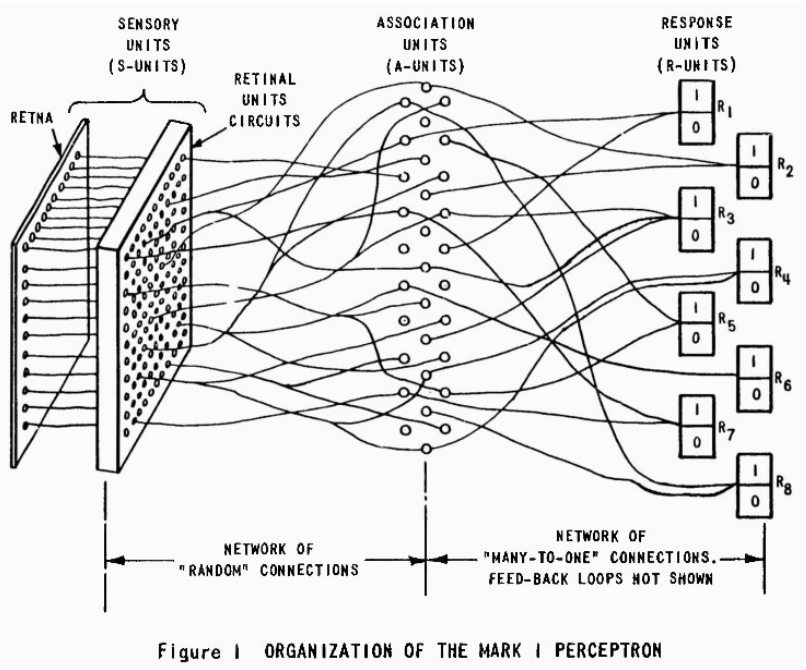


Figure 1 ORGANIZATION OF THE MARK I PERCEPTRON

A bit of History

McCullock & Pitts / Hebb

Rosenblatt's Perceptron

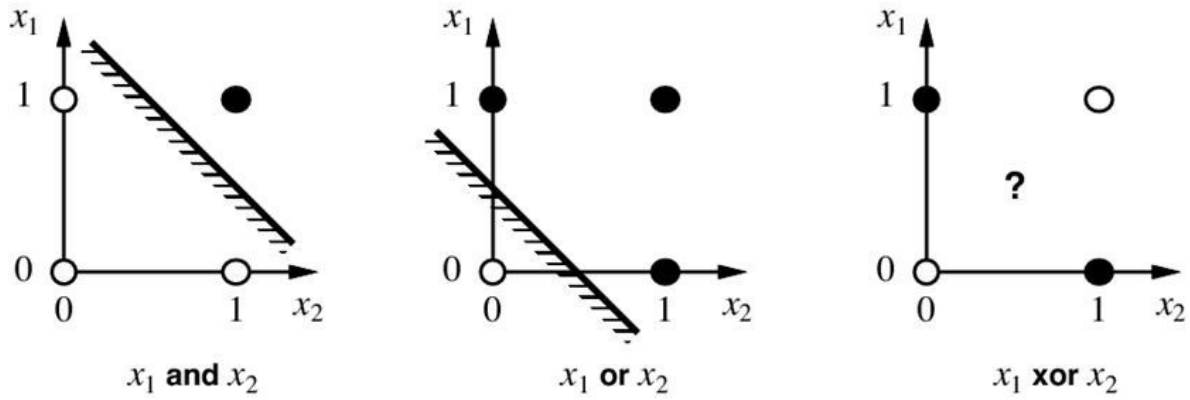
Minsky & Papert - XOR **[Minsky, 69]**

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

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.

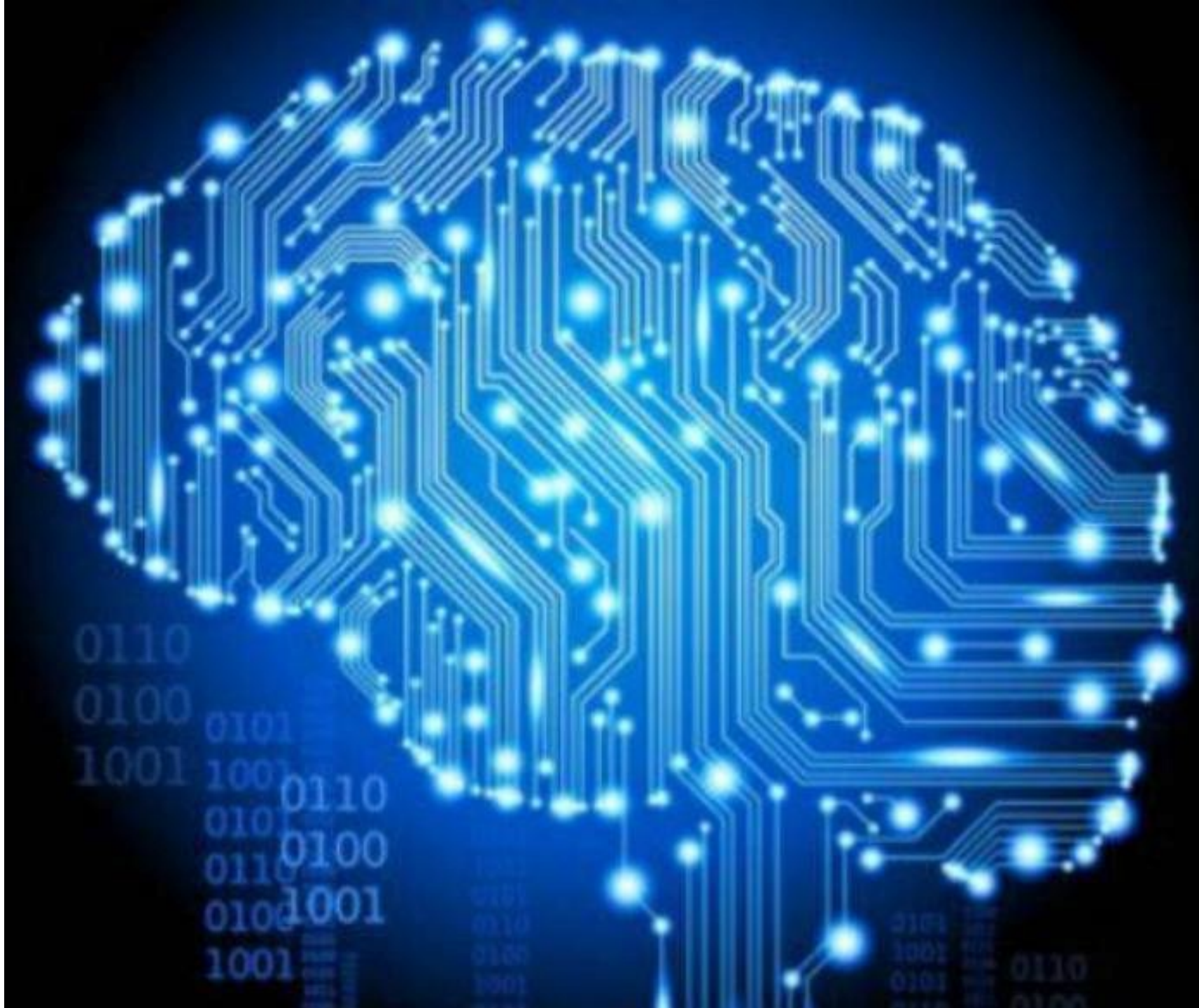
End of NNs Winter

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)

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks



HP AI
HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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.

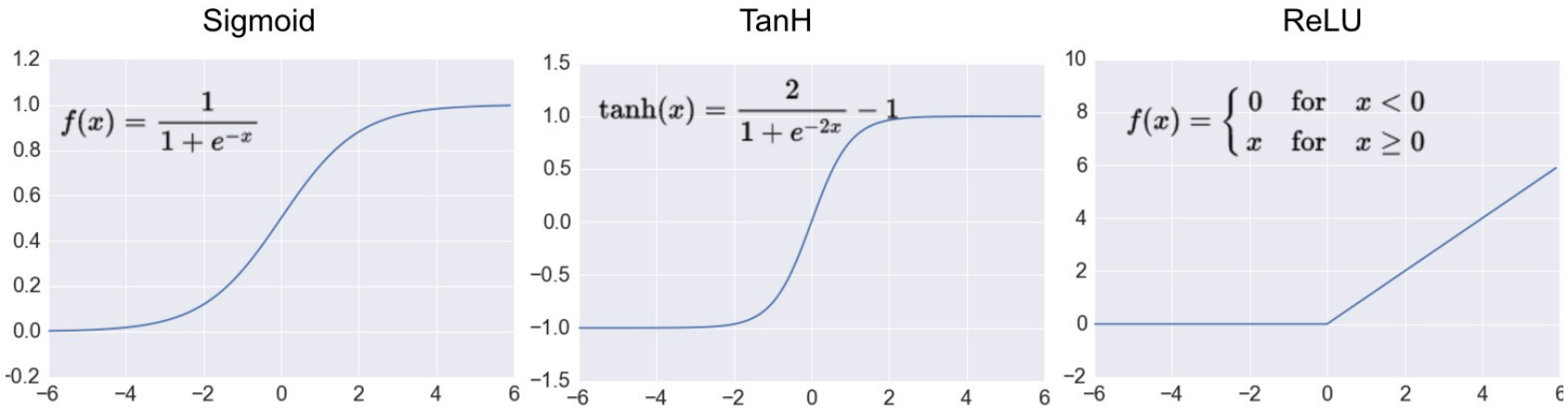
- Typical batch sizes: Powers of 2.
- Batch size = 1 --> Full stochastic (slower)
- Batch size = dataset_size --> Deterministic (bad generalization)

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

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, ...

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

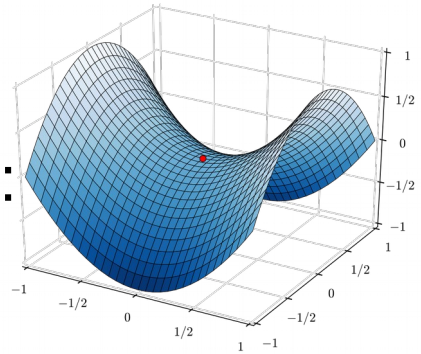
HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

Gradient descent is an 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, a saddle point, or be too slow.

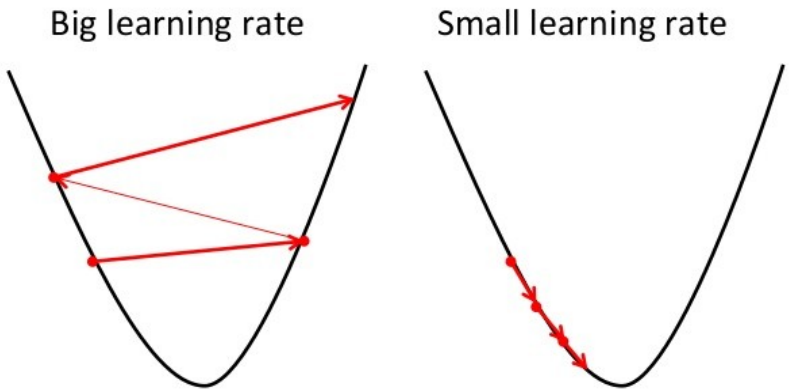
Saddle point:

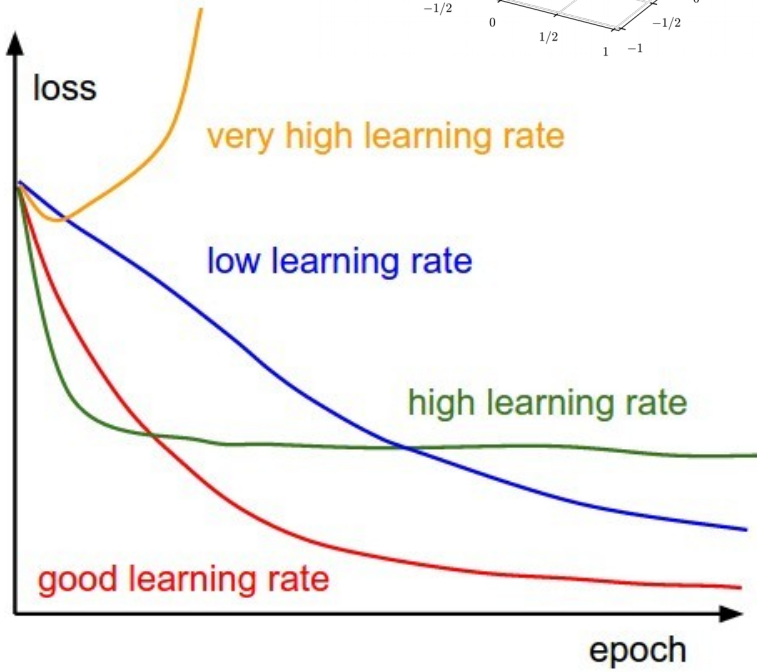


SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

[Dauphin, 14]

SGD will overshoot unless we keep decreasing the LR.





HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

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

[Dauphin, 14]
[Ruder,www]

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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. Vanishing LR due to growing history.

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

Adam: Adadelta + Momentum

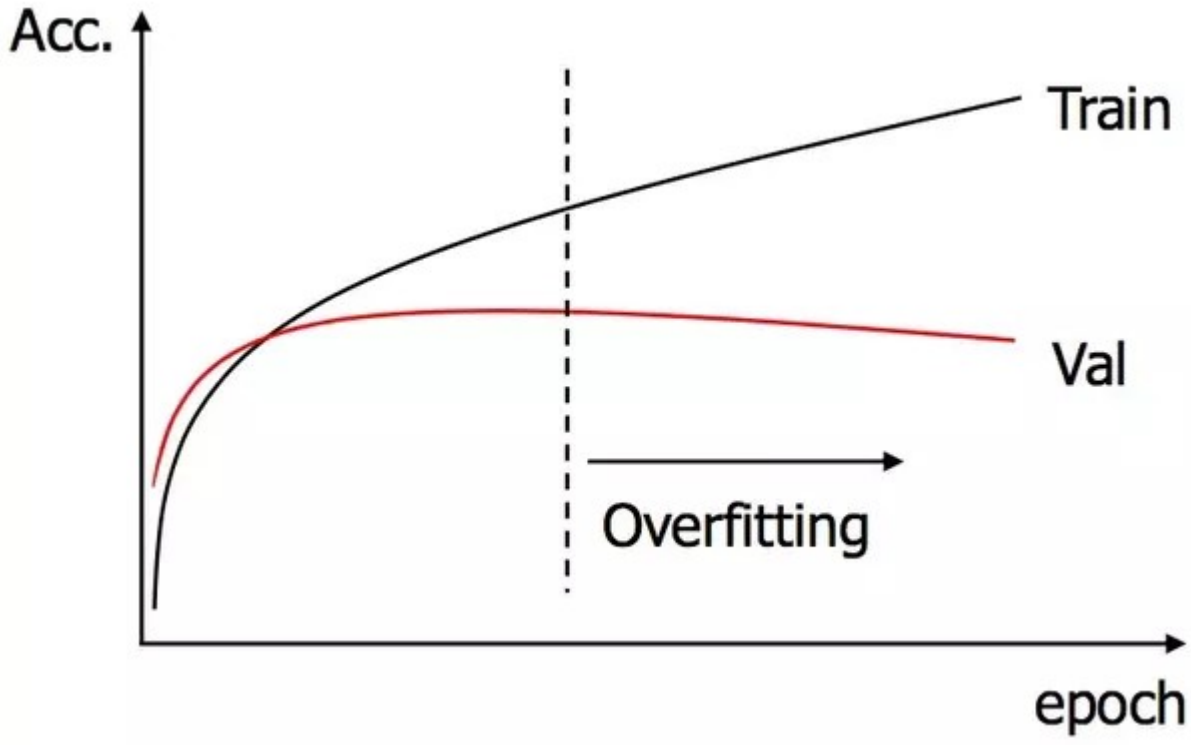
Feedforward Neural Networks

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Why do we need regularization?

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



Generalization

Feedforward Neural Networks

Polynomial regression

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

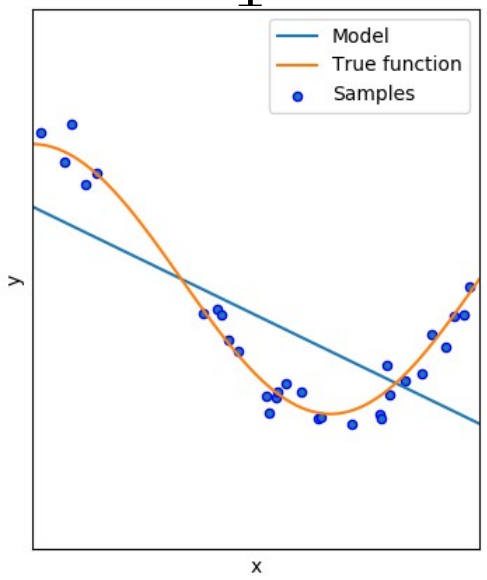
2 $h(x) = w_3 x^3 + w_2 x^2 + w_1 x + b$

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

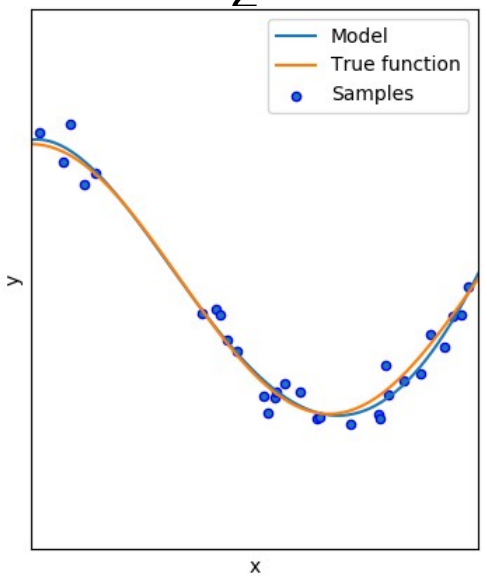
SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

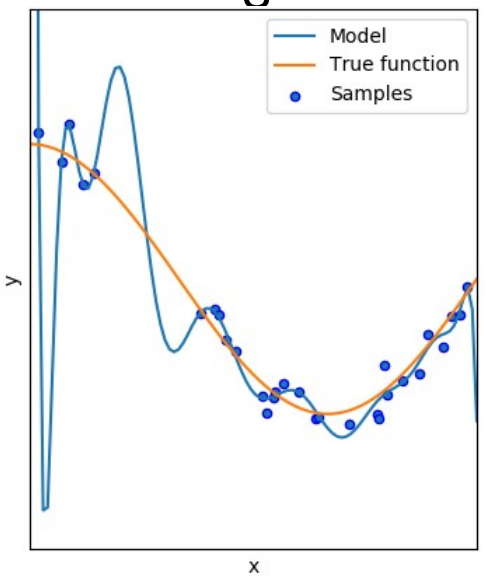
1



2



3



Feedforward Neural Networks

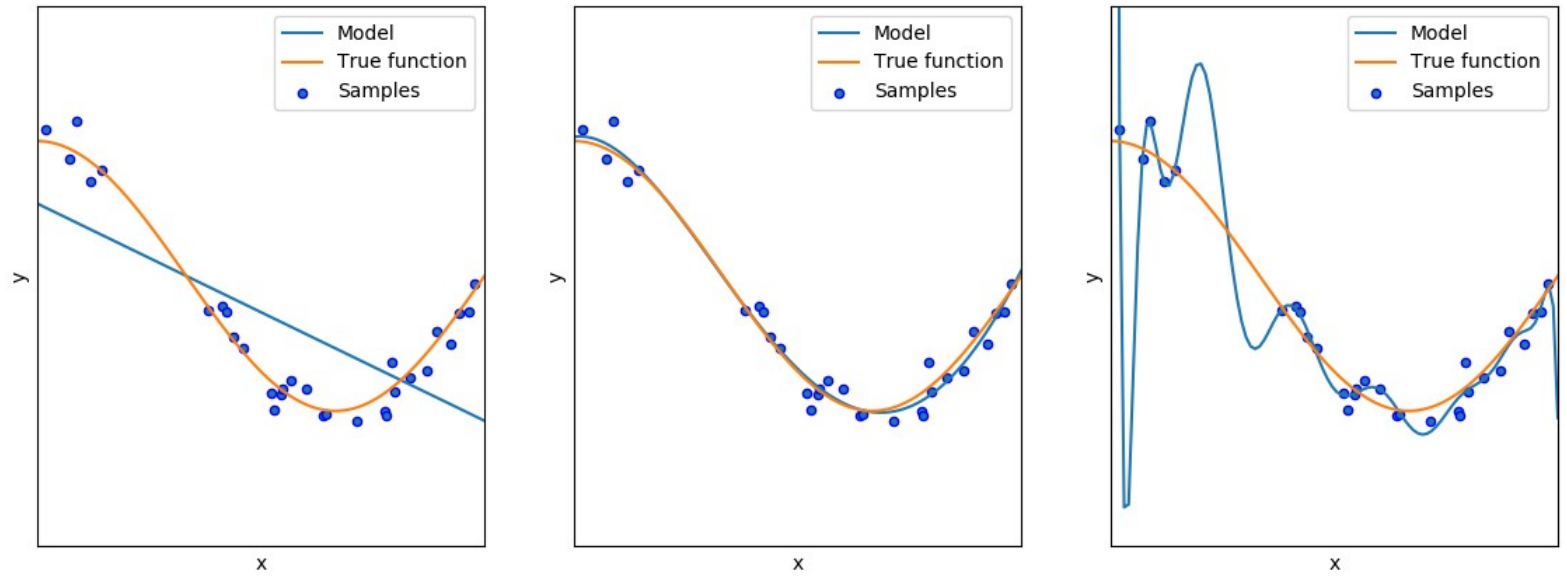
SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

Generalization

Polynomial regression

Training Error		
Huge	Small	Almost Zero

Model Generalization		
Bad	Good	Horrible



HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

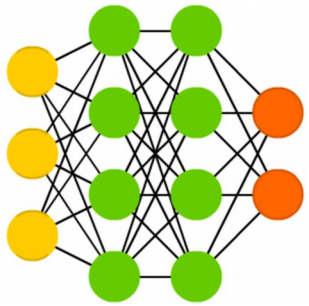
What **policy** can we use to **improve model generalization**?

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**

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

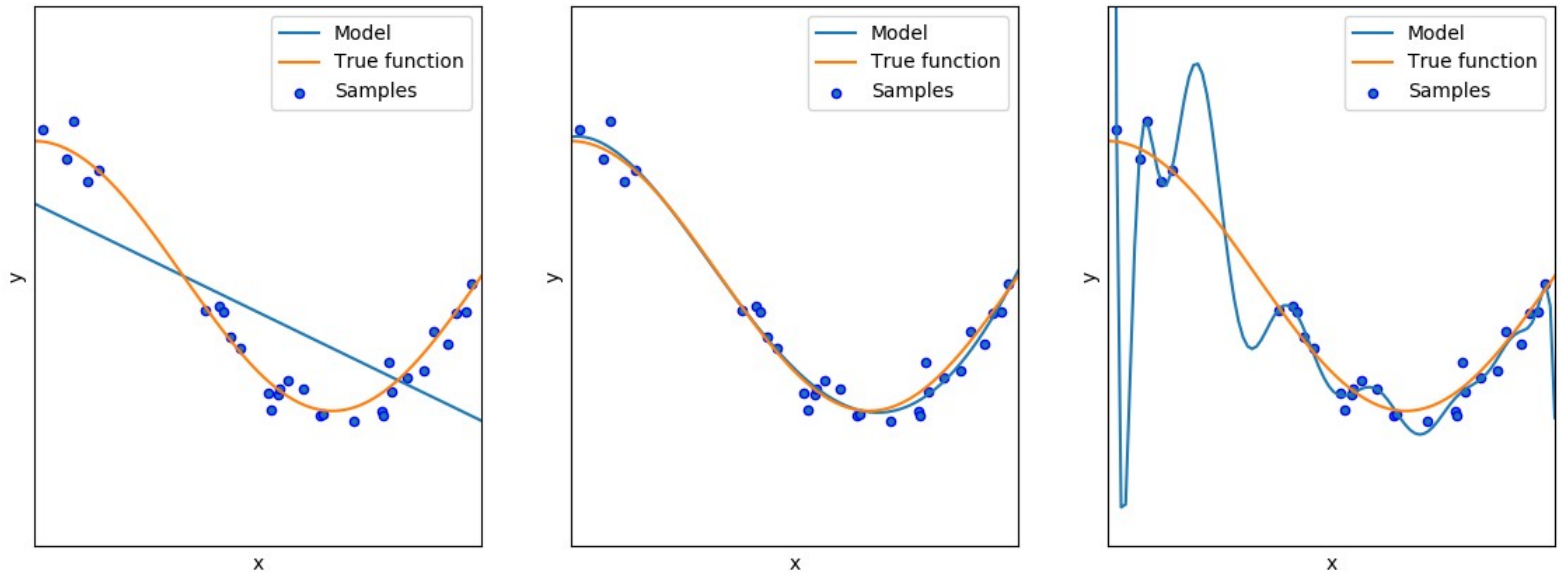
Feedforward Neural Networks

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

Model Complexity

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

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



HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

Feedforward Neural Networks

Model Complexity

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

Model Complexity

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

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

Feedforward Neural Networks

Model Complexity

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

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

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

Feedforward Neural Networks

Model Complexity

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

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

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

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

Feedforward Neural Networks

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

? ? ?

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

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

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

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 $w_0, ..., w_d$

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

Model Complexity

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

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

ℓ_0 complexity

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

ℓ_1 complexity

$|1.3| + |-1.2| = 2.5$ VS $|2.2| = 2.2$

ℓ_2 complexity

$1.3^2 + (-1.2)^2 = 3.13$ VS $2.2^2 = 4.84$

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

L1 / L2 Regularization

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

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

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

What **complexities** do these methods use?

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

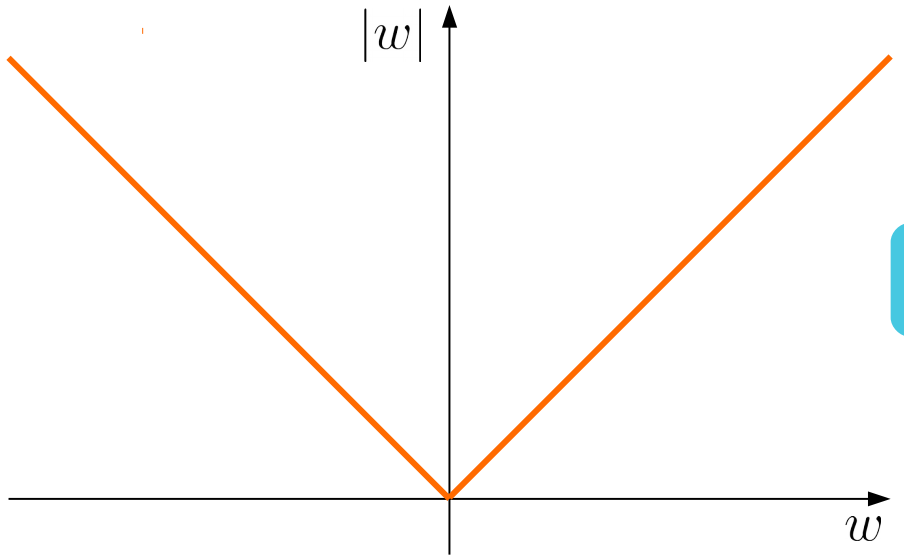
ℓ_2 "ridge" complexity: $\sum_{i=0}^{d} w_i^2$, for coefficients $w_0, ..., w_d$

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

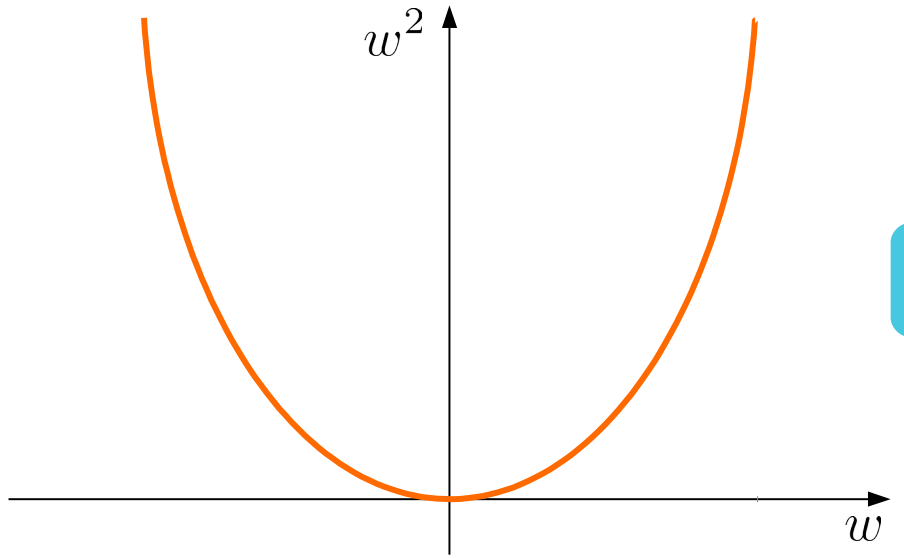
Feedforward Neural Networks

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



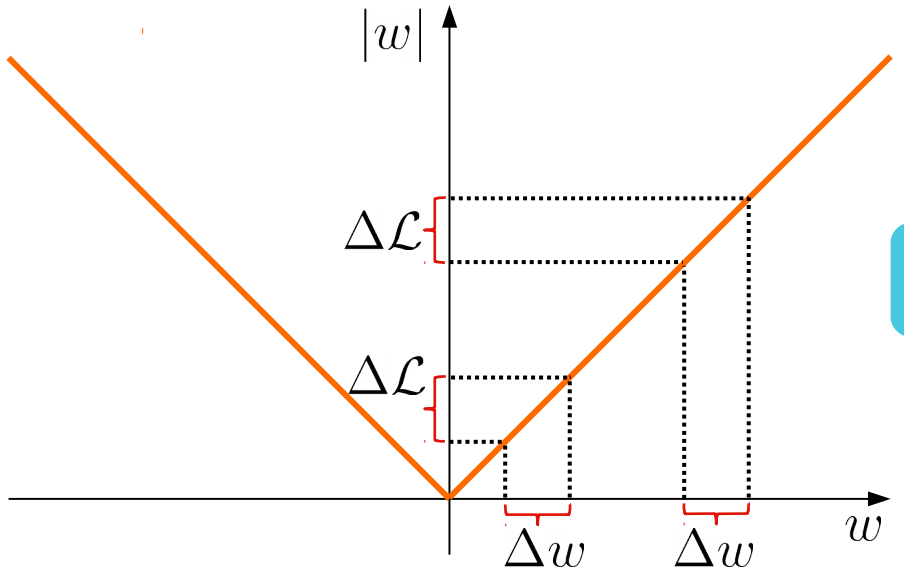
L2 Penalty

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

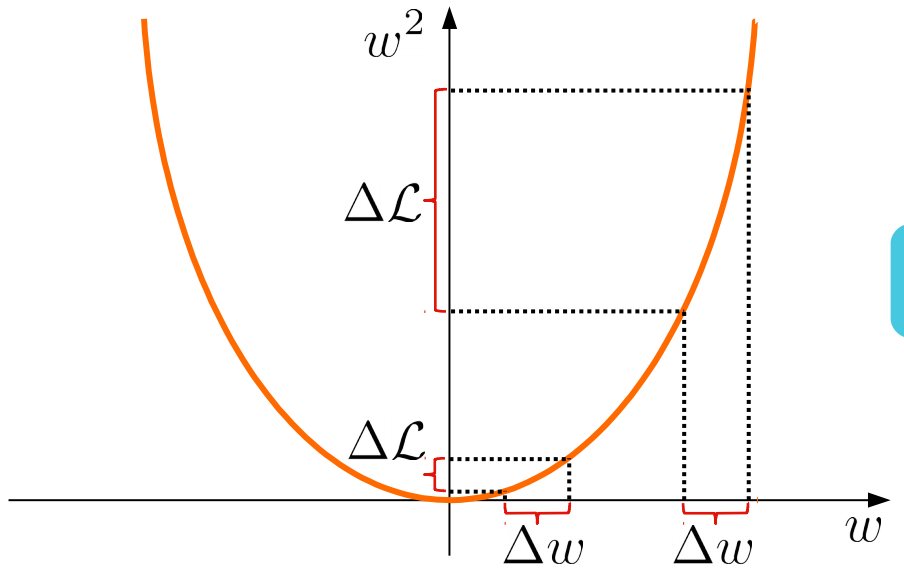
Feedforward Neural Networks

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



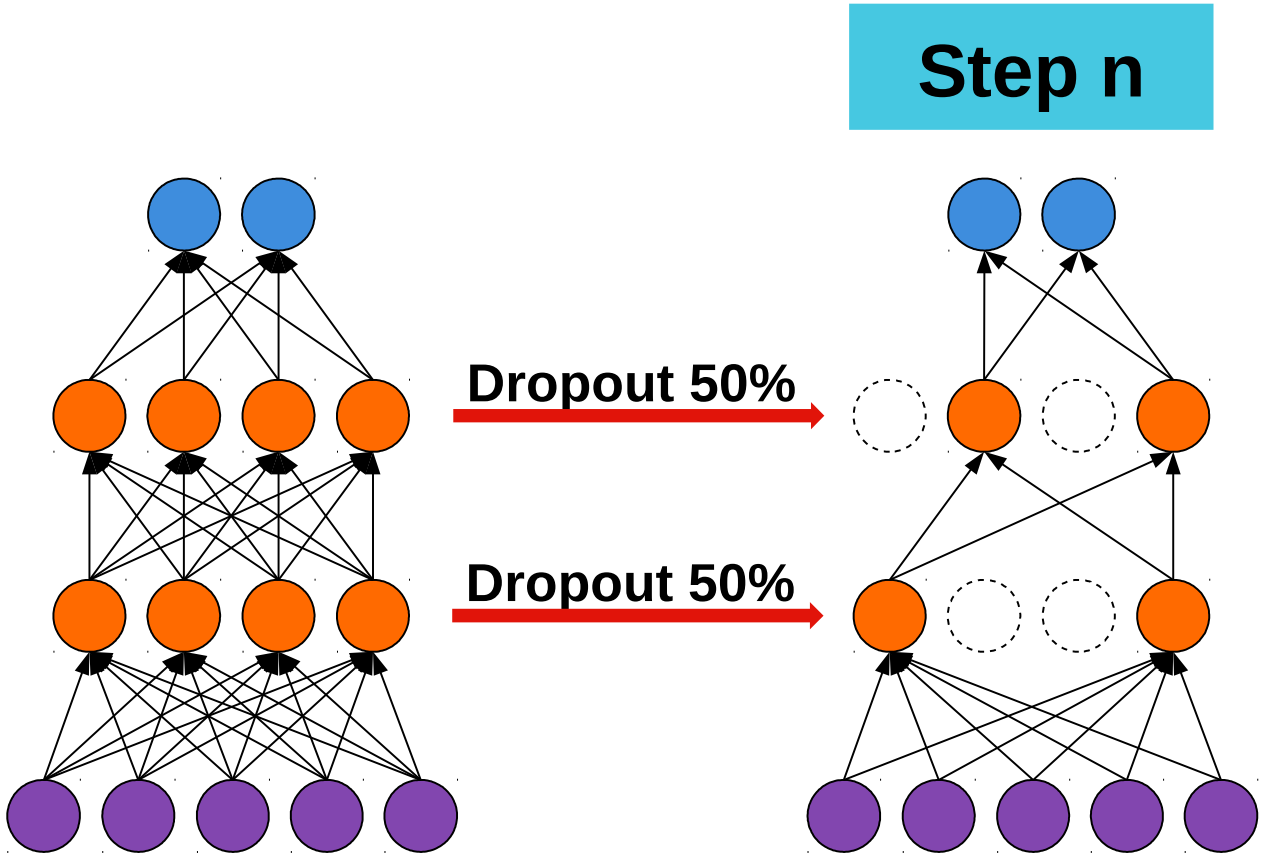
L2 Penalty

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

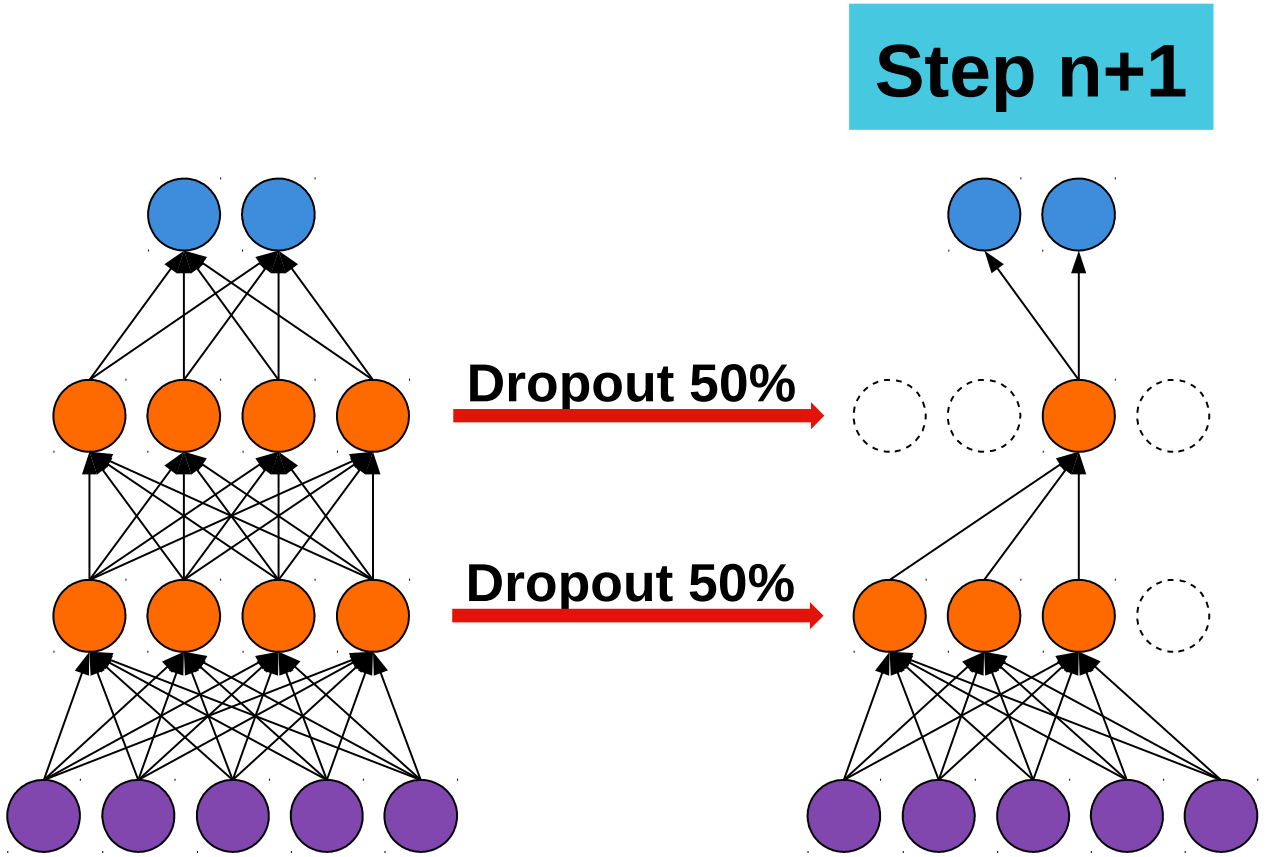


HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

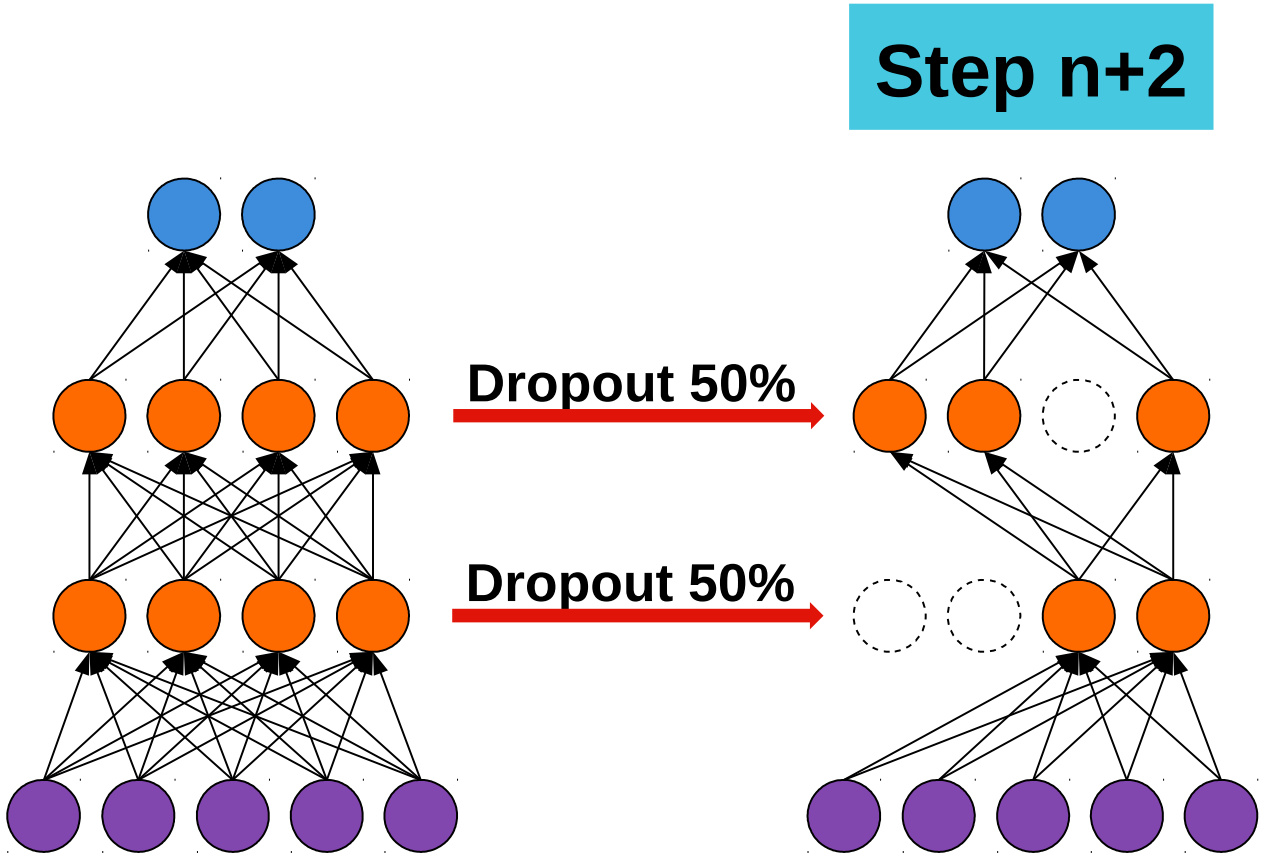


HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

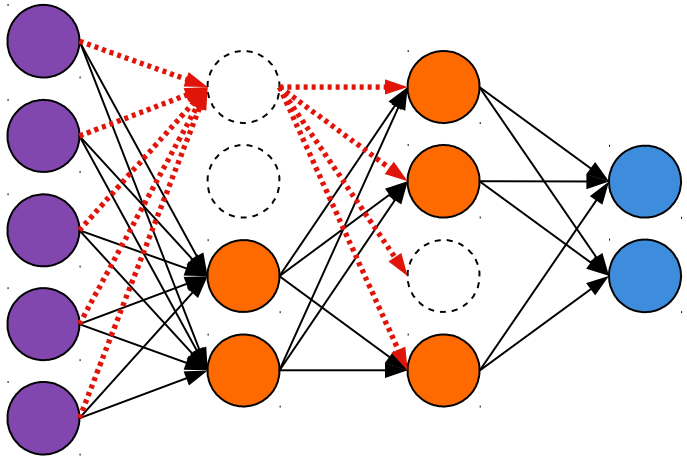


HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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



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?

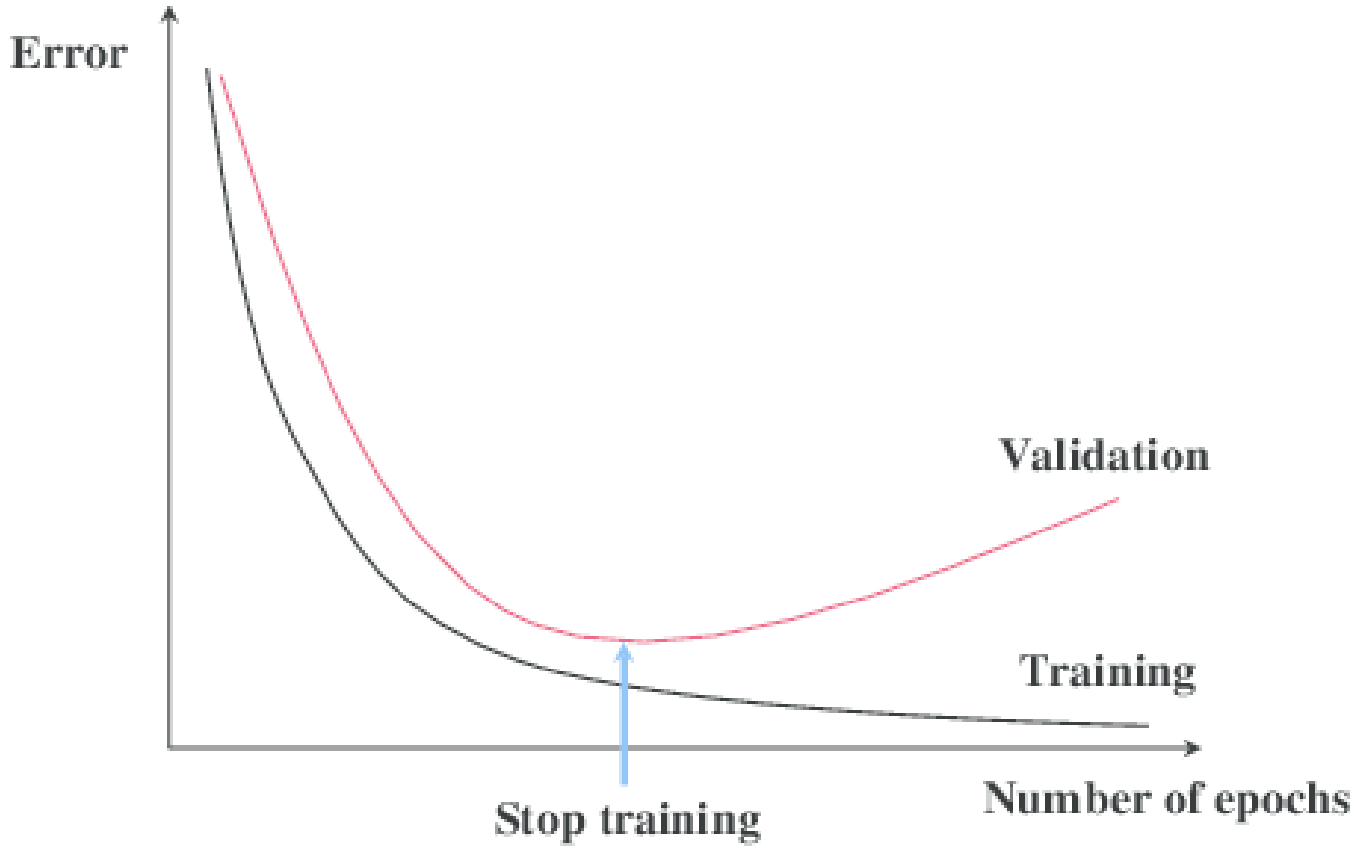
ℓ_0 complexity: Number of non-zero coefficients

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

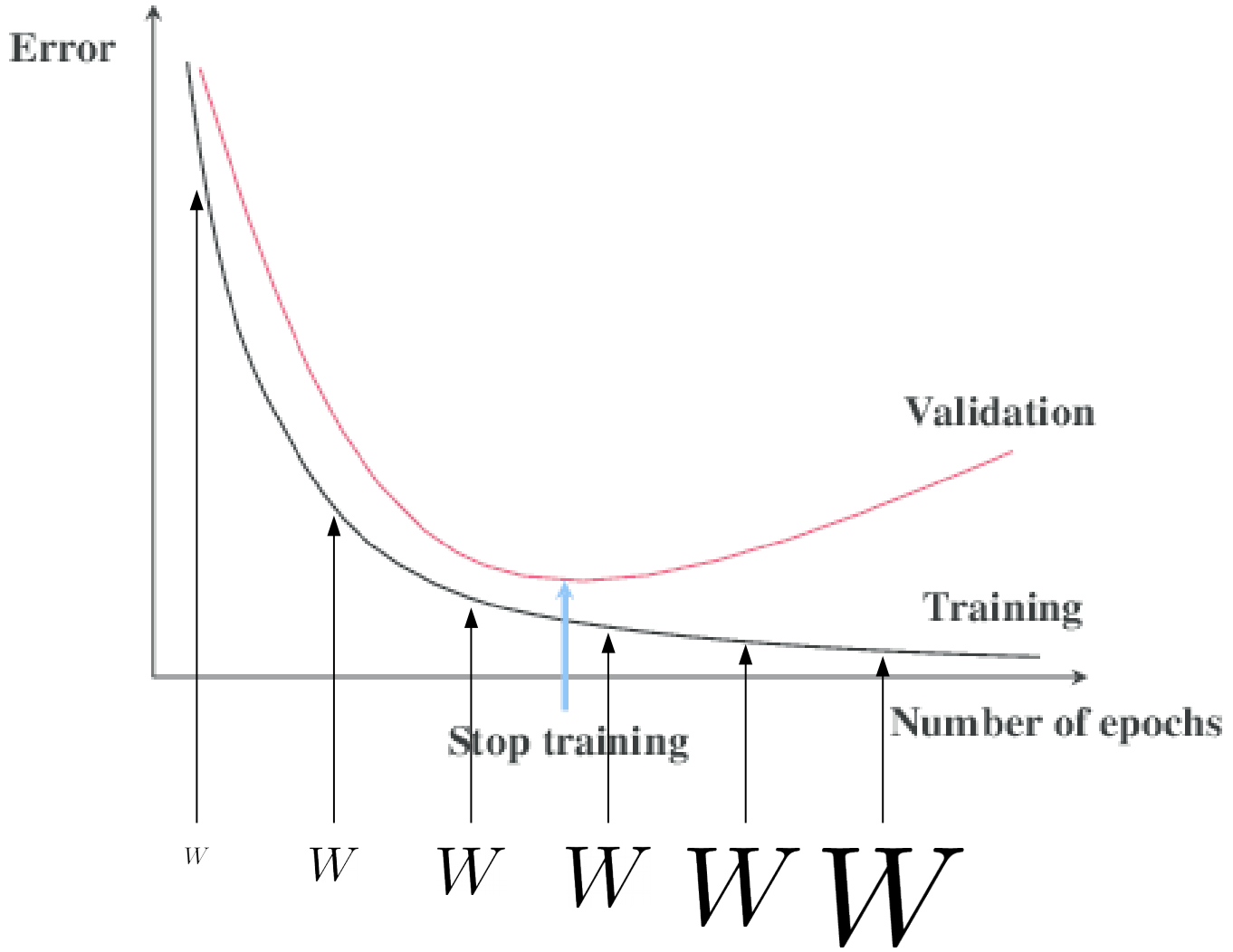


HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

Early Stopping

Feedforward Neural Networks

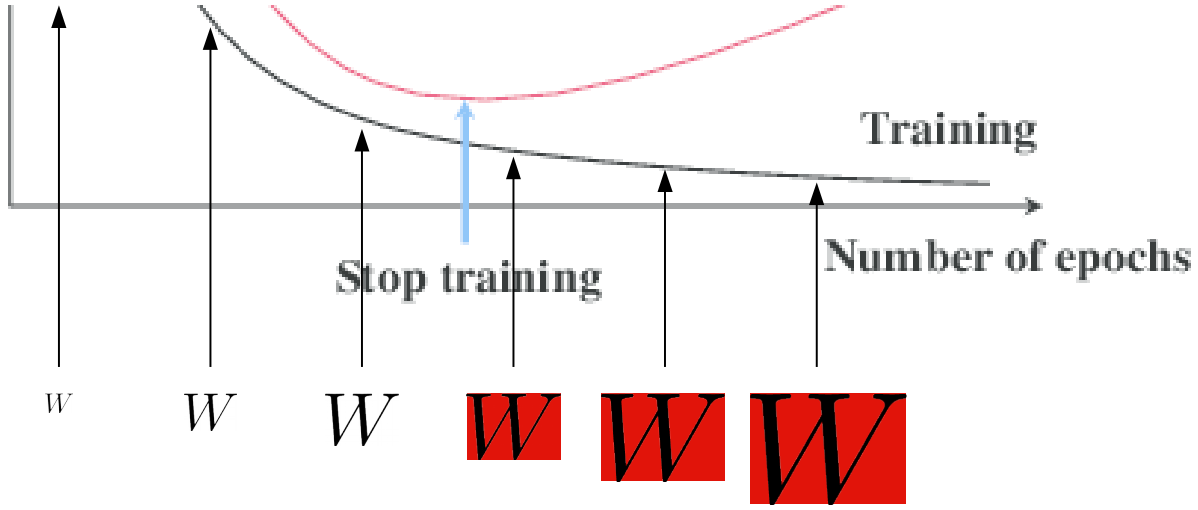


HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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?

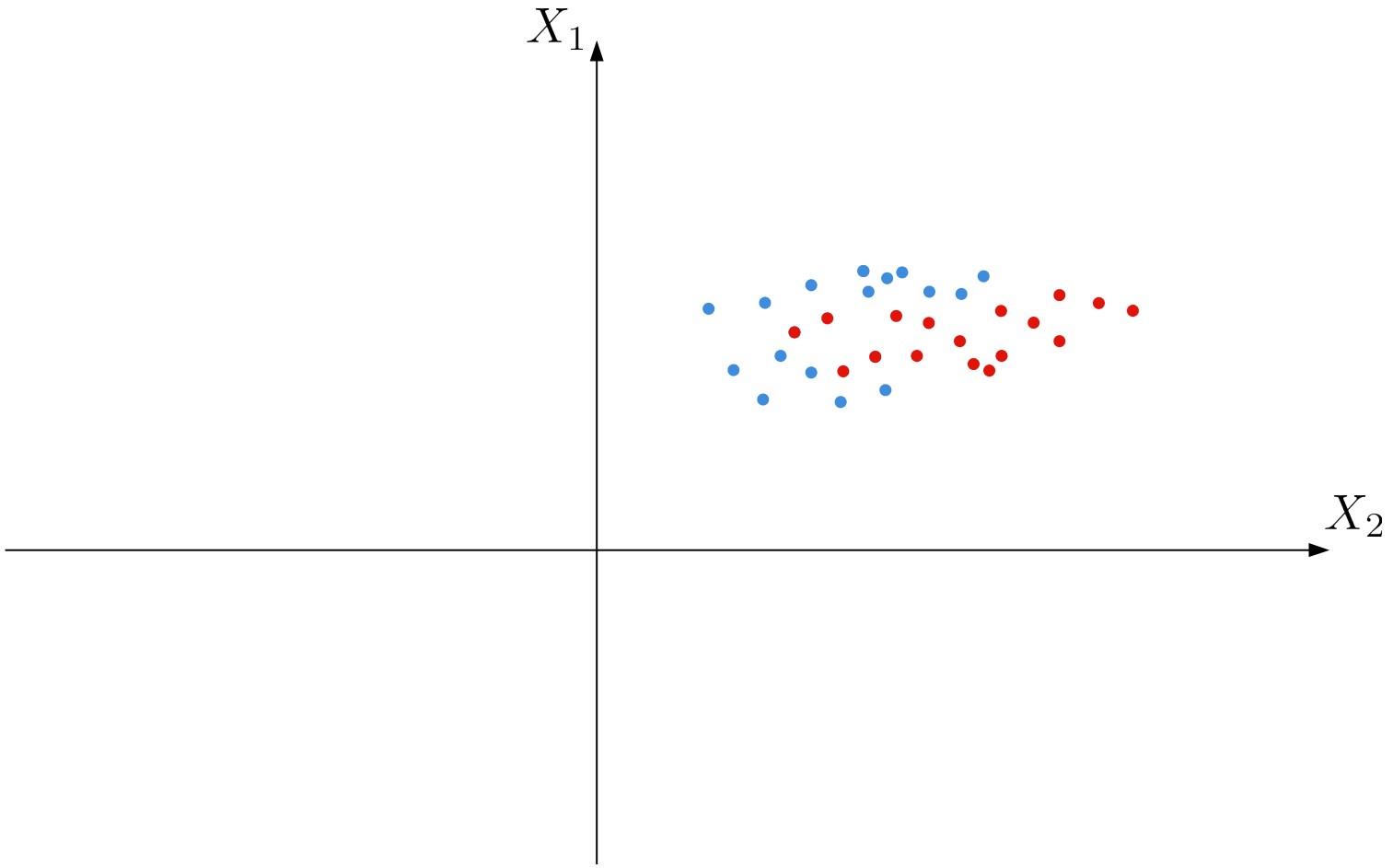
ℓ_1 "lasso" complexity: $\sum_{i=0}^{d} |w_i|$, for coefficients $w_0, ..., w_d$

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

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

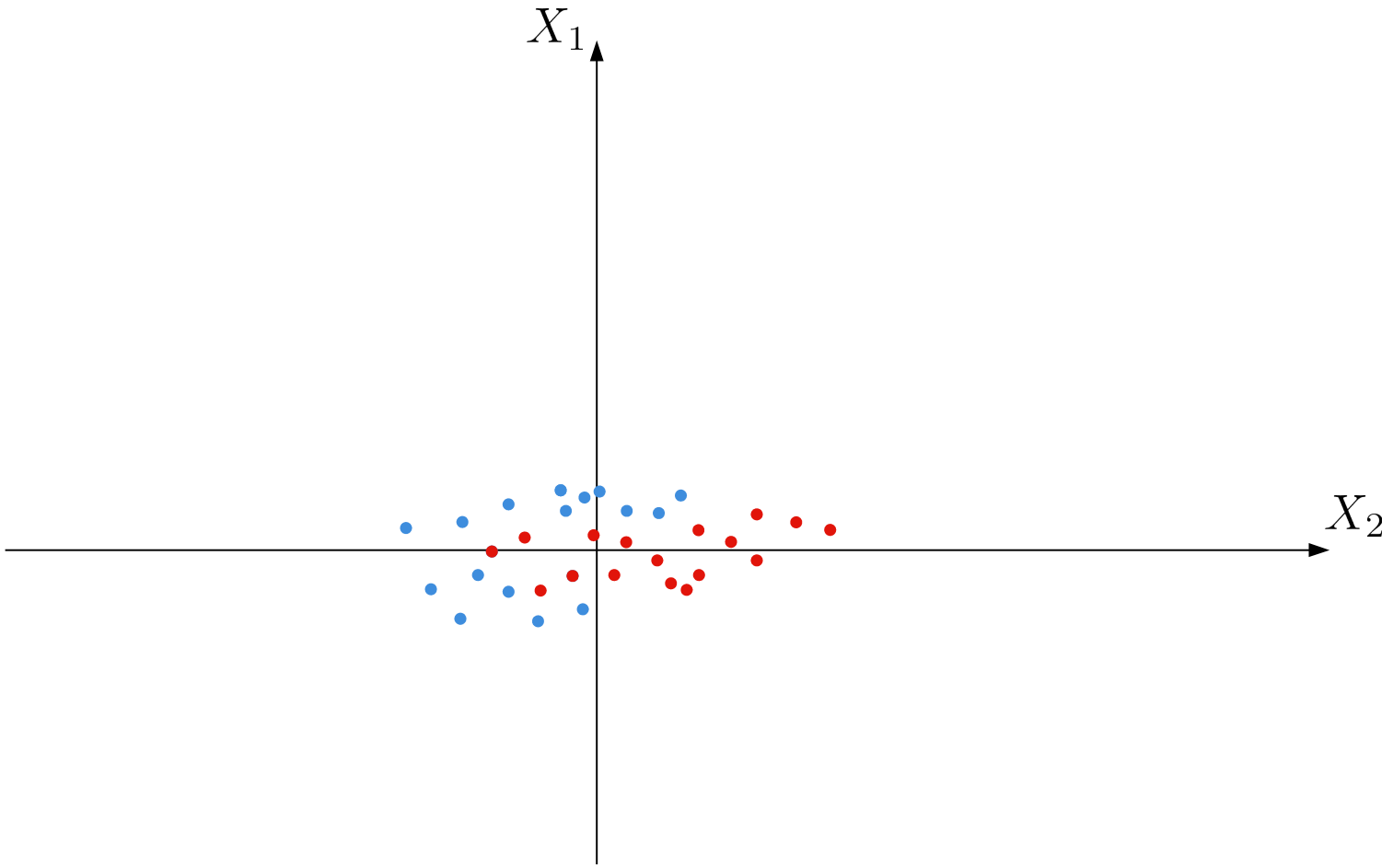


$$x = \frac{x - \mu}{\sigma^2}$$

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

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

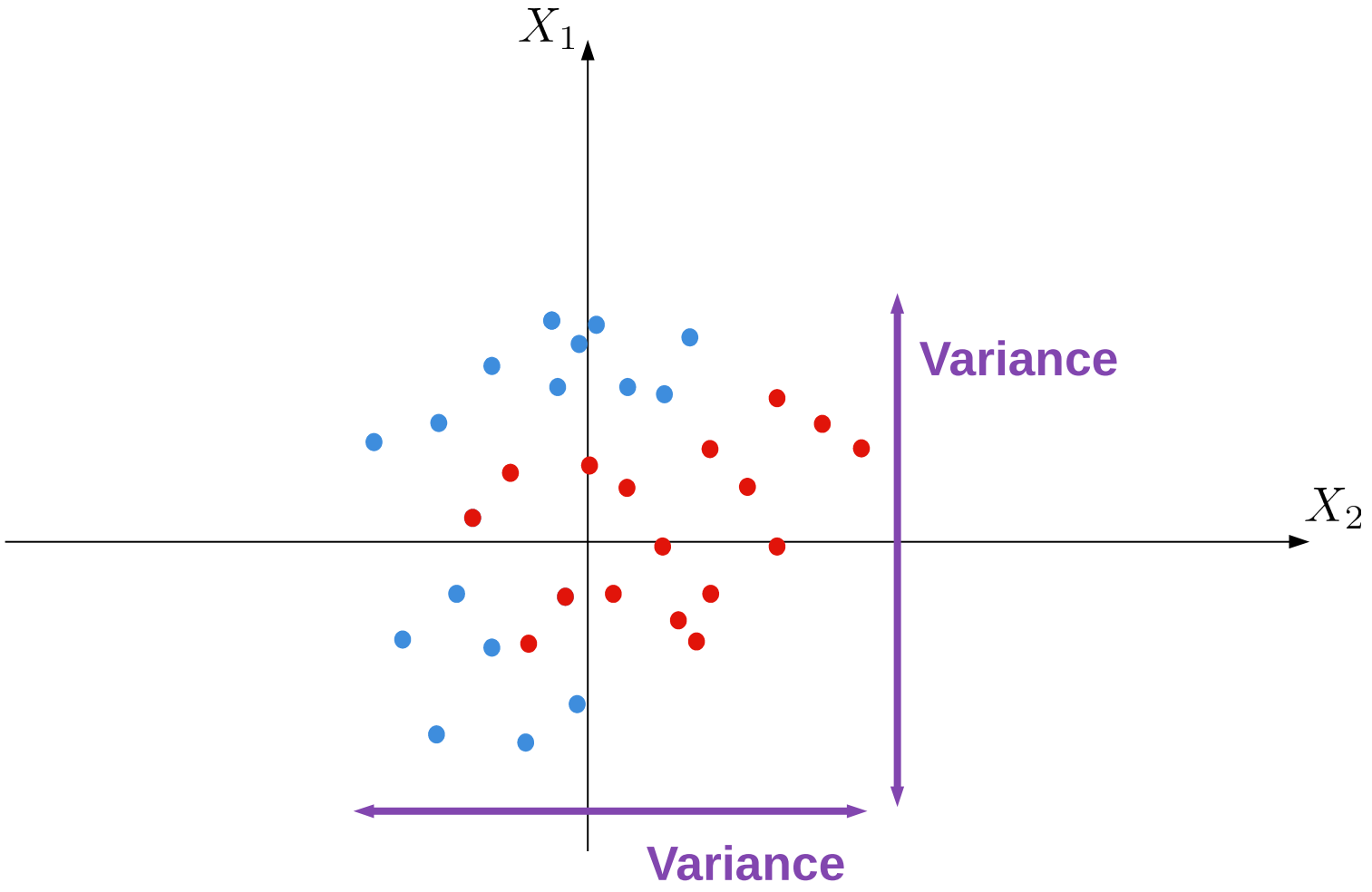


$$x = \frac{x - \mu}{\sigma^2}$$

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

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



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

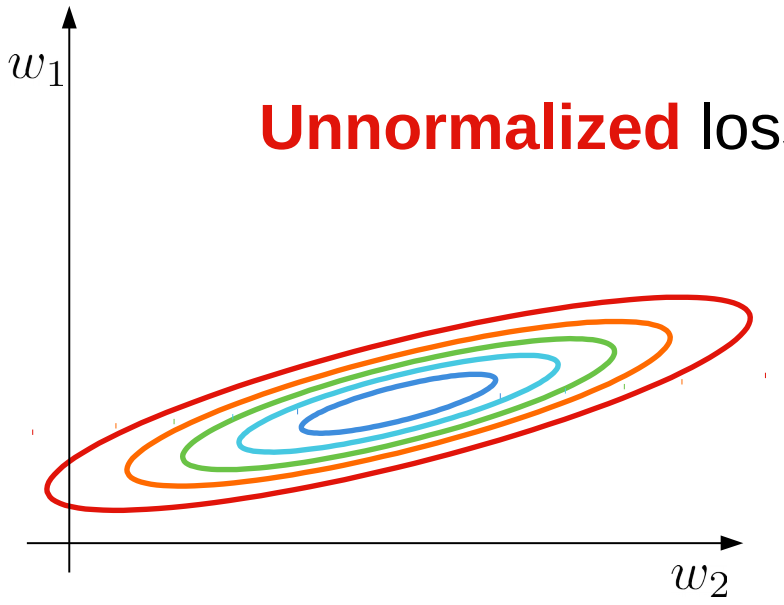
HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

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

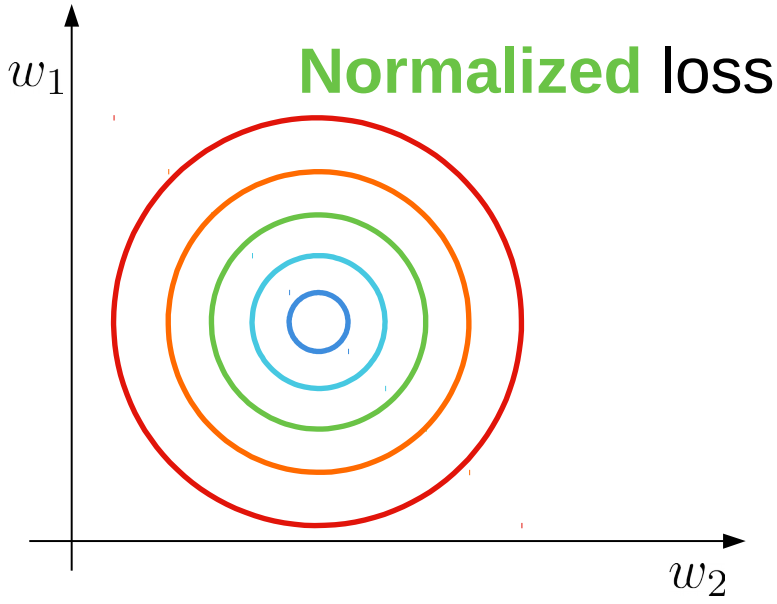
HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Why **input normalization** matters?



Unnormalized loss space

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



Normalized loss space

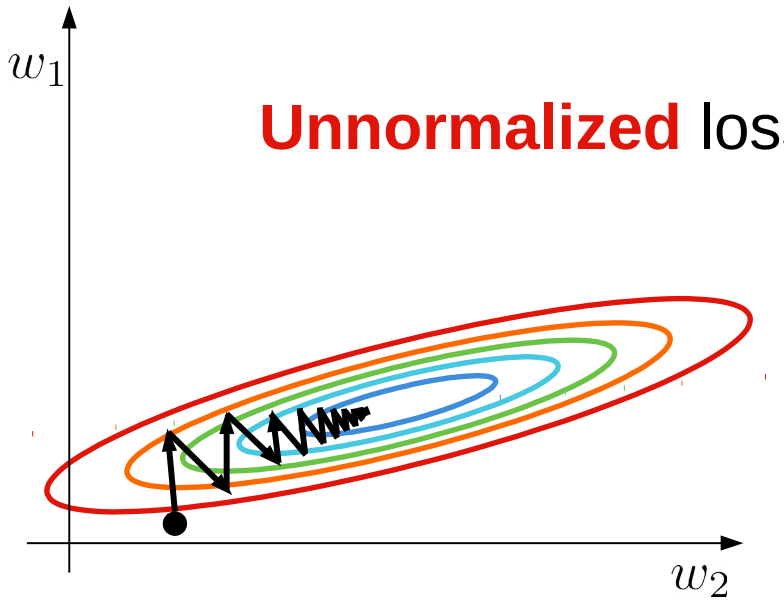
$$f_1 \in [-0.5, 0.5]$$
$$f_2 \in [-0.5, 0.5]$$

Feedforward Neural Networks

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

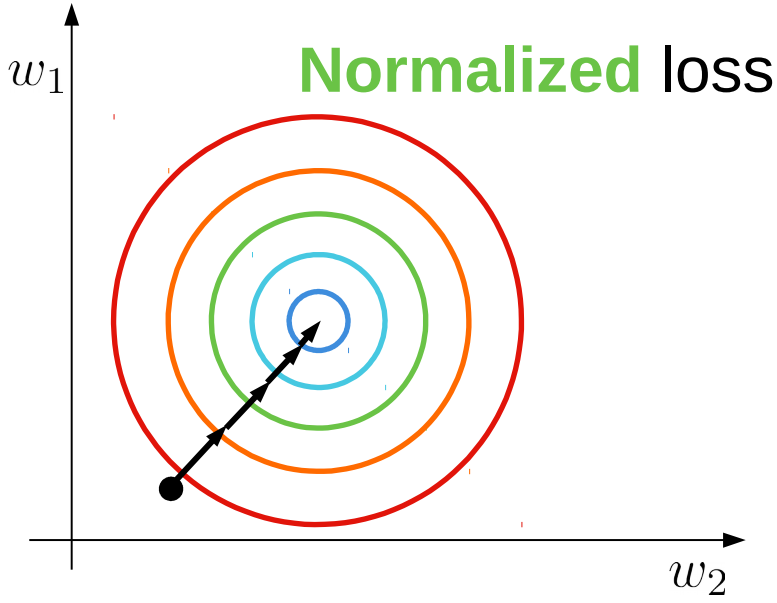
HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Why **input normalization** matters?



Unnormalized loss space

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

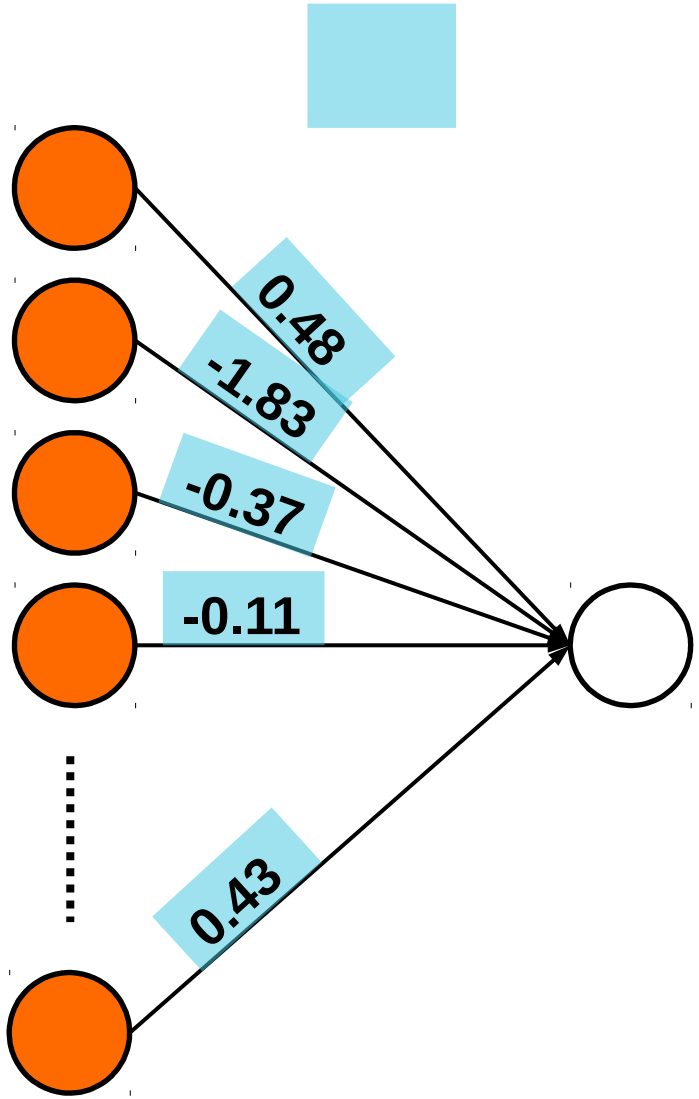


Normalized loss space

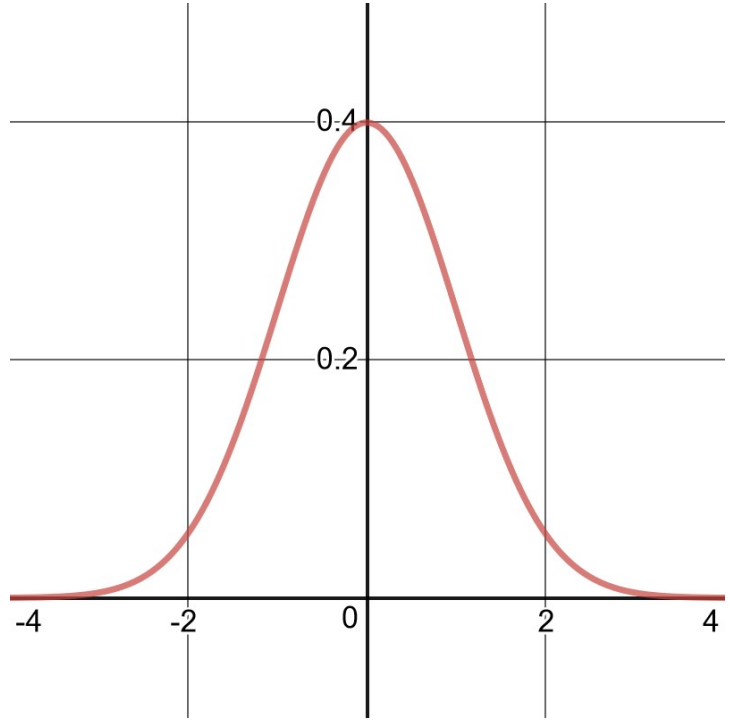
$$f_1 \in [-0.5, 0.5]$$
$$f_2 \in [-0.5, 0.5]$$

Feedforward Neural Networks

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

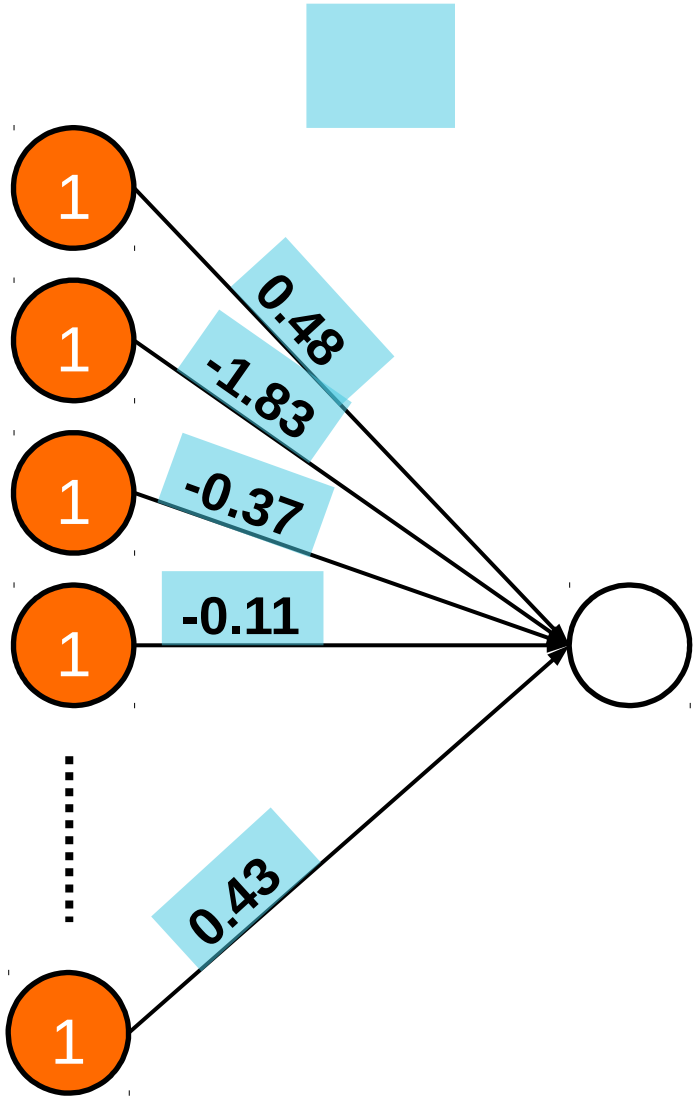


Normal Distribution

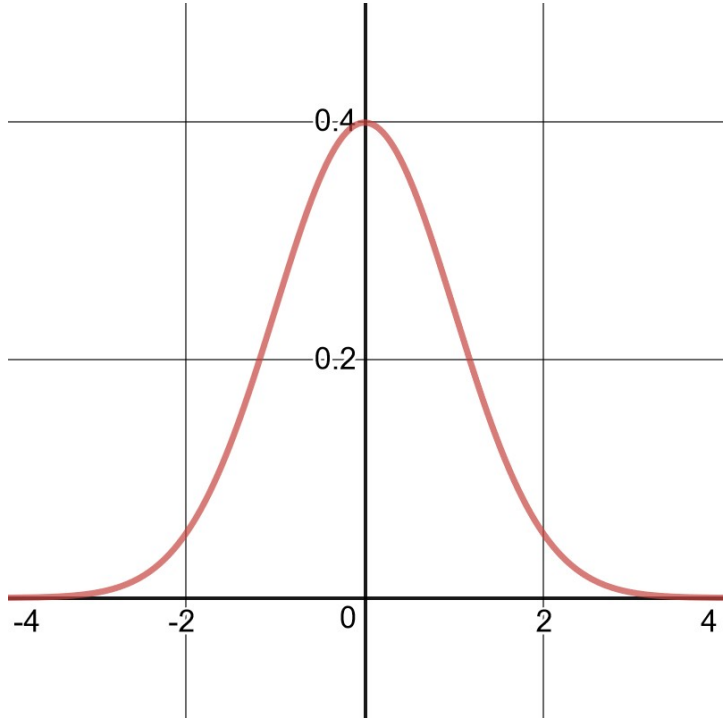


HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks



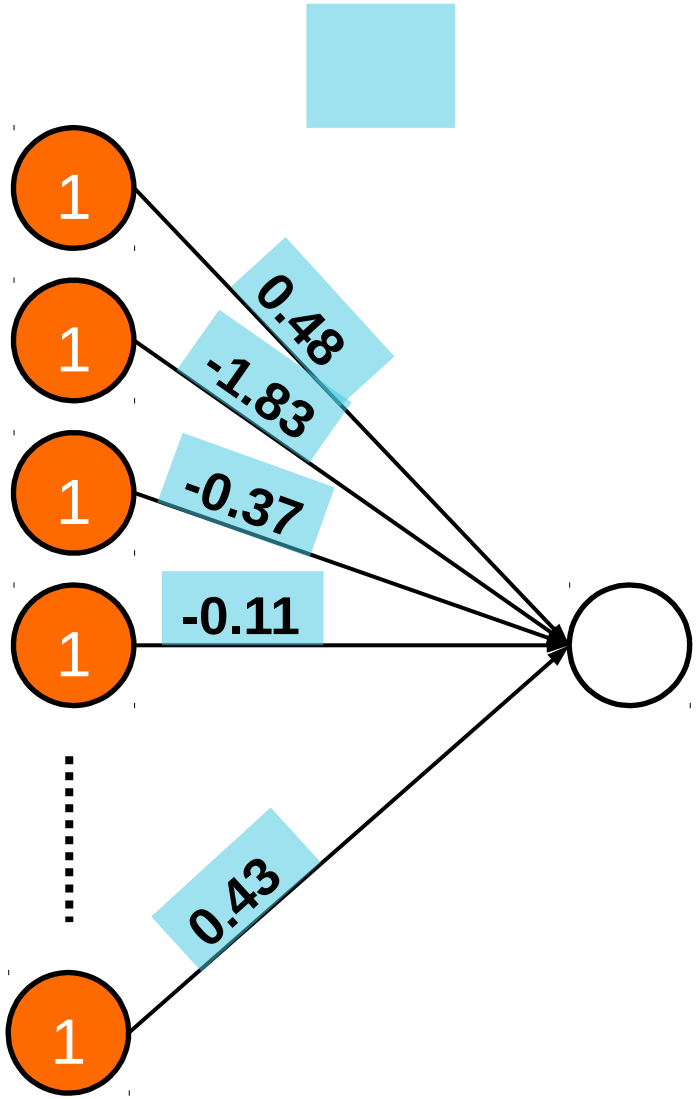
Normal Distribution



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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

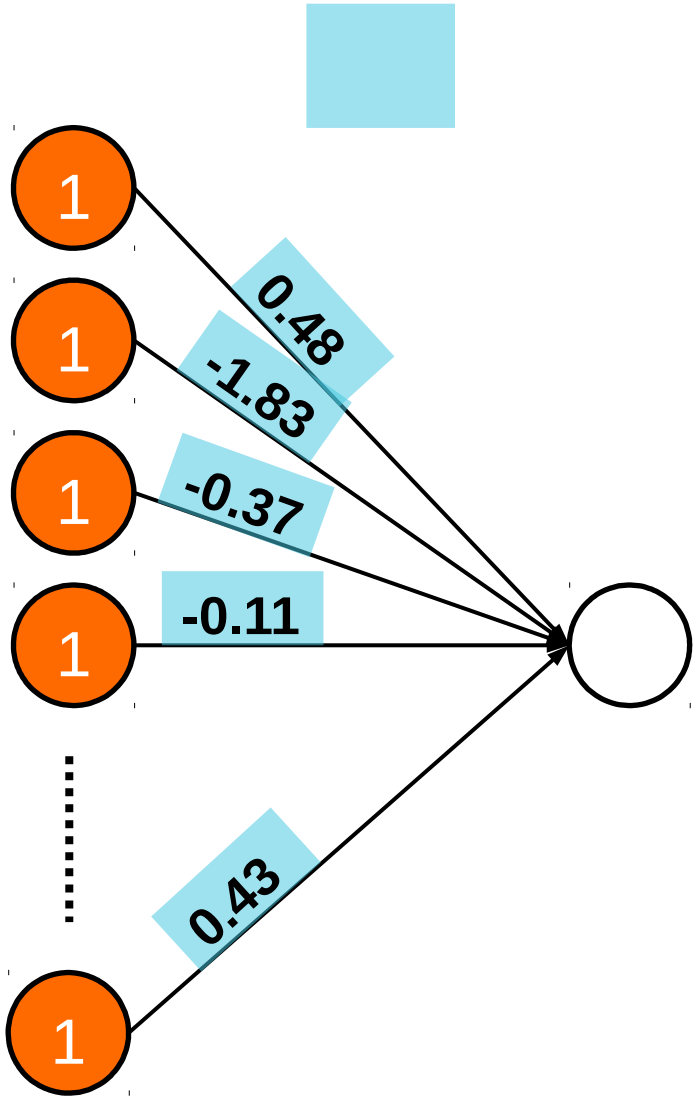


Sum of **independent random variables** that are **normally distributed**:

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

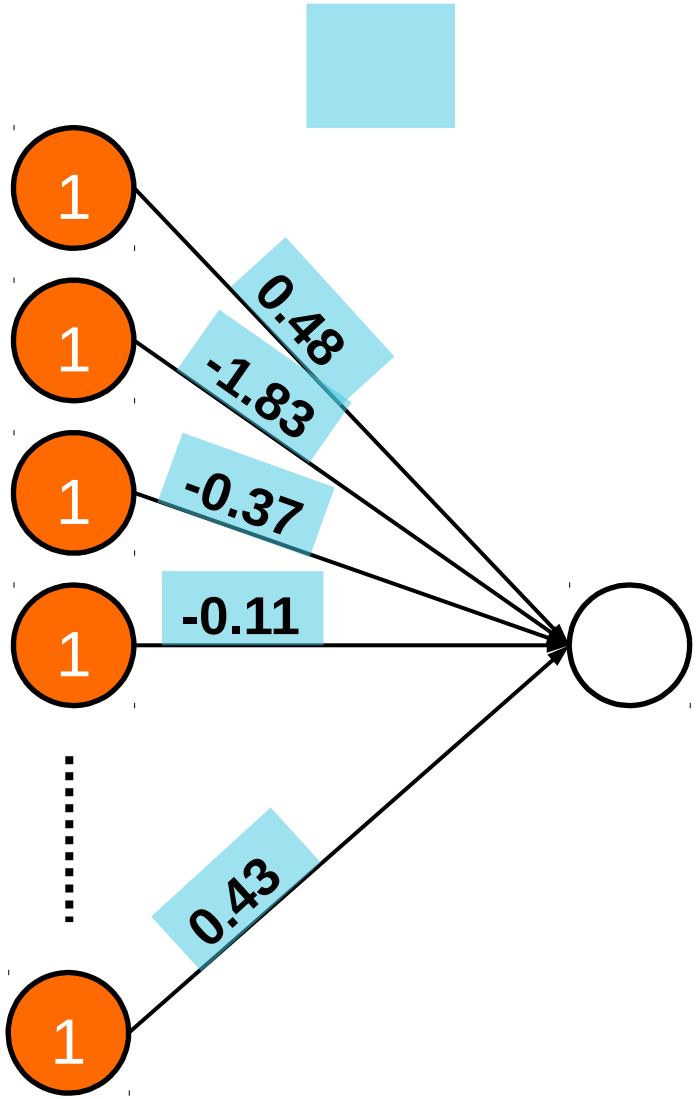


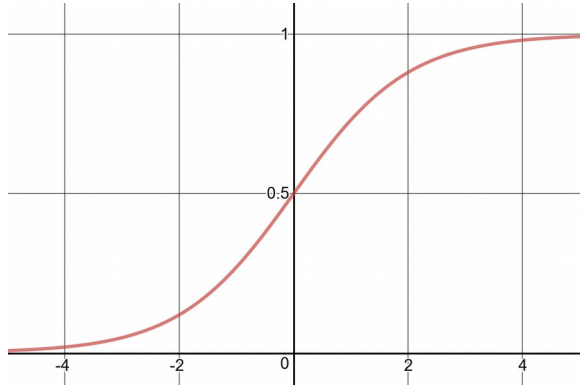
Sum of **independent random variables** that are **normally distributed**:

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks

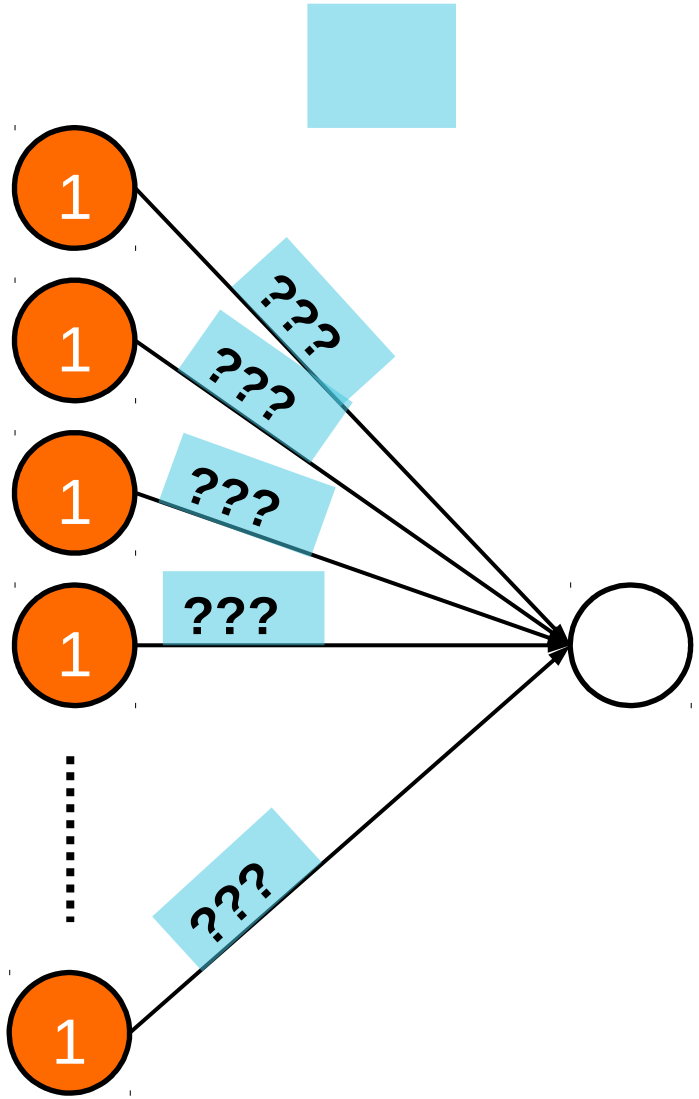




SGD, Epochs, Batches and Steps
Activation functions
SGD learning rate
Other optimization methods
Regularization
Normalizing inputs
Vanishing/Exploding Gradients
Weights initialization

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Feedforward Neural Networks



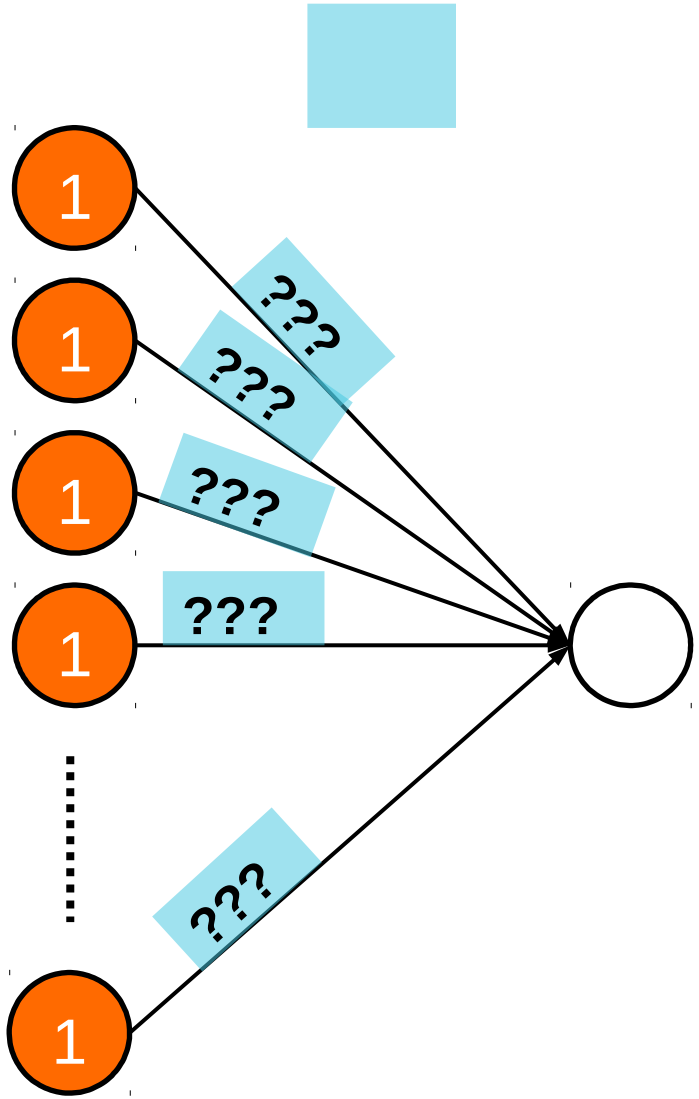
Alternative weights initializations?

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

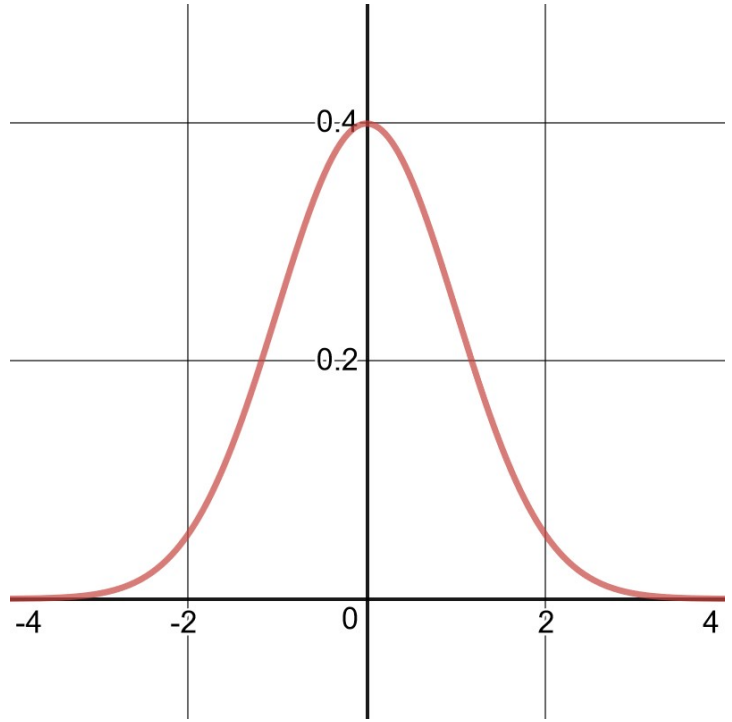
Feedforward Neural Networks

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



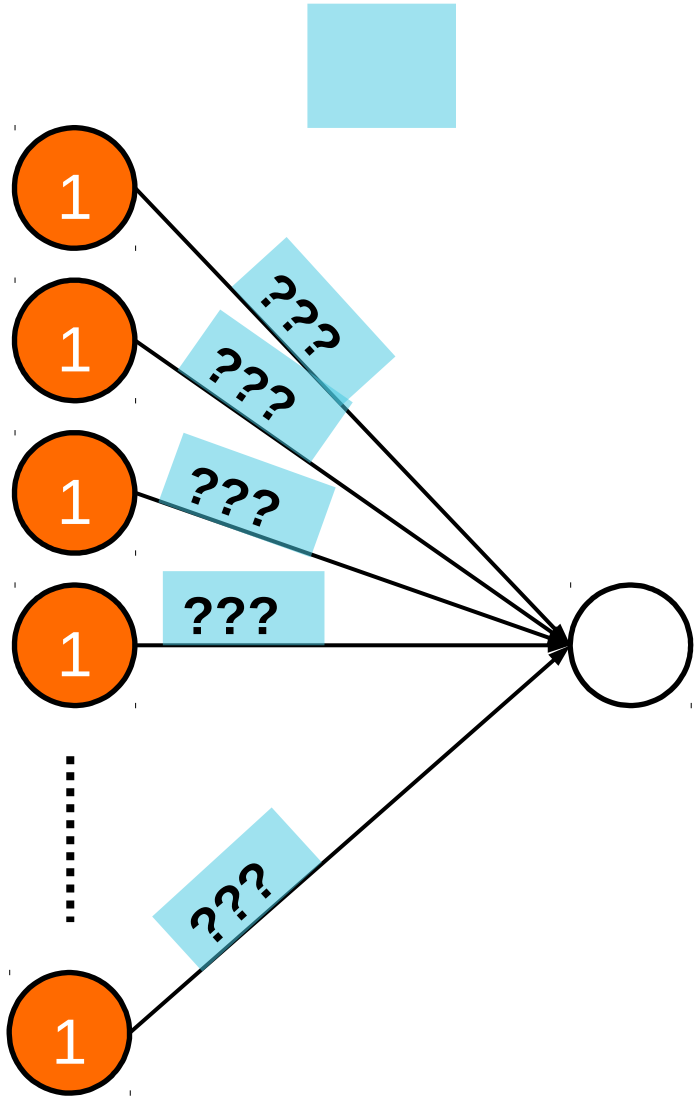
Truncated Normal Initialization

Normal Distribution



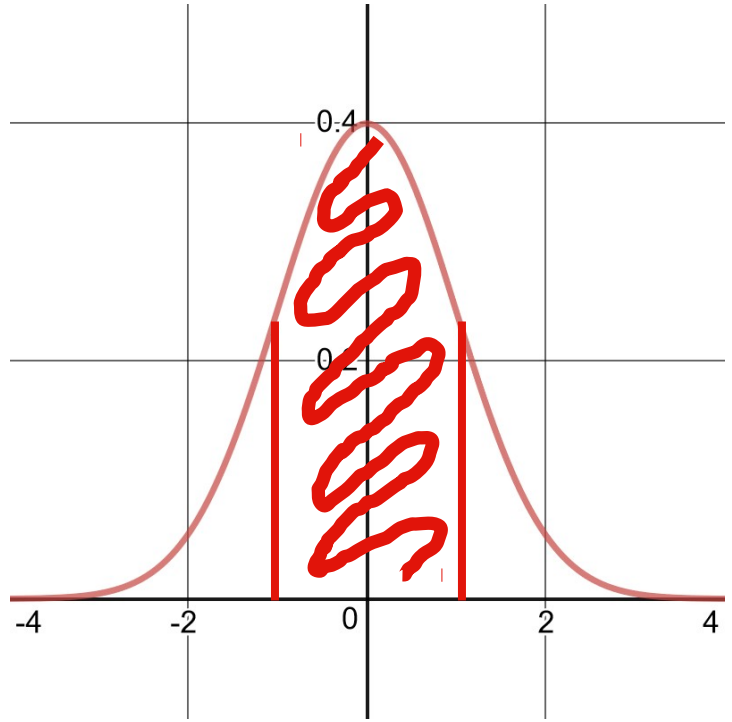
Feedforward Neural Networks

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



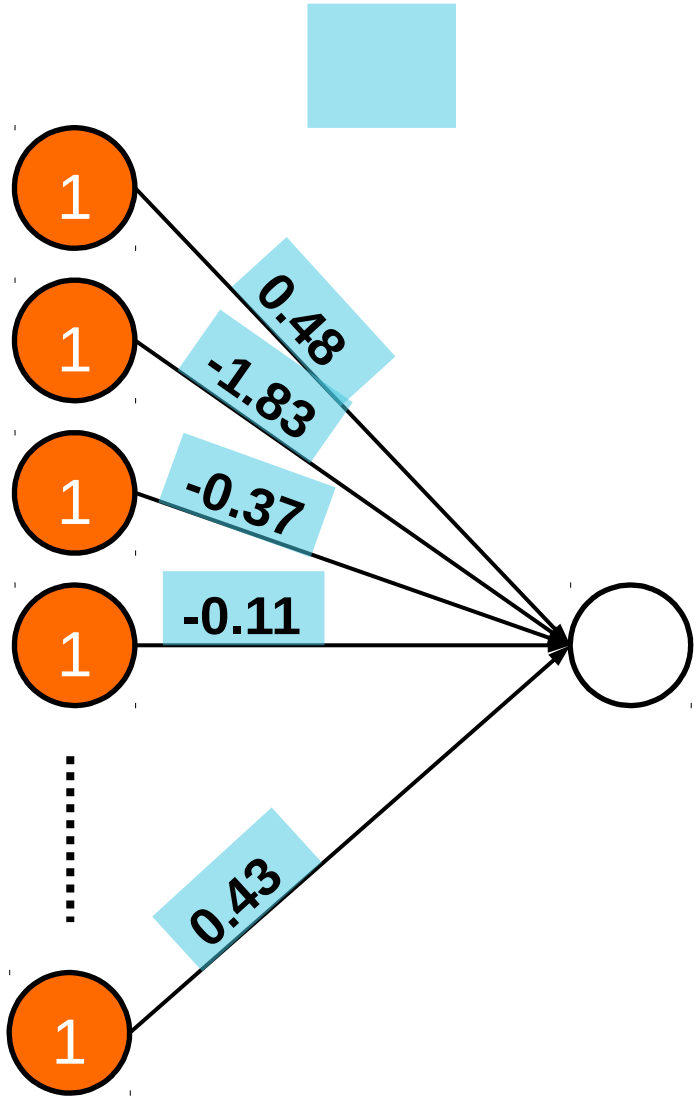
Truncated Normal Initialization

Normal Distribution



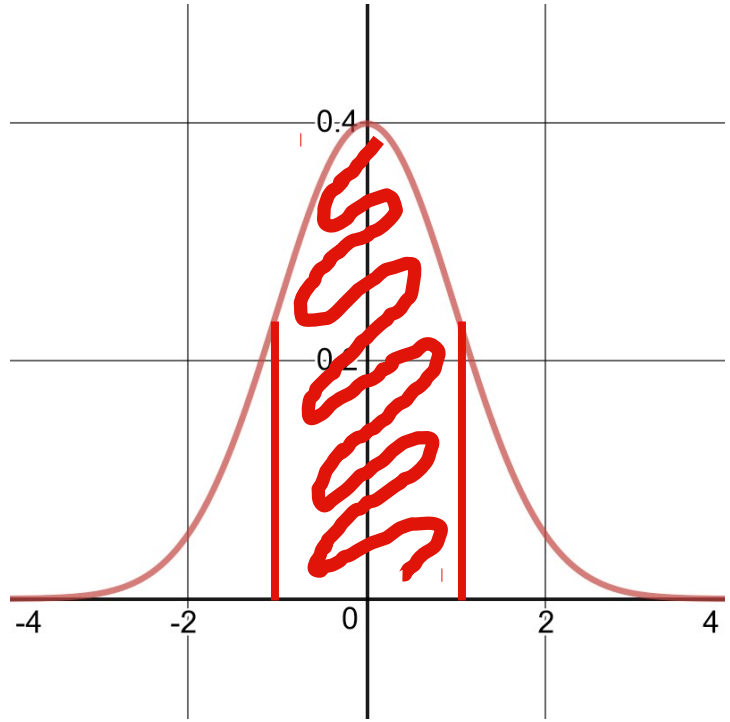
Feedforward Neural Networks

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



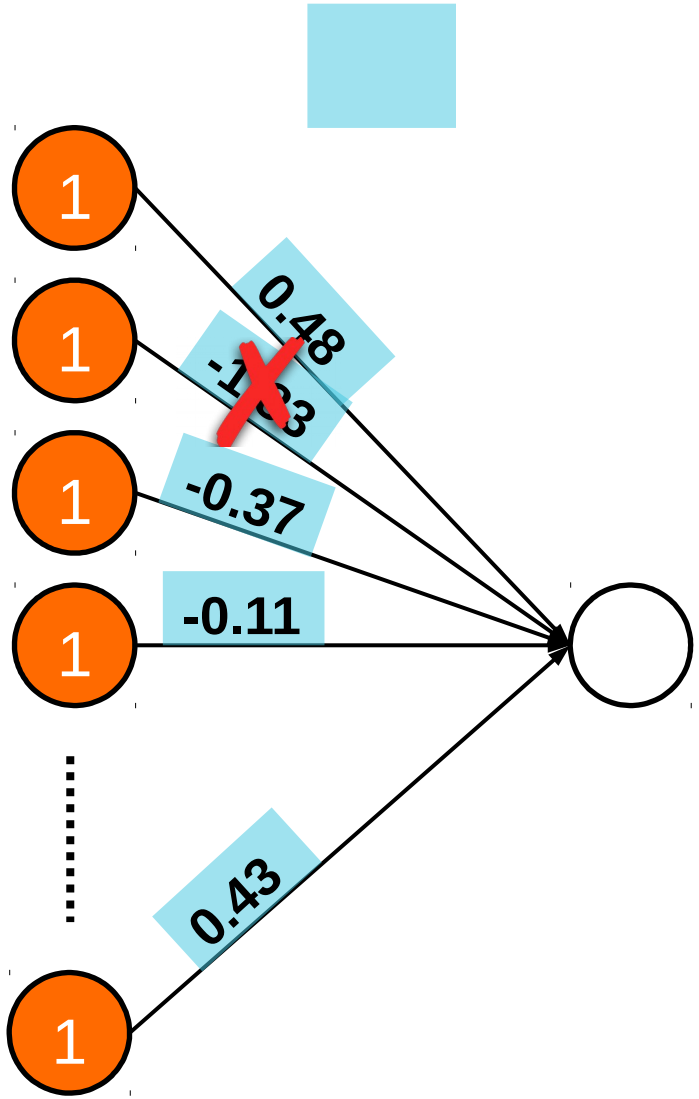
Truncated Normal Initialization

Normal Distribution



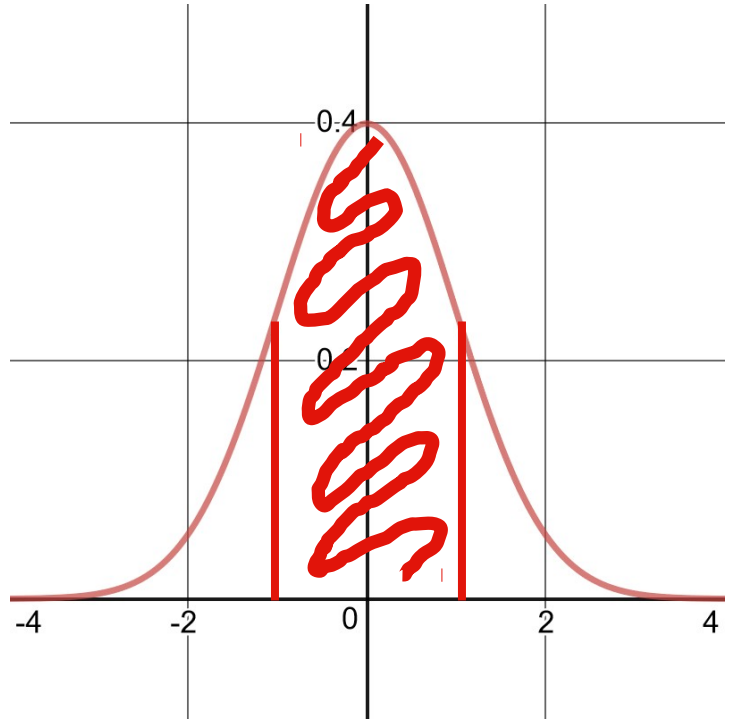
Feedforward Neural Networks

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



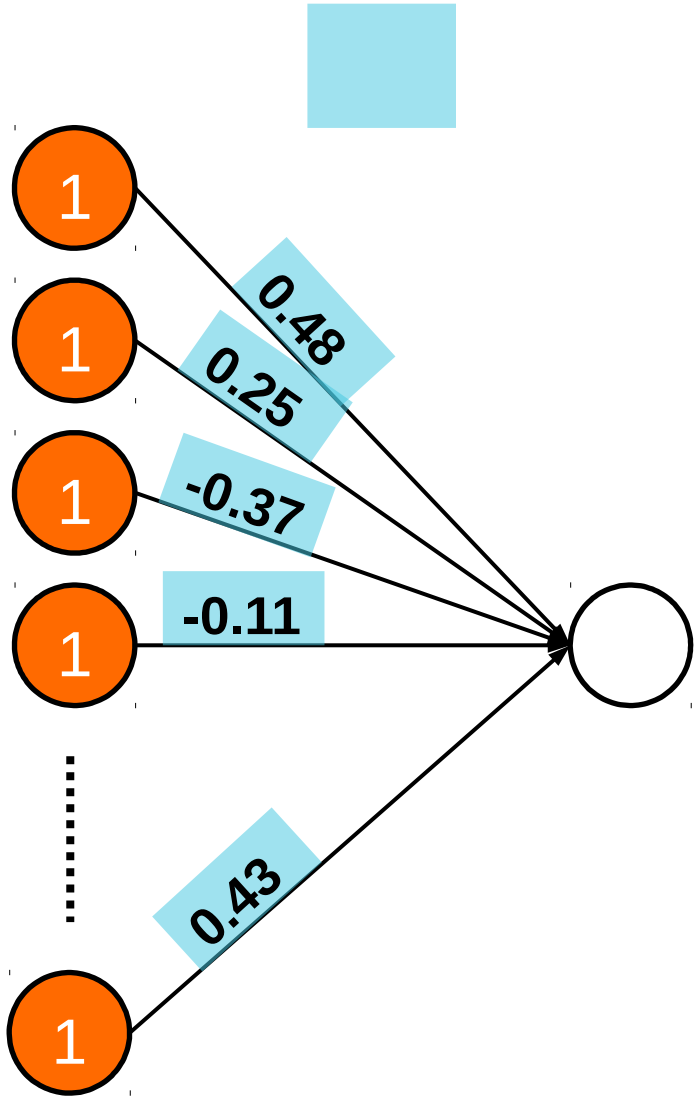
Truncated Normal Initialization

Normal Distribution



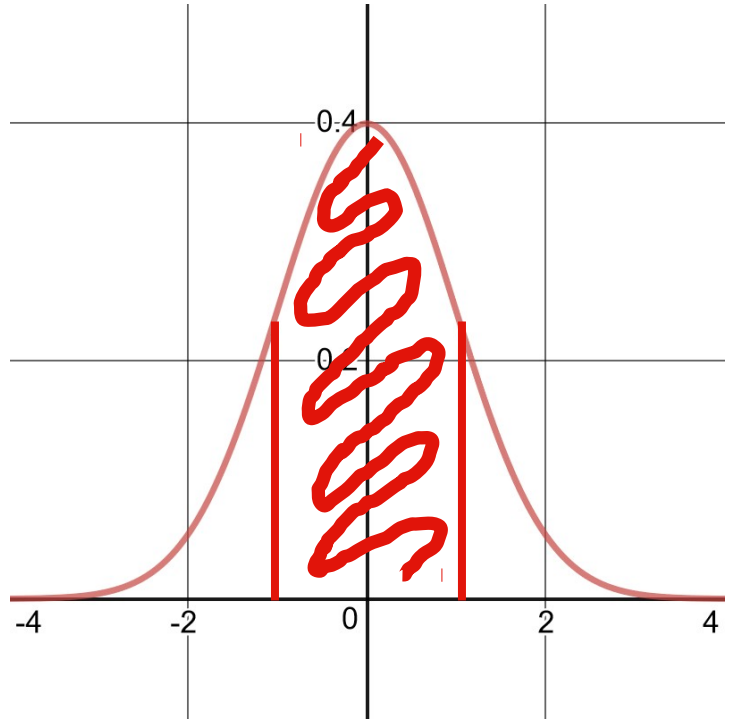
Feedforward Neural Networks

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



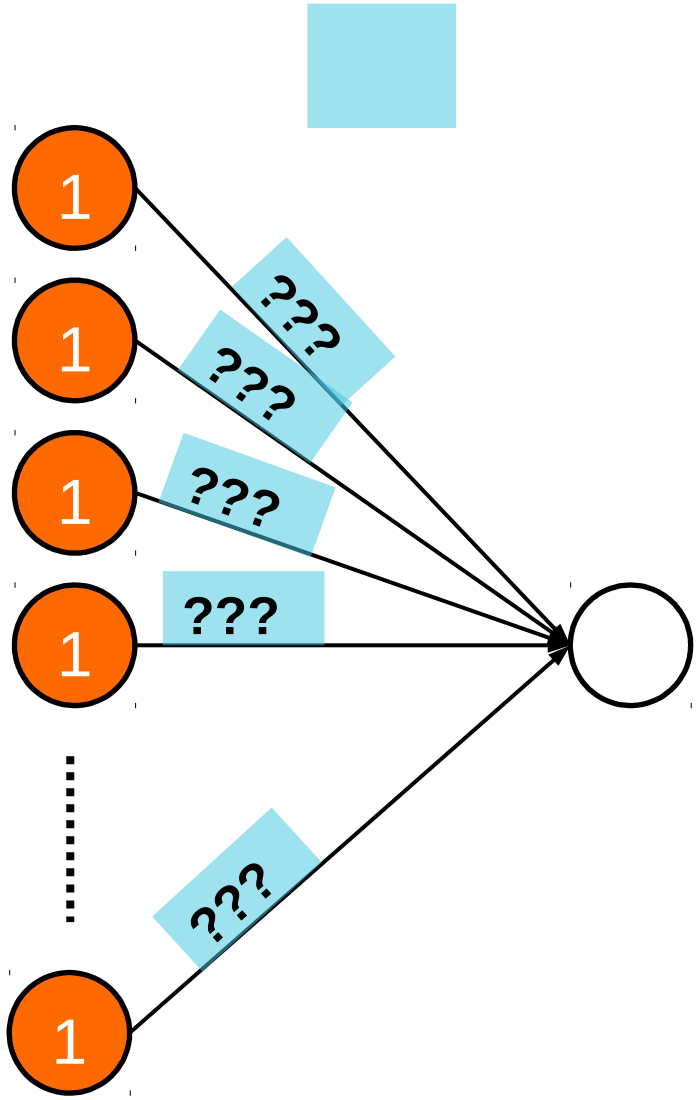
Truncated Normal Initialization

Normal Distribution



Feedforward Neural Networks

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

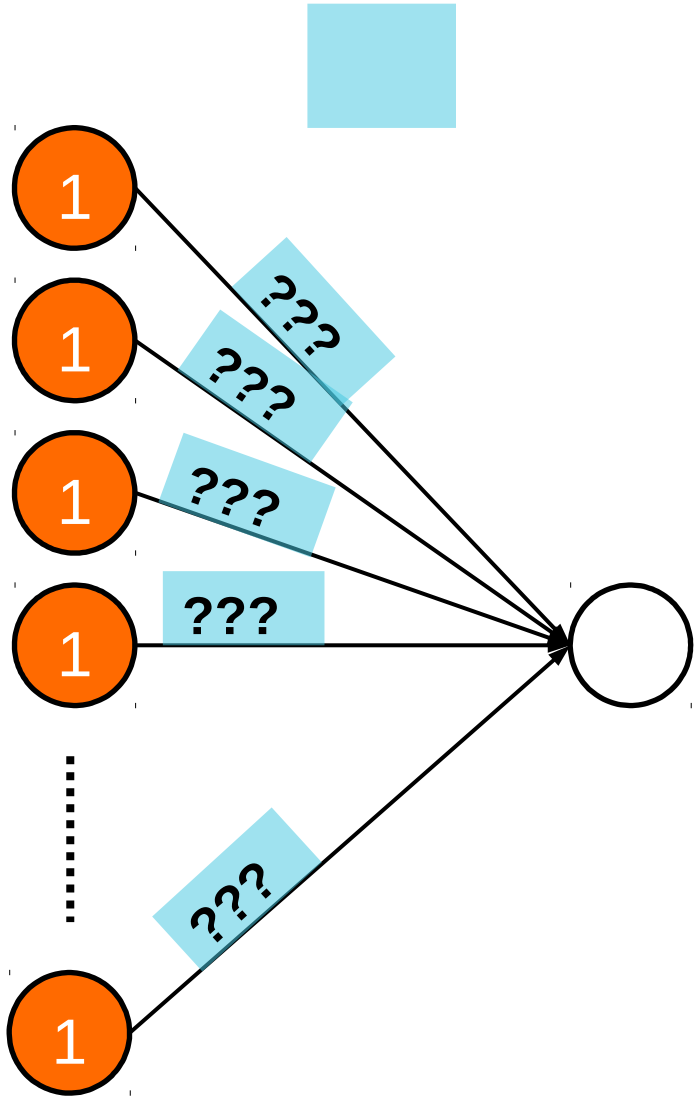


Xavier / Glorot Initialization

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

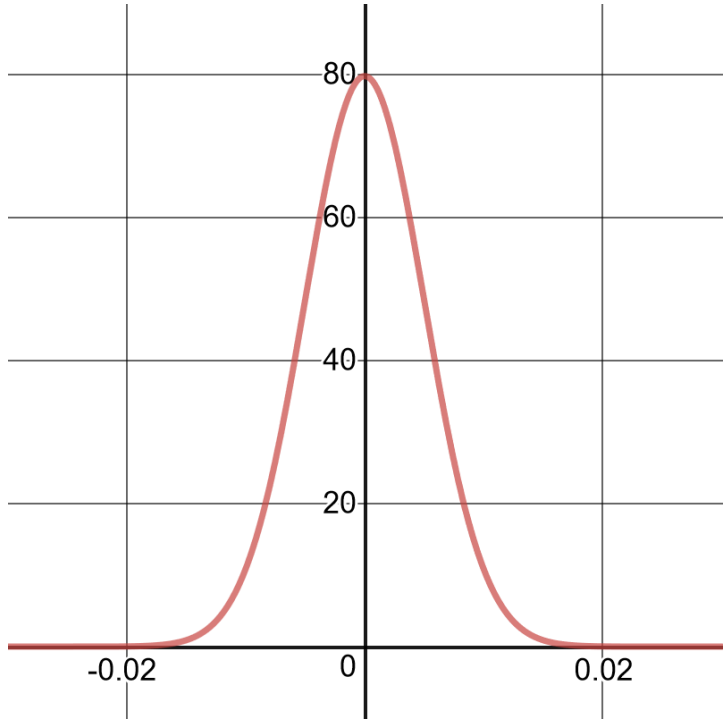
Feedforward Neural Networks

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

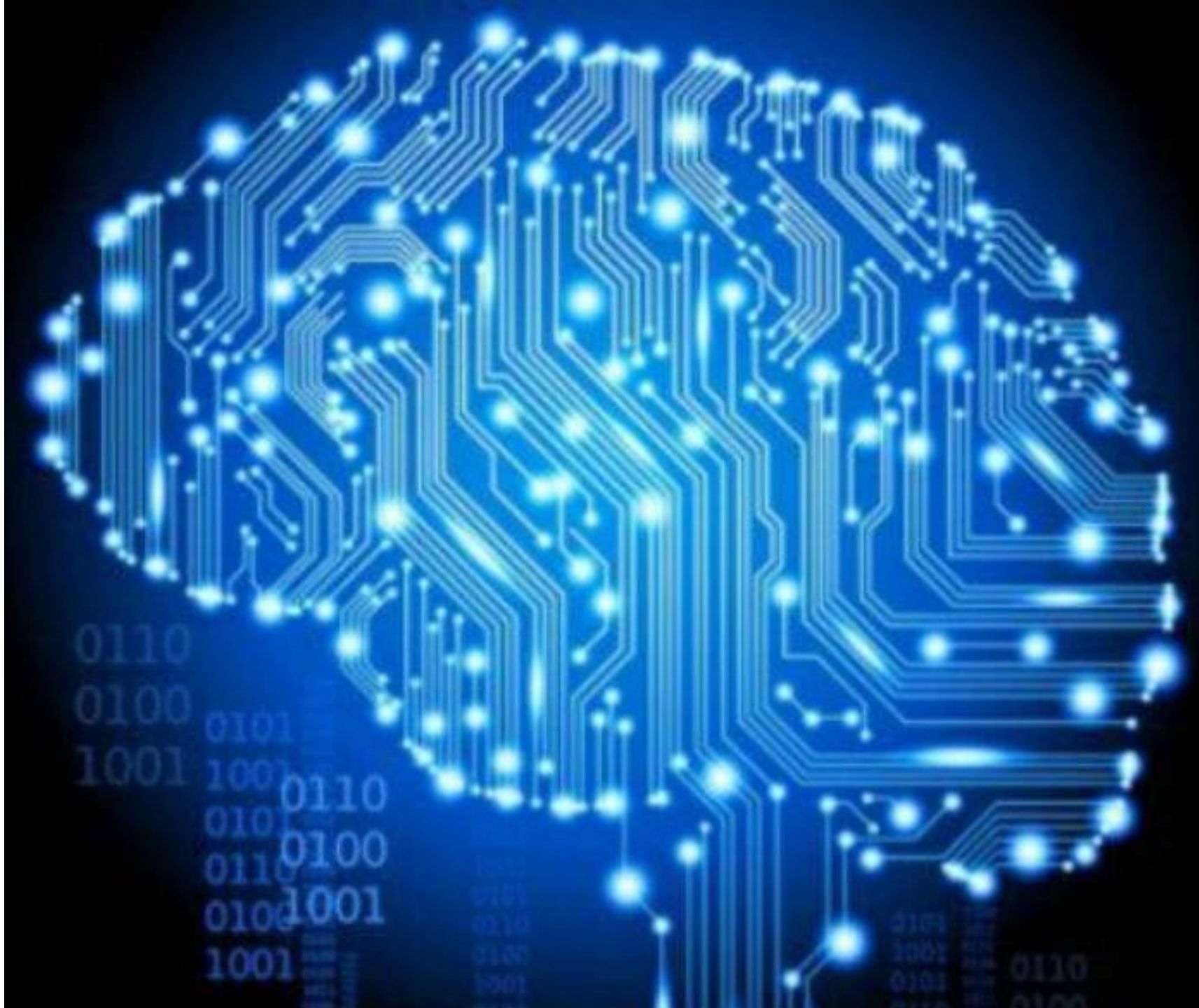


Xavier / Glorot Initialization

Normal Distribution

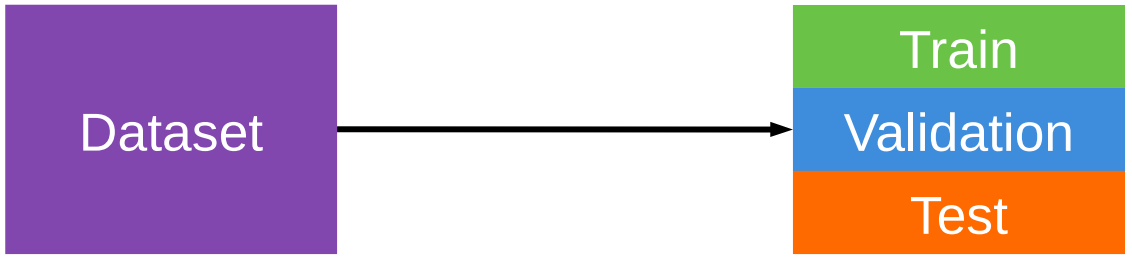


Practical Aspects



HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE

Practical Aspects



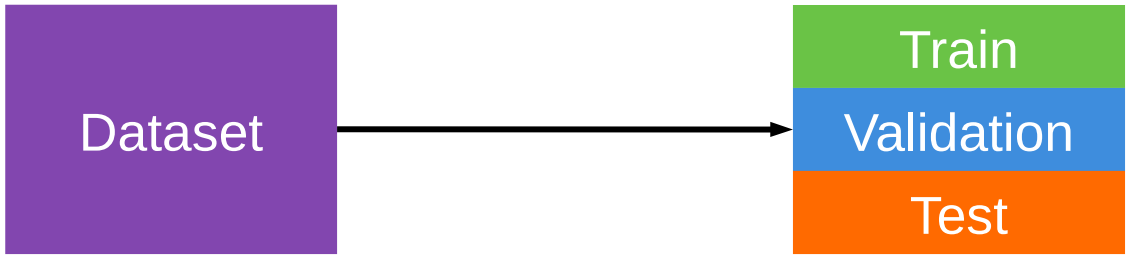
Train / Validation / Test Workflow

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Practical Aspects

Train / Validation / Test Workflow



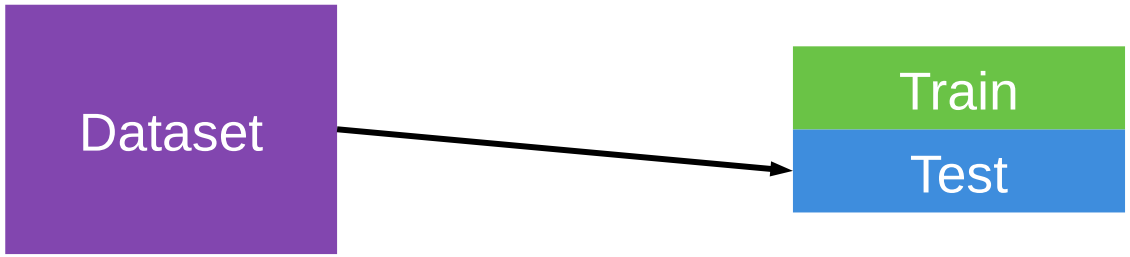
Train — Set used to **train & control bias**

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

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

Test — Set used to **control variance**

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

OPTIONAL

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Practical Aspects

Machine Learning

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

Train / Validation / Test Workflow

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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

HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE

Practical Aspects

Train / Validation / Test Workflow

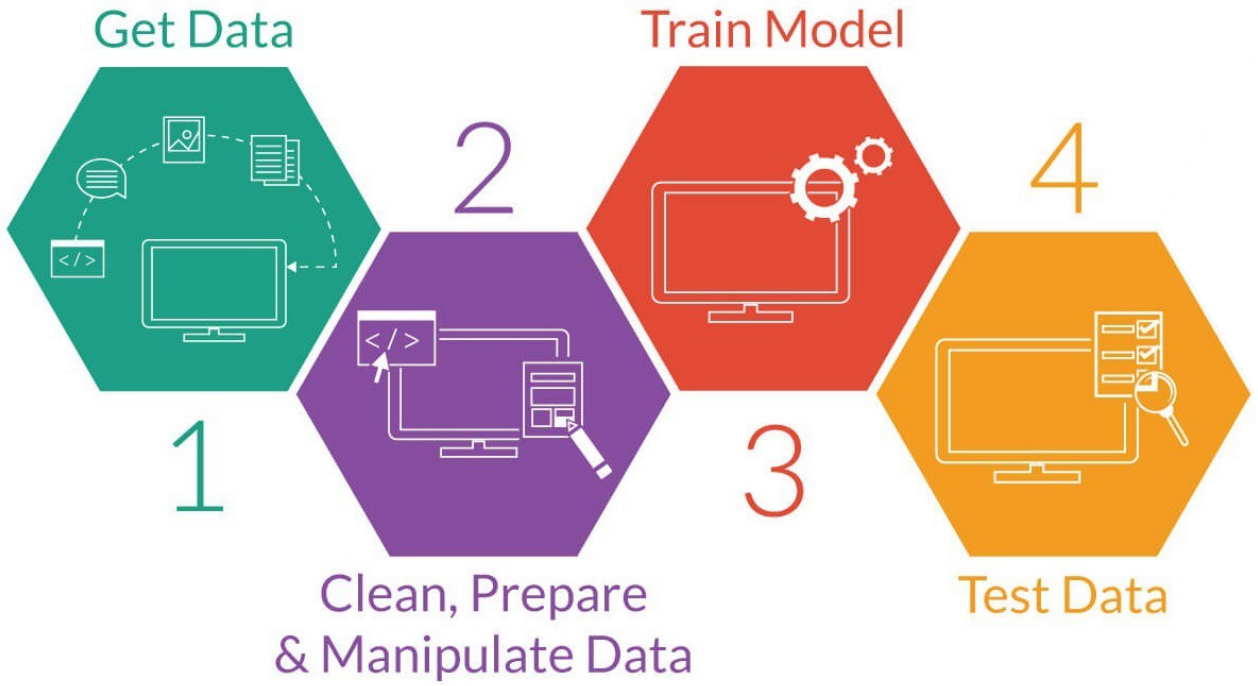
Regardless of what happens, **never never ever**

- Mix correlated data in train/val/test
    - Not straightforward!
- Use different sources for train/val/test
    - Shuffle is your friend
- Perform unreproducible results
    - Seed is a good friend of Shuffle, which makes it your friend too
- Trust test results only
- Trust validation results only if you have done thorough hyperparameter tuning

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Practical Aspects

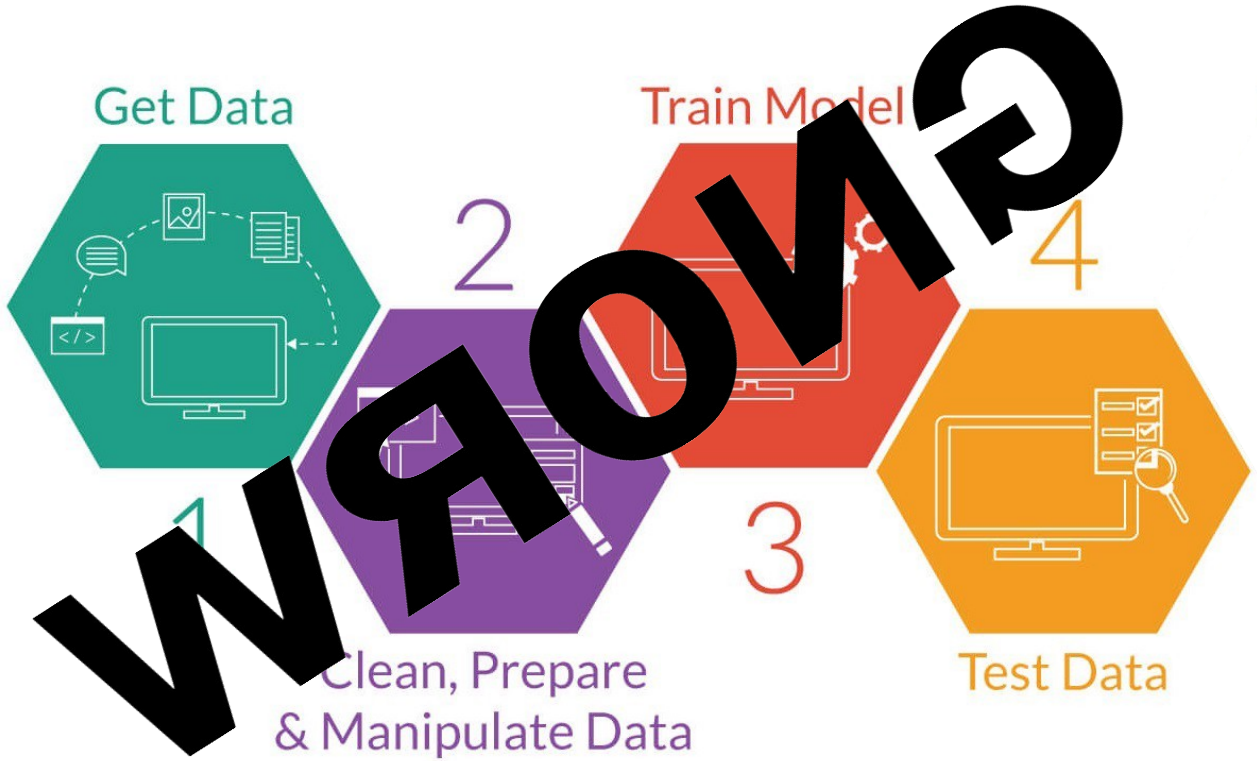
Train /
Validation / Test
Workflow



HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE

Practical Aspects

Train / Validation / Test Workflow

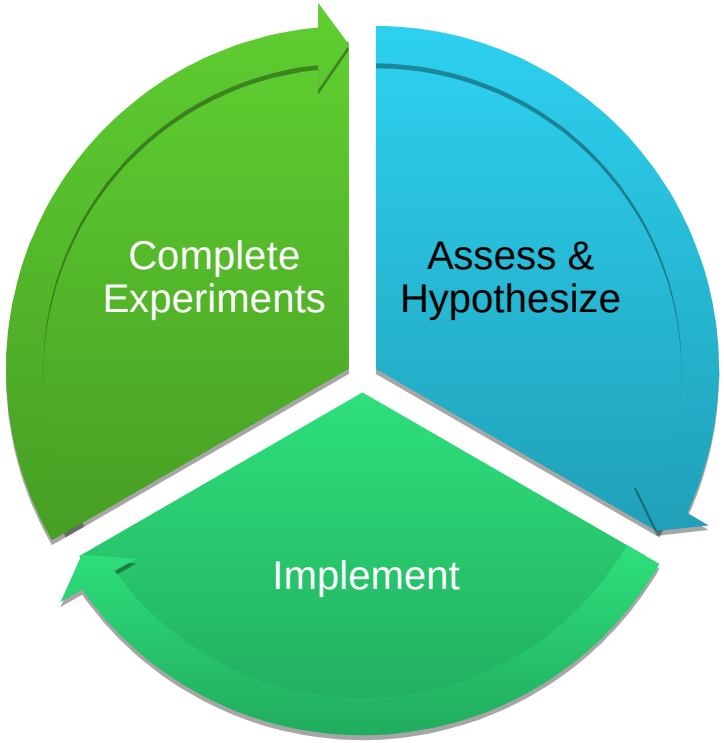


Practical Aspects

Train / Validation / Test Workflow

Lots of hyper-parameters:

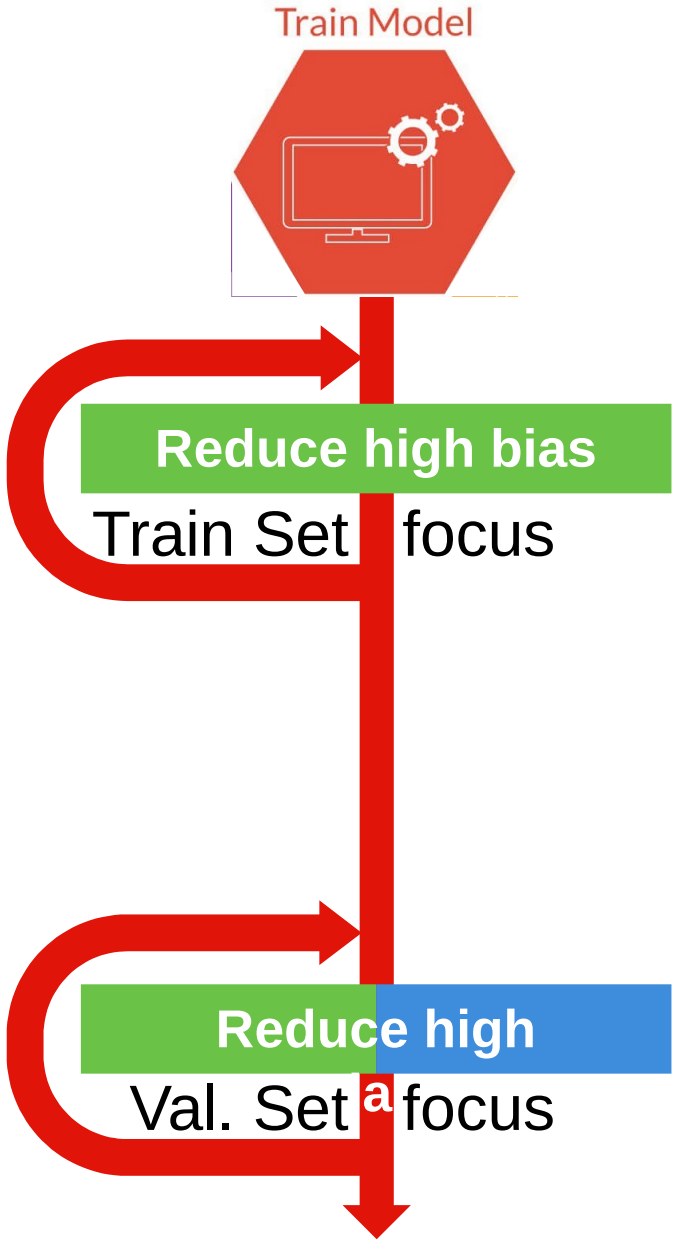
- Network architecture:
  - # layers
  - # neurons
  - Activation function
  - ...
- Learning rate
- Optimization algorithm
- Generalization methods
- Initialization
- Data pre-processing
- ....



HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Practical Aspects

Train / Validation / Test Workflow

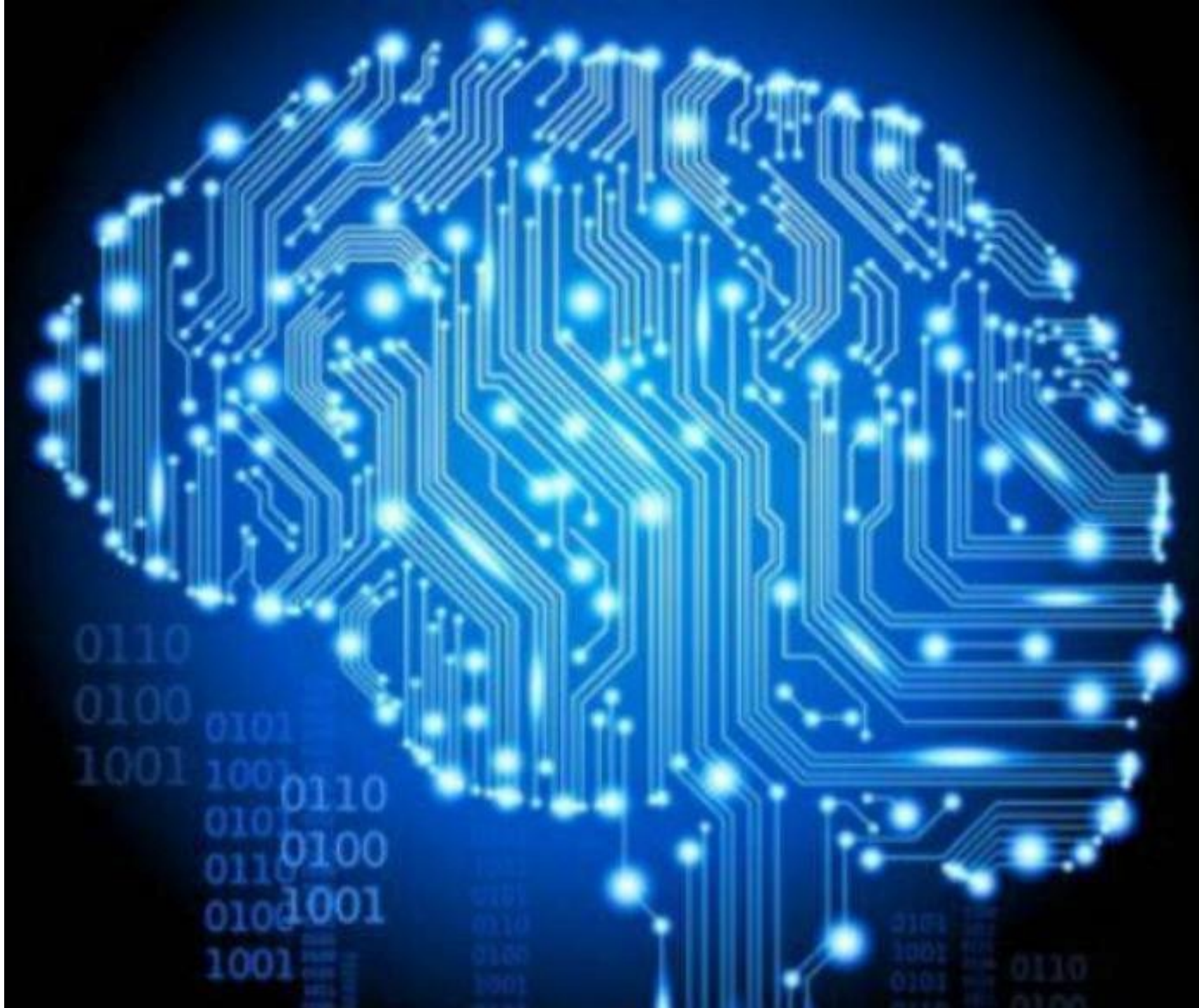


My recipe

- Pre-process data
  - KISS
- Start small
  - Underfit
- Find a setting where the problem can be learnt
  - Hyperparameter tunning
  - Random baseline
- Increase model's capacity
  - Depth vs Width
  - Overfitting
- Reduce gap between TR&VAL
  - Regularization
  - Data Augmentation
  - Reasonable fit
- Hyperparameter tunning
- Are we Fit yet?

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Convolutional Neural Networks



HIGH PERFORMANCE
ARTIFICIAL INTELLIGENCE

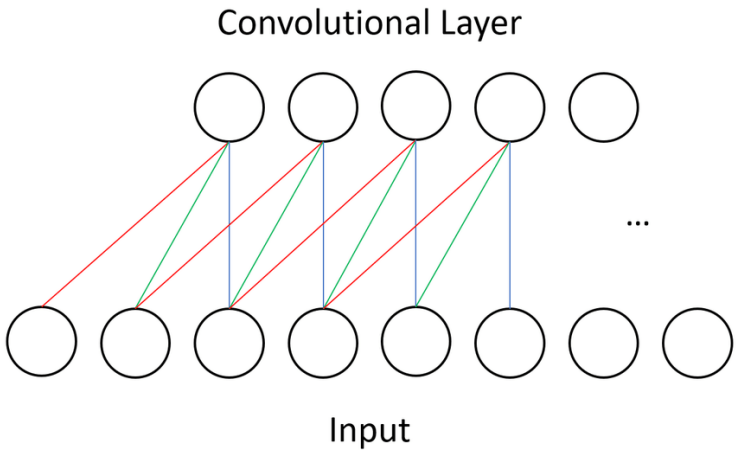
Convolutional Neural Networks

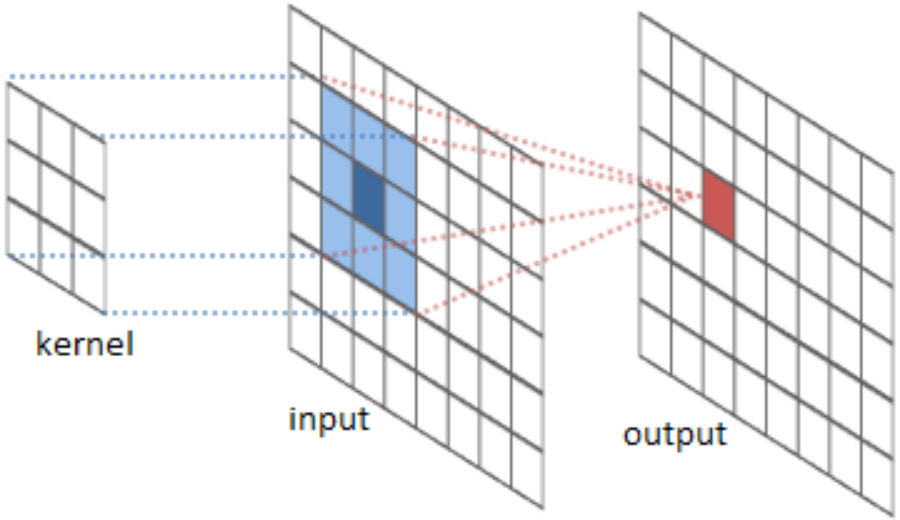
Some data has spatial correlations that could be exploited (in 1D, 2D, 3D, ...):

- *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**

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





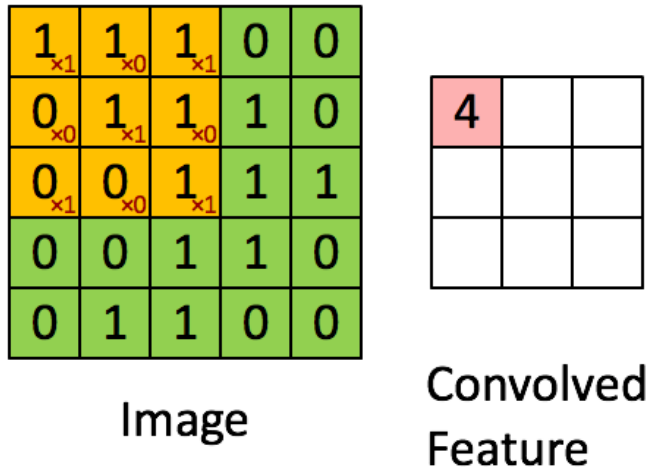
Convolutional Neural Networks

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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**

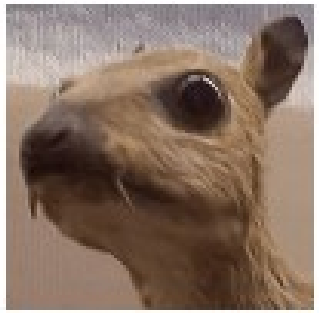
Convolutional Neural Networks

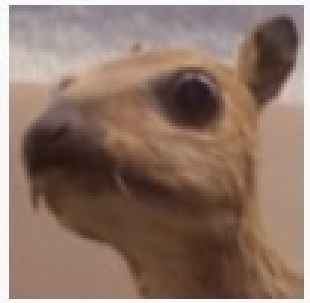
- Limited connectivity
- Convolution & weight sharing
- Filters
- Kernel size, stride and padding
- Convolutional volumes
- Pooling layers
- Convolutional architectures
- CNNs from the inside
- CNN Applications

HPAI
HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

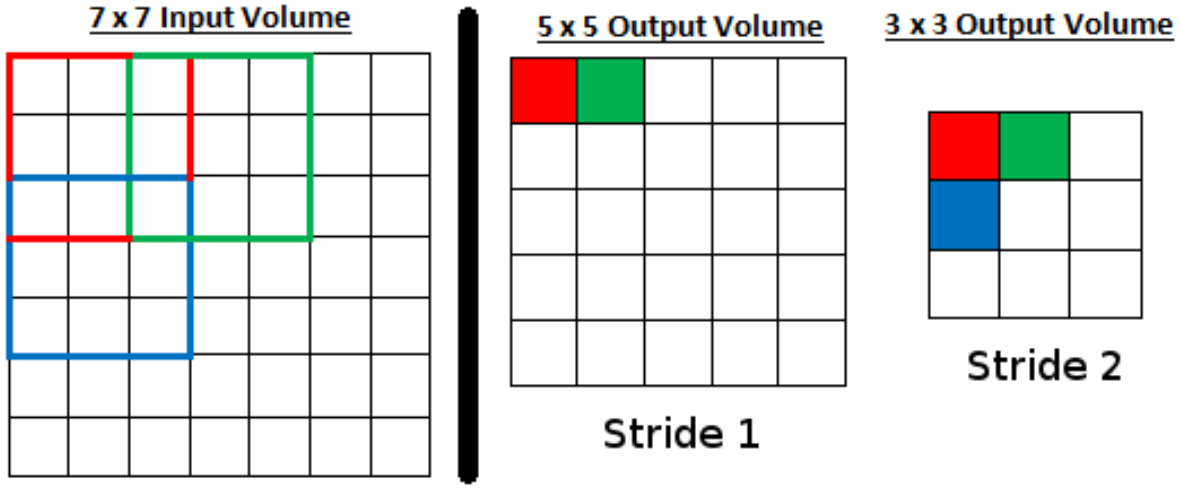
Let's let the model learn them

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.

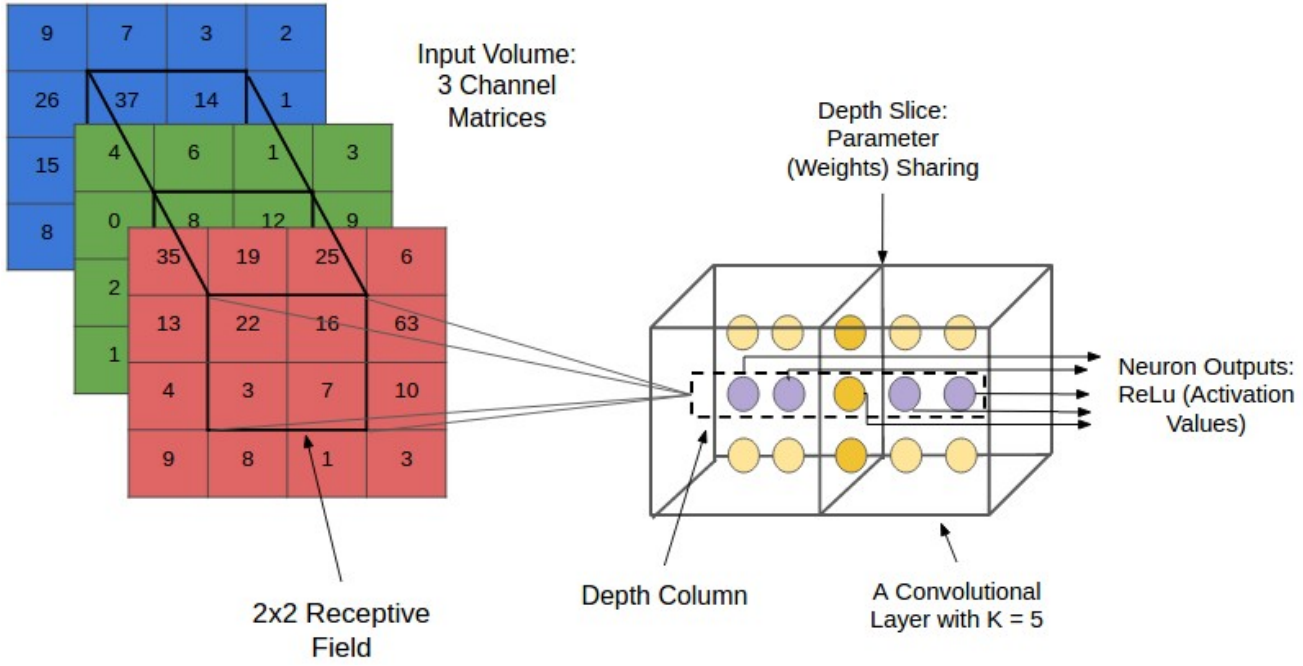
Padding: Border added to center conv. everywhere

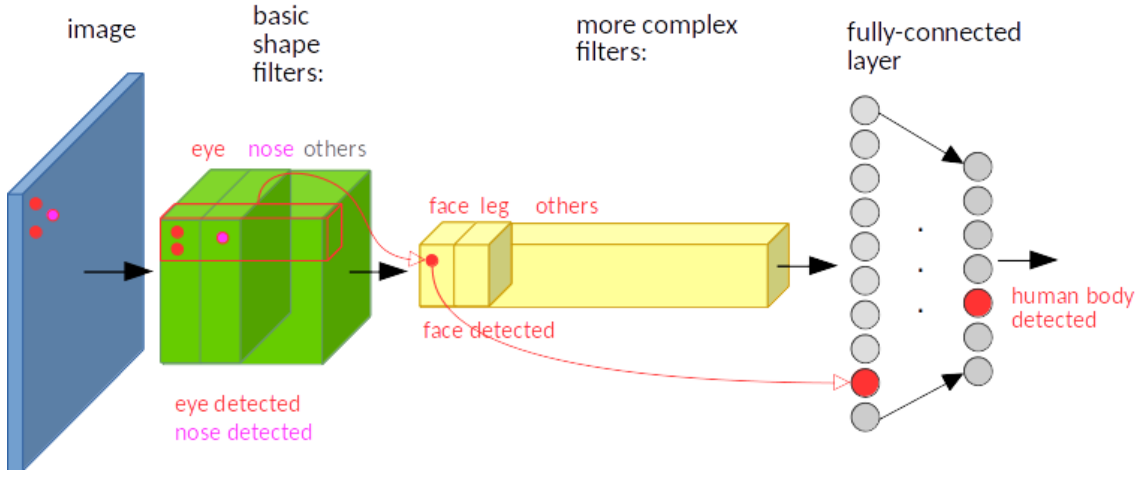
- No padding: Dimensionality reduced
- Most common, zero equal/same padding

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

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

HPAI
HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

Convolutional Neural Networks

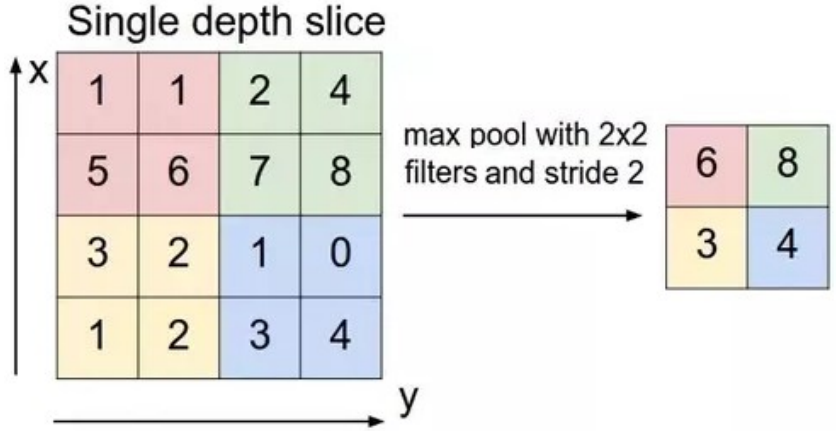
Limited connectivity
Convolution & weight sharing
Filters
Kernel size, stride and padding
Convolutional volumes
Pooling layers
Convolutional architectures
CNNs from the inside
CNN Applications

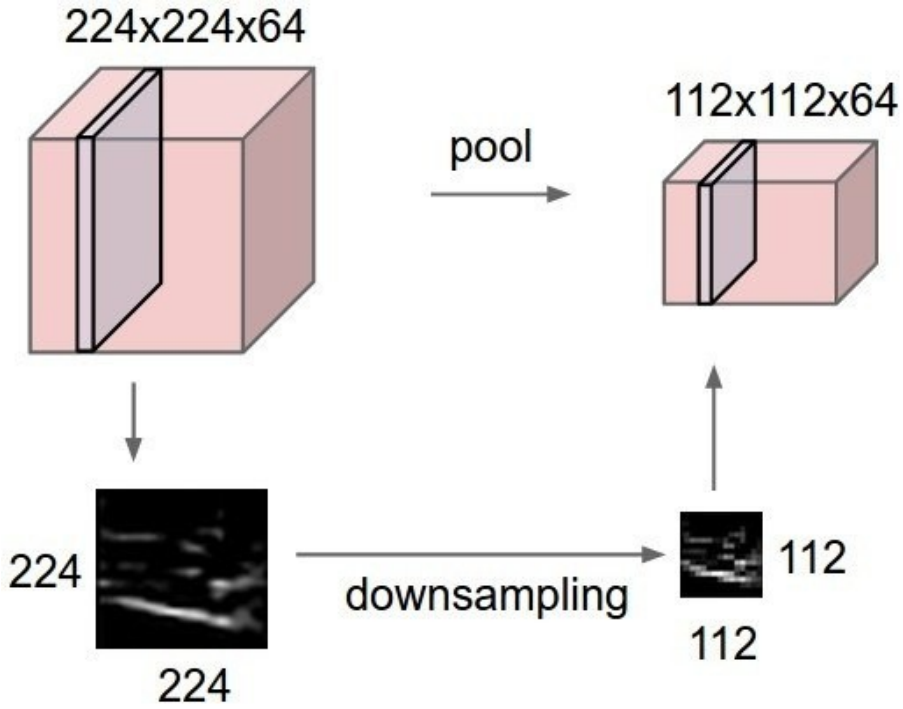
HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

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 **[AlexNet,12] [VGG,14]**
Convolutional volumes
Pooling layers
Convolutional architectures
CNNs from the inside
CNN Applications

The first influential architecture was **AlexNet**:

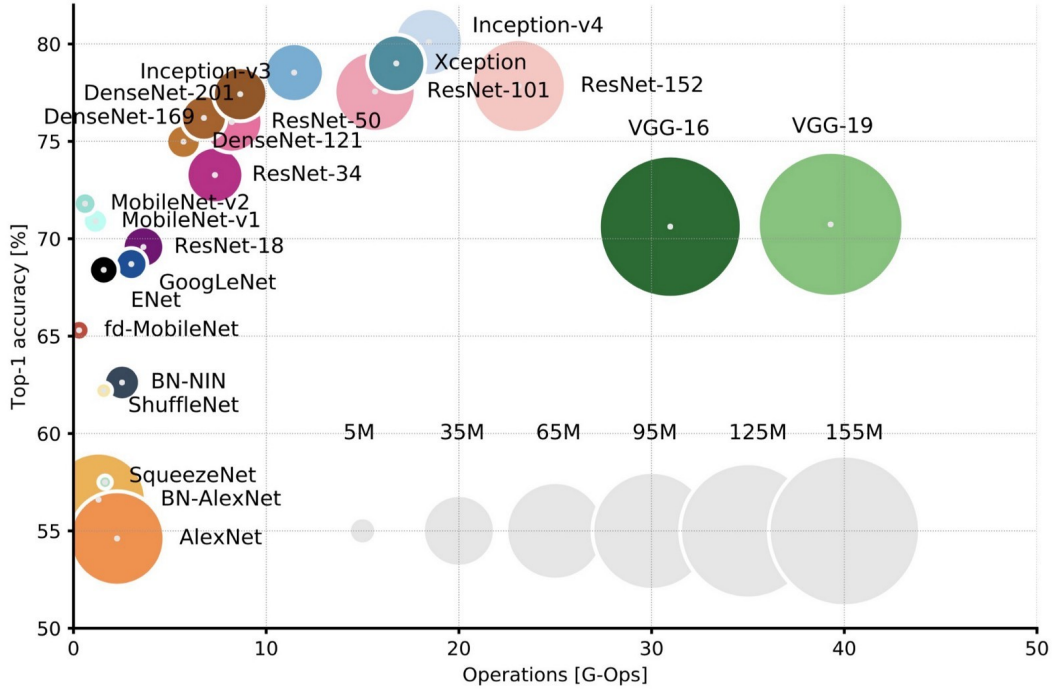
- 5 layers using convs, pools, *ReLU,* 2 dense, and *dropout*.
- 62M parameters

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]

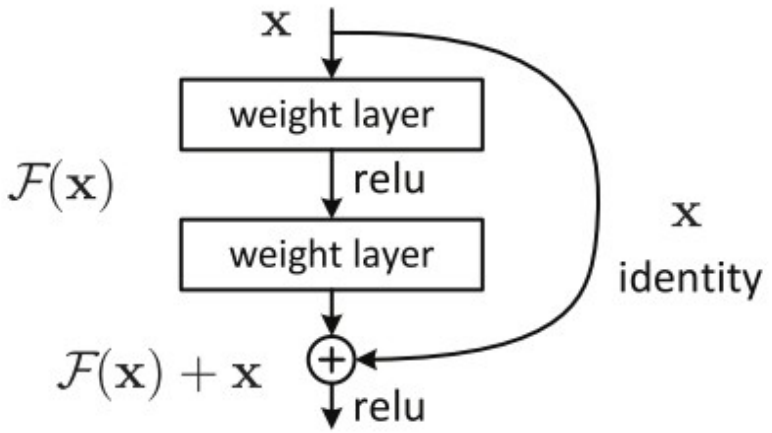
But deeper should never be worse!

- In theory, one more layer is innocuous
- In practice, its often terrible. Why?
-
-
-
-

Learning not alter anything is harder than it seems.
Its easier to learn to say always NO.

ResNet: Learning zero is easier than learning the identity function

- We can now train a 1K layer net
- They call them skip connections and residual blocks



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]

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

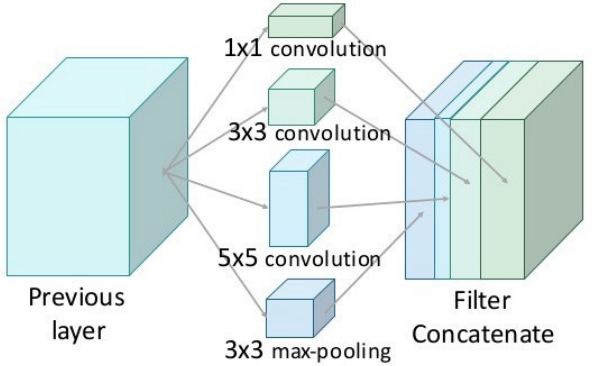
Inception:

How to choose filter size?

- Let the net decide which is best

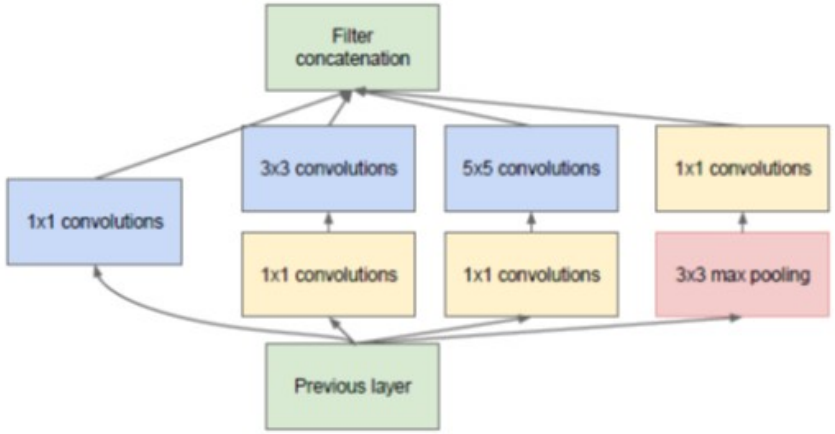
But bigger filters are expensive...

- Use 1x1 convs to reduce depth

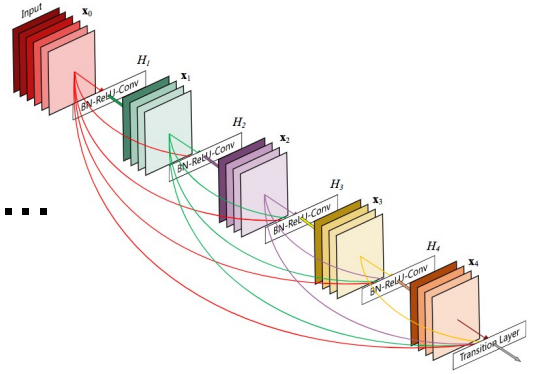


And fully connecteds require many parameters...

- Use global average pooling before to reduce width and height



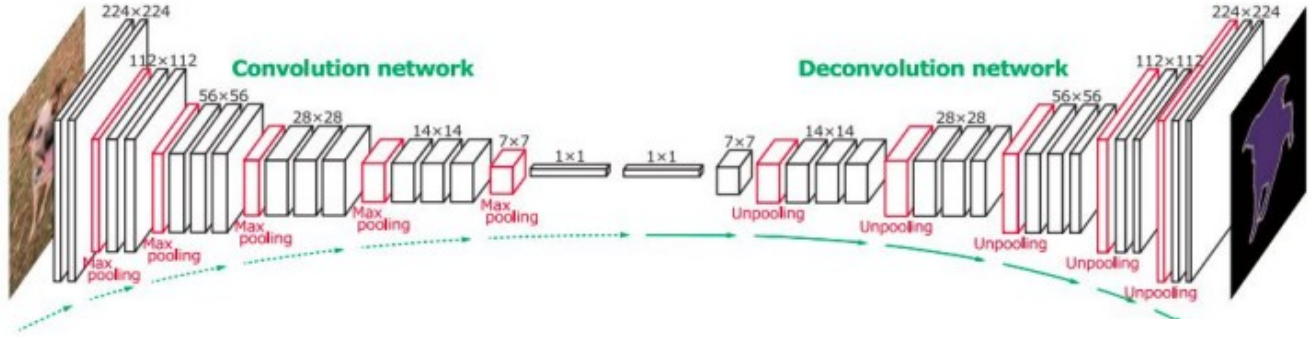
DenseNet and others build on top, ...

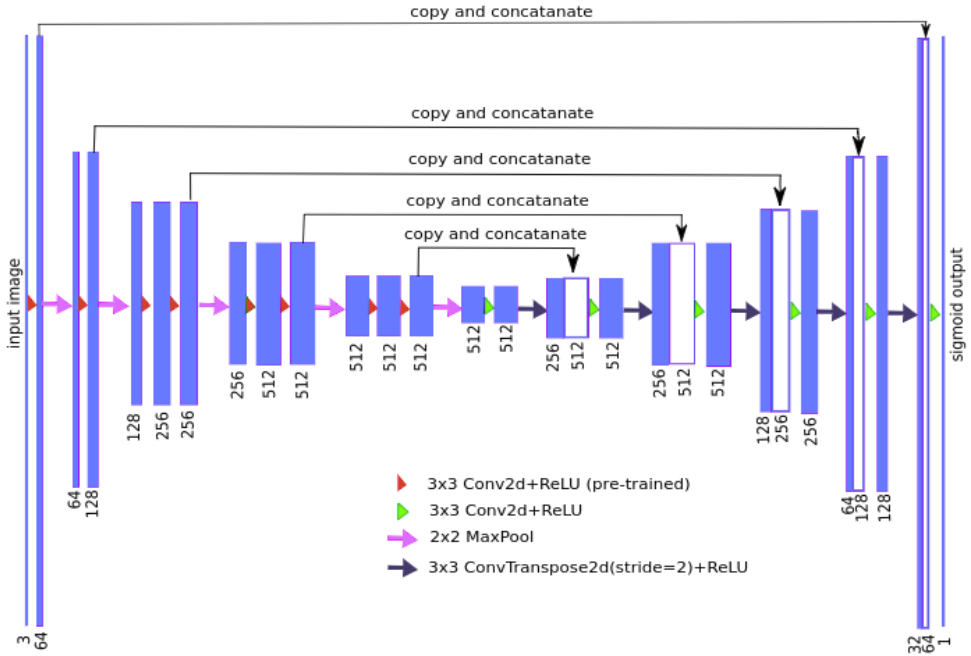


Convolutional Neural Networks

Different architectures can be done...

- Convolution – Transposed convolution (pixel-wise)





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

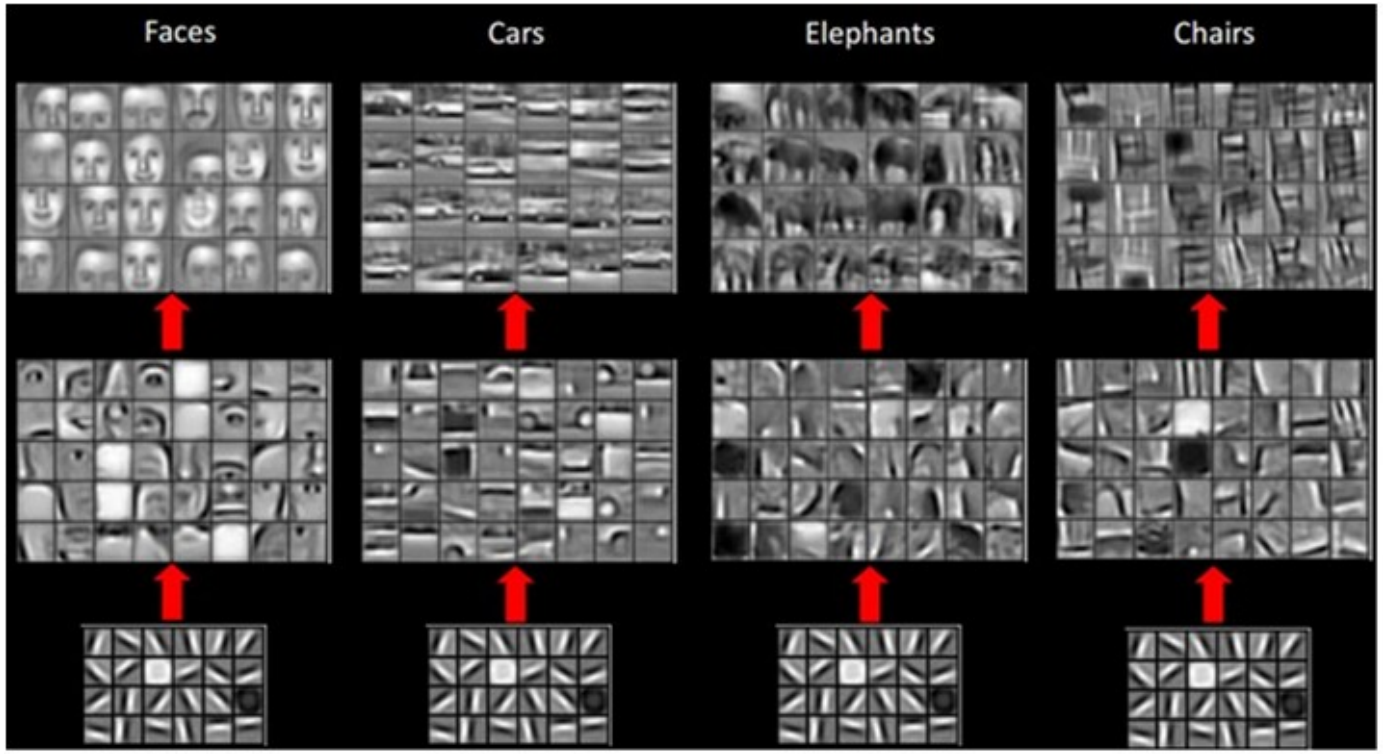
Convolutional Neural Networks

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

HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

What do filters learn?

- More depth more complexity
-
- Close to input: More relevant to data, less to task
- Close to output: More relevant to task, less to data



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

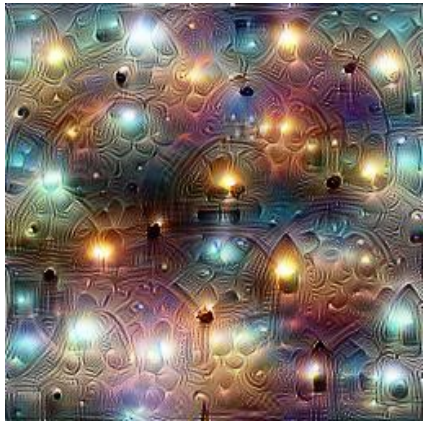
HIGH PERFORMANCE ARTIFICIAL INTELLIGENCE

- Low-level, channel wise



- Channel-wise vs neuron-wise


- High-level

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

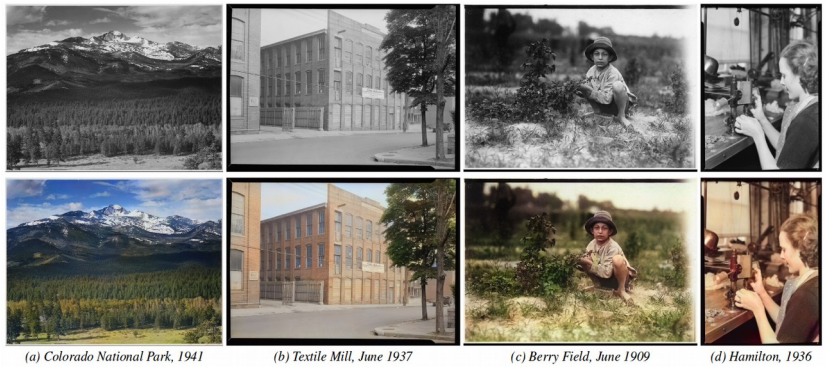
[Gatys,15]
[Kiros,14]

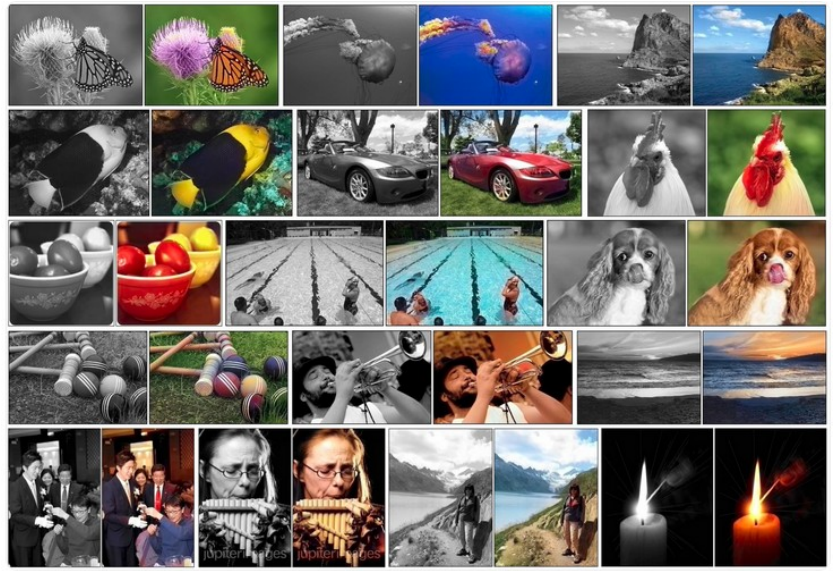
Style transfer

Multimodal pipelines

Convolutional Neural Networks

-
- Image colorization





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

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

Image segmentation