INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Machine Learning – 4A
Perceptron

Team: A. Bit-Monnot, E. Chanthery, A. Dorise, M-J. Huguet, P. Leleux, M. Siala

Université Fédérale Toulouse Midi-Pyrénées

Course objectives

Courses : 6 CM

1. AI and machine learning (2CM – E. Chanthery) :
   - Data : preparation and quality,
   - Tasks : **supervised (regression/classification)** vs. unsupervised vs. reinforcement learning,
   - Machine learning workflow : train/test/validation,
   - **Evaluation :** bias vs. variance $\Longleftrightarrow$ underfitting vs. overfitting,
   - **Introduction to gradient descent**.
2. **Perceptron and (Artificial) neural networks (2CM – P. Leleux).**
3. Interpretable methods (2CM – M. Siala).

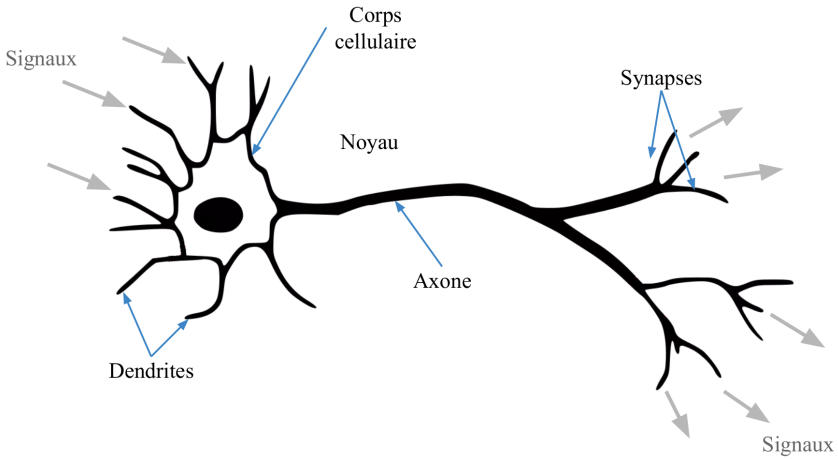
Labs (3 TP)

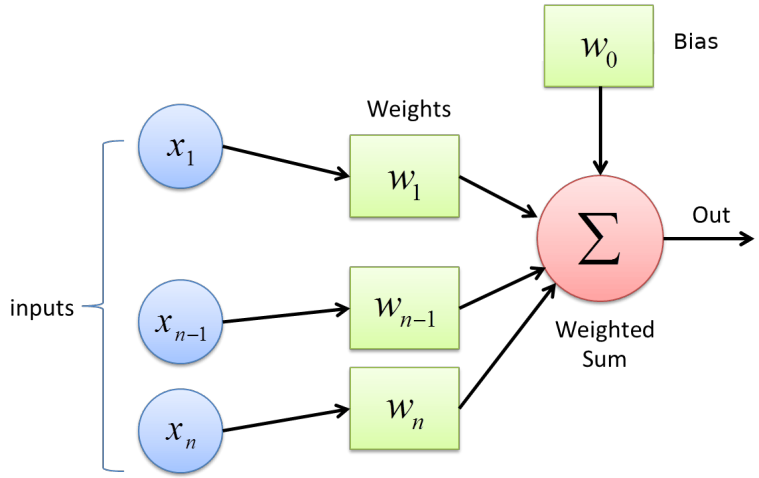
- Perceptron : linear regression & gradient descent (hand-rolled, python),
- Neural Networks (scikit-learn),
- Decision Trees (scikit-learn).

Section 1

The perceptron (regression)

From biological to artificial neurons
the perceptron





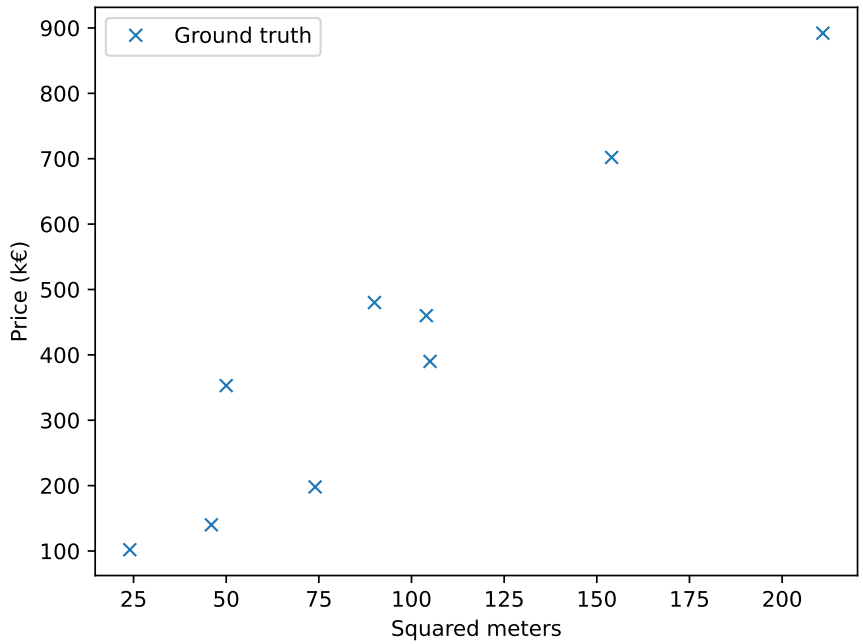
Given an **input** $x^T = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}$, we define a **perceptron** with the (synaptic) **weights** $w^T = \begin{bmatrix} w_1 & \cdots & w_n \end{bmatrix}$ and bias w_0 to compute the **output** $h_w(x)$ as

$$h_w(x) = w_0 + w_1 \times x_1 + w_2 \times x_2 + \cdots + w_n \times x_n \qquad (1)$$

The set of functions representable by a perceptron is the set of linear functions. That's our hypothesis space $\mathcal{H}_{perc}$.

Simple dataset
apartments prices

X	Y
m^2	Price (€)
24	102 000
46	140 000
50	353 600
211	892 000
74	198 000



Classical dataset :

https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques
https://inria.github.io/scikit-learn-mooc/python_scripts/datasets_california_housing.html

Perceptron for regression
Prediction of prices

We have a single feature (x_1 : squared meters), thus a function representable by a perceptron would have the form :

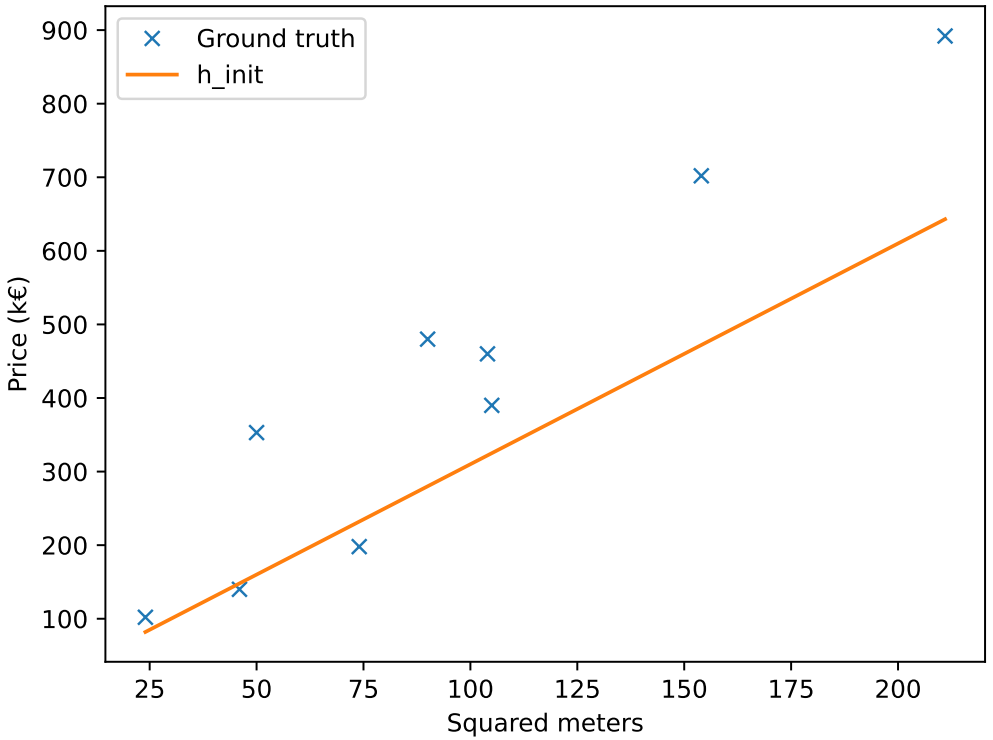
$$h_w(sqm) = w_0 + w_1 \times x_1$$

where we can interpret :

- w_0 : base price
- w_1 : price per squared meters

We can make an educated initial guess :

$$w_0 = 10\,000 \quad (€)$$
$$w_1 = 3\,000 \quad (€/m^2)$$



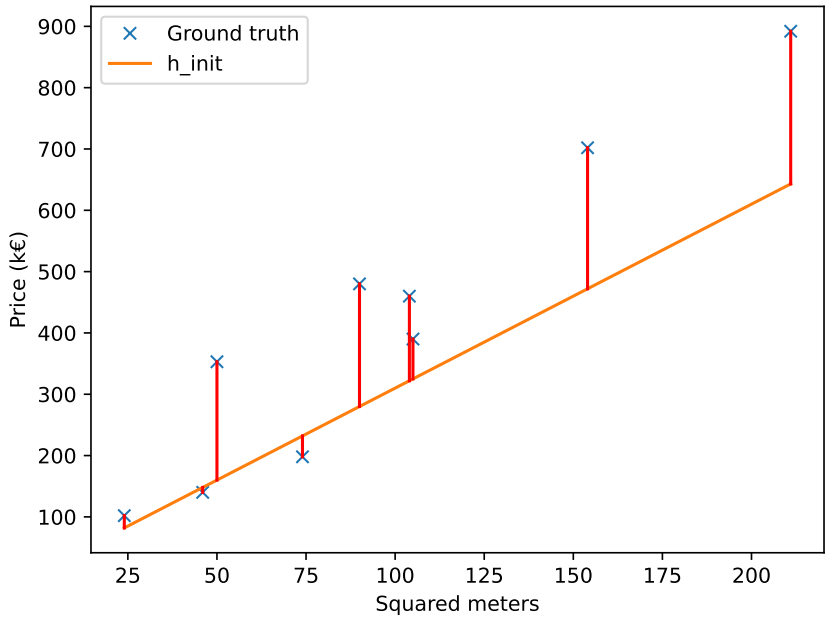
Perceptron for regression
Prediction error

For an example (x, y) and predictor h_w :

- prediction error $= |y - h_w(x)|$ (length of red segments)
- L_2-loss : $L_2(y, \hat{y}) = (y - h_w(x))^2$ (squared error)

This leads to the empirical loss for L_2 (Mean Squared Error (MSE)) over the entire dataset E :

$$EmpLoss_{L_2,E}(h_w) = \sum_{(x,y)\in E} \frac{(y - h_w(x))^2}{|E|}$$



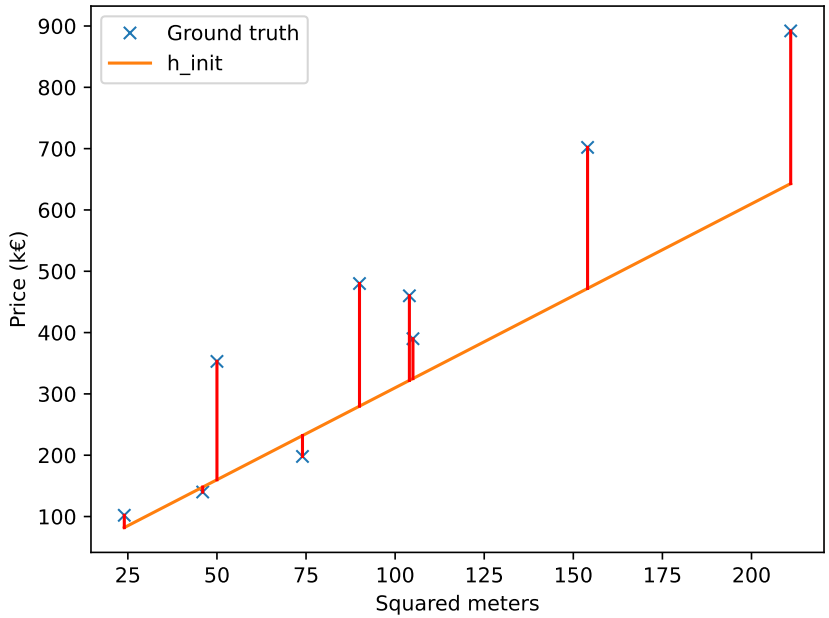
Perceptron for regression
Prediction error

For an example (x, y) and predictor h_w :

- prediction error $= |y - h_w(x)|$ (length of red segments)
- L_2-loss : $L_2(y, \hat{y}) = \frac{1}{2}(y - h_w(x))^2$ (squared error)

This leads to the empirical loss for L_2 (~~Mean~~ Squared Error (MSE)) over the entire dataset E to simplify computations :

$$EmpLoss_{L_2,E}(h_w) = \frac{1}{2} \sum_{(x,y)\in E} (y - h_w(x))^2$$



Perceptron for regression
Training : finding the best hypothesis

I want the hypothesis $\hat{h}^*$, with the minimum empirical loss :

$$\hat{h}^* = \underset{h_w \in \mathcal{H}_{perc}}{\arg\min}\, EmpLoss_{L_2,E}(h_w)$$

For the perceptron, this means finding the best weights $\hat{w}^*$ in the weight space.

$$\hat{w}^* = \underset{w}{\arg\min}\, EmpLoss_{L_2,E}(h_w)$$

Posing $Loss(w) = EmpLoss_{L_2,E}(h_w)$, we obtain :

$$\hat{w}^* = \underset{w}{\arg\min}\, Loss(w)$$

Gradient
Direction of steepest ascent

In any point w of the function, the gradient defines the direction of steepest ascent :

$$\vec{\nabla} g(w)$$

It can be computed from the partial derivatives :

$$\vec{\nabla} g(w) = \begin{bmatrix} \frac{\partial}{\partial w_0} g(w) \\ \frac{\partial}{\partial w_1} g(w) \\ \vdots \\ \frac{\partial}{\partial w_m} g(w) \end{bmatrix}$$

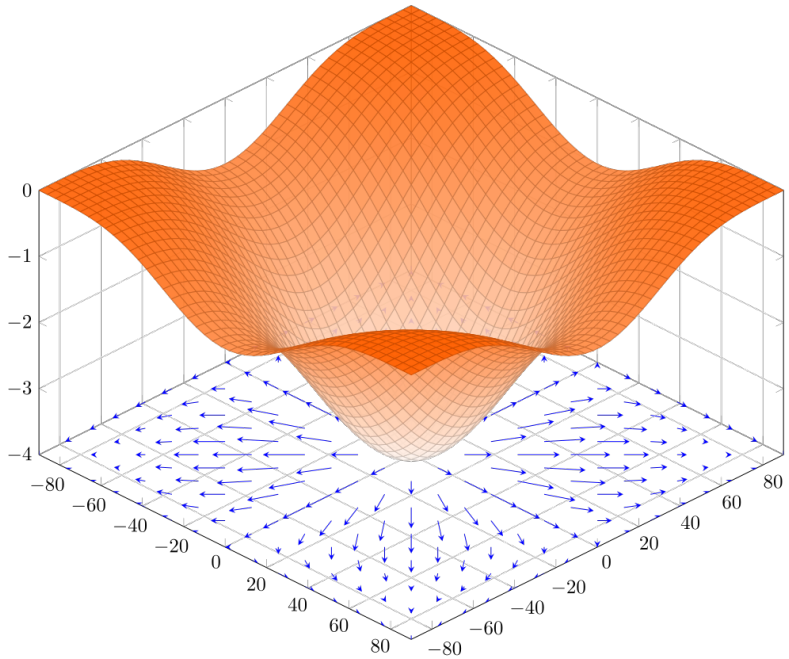


Figure – Gradient of $f(x, y) = -(cos^2(x) + cos^2(y))^2$

Gradient descent

From a point w, compute a new candidate w' by following the direction of steepest descent (opposite of the gradient).

$$w' = w - \alpha \times \vec{\nabla} g(w)$$

The distance traveled is parameterized by the **step size** $\alpha \in (0, 1]$.

Since the function is decreasing in this direction, there is a good chance that

$$g(w') < g(w)$$

Gradient descent
Algorithm

Algorithm Gradient descent algorithm

Dataset $E = \{(x, y)\ s.t.\ x \in X, y \in Y\}$ with : matrix of inputs $X = \left[x^{(1)}, \ldots, x^{(p)}\right]$, vector of truth $y^T = [y_1, \ldots, y_p]$ [1]

Initialize weights w_i

while not converged **do**

Compute prediction $h_w(x)$ and loss $Loss(w)$

Update weights with step size α :

$$w \leftarrow w - \alpha \times \vec{\nabla} Loss(w)$$

Typical convergence criteria ? no improvement of $Loss(w)$, no change in w for last k iterations (e.g. $k = 5$).

1. Recall that $Loss(w)$ is shortcut for $EmpLoss_{L_2,E}(h_w)$

Gradient descent
Computing the gradient for L_2-loss

Compute $\vec{\nabla} Loss(w)^T = \left[\frac{\partial}{\partial w_0} Loss(w), \frac{\partial}{\partial w_1} Loss(w), \ldots, \frac{\partial}{\partial w_m} Loss(w)\right]$?

Partial derivative of the L_2 loss for a single example (x, y) [2] :

$$\frac{\partial}{\partial w_i} Loss_{(x,y)}(w) = \frac{\partial}{\partial w_i} \frac{1}{2}(y - h_w(x))^2$$
$$= (y - h_w(x)) \times \frac{\partial}{\partial w_i}(y - h_w(x))$$

Applied to our system with a single feature (x_1) we obtain :

$$\frac{\partial}{\partial w_0} Loss_{(x,y)}(w) = -(y - h_w(x))$$
$$\frac{\partial}{\partial w_1} Loss_{(x,y)}(w) = -(y - h_w(x)) \times x_1$$

2. Here, $Loss_{(x,y)}(w)$ is for a single example, i.e. $EmpLoss_{L_2,\{(x,y)\}}(h_w) = (y - h_w(x))^2$

Gradient descent
Update rules

Updating the weights based on a single example : [3]

$$w_0 \leftarrow w_0 + \alpha \times (y - h_w(x))$$
$$w_1 \leftarrow w_1 + \alpha \times (y - h_w(x)) \times x_1$$

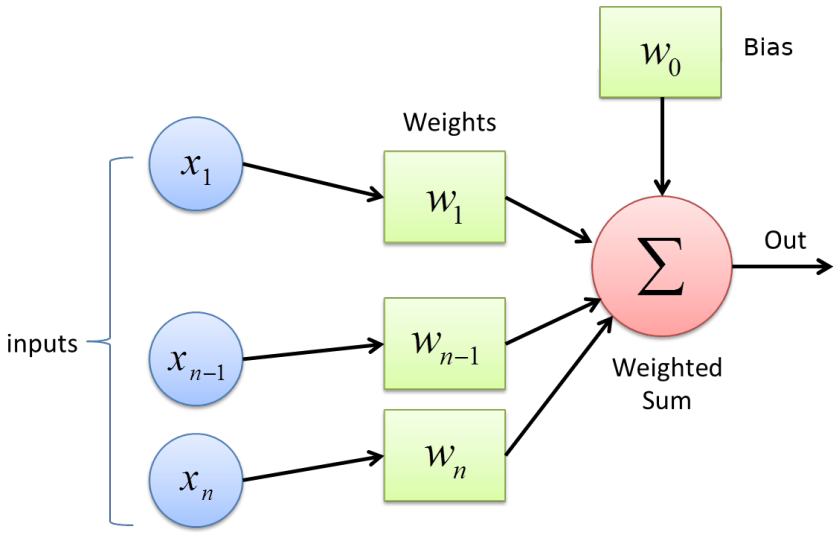
Updating the weights based on the entire training set E :

$$w_0 \leftarrow w_0 + \alpha \times \sum_{(x,y)\in E} (y - h_w(x))$$
$$w_1 \leftarrow w_1 + \alpha \times \sum_{(x,y)\in E} (y - h_w(x)) \times x_1$$

3. Note that the -2 factor from the previous equation is included in α term.

Representation trick

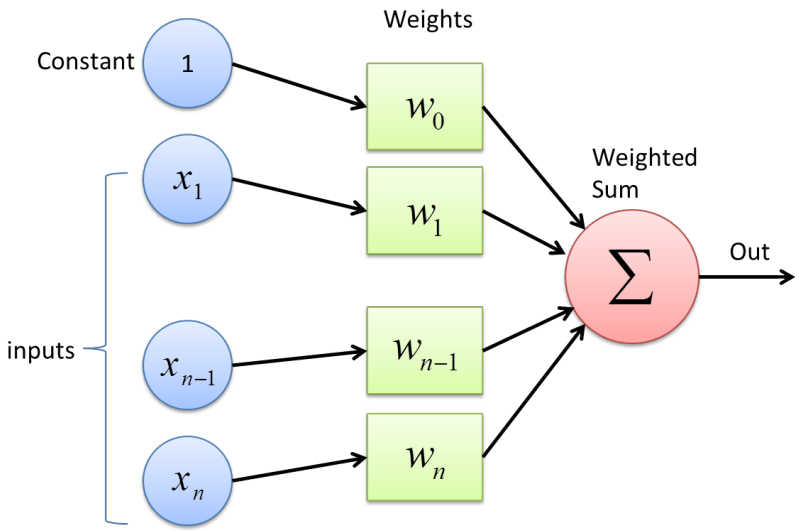
In the previous slides, we had to deal with the w_0 weight separately because it has no corresponding feature.



Representation trick

In the previous slides, we had to deal with the w_0 weight separately because it has no corresponding feature.

Instead, we can define an artificial feature x_0 that always has the value 1.



Representation trick

In the previous slides, we had to deal with the w_0 weight separately because it has no corresponding feature.

Instead, we can define an artificial feature x_0 that always has the value 1. Define $x^T = [x_0, \cdots, x_n]$, and reformulate directly h_w :

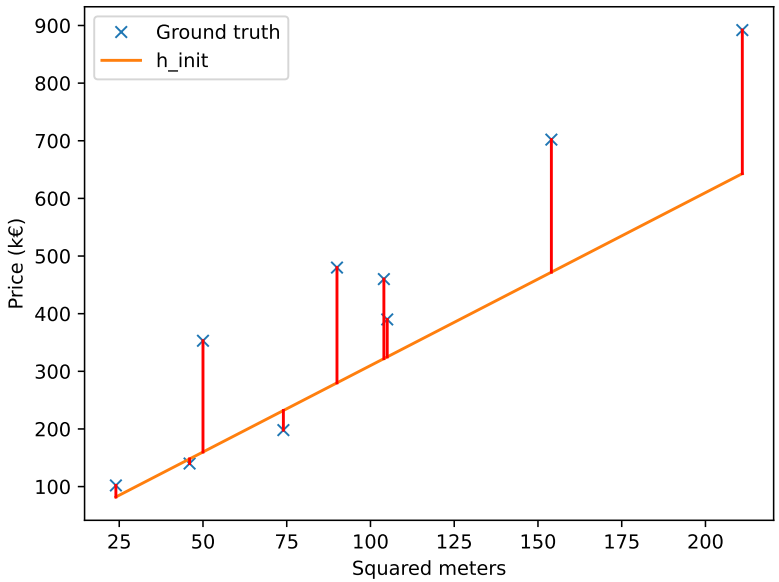
$$h_w(x) = \sum_{i=0}^{n} w_i \times x_i = w \cdot x \quad \text{(scalar/dot product)}$$

$$\frac{\partial}{\partial w_i} Loss_{(x,y)}(w) = -(y - h_w(x)) \times x_i$$

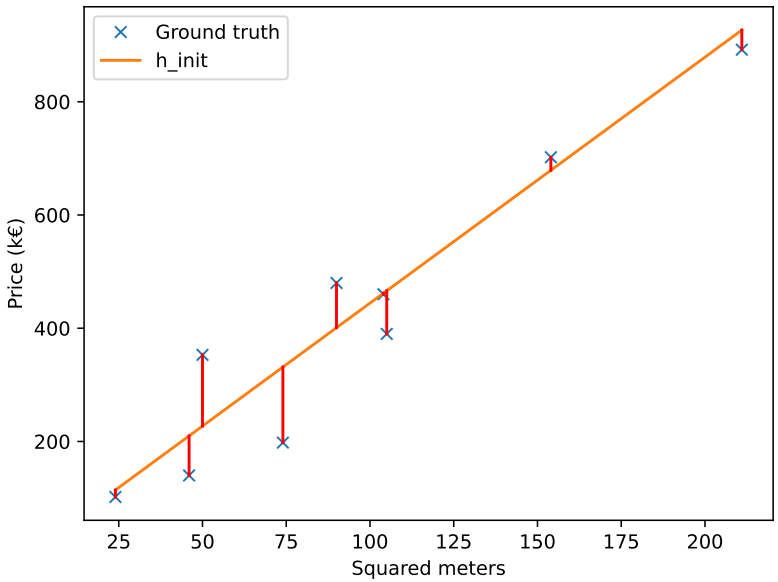
and the update rule (for L_2 loss) :

$$w_i \leftarrow w_i + \alpha \times \sum_{(x,y)\in E} (y - h_w(x)) \times x_i$$

Gradient descent
Updating our initial guess [4]



$w = [10\,000, 3\,000]$



$w' = [10\,010.53, 4\,343.82]$

4. With $alpha = 10^{-5}$

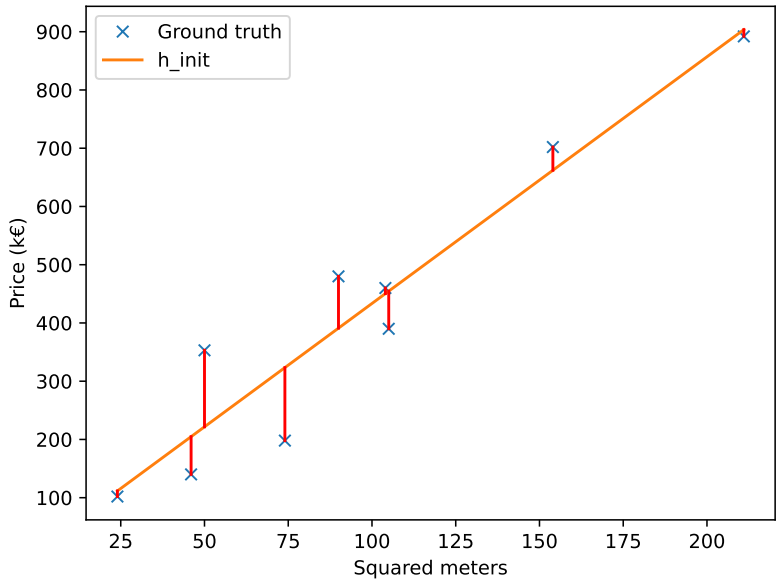
Gradient descent
Result

Repeating the gradient descent step, we eventually converge to our best solution.

$$\hat{w}^* = [9\,947.29, 4\,235.02]$$

I should sell my apartment for :

$$9\,947.29 + 4\,235.02 \times 165 = 708\,725€$$



Gradient descent
algorithm for perceptron with L_2 loss

Algorithm Gradient descent algorithm

Dataset $E = \{(x, y)\ s.t.\ x \in X, y \in Y\}$ with : matrix of inputs $X = \left[x^{(1)}, \ldots, x^{(p)}\right]$, vector of truth $y^T = [y_1, \ldots, y_p]$

Initialize weights w_i

while not converged **do**

Compute predictions : $h_w(x^{(j)}),\ j = 1, \ldots, p$

Compute errors : $\Delta_j = y_j - h_w(x^{(j)})$

Compute loss : $Loss(w) = \frac{1}{p} \sum_{j=1}^{p} \Delta_j^2$

Update weights with step size α :

for $i \in 1 \ldots n$ **do**

$$w_i \leftarrow w_i + \alpha \times \sum_{j=1}^{p} \Delta_j \times x_i^{(j)}$$

$\Longrightarrow$ Perceptron monocouche équivalent à la régression linéaire !

Gradient descent
algorithm for perceptron with L_2 loss (vector version)

Algorithm Gradient descent algorithm

Dataset $E = \{(x, y)\ s.t.\ x \in X, y \in Y\}$ with : matrix of inputs $X = \left[x^{(1)}, \ldots, x^{(p)}\right]$, vector of truth $y^T = [y_1, \ldots, y_p]$

Initialize weights w_i

while not converged **do**

Compute vector of predictions : $h_w(X)^T = \left[h_w(x^{(1)}), \ldots, h_w(x^{(p)})\right]$

Compute vector of errors : $\Delta = Y - h_w(X)$

Compute loss (convergence ?) : $Loss(w) = \frac{1}{p}\Delta^T\Delta = \frac{1}{p}\|\Delta\|_2^2$

Update weights with step size α : $w \leftarrow w + \alpha X^T\Delta$

$\Longrightarrow$ Perceptron monocouche équivalent à la régression linéaire !

Section 2

A perceptron for classification

A classification problem
Looking for a apartment

I now want to buy a new apartment to replace the one I just sold. To be reactive I built an automated system that sends me any new announce of an apartment for sale.

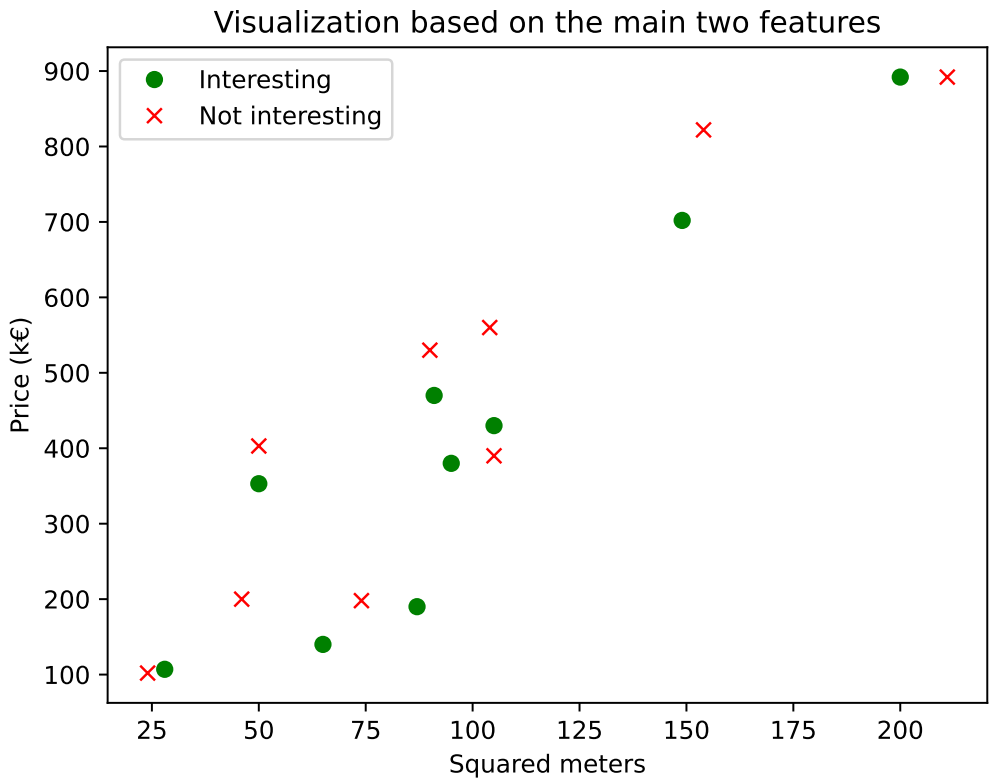
- Problem : there are dozens of announces every day and I don't have time to look at them all.
- Solution : build an AI system that will predict whether I will be interested in a particular apartment based on a few of its features. If it predicts that I am not interested, it will discard the announce.

A classification problem
Dataset

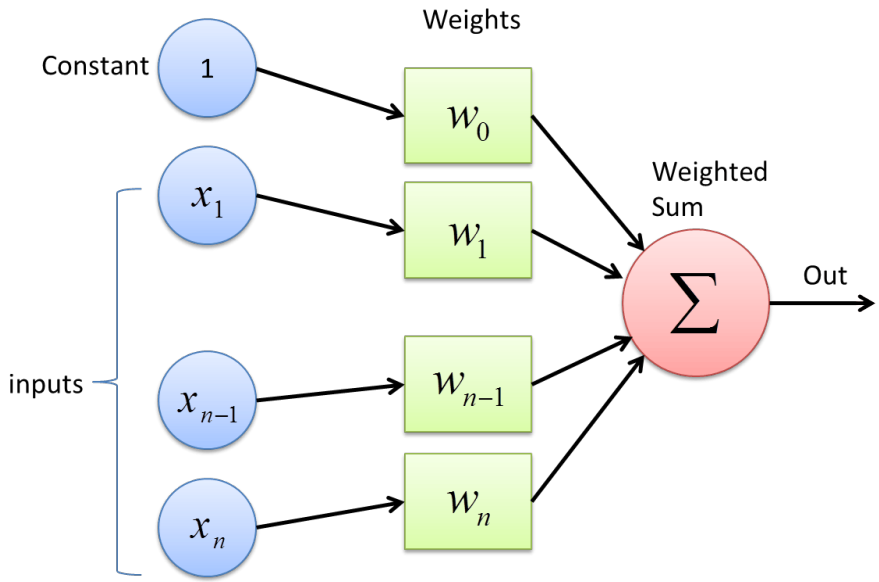
So far, I collected the following dataset with a label defining whether an announce that I previously saw was interesting.

X				Y
m^2	Num Rooms	Floor	Price (€)	Interesting
24	1	4	102 000	true
46	3	2	140 000	false
50	3	6	353 600	false
211	5	3	892 000	true
74	3	1	198 000	true

A classification problem
Dataset visualization

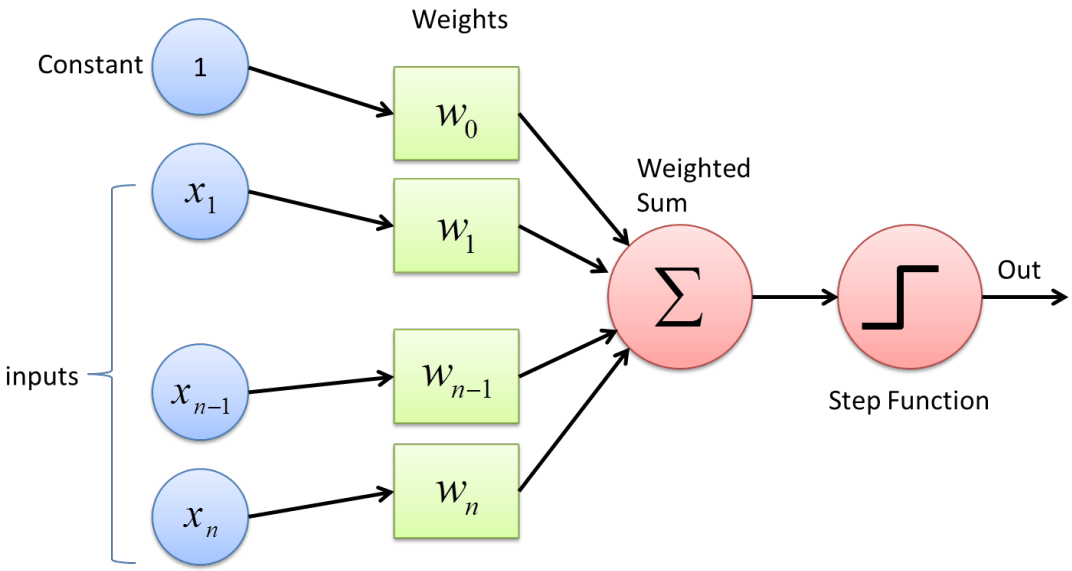


Our previous perceptron



$$h_w(x) = w_0 + w_1 \times x_1 + w_2 \times x_2 + \cdots + w_n \times x_n$$
$$= w \cdot x$$

Perceptron for classification

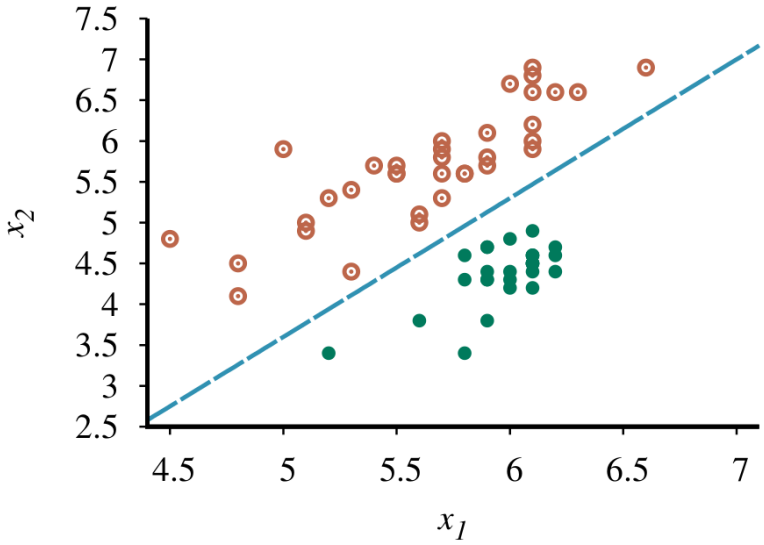


$$h_w(x) = Step(w \cdot x)$$

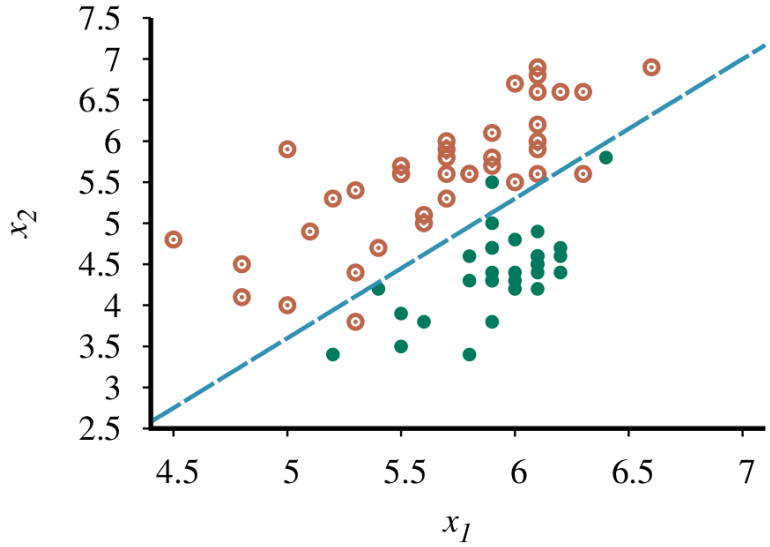
where $Step(z) = 1$ if $z \geq 0$ and 0 otherwise

Perceptron for classification
The perceptron as a linear classifier

The perceptron defines a **decision boundary** that separates two classes.



Linearly separable
Perfectly classifiable by a perceptron

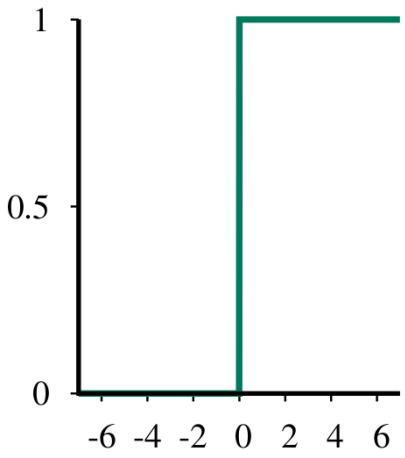


Not linearly separable
Not perfectly classifiable by a perceptron

Perceptron for classification
The step function

$$h_w(x) = Step(w \cdot x)$$

where $Step(z) = 1$ if $z \geq 0$ and 0 otherwise



$Step(z)$

We now have a function that we could train in order to output :

- 1 if the example is in the class (interesting)
- 0 otherwise (not interesting)

Problem : the function is :

- non-differentiable in 0
- the gradient is 0 everywhere else

Perceptron for classification
The perceptron learning rule

Nevertheless, an empirical rule was proposed : **the perceptron update rule** (here for a single example (x, y)) :

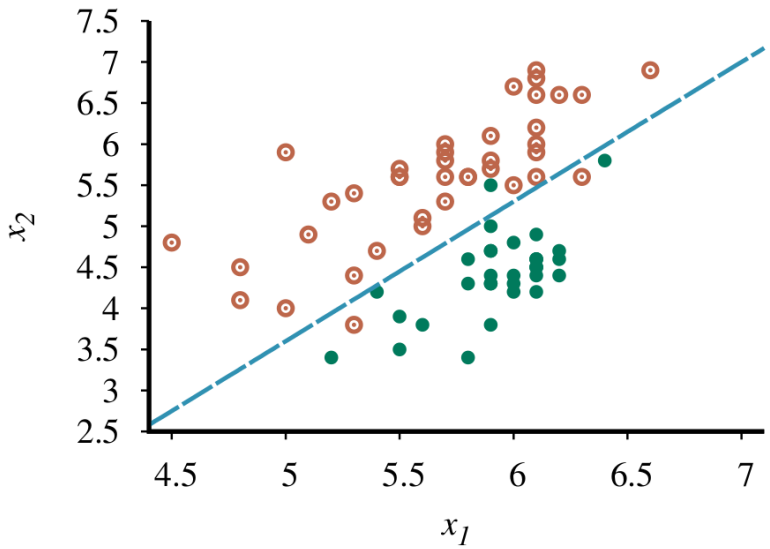
$$w_i \leftarrow w_i + \alpha \times (y - h_w(x)) \times x_i$$

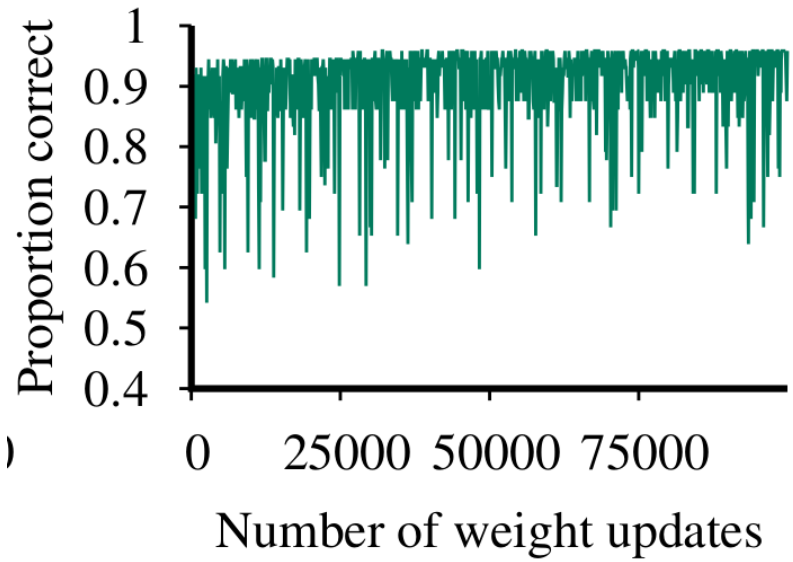
which is identical to the update rule for linear regression (for L_2).

The rule is shown to converge to a solution when the data is linearly separable.

Perceptron for classification
The perceptron learning rule (under non separable data)

However the perceptron learning rule is unstable when the data is not linearly separable :





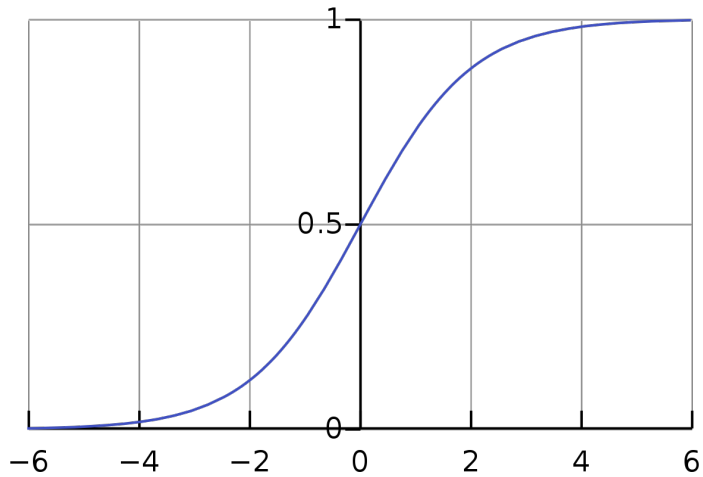
Replacing the step function

Turns out we can replace the step function with one with nicer properties : the logistic (or sigmoïd) function

$$Logistic(z) = \frac{1}{1 + e^{-z}}$$

and redefine our hypothesis function :

$$h_w(x) = Logistic(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}}$$



Often called the **logistic regression** [a].

a. **N.B. : logistic regression is a method for classification !**

Back on track : gradient descent

Logistic(x) is differentiable on $]-\infty, \infty[$. This allows us to reuse gradient descent for training :

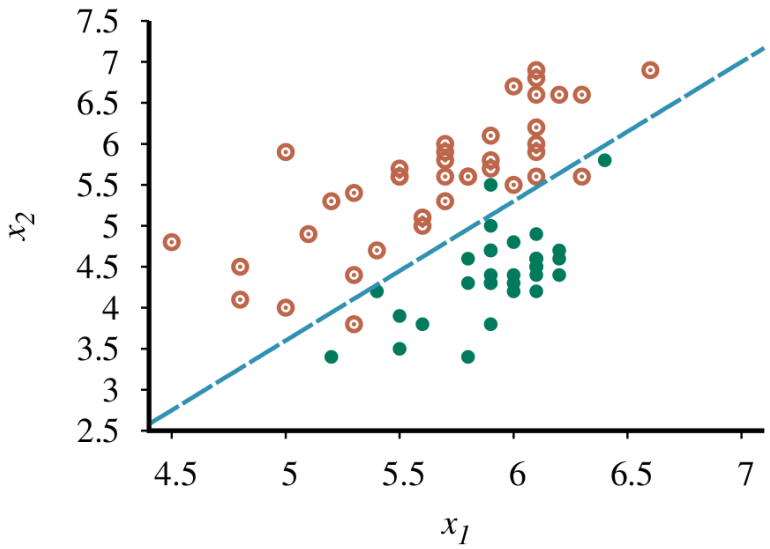
$$w \leftarrow w - \alpha \times \vec{\nabla} Loss(w)$$

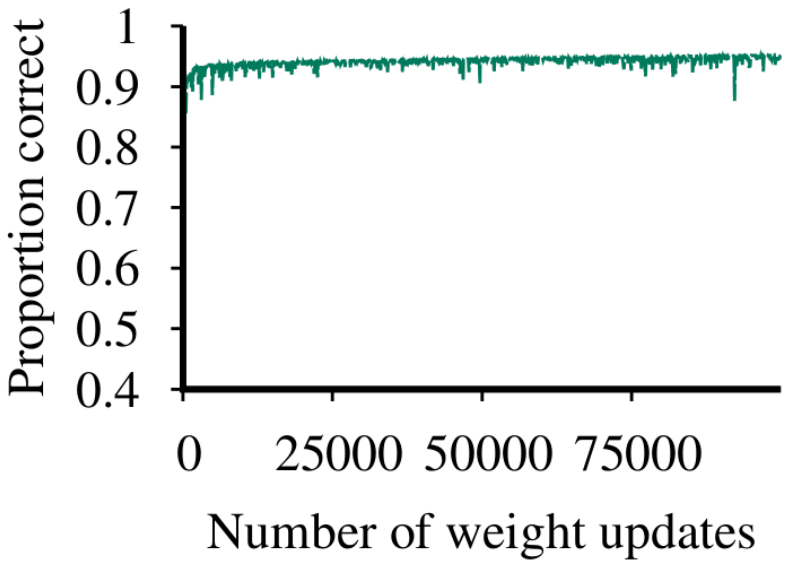
For an L_2 loss we obtain the update rule :

$$w_i \leftarrow w_i + \alpha(y - h_w(x)) \times h_w(x) \times (1 - h_w(x)) \times x_i$$

Perceptron for classification
Training the logistic regression (under non-separable data)

Compared to the step function, the logistic regression tends to converge more quickly and reliably in the presence of noisy and non-separable data.





Section 3

Synthesis

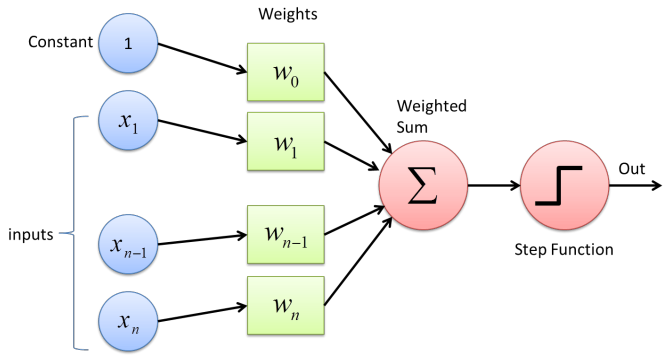
Synthesis

We saw two classes of perceptrons :

- linear regressor
- linear classifier

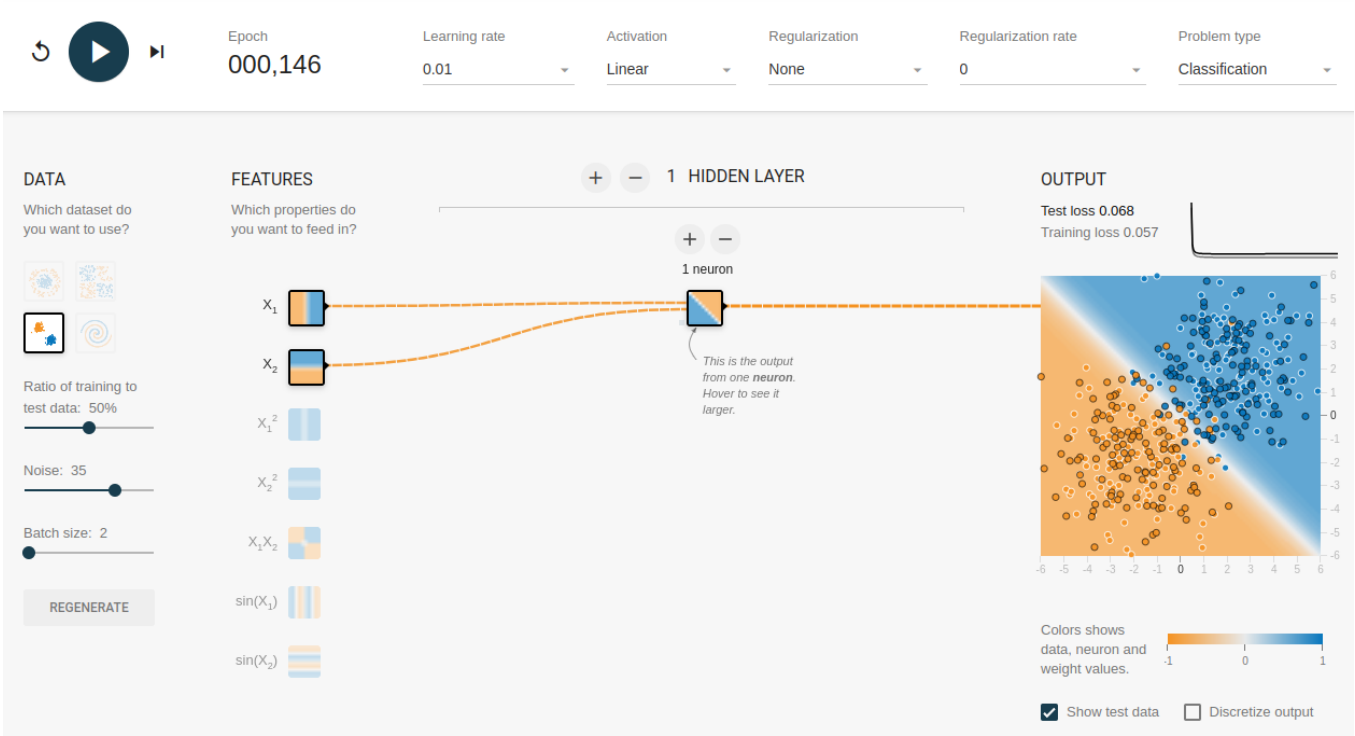
Both can be trained with **gradient descent** in attempt to **minimize the loss**.

In the next course, the perceptron will be a **neural unit** in a **neural network**.



Perceptron in practice
Playground Tensorflow

https://playground.tensorflow.org/



Exercises

- For each of the following datasets, propose (without calculus) a perceptron that would correctly classify it.

X	Y
-1	Flower
2	Cat
1	Flower
0	Flower

X	Y
-1	Flower
2	Cat
1	Cat
0	Cat

X	Y
-1	Flower
2	Flower
1	Cat
0	Cat

Exercises

1. You have N examples in your dataset, each with M features. Give an estimate of the computational cost of a single update step. Does it scale to large-scale datasets (e.g. $N = 10^5, M = 10^4$)
2. For the linear regression (regression perceptron), are we guaranteed to find the optimal weights with gradient descent ?
3. What's a reasonable loss for spam detection ? (you can make some assumptions about the proportion of spam)
4. What's the update formula of a regression perceptron using the L_1 loss ?