ARCHITECTURES DE RÉSEAUX DE NEURONES

Agro–2A

Vincent Guigue

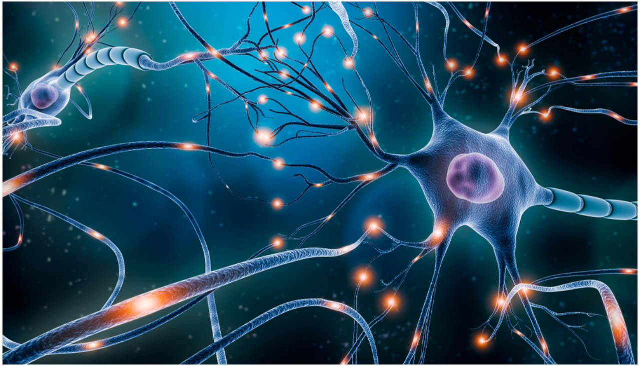
université PARIS-SACLAY

MIA PARIS-SACLAY
EKINOCS

AgroParisTech
Institut des Sciences et Industries du Vivant et de l'Environnement

Introduction au Deep Learning

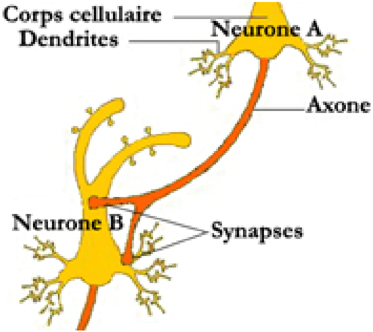
Inspiration biologique [plus ou moins lointaine]

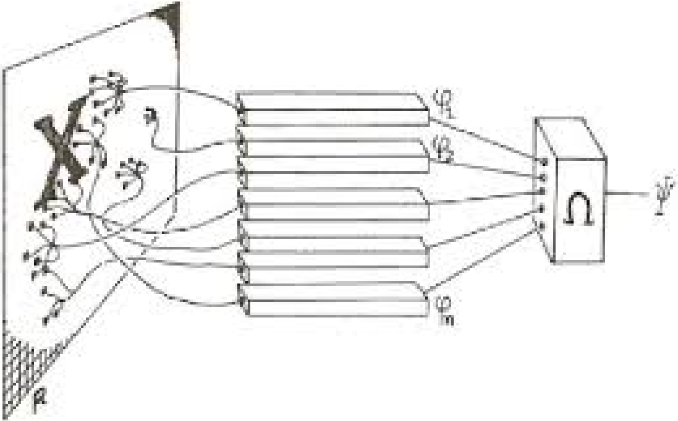


Réseau de neurones

- Opérateur complexe
- Logique d'activation et de fusion des messages
- Nom évocateur et vendeur

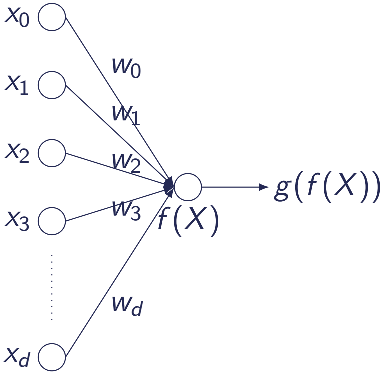
Inspiration biologique [plus ou moins lointaine]





- Feature
- Fusion de message = addition
- Activation = signe (=décision)

Les origines de l'apprentissage profond : le perceptron



Le perceptron

Sur un jeu de données $(\mathbf{x}, y) \in \mathbb{R}^d \times \{-1, 1\}$

- $f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{i=1}^{d} x_i w_i = w_0 + < \mathbf{x}, \mathbf{w} >$
- Fonction de décision : $g(x) = sign(x)$

$\rightarrow$ Sortie : $g(f(\mathbf{x})) = sign(< \mathbf{x}, \mathbf{w} >)$

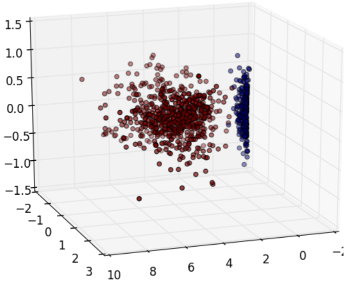
- Problème d'apprentissage : $argmax_{\mathbf{w}} \mathbb{E}_{x,y}[max(0, -yf_{\mathbf{w}}(\mathbf{x}))]$

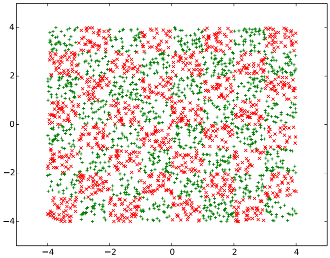
Algorithme du perceptron

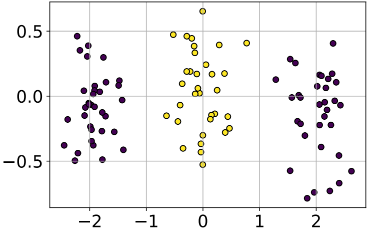
- Tant qu'il n'y a pas convergence :
  - pour tous les exemples (x^i, y^i) :
    - si $(y^i \times < \mathbf{w}.\mathbf{x}^i >) < 0$
      alors $\mathbf{w} = \mathbf{w} + \epsilon y^i \mathbf{x}^i$
- Descente de gradient sur le coût

Limites du perceptron

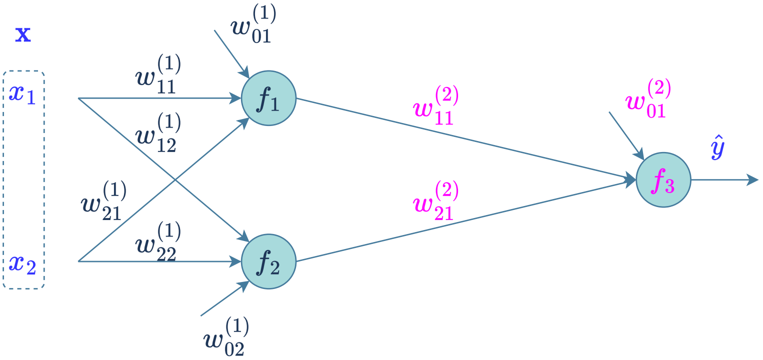
Est-il capable de séparer ces données ?

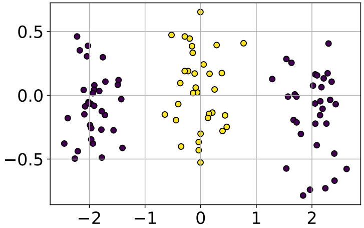






Combinons deux neurones



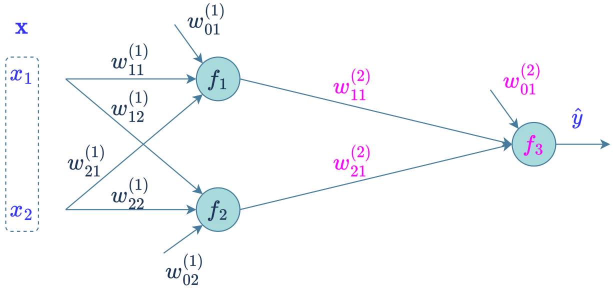


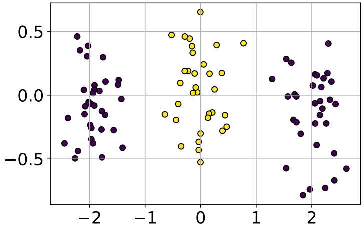
$$f_1(\mathbf{x}) = w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{01}^{(1)}, \qquad f_2(\mathbf{x}) = w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{02}^{(1)}$$

$$f_3(\mathbf{x}) = w_{11}^{(2)} f_1(\mathbf{x}) + w_{21}^{(2)} f_2(\mathbf{x}) + w_{01}^{(2)}$$

Combiner des neurones $\Rightarrow$ suffisant ?

Combinons deux neurones





$$f_1(\mathbf{x}) = w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{01}^{(1)}, \qquad f_2(\mathbf{x}) = w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{02}^{(1)}$$

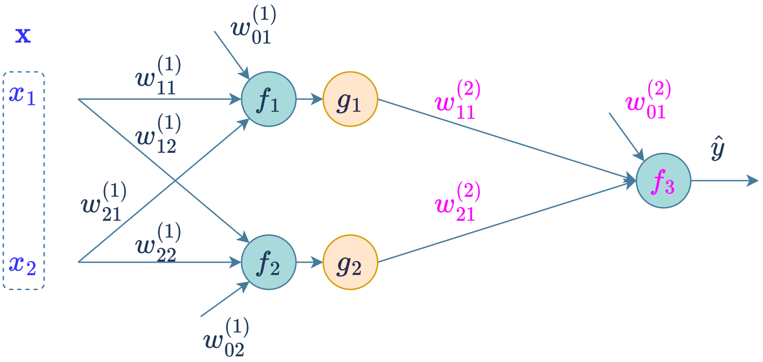
$$f_3(\mathbf{x}) = w_{11}^{(2)} f_1(\mathbf{x}) + w_{21}^{(2)} f_2(\mathbf{x}) + w_{01}^{(2)}$$

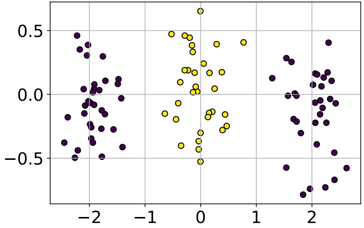
$$f_3(\mathbf{x}) = w_{11}^{(2)}(w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{01}^{(1)}) + w_{21}^{(2)}(w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{02}^{(1)}) + w_{01}^{(2)}$$

$$\Leftrightarrow f_3(\mathbf{x}) = x_1(w_{11}^{(2)} w_{11}^{(1)} + w_{21}^{(2)} w_{12}^{(1)}) + x_2(w_{11}^{(2)} w_{21}^{(1)} + w_{21}^{(2)} w_{22}^{(1)}) + w_{01}^{(2)} + w_{11}^{(2)} w_{01}^{(1)} + w_{21}^{(2)} w_{02}^{(1)}$$

Non ! il faut introduire de la non linéarité, sinon équivalent à un perceptron . . .

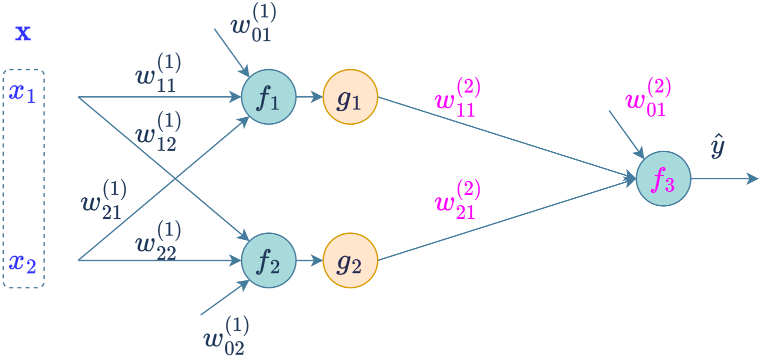
Non-linéarité

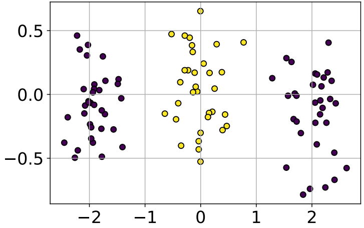




- Quelle non-linéarité ?
  - Fonction *signe* ?
  - $\Rightarrow$ dérivée problématique ...
  - Fonctions *tanh*, *sigmoïde*, ... + biais

Non-linéarité

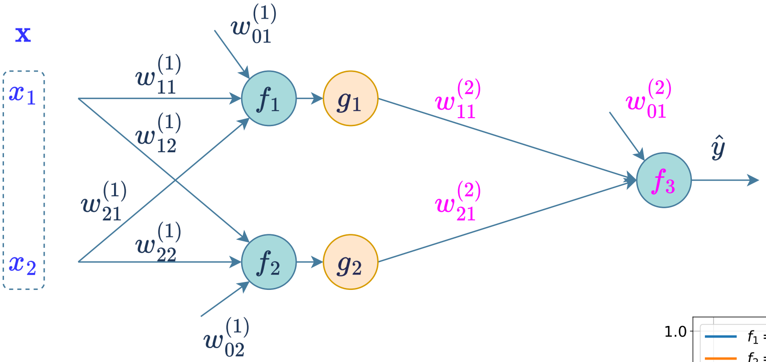


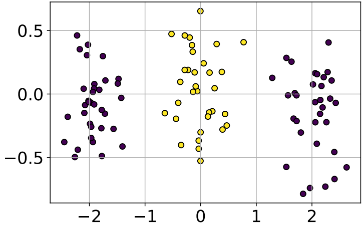


- Quelle non-linéarité ?
  - Fonction *signe* ?

  $\Rightarrow$ dérivée problématique ...
  - Fonctions *tanh*, *sigmoïde*, ... + biais

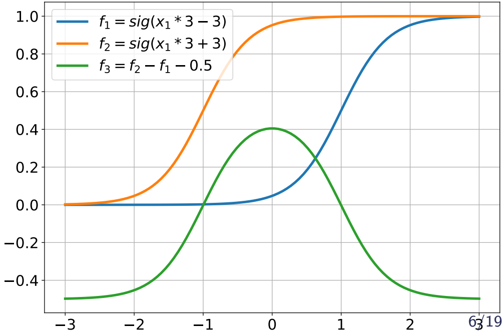
Non-linéarité



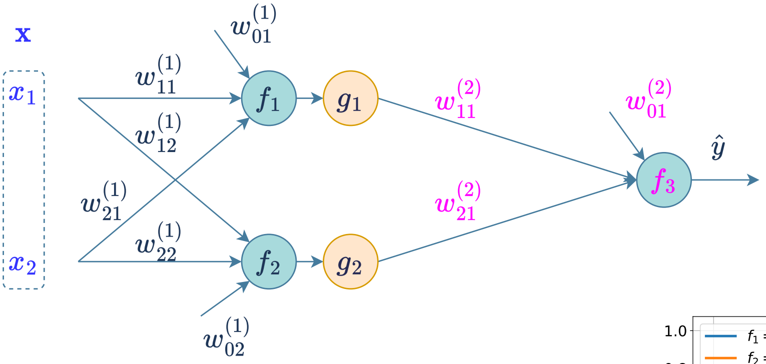


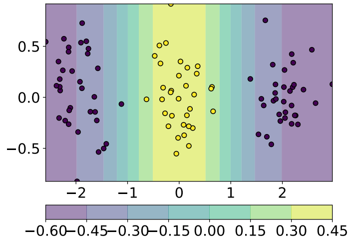
- Quelle non-linéarité ?
  - Fonction *signe* ?
  - ⇒ dérivée problématique ...
  - Fonctions *tanh*, *sigmoïde*, ... + biais

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



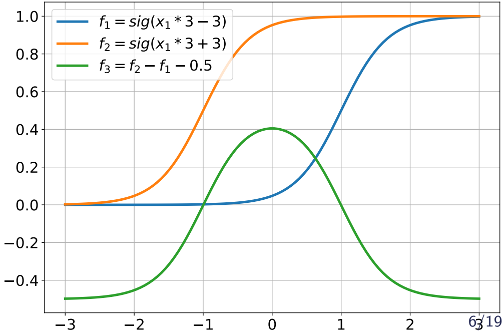
Non-linéarité



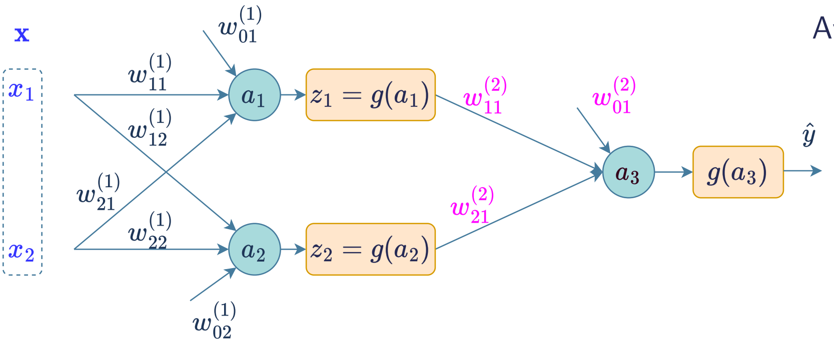


- Quelle non-linéarité ?
  - Fonction *signe* ?
  - $\Rightarrow$ dérivée problématique ...
  - Fonctions *tanh*, *sigmoïde*, ... + biais

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



Vocabulaire de l'inférence



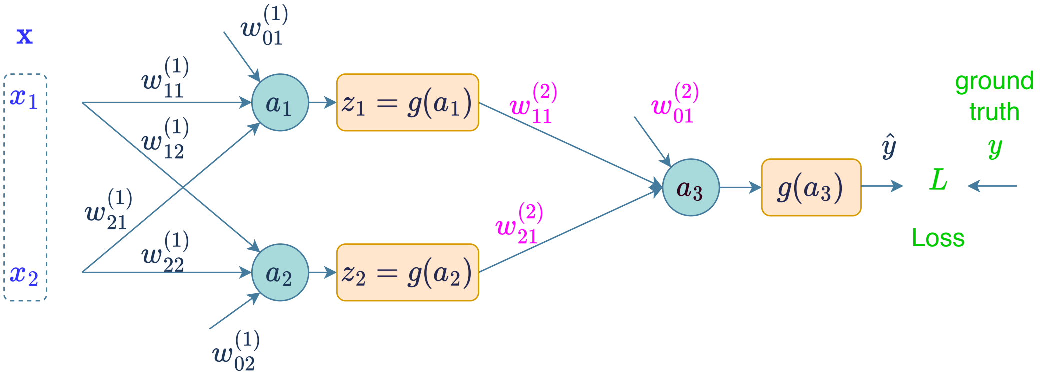
Avec $g(x) = \dfrac{1}{1 + \exp(-x)}$

- $a_1 = w_{01} + w_{11} x_1 + w_{21} x_2$
- $a_2 = w_{02} + w_{12} x_1 + w_{22} x_2$
- $z_1 = g(a_1)$
- $z_2 = g(a_2)$
- $a_3 = w_{01} + w_{11} z_1 + w_{21} z_2$
- $\hat{y} = g(a_3)$

Vocabulaire

- Inférence : *passe forward*
- g fonction d'activation (non linéarité du réseau)
- a_i activation du neurone i
- z_i sortie du neurone i (transformé non linéaire de l'activation).

Apprentissage



Objectif : apprendre les poids

- Choix d'un coût : moindres carrés
  $L(\hat{y}, y) = (\hat{y} - y)^2$ [pourquoi est ce un bon choix ?]
- Mais comment répartir l'erreur entre les poids ?

$\Rightarrow$ Rétro-propagation de l'erreur

Descente de gradient

Objectif: calculer les gradients partiels par rapport aux paramètres

$$\forall i, j, \qquad \frac{\partial L(\hat{y}, y)}{\partial w_{ij}}$$

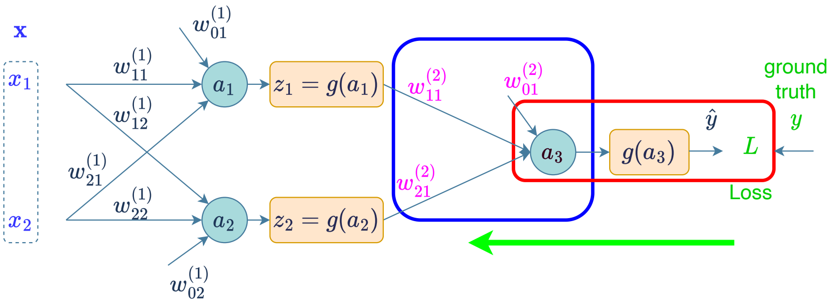
Forward: calcul de $\hat{y}$ [entre autres]

Backward: calcul des gradients

Optimisation: descente de gradient

$$w_{ij} \leftarrow w_{ij} - \underbrace{\varepsilon}_{\text{Learning rate}} \frac{\partial L(\hat{y}, y)}{\partial w_{ij}}$$

Calcul du gradient: chain rule



Forward:
$\hat{y} = 0.5$
$y = -1$

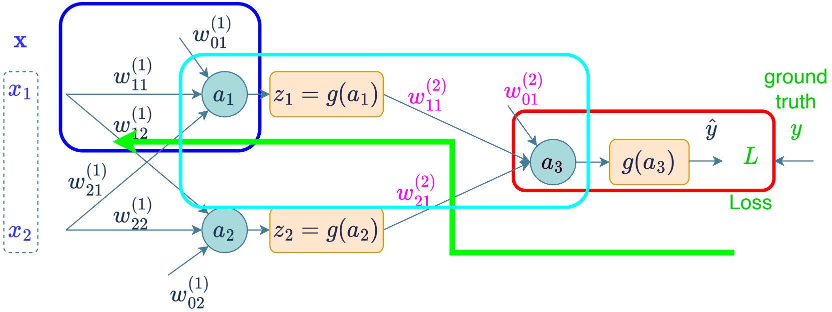
Backward, poids de la **dernière couche** : $\nabla_{w_{ij}^{(2)}} L(\hat{y}, y)$

$$L(\hat{y}, y) = (g(a_3) - y)^2 = \left(g\left(w_{01}^{(2)} + w_{11}^{(2)} z_1 + w_{21}^{(2)} z_2\right) - y\right)^2$$

$$\frac{\partial L}{\partial w_{i1}^{(2)}} = \frac{\partial L}{\partial a_3} \frac{\partial a_3}{\partial w_{i1}^{(2)}} \quad \text{avec} \quad \left| \begin{array}{llll} \frac{\partial L}{\partial a_3} & = \frac{\partial L}{\partial g(a_3)} \frac{\partial g(a_3)}{\partial a_3} = \frac{\partial (g(a_3) - y)^2}{\partial a_3} & & = 2g'(a_3)(g(a_3) - y) \\ \frac{\partial a_3}{\partial w_{i1}^{(2)}} & = \frac{\partial (w_{01}^{(2)} + w_{11}^{(2)} z_1 + w_{12}^{(2)} z_2)}{\partial w_{i1}^{(2)}} & & = z_i \end{array} \right.$$

Soit: $\frac{\partial L}{\partial w_{i1}^{(2)}} = 2g'(a_3)(\hat{y} - y) z_i \qquad \Longrightarrow$ Mise à jour possible

Calcul du gradient: chain rule [suite]



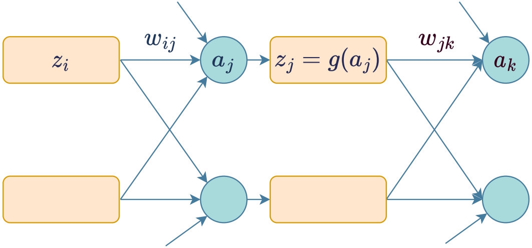
Forward:
$\hat{y} = 0.5$
$y = -1$

Backward, poids de la première couche: $w_{i1}^{(1)}$ (par exemple)

$$\frac{\partial L}{\partial w_{i1}} = \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial w_{i1}} \quad \text{avec} \quad \left| \begin{array}{lll} \frac{\partial L}{\partial a_1} & = \frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial a_1} & = \frac{\partial L}{\partial a_3} g'(a_1) w_{11}^{(2)} \\ \frac{\partial a_1}{\partial w_{i1}} & = \frac{\partial w_{01}^{(1)} + w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2}{\partial w_{i1}^{(1)}} & = x_i \end{array} \right.$$

Soit: $$\underbrace{\frac{\partial L}{\partial w_{i1}}}_{\text{correction de } w_{i1}} = \frac{\partial L}{\partial a_1} x_i = \underbrace{\frac{\partial L}{\partial a_3}}_{\text{erreur à propager}} \underbrace{g'(a_1) w_{13}}_{\text{poids de la connexion}} x_i$$

Cas général dans les couches intermédiaires



$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial a_j}{\partial w_{ij}} \frac{\partial L}{\partial a_j} = z_i \frac{\partial L}{\partial a_j}$$

$$\frac{\partial L}{\partial a_j} = \sum_k \frac{\partial a_k}{\partial a_j} \frac{\partial L}{\partial a_k}$$

$$\underbrace{\frac{\partial L}{\partial a_j}}_{\text{erreur sur } j} = \sum_k (g'(a_k) w_{jk}) \underbrace{\frac{\partial L}{\partial a_k}}_{\text{erreur à propager}}$$

On note: $\delta_j = \dfrac{\partial L}{\partial a_j}$

- Lorsque l'erreur *arrive* de plusieurs sources $\Rightarrow$ somme
- Expression de l'erreur de la couche j par rapport à l'erreur de la couche k

Conclusion

- Une architecture **modulaire**
- ...qui calcule les gradients de manière autonome

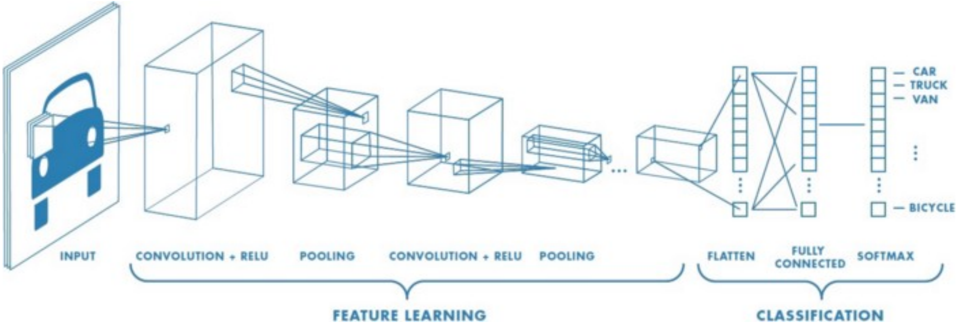
$\Rightarrow$ Beaucoup de choses se jouent dans le choix de la **fonction coût**

Les questions ouvertes

- Quels modules?
- Pour construire quelle architecture?

Les architectures

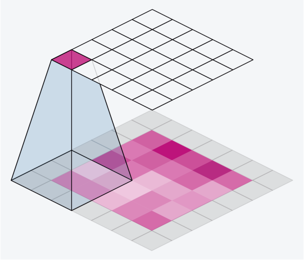
Réseaux convolutifs



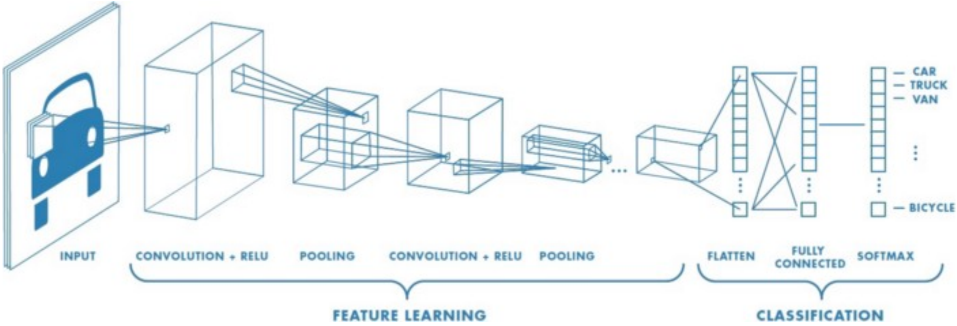
Convolution

Filtre d'analyse glissant sur l'image

- Peu de paramètres
- Apprentissage des motifs à extraire
- Agrégation progressive des échelles



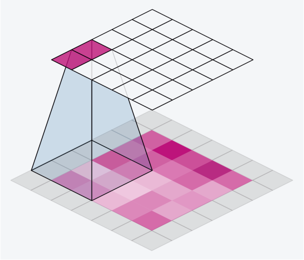
Réseaux convolutifs



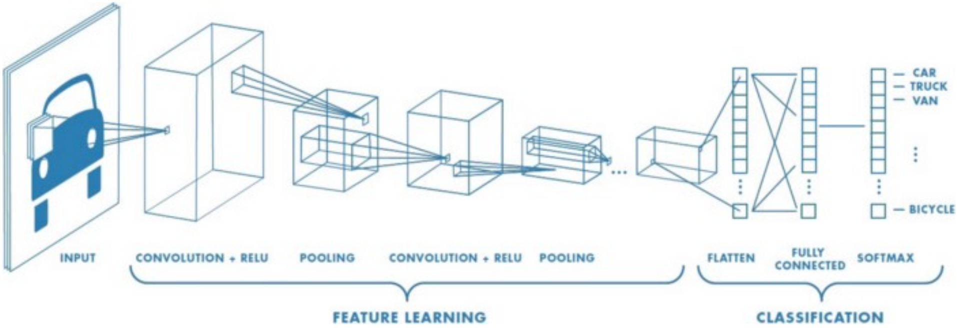
Convolution

Filtre d'analyse glissant sur l'image

- Peu de paramètres
- Apprentissage des motifs à extraire
- Agrégation progressive des échelles



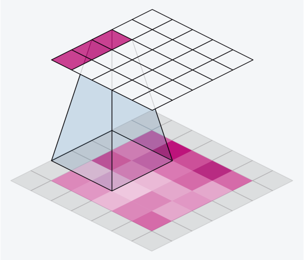
Réseaux convolutifs



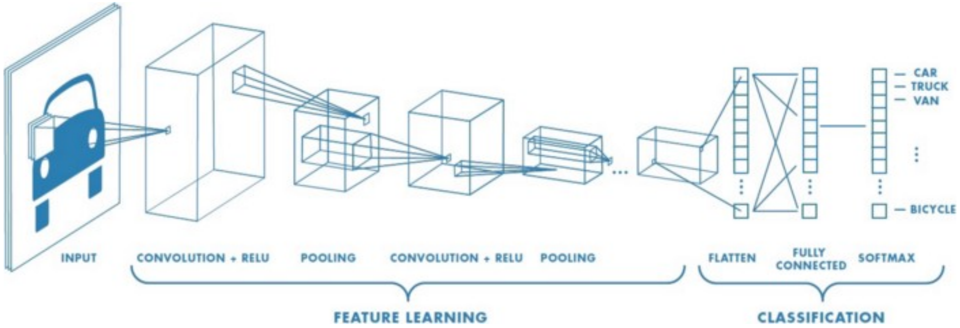
Convolution

Filtre d'analyse glissant sur l'image

- Peu de paramètres
- Apprentissage des motifs à extraire
- Agrégation progressive des échelles



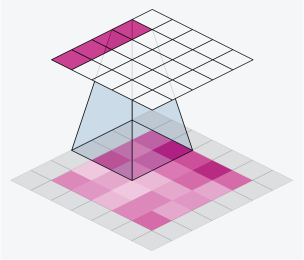
Réseaux convolutifs



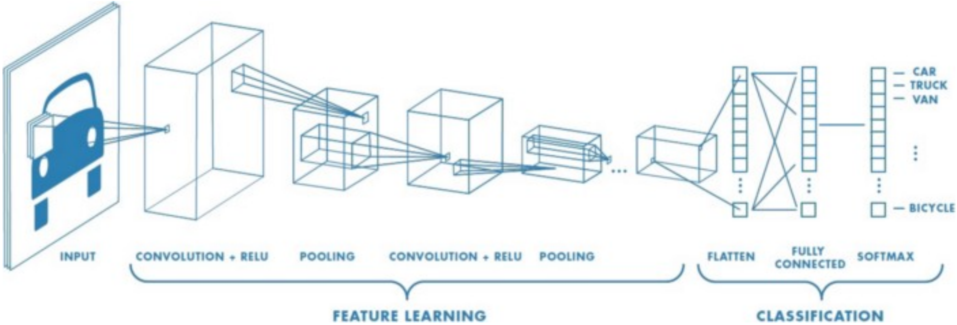
Convolution

Filtre d'analyse glissant sur l'image

- Peu de paramètres
- Apprentissage des motifs à extraire
- Agrégation progressive des échelles



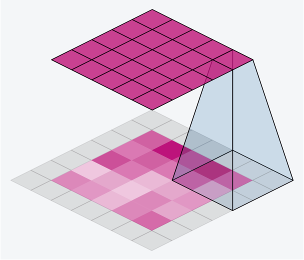
Réseaux convolutifs



Convolution

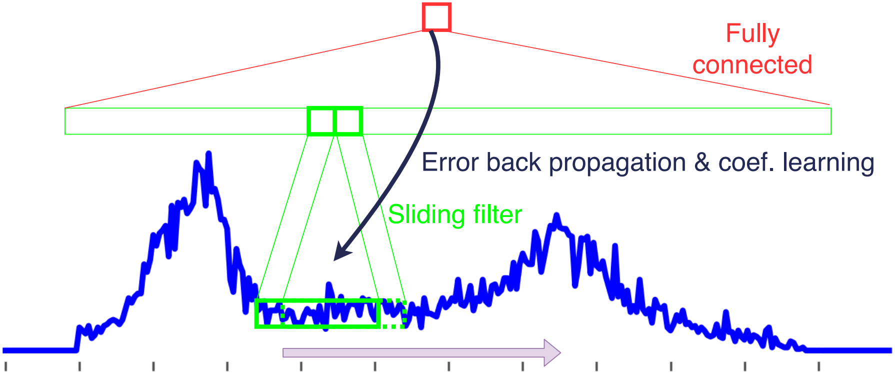
Filtre d'analyse glissant sur l'image

- Peu de paramètres
- Apprentissage des motifs à extraire
- Agrégation progressive des échelles



Convolution 1D

Commençons par un exemple 1D pour bien comprendre.

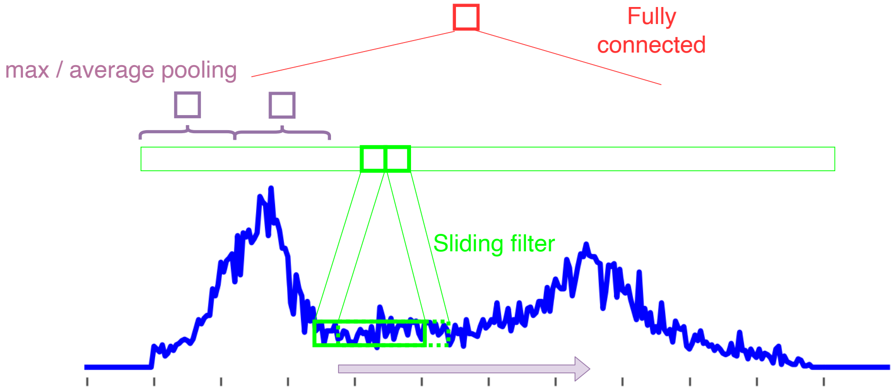


- Fenêtre glissante
- Peu de paramètres
- Largeur / stride / padding

- Apprentissage des motifs à détecter
- Invariance / dépendance à la localisation

Convolution 1D

Commençons par un exemple 1D pour bien comprendre.

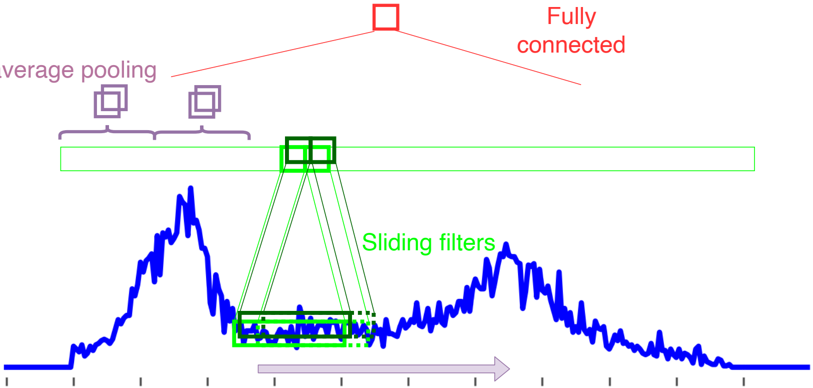


- Fenêtre glissante
- Peu de paramètres
- Largeur / stride / padding

- Apprentissage des motifs à détecter
- Invariance / dépendance à la localisation

Convolution 1D

Commençons par un exemple 1D pour bien comprendre.

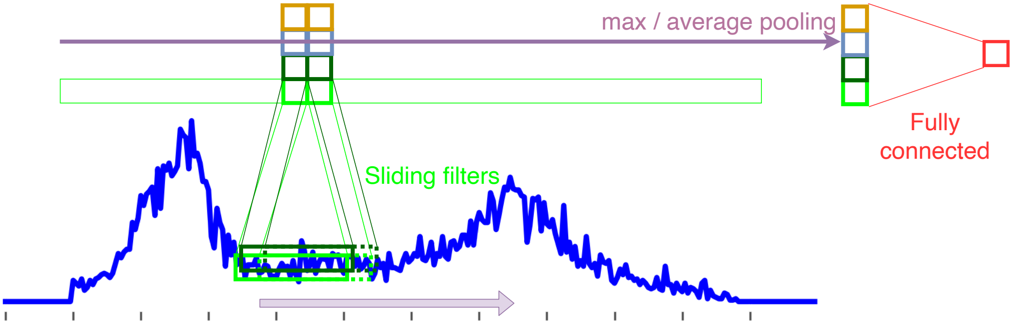


- Fenêtre glissante
- Peu de paramètres
- Largeur / stride / padding

- Apprentissage des motifs à détecter
- Invariance / dépendance à la localisation

Convolution 1D

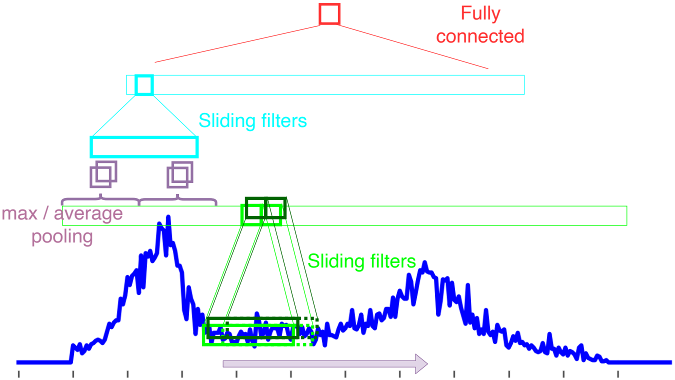
Commençons par un exemple 1D pour bien comprendre.



- Fenêtre glissante
- Peu de paramètres
- Largeur / stride / padding
- Apprentissage des motifs à détecter
- Invariance / dépendance à la localisation

Convolution 1D

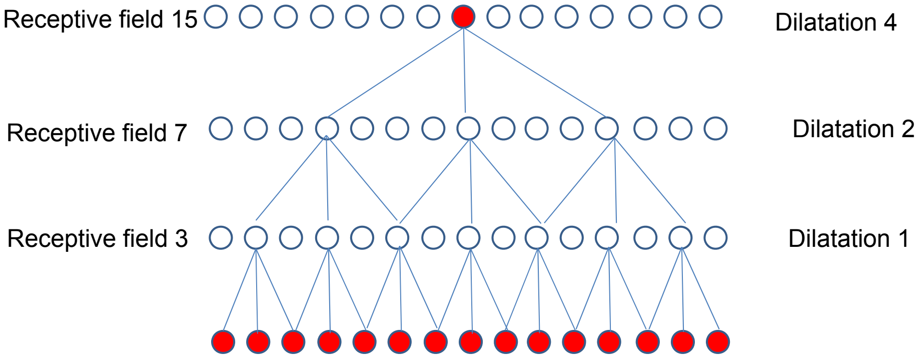
Commençons par un exemple 1D pour bien comprendre.



- Fenêtre glissante
- Peu de paramètres
- Largeur / stride / padding
- Apprentissage des motifs à détecter
- Invariance / dépendance à la localisation

Convolution 1D

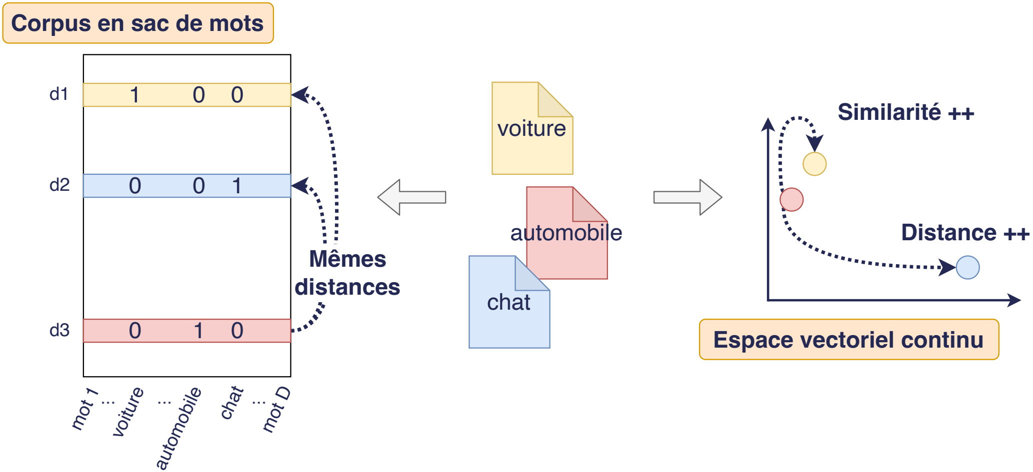
Commençons par un exemple 1D pour bien comprendre.



- Fenêtre glissante
- Peu de paramètres
- Largeur / stride / padding
- Apprentissage des motifs à détecter
- Invariance / dépendance à la localisation

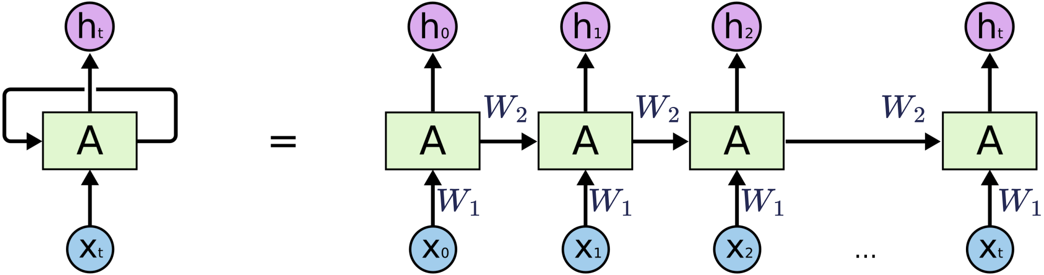
Représentation d'éléments discrets ou continus?

Machine Learning = difficile de représenter des éléments discrets:
variables catégorielles, mots, utilisateur



Objectif: introduire la notion de distance $\neq$ orthogonalité entre les catégories

Recurrent Neural Network



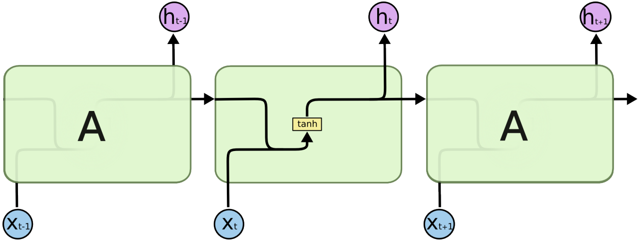
General RNN:

$$\mathbf{h}_t = \tanh(\mathbf{x}_t W_1 + \mathbf{h}_{t-1} W_2 + b) \text{ or } \tanh((x_t \oplus h_{t-1})W + b)$$

- **Few parameters**
- Ability to deals with **variable lengths**
- Capture the **dynamics** of the sequence

Cell details

Classical RNN :



- Latent state $h_t \in \mathbb{R}^d$
- input $x_t \in \mathbb{R}^n$
- $h_t = f_W(h_{t-1}, x_t) = \tanh((x_t \oplus h_{t-1})W + b)$
- $W \in \mathbb{R}^{n\times d}$, $b \in \mathbb{R}$

  - Initial state: h_1
  - Sequence $[x_1, \ldots x_T] \Rightarrow [h_1, \ldots h_T]$
  - May be computed left $\rightarrow$ right (h_t) ... Or right $\rightarrow$ left (h_t^R)

RNN... For which purpose?

- Predicting the next step (from h_T)
- Generation = prediction (in loop)
- Classifying a sequence (from h_T)
- Detecting an event (at every h_t)

Bi-RNN: use h_t & h_t^R
(h_T & h_1^R = caract. whole sequence)