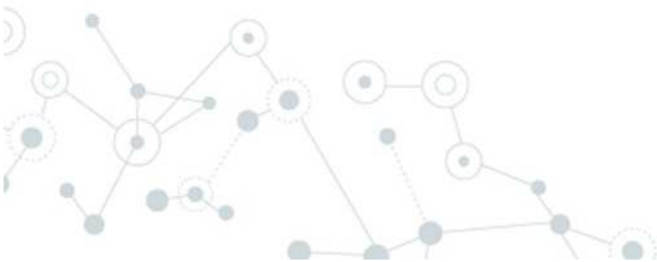




# Apprentissage automatique

Sous-domaine de l'informatique qui développe des méthodes (algorithmes) permettant aux machines de construire automatiquement des modèles à partir de données.



# Types d'apprentissage automatique

Régression: Apprendre un modèle qui prédit des **valeurs continues**

- le prix d'une maison à partir d'un certain nombre de caractéristiques
- l'âge d'une personne à partir de sa taille

Classification : Apprendre un modèle qui prédit des **classes discrètes** prédéfinies

- si un mail est un spam ou pas spam
- le genre d'un texte (sport, politique, loisirs, etc.)
- la partie de discours d'un mot (déterminant, nom commun, verbe, etc.)

Clustering: Apprendre à regrouper des observations en classes discrètes non prédéfinies (ou **clusters**) qui ont des caractéristiques homogènes

- études socio-démographiques (identification de classes de personnes qui ont des comportements comparables)

Ordonnancement (ou *ranking*):

- étant donnée une requête dans un navigateur de recherche (Google, Bing, etc.), ordonnancer les pages retrouvées selon leur pertinence
- étant donnée une phrase en anglais, ordonnancer plusieurs traductions françaises possibles selon leur fiabilité/fluidité (c'est-à-dire laquelle est la meilleure)


# Types d'apprentissage automatique



- **Apprentissage supervisé**

- La machine apprend un modèle des données à partir des données étiquetées.
  - à partir de mails étiquettes "spam" ou "pas spam", apprendre si un mail est un spam ou non
  - à partir de phrases étiquetées avec des parties du discours, apprendre à assigner des étiquettes à de nouvelles phrases

- **Apprentissage non supervisé**

- La machine apprend un modèle des données sans avoir accès aux étiquettes des données
    - découvrir des groupes d'utilisateurs Twitter à partir du contenu de leurs tweets
    - découvrir des groupes de parties du discours selon les distributions des mots (sans regarder des étiquettes assignées par un expert)
- 

# Etapes de l'apprentissage supervisé




## 1. **Entraînement:**

Apprendre la relation entre les observations et la variable à prédire (la classe, un nombre réel, etc.)

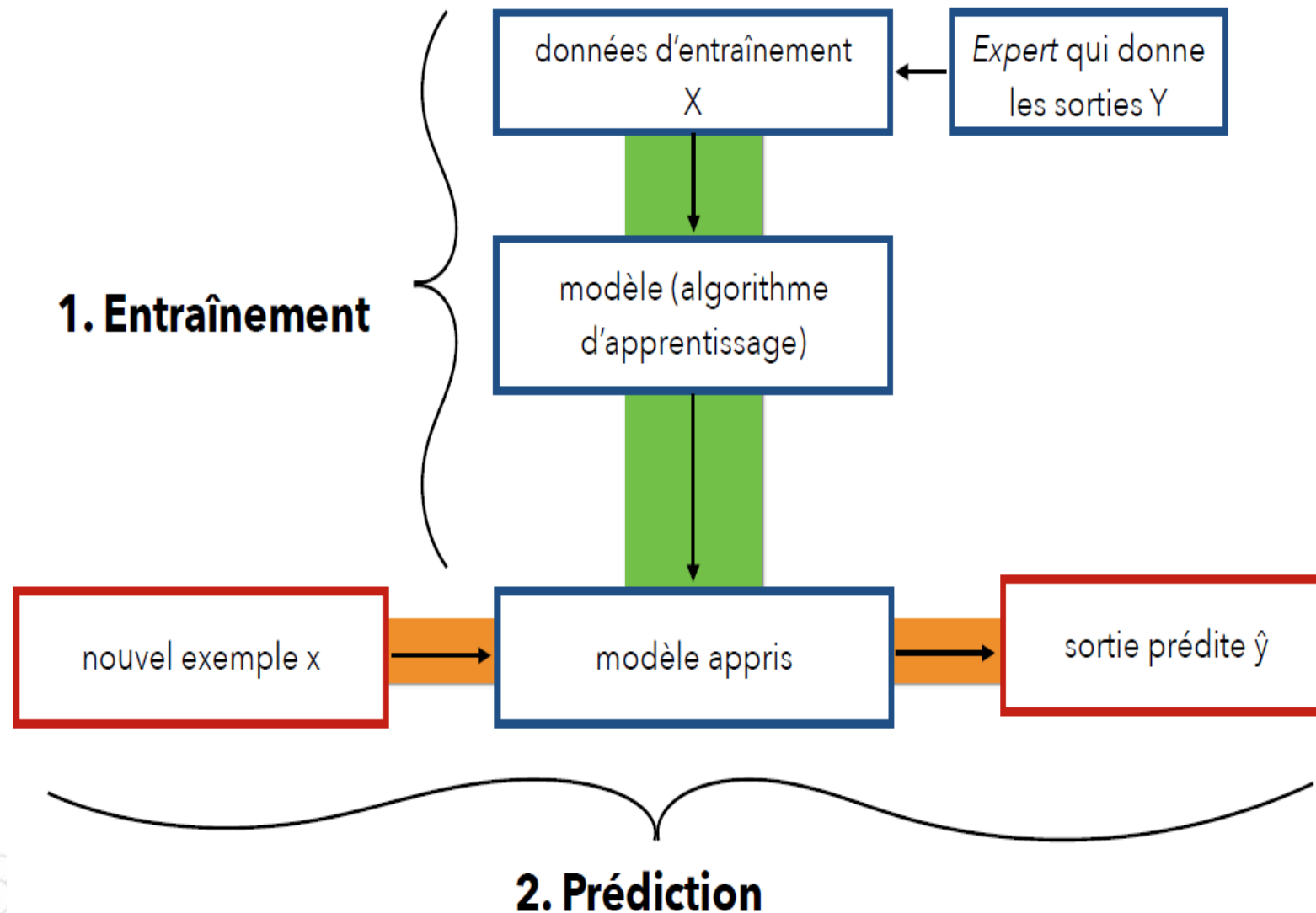
- En utilisant des données d'entraînement
- Où on connaît la sortie attendue

## 2. **Prédiction:**

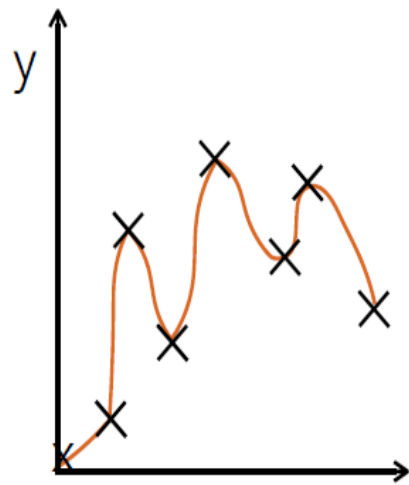
Utiliser cette relation apprise pour faire des prédictions sur des observations non encore vues



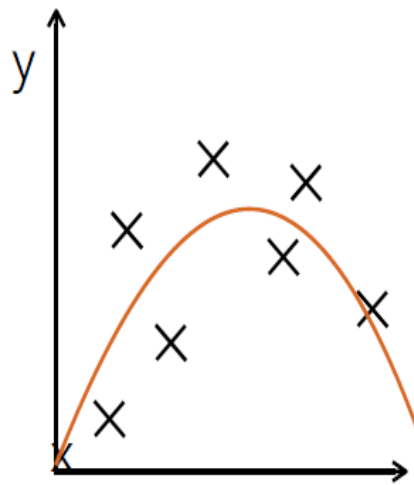
# Etapes de la classification supervisée



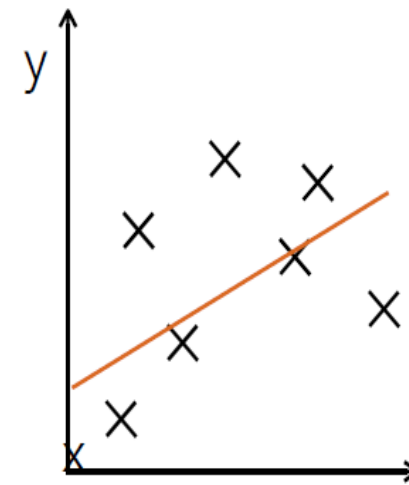
# Sur- et sous-apprentissage



sur-apprentissage  
(overfitting)



un bon compromis !



sous-apprentissage  
(underfitting)

**Très bon**

Performance sur les données d'entraînement

**Médiocre**

**Très mauvais**

Performance sur les données de test

**Médiocre**

# Sur- et sous-apprentissage



- **Comment éviter le sous-apprentissage ?**
  - Utiliser plus de traits pour créer un modèle plus expressif - mieux caractériser les données
  - Utiliser une fonction qui permet de mieux exprimer la relation entre les entrées et les sorties
- **Comment éviter le sur-apprentissage ?**
  - Utiliser moins de traits (ou des traits moins spécifiques) - permet de mieux généraliser aux données non vues
  - Paramétrer le modèle en utilisant un jeu de développement

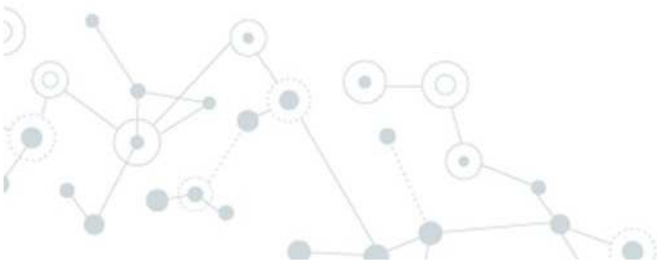


# Jeux de données



On sépare typiquement les données en trois jeux de données:

- **Entraînement (train)** - pour estimer notre fonction
- **Développement (dev)** - pour peaufiner (et choisir les valeurs des hyper-paramètres)
- **Test (test)** - pour tester sur des données non vues une fois que notre modèle a été choisi





# Jeux de données pour la tâche de désambiguïsation morpho-syntaxique (POS tagging)

Corpus brut:

Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 .  
Mr. Vinken is chairman of Elsevier N.V. , the Dutch publishing group .

Corpus annoté:

Pierre Vinken    PROPN  
,    COMMA  
61 years old    ADJ  
,    COMMA  
will    AUX  
join    VERB  
the DET  
board    NOUN  
as    ADP  
a    DET  
nonexecutive    ADJ  
director    NOUN  
Nov. 29 PROPN  
.    SENT

Mr. PROPN  
Vinken    PROPN  
is    VERB  
chairman    NOUN  
of    ADP  
Elsevier    PROPN  
N.V.    PROPN  
,    COMMA  
the DET  
Dutch    PROPN  
publishing    VERB  
group    NOUN  
.    SENT

# Jeux de données pour la tâche de reconnaissance d'entités nommées (Named Entity Recognition)

Corpus brut:

Mary Barra appointed as General Motors chief

December 10 , 2013

The United States 's largest car manufacturer General Motors today named Mary Barra as its new chief executive .

Corpus annoté:

Mary B-PER

Barra I-PER

appointed O

as O

General B-ORG

Motors I-ORG

chief O

December O

10 O

, O

2013 O

The B-ORG

United I-ORG B-LOC

States I-ORG I-LOC

's I-ORG

largest I-ORG

car I-ORG

manufacturer I-ORG

General I-ORG

Motors I-ORG

today O

named O

Mary B-PER

Barra I-PER

as O

its O

new O

chief O

executive O

. O



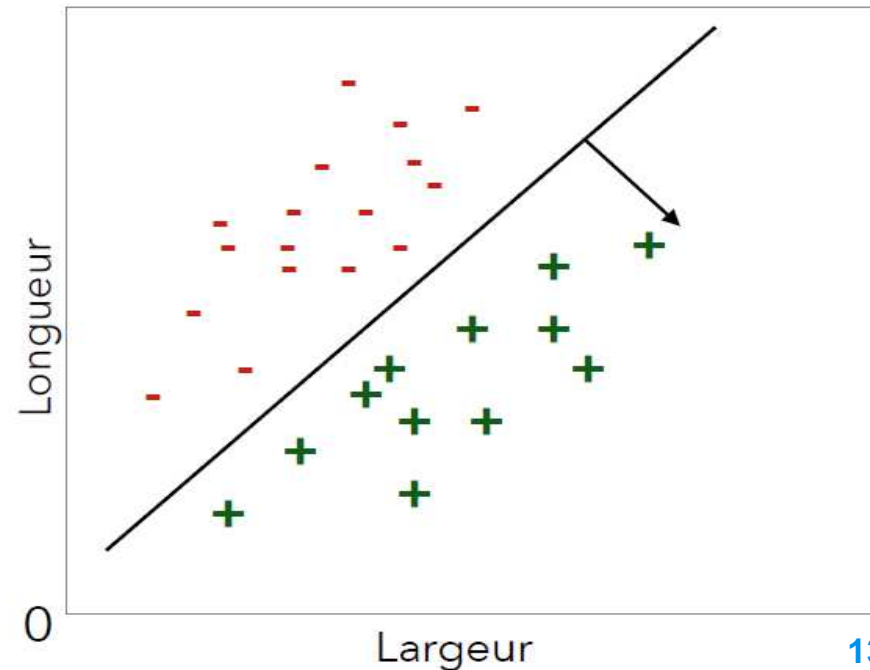
# Les réseaux de neurones

# Le Perceptron

- Rosenblatt (1957) - algorithme du perceptron
- Apprentissage d'un classifieur linéaire
- Algorithme d'apprentissage **"en ligne"**
  - les poids sont mis à jour au fur et à mesure
- Si les données d'entraînement sont séparables, le perceptron trouvera des valeurs pour séparer les données

# Le Perceptron – Exemple d'application: Séparation des données

- Trouver un hyperplan qui sépare les données tel que toutes les données d'une même classe se trouvent d'un même côté de l'hyperplan
- Exemple: classification d'espèces d'iris
  - Deux classes:
    - + : espèce 1
    - : espèce 2
  - Deux traits:
    - $x_1$  : longueur des pétales
    - $x_2$  : largeur des pétales



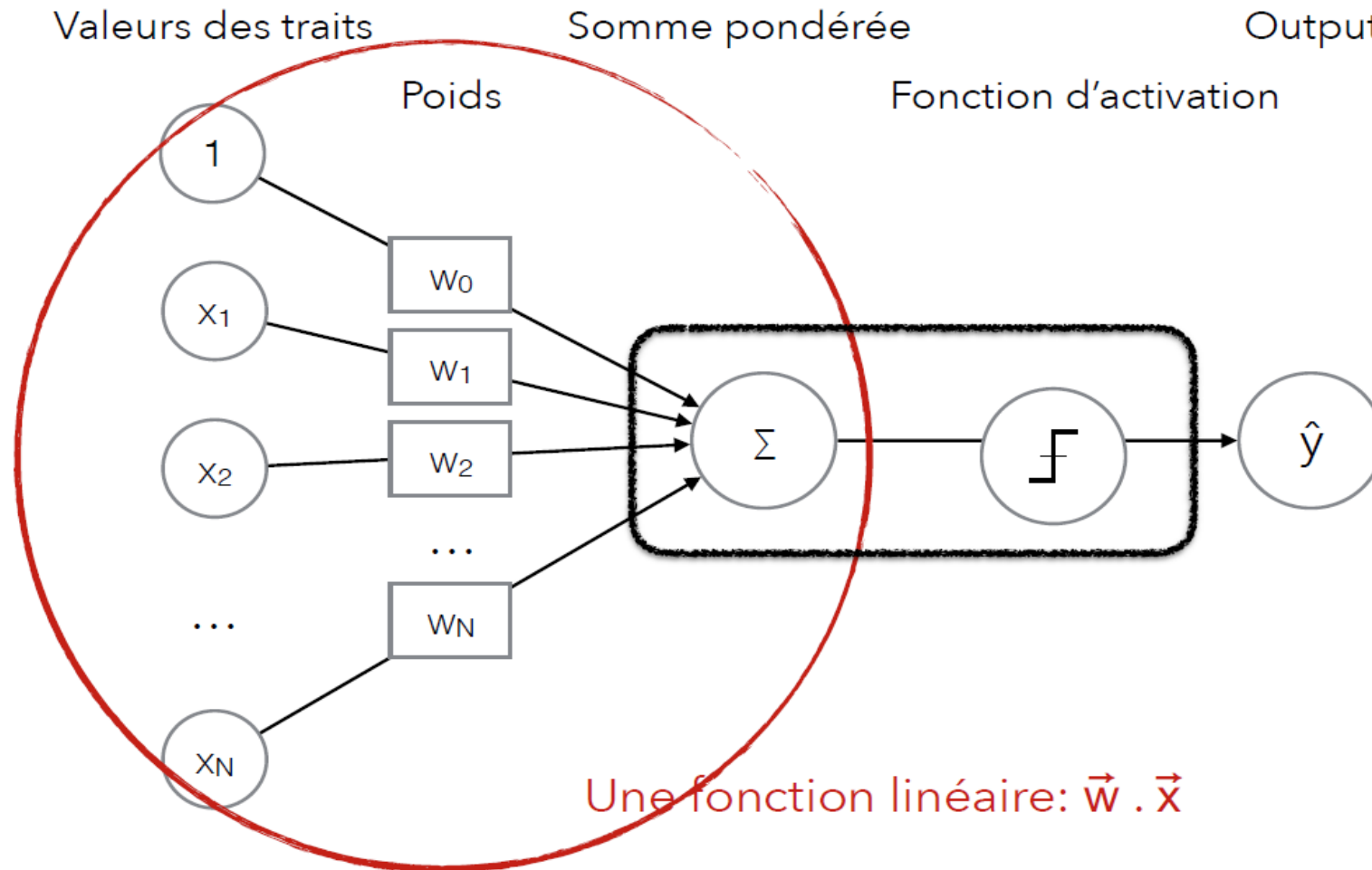
# Le Perceptron – Exemple d'application: Séparation des données - Modèle linéaire

- Comme avant:
  - traits  $\vec{x}$  (vecteur de nombres réels)
  - poids  $\vec{w}$  (vecteur de nombres réels)
  - combinaison linéaire:  $\vec{w} \cdot \vec{x}$

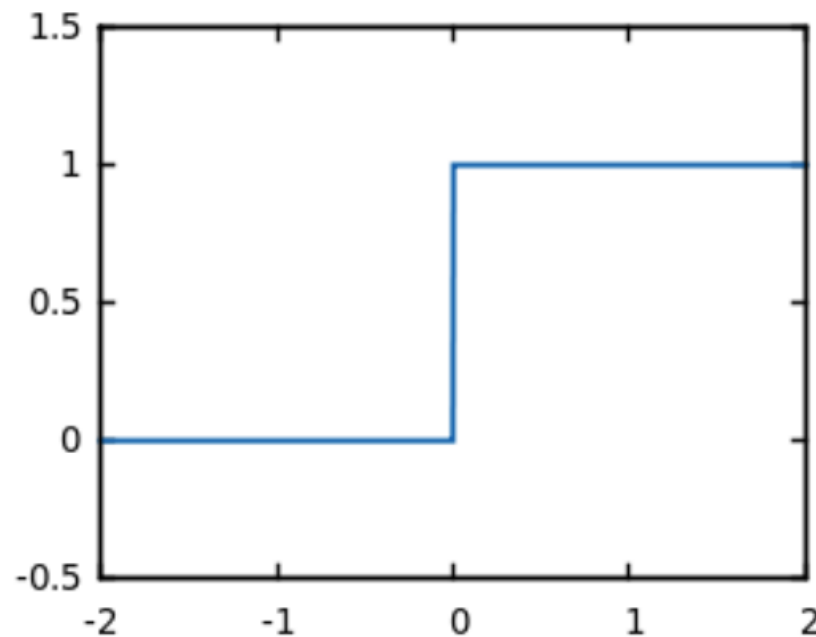
partie linéaire

- Ensuite, on donne le résultat en entrée à une fonction d'activation pour prédire la classe
  - typiquement:
    - si  $\vec{w} \cdot \vec{x} > 0$ , alors classe 1
    - sinon, classe -1

# Le Perceptron – Concepts de base



# Le Perceptron – Fonction d'activation



Un neurone prototypique:

- addition des entrées
- comparaison à un seuil prédéfini
- si en-dessous, le neurone ne s'active pas et sa sortie est zéro
- si au-dessus, le neurone s'active et sa sortie est un



# Le Perceptron binaire

- Appliquer la fonction linéaire
  - = somme pondérée des valeurs de traits
  - = un produit scalaire  $\vec{x} \cdot \vec{w}$
- Passe le résultat par une fonction d'activation
  - E.g. 
$$\phi(z) = \begin{cases} 1 & \text{si } z > 0 \\ 0 & \text{sinon} \end{cases}$$
- Fonction de perte:
  - 0-1 (1 si correct, 0 si pas correct)

# Le Perceptron: Algorithme d'apprentissage pour 2 classes

- Données d'entraînement  $X, Y$  de taille  $D$
- Deux classes:  $+1$  et  $-1$
- Paramètres du modèle  $\vec{w}$  (initialisés aléatoirement)
- $\alpha$ : degré de changement

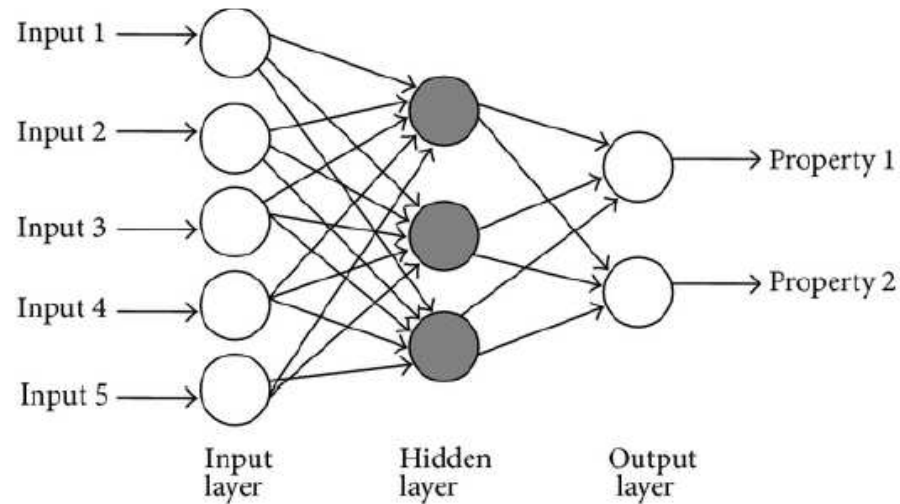
- Pour un certain nombre d'*epochs* (itérations, époques):
  - Pour  $i$  allant de 1 à  $D$ :
    - $\hat{y}^{(i)} = \text{step\_function}(\vec{w} \cdot \vec{x}^{(i)})$  // prediction
    - si  $\hat{y}^{(i)} \neq y^{(i)}$ :
      - $\vec{w} = \vec{w} + y^{(i)} \cdot \alpha \cdot \vec{x}^{(i)}$  // update
  - Si toutes les prédictions sont correctes, s'arrêter

# Le Perceptron: Algorithme d'apprentissage pour 3 classes et plus

- Données d'entraînement  $X, Y$  de taille  $D$
- **Plusieurs classes:  $c \in C$**
- **Paramètres du modèle  $\vec{w}$  (initialisé aléatoirement),**
  - **un vecteur  $\vec{w}$  pour chaque classe, noté  $\vec{w}_c$**
- $\alpha$ : degré de changement

- Pour un certain nombre d'*epochs* (itérations):
  - Pour  $i$  allant de 1 à  $D$ :
    - **$\hat{y}^{(i)} = \operatorname{argmax}_{c \in C} \vec{w}_c \cdot \vec{x}^{(i)}$  // prediction**
    - si  $\hat{y}^{(i)} \neq y^{(i)}$ :
      - **$\vec{w}_{y^{(i)}} = \vec{w}_{y^{(i)}} + \alpha \cdot \vec{x}^{(i)}$  // update**
  - Si toutes les prédictions sont correctes, s'arrêter

# Le Perceptron multi-couches



## Définition

- Au moins une couche avec fonction d'activation non-linéaire
- Les neurones de la couche  $i$  servent d'entrées aux neurones de la couche  $i + 1$
- Neurones d'entrées, neurones de sortie, neurones cachés
- Généralement : tous les neurones d'une couche sont connectés à tous les neurones des couches précédentes et suivantes
- Toutes les connections  $i \rightarrow j$  sont pondérées par le poids  $w_{ij}$

# Le Perceptron multi-couches: Apprendre avec un risque le plus faible

Cas d'un PMC à une couche cachée  $n$  entrées,  $p$  sorties

- Soit un exemple  $(x, y) \in S$ ,  $x = (x_1, \dots, x_n)$ ,  $y = (y_1, \dots, y_p)$ , la fonction de décision à la sortie  $k$  est :

$$h_k(x) = A_o(W_o A(\langle w, x \rangle))$$

- avec  $A$  non linéaire, non polynomiale, continue, dérivable, approximant la fonction de Heaviside :
  - $A(x) = x \tanh(x)$
  - $A(x) = \frac{1}{1+e^{-x}}$  fonction logistique (sigmoïde)
- et  $A_o$  continue dérivable, softmax en classification multi-classes

Quel que soit le nombre de couches

Erreur et risque : dépend de tous les  $w$

- Apprentissage avec **risque minimal** = minimiser  $\ell(h_k(x), y_k)$
- Fixons la fonction de risque  $\ell(h_k(x), y) = (h_k(x) - y)^2$ , continue dérivable si  $h_k(x), y \in \mathbb{R}$  (régression).



# Le Perceptron multi-couches: Apprendre avec un risque le plus faible

## Expression de la fonction à minimiser

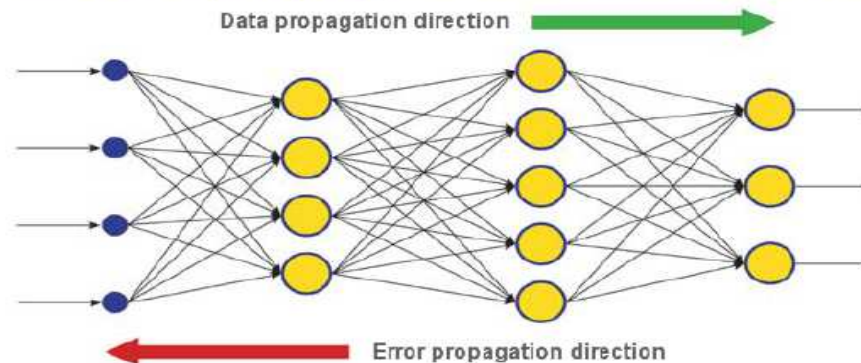
L'erreur (si fonction de perte = mean-square,  $h_k$  sortie du neurone  $k$ ) :

$$E(w) = \frac{1}{2} \sum_{(x,y) \in S} \sum_{k=1}^p (h_k - y_k)^2$$

avec  $h_k = A_0(w_{k0} + \sum_{j \in \text{Pred}(k)} (w_{kj} a_j))$

- Contribution de tous les neurones et de toutes les synapses
- Prise en compte de tous les exemples de l'échantillon

## Activation dans un sens, propagation de l'erreur dans l'autre



# Apprentissage PMC: Problème d'optimisation

Optimisation non-linéaire

Fonctions continues, dérivables

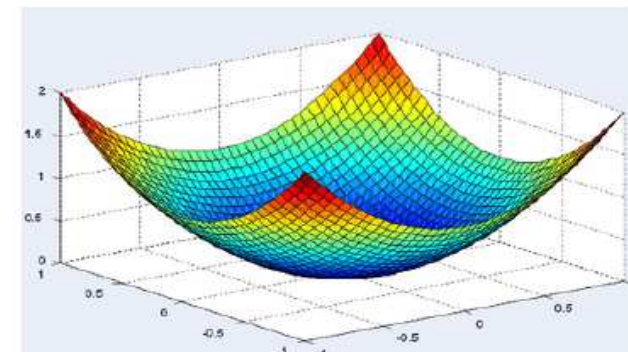
Similarité avec le perceptron

- Minimisation de l'erreur sur chaque présentation individuelle des exemples (règle de Widrow-Hoff)
- On doit **minimiser** :

$$E = E_{(x,y)}(w) = \frac{1}{2} \sum_{k=1}^p (h_k - y_k)^2$$

- Plus généralement  $\arg \min_{\theta} L(f_{\theta}, S) = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \ell(f_{\theta}(x^i), y^i)$ ,

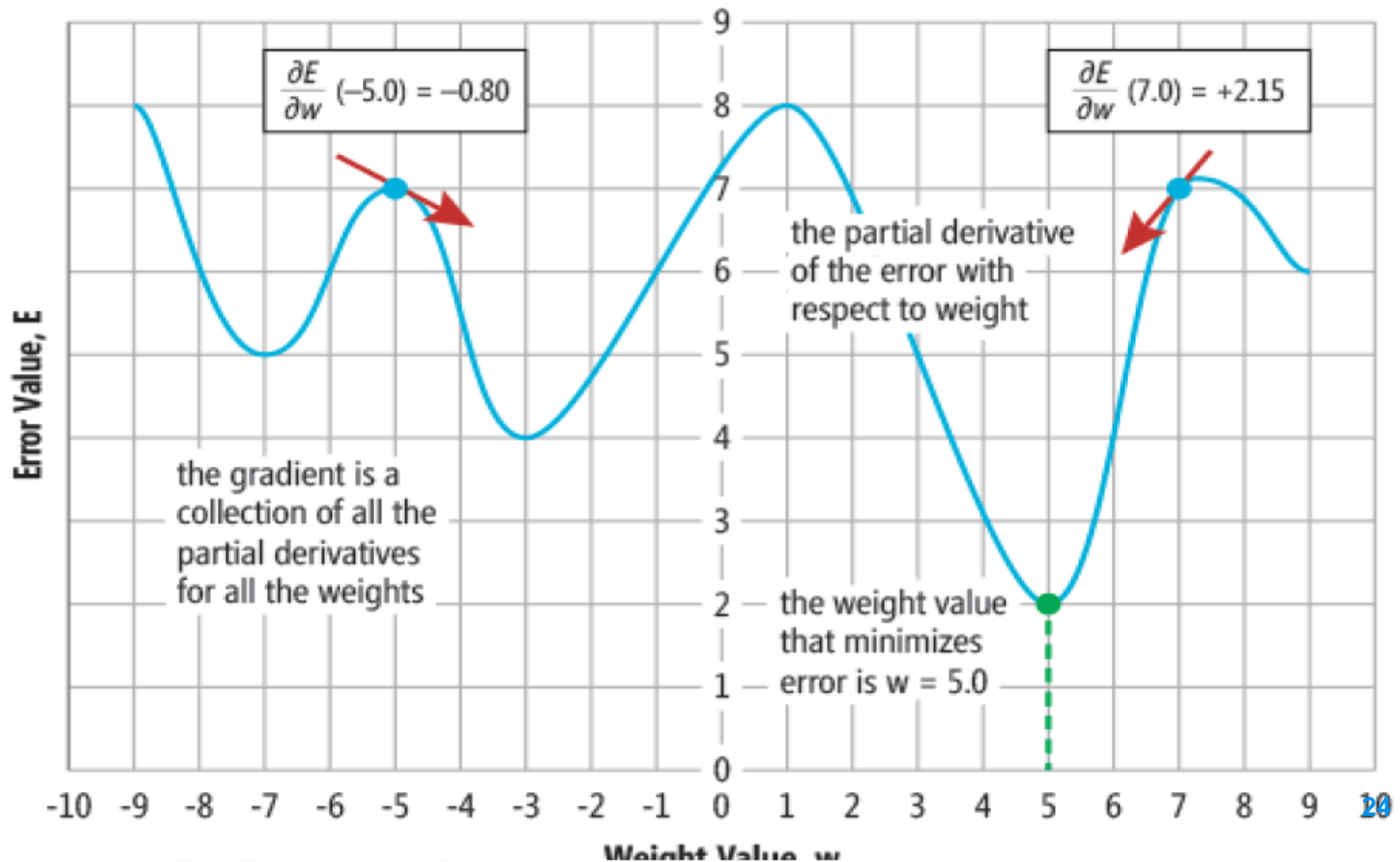
Minimiser une fonction : recherche descendante du point de dérivée nulle de la fonction



# Apprentissage PMC: Problème d'optimisation

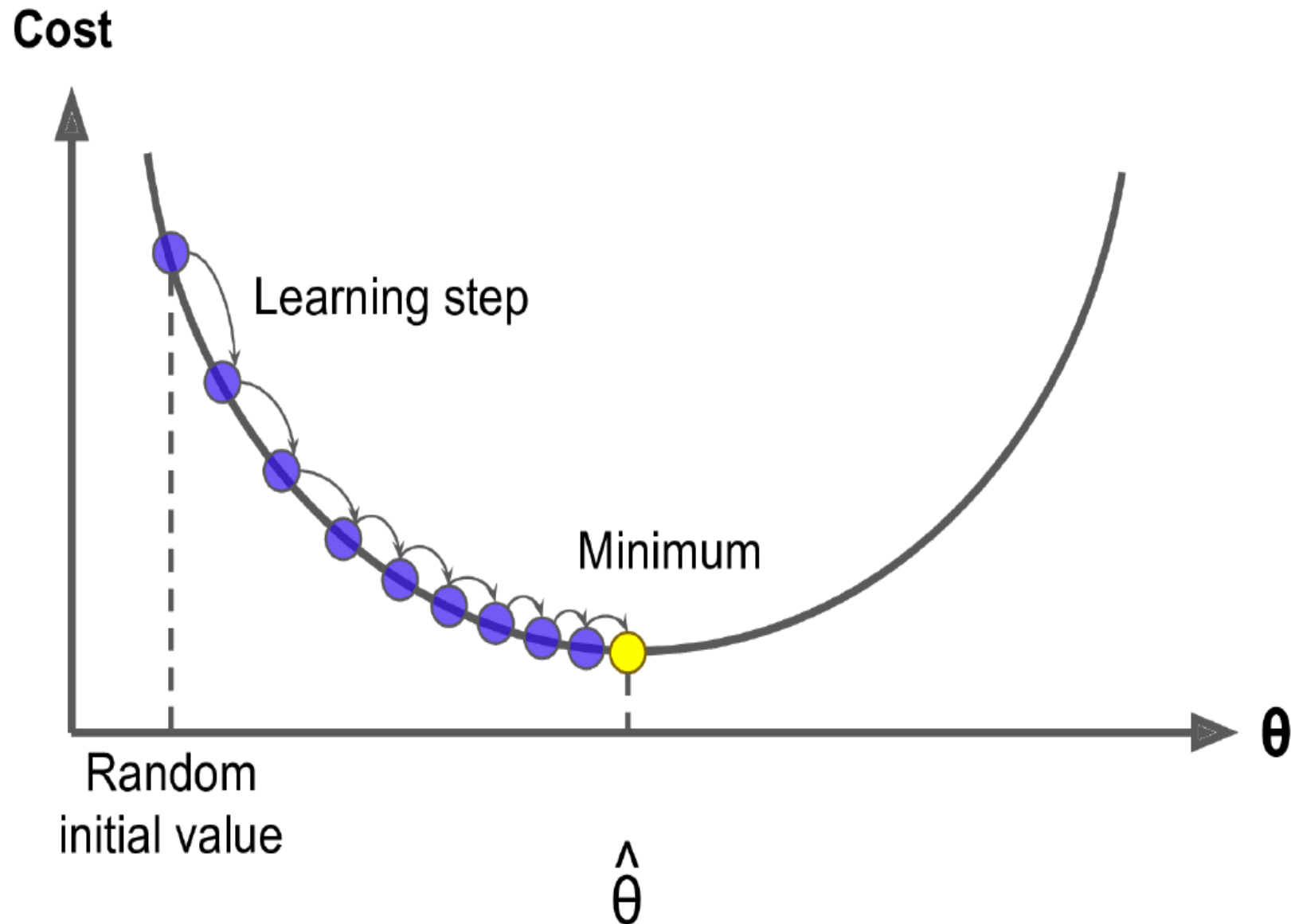
Trouver les  $w$  qui mènent à la plus petite erreur, si possible

Error Depends on Weight Value

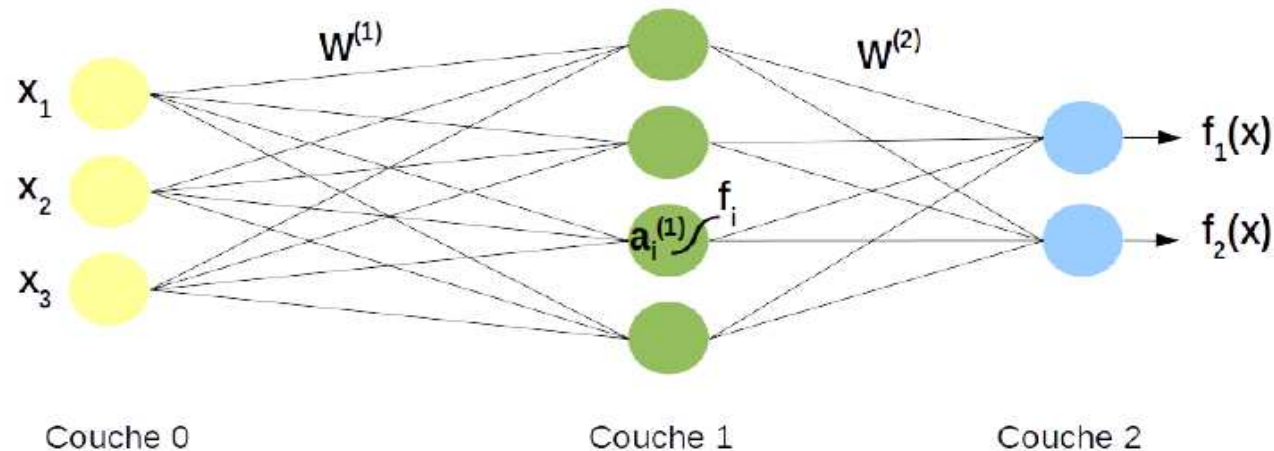




# Apprentissage PMC: Descente de gradient



# Apprentissage PMC: Algorithme dans le cas d'une seule couche cachée



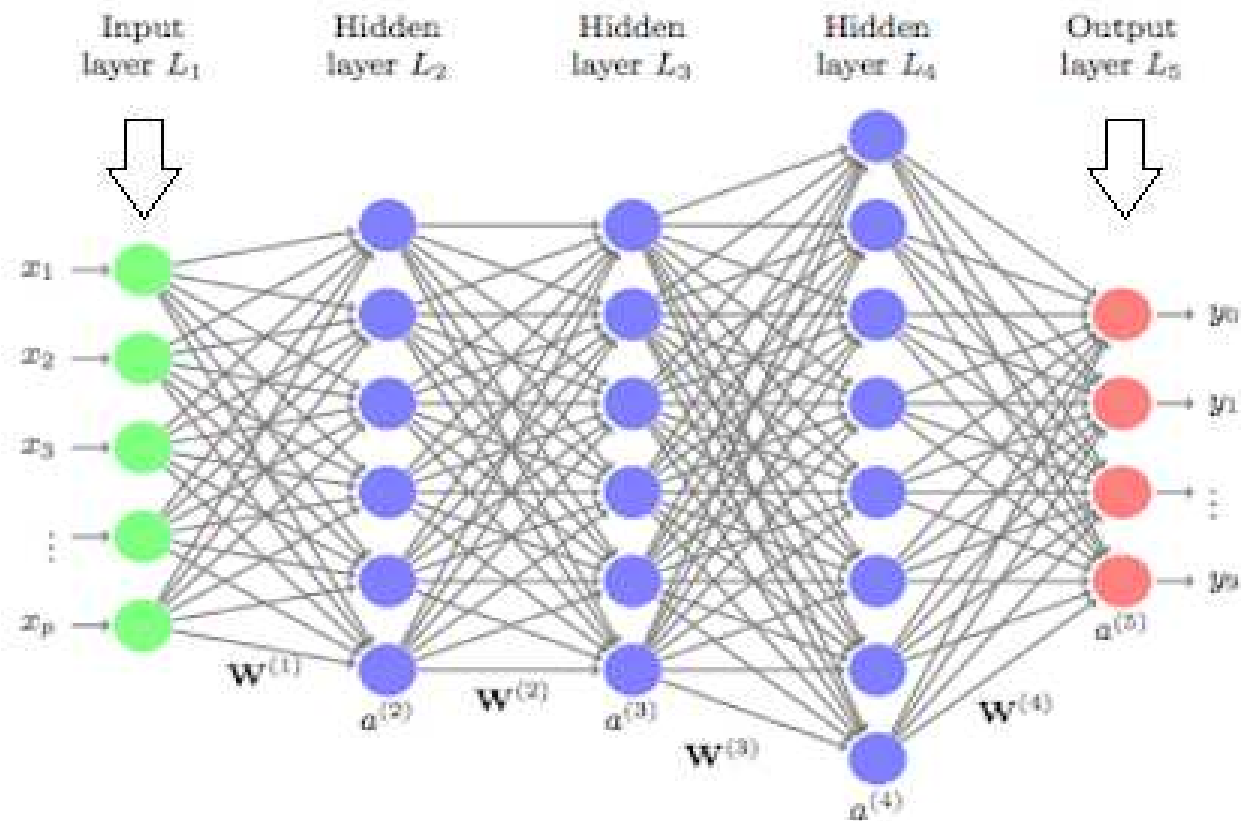
1. Initialisation de tous les poids du réseau (petites valeurs)
2. **Pour chaque exemple  $(x, y)$  de  $S$  faire :**
  - 2.1 prediction =  $f_k(x) = \text{activation-du-MLP}(x)$
  - 2.2 calculer l'erreur  $e_k = \ell(f_k - y_k)$  pour chaque sortie, et  $E = \sum_k e_k$
  - 2.3 calculer  $\Delta w_{jk}$  pour tous les poids entre la couche cachée et la couche de sortie
  - 2.4 calculer  $\Delta w_{ij}$  pour tous les poids entre la couche d'entrée et la couche cachée
  - 2.5 Mettre à jour tous les poids

**Jusqu'à satisfaction d'un critère d'arrêt**
3. Retourner le réseau

# Apprentissage PMC: Algorithme générique

1. initialiser le vecteur de paramètres  $w$
2. initialiser les paramètres - méthode de minimisation
3. **Répéter**
4.   **Pour tout**  $(x, y) \in D$  **faire**
5.     appliquer  $x$  au réseau et calculer la sortie correspondante
6.     calculer  $\frac{\partial \ell(f(x), y)}{\partial W_{ij}}$  pour toutes les pondérations
7.   **Fin pour**
8.   calculer  $\frac{\partial L(f, D)}{\partial W_{ij}}$  en sommant sur toutes les données d'entrée
9.   appliquer une mise à jour du vecteur de pondération - méthode de minimization
10. **Tant que** l'erreur n'a pas convergé

# Réseaux de neurones avec plusieurs couches cachées



# Réseaux de neurones récurrents

1. & 2. Vecteurs 1-hot + projection de  $w_i$  dans un espace continu

$$\rightarrow \mathbf{c}_i = \mathbf{W}^t w_i \in \mathbb{R}^d$$

avec  $\mathbf{W} \in \mathbb{R}^{\|V\| \times d}$  et  $d \ll \|V\|$

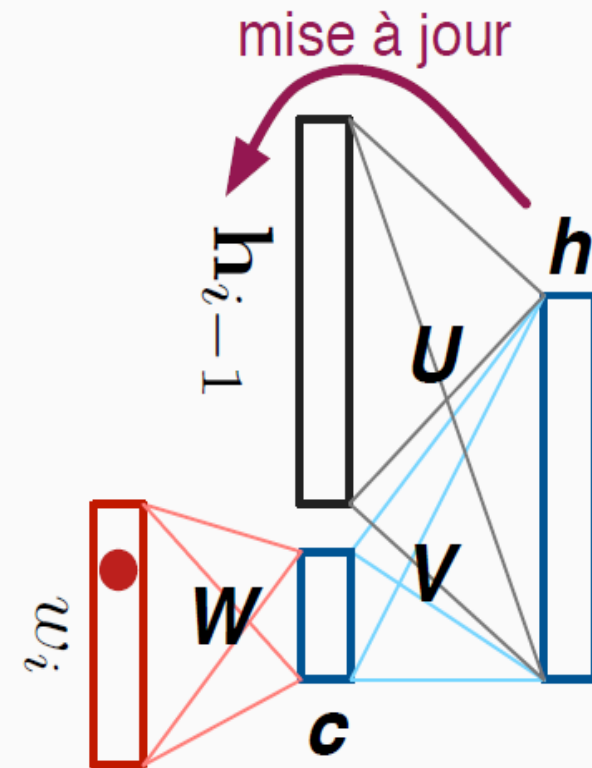
$\|V\|$ : taille du vocab

3. Mise à jour de l'état interne

$$\rightarrow \mathbf{h}_i = \phi(\mathbf{V}^t \mathbf{c}_i + \mathbf{U}^t \mathbf{h}_{i-1})$$

avec  $\mathbf{U} \in \mathbb{R}^{\|\mathbf{h}\| \times \|\mathbf{h}\|}$  et  $\mathbf{V} \in \mathbb{R}^{d \times \|\mathbf{h}\|}$

$\|\mathbf{h}\|$ : taille de la couche cachée



# Réseaux de neurones récurrents

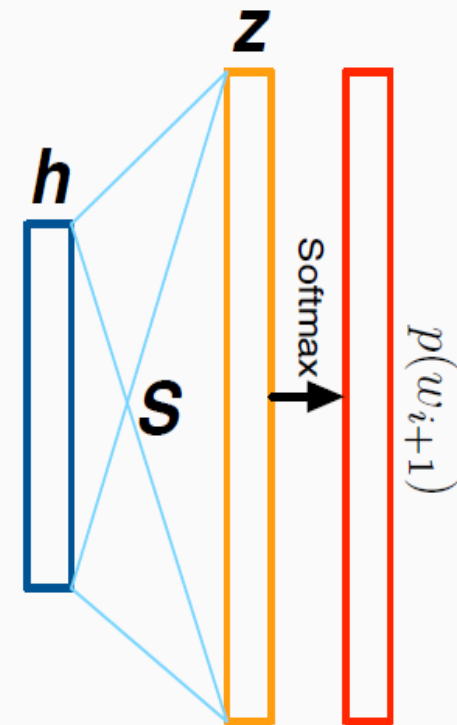
4 . Calcul du score non normalisé

→  $\mathbf{z} = \mathbf{S}^t \mathbf{h} + \mathbf{b}_s$  avec  $\mathbf{b}_s$  le biais

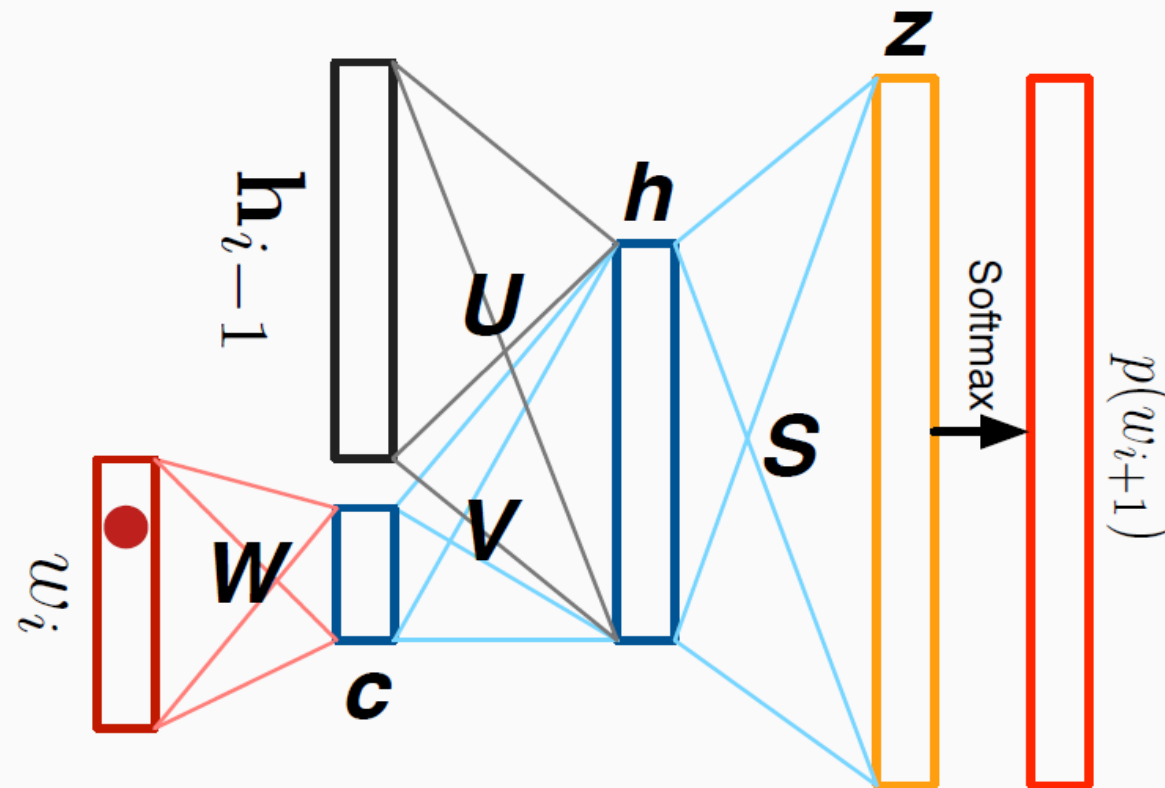
5 . Calcul de la probabilité par softmax

$$\rightarrow p(w_{i+1} = j | \dots) = \frac{e^{z_j}}{\sum_{k=1}^{\|V\|} e^{z_k}}$$

avec  $z_j$  le  $j^{\text{ème}}$  élément de  $\mathbf{z}$



# Réseaux de neurones récurrents



→ séquence de symboles d'entrée compressée dans un vecteur de taille fixe à l'aide d'une récursion

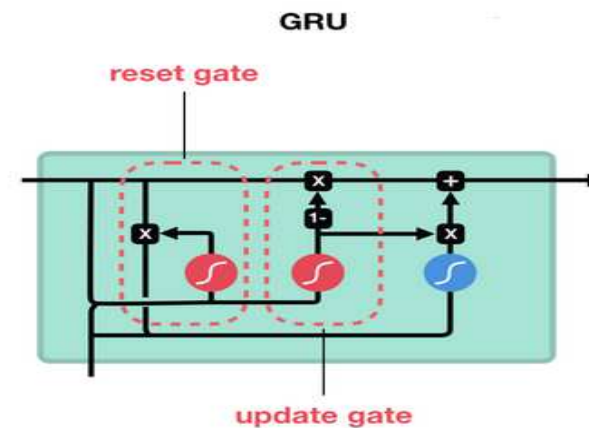
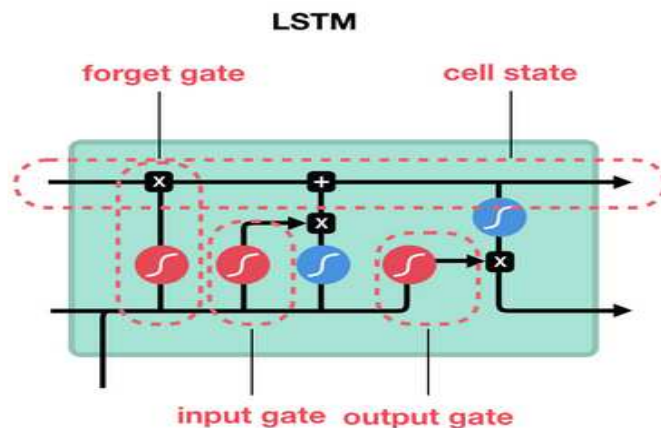


# Réseaux de neurones récurrents LSTM-GRU

**LSTM** (*Long Short-Term Memory*), est une cellule composée de trois « portes » : ce sont des zones de calculs qui régulent le flot d'informations en réalisant des actions spécifiques. Il y a également deux types de sorties (états):

- Forget gate (porte d'oubli)
- Input gate (porte d'entrée)
- Output gate (porte de sortie)
- Hidden state (état caché)
- Cell state (état de la cellule)

Les actions spécifiques sont des opérations dans les portes permettent au LSTM de **conserver** ou **supprimer** des informations qu'il a en mémoire.



sigmoid



tanh



pointwise  
multiplication



pointwise  
addition



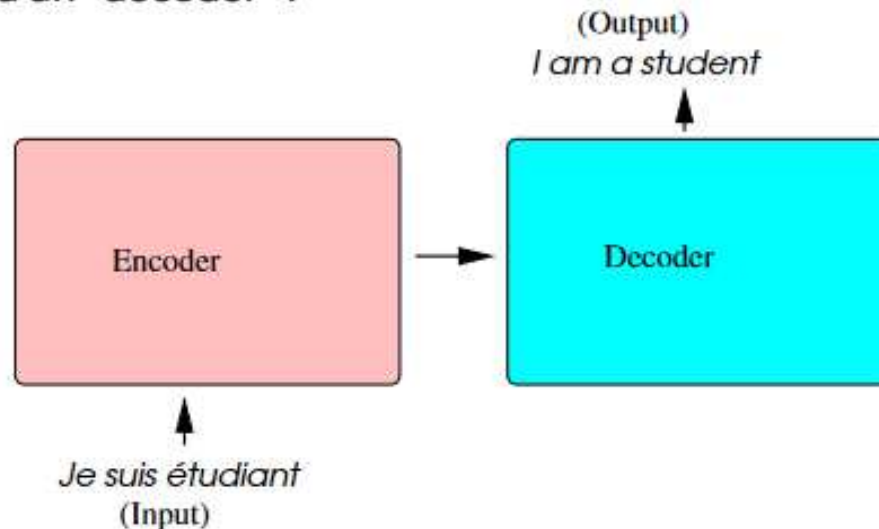
vector  
concatenation



# Réseaux de neurones Transformers

Les réseaux de neurones transformers ont pour but de prévoir une séquence de longueur variable en fonction d'une autre séquence de longueur variable. Le principe est de tenir compte du contexte des observations à prévoir (concept d'attention) :

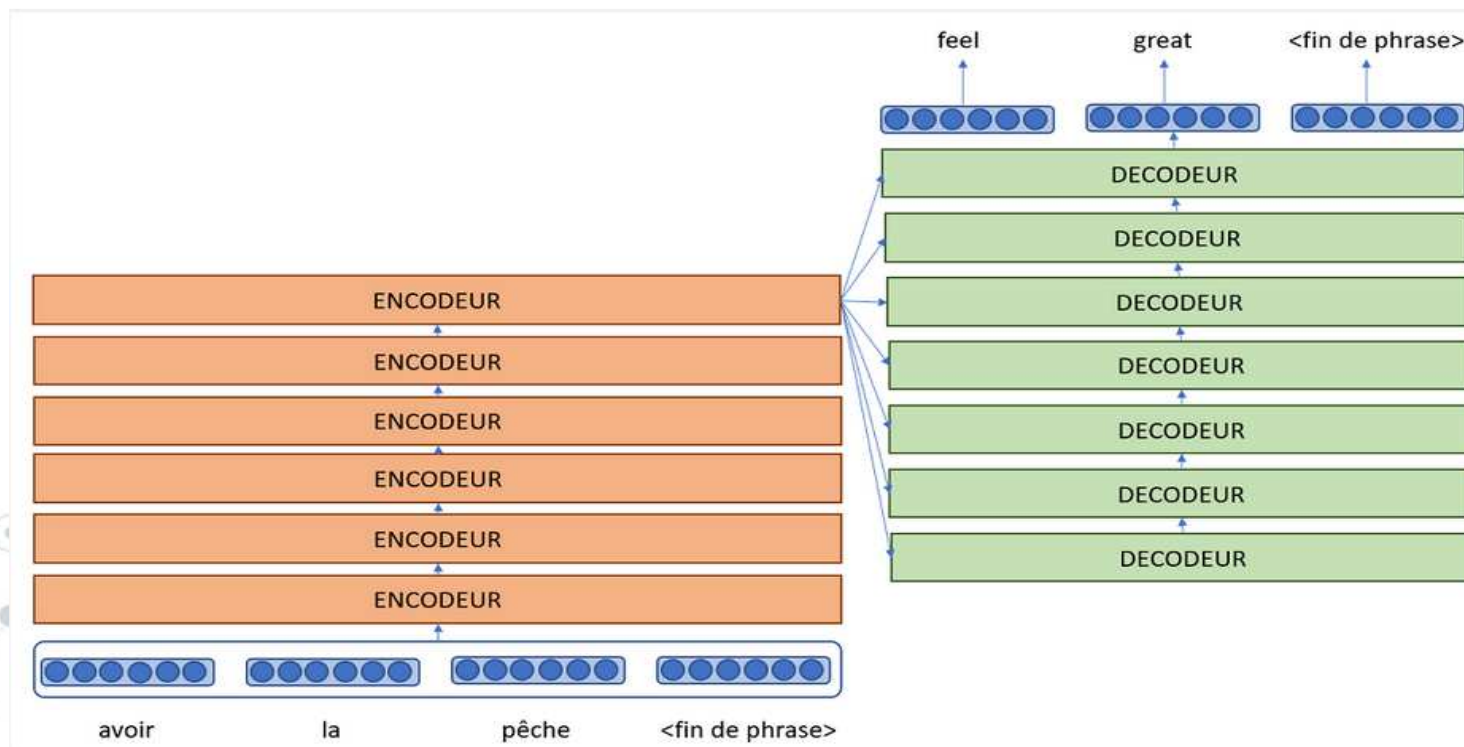
- Notons :
  - $(X_1, \dots, X_{T_X})$ , la séquence explicative.
  - $(Y_1, \dots, Y_{T_Y})$  la séquence à prévoir. Remarquons que  $T_Y$  est aussi à prévoir.
  - $\theta$  le vecteur paramètre du modèle.
- Ce formalisme est adapté aux "Chatbot" ou bien aux systèmes de traduction automatique.
- En général, l'architecture de ce réseau de neurones est composée d'un "encoder" et d'un "decoder" :



# Utilisation des Transformers en Traduction

Un Transformer est composé de:

- Six encodeurs empilés (le Nx du schéma), chaque encodeur prenant en entrée la sortie de l'encodeur précédent
- Chaque encodeur se compose de deux sous-couches : une couche d'auto-attention "multi-têtes", suivie d'un FFN complètement connecté
- Chaque décodeur se compose de trois couches : une couche d'auto-attention "multi-têtes", suivie d'une couche d'attention avec le dernier encodeur, puis d'un FFN complètement connecté
- Trois mécanismes d'attention
- Pour la génération de texte ou la traduction simultanée le **mécanisme est auto-régressif**, c'est à dire qu'on fait entrer une séquence dans le premier encodeur, la sortie prédit un élément, puis on repasse toute la séquence dans l'encodeur



# Architecture du Transformer

