

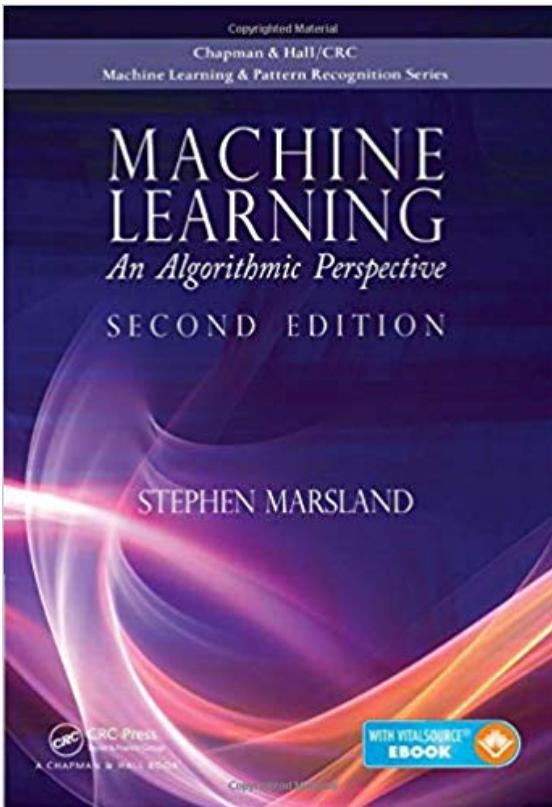
Introduction to Machine Learning and Artificial Neural Networks

• David Cornu

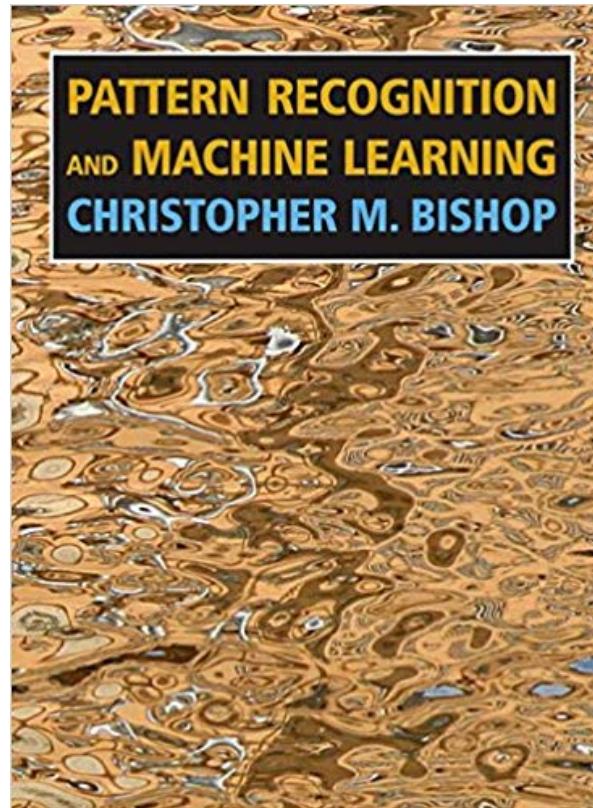
LUX, Observatoire de Paris, PSL

M2 OSAE 2025/2026

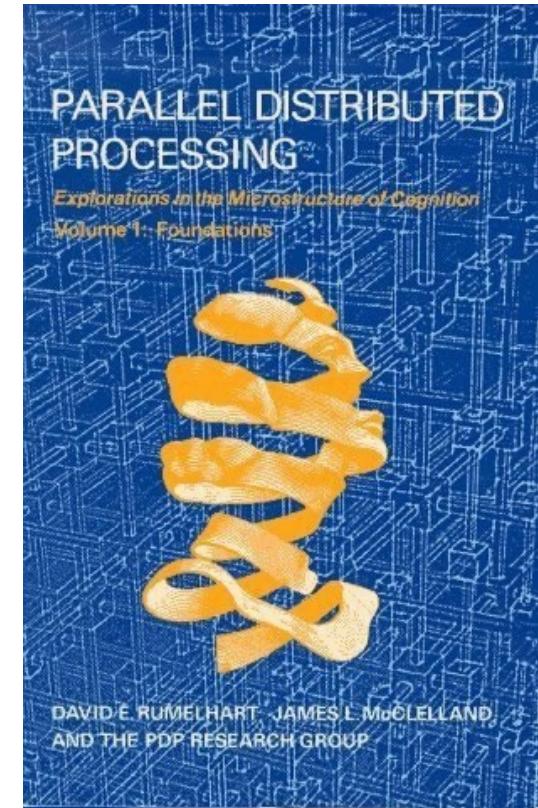
Some learning materials and references



Stephen Marsland.
**Machine Learning an
Algorithmic Perspective.**
Chapman and Hall/Crc,
2015



Christopher M. Bishop.
**Pattern Recognition and
Machine Learning.**
Springer, 2011



David E. Rumelhart et al.
**Parallel Distributed
Processing.** MIT press,
1989

Some learning materials and references

- My PhD Thesis: <https://arxiv.org/abs/2010.01431>
or its updated version: <https://share.obspm.fr/s/X3xN6NxJbnZDBWX>
Chapters 2, 4 and 11 can be considered as a handout for this course
- **The Fidle** (Formation Introduction au Deep Learning) **course from the CNRS:**
<https://gricad-gitlab.univ-grenoble-alpes.fr/talks/fidle/>
- The Neural Network CS231n course from Stanford:
<https://cs231n.github.io/>
- Some more references on a dedicated page on the MINERVA website
<https://minerva-astro.obspm.fr/learning/>

Take everything you read about ML and AI with a grain of salt! (This course included)

*Lots of materials still contain practical or theoretical approximations and errors.
Crossing references is most of the time necessary.*

1) What is Machine Learning ?

Any method that extracts statistical information about a dataset through a learning process.

Another appropriate name is « **statistical learning** » .

But how to define learning ? (still a relatively open question depending on the research field)

And what is **Artificial Intelligence** then ? Tricky and subjective question ...

→ Most of the time ML is considered as a sub-field of a larger AI domain.

2) Why is it interesting ?

Learns directly from data:

- Can learn **unknown relations** that are too difficult to formulate analytically
- Can work efficiently in **high dimensional parameter spaces** and for **large volumes of data**
- Can approximate **very complex relations** (methods that are Universal Function Approximators)

3) For which applications ?

Almost anything you can think of !

BUT it is limited by the data availability and computing power.

Are ML methods better for any application ? **Not at all !**

Warning: AI is often used as a selling argument, while it is not automatically better than classical data analysis

*in a 2019 report from the London venture capital firm MMC, 40% of « AI startups » don't really use AI or ML

Attempt of learning definition

The online Cambridge dictionary:

« *The process of getting an **understanding** of something by **experience*** »

The *Tresor de la langue Française informatisé*:

« *Acquérir la **connaissance** d'une chose par l'**exercice** de l'**intelligence**, de la **mémoire**, des mécanismes gestuels, appropriés, etc.* »

« *Acquire the **knowledge** of something through the **practice** of **intelligence**, **memory**, appropriate gestures, etc.* »

But these definitions are biased toward a human definition of the learning process...

Attempt of learning definition

In a more generic way, **animal learning** is usually summarized by these capabilities:

- Sensitivity

- Memory

- Adaptation

- Generalization

By adopting the proposed **statistical learning** definition, any algorithm that works using the following elements can be considered as **Machine Learning**:

- An adaptable input construction, which represents the **sensitivity** of the method.
- A form of **memory** to remember either the previous situations, or a reduced version of them.
- A way to estimate if its behavior (output prediction) is appropriate.
- A way to **adapt** its behavior during the learning process.
- A way to **generalize** its behavior to new outputs.

Keywords to explore the notion of Artificial Intelligence from there (beyond the scope of this course) :

- Symbolic and sub-symbolic representations
- Artificial General Intelligence
- Strong AI

Types of Machine Learning

Supervised

Learn from **labeled examples**. Try to find a **generalization** to get correct prediction most of the time.

Semi-Supervised

Usually a **mix of two algorithms**. A un-supervised one first that minimize the definition biases. A supervised one that remap to the expected groups.

Un-Supervised

Dataset with examples is provided but with **no labels**. Try to find **similarities** in the input in order to **create groups**.

Self-supervised

Dataset with **no external labels**. Use the data themselves to define labels and learn using a supervised formalism. (e.g., auto-encoders)

Reinforced

Based on an **reward function**. Used when there is no easy error function. Corrections are applied based on an **external performance measurement**.

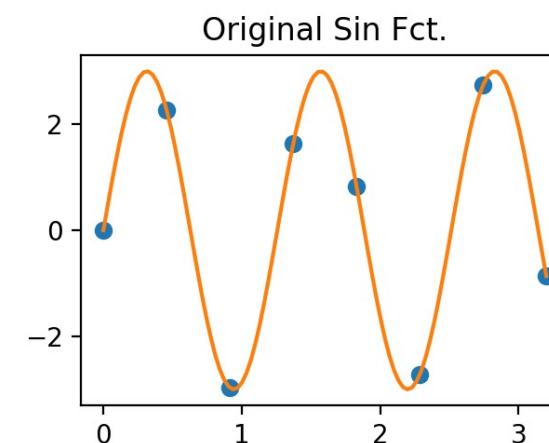
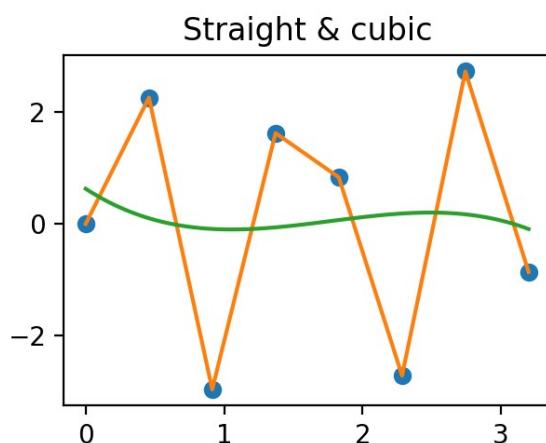
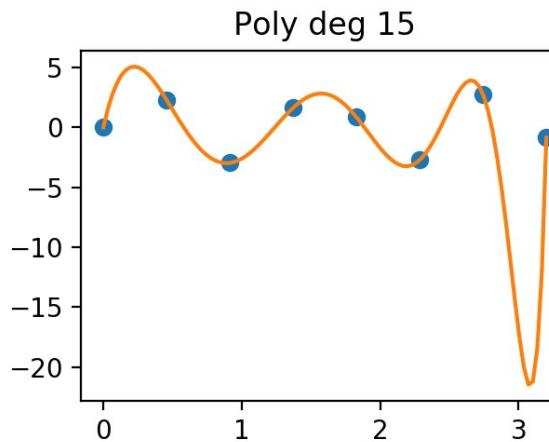
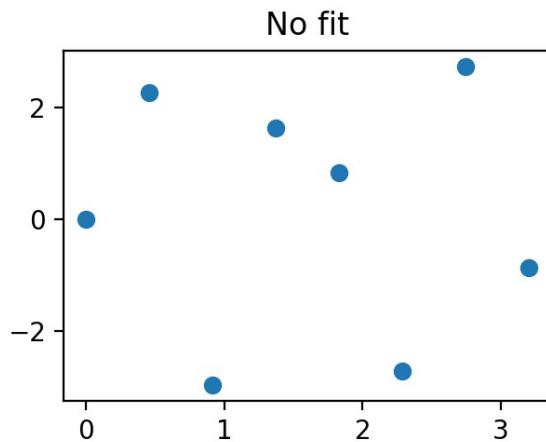
Evolutionary

Similar to reinforced, explore the possibility space based on biological **evolution processes**. Selection effects, reproduction, mutation, etc.

Use cases

Simple example of regression

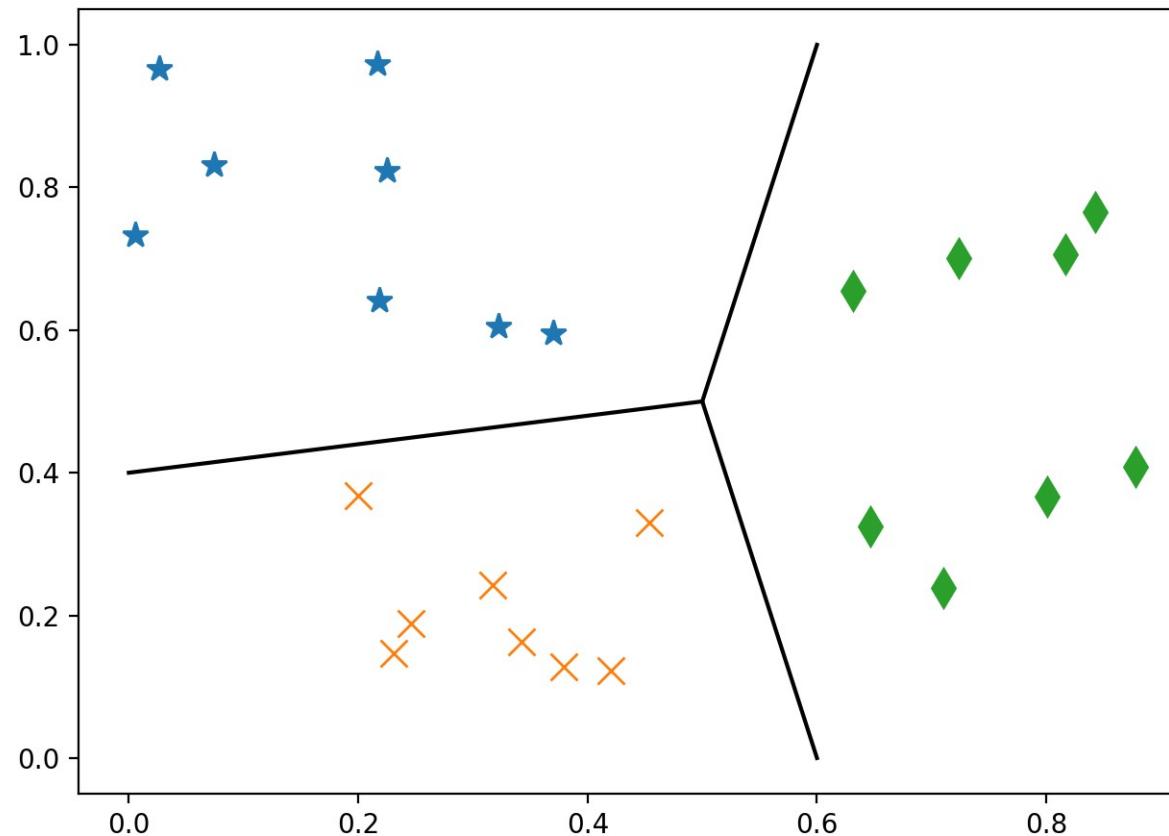
Reconstruct functions from a sample of data points (in any number of dimensions)



Use cases

Simple example of classification

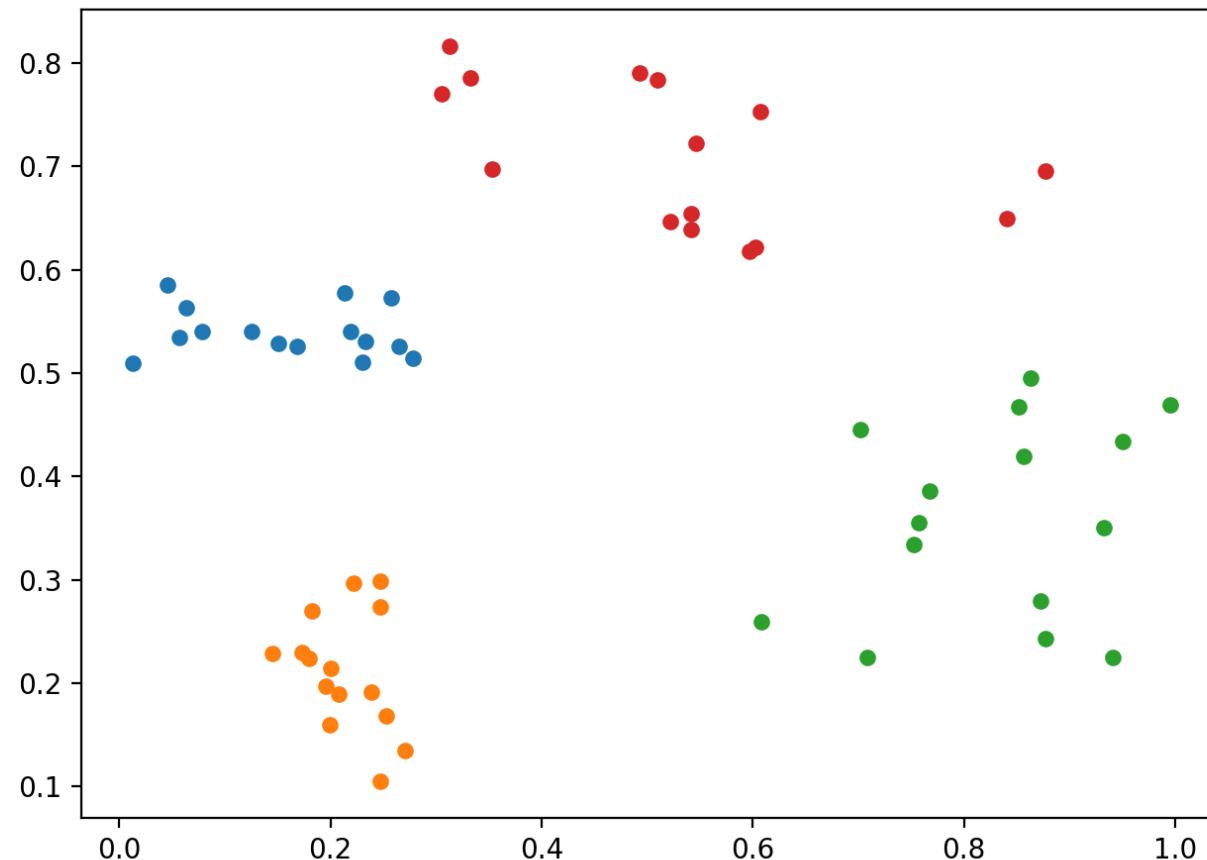
Decide from an input vector with several dimensions to which class it should be associated.

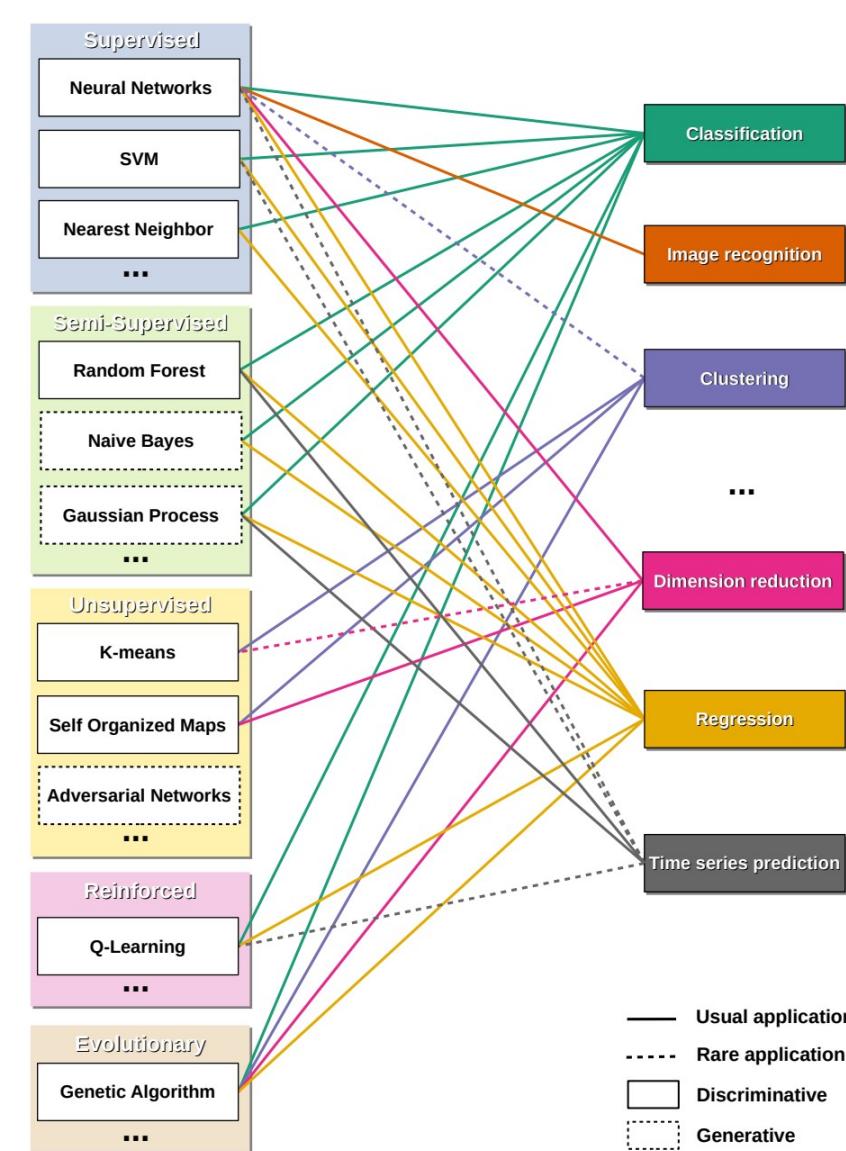


Use cases

Simple example of clustering

Find groups in data based on a predefined strategy (e.g geometric distance) without prior knowledge on the number of groups, nor the solution for any example.





A wide variety of applications ...

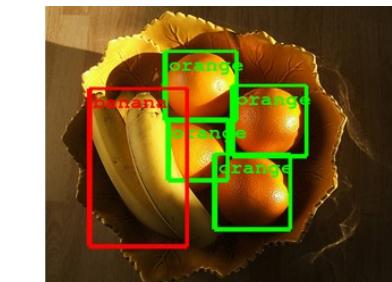
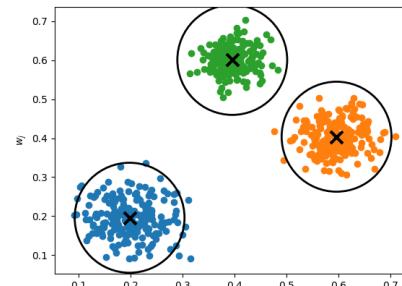
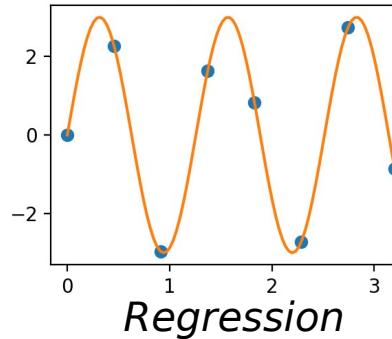
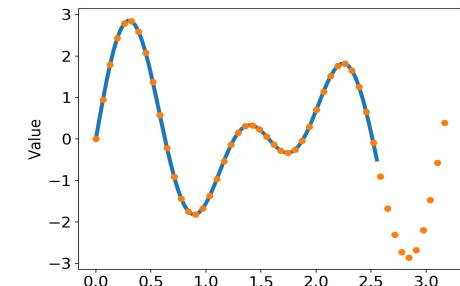
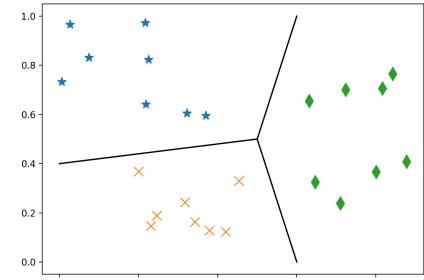
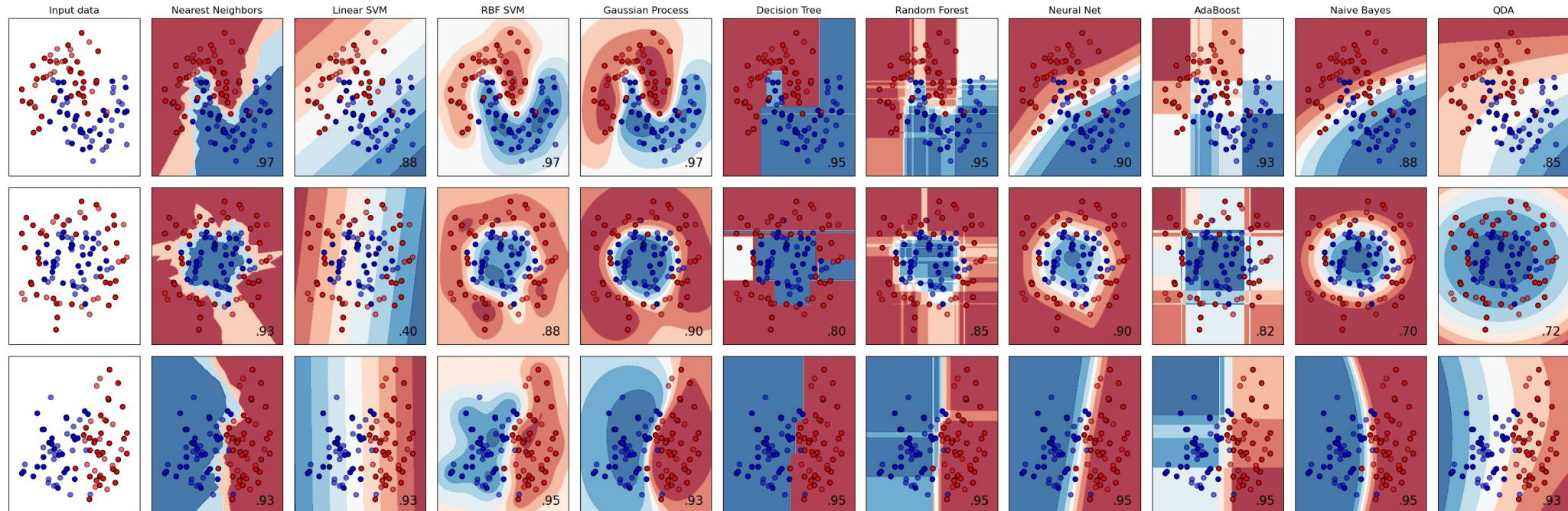


Image recognition



And much more ...

Methods comparison regarding data distribution



Some methods are **more suited for specific data distributions**, while others work well enough for most data distributions you might encounter.

In any case, **every method has its weaknesses** that you must identify and understand.

Choosing a method is selecting a balance between:

- The method capability on your data
- The computational efficiency of the method
- The required time of development / integration
- Your own proficiency with the various methods
- ...

How to use Machine Learning ?

Software



Nowadays, there are dedicated ML frameworks for almost **every programming language**.

They all have specific advantages, but any of them is enough to start learning and using ML.

Hardware

ML methods require a **lot of data** and **computing power** during the **training phase**.

However, these methods are very **compute efficient** when making **predictions**, meaning that it is often possible to deploy already trained models on light hardware (laptops or even modern smartphones)

Still, light resources are usually enough to **explore relatively simple ML methods**, and many complex problems can be expressed in a simple form, allowing preliminary studies on low-end hardware resources to be conducted.

When training powerful models that work on images, it is helpful to **use GPU hardware** that is very efficient for most modern ML approaches.



Note that **hardware dedicated to ML methods** are more and more present and accessible.

(*Tensor Cores in CPUs and GPUs, Neural Processing Units (NPU), etc ...*)

Use cases

Examples of modern applications

The screenshot shows the DeepL translation interface. At the top, there's a navigation bar with links for DeepL Traducteur, DeepL Pro, Services, API, Forfaits et tarifs, Applications (with a GRATUIT button), Nous contacter, Commencer l'essai gratuit, Connexion, and a menu icon. Below the bar, there are two main translation options: "Traduire du texte" (26 langues) and "Traduire des fichiers" (.pdf, .docx, .pptx). The main area shows a text input field for English and a corresponding French output field. The English text is a quote by Alan Turing about computer imitation games. The French translation is also provided, with some parts underlined in red, likely indicating differences or edits. At the bottom of the interface, there are various interaction icons like volume, thumbs up, thumbs down, and share.

DeepL Traducteur DeepL Pro Services API Forfaits et tarifs Applications GRATUIT

Nous contacter Commencer l'essai gratuit Connexion

Traduire du texte 26 langues

Traduire des fichiers .pdf, .docx, .pptx

Anglais (langue détectée) ↗ français ↗ formel/informel Glossaire

I believe that in about fifty years' time it will be possible to programme computers, with a storage capacity of about 10^9 , to make them play the imitation game so well that an average interrogator will not have more than 70 per cent. chance of mating the right identification after five minutes of questioning.

Je pense que d'ici une cinquantaine d'années, il sera possible de programmer des ordinateurs, avec une capacité de stockage d'environ 10^9 , pour qu'ils jouent si bien le jeu de l'imitation qu'un interrogateur moyen n'aura pas plus de 70 % de chances de faire la bonne identification après cinq minutes d'interrogatoire.

311 / 5000

Use cases

Examples of modern applications

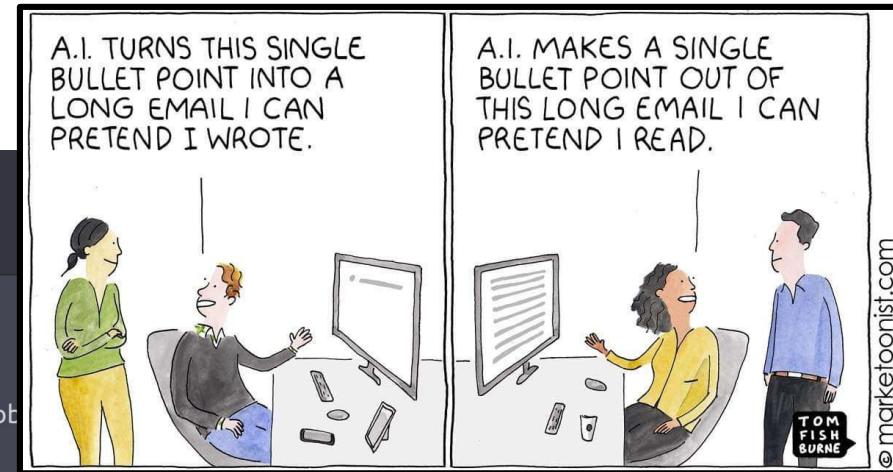
TH

Got any creative ideas for a 10 year old's birthday?



Sure! Here are some creative birthday ideas for a 10-year-old:

1. A themed birthday party such as a movie or video game character or a favorite hobby
2. A scavenger hunt with friends, either indoors or outdoors
3. A craft or DIY project, like creating tie-dye shirts or friendship bracelets
4. An outing to an amusement park, trampoline park, or laser tag center
5. A baking or cooking class, where the birthday child and friends can make and enjoy treats together
6. A sleepover with friends, complete with games, movies, and a special breakfast
7. A science or art experiment-themed party, where children can participate in hands-on activities
8. A sporty day with a visit to a rock climbing gym, batting cage, or mini golf course
9. A virtual escape room or game experience, played together with friends over video chat
10. A wildlife or nature exploration outing, such as a hike or a trip to a local zoo or aquarium.



© marketoonist.com

Modern AI LLMs generated text is now quite convincing for human reader, but it remains detectable with automated tools based on ... AI LLMs (adversarial attack)!

Use cases

Examples of modern applications



Use cases Pricing FAQ API ⚙

Use cases

Photographers

Creative Agencies

Real Estate

E-commerce

Remove text, logo or watermark

Developers API



Photographers use Cleanup.pictures to remove time stamps or remove tourists from holiday pictures before printing them for their customers.

They clean portrait photos to create the perfect profile pictures.

Cleanup.pictures is the perfect app to remove cracks on photographs. You can clean any images, removing any unwanted things. It is a must-have for professional studios.

*Based on the LaMa: Resolution-robust Large Mask Inpainting with Fourier Convolutions method
→ <https://arxiv.org/abs/2109.07161> and for fun <https://cleanup.pictures>

Use cases

Examples of modern applications

These persons do not exist !



*Based on the Style Generative Adversarial Network model

→ <https://arxiv.org/abs/1812.04948> and for fun <https://thispersondoesnotexist.com/>

Use cases

Examples of modern applications

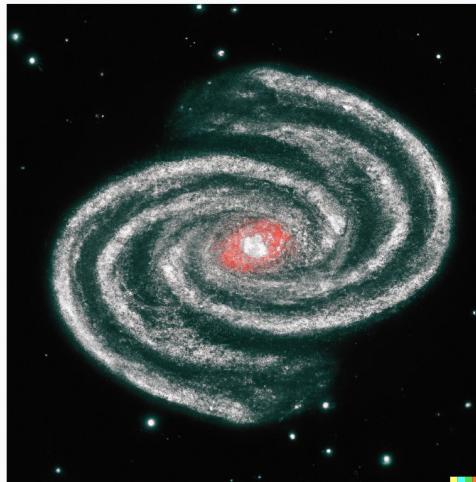
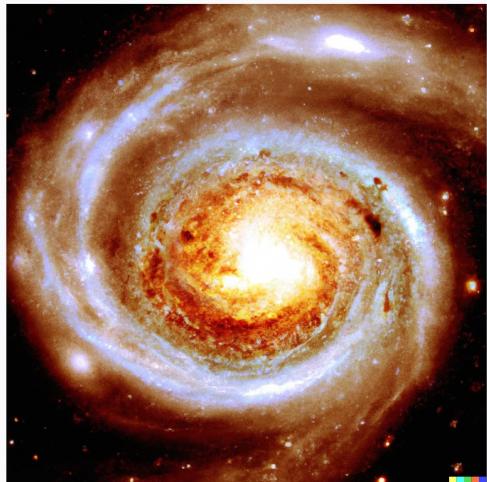
Edit the detailed description

Surprise me

Upload →

An artistic view of a barred spiral galaxy observed in the infrared

Generate



*Dall-E, based on CLIP word embedding and diffusion decodeur => <https://arxiv.org/abs/2204.06125>

Use cases

Examples of modern applications

Generate an image of "An artistic view of a barred spiral galaxy observed in the infrared"



Here are the artistic views of a barred spiral galaxy observed in the infrared. The images highlight the central bar structure, swirling spiral arms, and vibrant infrared colors with distant stars and cosmic dust in the background.

Generate a more realistic image that would resemble an astronomical observation



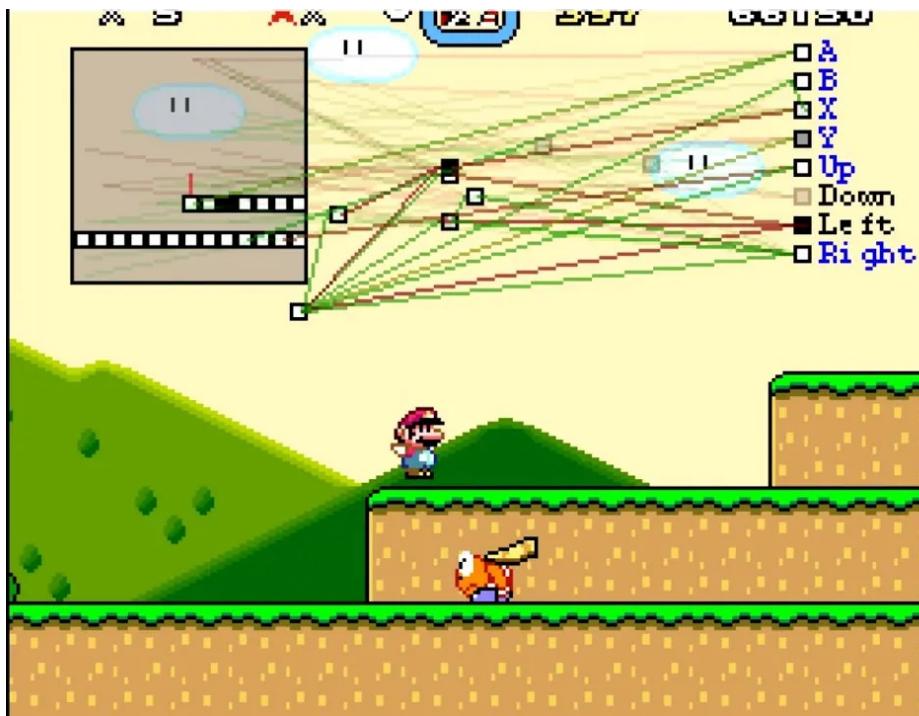
Here are the more realistic images of a barred spiral galaxy observed in the infrared, resembling actual astronomical observations.

With DALL-E 3 through ChatGPT 4 interface

Use cases

Examples of modern applications

The same processes are used to train robots to walk. But can also be used to infer any function from which you can only define a global posterior fitness score.



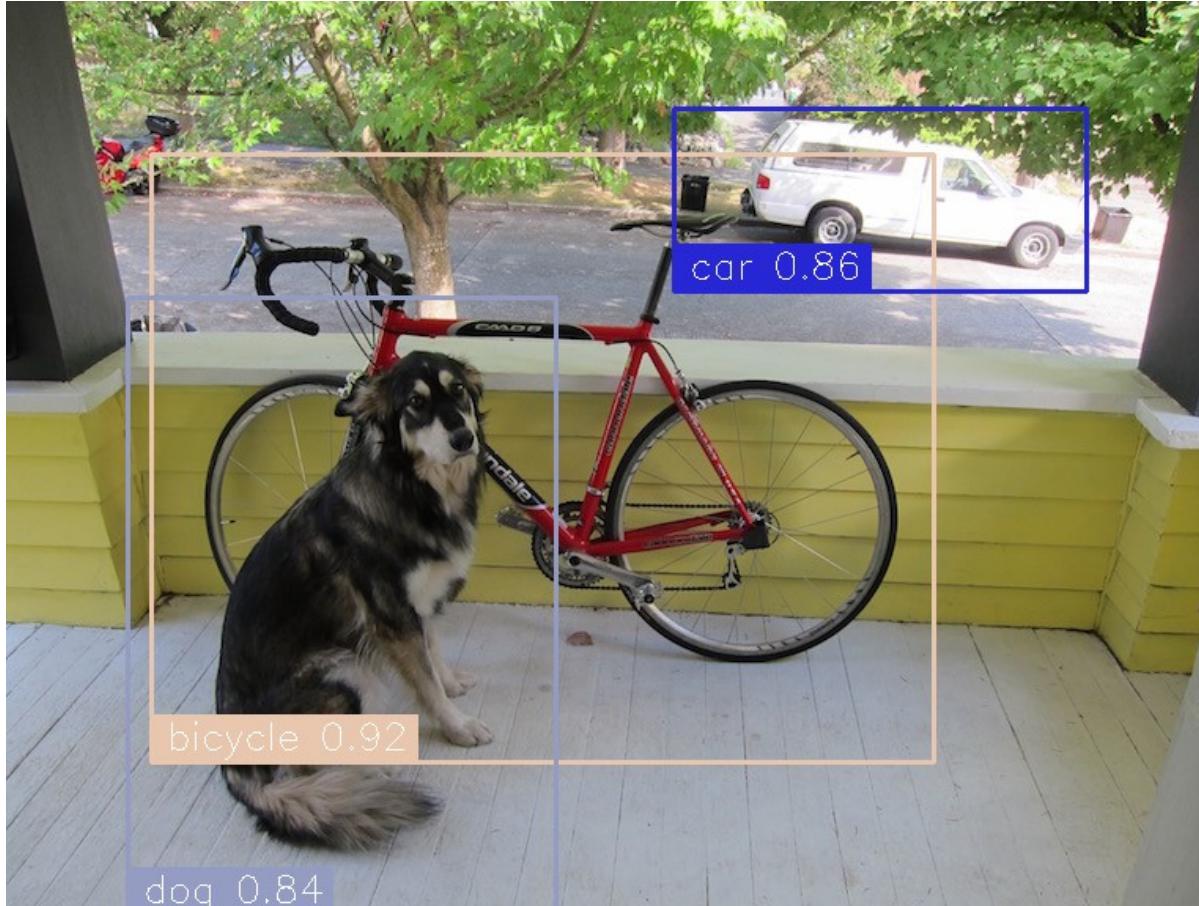
*Based on Genetic Neural Evolution
→ <https://www.youtube.com/watch?v=qv6UVQ0F44>



*Based on Deep Reinforced Learning method AC3
→ <https://arxiv.org/abs/1602.01783v2>
→ <https://youtu.be/nMR5mjCFZCw>

Use cases

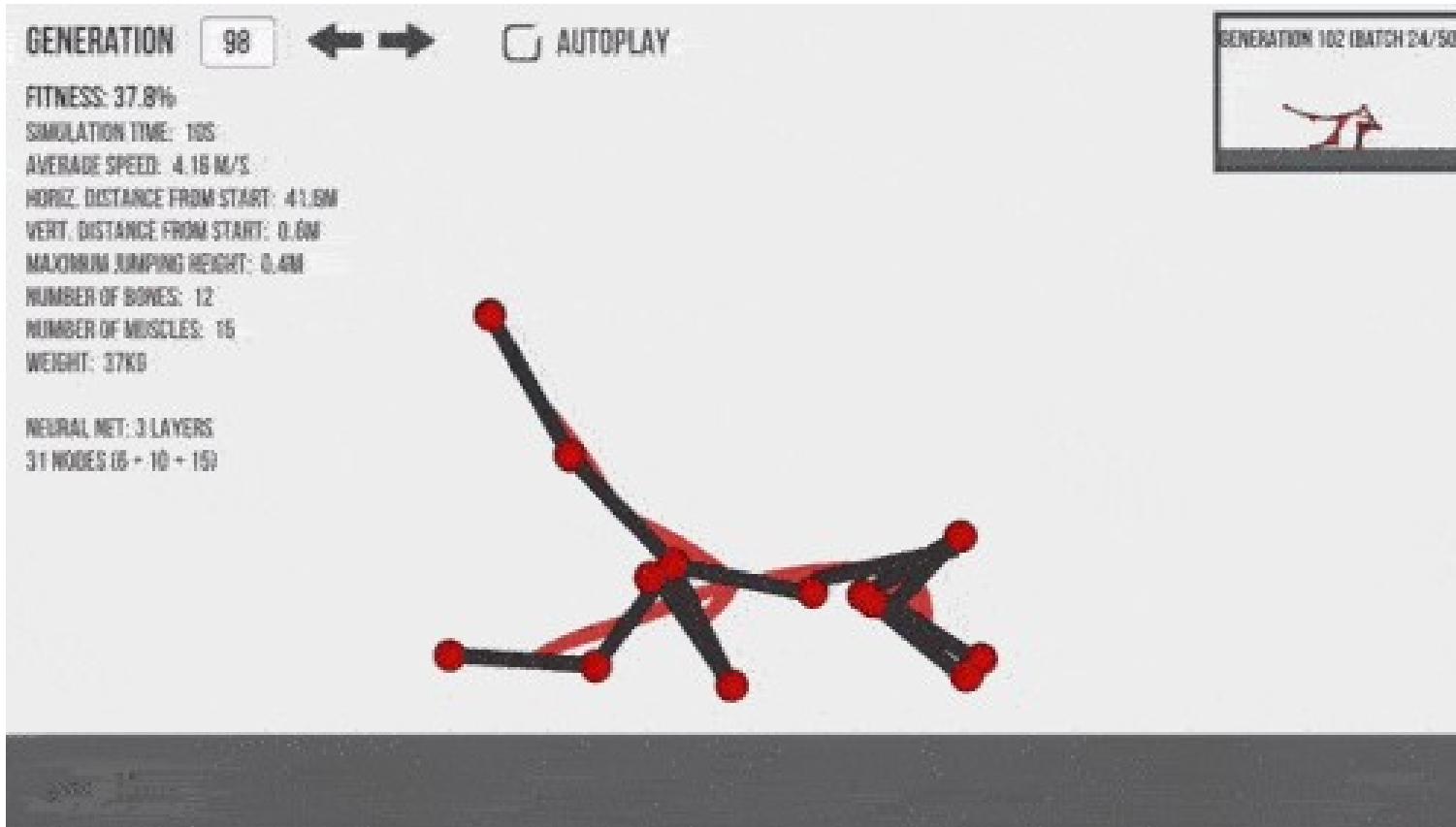
Examples of modern applications



*Based on the YOLO Convolutional Neural Network architecture for object detection
→ <https://www.youtube.com/watch?v=MPU2HistivI> and <https://arxiv.org/abs/1804.02767>

Use cases

Examples of modern applications

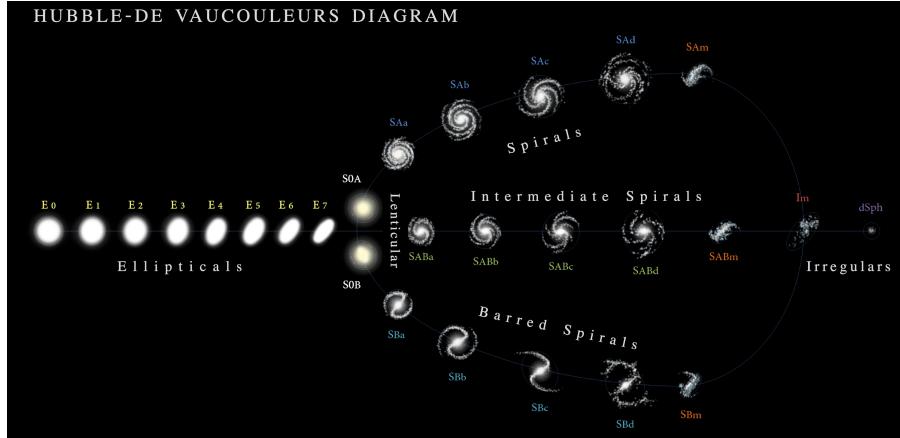


*Based on Genetic Neural Evolution

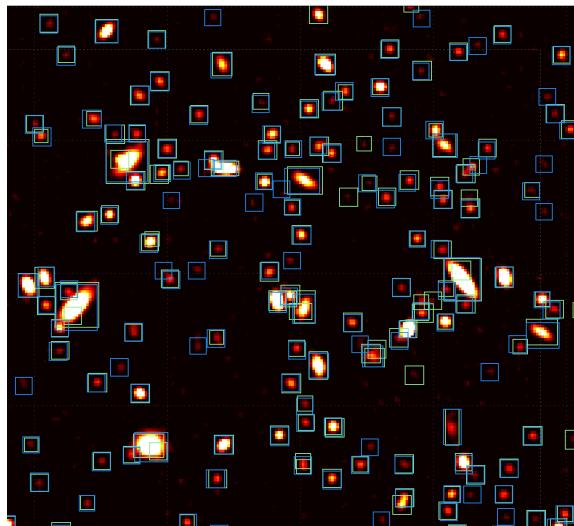
→ Interactive creature creation at <https://keiwan.itch.io/evolution>

What about astronomical applications ?

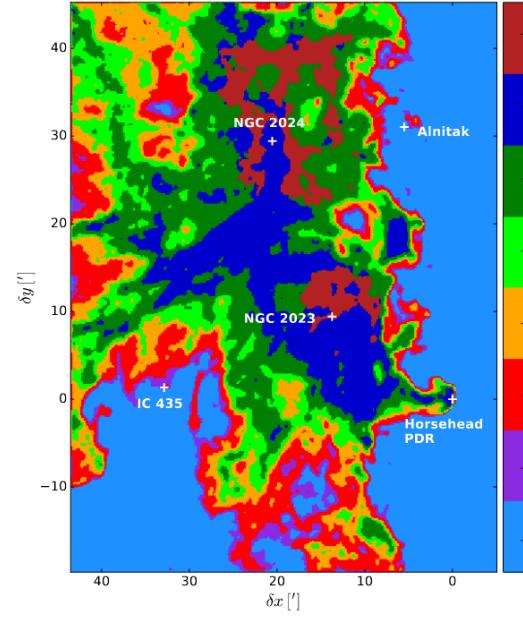
HUBBLE-DE VAUCOULEURS DIAGRAM



Morphological classification of galaxies



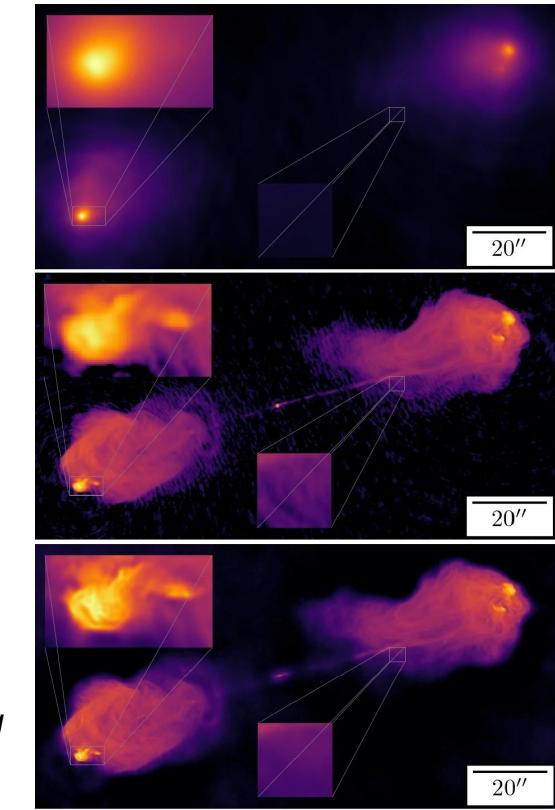
Detection and characterization of galaxies



Molecular cloud emission clustering

And much more :

- Cosmological simulation acceleration
- Inverse modeling for cosmological models
- Stellar classification based on spectra
- ISM turbulence regime classification
- Transient events time series prediction
- ...



Radio-astronomical image enhancing

Wide diversity of objects, methods, scales, data representation, etc...

Simple Neural Networks

Neural Network: The brain as model

« *There is a fantastic existence proof that learning is possible, which is the bag of water and electricity (together with a few trace chemicals) sitting between your ears [...] which is the squishy thing that your skull protects* »

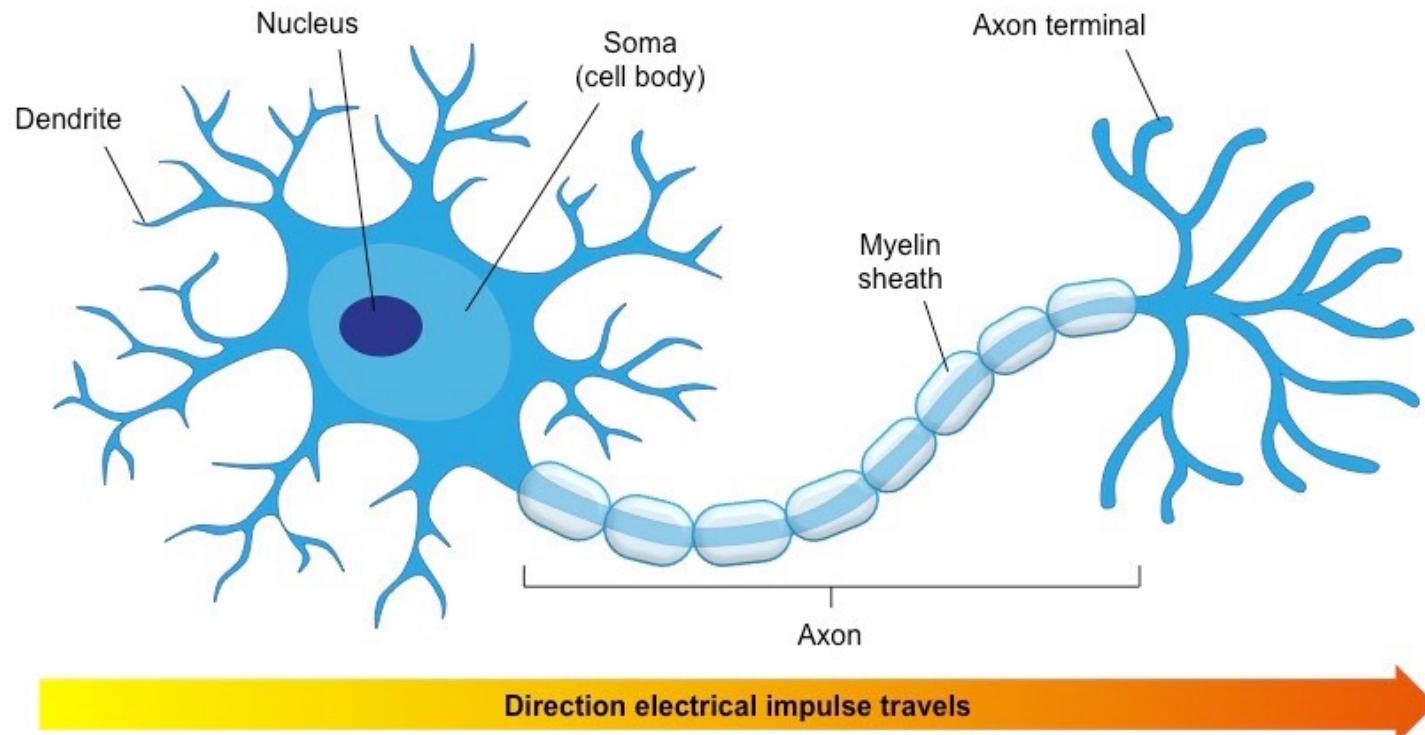
*Stephen Marsland

The brain does exactly what we want for data analysis:

- Extract complex information in a compressed form
- Deal with noisy and/or inconsistent data
- Work in highly-dimensional spaces
- Give the appropriate answer most of the time
- Provide results very quickly
- Remain robust through aging (neuron loss)

The biological neuron

Elementary brick of a biological brain (10^{11} in the human brain).
The idea will be to use it as a model to emulate learning capabilities.



Perform the **sum** of various electrochemical **input signals**. If this total signal is sufficient, it sends a **new signal** through its axon to **transfer information** (toward other neurons).

Biological Neural Networks

A connection between two neurons is called **a synapse** (10^{14} in the human brain).

A neuron is a binary compute unit that either “fire” or “not-fire” in response to a signal.

→ In this view, a brain is a **massively parallel super-computer of 10^{11} processing units**.

A simplified view of how it learns

The synapses represent the “strength” of the connection between two neurons.

Learning = modifying this connection (either positive or negative and changing its intensity)

→ This is called **plasticity**

The **Hebb law** defines a simplified rule for learning:

If two neurons “fire” **at the same time**, there must be some **correlation** between them, and their connection must be strengthened.

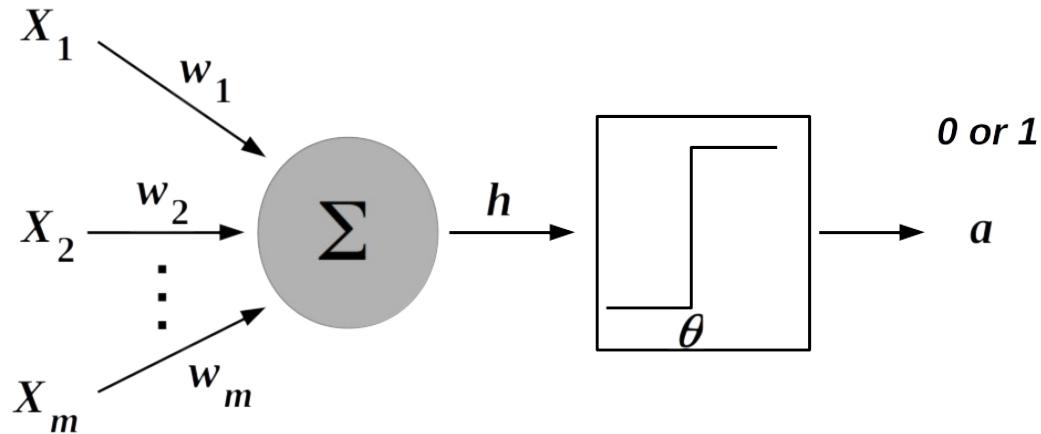
→ This is called **conditioning**

*These rules are not enough to train a neural system but illustrate the processes that occur in the biological brain.

To create an algorithm from these biological concepts, one first need to create a **mathematical model**.

Model of a Neuron

Mathematical model from **McCulloch and Pitts**, inspired by the biological neuron.

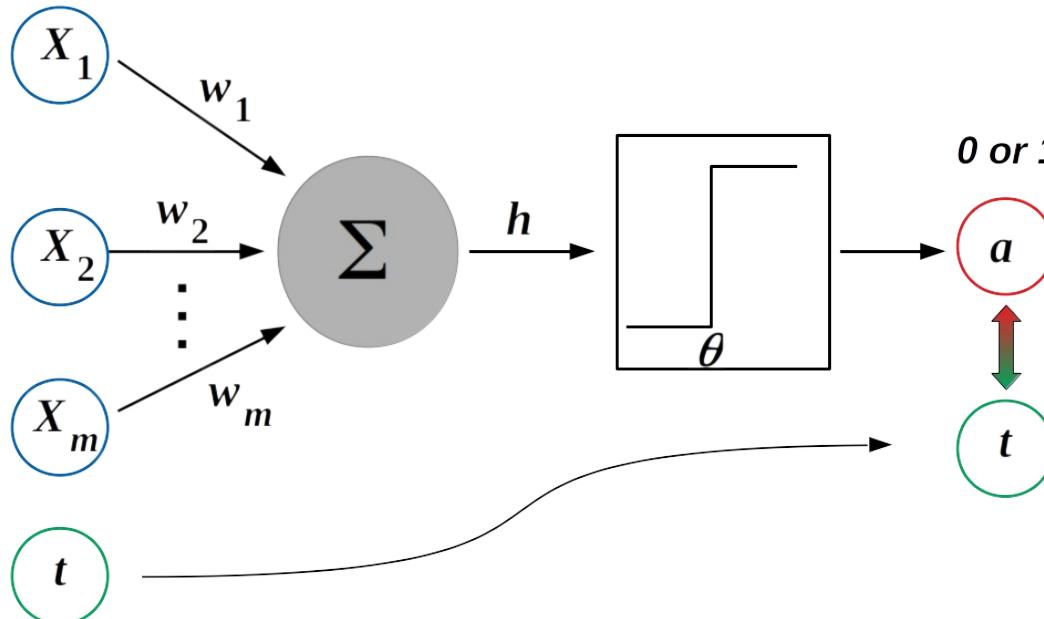


$$h = \sum_{i=1}^m X_i w_i \quad a = g(h) = \begin{cases} 1 & \text{if } h > \theta \\ 0 & \text{if } h \leq \theta \end{cases}$$

Its main components are:

- An **input vector X_i** that represents the dimensions of a given object
- A **set of weights w_i** that links the various input dimensions to the neuron
- A **sum function h** that defines how these weights are combined with the input dimensions
- An **activation function $g(h)$** that defines if the network should remain in a “0” state or should be activated to a “1” state, depending on the results of the sum.

Training a Neuron



The learning process for this model is supervised
→ Each **input vector** is associated to a **target value**

So what is the purpose of the neuron if the expected value is already known ?

→ **Generalization**

Find **patterns** in the parameter space so it can perform prediction on vectors with unseen (but close) values.

In practice, such a simple neuron **optimizes a linear separation** in the parameter space (**hyperplane**).

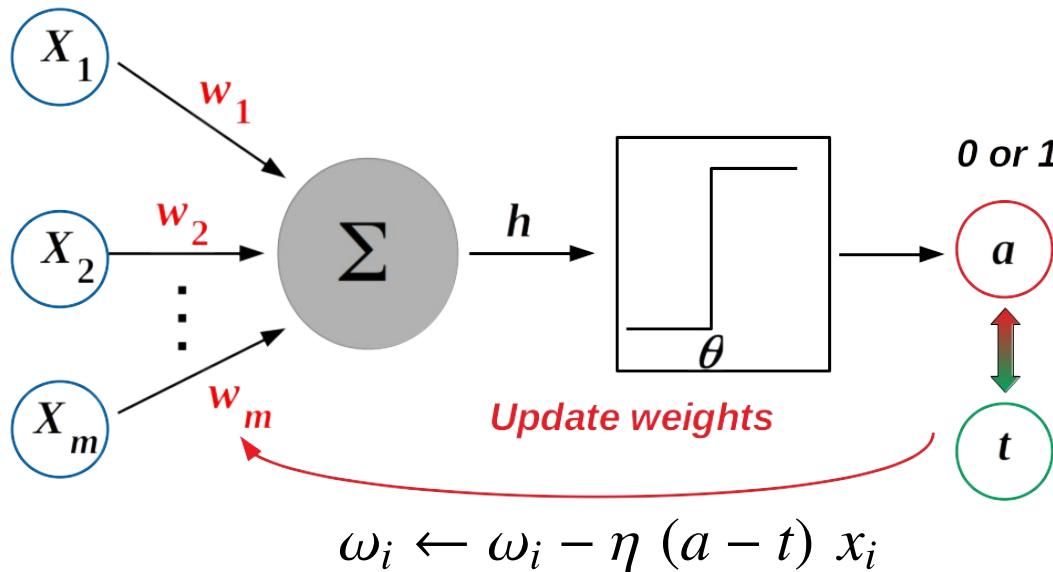
How does this model learn ?

Which parameters can or cannot be modified ?

The **input, output, and targets are fixed**, so the learning relies on modifying the **weights and the activation threshold**.

$$h = \sum_{i=1}^m X_i w_i \quad a = g(h) = \begin{cases} 1 & \text{if } h > \theta \\ 0 & \text{if } h \leq \theta \end{cases}$$

Training a Neuron



$$h = \sum_{i=1}^m X_i w_i \quad a = g(h) = \begin{cases} 1 & \text{if } h > \theta \\ 0 & \text{if } h \leq \theta \end{cases}$$

How to proceed when the output does not correspond to the target ?

There are m weights w_i associated with the network corresponding to the input dimensions.

How to modify the weights?

- If the output state is 1 while it should be 0, the weights need to be **lowered**
- If the output state is 0 while it should be 1, the weights need to be **increased**

To quantify this modification, one needs to choose an **error function E** .

$$E = 0.5 \times (a - t)^2$$

In the end, the weight update is defined as proportional to the **input value**, to the **derivative of the error function** for each dimension, and scaled by a **learning rate*** factor.

$$\omega_i \leftarrow \omega_i - \eta (a - t) x_i$$

*Typically between 0.1 and 0.4, details will be given later. 30 / 82

The bias node

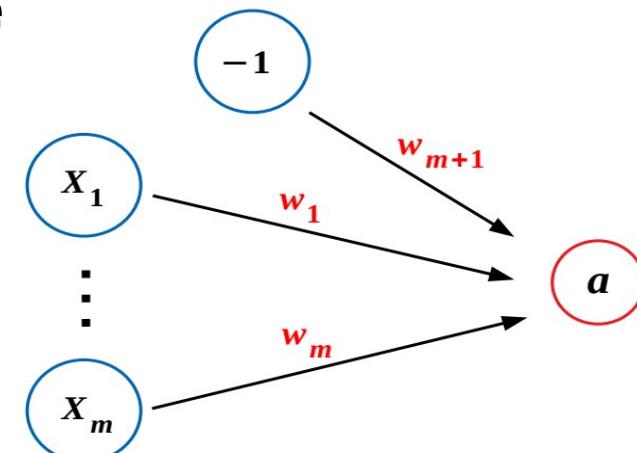
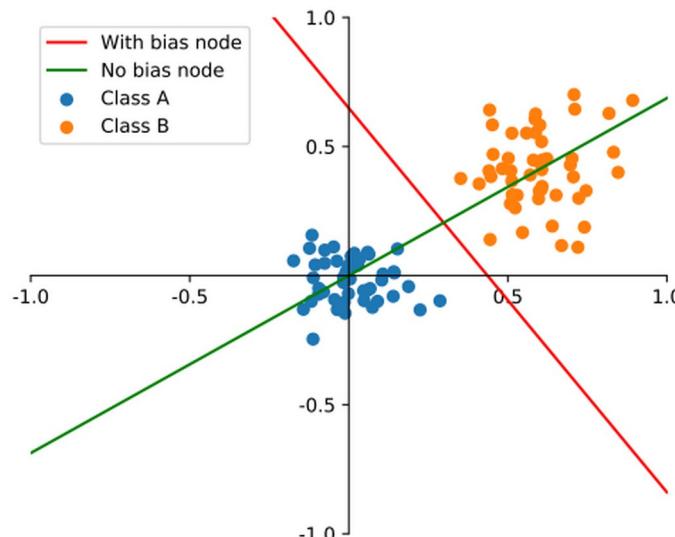
Problem with the previous formalism

The linear separation presents a **fixed $f(0) = 0$ point**.

The weight correction is also 0, regardless of the input value.

Solution

Add a **bias input node** that acts as an additional input with a **constant value** (usually -1). It has its own **variable weight** so it can learn the shift of the intercept position. The size of both the input vector and the weights vector is now $m+1$.



To plot the found linear separation of a neuron, we need to find where in the parameter space the neuron changes state, which corresponds to:

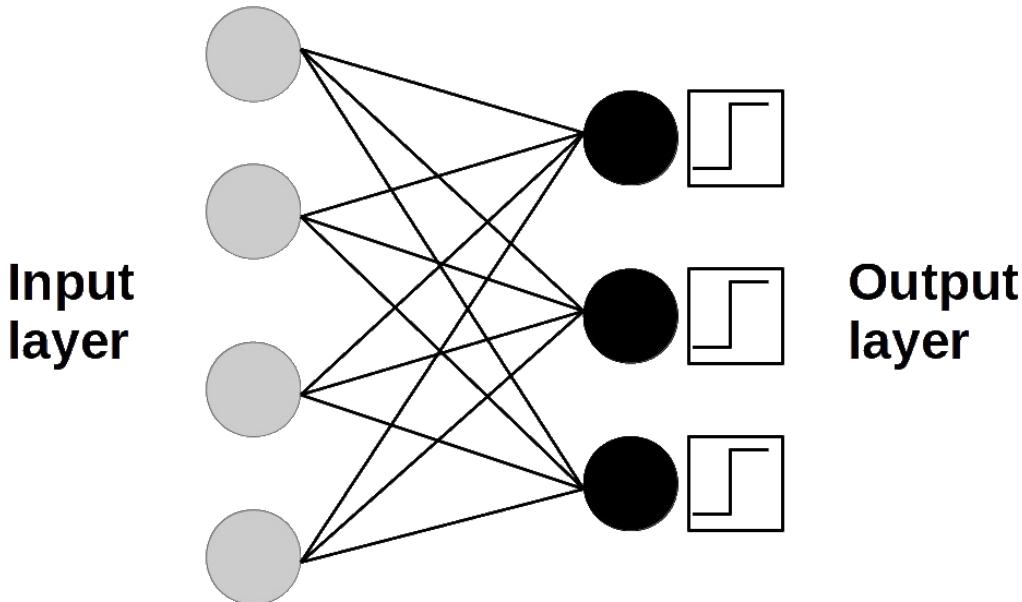
$$\sum_{i=1}^m X_i w_i = h = 0$$

For a 2D parameter space it results in:

$$X_1 = (W_b - X_0 W_0) / W_1$$

Note that the direction of activation is orthogonal to the linear separation.

The Perceptron algorithm



A single neuron **only performs a linear separation**, which is insufficient for many applications.

The simplest way to combine neurons on a single problem is to **stack them independently**. Each neuron is connected to the input vector with **its own set of weights**.

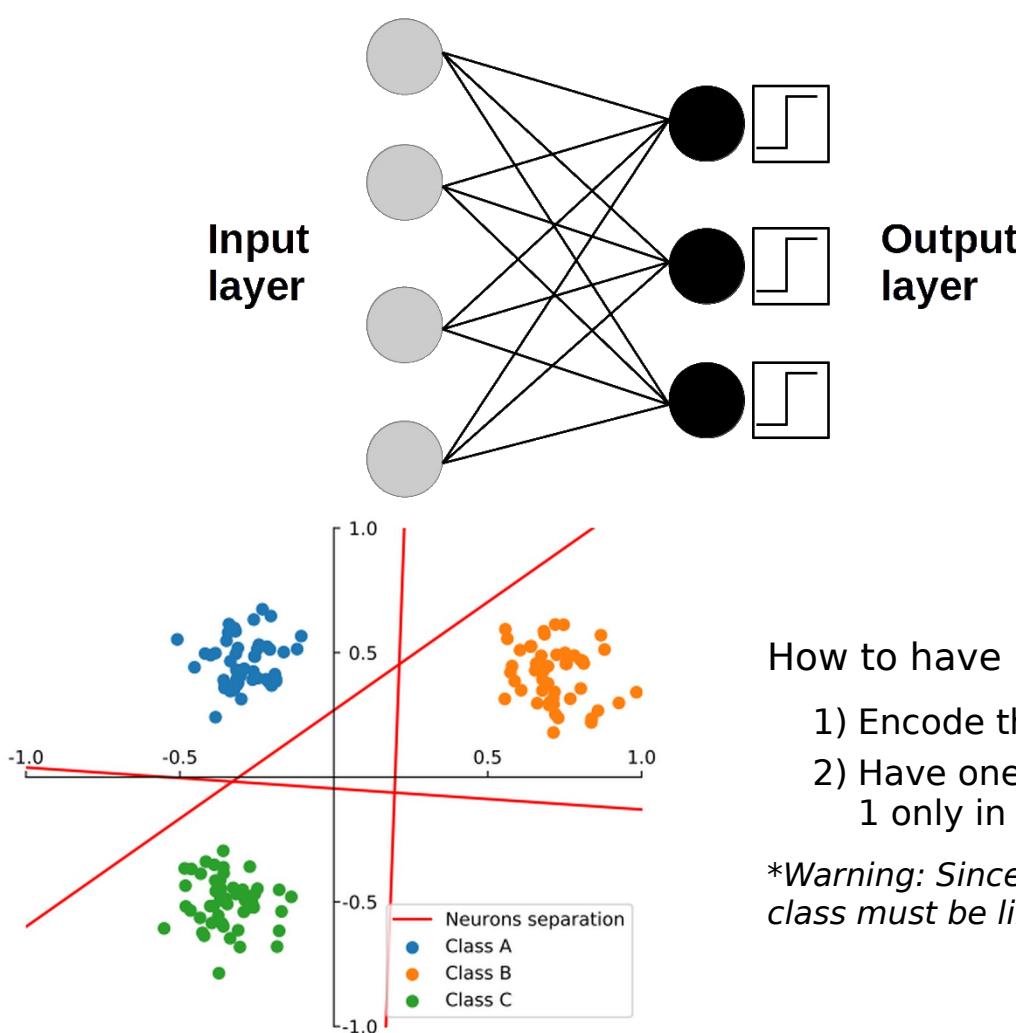
The training procedure is identical, but this time there is an index j to represent the neurons, and **the weights are now in the form of a 2D matrix**.

$$h_j = \sum_{i=1}^{m+1} x_i \omega_{ij} \quad a_j = g(h_j) = \begin{cases} 1 & \text{if } h_j > \theta \\ 0 & \text{if } h_j \leq \theta \end{cases}$$

$$\omega_{ij} \leftarrow \omega_{ij} - \eta (a_j - t_j) \times x_i$$

The combination of this training procedure and this neuron connection scheme is called the single layer "Perceptron" (Rosenblatt 1958).

The Perceptron algorithm



C1	C2	C3
1	0	0
0	1	0
0	0	1

How to have multiple neurons work on the same problem ?

- 1) Encode the output vector (e.g in binary format)
- 2) Have one neuron per output class and target a format with a 1 only in the appropriate class, e.g [1,0,0]-[0,1,0]-[0,0,1]

*Warning: Since each neuron only performs a *linear separation*, each class must be *linearly separable* from all the others.

The Perceptron algorithm

- **Initialization**

- Set the starting weights to small random values (positive and negative).
Can be drawn from a uniform or Gaussian distribution centered on zero.

- **Training**

- For a given number of steps T, or until the output is “correct”
 - For each input vector
 - Compute the activation of each neuron j using the activation function g :

$$a_j = g\left(\sum_{i=0}^{m+1} w_{ij}x_i\right) = \begin{cases} 1 & \text{if } \sum_{i=0}^{m+1} w_{ij}x_i > 0 \\ 0 & \text{if } \sum_{i=0}^{m+1} w_{ij}x_i \leq 0 \end{cases}$$

- Update each weight individually using :

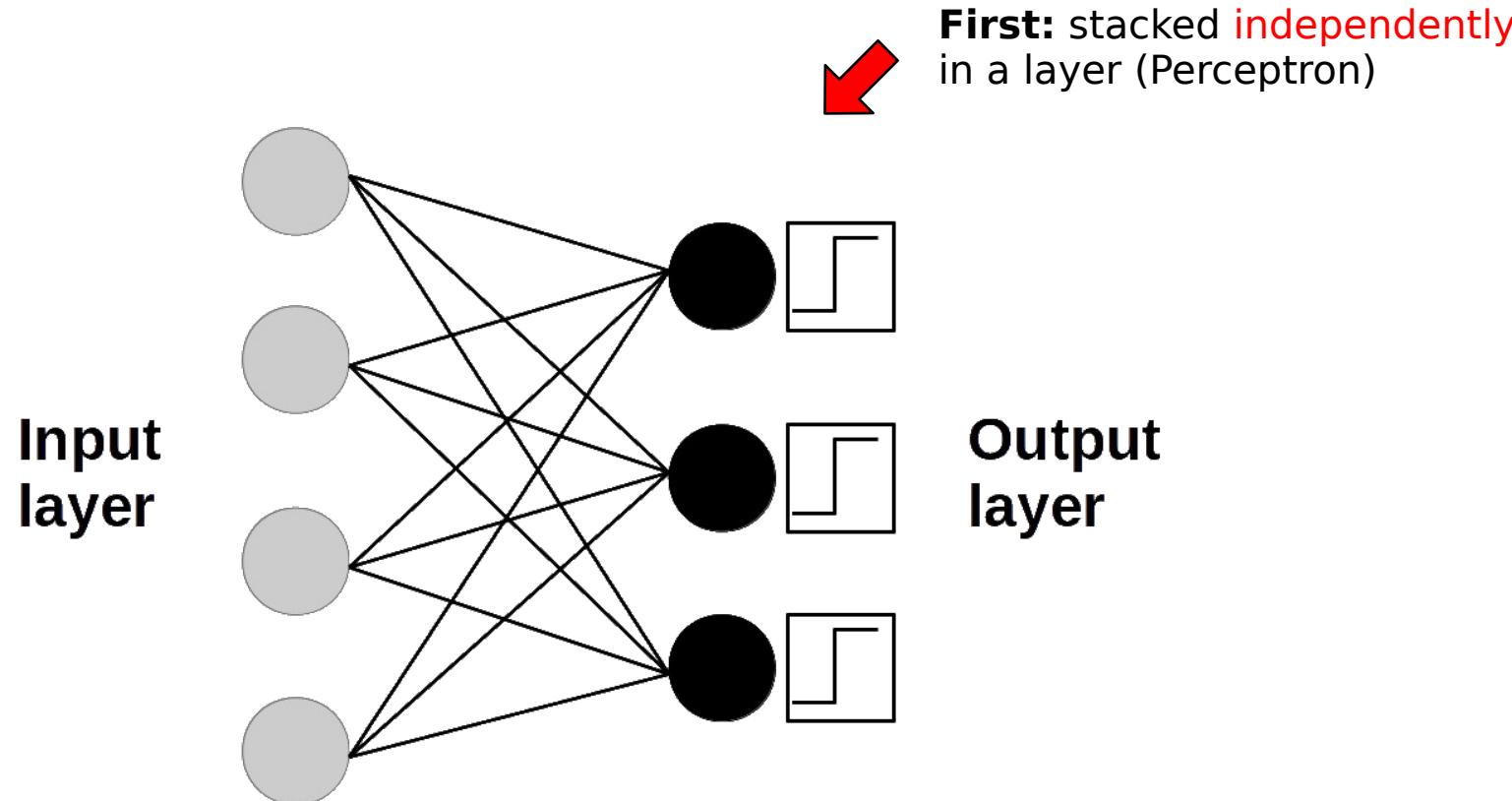
$$\omega_{ij} \leftarrow \omega_{ij} - \eta (a_j - t_j) \times x_i$$

- **Inference**

- Compute the final activation of each neuron j for each input vector to test

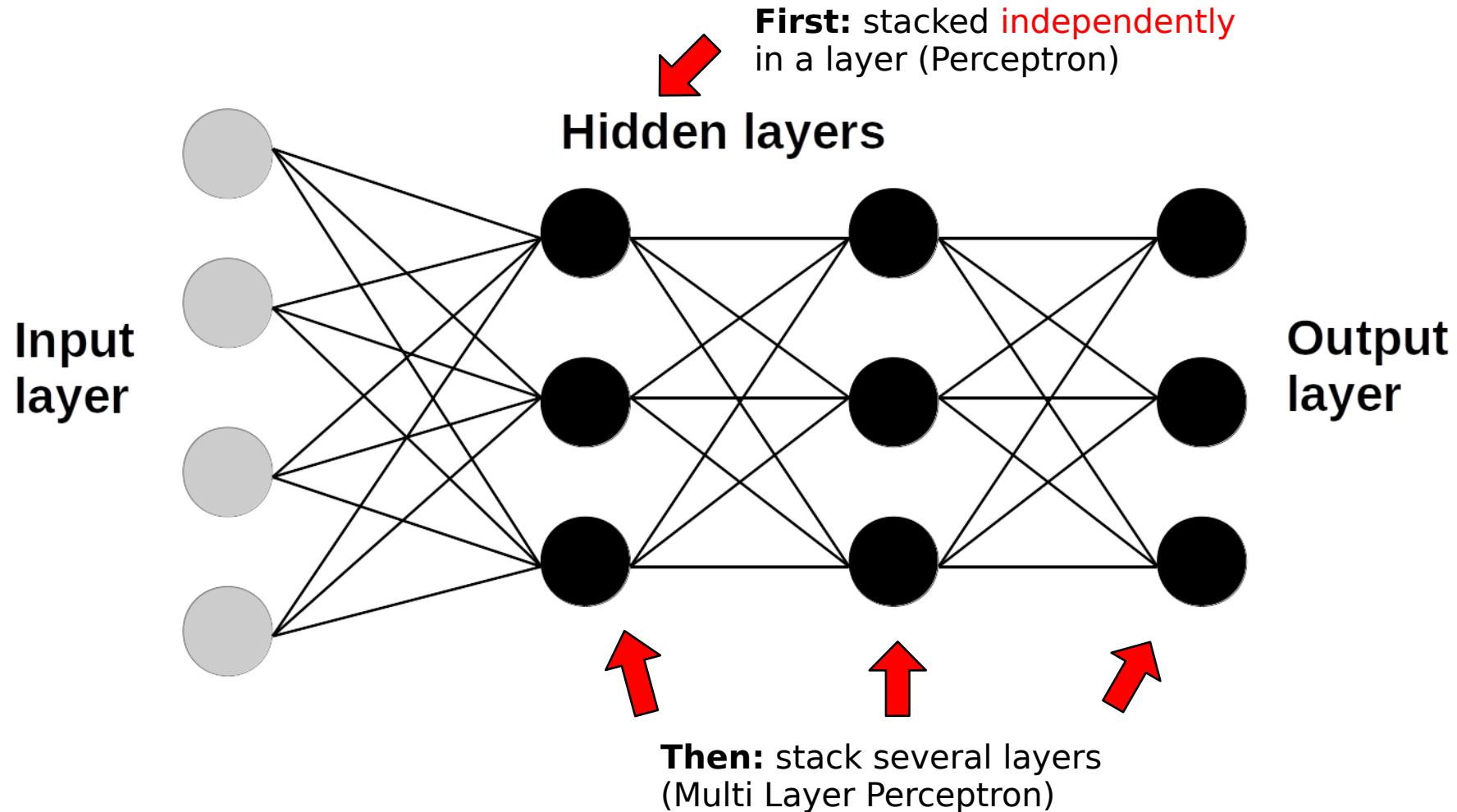
Constructing a “Deep” Neural Network

For more complex problems **more neurons must be used.**



Constructing a “Deep” Neural Network

For more complex problems **more neurons must be used.**



Constructing a “Deep” Neural Network

In addition to the MLP architecture, it is necessary to **change the activation** to allow **non-linear combinations**

→ **Change for a Sigmoid (logistic) function**

$$g(h) = \frac{1}{1 + \exp(-\beta h)}$$

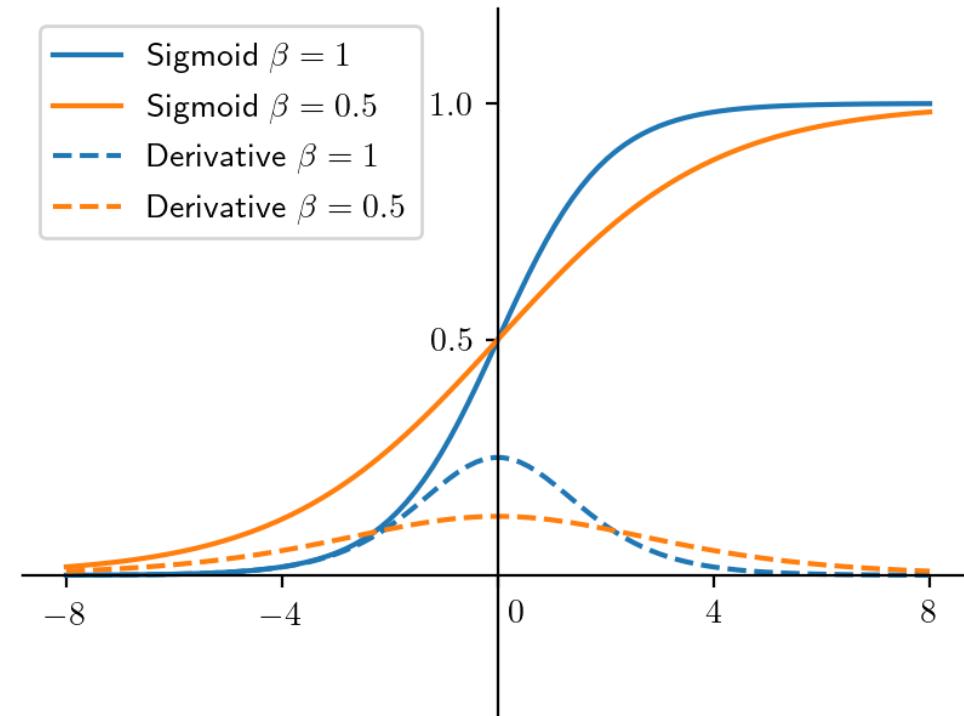
Beta represents the **slope** of the function.

The sigmoid is continuous while preserving the desired global binary behavior in its extremity.

It has a nice derivative form :

$$\frac{\partial a_j}{\partial h_j} = \beta a_j(1 - a_j)$$

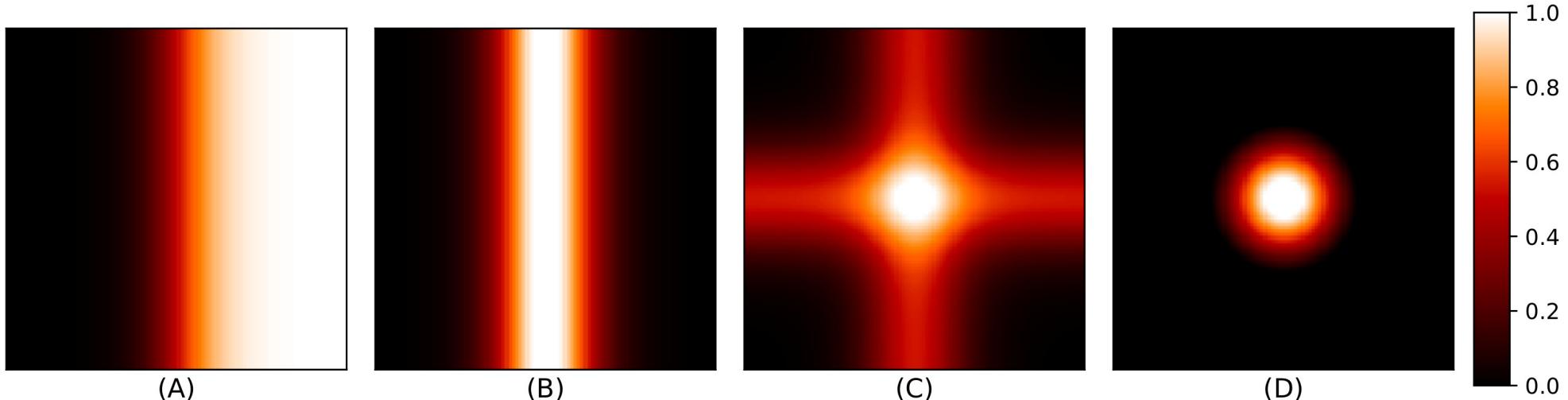
For regression problems, the output layer can either use sigmoids or simple linear activation functions since the previous layers will handle non-linearity.



Constructing a “Deep” Neural Network

Stacking layers that use a sigmoid activation allows the construction of **more and more complex functions**. The image below demonstrates that it is possible to reconstruct hill shapes (B), bumps (C), and point-like functions.

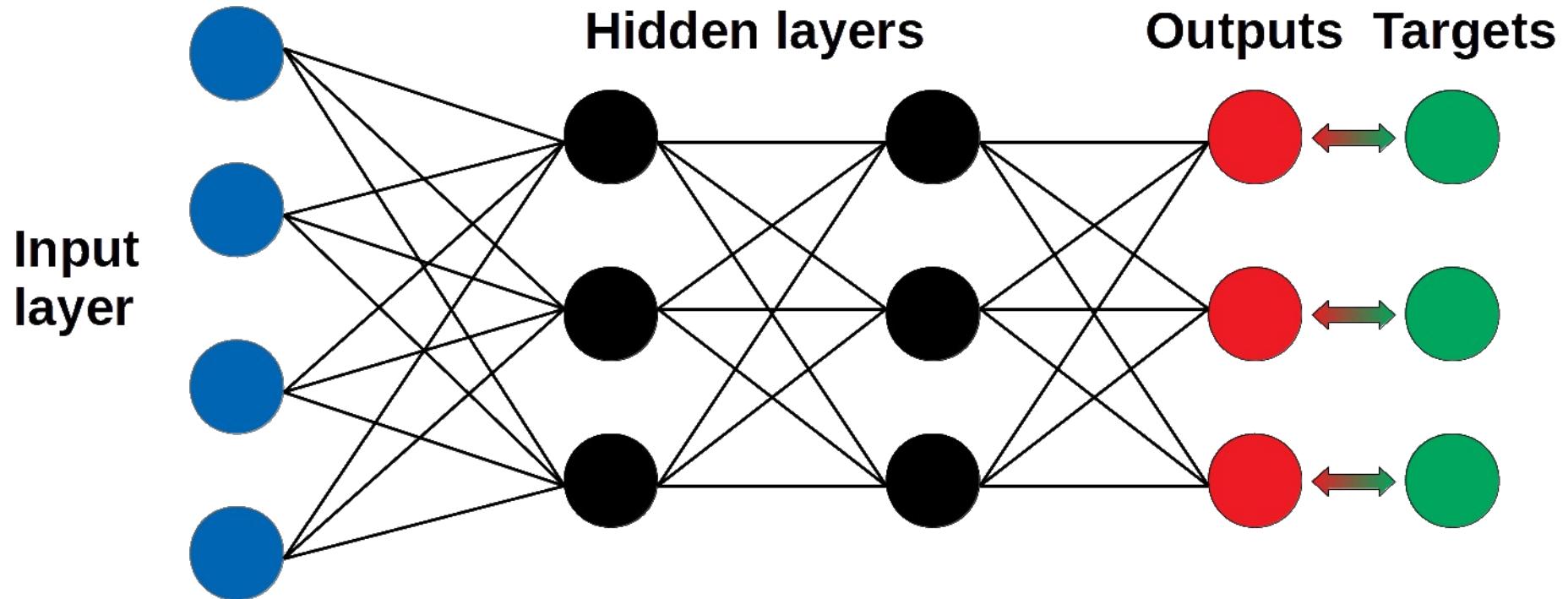
→ With these improvements, the MLP is a **Universal Function Approximator**



In practice, based on the universal approximation theorem, a MLP with a **single hidden layer** remains a UFA at the condition of having a large enough number of neurons.

Error gradient backpropagation

Problem : how to correct the output of neurons in the hidden layers ?



$$\omega_{ij} \leftarrow \omega_{ij} - \eta \frac{\partial E}{\partial \omega_{ij}} \quad \xrightarrow{\text{red arrow}} \quad \frac{\partial E}{\partial \omega_{ij}} = \delta_l(j) \frac{\partial h_j}{\partial \omega_{ij}} \quad \text{with} \quad \delta_l(j) \equiv \frac{\partial E}{\partial h_j} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial h_j} = \frac{\partial a_j}{\partial h_j} \sum_k \omega_{kj} \delta_{l+1}(k)$$

Actual Network equations for an MLP with a single hidden layer

Changes in the Perceptron algorithm required to obtain the Multi-Layer Perceptron algorithm :

- Change the activation function of all neurons, with $\beta \geq 1$

$$g(h) = \frac{1}{1 + \exp(-\beta h)}$$

- Add a hidden layer with its own bias node (connect input \rightarrow hidden, and hidden \rightarrow output)
- Implement the “back-propagation” with $E(a^o, t) = \frac{1}{2} \sum_{k=1}^{N_o} (a_k^o - t_k)^2$ using the following equations:

$$\delta_o(k) = \beta a_k^o (1 - a_k^o) (a_k^o - t_k)$$

$$\delta_h(j) = \beta a_j^h (1 - a_j^h) \sum_{k=1}^{N_o} \delta_o(k) \omega_{jk}$$

$$\omega_{jk} \leftarrow \omega_{jk} - \eta \delta_o(k) a_j^h$$

$$v_{ij} \leftarrow v_{ij} - \eta \delta_h(j) x_i$$

Example with classification

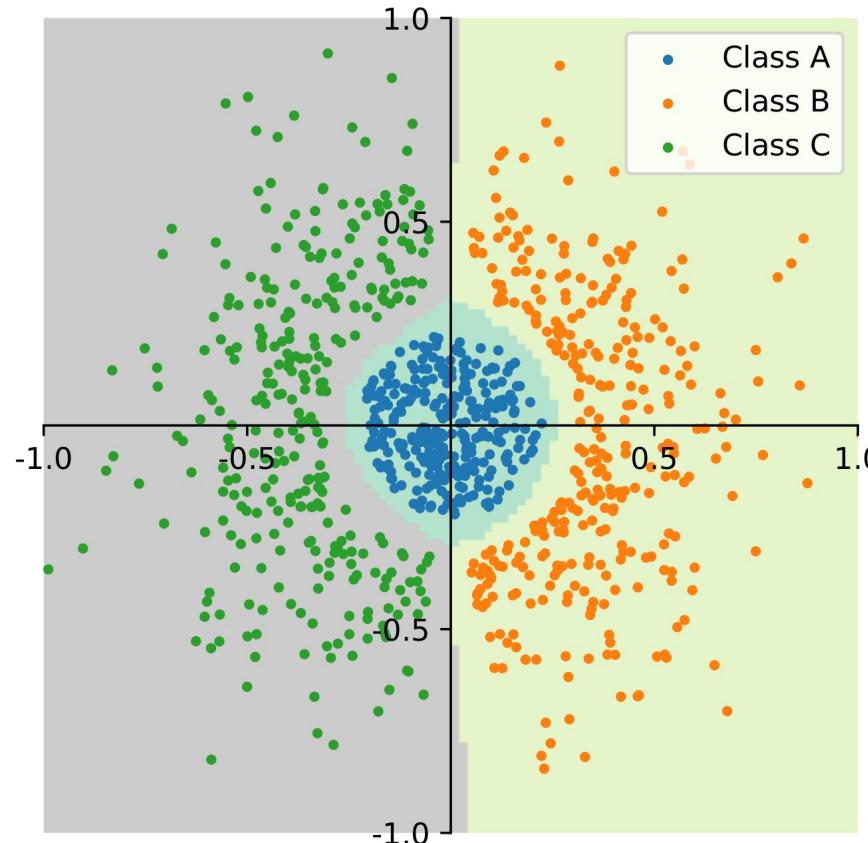


Illustration of a three-class separation in a two-dimensional feature space using a trained MLP with 3 output neurons and 8 hidden neurons, all with sigmoid activations. The light background colors indicate the regions of the feature space that the network has attributed to each class.

How to measure classification performance ?

→ Confusion Matrix

- Uses “**observational proportions**”
- Provides confusion between classes
- Is much more precise than the average accuracy

Predicted

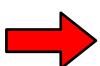
$$\text{Recall} = \frac{TP}{TP + FN}$$

Actual	Class	Positive	Negative	Recall
Positive	Positive	93	7	93.0%
Negative	Negative	46	954	95.4%
	Precision	66.9%	99.27%	95.18%

$$\text{Precision} = \frac{TP}{TP + FP}$$

$TP \equiv$ True Positive

$FP \equiv$ False Positive



$TN \equiv$ True Negative

$FN \equiv$ False Negative

66.9%

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

ML methods usually learn better on balanced dataset **BUT**

the results must always be represented using “**observational proportions**”, with imbalance.

Example: The Pima Indian dataset

Dataset from the UCI Machine Learning repository.

Provides 8 physiological measurements for 768 female native Americans and provides a class depending on whether the individual has developed diabetes.

The objective is to **predict if a person has diabetes** based solely on the 8 input measurements.

A simple **Perceptron** can reach **70% accuracy**, while an **MLP** with 8 hidden neurons with can reach **90% accuracy**.

Iteration : 1400

Confmat :

31	2	93.94
3	14	82.35

Precision 91.18 87.50 Accu 90.00

Average test set quadratic error: 0.158614

Average training dataset quadratic error : 0.135860

Attribute Number	Attribute
1	Patient age
2	Body mass index (kg/m ²)
3	Concentration of plasma glucose
4	2-h serum insulin (mu U/mL)
5	Thickness of triceps skin-fold (mm)
6	Pedigree function of diabetes
7	Number of times patient pregnant
8	Diastolic blood pressure (mmHg)
9	Class 0 or 1

Input dimension : 8 (+1)

*Output dimension : 1
(or 2 if using one neuron per class)*

Number of example : 768

Class distribution : 500 class 0; and 268 class 1

Example: The Iris dataset

Dataset from the UCI Machine Learning repository.

Provides 4 sizes for a set of 150 Iris flowers and provides membership to one of 3 classes.

The objective is to **predict the corresponding class** based on the 4 input measurements for each flower.

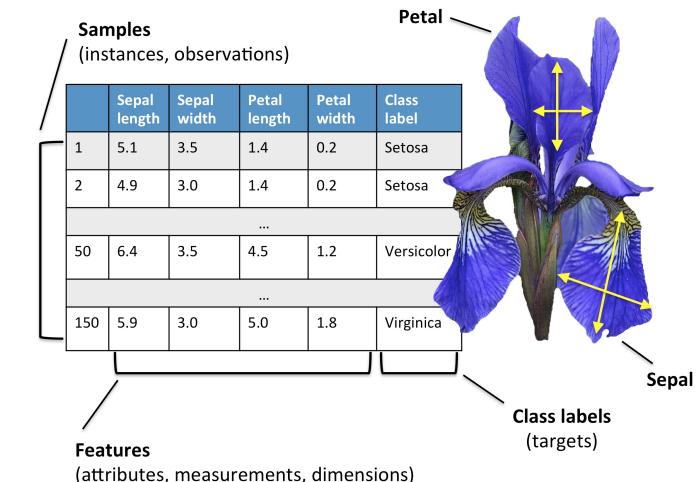
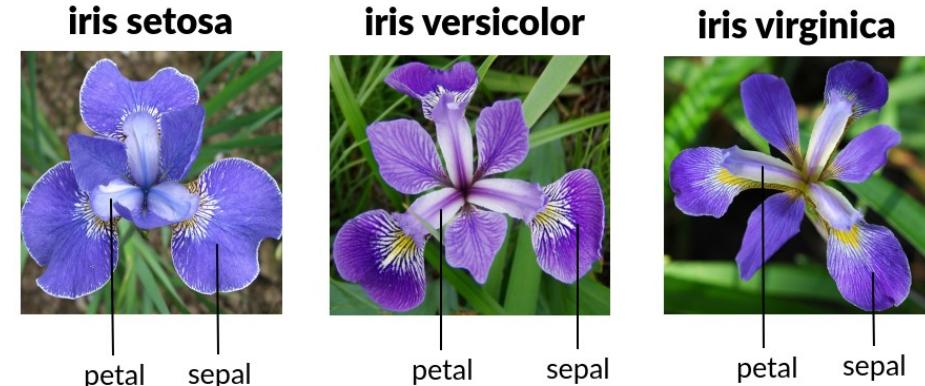
A simple **Perceptron** can reach **80% accuracy**, while an **MLP** with 4 hidden neurons can reach **100% accuracy**.

Iteration : 800

Confmat :

18	0	0	100.00
0	17	1	94.44
0	0	14	100.00

Precision 100.00 100.00 93.33 Accu 98.00



Input dimension : 4 (+1)

Output dimension : 3

Number of example : 150

Class distribution : 50 examples per class

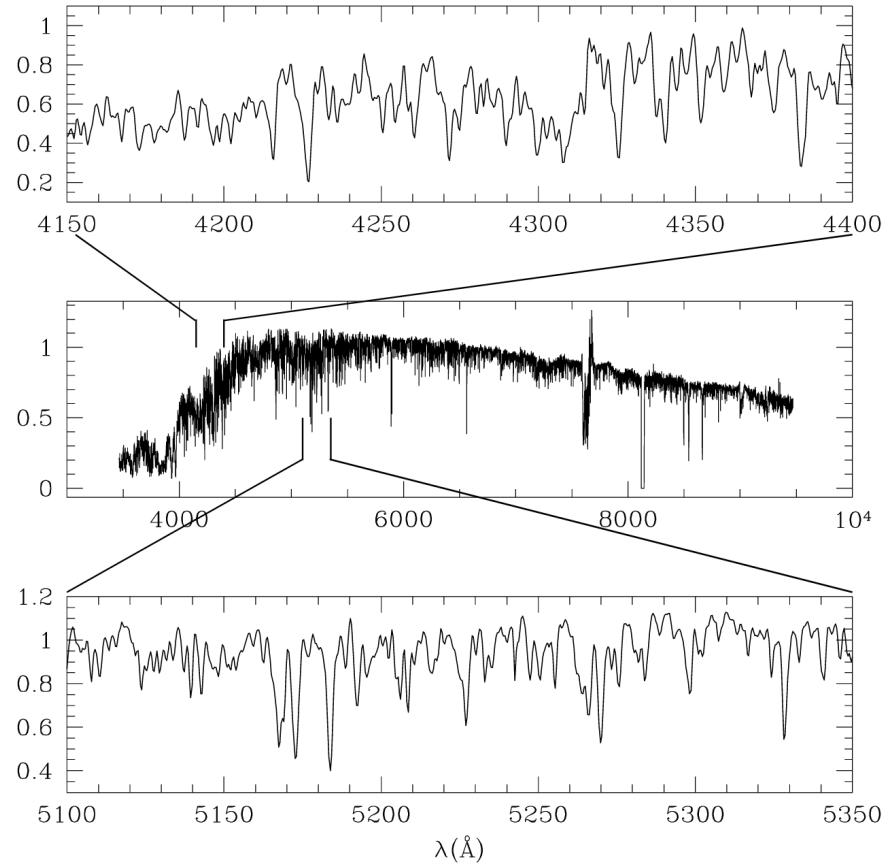
Example: The Spectra dataset

This dataset contains **stellar spectra** obtained with the 0.9m Coudé Feed telescope at Kitt Peak National Observatory, classified into various spectral types (*F. Valdes et al. 2004*).

The provided dataset here is a simplification that only keeps 1115 homogeneous spectra with half the resolution (3753 “pixels” remains).

The target only provides the 7 regular spectral types for classification to increase the number of examples per class. The dataset is highly imbalanced which require rebalancing the training dataset!

Confmat :							Precision	Recall	Accu
5	0	0	0	0	0	0	62.50	100.00	61.11
1	37	5	0	0	0	0	76.19	86.05	75.62
1	0	11	1	0	0	0	93.62	84.62	93.00
1	0	1	16	1	0	4	82.22	69.57	81.11
0	0	1	2	44	6	1	50.00	81.48	50.00
0	0	0	2	2	37	7	85.71	77.08	85.71
0	0	0	0	0	2	12			



Input dimension : 3753 (+1)
Output dimension : 7
Number of example : 1115
Class distribution : strong imbalance

Network depth and size

How to choose the **network architecture**?

In the case of the MLP, how many layers and neurons should be used per layer?

In this context, the objective is to find a **trade-off between**:

The **strength of the network** (its expressive power) and the **difficulty of training the model**

A few ideas that help explore the architectures:

- More data allows to constrain more parameters

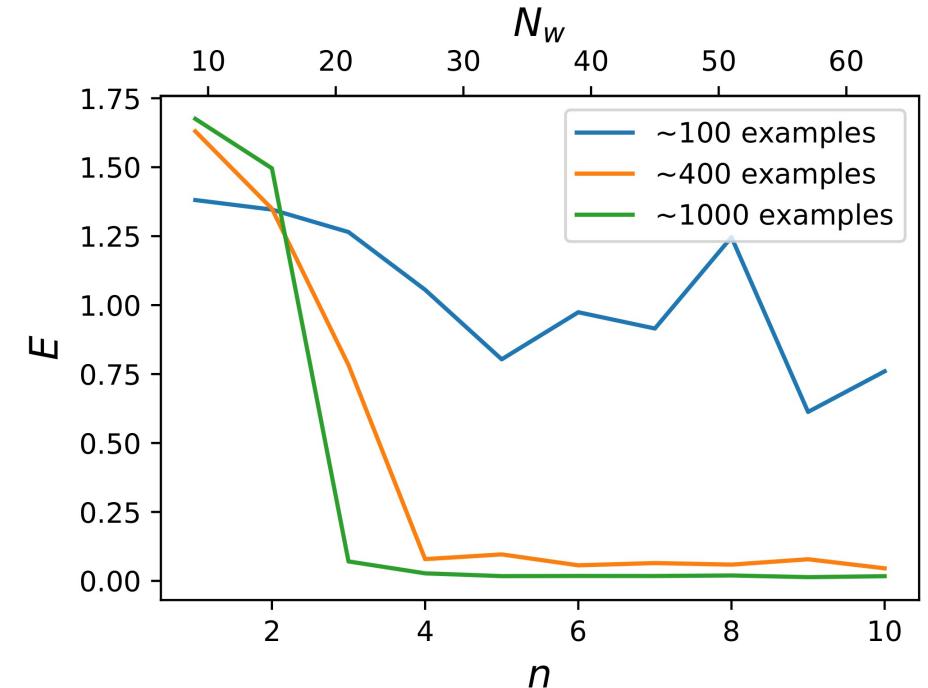
Rule of thumb: need roughly 10 times the number of weights as training examples

- More neurons increase the strength but add parameters to fit.

Reminder: a neuron is a linear separation in the parameter space → how many would you need by eye?
(starting point before exploration)

- More layers allow the construction of complex functions with fewer parameters, but increases training difficulty.

Reminder: a single hidden layer can approximate any function, but it is not always the optimal solution
(nb. of parameters, time to converge, etc.)



Finding the appropriate architecture is always an **exploratory and iterative process**.

Training different networks on the same data is almost always necessary to obtain good results.

Learning rate

Choosing the **appropriate learning rate** for the task is very important and must consider several aspects of the error space.

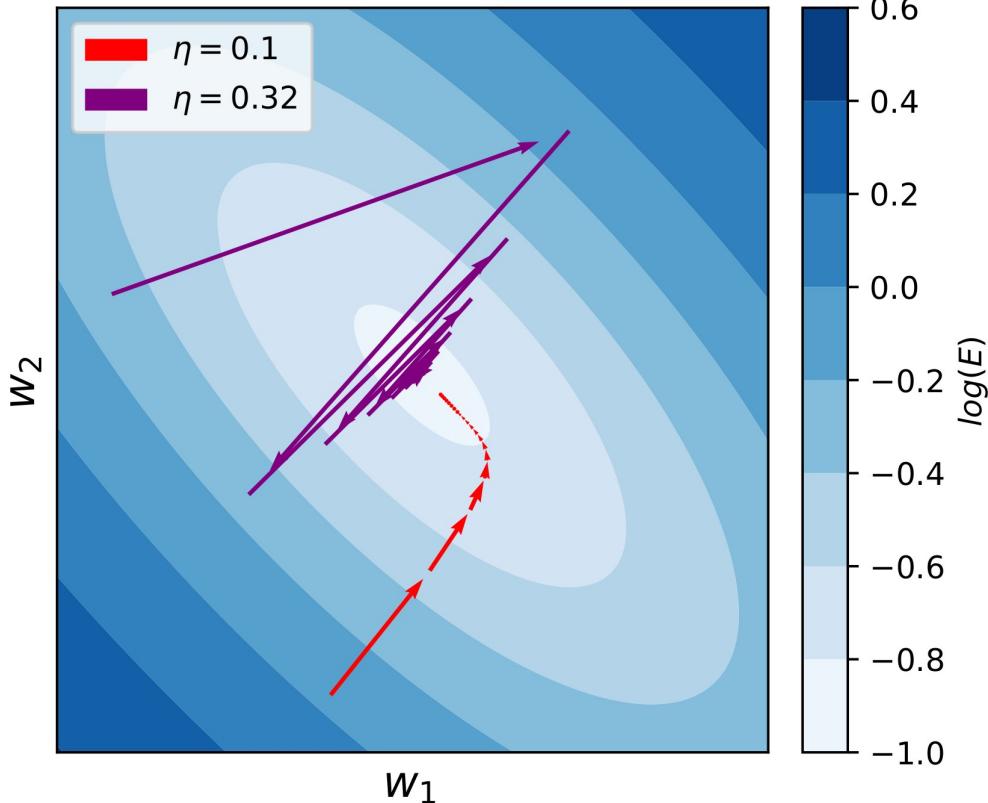
	Small η	Large η
Pros	<ul style="list-style-type: none">• Increase training stability• Can find narrower minima	<ul style="list-style-type: none">• Faster training• Easily switch from a local minima to another
Cons	<ul style="list-style-type: none">• Slow training• Might get stuck on local minima	<ul style="list-style-type: none">• Might fail to find a narrow minima• Training can be very unstable

The learning rate does not have to be a single value, it can **change** during the training phase.

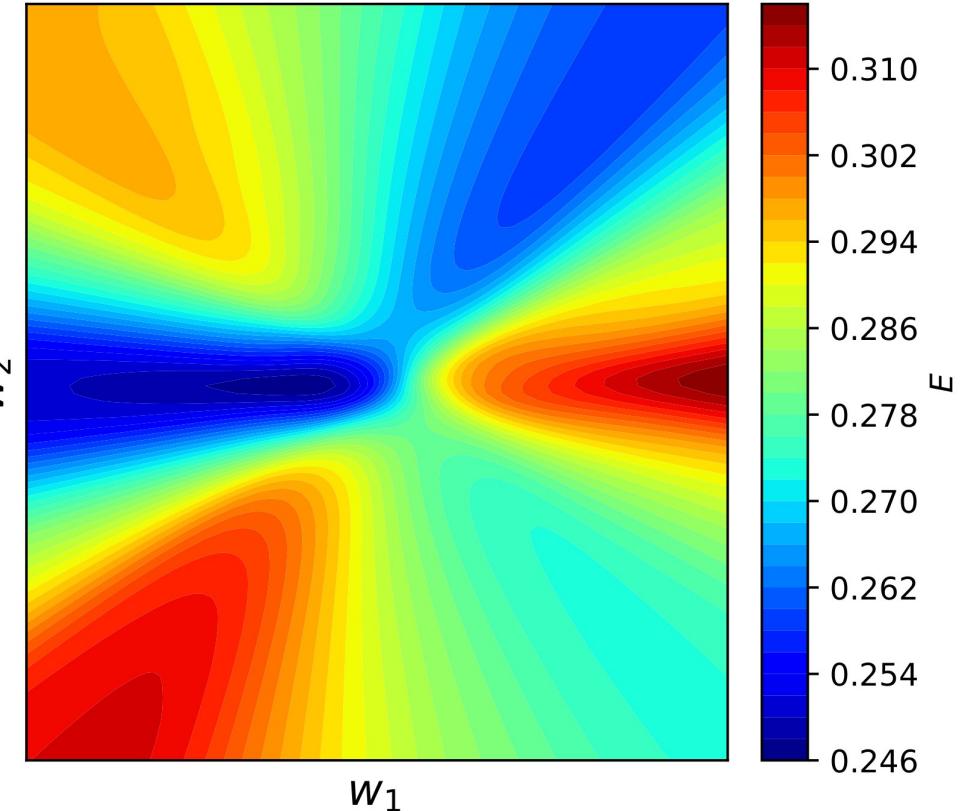
The simple approaches are: pre-established fixed values, linear decay, **exponential decay**, ...

Most frameworks allow the use of advanced gradient descent scheme that also automatically adapts the learning rate: Adam, RMSprop, Adagrad, etc ...

Learning rate



Logarithm of the error term in the weight space of a simple binary neuron with a two dimensional input parameter. The arrows represent the correction after each epoch for two different learning rates.



Error term in a 2D weight space of neuron in the hidden layer of a 3 class output MLP. The other weights of the network are frozen while exploring this 2D weight space.

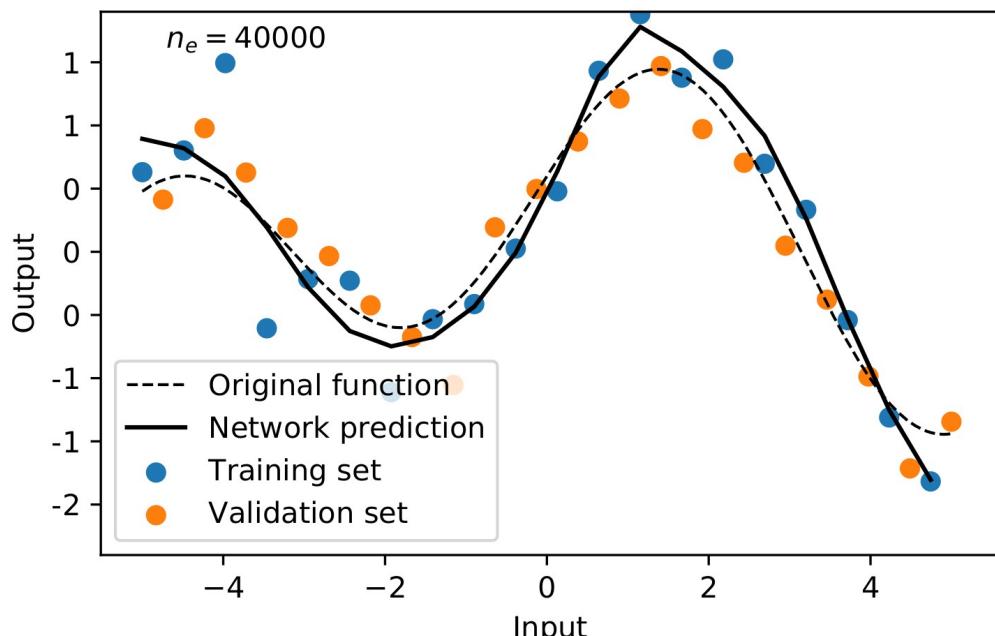
Overtraining



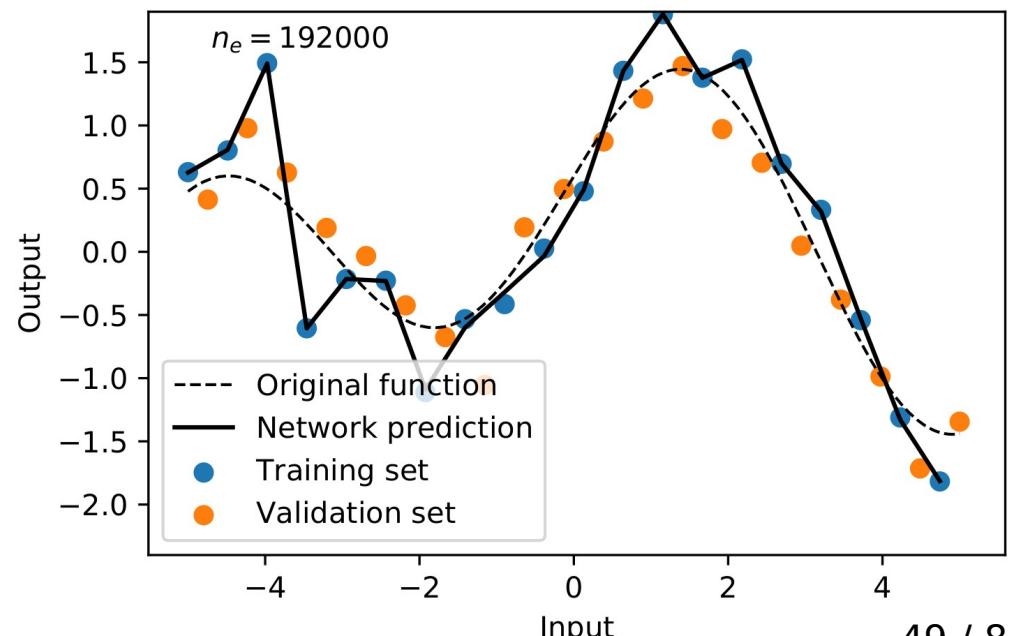
Overtraining: At some point during the training phase, the network that looks at the same data again and again, might start to over-fit!

It means that the network is learning specificities of the dataset that are no longer related to the average function that it should approximate.

Proper fit



Over-fitting

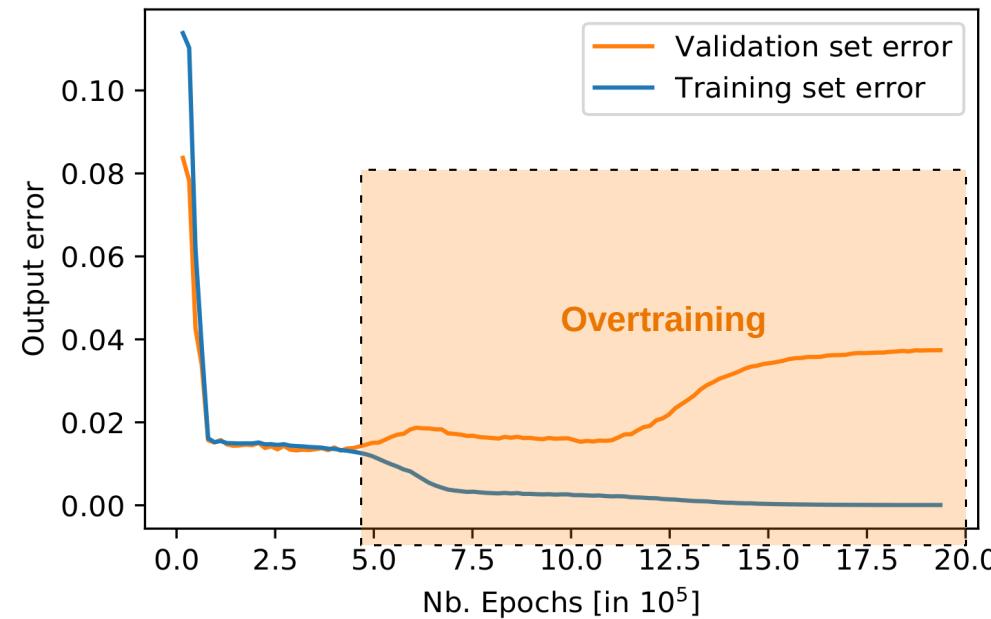
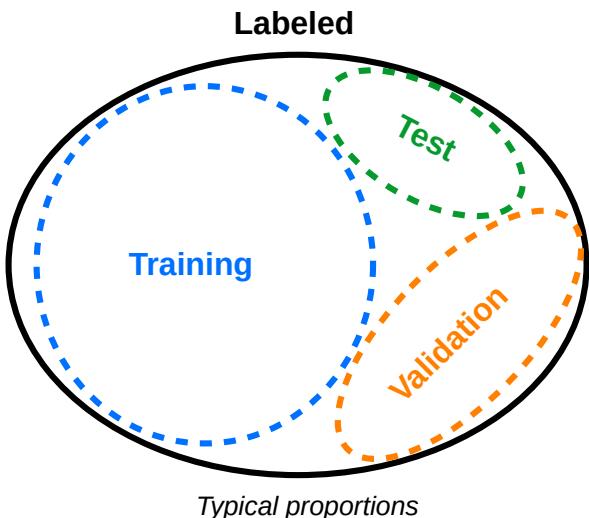


Monitoring overtraining

While several techniques can be used to moderate overtraining (like regularization), it is always **necessary to monitor it** to stop the training at the appropriate time.

Split **labeled dataset** into 3 subsets

- Training set** ➤ To train the network
- Validation set** ➤ To monitor the training process
- Test set** ➤ To assess the quality of the prediction

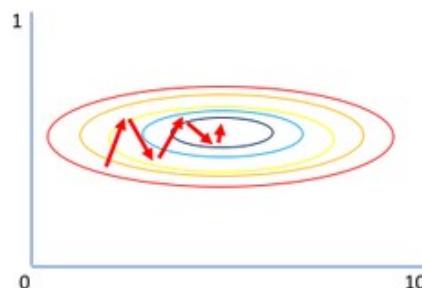


Normalization

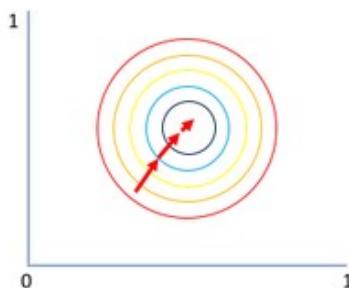
Why normalize ?

It is always necessary to **normalize each dimension of the dataset**, usually in the range [0,1] or [-1,1].

Otherwise, some dimensions might dominate the weight update gradient, eclipsing the information contained in the others



Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

Different approaches to rescaling

Centered:

$$X - \text{mean}$$

Standardized:

$$\frac{X - \text{mean}}{\text{sd}}$$

Normalized:

$$\frac{X - \min(X)}{\max(X) - \min(X)}$$

WARNING !

All labeled data must be normalized with the same values (not only with the same equation)!

This include both the training and testing samples but also the data used for prediction at deployment time!