Another «simple» classifier: Perceptron

Perceptron is a «building block» of Neural Networks

- NN is **a class** of ML algorithms belonging to both the categories of **supervised and semi-supervised models** (depending on specific implementations/algorithms)
- The learned model f(x) is an **algebraic function** (or a set of functions), rather than a **boolean** function, as for DTrees. The learned function is **linear** for **Perceptron** algorithm, **non-linear** for **the majority of other NN** algorithms.
- In general, both features and output function(s) are allowed to be *real-valued* (rather than only discrete, as in Dtrees). So, with neural networks we can train **regressors**.
- The simple Perceptron model is, however, a binary classifier.
- By establishing a **cut-off** value on a continuous output, we can still obtain a classifiers (e.g. if $y<y_c$ then c=positive, else c=negative)

History of Neural Network models

- NNs are similar to biological neural systems **which are the** most robust learning systems we know.
- Initially, it was an attempt to understand natural biological systems through computational modeling.
- Allow massive parallelism for computational efficiency.
- Help to understand the "distributed" nature of neural computation (rather than "localist"), that allow robustness and graceful degradation.
- **Intelligent behaviour is due to an "emergent" property of a large number of simple units rather than from explicitly encoded symbolic rules and algorithms**.
- Problem is, as we will see, that this emergent behaviour cannot be explained (black box), contrary to Dtrees and regression trees.

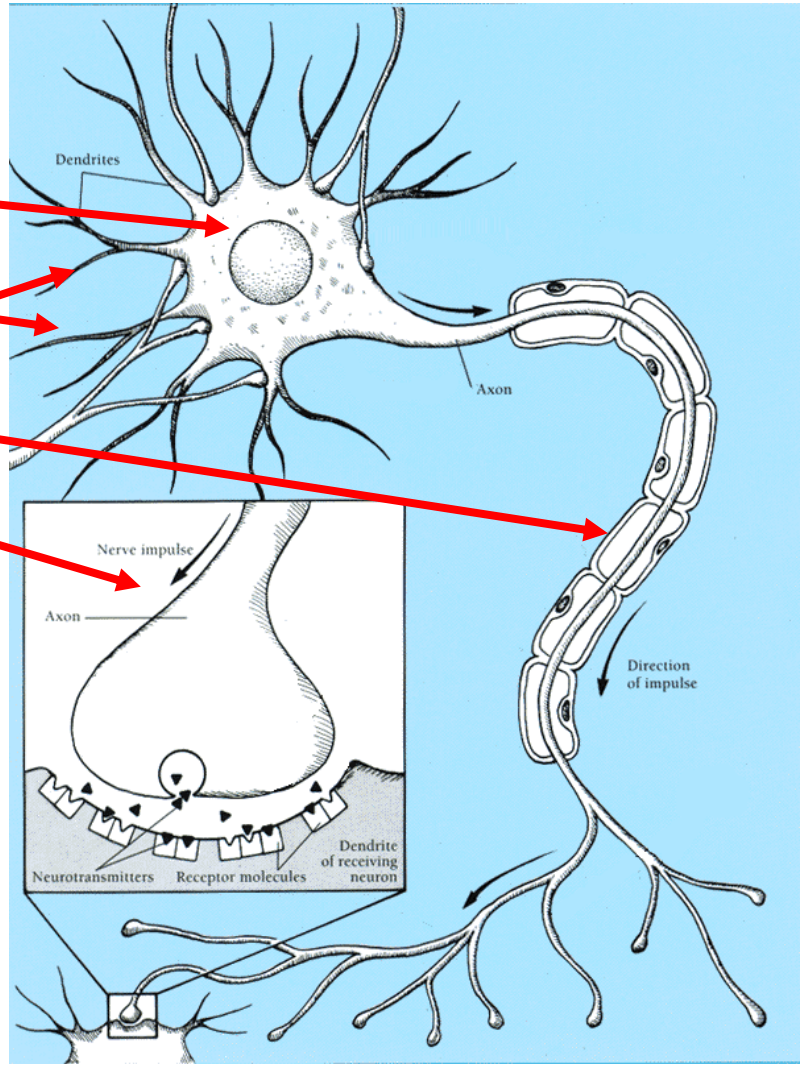
Neural Network Learning

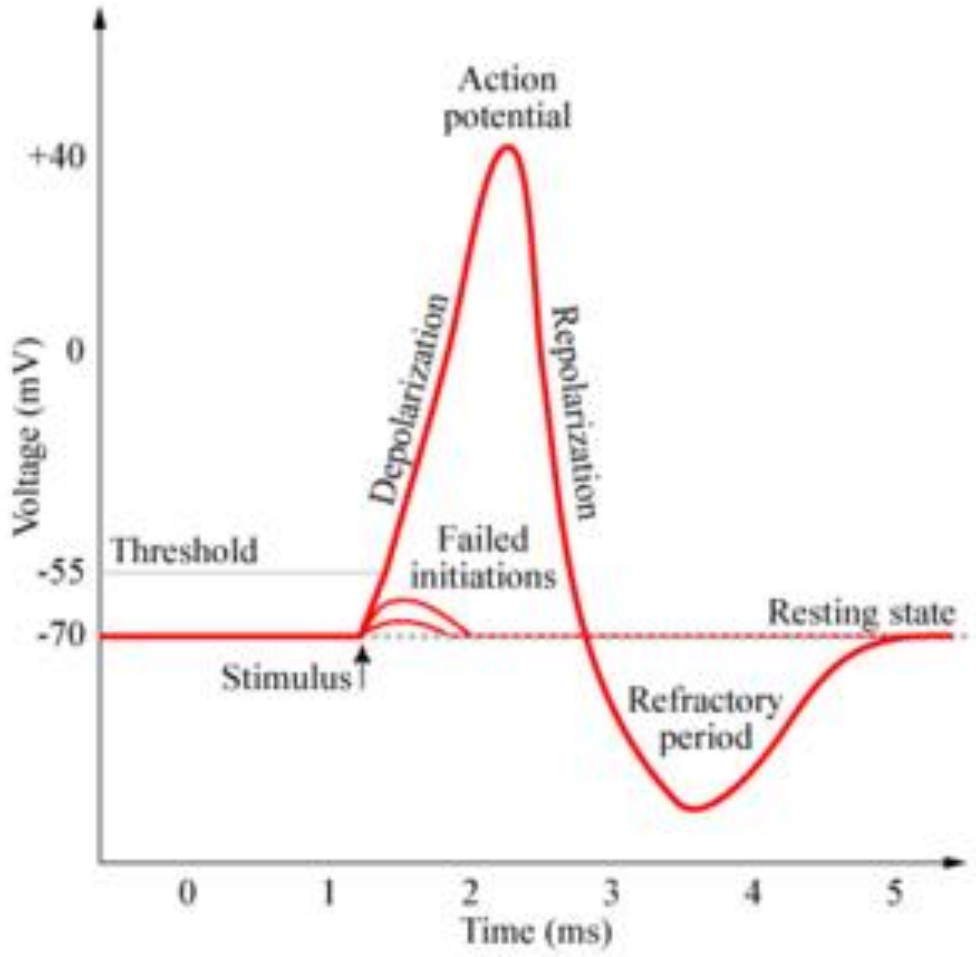
- Learning approach of NN algorithm based on modeling **adaptation** in biological neural systems.
- History of algorithms:
  - ➢**Perceptron**: Initial algorithm for learning simple neural networks (single layer) developed in the 1950's.
  - ➢**Backpropagation: a** more complex algorithm for learning multi-layer neural networks developed in the 1980's.
  - ➢**Convolutional Neural Networks, Recurrent Neural Networks** (since past 10 years more or less), still mostly based on «old» backpropagation principle + other mechanisms to meet the challenge of large, complex data such as images and text

Real Neurons

Cell structures:

- ➢Cell body
- ➢Dendrites
- ➢Axon
- ➢Synaptic terminals

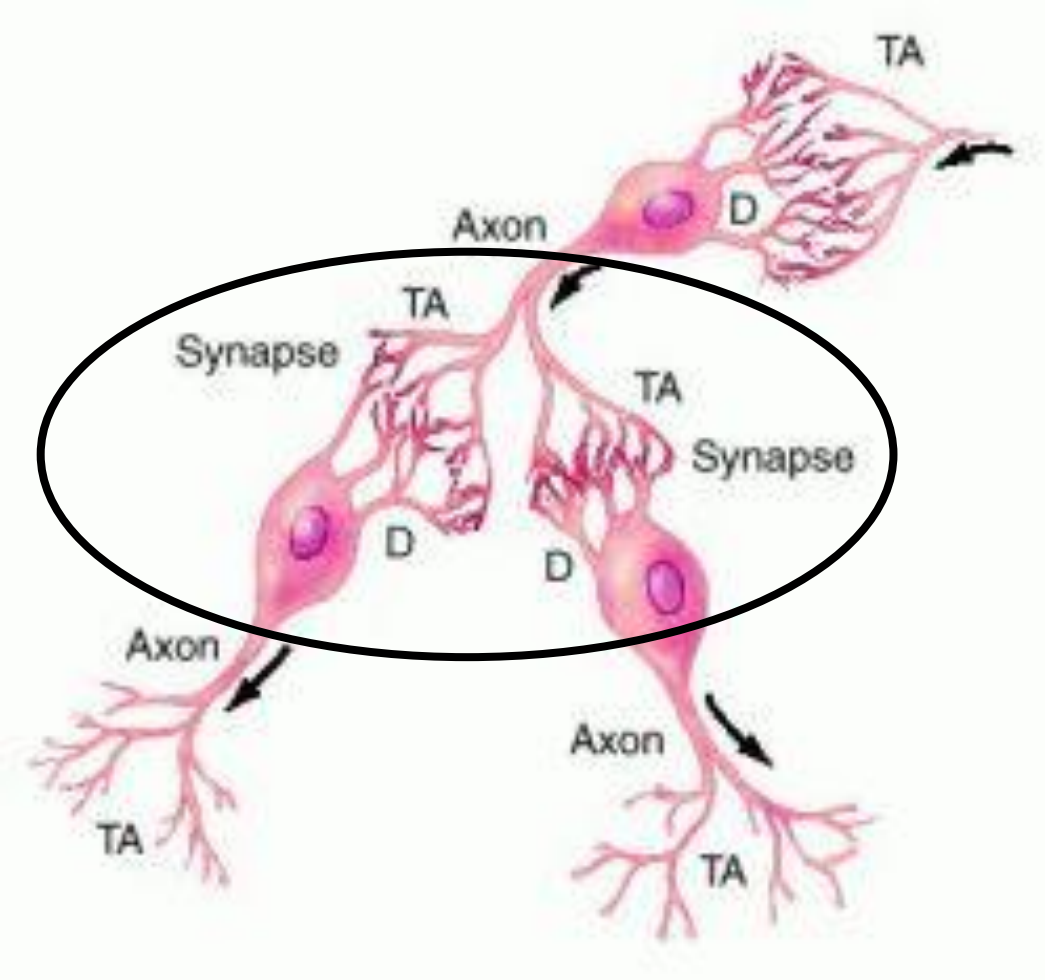




Neural Communication

- The electrical potential across cell membrane exhibits spikes called **action potentials**.
- Spike originates in the cell body, travels down axon, and **causes synaptic terminals to release neurotransmitters.**
  - Chemical diffuses across the synapse **to dendrites of other neurons (synaptic terminals, dendrites).**
- Neurotransmitters can be **excitatory** or **inhibitory.**
- If net input of neurotransmitters to a neuron from other neurons is **excitatory and exceeds some threshold, it fires an action potential.**

Neural connections



Real Neural Learning

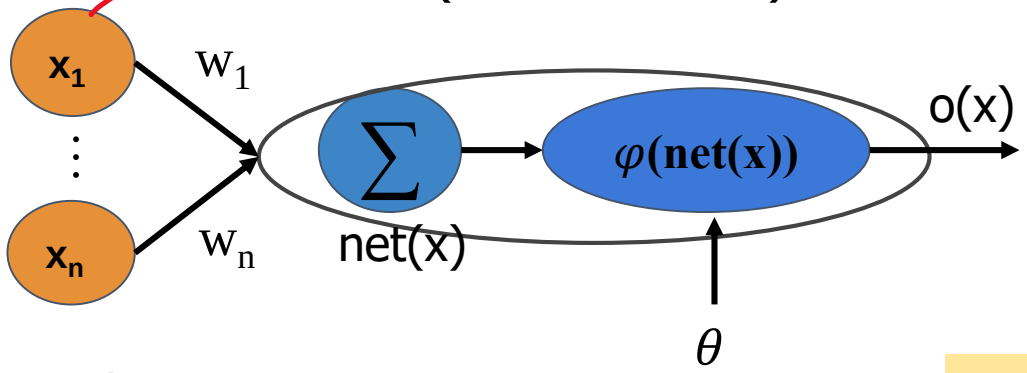
- Synapses **change size and strength** with experience (evolving structure).
- **Hebbian learning**: When two connected neurons are firing at the same time, the strength of the synapse between them increases.
- “Neurons that fire together, wire together.”

The computational model of a neuron (perceptron):

- **The network model of a single neuron** is a graph with cells as nodes, and synaptic connections as *weighted edges* from node x_i to neuron node n
- First, the perceptron computes a linear combination (**convolution**) of the input (x):

xi are the features values of instances x

$$net(x) = net(x_1, x_2, \dots x_n) = \sum_i w_i x_i$$

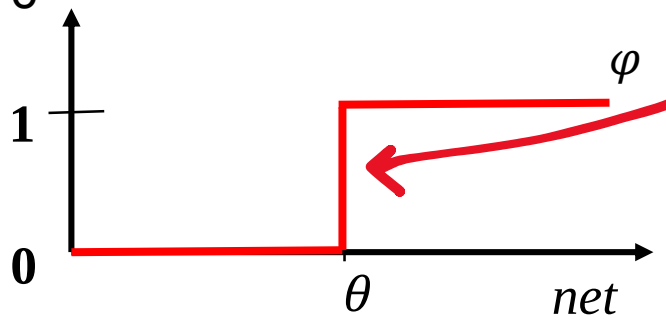


- Next, the output function is computed: o

$o=\varphi(net(x)) = 0$ if $net(x) < \theta$

$o=\varphi(net(x)) = 1$ if $net(x) \geq \theta$

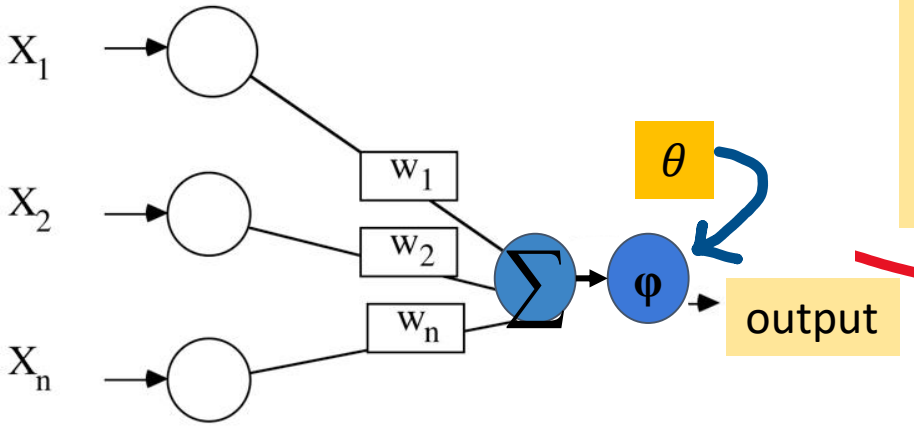
θ_i *is a constant called threshold or bias*



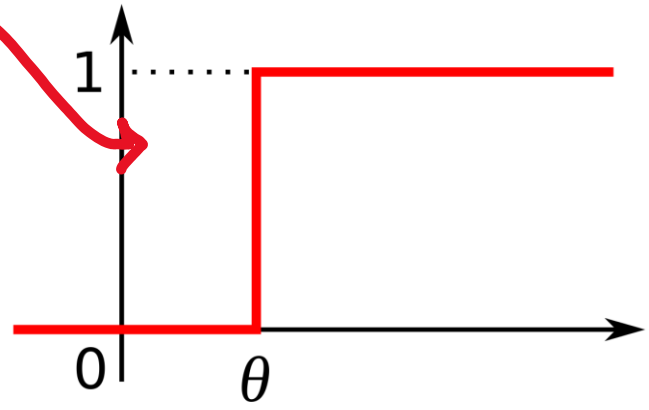
Note: $\varphi()$ is the step function, a **binary** function.

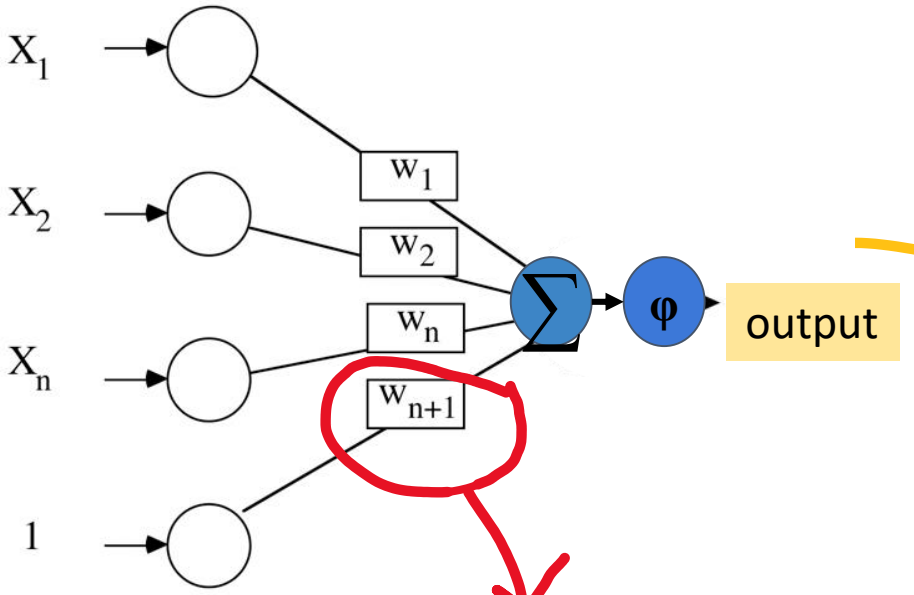
Perceptron training phase = estimating edge weights w_{ij} and threshold θ

Weight Versus Threshold

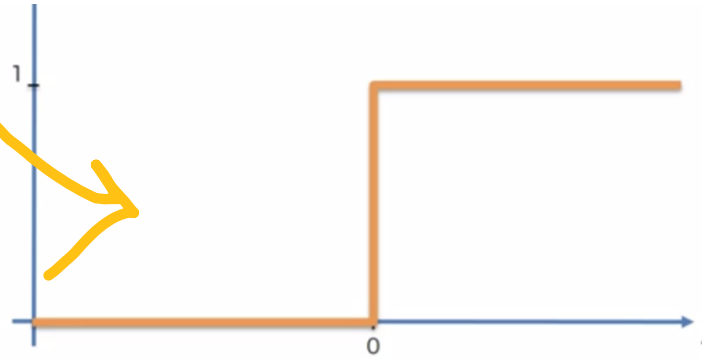


The threshold can be equivalently incorporated
in the convolution or in the step function





where $w_{n+1} = -\theta$



Perceptron learns a Linear Decision Boundary

- This is a **hyperplane** in an n-**dimensional space** (n is the number of features)**. What is learned are the** coefficients w_i *and* θ *(the parameters of the model):*

$$f(x) = \sum_i w_i x_i - \theta$$

- So, φ classifies the instance x **(a feature vector* $\langle x_1, x_2 .. x_n \rangle$)** as positive ($\varphi(x) = 1$) if:

$$\sum_i w_i x_i > \theta$$

➢ else it is classified as **negative**

Will interchageably use feature vector or instance to denote examples of objects in a given domain, represented as lists of feature values.

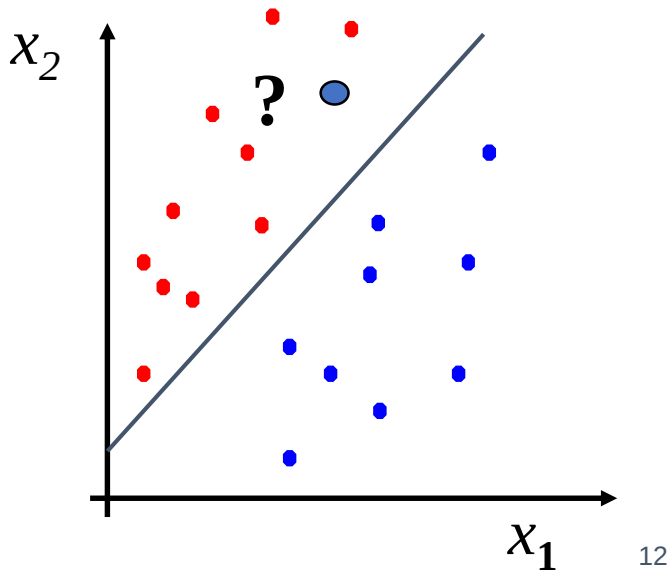
Note: f(x) is the learned MODEL, the classification function is $\varphi(x)$!!

Perceptron learns a Linear Decision Boundary

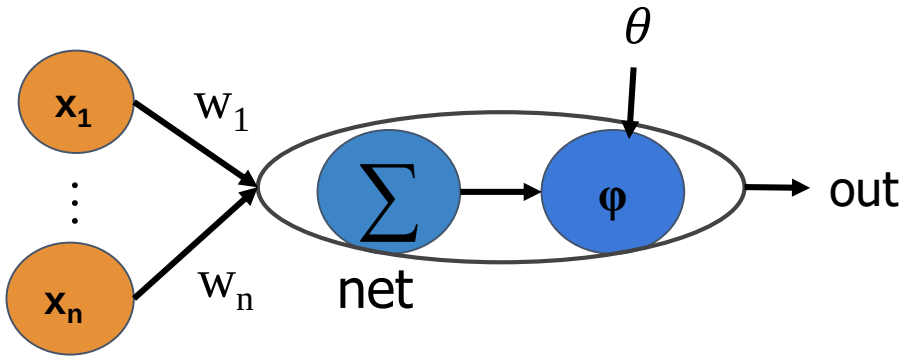
Example in a two-dimensional space

We can compute the coordinates of x in the n-dimensional space:

$w_1x_1 + w_2x_2 - \theta = 0$ → $x_2 = mx_1 + q$

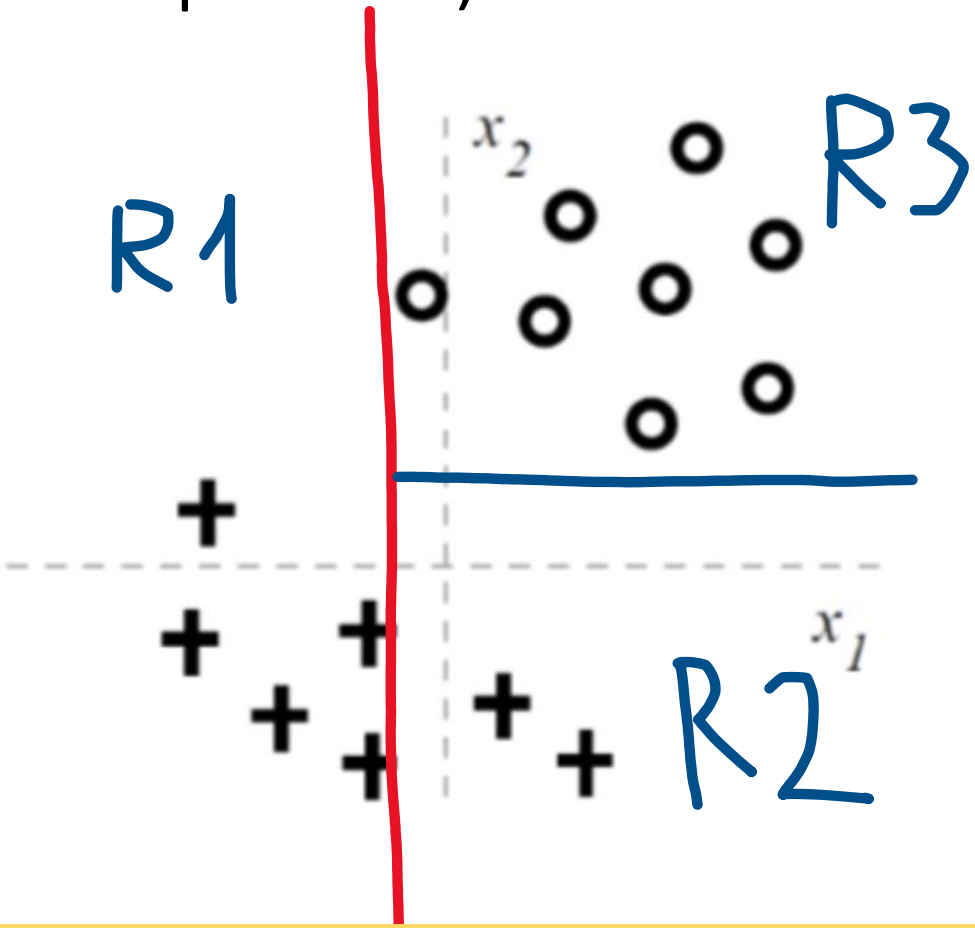


➢ **A line or a *hyperplane* in *n*-dimensional space**

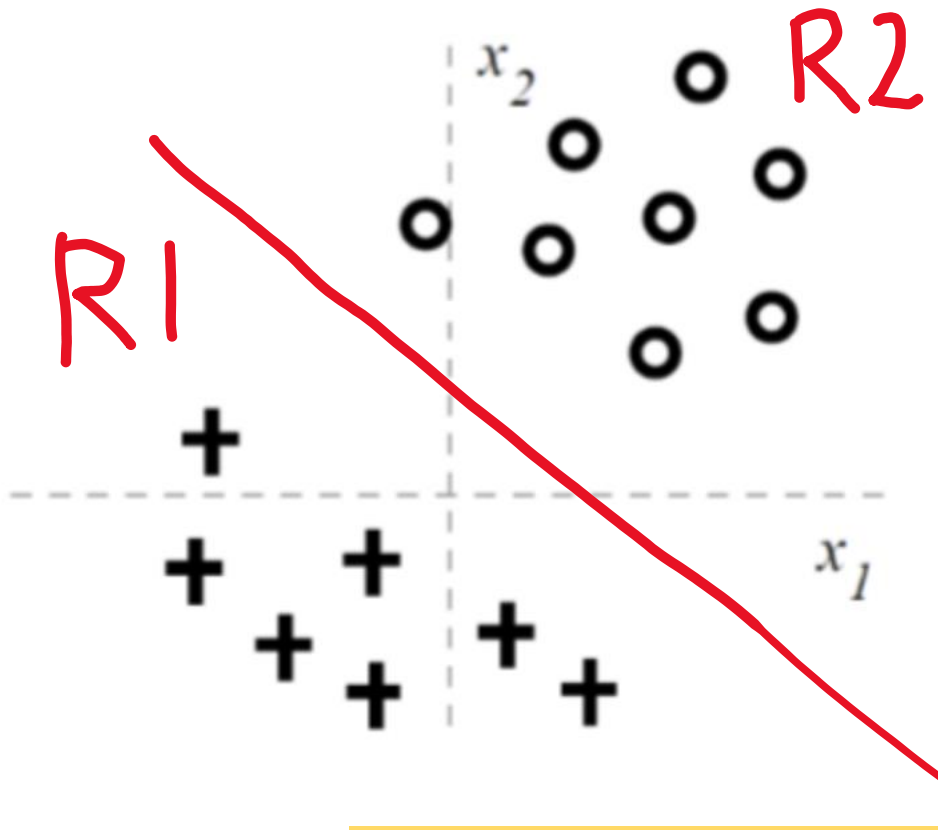


Note: the geometric representation of domain objects x in D as vectors, rather than records of a table, allows a more intuitive representation of the data and of the learned prediction function

Note the different decision regions learned by Dtrees (rectangual regions) and the Perceptron (semiplanes)

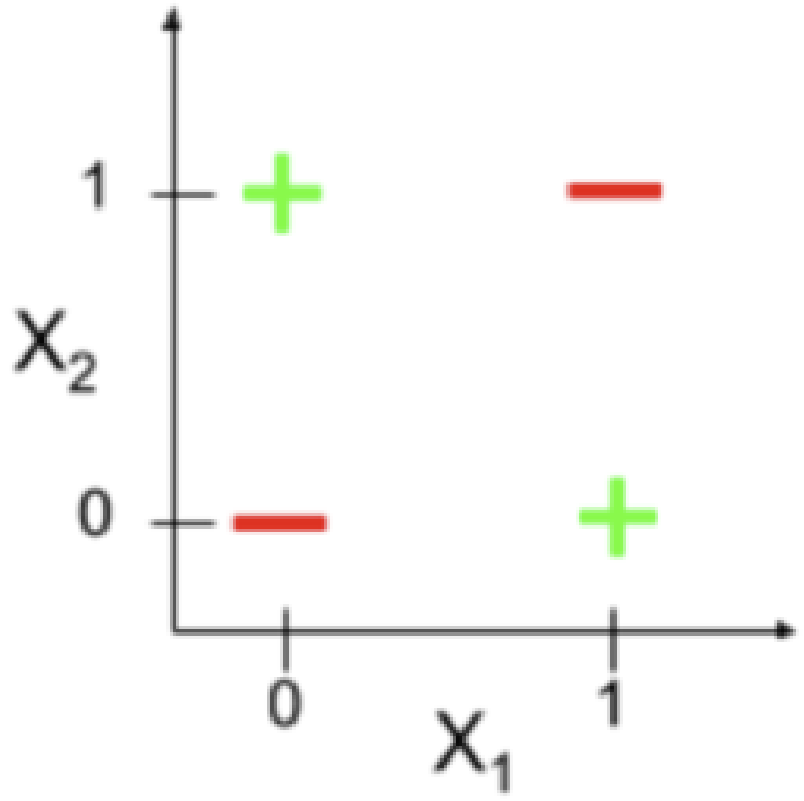


If x is in R1 or R2 then positive, else if x is in R3 the negative



If x is in R1 then positive, if in R2 then negative

What if your training data is as in this example?



Can we learn a percepton model?
Can we learn a decision tree?

Perceptron Training Algorithm

- Assume **supervised training** examples in the training dataset D: $<\mathbf{x}_j, y_j>$ giving the **desired output y=c(x)**, for a set of known instances $\mathbf{x}_j$
- Each instance is represented by a feature vector. Feature values are «fed» to the input nodes of the perceptron one at the time (rather than all together like in Dtrees)
- **Objective**: Learn the **synaptic weights (w_i)** «forcing» the model to produce the correct output o_j for each example x_j (**o_j is correct if equal to the provided label of xj in D, y_j**).
- Perceptron uses an **iterative updating algorithm** to learn a «correct» set of weights and thresholds.

Perceptron Learning Algorithm

Set the weights and the threshold **to random** values
Until all instances, x_j in D, are correctly classified
For each instance x_j, $<x_{j1}, x_{j2}, \ldots, x_{jn}>$, in D:
Compute the output: $o_j := \varphi(net_j - \theta_j)$
Update weights and the threshold by:

$$w_i := w_i + \Delta w_i = w_i + \eta\, Err_j\, x_{ji} = w_i + \eta\, (y_j - o_j)\, x_{ji}$$

$$\theta := \theta + \eta\, Err_j = \eta\, (y_j - o_j)$$

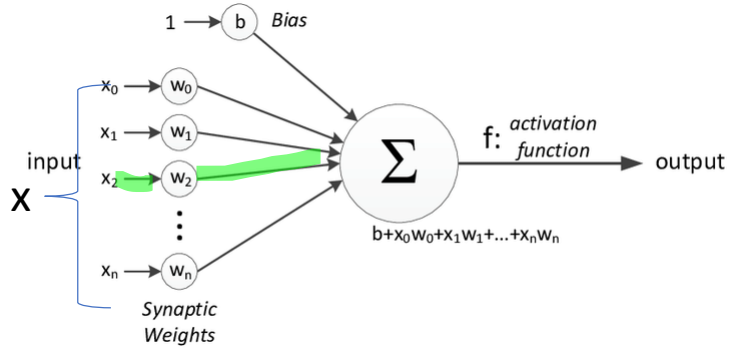
- Where η is a constant (hyperparameter) called the "learning rate"
- The «until» condition is a **convergence condition**

Note: weights on edges are updated proportionally to the observed error on the output (Err_j) **and** to the intensity of the signal x_{ij} traveling on the edges. η controls the proportion of this adjustment.

Perceptron iterative Learning Rules

$$w_i := w_i + \Delta w_i = w_i + \eta\, Err_j\, x_{ji} = w_i + \eta\,(y_j - o_j)$$

$$\theta_j := \theta_j + \eta\, Err_j = \theta_j + \eta\,(y_j - o_j)$$

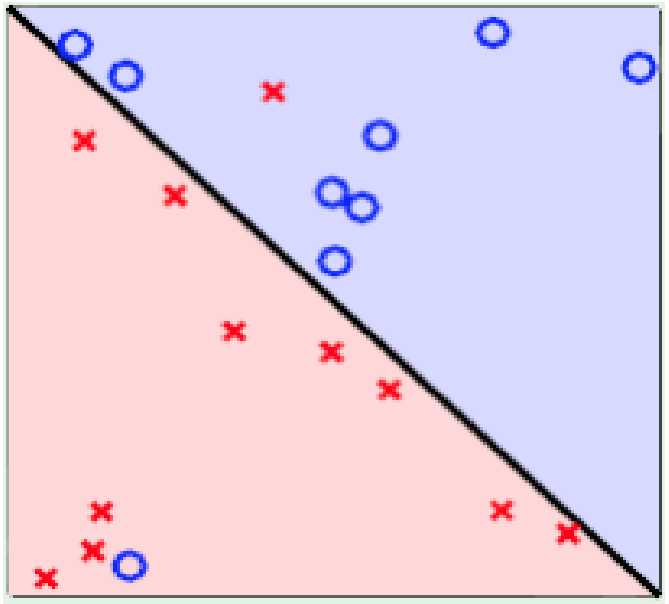


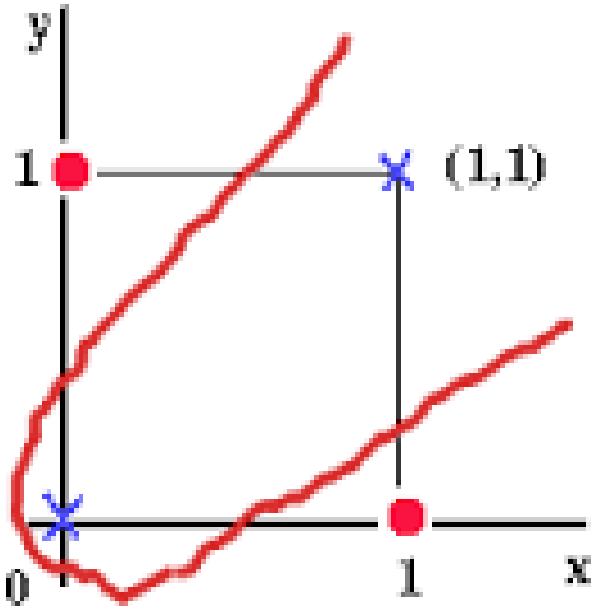
- It is equivalent to **the** rules:
  - ➢ If $\mathbf{o_j = y_j}$ (i.e. for $<x_j,y_j>$ in D the predicted output is **correct**) ⇒ $Err_j=0$: **no update**
  - ➢ If $\mathbf{o_j > y_j}$ (i.e. $o_j=1$, $y_j=0$ ⇒ $Err_j = -1$): output is higher than the correct one, so we **decrease the weights** on the active inputs of the quantity $\eta\, x_{ji}$ (x_{ji} is the current signal on synapsis i)
  - ➢ If $\mathbf{o_j < y_j}$ (i.e. $o_j=0$, $y_j=1$ ⇒ $Err_j=+1$): output is smaller than the correct one, so we **increases weights/threshold** on the active inputs of the quantity $\eta\, x_{ji}$

η controls the amount of increase/decrease

Perceptron cannot learn everything!

- Cannot learn **non-linearly separable** functions! f(x) is a (hyper-)line
- If our data (the learning set) are not separable by a line **(or by an hyper-plane, if many dimensions)**, then we need a more complex (polynomial?) decision boundary



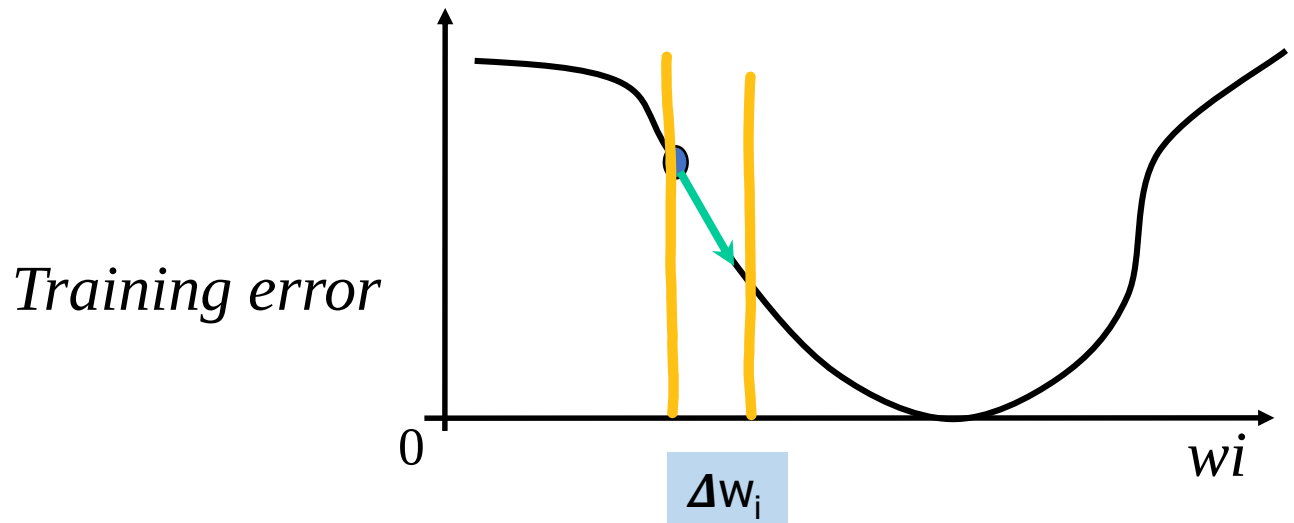


Perceptron leaning as Hill Climbing

- The hypothesis space being searched is a set of **weights and a threshold**. (the w_{ij} and θ)
- The objective is to **minimize the classification error** on the training set (an **optimization problem**, as for all ML algorithms).

Perceptron effectively does hill-climbing (**gradient descent**) in this space, changing the weights of a small amount at each step ($\Delta w_i = \eta\ Err_j\ x_{ji}$), to decrease the error observed on the training set (will learn more in future lessons on gradient descent for those who don't know)

- For a **single neuron**, the search space is well behaved with a **single minimum**



Perceptron Performance

- **In practice**, **results converge** only for linearly (or nearly lineary) separable data.
- Unfortunately, this is too simple model (like DT and RT) for many tasks, and sub-optimal (a better linear separator is SVM)
- With Multilayer perceptron networks MPN, convolutional neural networks and recurrent neural networks, things will get more complicated..
- Perceptrons are the “building blocks” of MPN and NN in general