Cours de Traitement Automatique du Langage

Chapitre 02 : Word Embeddings

DI5 – 2021-22

Polytech ’Tours J.Y. RAMEL

Introduction

pomme = $\begin{bmatrix} 15 \\ 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{bmatrix}$

Idée principale du Word Embeddings

- Apprentissage de « représentations contextuelles » des mots
- Un vecteur qui représente le lien sémantique entre les mots

  1 mot (ou phrase) → 1 vecteur (dense)

- Hypothèse : « distributional semantics » → 2 mots se trouvant dans un même contexte ont une proximité sémantique et syntaxique

Contextual information is sufficient to obtain a viable representation of words

- "For a large class of cases [...] the meaning of a word is its use in the language." Wittgenstein (Philosophical Investigations, 43 - 1953)
- "You shall know a word by the company it keeps", Firth ("A synopsis of linguistic theory 1930-1955." - 1957)

Exemple :

- contexte: Mon fruit préféré est une POMME - contexte: Mon fruit préféré est une ORANGE

  ➔ $vect_{ORANGE} \approx vect_{POMME}$

- Grande similarité sémantique entre ORANGE et POMME (tous deux sont des fruits)
- Ils ont tendance à apparaître dans un contexte similaire dans les textes.

Soustraction vectorielle dans l'espace de projection
- proximité syntaxique : $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$
- proximité sémantique : $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$ / $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

Word embeddings

Idées sur la mise en œuvre ?

d^2(orange, pomme) << d^2(orange, chat)

$$\text{pomme} = \begin{bmatrix} 15 \\ 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{bmatrix}$$

Analyse des co-occurences

Matrice de co-occurences

- Cooccurrence-Matrix(i ; j) = nombre d'occurrences du moti à côté de motj
- Définition d'une fenêtre de taille n (5-10 mots plutôt que par phrase)
- Matrice C symétrique N x N où N est la taille du vocabulaire !
- Capte à la fois les informations syntaxiques et sémantiques
- 1 embedding = 1 colonne $\in R^N$
- Augmente en taille avec le vocabulaire
  - nécessite beaucoup de stockage
  - sparsité → faible robustesse

Exercice : Construire la matrice de co

- I like deep learning.
- I like NLP.
- I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Analyse des co-occurences

Matrice de co-occurences

- On veut un vecteur dense (de dimension assez faible entre 25-1000)
- Réduction de la dimensionnalité possible en utilisant la décomposition en valeurs singulières (SVD) de C (k valeurs / vecteurs propres - cf ACP)

Problèmes

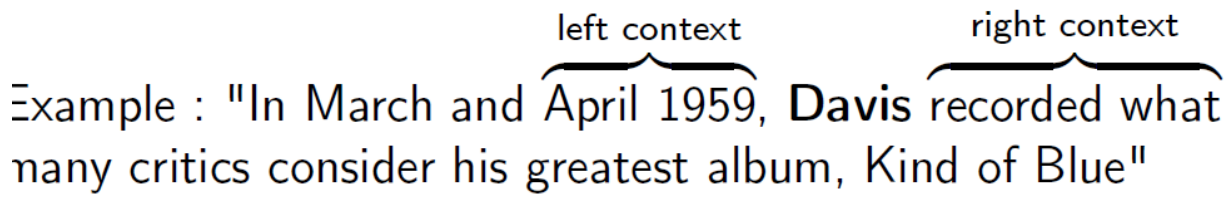
- Coût de calcul
- Sparsité partiellement résolue
- Difficile d'intégrer de nouveaux mots

Nouvelles méthodes

- Apprendre à prédire les mots voisins plutôt que de compter des cooccurrences
- Apprendre directement des vecteurs de faible dimension
- Utilisation de Réseaux de Neurones de type MLP

Word2Vec

Définitions



- **Contexte** d'un mot

$$\{April, 1959, recorded, what\}$$

  - Observation : l'ordre des mots dans le contexte n'a pas d'influence sur le calcul de l'embeddings

- **Mot cible (target)**
  - **DAVIS** est le mot cible dont on veut calculer le vecteur

Word2Vec

Combinaison de 2 approches

- CBOW (Continuous Bag Of Words) → prédire la target en fonction du contexte
- Skip-gram → prédire le contexte en fonction de la target

Example : "In March and April 1959, **Davis** recorded what many critics consider his greatest album, Kind of Blue"

(left context: "In March and April 1959," ; right context: "recorded what many critics consider his greatest album, Kind of Blue")

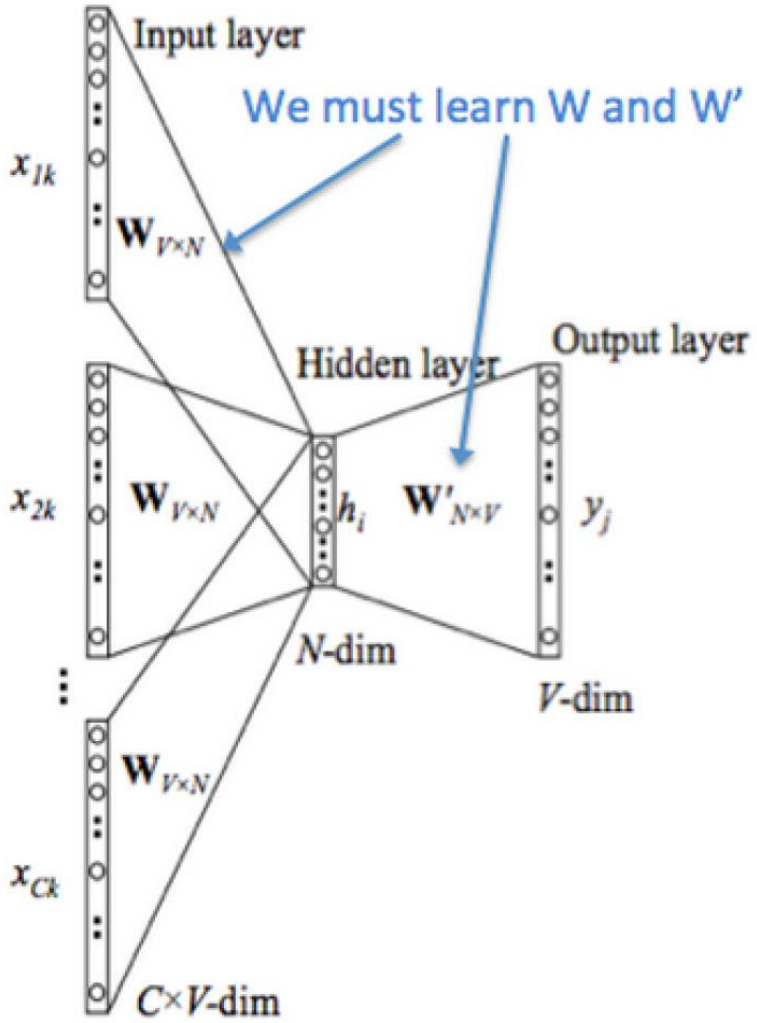
Basées sur une architecture MLP

Word2Vec

Modèle CBOW (Continuous Bag Of Word)

- MLP à 2 couches + sortie
- Apprendre W et W' pour obtenir :
  - Input = {April, 1959, recorded, what}
  - Output = {Davis}
  - Les mots sont représentés par des vecteur "one-hot" → x_i
  - $x_i \in R^v$ avec V = taille du vocabulaire

$$at \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

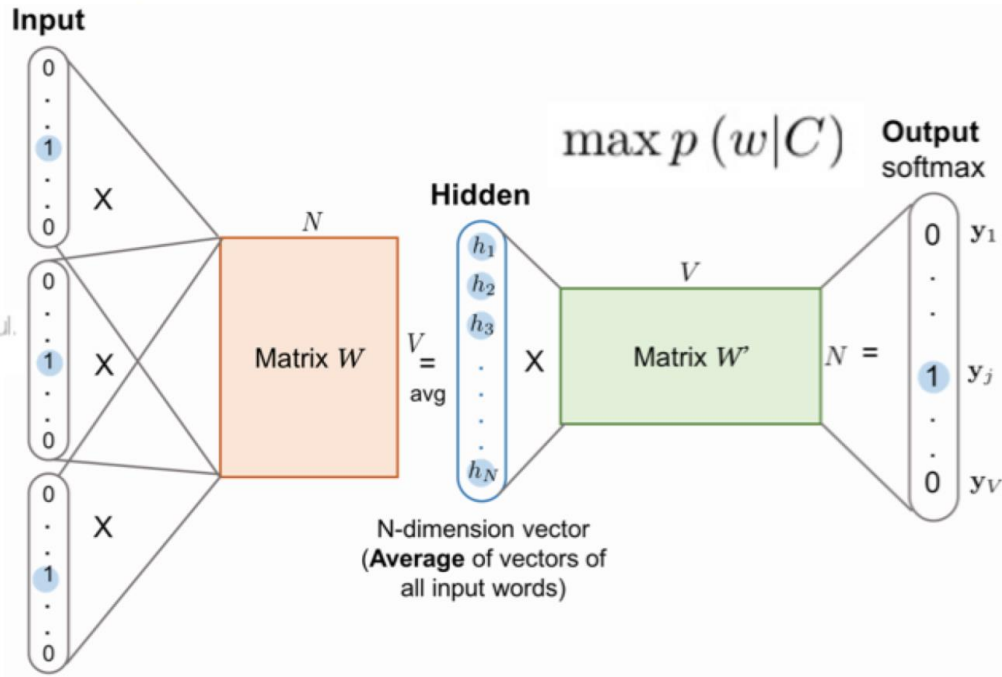


Word2Vec

Modèle CBOW (Continuous Bag Of Word)

Apprentissage

- Entrée : les one-hot vecteurs des mots du contexte → (x_{c-m};…; x_{c-1}; x_{c+1};…; x_{c+m})
- *V = taille du vocabulaire*
- Sorties souhaitées du réseau : ^y = un one-hot vecteur correspondant au mot cible
- Couche cachée :
  - h = moi ($W.X_c$) - Taille N à définir
  - fonction d'activation = identité + Moyenne des Wj correspondant aux "mots contextes"
  - *h = ^v = moyenne des embedding d'entree (de taille 1xN)*
  - $W \in R^{VxN}$ et $W' \in R^{NxV}$
  - Les matrices W et W' correspondent à des embeddings en entrée et en sortie
    - En entrée → la $j^{ème}$ colonne de W est le vecteur d'embeddings du mot *dj*
    - En sortie → la $i^{-ème}$ ligne de W' est le vecteur d'embedding pour le mot *di (target)*
    - Nous apprenons en fait deux vecteurs pour chaque mot.



Word2Vec

Modèle CBOW (Continuous Bag Of Word)

Apprentissage

- Objectif d'apprentissage : mettre à jour W et W' afin de minimiser la *Loss*

$$H(\hat{y}, y) = -\sum_{j=1}^{\|V\|} y_j log(\hat{y}_j)$$

$H(\hat{y}, y) = -log(\hat{y}_c)$ (the probability of target word to appear with the context words)

$\hat{y}_c = softmax(W'_c * \hat{v})$,

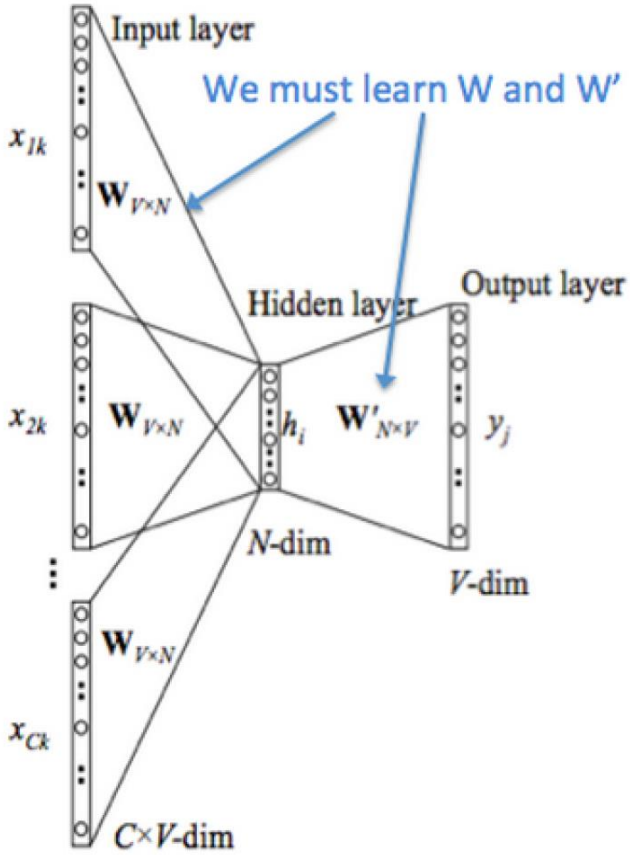
$H(\hat{y}, y) = -log(softmax(W'_c * \hat{v})) = -log(\frac{exp(W'_c * \hat{v})}{\sum_{j=1}^{|V|} e^{W'_j * \hat{v}}})$

$H(\hat{y}, y) = -W'_c * \hat{v} + log(\sum_{j=1}^{|V|} e^{W'_j * \hat{v}})$

where :

$\hat{v} = mean(W * x^{c-m}, ..., W * x^{c-1}, W * x^{c+1}, ..., W * x^{c+m})$

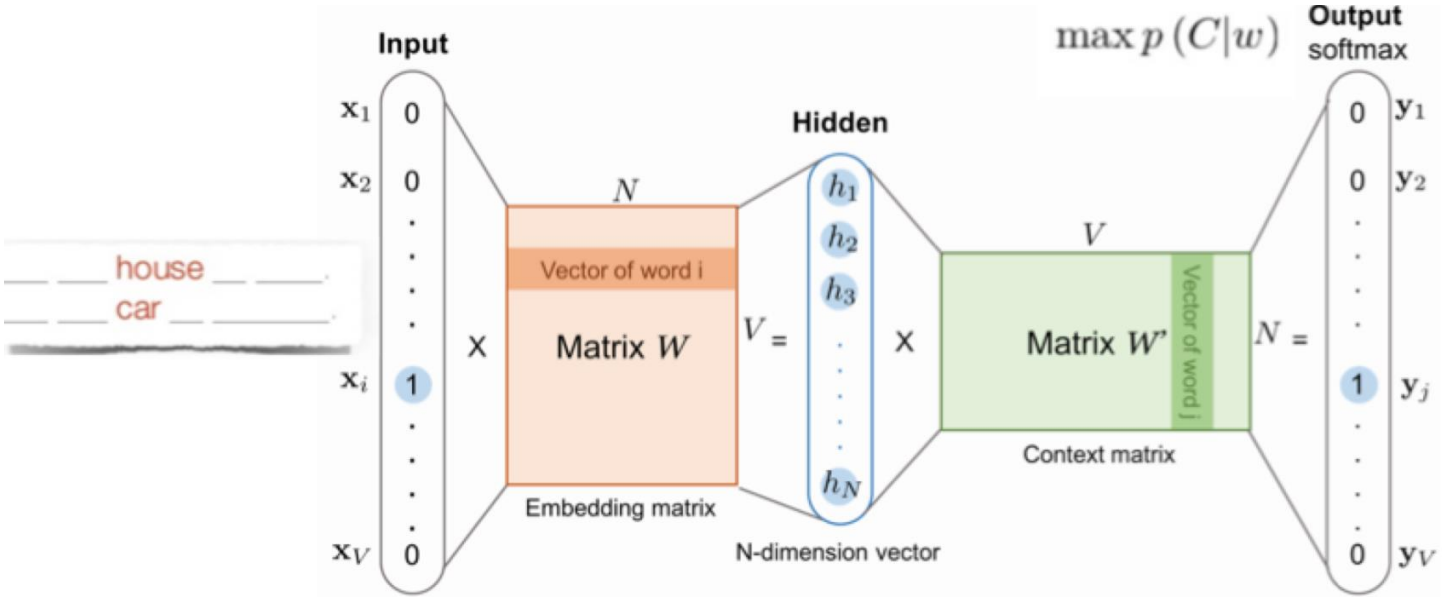
and W'_j is the j^{th} line of W'



Word2vec

Modèle Skip-Gram

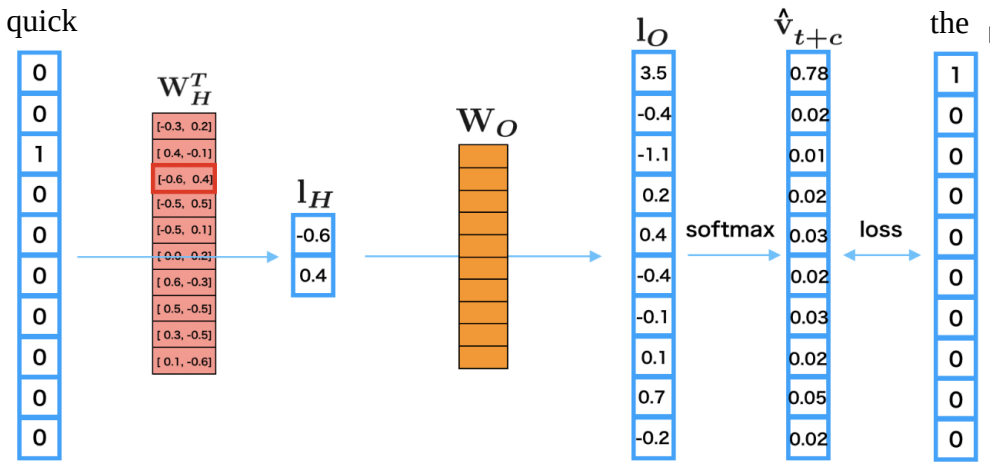
- A partir d'un mot X (one-hot dfe taille |V|), inférer son contexte C
- Projection de X dans l'espace latent h →sélection jème colonne de matrice W de taille N
- Réinjecter h dans l'espace |V| → matrice W' + softmax()
- Fonction de Loss → Moyenne des sorties pour des mots contexte choisis



Word2vec

Modèle Skip-Gram - Apprentissage

- Dataset : "The quick brown fox jumped over the lazy dog"
- La tache est de prédire selon les mots targets (avec m =1) :
  - ’the’ et ’brown’ pour ’quick’ - ’quick’ et ’fox’ pour ’brown’ - etc…
- Procédure
  - Calcul du vecteur embedding pour le mot central (quick) → $v_c = W.x.$
  - Pas de calcul de moyenne (car une seule entrée), il suffit de définir $\hat{v} = v_c$
  - Génération du vecteur de score en utilisant $u = W'.v_c$
  - Calcul de la loss en prenant
    - un seul mot context choisi au hasard (brown ou the)
    - ou pour les 2m mot context (2m)
      - Transformer chacun des scores en probabilités, $\hat{y} = softmax(u)$
      - Calcul de la Loss globale en tenant compte des 2m probablités (sum ou moyenne).



Word embeddings

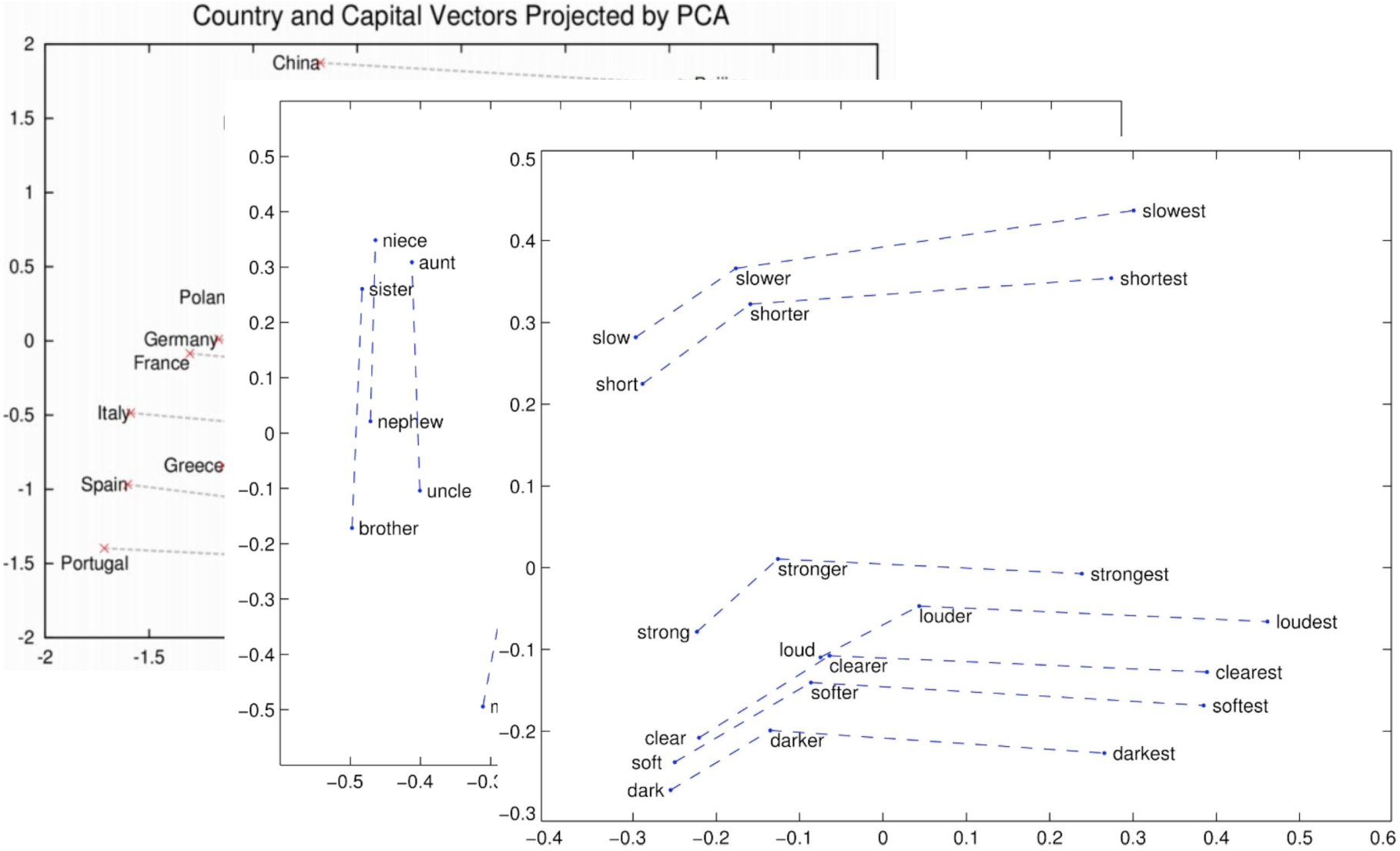
- Word2Vec
  - CBoW fonctionne bien pour les mots fréquents
  - Skipgram pour les mots rares

- Approche «pseudo-supervisée » :
  - peut être entraîné sur un énorme corpus généraliste.
  - Transfert et réglage fin possibles sur des tâches supervisées spécifiques.

- Le Word Embedding fonctionne bien et est donc très utilisé
  - Transformation d'un espace discret en un espace continu et dense
  - Possibilité de réaliser des recherches d'analogie (par requetés algébriques) → cf plus loin

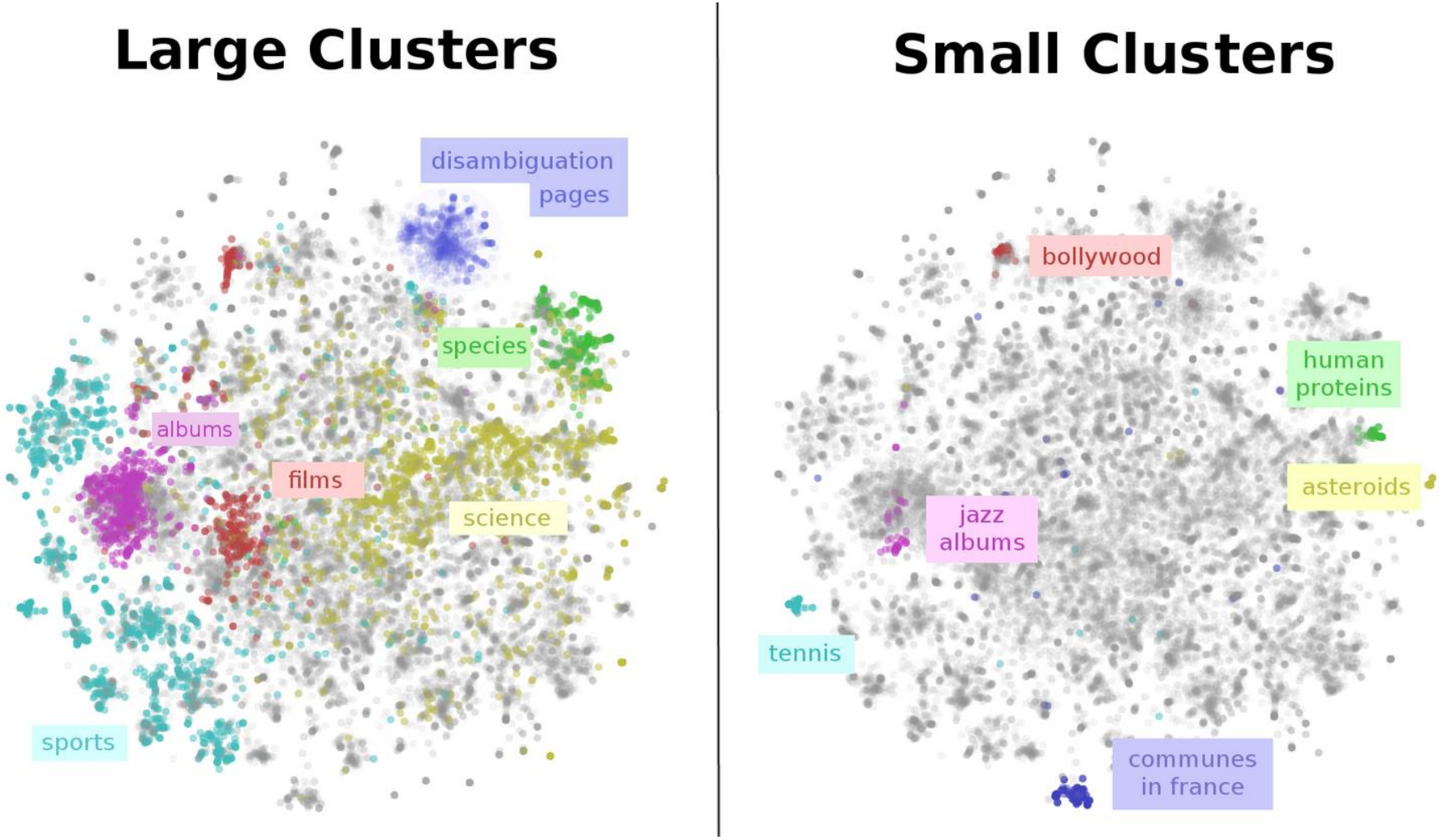
- Nombreuses extensions possibles
  - GloVe: Global Vectors for Word Representation [Pennington et al., 2014] → Basé sur ma Matrice des co-occurrence→ probabilité de co-occurrence → prise en compte d'un contexte plus global
  - Skip-ThoughtVecteurs [Kiros et al., 2015] → prédire les phrases environnantes d'une phrase donnée
  - BERT (Transformer) [Devlin et al., 2018]: Bidirectional Encoder Representations from Transformers

Avec Word2vec et GloVe



Avec Word2vec et GloVe

- Clustering de mots → Visualisation de l'espace "latent" h (t-SNE)



ANNEXE

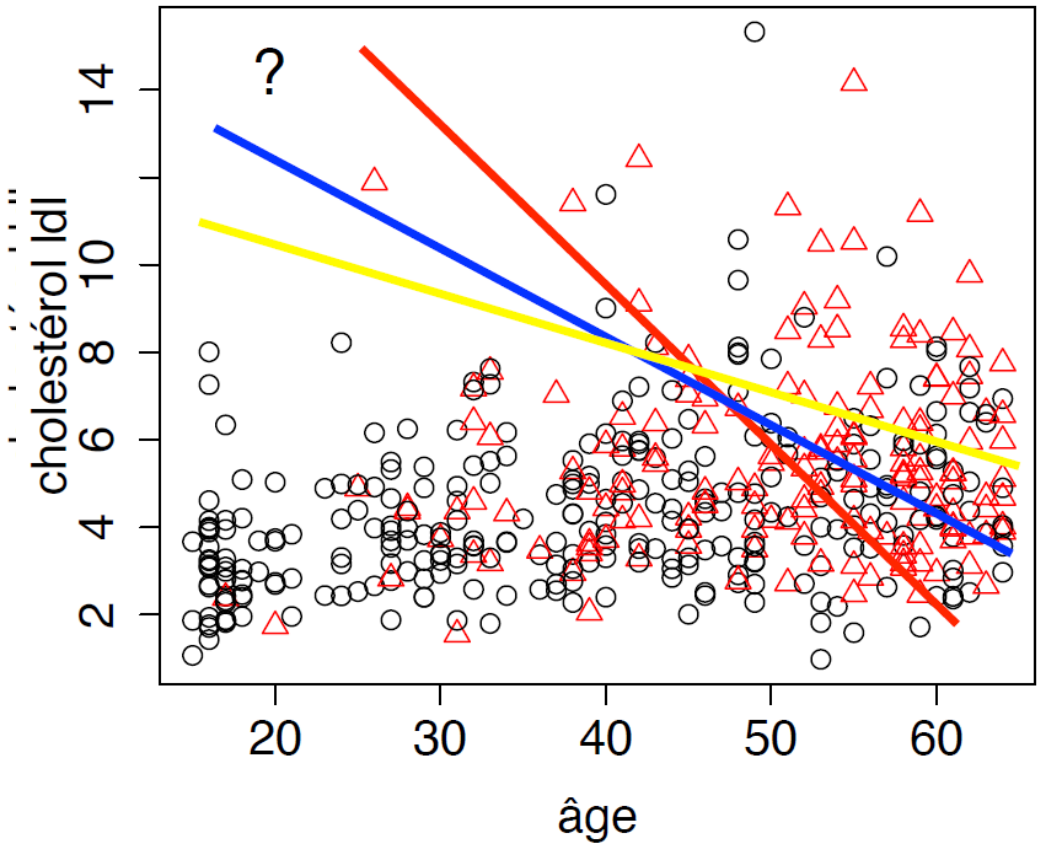
Rappels RN

Approche de type ML ➔ Premier modèle : Régression logistique

Exemple : Prédiction Infarctus (1 classe) selon âge (x_1) et cholestérol (x_2)

Préparation des données

- Constituer une base d'apprentissage
- Comment distinguer / séparer au mieux les deux classes ?
- Trouver la frontière
- Modèle linéaire :
  la droite séparatrice optimale ?



Rappels RN

Approche de type ML ➔ Premier modèle : Régression logistique

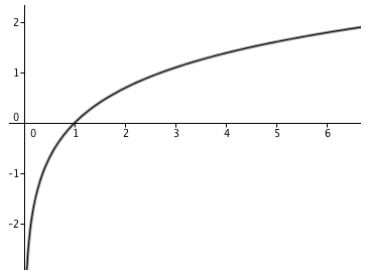
- On ne peut pas prédire à coup sûr l'occurence d'un IM à partir de l'âge et du taux de cholestérol, mais on peut chercher à estimer sa probabilité

  notée p(x) = $\mathbb{P}(\underbrace{\text{IM}}_{y=1} \mid \underbrace{\text{âge, ldl}}_{\mathbf{x}})$

- Modèle linéaire, on pose : $\ln \frac{p(\mathbf{x})}{1-p(\mathbf{x})} = w_0 + w_1 \times \text{âge} + w_2 \times \text{ldl}$

- Formulation équivalente : $p(\mathbf{x}) = \frac{1}{1+\exp(-w_0 - w_1 \times \text{âge} - w_2 \times \text{ldl})}$

- Problème : comment déterminer les coefficients w0, w1 et w2 ?
  - Si y = 1 (IM avéré), on veut avoir p(x) aussi grand que possible.
    On définit l'erreur dans ce cas par **-** *ln (p(x))* ➔ erreur grande qd p(x) est proche de 0
  - Symétriquement, si y = 0 (pas d'IM), on veut avoir p(x) aussi petit que possible.
    L'erreur est alors **-** *ln (1 - p(x))* ➔ erreur grande qd p(x) est proche de 1
  - Formule générale : $\text{erreur} = -y \ln p(\mathbf{x}) - (1-y) \ln (1 - p(\mathbf{x}))$
  - Erreur totale pour un ensemble d'apprentissage {(x1; y1), … ,(xn; yn) } mesurée par l'**entropie-croisée** : $C(\underbrace{w_0, w_1, w_2}_{\mathbf{w}}) = \sum_{i=1}^{n} \text{erreur}_i$

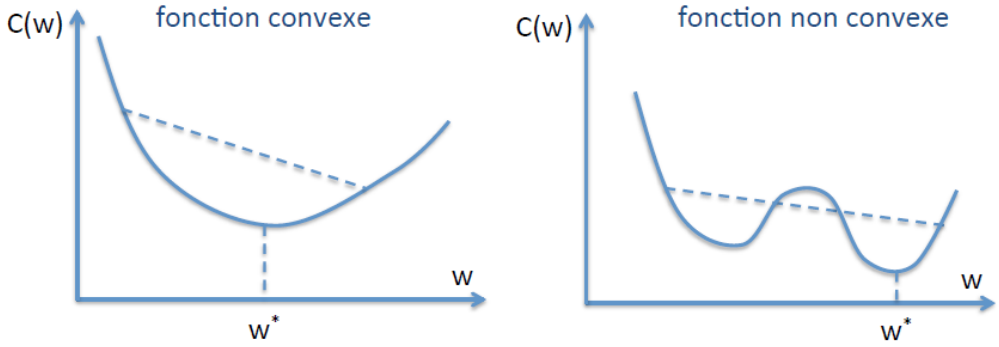


Rappel RN

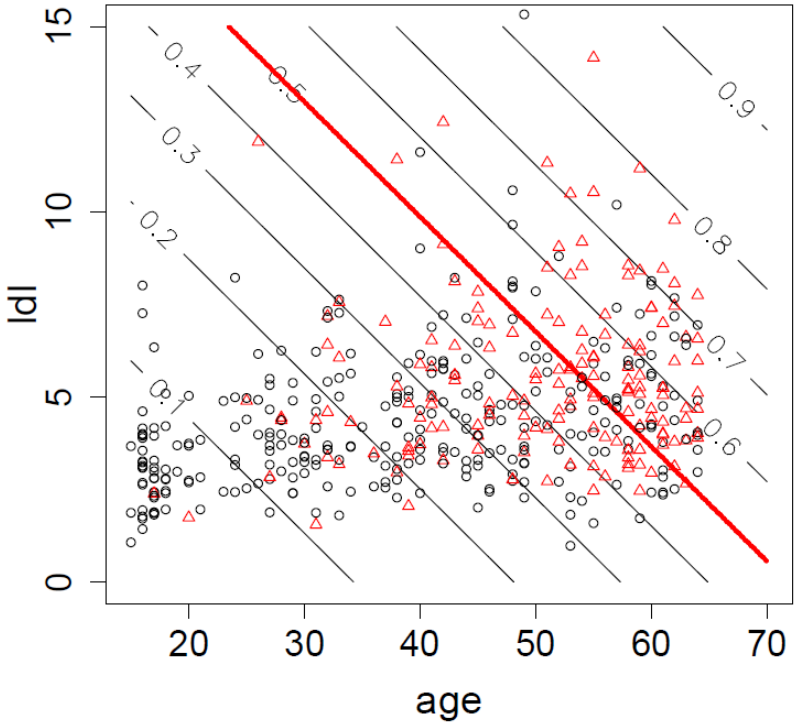
Approche de type ML ➔ Premier modèle : Régression logistique

Apprentissage

- Une fois définie une fonction d'erreur, le problème de l'apprentissage devient un problème d'optimisation : rechercher le vecteur de coefficient w* qui minimise l'erreur.
- Dans le cas de la régression logistique, ce vecteur est unique car la fonction d'erreur est convexe.



- Le vecteur solution w* peut être obtenu par un algorithme itératif.



Approche par apprentissage

Approche de type ML ➔ Premier modèle : Régression logistique

Exploitation

- Une fois déterminé le vecteur de coefficient w optimal, on dispose d'un **programme classifieur permettant de classer de nouveaux individus**.

Evaluation des performances

- Pour estimer la probabilité d'erreur du classifieur, il faut disposer d'un ensemble de test indépendant (**base de test**).
- On construit alors la **matrice de confusion** pour cet ensemble
- Avec 100 exemples

		Vraie classe	
		Positif	Négatif
Prediction	Positif	14	10
	Négatif	21	55

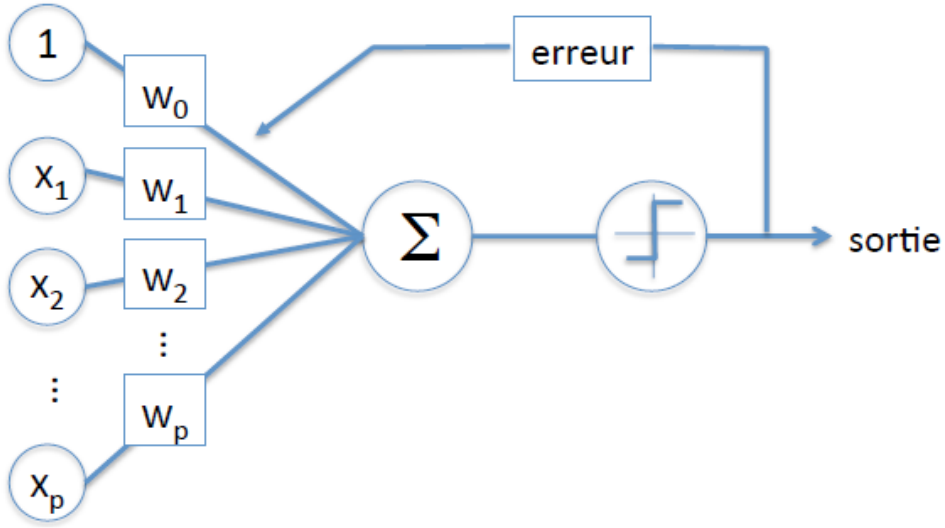
- Taux d'erreur = (10+21)/100=31%.

Approche par apprentissage

De la régression logistique aux réseaux de neurones

Le Perceptron [Rosenfeld1957]

- Idée : une algorithme qui apprend les poids pour résoudre des problèmes de classification binaire.



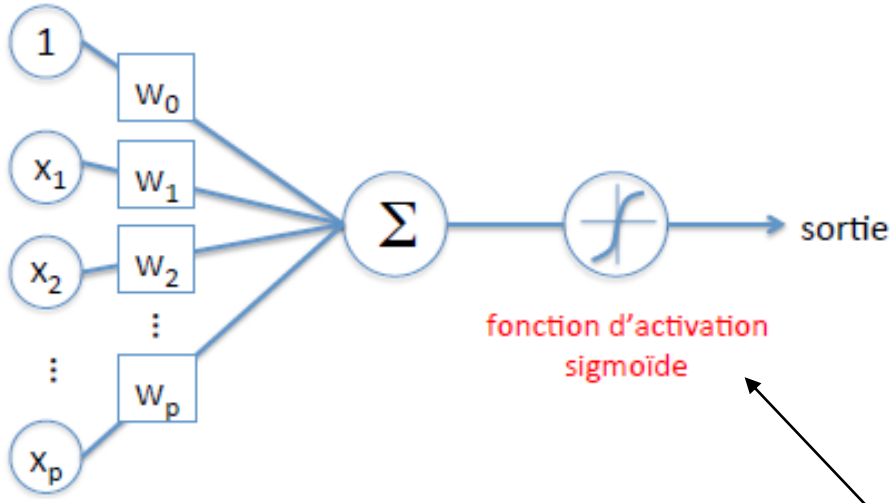
Limites :

- L'algorithme ne converge que si les deux classes sont bien séparées
- Difficilement généralisable à plus de deux classes

Approche par apprentissage

De la régression logistique aux réseaux de neurones

Version moderne du perceptron



Sortie :

$$g(\mathbf{x}) = \frac{1}{1 + \exp\left[-(w_0 + w_1 x_1 + \ldots w_p x_p)\right]}$$

Sigmoid pour avoir une valeur entre [-1;+1] ou [0;1] qd x > 0

Apprentissage des poids par minimisation de l'entropie croisée

C'est exactement le modèle de la régression logistique !

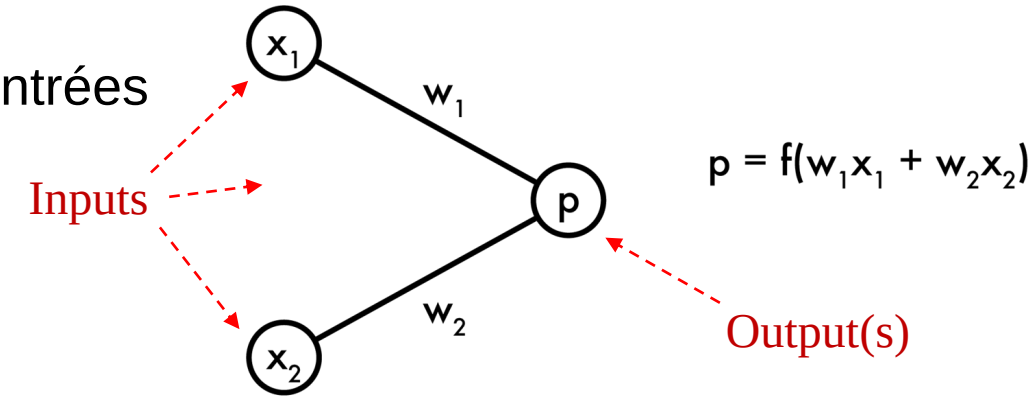
Outrepasser le problème de "feature engineering”

Le choix des caracteriistiques → de l’espace de représentation

- Il s’agit ici de la limitation principale du machine learning (surtout en pratique)
- ML fonctionne bien si il existe une relation claire entre les entrées du système (espace de représentation: x_i) et les sorties désirées
- La relation *p* est la fonction, le modéle que l’on cherche à apprendre

Avec les modèles linéaires

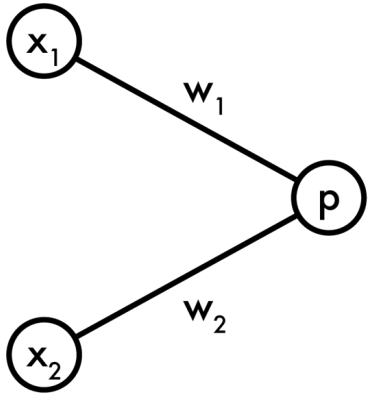
- Calcul d’une probabilité d’appartenance pour chaque classe à partir des valeurs prises par les feature choisies
- Sortie = combinaison linéaire des entrées
- Apprentissage des poids



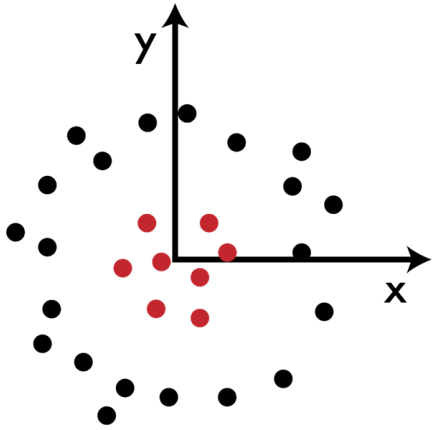
Outrepasser le problème de "feature engineering”

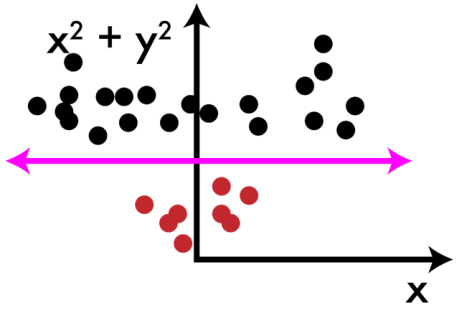
Avec d’autres modèles

- Même avec des modèles plus sophistiqués, on ne fait que proposer des combinaisons des caractéristiques choisies
- Impossible d'apprendre les transformations des caractéristiques
- L'homme doit choisir, créer de bonnes caractéristiques



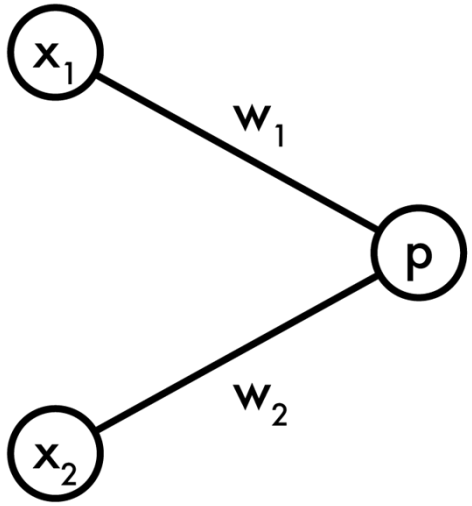
$$p = f(w_1x_1 + w_2x_2)$$





Outrepasser le problème de "feature engineering”

Et si on ajoutait des transformations?

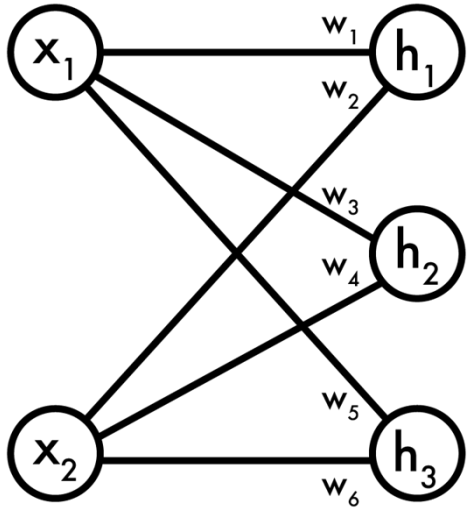


$$p = f(w_1x_1 + w_2x_2)$$

Outrepasser le problème de "feature engineering”

Et si on ajoutait des transformations?

- Créer de “nouvelles” caractéristiques à partir des précédentes
- Ajout d’une couche supplémentaire (cachée) nommée **H**

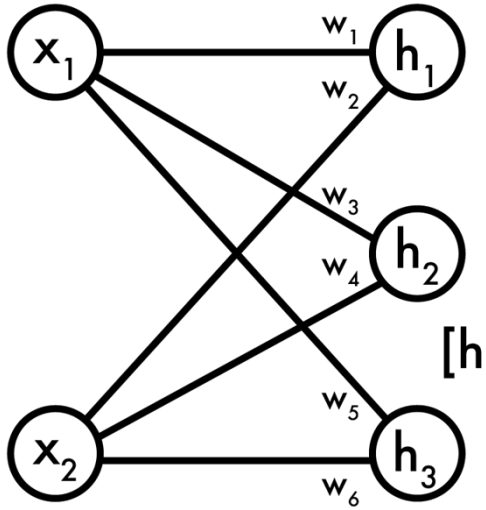


$$h_1 = \varphi(w_1 x_1 + w_2 x_2)$$
$$h_2 = \varphi(w_3 x_1 + w_4 x_2)$$
$$h_3 = \varphi(w_5 x_1 + w_6 x_2)$$

Outrepasser le problème de "feature engineering”

Et si on ajoutait des transformations?

- Comme le modele lineaire, **H** peut etre exprimé sous forme matricielle



$$h_1 = \varphi(w_1 x_1 + w_2 x_2)$$
$$h_2 = \varphi(w_3 x_1 + w_4 x_2)$$
$$h_3 = \varphi(w_5 x_1 + w_6 x_2)$$

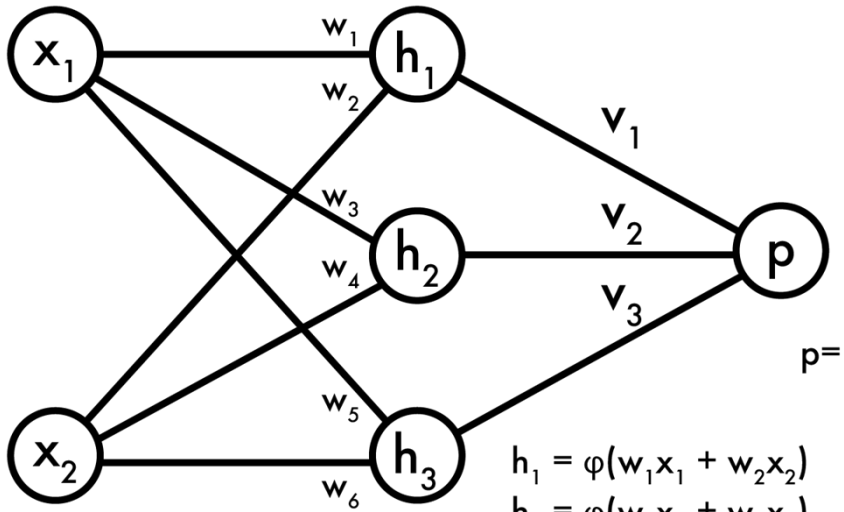
$$[h_1\ h_2\ h_3] = \varphi([x_1\ x_2]\begin{bmatrix} w_1 & w_3 & w_6 \\ w_2 & w_4 & w_5 \end{bmatrix})$$

$$\mathbf{H} = \varphi(\mathbf{Xw})$$

Outrepasser le problème de "feature engineering”

Et si on ajoutait des transformations?

- Maintenant, les prédictions p sont fonction de la couche cachée



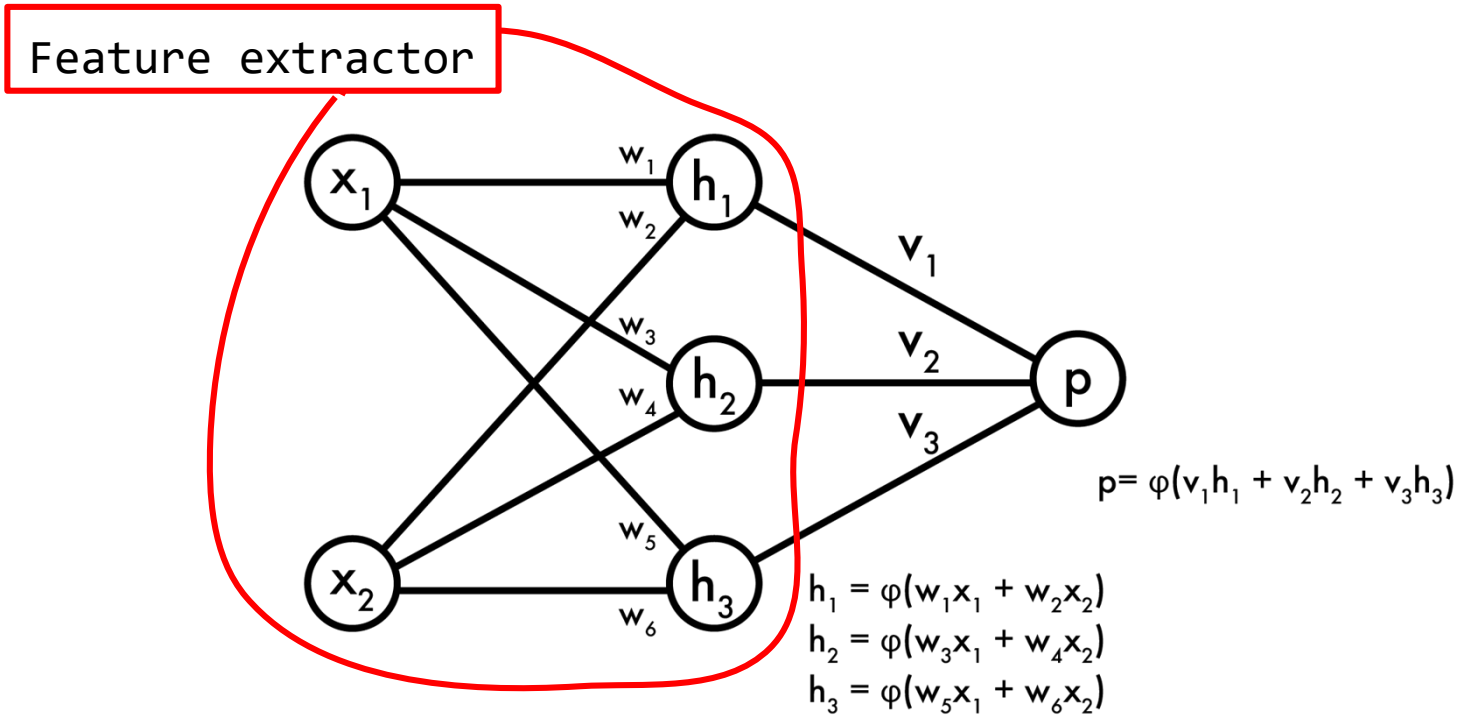
$p = \varphi(v_1 h_1 + v_2 h_2 + v_3 h_3)$

$h_1 = \varphi(w_1 x_1 + w_2 x_2)$
$h_2 = \varphi(w_3 x_1 + w_4 x_2)$
$h_3 = \varphi(w_5 x_1 + w_6 x_2)$

Outrepasser le problème de "feature engineering”

Et si on ajoutait des transformations?

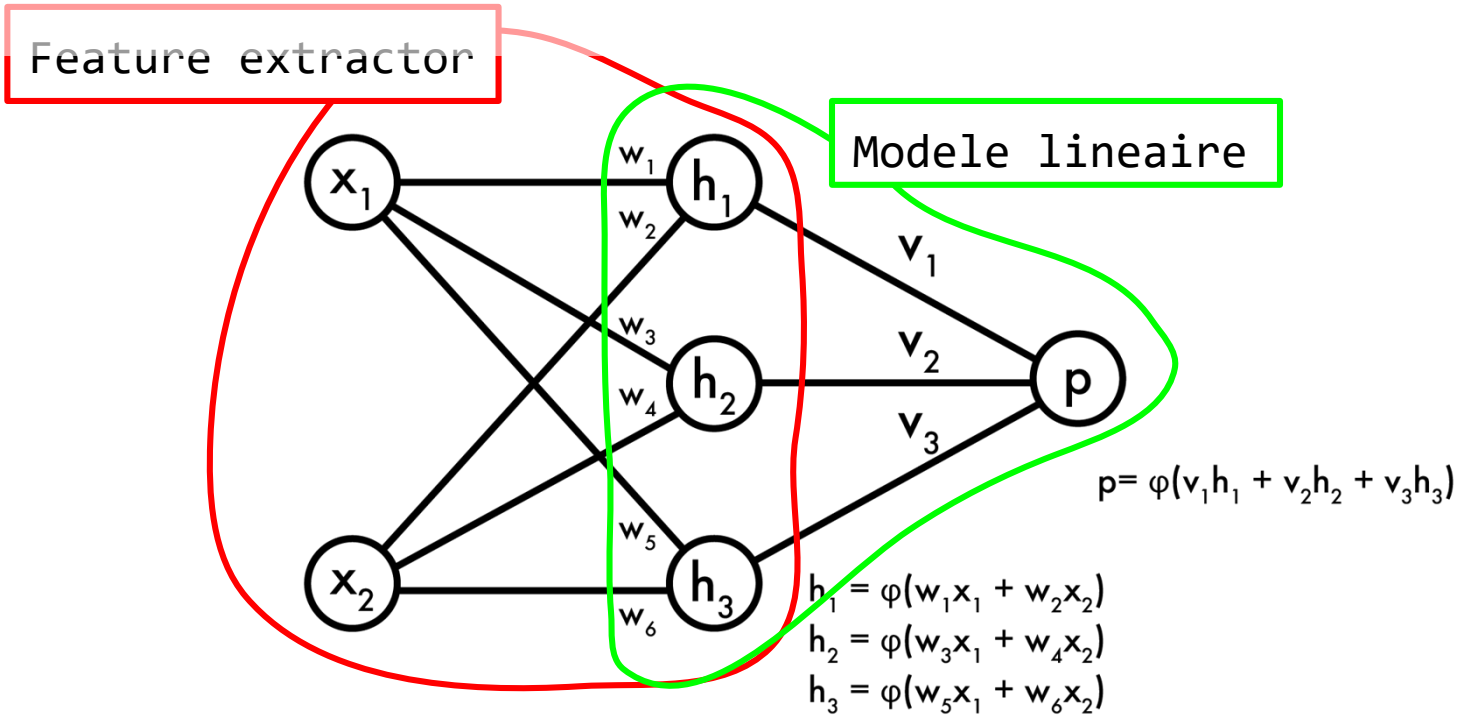
- Maintenant, les prédictions p sont fonction de la couche cachée



Outrepasser le problème de "feature engineering”

Et si on ajoutait des transformations?

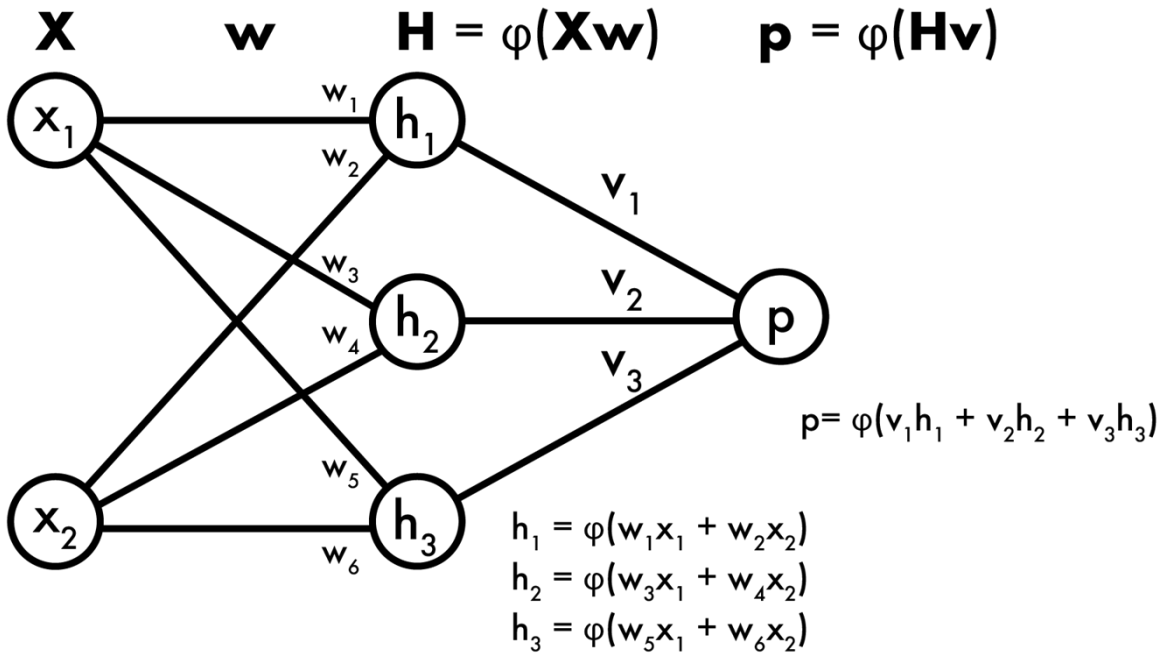
- Maintenant, les prédictions p sont fonction de la couche cachée



Outrepasser le problème de "feature engineering”

Et si on ajoutait des transformations?

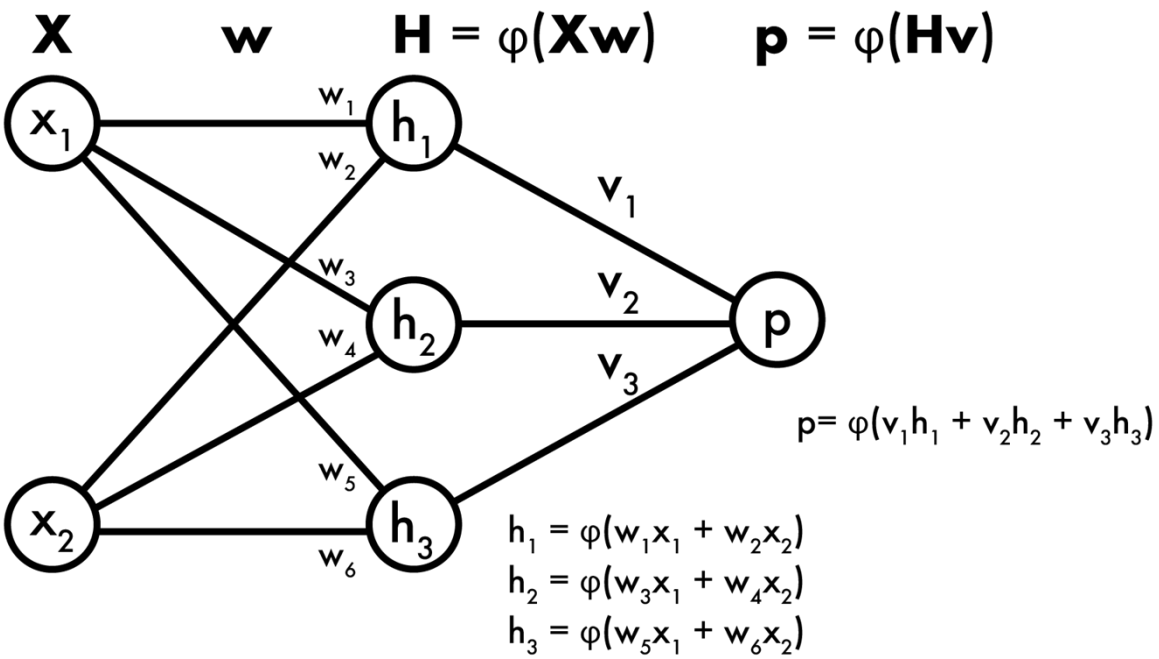
- L’ensemble du processus peut s’exprimer sous forme matricielle
- Très important pour des questions de temps de calcul



Outrepasser le problème de "feature engineering”

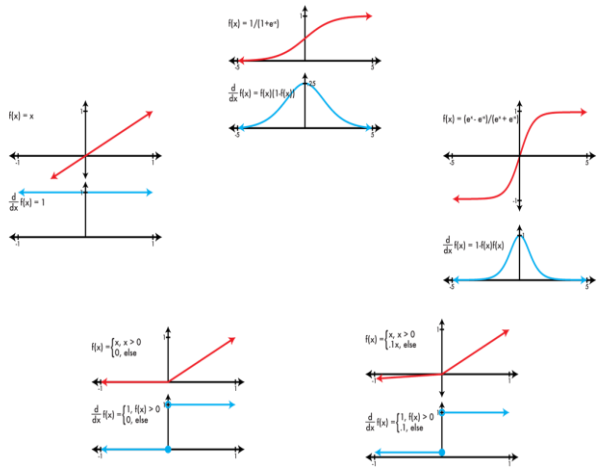
Ici commence à apparaitre la notion « Deep »

- Il est possible de multiplier le nombre de couches cachées
- Chaque couche correspondant à des fonctions φ appliquées aux combinaisons linéaires de la couche précédente



What about activation functions φ ?

- So many possible options
- want them to be easy to take derivative

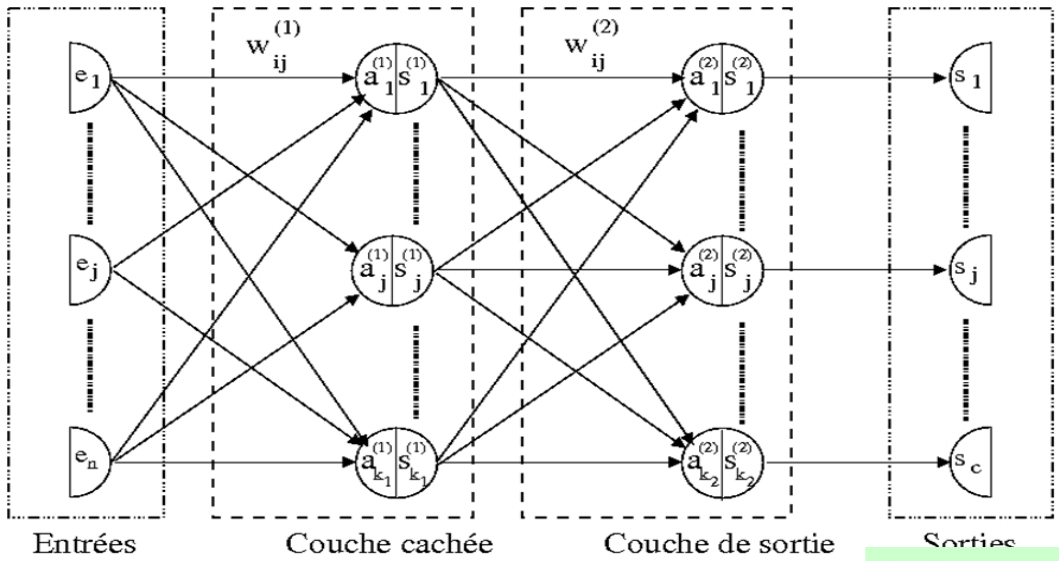


MLP Multi-classes

Architecture Multi-classes

- Des couches cachées
- Nombre de sorties selon la tache à résoudre → c
- Normalisation ***Softmax*** des valeurs de sortie pour générer des probabilités d'appartenance

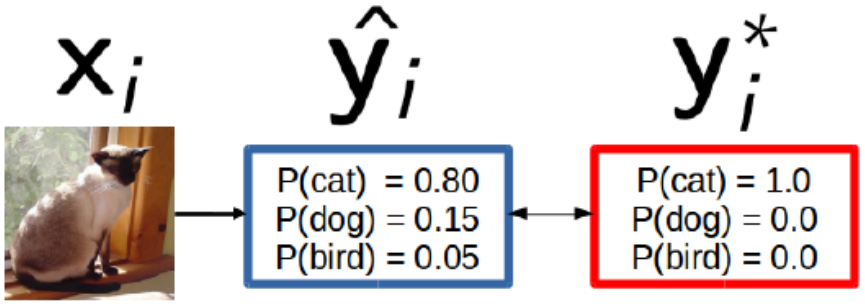
$$y_j = softmax(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{c} e^{z_k}}$$



- Flux de l'information
  - FORWARD : Les informations circulent « en avancant » dans le réseau. Il faut calculer toutes les sorties de la couche m -1 pour calculer les sorties de la couche m.
  - BACKPROPAGATION : L'information de coût circule « en reculant » à travers le réseau afin de calculer les gradients

MLP Multi-classes

Apprentissage



- Apprentissage
  - Trouver les w_{ij} minimisant l'erreur d'apprentissage $\mathcal{L}(W)$ sur la base d'apprentissage

$$\ell_{CE}(\hat{y}_i, y_i^*) = KL(y_i^*, \hat{y}_i) = -\sum_{c=1}^{K} y_{c,i}^* log(\hat{y}_{c,i}) = -log(\hat{y}_{c^*,i})$$

  - $\mathcal{L}(W)$ servira de base à la définition de la fonction *Loss()* = ∩ → à bien définir !
  - Souvent Loss() MLP = ∩ = Moyenne des erreurs sur 1 lot d'apprentissage de taille N

$$\mathcal{L}_{CE}(W, b) = \frac{1}{N} \sum_{i=1}^{N} \ell_{CE}(\hat{y}_i, y_i^*) = -\frac{1}{N} \sum_{i=1}^{N} log(\hat{y}_{c^*,i})$$

  - Utiliser la descente de gradient pour la minimisation de cette fonction

Rappels RN

Descente du gradient

- Processus itératif de modification des poids selon le signe du gradient :

$$W^{(t+1)} = W^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial W}$$

Sur la base d'appr.

$$\nabla_w^{(t)} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}(\hat{y}_i, y_i^*)}{\partial w} \left(w^{(t)}\right)$$

  - η est le taux d'apprentissage

- La propriété principale exploitée : chaine de dérivation (chain rule)

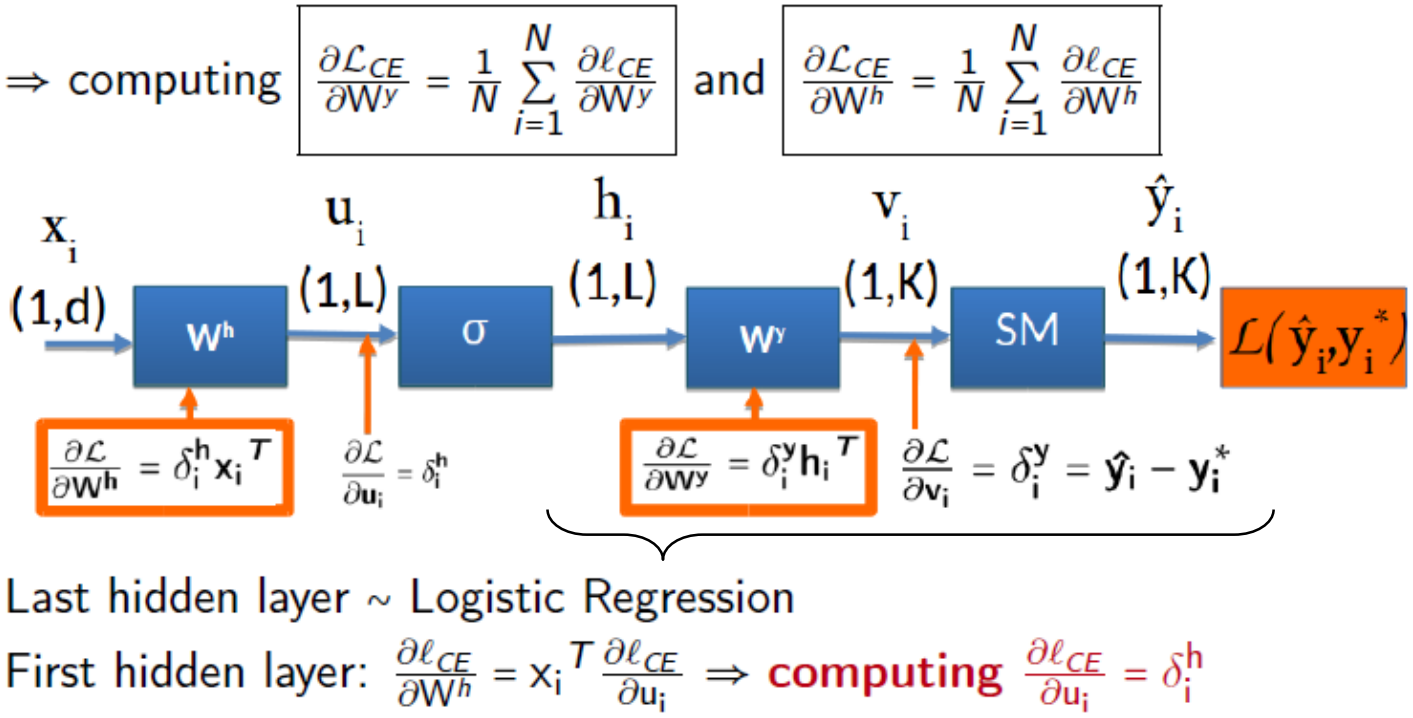
- **Pour un MLP :**

$$\frac{\partial x}{\partial z} = \frac{\partial x}{\partial y} \frac{\partial y}{\partial z} \Rightarrow \boxed{\frac{\partial \ell_{CE}}{\partial W} = \frac{\partial \ell_{CE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial W}}$$

Rappels RN

Descente du gradient

- Derivaton Chain Rule – Pour un échantillon



- Last hidden layer ~ Logistic Regression
- First hidden layer: $\frac{\partial \ell_{CE}}{\partial W^h} = x_i^T \frac{\partial \ell_{CE}}{\partial u_i}$ ⇒ **computing** $\frac{\partial \ell_{CE}}{\partial u_i} = \delta_i^h$

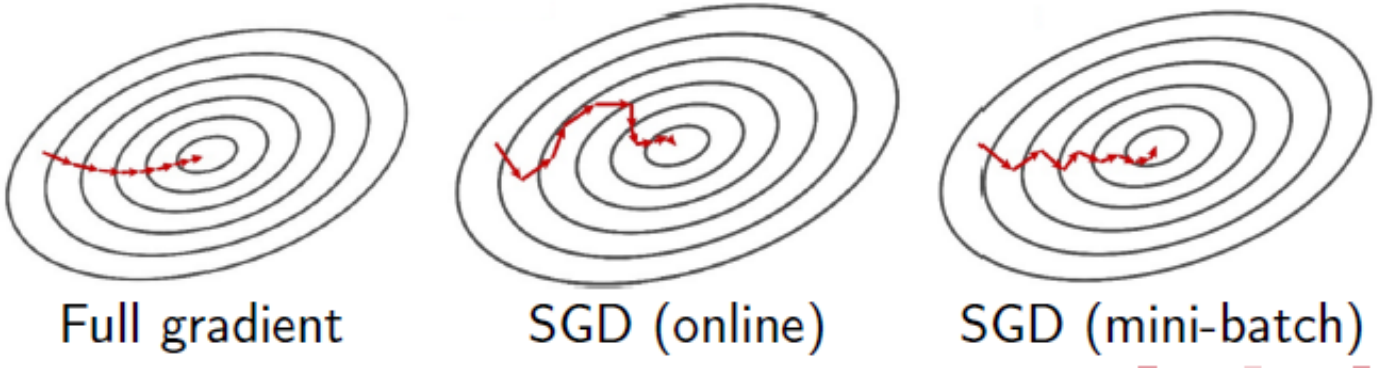
Voir bonne vidéo explicative → https://www.youtube.com/watch?v=tIeHLnjs5U8

→ Formulation extensible ensuite, sous forme matricielle pour un batch complet

Rappels MLP

Descente du gradient

- Le gradient peut être calculé pour
  - Toute la base N exemples → Full-Gradient → Convergence tres lente !
  - Chaque exemple → gradient stochastique (SGD)
  - Un mini-batch → 1 sous-partie B de la base d apprentissage N → Nb batches = N / B
  - SGD → Beaucoup mises à jour de paramètres : N,
  - Mini-batch → N / B → Convergence plus rapide
  - Approximation du vrai gradient

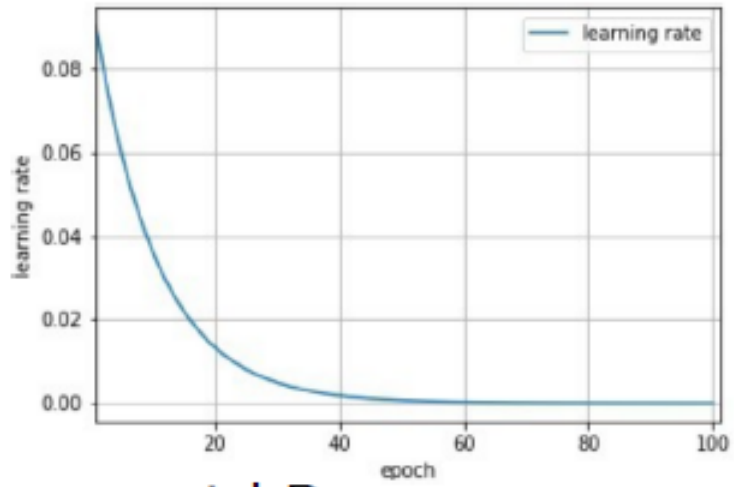


- 1 époque = 1 exploitation de tous les exemples de la base d'apprentissage

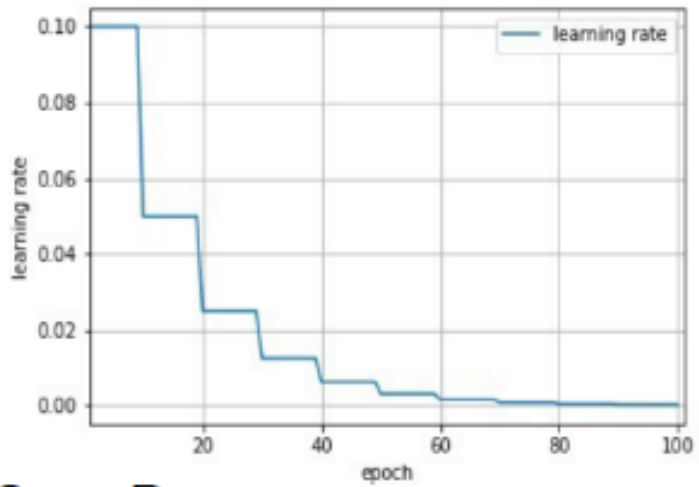
Rappels MLP

Réglages des hyper-parametres

- Choix de la valeur du Learning Rate η ?
- Diminution pendant la progression de l'apprentissage
  - Décroissance inverse : $\eta = \eta_0 / (1 + \lambda.t)$, λ = taux de décroissance
  - Décroissance exponentielle : $\eta = \eta_0 . exp(\lambda.t)$,
  - Décroissance par paliers : $\eta = \eta_0 . r^{(t / t_u)}$



Exponential Decay ($\eta_0 = 0.1$, $\lambda = 0.1s$)

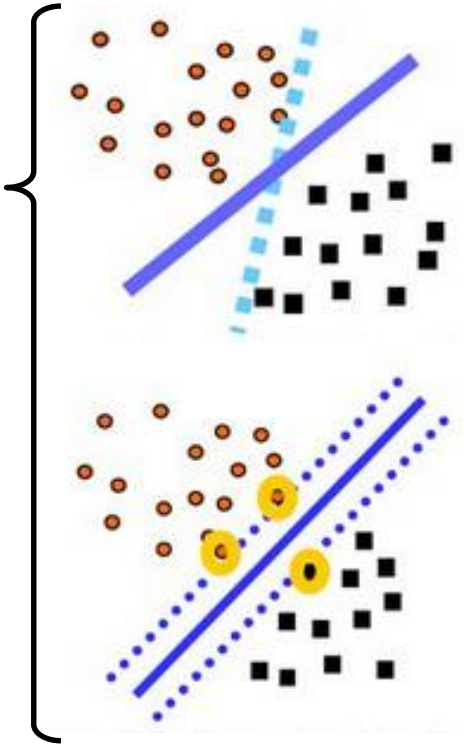


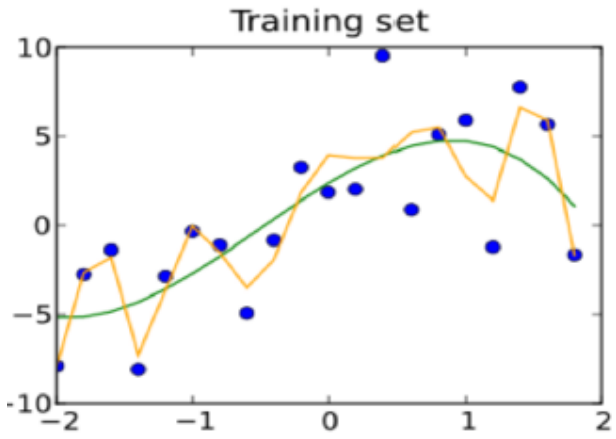
Step Decay ($\eta_0 = 0.1$, $r = 0.5$, $t_u = 10$)

Rappels MLP

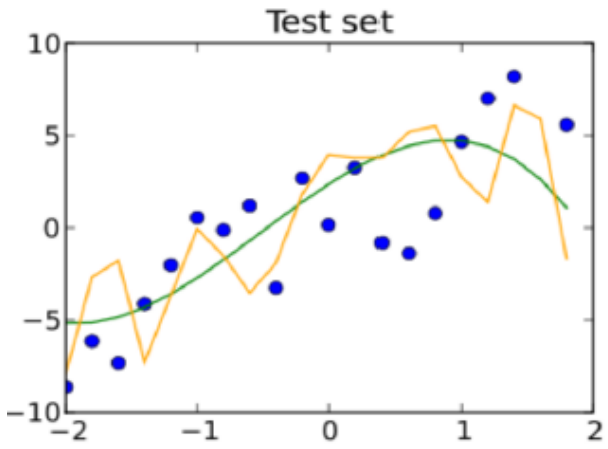
Réglages des hyper-parametres

- Régularisation : amélioration de la généralisation, c'est-à-dire des performances sur la base de test
  - Régularisation structurelle: ajouter un terme *R(w)* dans la Loss :
  - $\mathcal{L}(w) = \mathcal{L}_{CE}(w) + \lambda.R(w)$
  - Régularisation L_2 : décroissance du poids, $R(w) = ||w||^2$
  - Régularisation L_1, dropout, etc
- Illustration des limites de généralisation (par analogie SVM)





$\mathcal{L}_{train} = 4$, $\mathcal{L}_{train} = 9$



$\mathcal{L}_{test} = 15$, $\mathcal{L}_{test} = 13$

Rappels MLP

Réglages des hyper-parametres

- Paramètres d'apprentissage :
  - taux d'apprentissage, décroissance des poids, taille des mini batch, # époques, etc.
- Paramètres architecturaux :
  - nombre de couches, nombre de neurones,
  - type de non-linéarité,
  - etc.
- Réglage des hyper-paramètres :
  - estimer les performances sur un ensemble de validation en // de l'apprentissage