TUTORIAL
Supervised Artificial Neural Networks for classification

David Cornu

Institut UTINAM, Univ. Bourgogne Franche-Comté, OSU THETA, Besançon, France

Semi-hackathon ASOV 2020

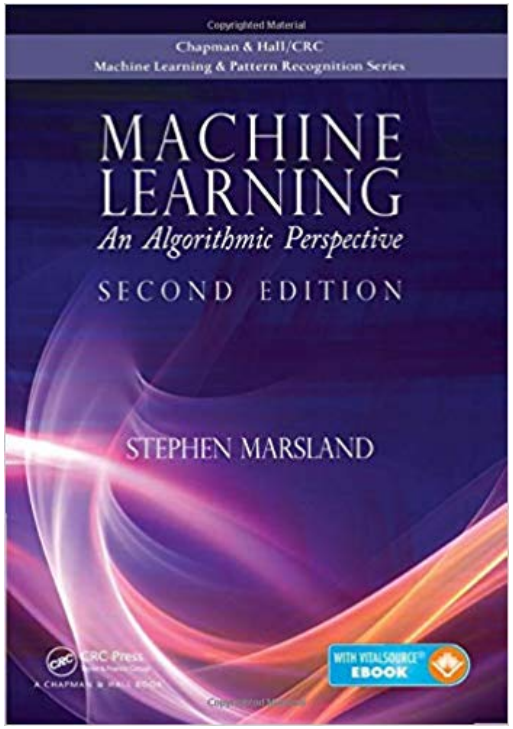
Tutorial materials

Support code, test datasets and slides for this tutorial are available at:

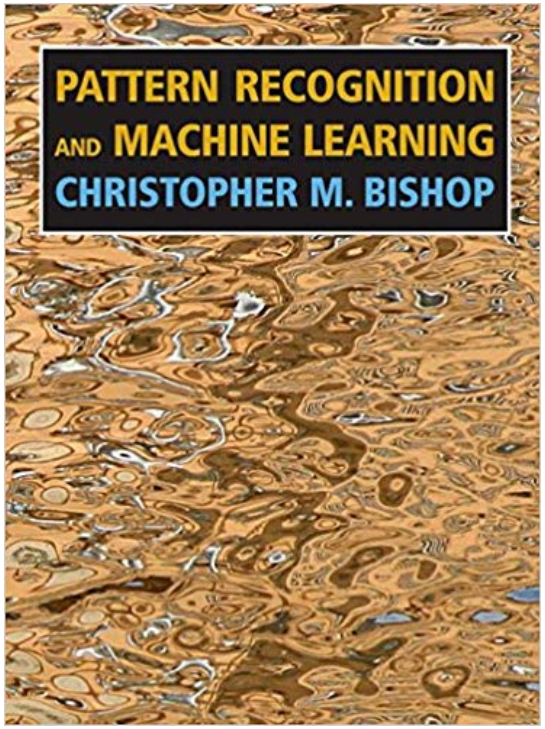
https://github.com/Deyht/ASOV_2020

Example codes are provided in **Python3**, **C** and **Fortran** for convenience.

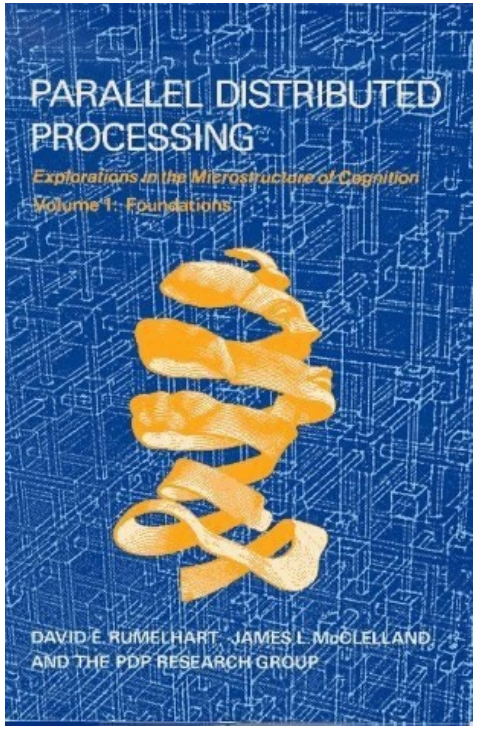
Some references



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



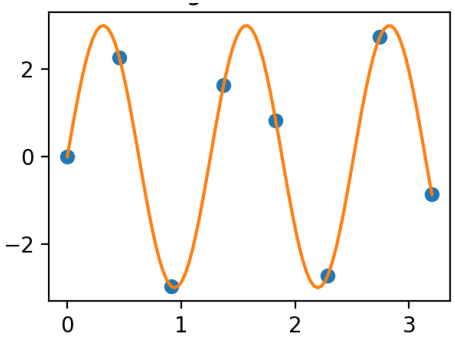
Christopher M. Bishop. **Pattern Recognition and Machine Learning**. Springer, 2011



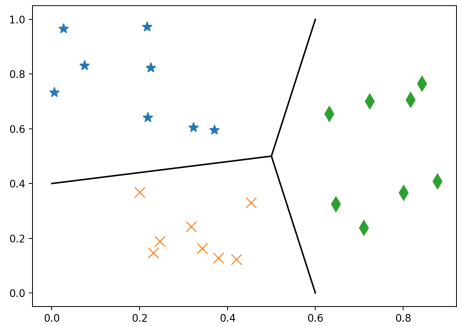
David E. Rumelhart et al. **Parallel Distributed Processing**. MIT press, 1989

Machine Learning capabilities

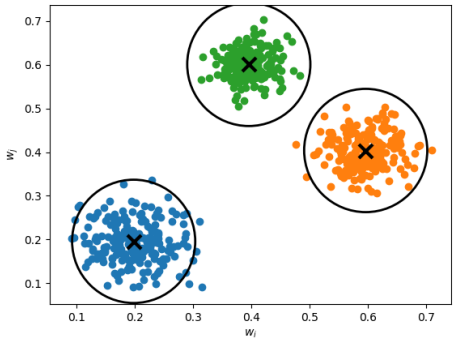
Regression



Classification



Clustering



Time series prediction

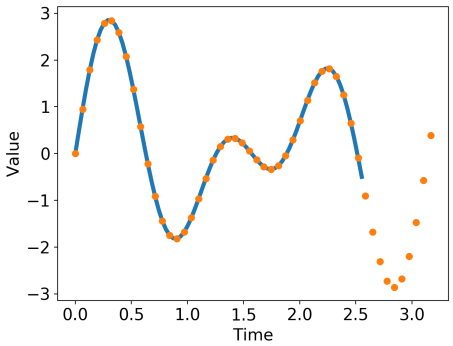
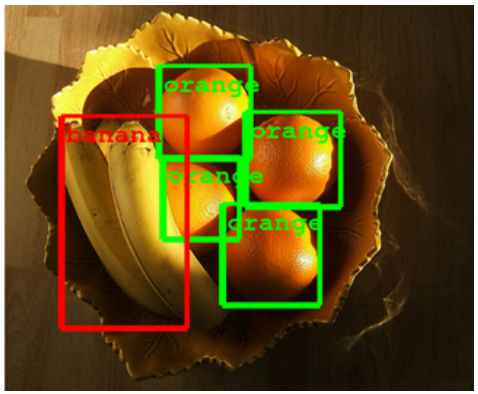


Image recognition



Frameworks / Tools

julia

mxnet

K

DEEP LEARNING

theano

Microsoft CNTK

Caffe2

Chainer

Machine Learning

$\Rightarrow$ **Core concept:** extract statistical information about a dataset and adapt the response accordingly through a learning process

Supervised:

- A training set with the expected targets is provided
- Try to find a generalization to get correct prediction most of the time

Unsupervised:

- Dataset without targets
- Try to find similarities in the input and categorize them together

Learning definition ?

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

* *Trésor de la Langue Française informatisé TLFi*

Learning definition ?

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

* *Trésor de la Langue Française informatisé TLFi*

Overall, (animal) learning can be summarized by three capabilities :

Learning definition ?

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

** Trésor de la Langue Française informatisé TLFi*

Overall, (animal) learning can be summarized by three capabilities :
- **Remember** - **Adapt** - **Generalize**

Learning definition ?

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

** Trésor de la Langue Française informatisé TLFi*

Overall, (animal) learning can be summarized by three capabilities :
- **Remember** - **Adapt** - **Generalize**

Recognize a situation that has been experienced before (having seen these data), **remember** the adopted **behavior** (having produce this output) and evaluate if it was **appropriate** (correct output).

Last step is the **generalization** that constructs similarities between situations.

The brain example

"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 example

"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."

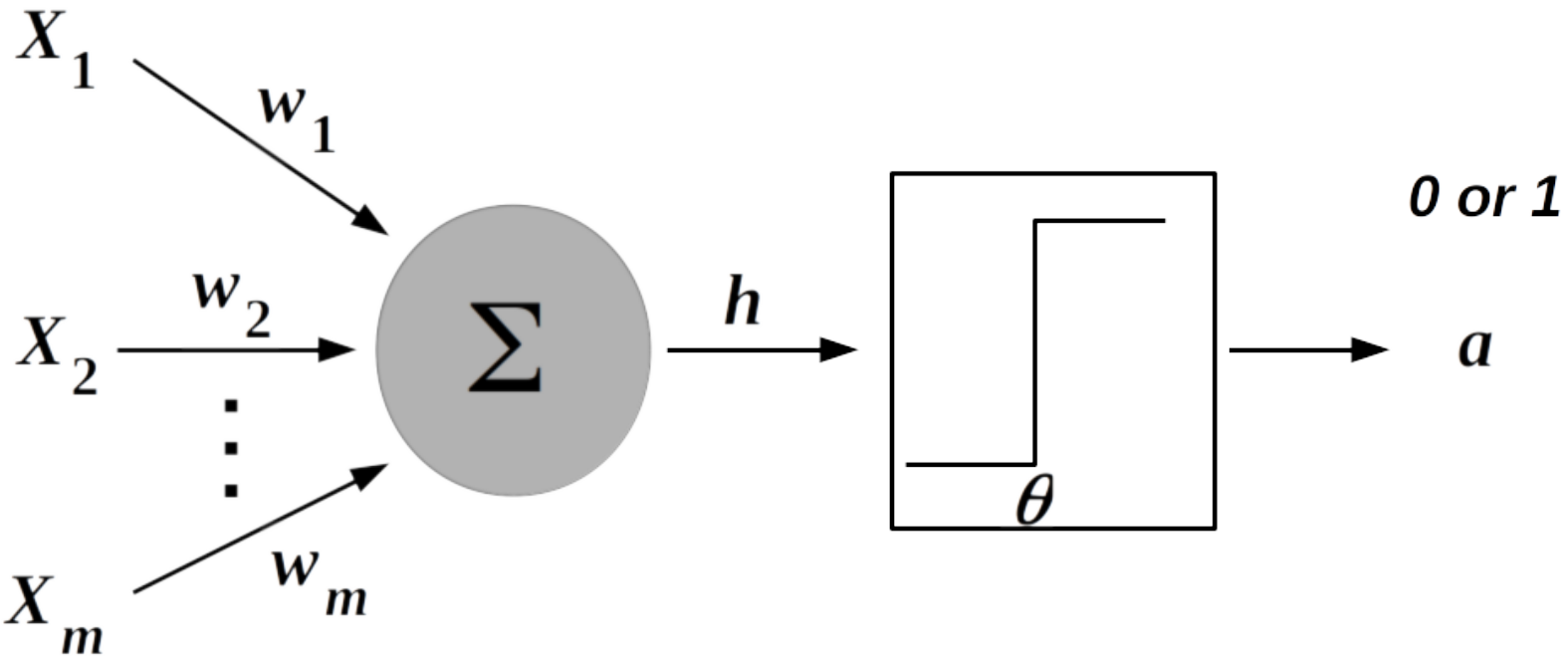
** Stephen Marsland*

The brain model has some very interesting aspect for science:

- Able to deals with noisy or incoherent data
- Deal with a large number of dimensions at the same time
- Mostly appropriate prediction depsite the previous points
- Provide results very quickly
- Remain robust with lose of neurons due to aging

Artificial Neural Network

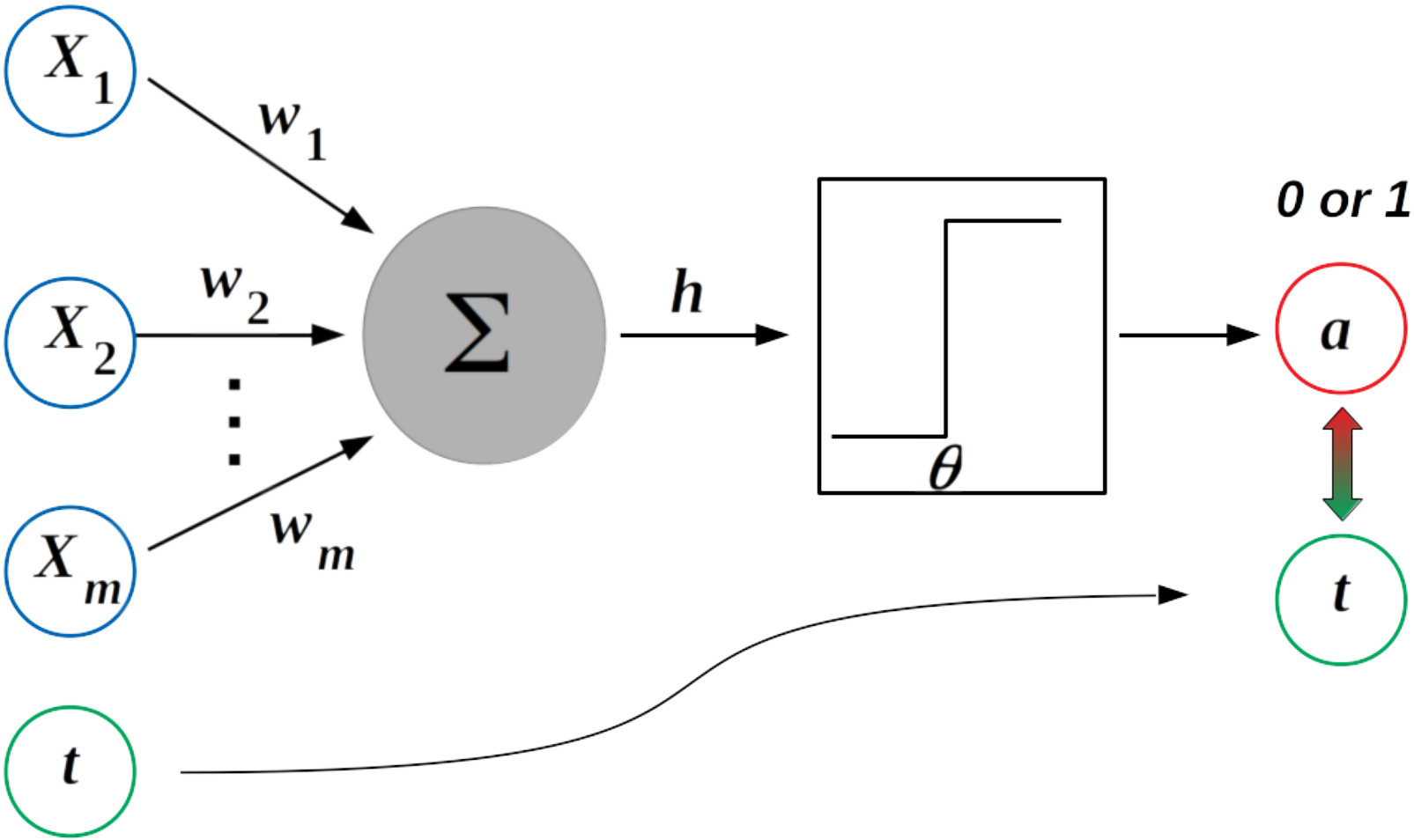
ANN are a famous way to implement Machine Learning
The basic element of such a network is the **neuron**.
A given input vector $(X_1, X_2, ..., X_m)$ in the training set is associate with a target t



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

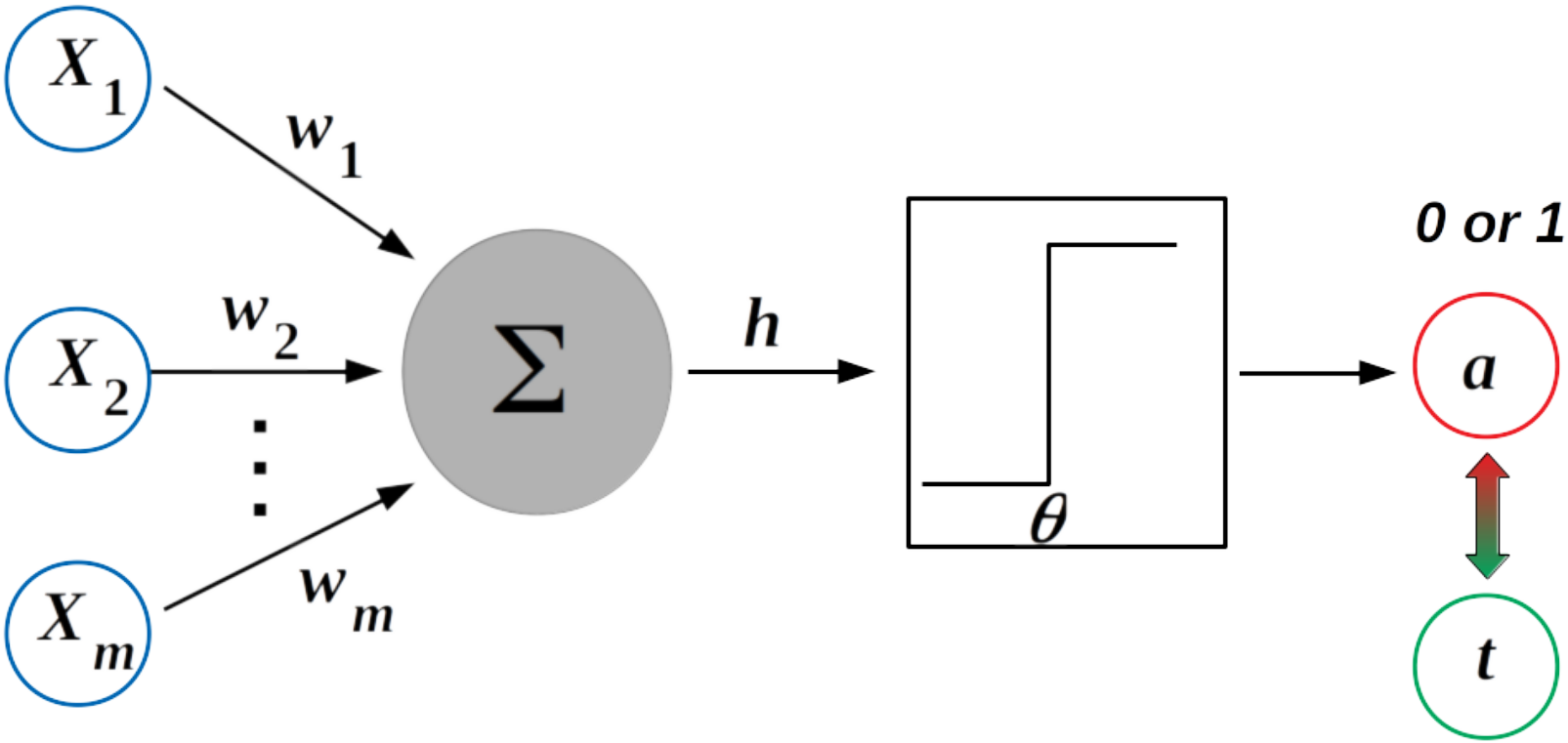
Artificial Neural Network

ANN are a famous way to implement Machine Learning
The basic element of such a network is the **neuron**.
A given input vector $(X_1, X_2, ..., X_m)$ in the training set is associate with a target t



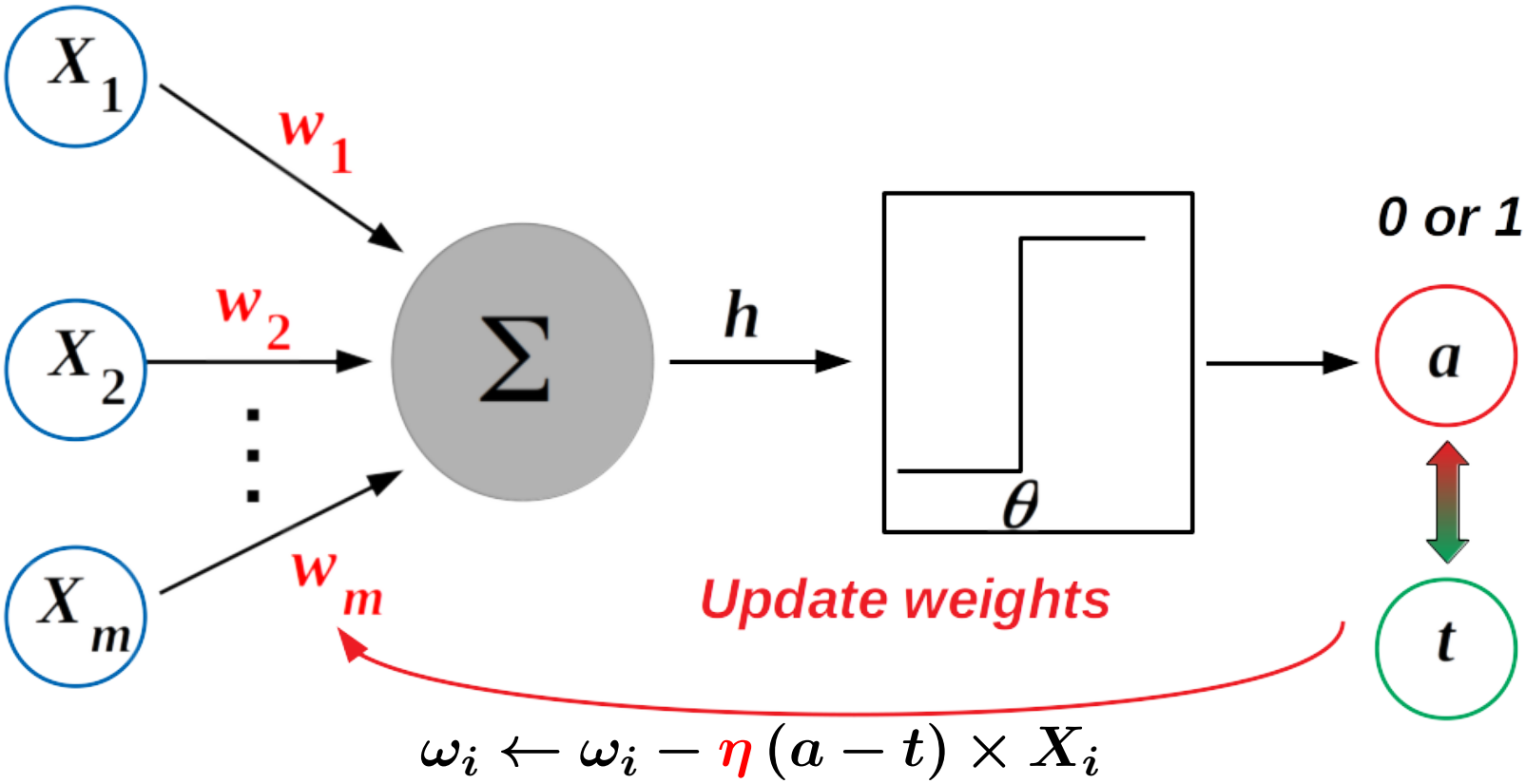
Artificial Neural Network

ANN are a famous way to implement Machine Learning
The basic element of such a network is the **neuron**.
A given input vector $(X_1, X_2, ..., X_m)$ in the training set is associate with a target t



Artificial Neural Network

ANN are a famous way to implement Machine Learning
The basic element of such a network is the **neuron**.
A given input vector $(X_1, X_2, ..., X_m)$ in the training set is associate with a target t



Learning rate

The learning rate η is a parameter that allows to quantify at which pace the algorithm updates the weights.

- **A too high value causes instabilities**
- **A too low value slows down the training**

Appropriate value depends on the specific algorithm in use, but for regular on-line and non too deep networks $0.1 < \eta < 0.4$ is usually working great.

The upper limit value will also depend on the noise level expected in the training data.

The Perceptron

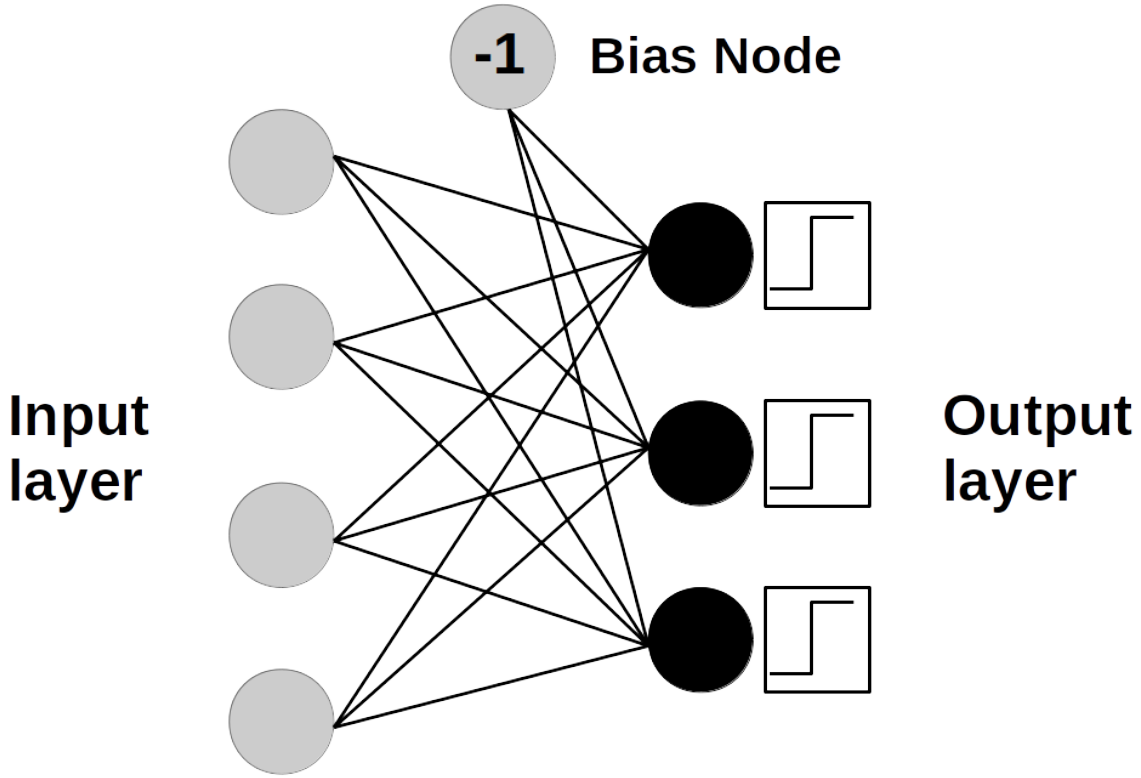
One neuron of the previous type just represents a linear separation in the input feature space, which is unlikely to be sufficient for most of the use cases. The easiest way to add neurons is in the form of a Layer. The neurons added this way are independent, and each of them add a linear separation in the input feature space, but they can be used as a all to *encode* information.

- Neurons are independent
- Each neuron has its own weights vector
- input and output dimensions are independent
- Input and output dimensions are defined by the problem to solve
- The output is a scheme of 0 and 1 that encodes the expected information

Biased input

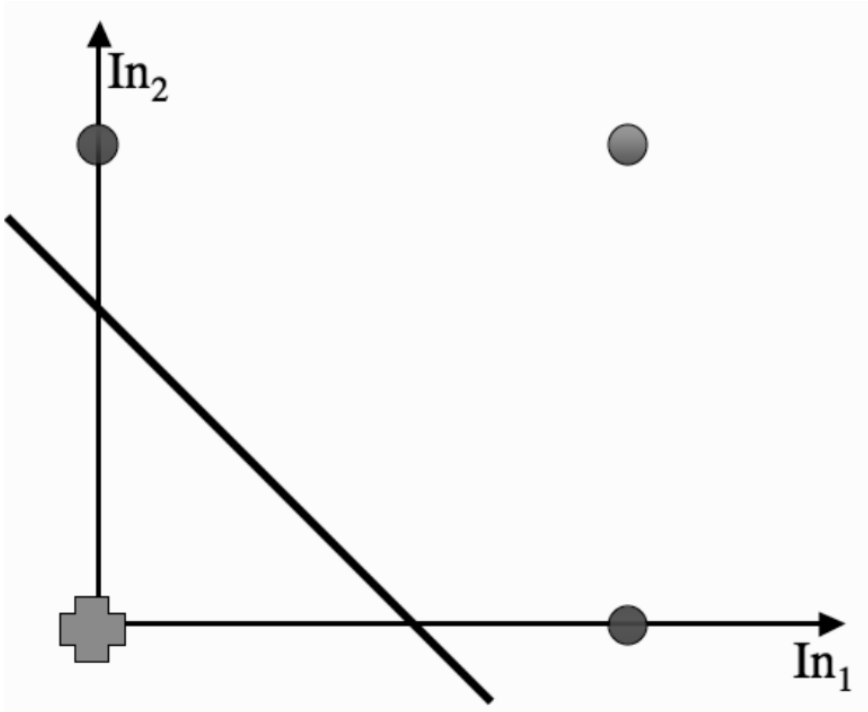
Issue:
If all inputs are equal to 0, then all the neurons behave the same way. To overcome this effect, the activation limit θ can be changed but it is difficult to implement.

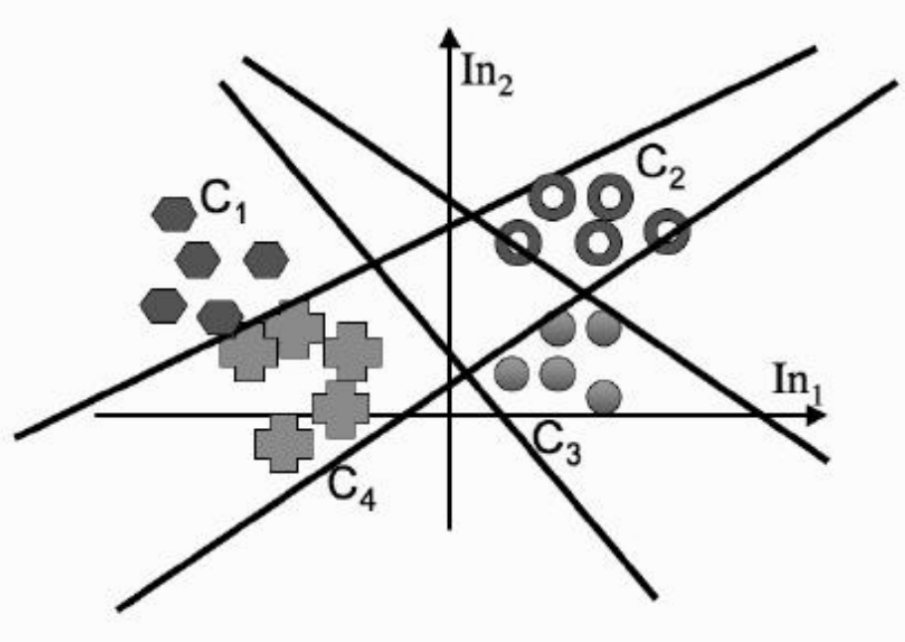
Solution:
One can add a bias node that has a fixed value -1 for all the inputs and its own weights. It allows each output neuron to tune his behavior for inputs near zero, acting as a shifting of the linear separation in the feature space.



Note on linear separability

Perceptron's neurons are only linear splittings in the feature space.





It is then necessary to represent data in an ingenious way to unsure their separability (try with the example of the exlusive-OR logic door).

Perceptron: Algorithm

- Initialization:
  Define all the weights w_{ij} to small random values (positve and negative)
- Training:
  - For T iterations (or until the prediction is good enough)
    - ⋆ Shuffle the dataset
    - ⋆ For every input vector (an epoch):
      - · Compute the activation function g of each neuron j:

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

      - · Update each weight individually :

$$w_{ij} \leftarrow w_{ij} - \eta\,(y_j - t_j) \cdot x_i \tag{2}$$

- Test:
  Compute the activation of each neuron for each input vector to get the final prediction on the dataset

Data pre-processing

In a classification case, an efficient way to represent the output is to have **one neuron per output class**. In the case of a 3 class problem the target will then be a vector of 3 elements in the form of : A (1,0,0), B (0,1,0), C(0,0,1).

An other key aspect of data pre-processing is the normalization of the input vector regarding each feature. It allows the different features to get the same initial impact on the network.In most of the cases a normalization between -1 and 1 with 0 mean is the most efficient.

The other preparations are dataset specific. A usual one is to **make groups** out of features that are not significant enough in their current state. On good example is age, that usually works better when using slides of 5 or 10 years.

Finally, for some datasets, networks are more stable with only part of the features when there is too much overlapping information between them.

Classification output: Confusion Matrix

		Predicted		
	Class	True	False	Recall
Actual	True	80	20	80.0%
	False	7	93	93.0%
	Precision	92.0%	82.3%	86.5%

$$\text{Recall} = \frac{TP}{TP + FN} \qquad \text{Precision} = \frac{TP}{TP + FP}$$

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

$TP \equiv$ True Positive $\quad TN \equiv$ True Negative

$FP \equiv$ False Positive $\quad FN \equiv$ False Negative

Datasets properties

All the datasets here correspond to classification problems. The list is ordered by increasing difficulty (i.e the level of fine tuning necessary to get nice results out of the problem).

IRIS - Difficulty : Easy
This dataset proposes to classify iris flowers into 3 categories based on 4 features that represents leafs properties. It contains 150 examples that distributes homogeneously over the output classes.

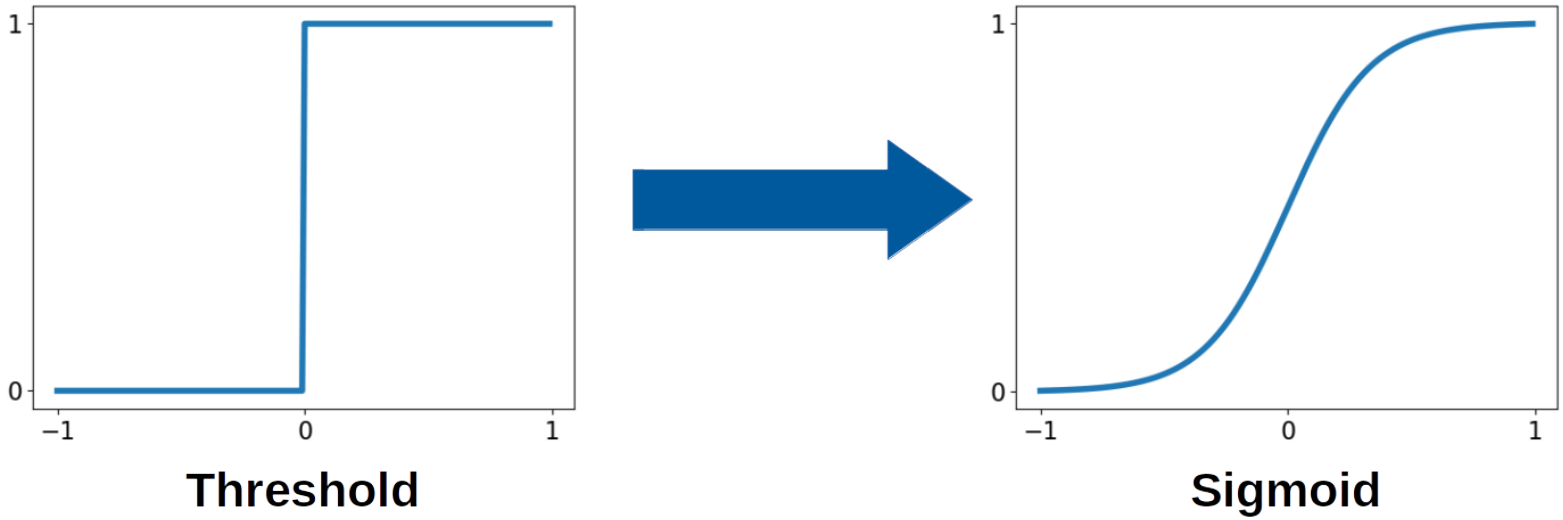
PIMA - Difficulty : Medium
This dataset proposes to detect persons who suffer of diabetes in a population of female native Americans (classification with two categories) based on 8 features that represents various indicators like age, the number of childrens, or different biological measurements. It contains 768 examples that are not represented equally (less people with diabetes).

Stellar spectral types - Difficulty : Hard
This dataset propose to classify stellar spectra obtained with the 0.9m Coudé Feed telescope at Kitt Peak National Observatory onto various spectral types. The provided dataset here is a simplification that only keeps 1115 homogeneous spectra with half the resolution (3753 "pixel" remains). The provided target only provides the 7 regular spectral types for classification.

Deep Learning

Construct deeper networks require two major improvements:

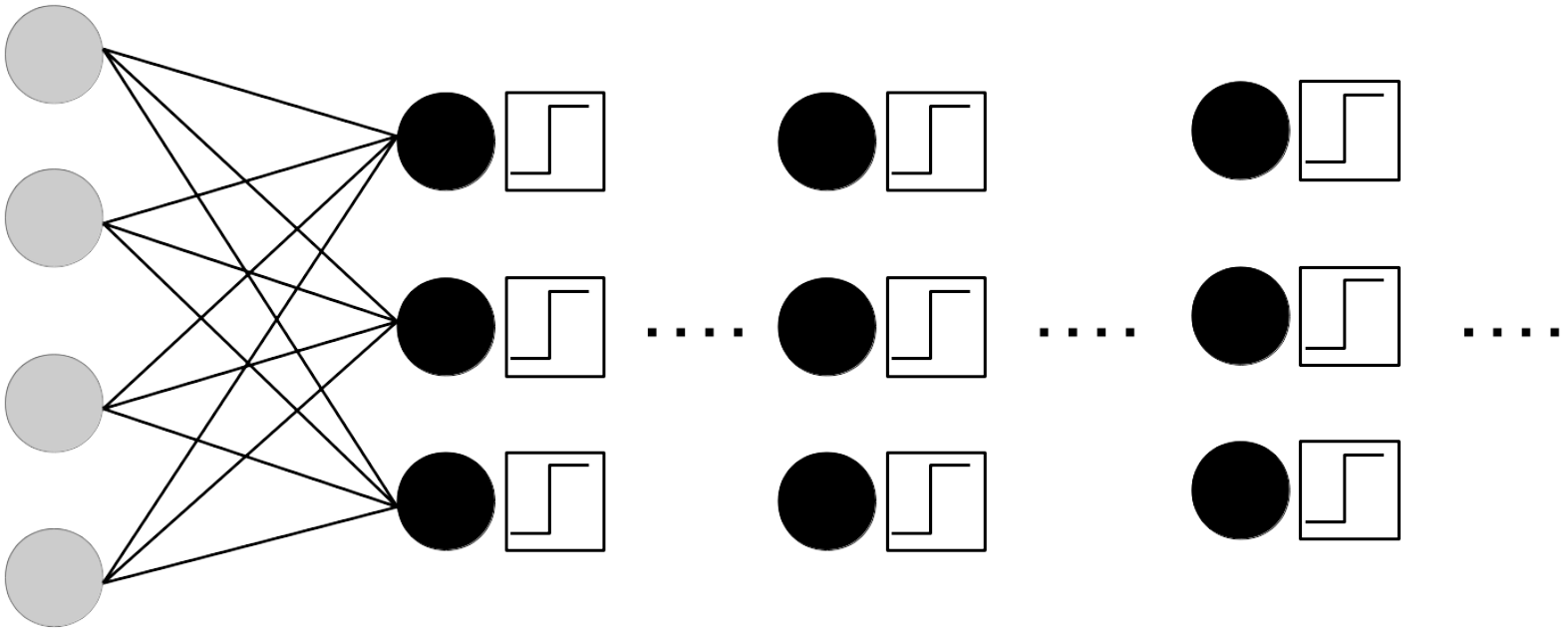
- Change the activation function: use a continuous function that conserves the global binary behavior $\Rightarrow$ Sigmoïd.



Deep Learning

Construct deeper networks require two major improvements:

- Change the activation function: use a continuous function that conserves the global binary behavior $\Rightarrow$ Sigmoïd.
- Adding a new hidden layer of neurons between the input vector and the output layer.

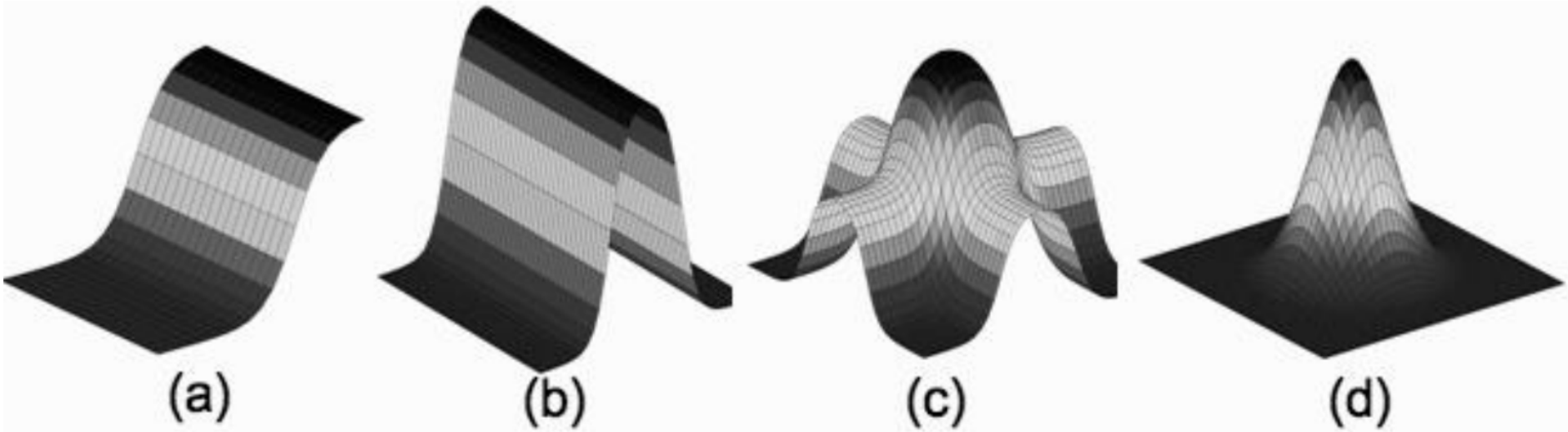


Deep Learning

Construct deeper networks require two major improvements:

- Change the activation function: use a continuous function that conserves the global binary behavior $\Rightarrow$ Sigmoïd.
- Adding a new hidden layer of neurons between the input vector and the output layer.

These additions allows non linear combination, making this new network a **"Universal Function Approximator"**.



Multi Layer Perceptron

Issue : how to compute the error of the hidden layer ?

Multi Layer Perceptron

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial h_j}\frac{\partial h_j}{\partial \omega_{ij}} \qquad \delta_l(j) \equiv \frac{\partial E}{\partial h_j} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial h_j} = g'(a_j)\sum_k \omega_{kj}\delta_{l-1}(k)$$

Algorithm changes

The previously described additions transcript into the following changes:

- Change of the activation function :

$$a_k = g(h_k) = 1/\big(1 + exp(-\beta h_k)\big)$$

  with $\beta >= 1.0$ that defines the slope of the sigmoïd.

- Adding a new layer with his set of weights and a bias node.
- Adding the "back propagation" based on the quadratic error $E(a,t) = \frac{1}{2}\sum_{k=1}^{N}(a_k - t_k)^2$ and update the weights of the two layers accordingly:

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

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

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

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

Overtraining : dataset splitting

Training for too long causes the network to **overtrain**!
$\rightarrow$ it learns either the noise or some specificities of the dataset.
It then looses track of a proper generalization. **To overcome this issue the data may be split into sub datasets :**

- **Train**: Biggest part of the data that is used for the training phase
- **Validation**: Smaller part that is used during the training to monitor the error of the network on it. The training must be stopped if this error rises too much.
- **Test**: Used at the end to asses the generalization capacity of the network

Usual splittings can be $50 : 25 : 25$ or $60 : 20 : 20$

Matrix formalism and batch

Learning on each object individually is especially slow in terms of raw compute speed.

Most of the frameworks use a "batch" training formalism, that considers the input as a matrix of the input vectors. Then almost all the network operations can be expressed in matrix formalism.

This highly speed up the networks training. Many library that propose parallellized matrix operations (OpenBlas). This is also the formalism used in most GPU accelerated frameworks.

Stochastic Gradient Descent (SGD)

The previous algorithm shuffles the training set and then learn on one epoch, which is necessary to break ordering effects.

An other approach that has shown better efficiency is the SGD. Using this method, an input vector is randomly picked from the dataset and used to train the networks once. This process is repeated several times with the possibility to pick the same vector several times in a row.

Stochastic Gradient Descent (SGD)

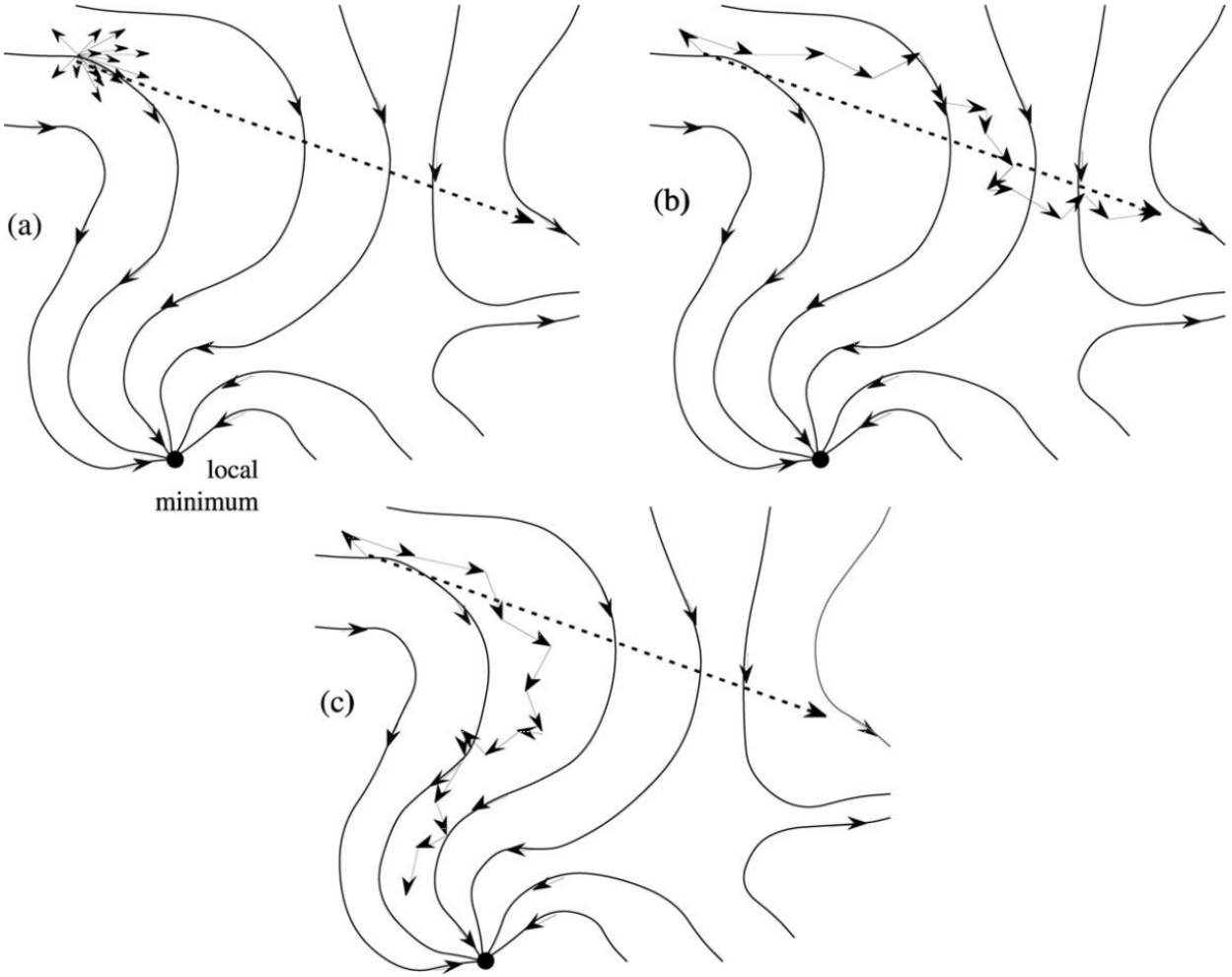


Fig. 4. Example of changes in weight space. The directed curves indicate the underlying true gradient of the error surface. (a) Batch training. Several weight change vectors and their sum. (b) Batch training with weight change vectors placed end-to-end. Note that batch training ignores curves and overshoots a 'valley,' thus requiring a smaller learning rate. (c) On-line training. The local gradient influences the direction of each weight change vector, allowing it to follow curves.

See Wilson & Martinez (2003) for more details

Stochastic Gradient Descent (SGD)

The previous algorithm shuffles the training set and then learn on one epoch. This is necessary to break ordering effect.

An other approach that has shown better efficiency is the SGD. Using this method, an input vector is randomly picked from the dataset and used to train the networks once. This process is repeated several times with the possibility to pick the same vector several times in a row.

As often, the best solution is the combination of the previous two, with the "mini batch" method. It constructs random small groups of input that are periodically shuffled, and train the network using the matrix formalism on each group at each epoch.

Imbalance learning

Machine learning algorithms are made to work on balanced datasets.

Imbalance learning

Machine learning algorithms are made to work on balanced datasets.

Example on disease detection, the majority of the tested persons are not sick AND the cure presents risks $\Rightarrow$ **must avoid to give unnecessary medication**

Imbalance learning

Machine learning algorithms are made to work on balanced datasets.

Example on disease detection, the majority of the tested persons are not sick AND the cure presents risks ⇒ **must avoid to give unnecessary medication**

		Predicted		
	Class	Unhealthy	Healthy	Recall
Actual	Unhealthy	8	2	80%
	Healthy	7	93	93%
	Precision	53.3%	97.9%	91.8%

It is often more efficient to re-balance the training dataset to manually avoid dilution or to give more importance to some classes.

However, results of a classification must be tested on a true use case scenario using "Observational proportions"

Probabilistic output

Unsupervised learning with neural networks