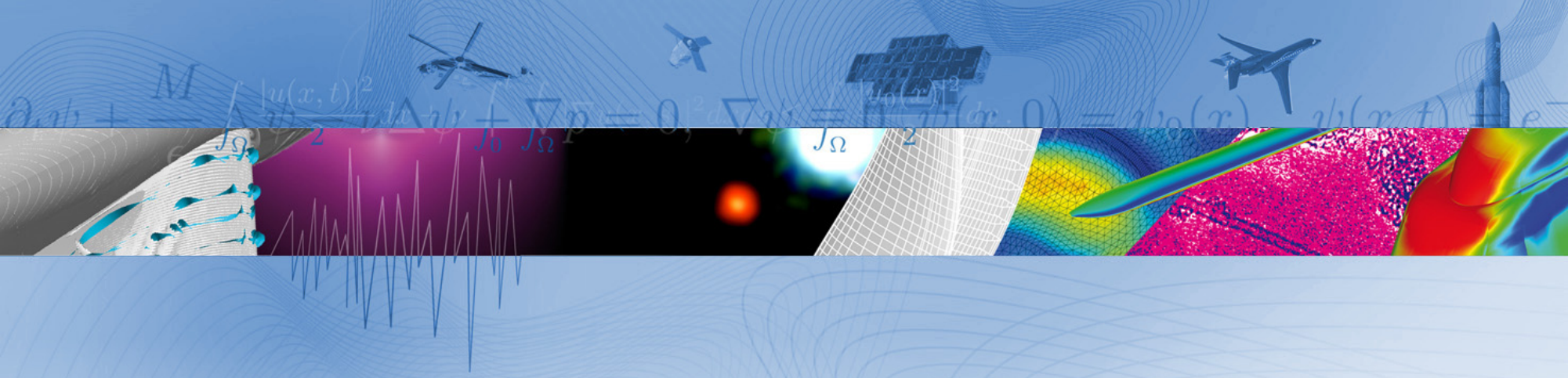


Neural Networks

Alexandre Boulch

ONERA
THE FRENCH AEROSPACE LAB

Table of contents

A first step toward Deep Learning

Objective of the session

- Understand neural networks (Multi Layer Perceptron)
- Code a neural network from scratch and optimize it (practical)
- Become familiar with PyTorch (practical session)

Historical Background

Table of contents

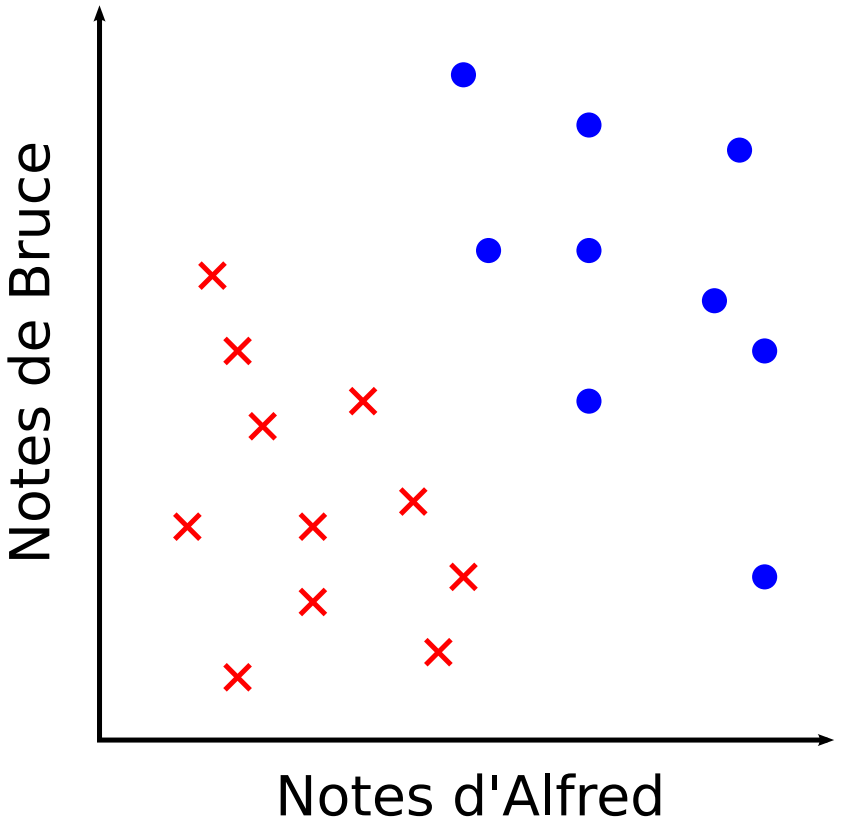
The recommandation problem

A simple example

- 3 persons : Alfred, Bruce and James
- Alfred and Bruce give marks to movies

Objective : James wants to predict if will like or not the movie.

Binary classification problem

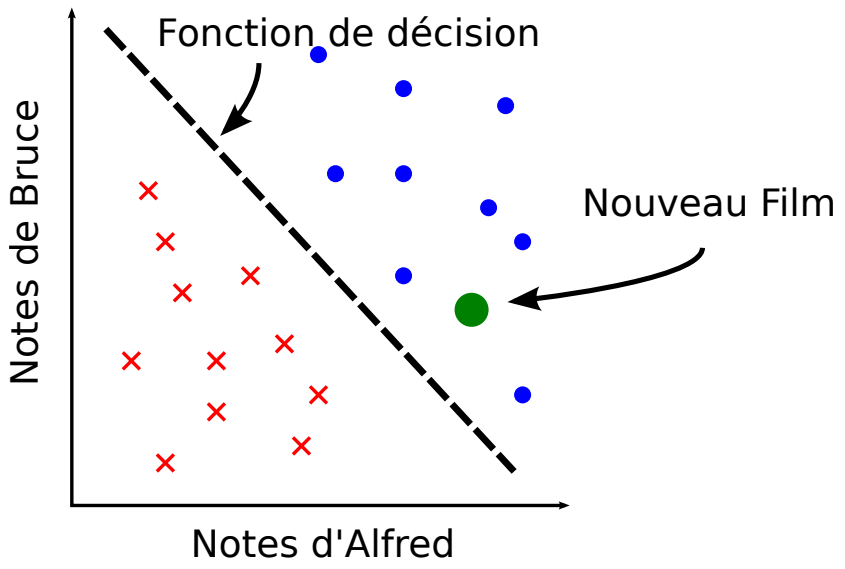


Linear decision

We suppose that there exists a linear separation between the two classes.

Objective

Find a function $\mathbb{R}^2 \longrightarrow \{0, 1\}$ that takes value 1 for good movies, 0 otherwise.



Mathematical expression

Let $\mathbf{x} = (x_1, x_2)$ be a mark vector.
The separation is an hyperplan of the affine space :

$$w_1 x_1 + w_2 x_2 + b = \mathbf{w}^T \mathbf{x} + b = 0$$

Let h be the function :

$$\begin{aligned} h : \mathbb{R}^2 &\longrightarrow \mathbb{R} \\ \mathbf{x} &\mapsto \mathbf{w}^T \mathbf{x} + b \end{aligned}$$

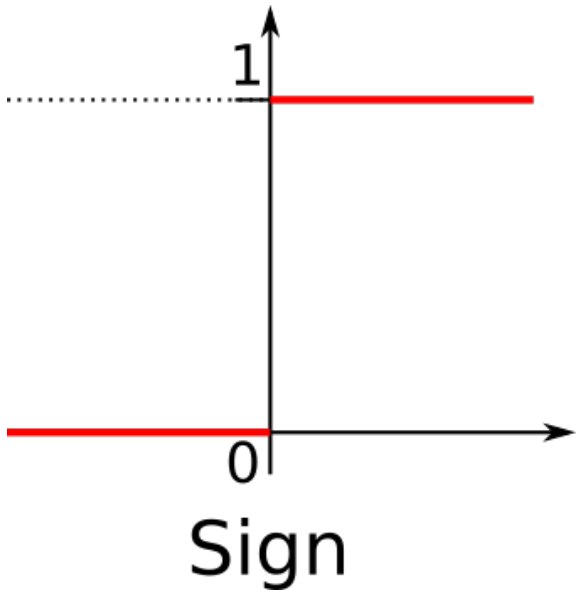
h value is 0 on the hyperplane, and his of different sign on both sides.

Mathematical expression

The sign function is what we looking for :

$$s(x) = \begin{cases} 1, & \text{if } x \geq 1 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

This function is piecewise constant, zero derivative. Makes the optimization difficult.

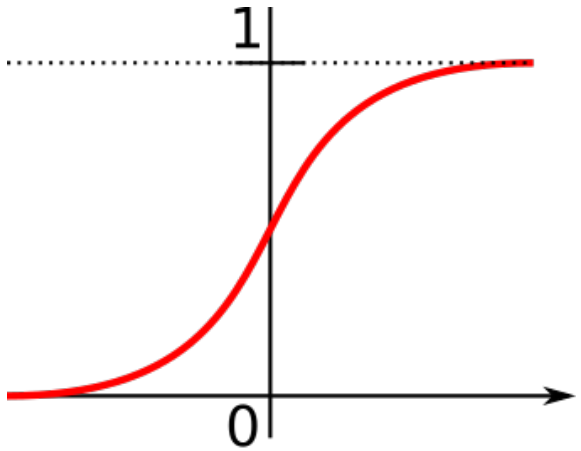


Sign

Mathematical expression

We use an approximation of the sign function, the sigmoid :

$$h : \mathbb{R}^2 \longrightarrow \{0, 1\}$$
$$\mathbf{x} \mapsto g(\mathbf{w}^T \mathbf{x} + \mathbf{b})$$
$$\text{where } g(z) = \frac{1}{1 + \exp(-z)}$$



Sigmoid

A neuron

Table of contents

Objective criterion

Objective

Find $\mathbf{w}$ and b such that

$$h(\mathbf{x}^{(i)}|\mathbf{w}, b) \approx y^{(i)}$$

for the data $(\mathbf{x}, y) \in \mathbb{R} \times \{0, 1\}$.

To do that we must use a criterion that :

- takes 0 value if $\mathbf{w}, b) = y^{(i)}$
- is differentiable
- increase along with the difference between $\mathbf{w}, b)$ and $y^{(i)}$

Criteria : square difference

Criterion

$$J_{\mathcal{M}}(\mathbf{X}, \mathbf{Y}|\mathbf{w}, b) = \sum_{i \in \mathcal{M}} (h(\mathbf{x}^{(i)}|\mathbf{w}, b) - y^{(i)})^2$$

The minimum of $J_{\mathcal{M}}$ is reached for $h(\mathbf{x}^{(i)}|\mathbf{w}, b) = y^{(i)}$:

$$\arg\min_{\mathbf{w},b} J_{\mathcal{M}}(\mathbf{X}, \mathbf{Y}|\mathbf{w}, b)$$

Gradient Descent

For every *smooth* functions f :

$$\forall x, \nabla f_x \neq 0 \Rightarrow \exists \epsilon > 0, f(x) > f(x - \epsilon \nabla f_x)$$

i.e. following the opposite direction of the gradient leads to a local minimum of the function.

Gradient Descent

This is the same for J.
Following the gradient of J will lead to a local minimum of our criterion.

$$J_{\mathcal{M}}(\mathbf{X}, \mathbf{Y}|\mathbf{w}, b)$$

Idea

Compute the gradient with respect to $\mathbf{w}$ and b and update them such that :

$$J_{t+1} = J_t + \epsilon \nabla J_t$$

J will become better and better with the iterations

Gradient Descent

The only parameters we can move are $\mathbf{w}$ and b.

- Compute the gradient according to $\mathbf{w}$: $\frac{\partial J}{\partial \mathbf{w}}$
- update $\mathbf{w} \leftarrow \mathbf{w} - \epsilon \frac{\partial J}{\partial \mathbf{w}}$
- Compute the gradient according to b : $\frac{\partial J}{\partial b}$
- update $b \leftarrow b - \epsilon \frac{\partial J}{\partial b}$

Stochastic Gradient descent (SGD)

In practice, $J_{\mathcal{M}}$ is computed on all the training set. Very difficult to compute if the training set is big.
Idea : approximate the training set by picking random subset for J computation

SGD

Estimate gradients with $J_{\mathcal{M}' \subset \mathcal{M}}(\mathbf{X}, \mathbf{Y}|\mathbf{w}, b)$ with $\mathcal{M}'$ randomly picked in $\mathcal{M}$.

Stochastic Gradient descent (SGD)

t is the number of iterations and Δ is the approximation of $\frac{\partial J}{\partial .}$. Then the update rule for the parameters becomes :

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t - \alpha * \Delta \mathbf{w} \\ b_{t+1} &= b_t - \alpha * \Delta b \end{aligned}$$

α is called the learning rate, it rules the speed and quality of the learning process.

Algorithm

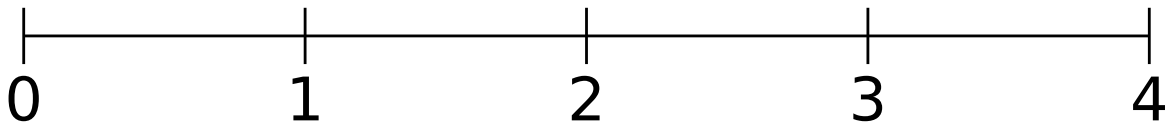
1. **Initialization** : $\mathbf{w} = (w_1, w_2)$ and b random
2. Loop
   - 2.1 Select and random $(x^{(i)}, y^{(i)})$
   - 2.2 Compute Δw_1, Δw_2 and Δb
   - 2.3 Update $\mathbf{w}$ and b

Table of contents

Multiclass hypothesis

Let's suppose for a moment that we do a multiple label classification :

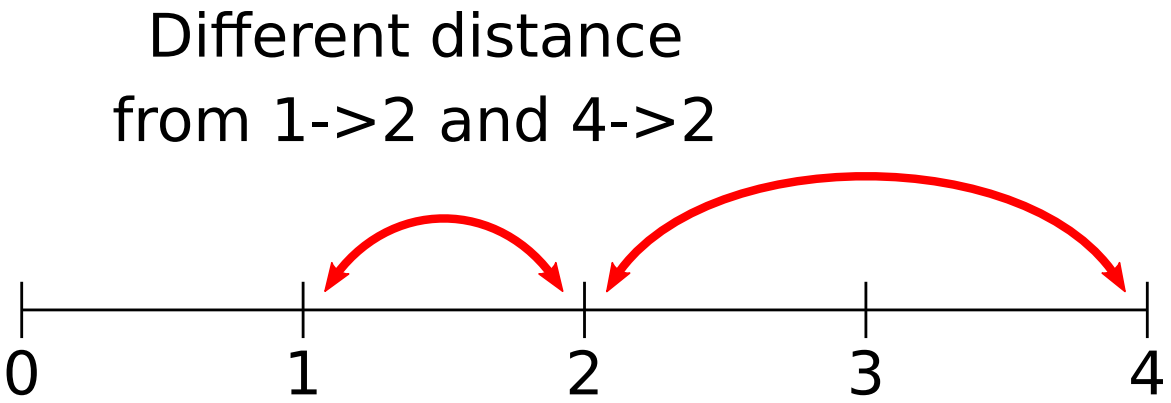
- we could try to predict between 0 and 4 (for 5 labels)
- only multiply by 4 the output of the precedent algorithm



Multiclass hypothesis

Let's suppose for a moment that we do a multiple label classification :

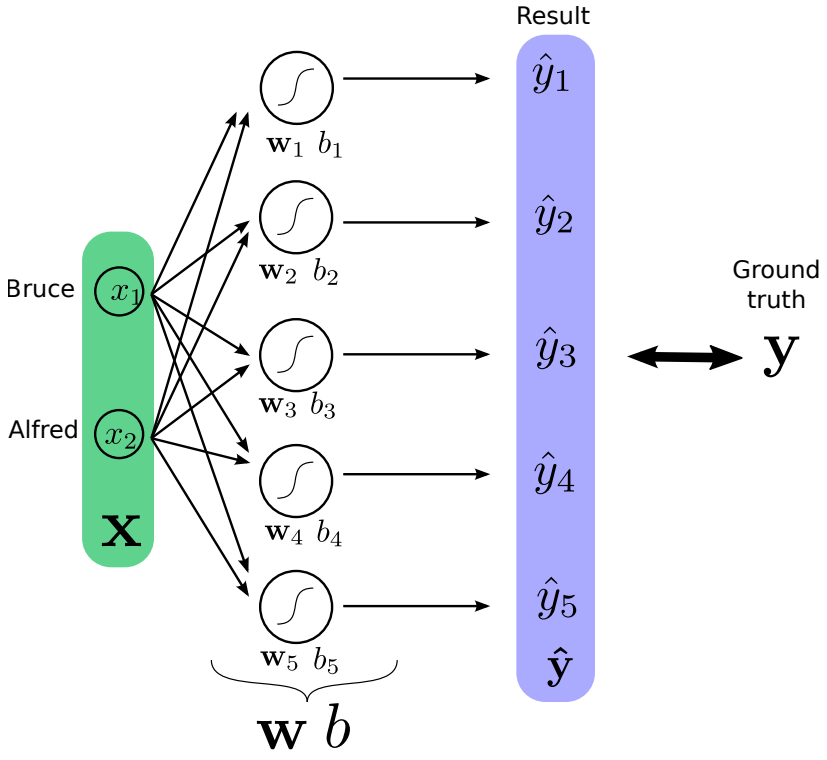
- we could try to predict between 0 and 4 (for 5 labels)
- only multiply by 4 the output of the precedent algorithm



Many neurons

Solution

Using one neuron for each label.



SoftMax

We could select the right class by taking the maximum. But as for the sign function, it is not differentiable.
We use a relaxation, the SoftMax function :

$$p_i = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

The p_is sum to 1 and are positive, they may be refered as a probability of belonging to class i.

The cross entropy

The cross entropy loss measures the similarity of two discrete distributions y and p.

$$L(p, y) = -\sum_i y_i log(p_i)$$

Le c be the indice of the class, i.e. $y_c = 1$ and all other y_i are zero. Then :

$$L(p, c) = -\log(p_c)$$

It is 0 if $p_c = 1$.

The cross entropy

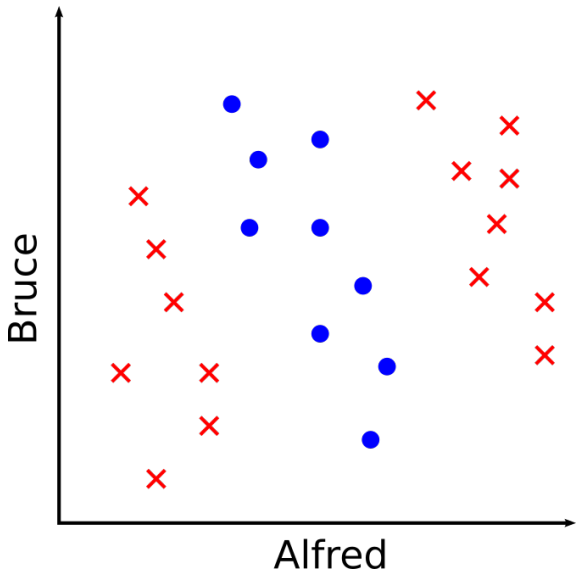
Expression function of the prediction :

$$L(p, c) = -\log(p_c) = -\log(\frac{e^{\hat{y}_c}}{\sum_j e^{\hat{y}_j}}) \quad (2)$$

$$= -\hat{y}_c + \log(\sum_j e^{\hat{y}_j}) \quad (3)$$

Table of contents

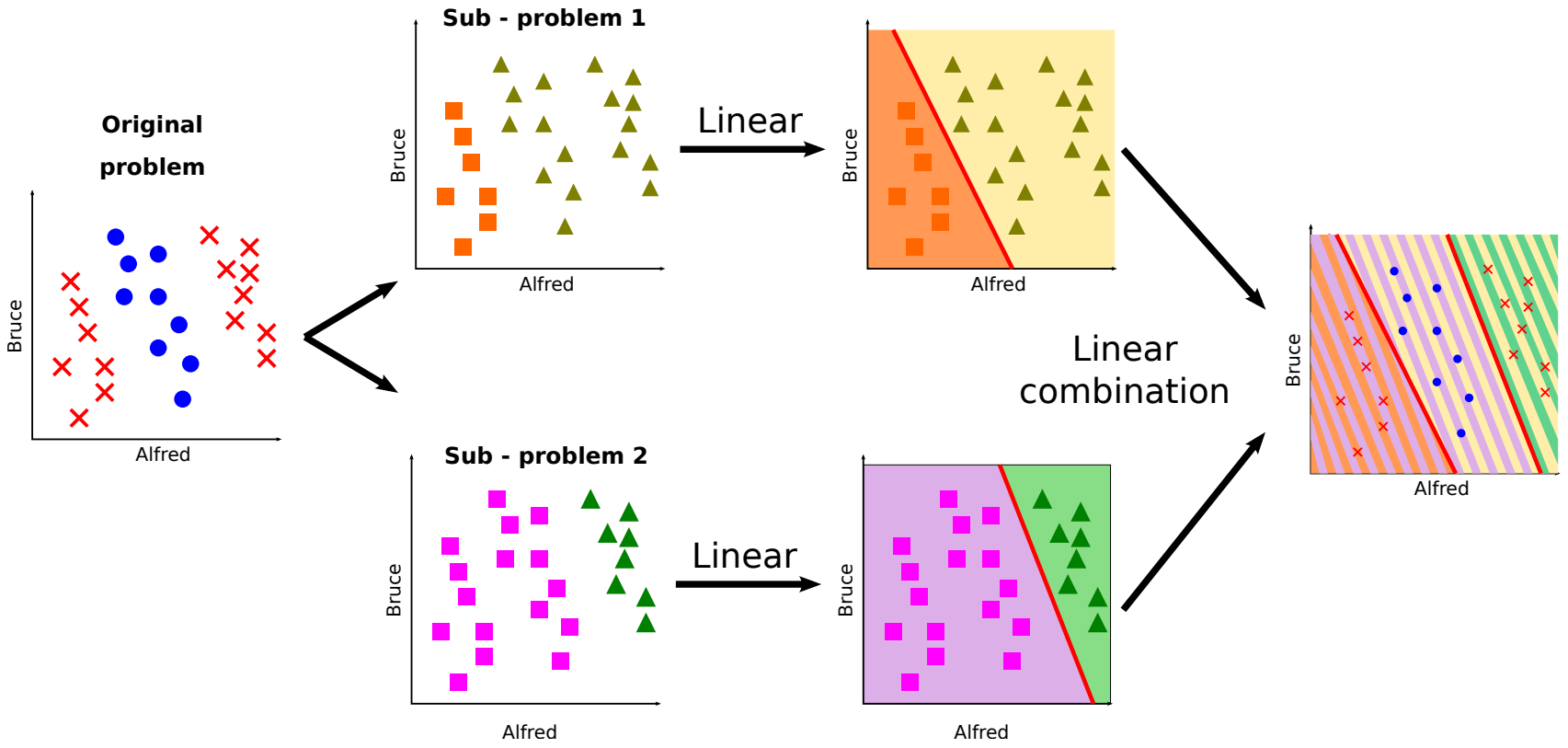
Non linear decision



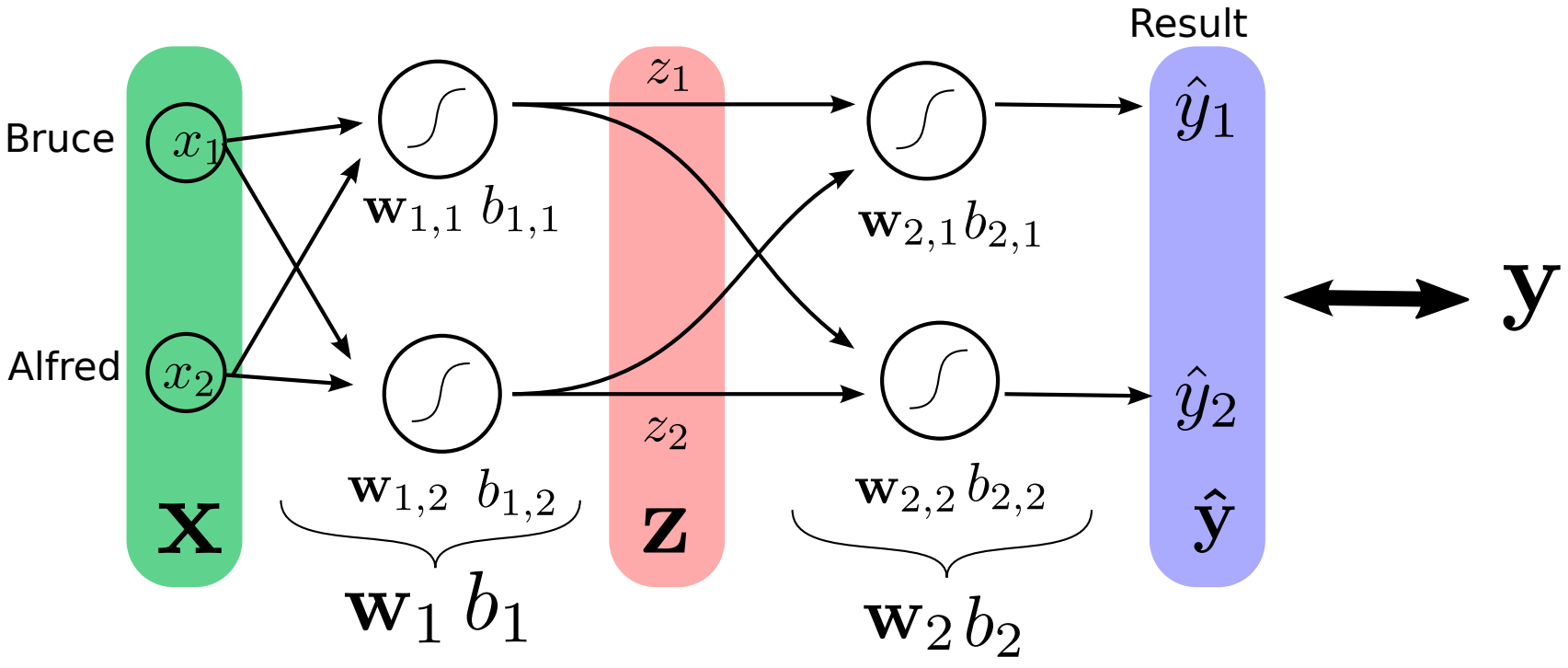
- More difficult problem
- No linear decision

Can we still use neural networks ?

Non linear decision



Non linear decision



Optimization

The whole expression of the neural network is :

$$L(\hat{y}, y) = -\sigma(\mathbf{w}_2 * \sigma(\mathbf{w}_1 * \mathbf{x} + b_1)_c + \log(\sum_j e^{\sigma(\mathbf{w}_2 * \sigma(\mathbf{w}_1 * \mathbf{x} + b_1) + b_2)_j})$$

To optimize it, we need to compute the derivatives according to each parameter.

Optimization

The whole expression of the neural network is :

$$L(\hat{y}, y) = -\sigma(\mathbf{w}_2 * \sigma(\mathbf{w}_1 * \mathbf{x} + b_1)_c + \log(\sum_j e^{\sigma(\mathbf{w}_2 * \sigma(\mathbf{w}_1 * \mathbf{x} + b_1) + b_2)_j})$$

To optimize it, we need to compute the derivatives according to each parameter. $\longrightarrow$ Direct computation may be difficult.

Computing the gradient : the chain rule

Considering a composition of functions f and g :

$$\frac{\partial g(f(a,b))}{\partial a} = \frac{\partial f(a,b)}{\partial a} g'(f(a,b))$$

Then, with three functions :

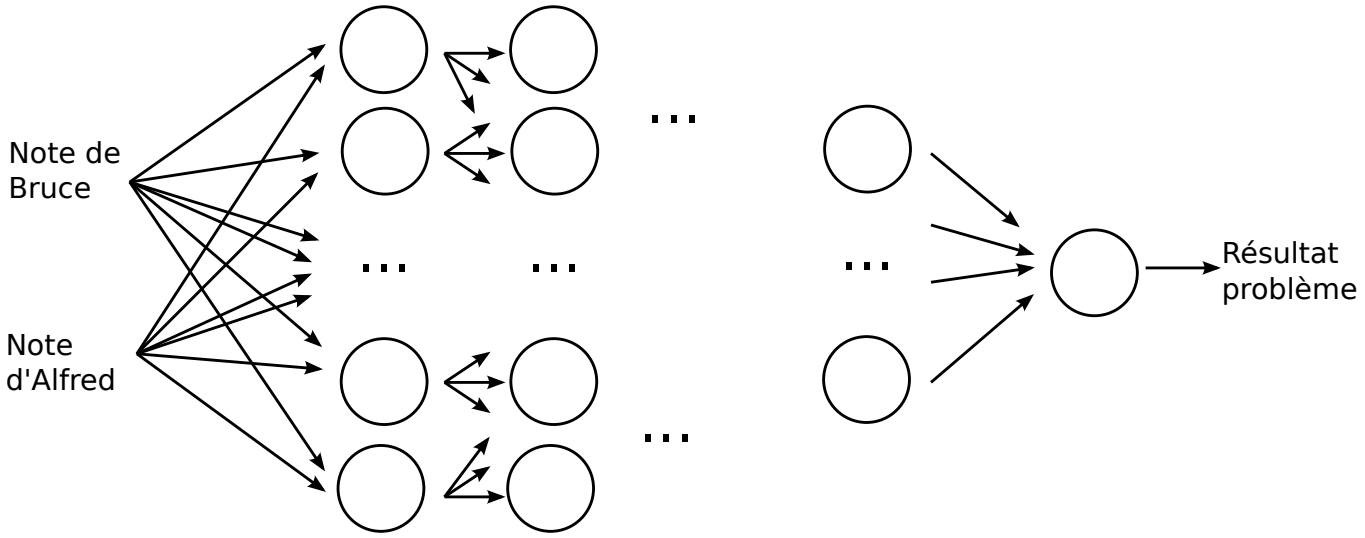
$$\frac{\partial h(g(f(a,b)))}{\partial a} = \frac{\partial f(a,b)}{\partial a} g'(f(a,b)) h'(g(f(a,b)))$$

And so on...

Computing the gradient : in practice

- Compute $d\hat{y} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}}$
- Compute $d\hat{y}_s = \sigma'(\hat{y}_s)d\hat{y}$
- Compute
  - $d\mathbf{w}_2 = \frac{\partial \mathbf{w_2} z + b2}{\partial \hat{\mathbf{w}}_2} * d\hat{y}_s$
  - $db_2 = \frac{\partial \mathbf{w_2} z + b2}{\partial b_2} * d\hat{y}_s$
  - $dz = \frac{\partial \mathbf{w_2} z + b2}{\partial z} * d\hat{y}_s$
- Compute $dz_s = \sigma'(z_s)dz$
- Compute
  - $d\mathbf{w}_1 = \frac{\partial \mathbf{w_1} x + b1}{\partial \hat{\mathbf{w}}_1} * dz_s$
  - $db_1 = \frac{\partial \mathbf{w_1} z + b1}{\partial b_1} * dz_s$

Neural networks



Forward

Iteratively compute the output of the network

Backward

Itertatively compute the derivatives starting from the output

Table of contents

The mean square loss

Expression

$L(\hat{y}, y) = \sum_i (\hat{y}_i - y_i)^2$

Derivative :

$$\frac{\partial L}{\partial \hat{y}} = 2\hat{y}$$

The cross entropy

Expression

$$L(p, c) = -\log(p_c) = -\log(\frac{e^{\hat{y}_c}}{\sum_j e^{\hat{y}_j}}) = -y_c + log(\sum_j e^{\hat{y}_j})$$

Derivative :

$$\frac{\partial H}{\partial y_c} = -1 + \frac{e^{\hat{y}_c}}{\sum_j e^{\hat{y}_j}} \tag{4}$$

$$\frac{\partial H}{\partial y_{i,i\neq c}} = 0 + \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}} \tag{5}$$

$$\tag{6}$$

Then for all i :

$$\frac{\partial H}{\partial y} = p - y$$

The sigmoid

Expression

$y = \sigma(x) = \frac{1}{1+e^{-x}}$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Matrix / Vector operation

Expression

$y = \mathbf{w}x + b$

$$\frac{\partial y}{\partial \hat{\mathbf{w}}} = x$$

$$\frac{\partial y}{\partial b} = 1$$

$$\frac{\partial y}{\partial x} = \mathbf{w}$$

Deep learning

The deep learning session :

- How to work with images ?
- Activations functions
- Optimization