

# Deep Natural Language Processing

Gilles Vanwormhoudt



**IMT Lille Douai**  
École Mines-Télécom  
IMT-Université de Lille

# Plan

Natural Language Processing (NLP)

Apprentissage automatique et NLP

Réseaux de Neurones

Représentations

Apprentissage Profond et NLP

# Plan

Natural Language Processing (NLP)

Apprentissage automatique et NLP

Réseaux de Neurones

Représentations

Apprentissage Profond et NLP

# Qu'est ce que le NLP

Le NLP (Natural Language Processing) vise à créer des outils informatiques de traitement de la langue naturelle pour diverses applications

C'est un domaine à l'intersection de plusieurs champs :

- ▶ L'informatique
- ▶ L'intelligence artificielle
- ▶ Les sciences cognitives
- ▶ La science de l'information
- ▶ La linguistique (informatique)

## Buts

- ▶ Permettre aux ordinateurs de communiquer avec les humains en utilisant leur langage
- ▶ Apprendre aux ordinateurs à reproduire le comportement humain concernant le langage

Mais une compréhension complète et une représentation du sens par les ordinateurs restent illusoires !

# Le NLP est partout

Les applications du NLP sont nombreuses et vont des tâches simples à des tâches complexes

- ▶ La correction orthographique et grammatical
- ▶ Le recherche d'informations par mots clés ou par similarité
- ▶ La classification (commentaires positifs/négatifs) et le filtrage de documents (spam)
- ▶ Le résumé automatique de textes
- ▶ La traduction automatique de textes
- ▶ Les applications en lien avec la parole (synthèse de paroles, Text2Speech, Speech2Text)
- ▶ Les applications de questions/réponses (chatbots, Google/Alexa/Siri)
- ▶ La génération automatique de textes

# Quelques succès du NLP

A screenshot of a Google search results page for the query "natural language processing". The results include:

- An ad for "Natural Language Processing | AI powered Text Analytics" from "DeepL".
- A "Web Services" section with links to "DeepL API Services", "How A NLP Engine Works", and "How To Write Plugins For Web Services".
- A "Products & Services" section with links to "DeepL API", "DeepL Services And Ready To Use Services", and "DeepL API Plugins".
- A main result for "Natural language processing" which includes a snippet about NLP being a subfield of linguistics, computer science, information engineering, and artificial intelligence, followed by a link to Wikipedia.
- A "People also ask" box with questions like "What are the steps of natural language processing?", "What is natural language processing with example?", "What is the use of natural language processing?", and "How difficult is natural language processing?".
- A "People also search for" box with terms like "Machine learning", "Deep learning", "Intelligence", "Speech recognition", and "Language".

## Recherche d'informations



## Questions/réponses

A screenshot of the DeepL Translator interface. The source text in ANGLAIS (detected) is:

In this province where francophones form the majority, members of linguistic minorities feel a far greater need to express themselves in French than members of the francophone majority do to express themselves in English.

The translated text in FRANÇAIS is:

Dans cette province où les francophones forment la majorité, les membres des minorités linguistiques ressentent un besoin beaucoup plus grand de s'exprimer en français que les membres de la majorité francophone de s'exprimer en anglais.

## Traduction automatique

A screenshot of the Grammarly writing analysis interface. The text analyzed is:

Rooms that are tiny can be tricky to decorate but they can also be a lot of fun. So when a client challenged us to give her pocket size space a summer makeover for under \$500 dollars, we just couldn't say no. Transforming a very small space doesn't have to blow your budget. Small things like finding a vintage piece of furniture from a relative or adding a fresh coat of paint to your own dated items can add a stylish splash to any abode.

The analysis results are:

- Correctness: 2 alerts
- Clarity: A bit unclear
- Engagement: A bit bland
- Delivery: Slightly off

## Correction grammaticale et classification

# Les difficultés du NLP

Le langage naturel est conçu pour faciliter la communication humaine

- ▶ Nous omettons beaucoup de sens communs qui est supposé acquis par l'auditeur/le lecteur
- ▶ Nous conversons avec beaucoup d'ambiguités que nous supposons solvable pour l'auditeur/le lecteur

Cela rend difficile les tâches du NLP

Exemple d'ambiguités :

- ▶ Lexicale : Il y a beaucoup de bouchons ce matin. / Le bouchon a sauté jusqu'au plafond.
- ▶ Syntaxique : La petite brise la glace / Le boucher sale la tranche

# Plusieurs niveaux de traitements

## Niveau lexical

Segmentation des caractères en mots, identification des unités linguistiques

## Niveau syntaxique

Identification des structures grammaticales

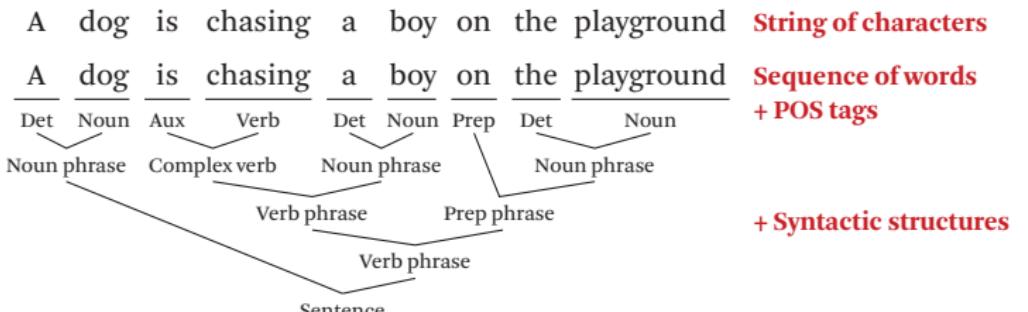
## Niveau sémantique

Représentation du sens

## Niveau pragmatique

Fonction de la phrase dans le texte, Analyse du discours

# Plusieurs niveaux de traitements



Dog(d1). Boy(b1). Playground(p1). Chasing(d1,b1,p1)

Speech act = REQUEST

+ Logic predicates

+ Speech acts

Deeper NLP: requires more human effort; less accurate

Closer to knowledge representation

# Les tâches du NLP

- ▶ Tâches lexicales
  - ▶ Segmentation mot/groupe de mots/phrase
  - ▶ Analyse morphologique des mots
- ▶ Tâches syntaxiques
  - ▶ Etiquettagé morpho-syntaxique
  - ▶ Regroupement syntaxique
  - ▶ Analyse syntaxique (arbre syntaxique, arbre de dépendances)
- ▶ Tâches sémantiques
  - ▶ Désambiguisation du sens
  - ▶ Identification des rôles sémantiques
  - ▶ Formulation logique
- ▶ Tâches pragmatiques
  - ▶ Résolution de coréférence
  - ▶ Analyse du discours

# Tâches lexicales

## Segmentation

Transformer la séquence de caractères en séquences de mots

- ▶ Utilisation des délimiteurs [<espace>, :,. ! ?]

## Analyse morphologique

- ▶ Identification du genre et nombre : images (féminin, pluriel)
- ▶ Identification du temps pour les verbes : analyserons (futur, 1er pers.pluriel)
- ▶ Décomposition en préfix, racine et suffix : [anti-con][stitu][tion-nelle-ment]
- ▶ Formes canoniques :
  - ▶ Racinisation : *automates, automatique, automatisation -> automat-*
  - ▶ Lemmatisation :  
*les voitures de garçons sont de différents couleurs – >*  
*le voiture un garçon être de différent couleur*

# Tâches syntaxiques

L'**étiquettagage morpho-syntaxique** (Part-Of-Speech Tagging) détermine la catégories syntaxiques des mots :

- ▶ Nom, Nom Propre, Verbe, Adverbe, Déterminant, Ponctuation, ...
- ▶ Un mot peut avoir plusieurs catégories possibles : *note (Nom, Verbe)* (sélection par estimation statistique - cf suite)

« Apple is looking at buying U.K. startup for \$1 billion »

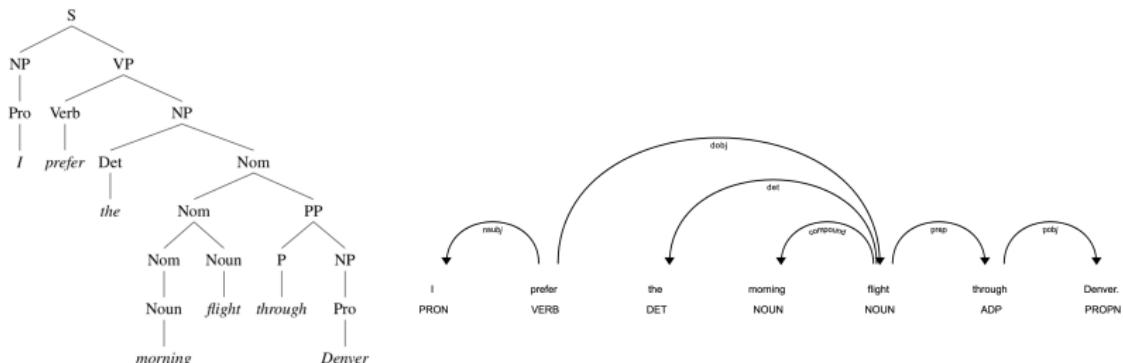
TEXT	LEMMA	POS	TAG	DEP
Apple	apple	PROPN	NNP	nsubj
is	be	VERB	VBZ	aux
looking	look	VERB	VBG	ROOT
at	at	ADP	IN	prep
buying	buy	VERB	VBG	pcomp
U.K.	u.k.	PROPN	NNP	compound
startup	startup	NOUN	NN	dobj
for	for	ADP	IN	prep
\$	\$	SYM	\$	quantmod
1	1	NUM	CD	compound
billion	billion	NUM	CD	pobj

# Tâches syntaxiques

L'**analyse syntaxique** détermine la structure existante entre les constituants d'un énoncé de façon à expliciter leur relation de dépendance.

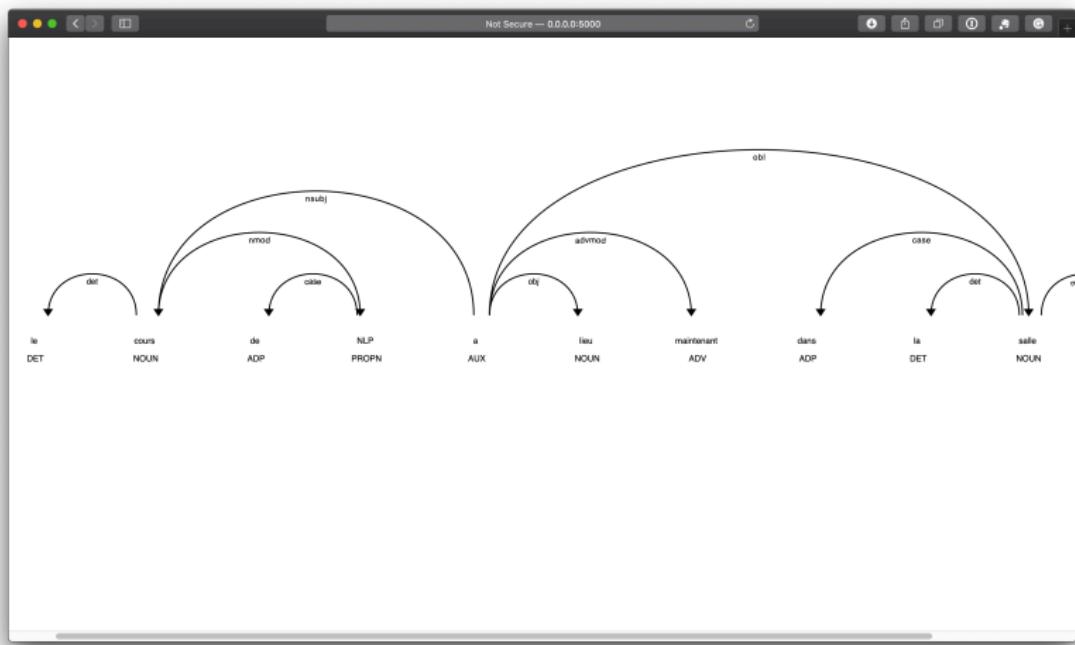
Principales représentation :

- ▶ représentation en constituants : décomposition en sous-phrases puis en mots
- ▶ représentation en dépendances : décomposition en identifiant les relations entre mots



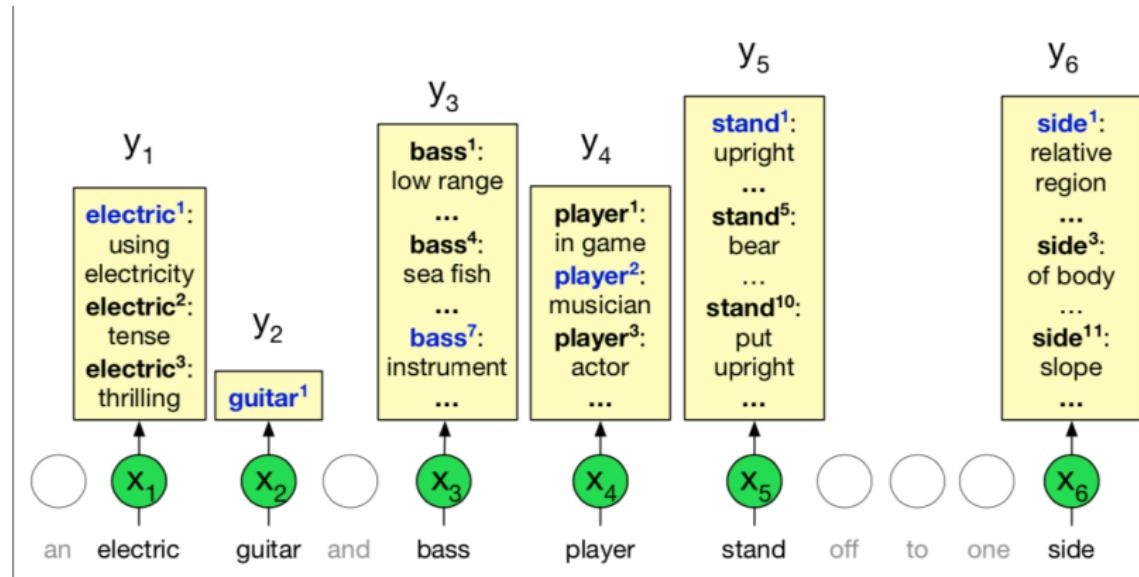
# Tâches syntaxiques

## Demo



# Tâches sémantiques

La **désambiguïsation du sens** est la détermination du sens d'un mot dans une phrase lorsque ce mot peut avoir plusieurs sens possibles.



# Tâches sémantiques

L'**identification des rôles sémantiques** consiste à déterminer le sens qui s'attache à un groupe nominal par rapport au procès exprimé par le verbe au sein d'une phrase.

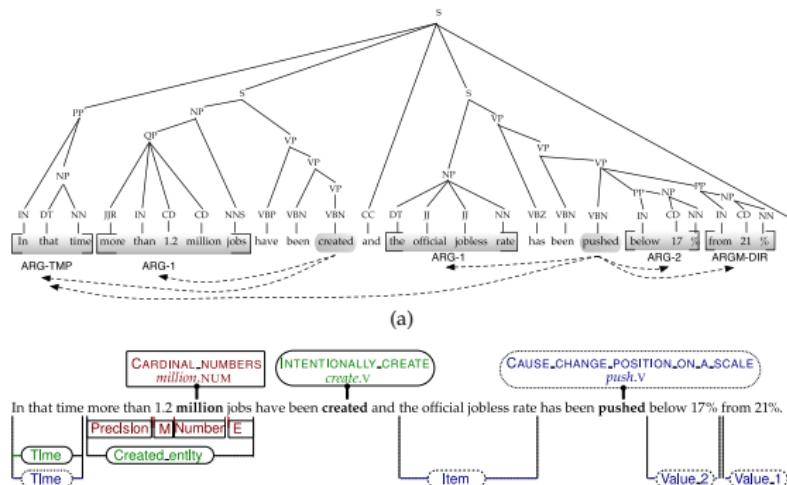
En effet, la fonction syntaxique d'un mot est insuffisante pour rendre compte du sens

- ▶ une même fonction peut recouvrir des situations très différentes
  - ▶ Exemple des plusieurs rôles sémantiques exprimés par le sujet :
  - ▶ L'homme ouvre la porte *{agent}*. La porte s'ouvre *{patient}*. La clé ouvre la porte *{instrument}*.
- ▶ une même situation peut s'exprimer par plusieurs fonctions
  - ▶ Exemple de l'expression du rôle sémantique *d'agent* par plusieurs fonctions :
  - ▶ Jacques a repeint la grille *{sujet}*. La grille a été repeinte par Jacques *{complément d' agent}*. On a fait repeindre la grille à Jacques *{COD}*.

# Tâches sémantiques

Il existe des référentiels pour l'identification de rôles sémantiques dont les plus connus sont :

- ▶ PropBank : identification des arguments associés à un sens du verbe
- ▶ Framenet : identification des rôles associés à un prédicat



On procède par l'identification du prédicat dans la phrase puis on cherche à classifier les autres mots en fonction de la structure correspondante

# Tâches pragmatiques

La **résolution de coréférence** consiste à lier les mots ou groupes désignant la même entité

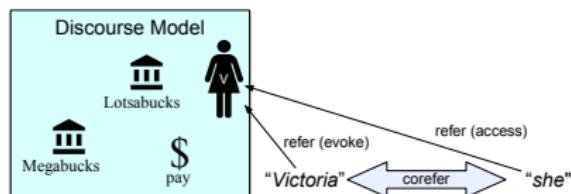
Vocabulaire de la coréférence : anaphore, antécédent

Anaphore :

- ▶ pronoms : il, elle, nous, ...
- ▶ groupes nominaux : le jeune homme, le président de la compagnie, ...

Victoria Chen, CFO of Megabucks Banking, saw her pay jump to \$2.3 million, as the 38-year-old became the company's president. It is widely known that she came to Megabucks from rival Lotsabucks.

1. {*Victoria Chen, her, the 38-year-old, She*}
2. {*Megabucks Banking, the company, Megabucks*}
3. {*her pay*}
4. {*Lotsabucks*}



Décomposition de la tâche

- ▶ Identification des mentions d'entités
- ▶ Regroupement des mentions désignant la même entité

# Tâches pragmatiques

## Détection d'entités : demo

The screenshot shows a web browser window with the URL [explosion.ai/demos/displacy-ent](https://explosion.ai/demos/displacy-ent). The page title is "displaCy Named Entity Visualizer". The main content area displays a news article snippet about Sebastian Thrun and Google's self-driving car project. The entities are highlighted with colored boxes: "Sebastian Thrun" (PERSON), "Google" (ORG), "2007" (DATE), "few" (QUANTITY), "American" (NORP), "Thrun" (ORG), and "earlier this week" (DATE). To the right of the text, there is a sidebar titled "Entity labels (select all)" with checkboxes for various entity types: PERSON, NORP, ORO, GPE, LOC, PRODUCT, EVENT, WORK OF ART, LANGUAGE, DATE, TIME, PERCENT, MONEY, QUANTITY, ORDINAL, and CARDINAL. The "PERSON" checkbox is checked. Below the sidebar, the text of the news article is shown again with the entities highlighted.

When Sebastian Thrun started working on self-driving cars at Google in 2007, few people outside of the company took him seriously. "I can tell you very senior CEOs of major American car companies would shake my hand and turn away because I wasn't worth talking to," said Thrun, now the co-founder and CEO of online higher education startup Udacity, in an interview with Recode earlier this week.

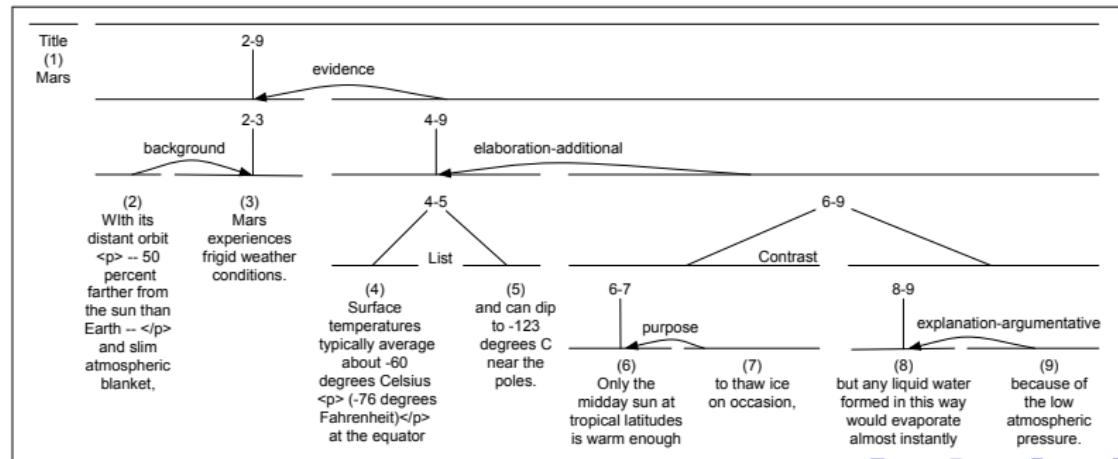
A little less than a decade later, dozens of self-driving startups have cropped up while automakers around the world clamor, wallet in hand, to secure their place in the fast-moving world of fully automated transportation.

# Tâches pragmatiques

L'analyse du discours consiste à établir les relations entre phrases pour déterminer la structure argumentaire et la cohérence d'un texte

## Rhetorical Structure Theory

- (23.12) With its distant orbit—50 percent farther from the sun than Earth—and slim atmospheric blanket, Mars experiences frigid weather conditions. Surface temperatures typically average about -60 degrees Celsius (-76 degrees Fahrenheit) at the equator and can dip to -123 degrees C near the poles. Only the midday sun at tropical latitudes is warm enough to thaw ice on occasion, but any liquid water formed in this way would evaporate almost instantly because of the low atmospheric pressure.



# Plan

Natural Language Processing (NLP)

Apprentissage automatique et NLP

Réseaux de Neurones

Représentations

Apprentissage Profond et NLP

# Apprentissage automatique

## Objectif

L'apprentissage automatique a pour but de concevoir des algorithmes capables d'apprendre à résoudre un problème à partir de données. Ces algorithmes améliorent leur performance sur une tâche donnée grâce à l'expérience.

L'apprentissage automatique permet d'apporter des solutions à des problèmes complexes pour lesquels la programmation classique trouve ses limites :

- ▶ Ecrire un programme pour reconnaître des caractères manuscrits est impossible à cause du nombre infini de cas à considérer.
- ▶ L'apprentissage automatique traite cette problématique différemment : la machine va apprendre par elle-même à partir d'observations.

Les algorithmes d'apprentissage sont des systèmes « entrée-sortie » avec une donnée en entrée et une information en sortie

# Familles d'apprentissage automatique

- ▶ Apprentissage supervisé : Apprentissage à partir de données étiquetées ou annotées (par des experts).
- ▶ Apprentissage non supervisé : Apprentissage à partir de données non étiquetés
- ▶ Apprentissage par renforcement : Apprentissage continu d'une machine (ou d'un agent) à partir d'un retour d'expérience.

L'apprentissage par renforcement est encore peu utilisé en NLP donc nous le laisserons de côté

# Apprentissage supervisé : Principe et objectif

Apprentissage à partir de données étiquetées ou annotées (par des experts).

## Données à fournir

- ▶ Spécification de l'entrée : une représentation  $x$  des données (caractéristiques des individus ou objets de l'étude)
- ▶ Spécification de la sortie : une cible souhaitée  $y$  qui peut être un nombre, une catégorie (label / étiquette), une entité ou une structure d'entités
- ▶ Un ensemble d'entraînement (training set), qui est un ensemble de couples entrée-sortie connus :  $\{(x^1, y^1), \dots, (x^n, y^n)\}$

## Objectif

A partir de l'ensemble d'entraînement, fournir un modèle qui prédit la sortie  $y$  pour une nouvelle entrée  $x$ .

Le but est donc de généraliser ce que l'algorithme a appris grâce aux données annotées à de nouvelles entrées inconnues.

# Apprentissage supervisé : Formalisation

## Formalisation

- ▶ Entrée : un vecteur  $x = (x_1, \dots, x_d)^T \in \mathcal{X}$ ,  $\mathcal{X} \subseteq \mathbb{R}^d$  comprenant les  $d$  caractéristiques (features) d'un objet
- ▶ Sortie :  $y \in \mathcal{Y} \subseteq \mathbb{R}^p$
- ▶ Ensemble d'apprentissage : un ensemble de  $n$  exemples  $\{(x^j, y^j) \in \mathcal{X} \times \mathcal{Y}\}_{j=1}^n$
- ▶ Objectif : Trouver une fonction / un modèle  $f : \mathcal{X} \rightarrow \mathcal{Y}$  qui minimise l'erreur de prédiction.

Différents algorithmes suivant la forme générale de  $f$  :

- ▶ Modèle linéaire :  $f$  linéaire ou affine
- ▶ Modèle polynomiale :  $f$  polynomiale
- ▶ Modèle non linéaire :  $f$  non linéaire
- ▶ Modèle non paramétrique :  $f$  sans forme prédéterminée

# Apprentissage supervisé : Modèle linéaire

La fonction  $f$  est linéaire de la forme :

$$f(x) = x \cdot \mathbf{W} + \mathbf{b}$$
$$x \in \mathbb{R}^d \quad \mathbf{W} \in \mathbb{R}^{d \times p} \quad \mathbf{b} \in \mathbb{R}^p$$

La matrice  $\mathbf{W}$  et le vecteur  $\mathbf{b}$  sont les paramètres pour explorer l'espace des fonctions linéaires.

Par la suite, les paramètres associés à la fonction  $f$  seront désignés par  $\Theta$  et rendu explicite :  $f(x; \Theta)$

Dans le cas linéaire :  $\Theta = \mathbf{W}, \mathbf{b}$

# Apprentissage supervisé : Problèmes

## Régression

- ▶ La sortie à prédire  $y$  est une valeur réelle :  $\mathcal{Y} \subseteq \mathbb{R}$
- ▶ Exemple : Prédiction économique :  $x$  = activité économique de la journée,  $y$  = valeur d'une action

## Classification

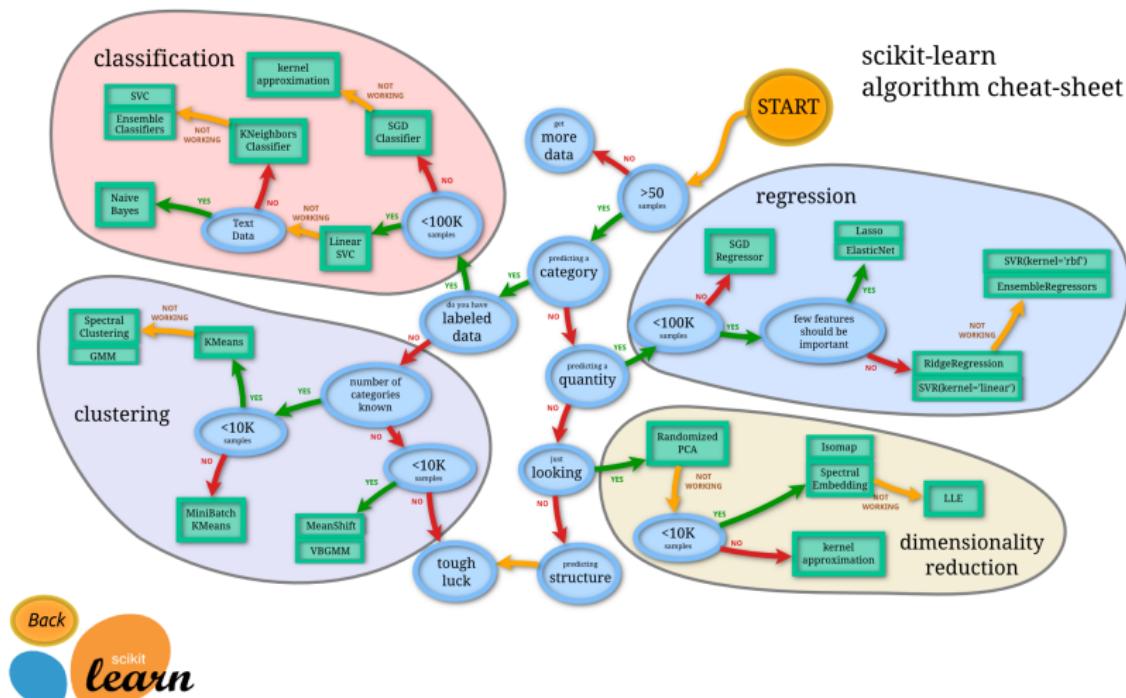
- ▶ La sortie à prédire  $y$  est une classe :  $\mathcal{Y}$  discret
- ▶ Si  $\#\mathcal{Y} = 2$ , classification binaire, si  $\#\mathcal{Y} > 2$ , classification multi-classes.
- ▶ Ex : Reconnaissance de caractères :  $x$  = image (matrice de pixels),  $y$  = caractère écrit

## Prédiction de sorties structurées

- ▶ La sortie à prédire  $y$  est un objet structuré :  
 $\mathcal{Y} \subseteq \{\text{ensemble de toutes les structures possibles}\}$
- ▶ Ex : Recherche d'informations,  $x$  = liste de mots clés,  $y$  = liste ordonnée de documents selon leur pertinence



## Apprentissage supervisé : Méthodes



# Apprentissage supervisé par optimisation

## Fonction de perte

- Soit la fonction de perte  $L(\hat{y}, y)$  qui évalue la perte obtenue quand la prédiction est  $\hat{y}$  et la sortie attendue est  $y$ .
- Soit  $\mathcal{L}$  la valeur moyenne des pertes sur l'ensemble d'entraînement :

$$\mathcal{L}(\Theta) = \frac{1}{n} \sum_{i=1}^n L(f(x_i; \Theta), y_i) \quad (1)$$

## Objectif

L'objectif de l'apprentissage par optimisation est de déterminer  $\hat{\Theta}$  tel que  $\mathcal{L}$  soit minimisée :

$$\hat{\Theta} = \operatorname{argmin}_{\Theta} L(\Theta) = \operatorname{argmin}_{\Theta} \frac{1}{n} \sum_{i=1}^n L(f(x_i; \Theta), y_i) \quad (2)$$

On ajoute souvent un terme de régularisation  $\lambda \mathcal{R}(\Theta)$  à l'équation précédente pour contrôler la complexité et éviter les cas d'overfitting.



# Fonctions de pertes

## Hinge (sortie linéaire)

- ▶ Binaire (sortie dans  $\{-1, +1\}$ ) :

$$L_{hinge(binary)}(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$$

- ▶ Multi-classes (sortie vectorielle one-hot) :

$$L_{hinge(multi-class)}(\hat{y}, y) = \max(0, 1 - (\hat{y}_{[t]} - \hat{y}_{[k]})) \text{ où}$$

$t = \operatorname{argmax}_i y_{[i]}$  la classe correcte et  $k = \operatorname{argmax}_{i \neq t} \hat{y}_{[i]}$  la classe prédictive avec le plus haut score

## Cross-Entropy

- ▶ Binaire (sortie sigmoid) :

$$L_{logistic}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

- ▶ Multi-classes (sortie softmax) :

$$L_{cross-entropy}(\hat{y}; y) = -\sum_i y_{[i]} \log(\hat{y}_{[i]})$$

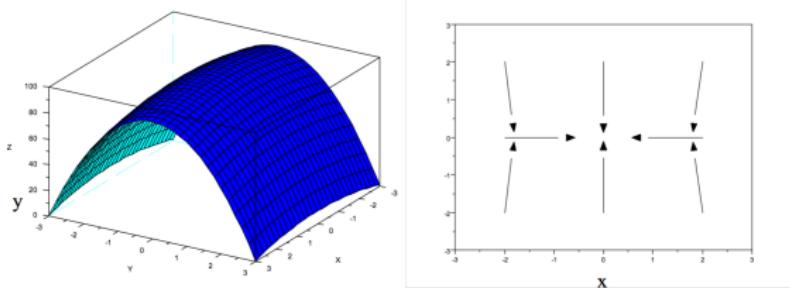
# Descente de gradient

Pour résoudre le problème d'optimisation précédent :

- ▶ La méthode la plus connue est la **descente de gradient**.
- ▶ L'idée est d'améliorer la solution courante (minimisant  $\mathcal{L}(\Theta)$ ) de façon itérative en progressant dans la direction de plus grande pente.

Le gradient représente la pente de la tangente du graphe de la fonction en un point :

- ▶ La direction du gradient correspond à la ligne de plus grande pente au point considéré.
- ▶ Le module du gradient correspond à la pente de la surface au point considéré.



En un point, pour faire décroître la fonction de perte  $L$ , il faut se déplacer dans la direction opposée au gradient.

# Algorithme de descente de gradient

*Input:*

- Function  $f(\mathbf{x}; \Theta)$  parameterized with parameters  $\Theta$ .
- Training set of inputs  $\mathbf{x}_1, \dots, \mathbf{x}_n$  and desired outputs  $y_1, \dots, y_n$ .
- Loss function  $L$ .

---

```
1: while stopping criteria not met do
2:   Sample a training example  $\mathbf{x}_i, y_i$ 
3:   Compute the loss  $L(f(\mathbf{x}_i; \Theta), y_i)$ 
4:    $\hat{\mathbf{g}} \leftarrow$  gradients of  $L(f(\mathbf{x}_i; \Theta), y_i)$  w.r.t  $\Theta$ 
5:    $\Theta \leftarrow \Theta - \eta_t \hat{\mathbf{g}}$ 
6: return  $\Theta$ 
```

- ▶  $\eta_t$  est le pas (learning rate). C'est un paramètre qu'il faut bien régler :
  - ▶ Si  $\eta_t$  trop faible, convergence lente
  - ▶ Si  $\eta_t$  trop élevé, risque d'oscillation
- ▶ Convergence  $\leftrightarrow$  nombre d'itérations fixé ou norme du gradient très faible
- ▶ Problème : l'algorithme peut converger vers un minimum local et non le minimum global si la fonction à minimiser n'est pas convexe.
- ▶ Attention au problème du "vanishing and exploding gradient" (resp. 0 et très élevé)

# Exemple

Exemple de descente de gradient en dimension  $d = 1$

$$\begin{cases} f(x) = x^2 - x + 1 \\ \nabla f(x) = \frac{\partial f(x)}{\partial x} = f'(x) = 2x - 1 \end{cases}$$

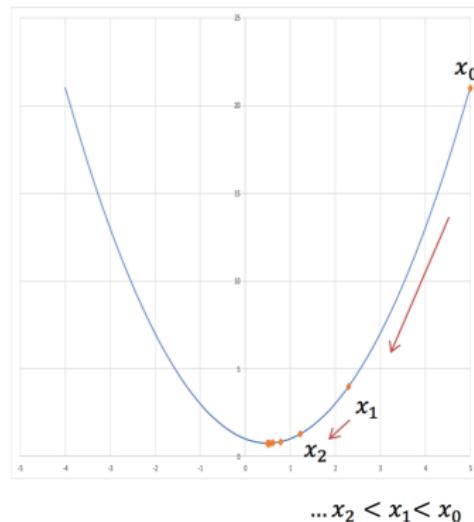
Il n'y a qu'un seul paramètre, la dérivée partielle est égale à la dérivée.

$\eta = 0.3$

$x_0 = 5$

x	f'(x_t)	f(x)
5.0000	21.0000	
2.3000	9.0000	3.9900
1.2200	3.6000	1.2684
0.7880	1.4400	0.8329
0.6152	0.5760	0.7633
0.5461	0.2304	0.7521
0.5184	0.0922	0.7503
0.5074	0.0369	0.7501
0.5029	0.0147	0.7500
0.5012	0.0059	0.7500
0.5005	0.0024	0.7500
0.5002	0.0009	0.7500
0.5001	0.0004	0.7500
0.5000	0.0002	0.7500

$$x_{t+1} = x_t - \eta \times \nabla f(x_t)$$



# Exemple de descente de gradient

On considère un problème de classification multilabels en utilisant la fonction de perte Hinge :

$$\hat{y} = \operatorname{argmax}_i \hat{y}_{[i]}$$

$$\hat{y} = f(x) = xW + b$$

$$L(\hat{y}, y) = \max(0, 1 - (\hat{y}_{[t]} - \hat{y}_{[k]}))$$

$$= \max(0, 1 - ((xW + b)_{[t]} - (xW + b)_{[k]}))$$

$$t = \operatorname{argmax}_i y_{[i]}$$

$$k = \operatorname{argmax}_i \hat{y}_{[i]} \quad i \neq t.$$

$$\frac{\partial L(\hat{y}, y)}{\partial W} = \begin{pmatrix} \frac{\partial L(\hat{y}, y)}{\partial W_{[1,1]}} & \frac{\partial L(\hat{y}, y)}{\partial W_{[1,2]}} & \cdots & \frac{\partial L(\hat{y}, y)}{\partial W_{[1,n]}} \\ \frac{\partial L(\hat{y}, y)}{\partial W_{[2,1]}} & \frac{\partial L(\hat{y}, y)}{\partial W_{[2,2]}} & \cdots & \frac{\partial L(\hat{y}, y)}{\partial W_{[2,n]}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial L(\hat{y}, y)}{\partial W_{[m,1]}} & \frac{\partial L(\hat{y}, y)}{\partial W_{[m,2]}} & \cdots & \frac{\partial L(\hat{y}, y)}{\partial W_{[m,n]}} \end{pmatrix}$$

$$\frac{\partial L(\hat{y}, y)}{\partial b} = \left( \frac{\partial L(\hat{y}, y)}{\partial b_{[1]}} \quad \frac{\partial L(\hat{y}, y)}{\partial b_{[2]}} \quad \cdots \quad \frac{\partial L(\hat{y}, y)}{\partial b_{[n]}} \right).$$

$$L(\hat{y}, y) = \max(0, 1 - (\hat{y}_{[t]} - \hat{y}_{[k]}))$$

$$= \max(0, 1 - ((xW + b)_{[t]} - (xW + b)_{[k]}))$$

$$= \max \left( 0, 1 - \left( \left( \sum_i x_{[i]} \cdot W_{[i,t]} + b_{[t]} \right) - \left( \sum_i x_{[i]} \cdot W_{[i,k]} + b_{[k]} \right) \right) \right)$$

$$= \max \left( 0, 1 - \sum_i x_{[i]} \cdot W_{[i,t]} - b_{[t]} + \sum_i x_{[i]} \cdot W_{[i,k]} + b_{[k]} \right)$$

$$t = \operatorname{argmax}_i y_{[i]}$$

$$k = \operatorname{argmax}_i \hat{y}_{[i]} \quad i \neq t.$$

$$\frac{\partial L}{\partial b_{[i]}} = \begin{cases} -1 & i = t \\ 1 & i = k \\ 0 & \text{otherwise.} \end{cases}$$

$$\frac{\partial L}{\partial W_{[i,j]}} = \begin{cases} \frac{\partial (-x_{[i]} \cdot W_{[j,0]})}{\partial W_{[i,j]}} & = -x_{[i]} \quad j = t \\ \frac{\partial (x_{[i]} \cdot W_{[j,k]})}{\partial W_{[i,k]}} & = x_{[i]} \quad j = k \\ 0 & \text{otherwise.} \end{cases}$$

# Apprentissage supervisé et NLP

De nombreuses tâches du NLP peuvent être formulées comme un problème d'apprentissage supervisé

Les entrées-sorties dépendent de la tâche

## Entrées

- ▶ Types : caractères, groupe de caractères, mots, propriétés linguistiques de mots, groupe de mots, phrases, paragraphe, document, structures linguistiques (arbre/graphe)
- ▶ Structure : ensemble, histogramme, distribution, séquence, arbre, graphe

## Sorties

- ▶ Types : étiquettes de classes, étiquette de mots, mots, groupe de mots, phrases, structures linguistiques
- ▶ Structure : distribution, séquence, ensemble, arbre, graphe

# Exemples d'apprentissage supervisé en NLP

## Identification de la langue utilisée dans les documents

- ▶ Entrée observée  $x_i$  : l'histogramme des paires de caractères dans le document
- ▶ Sortie observée  $y_i$  : langue du document

## Analyse de sentiments

- ▶ Entrée observée  $x_i$  : histogramme des mots dans le texte,
- ▶ Sortie observée  $y_i$  :  $\{-1 : \text{Negatif}, 0 : \text{Neutre}, 1 : \text{Positif}\}$

## Étiquetage morpho-syntaxique

- ▶ Entrée observée  $x_i$  : séquence de mots
- ▶ Sortie observée  $y_i$  : séquence d'étiquettes  
 $POS = \{NOUN, VERB, \dots\}$

# Exemples d'apprentissage supervisé en NLP

## Détection d'entités nommées

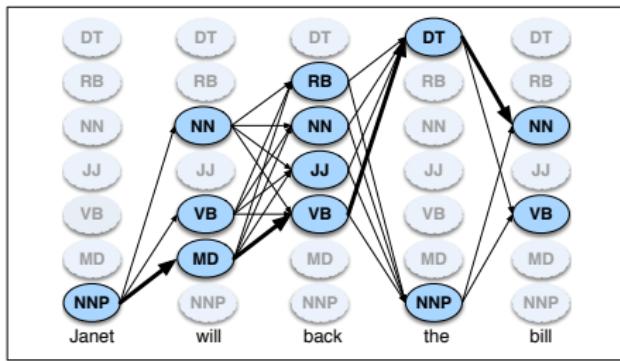
- ▶ Entrée observée  $x_i$  : séquence de mots
- ▶ Sortie observée  $y_i$  : séquence d'étiquettes  
 $BIO = \{O, B - PER, I - PER, B - LOC, I - LOC, \dots\}$

## Modèle de langue

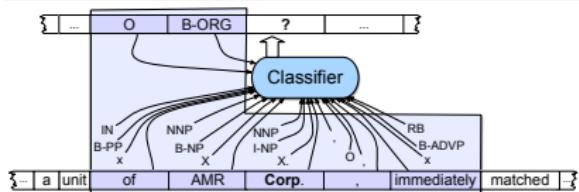
- ▶ Entrée observée  $x_i$  : séquence de mots
- ▶ Sortie observée  $y_i$  : 1 mot ou une séquence de mots

# Exemples d'apprentissage supervisé en NLP

## Etiquettagement morpho-syntaxique



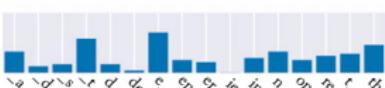
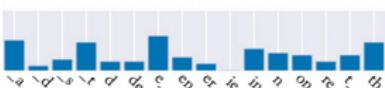
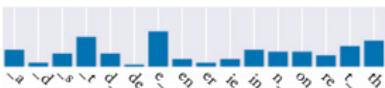
## Détection d'entités nommées



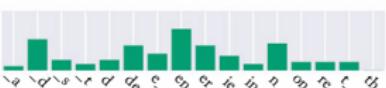
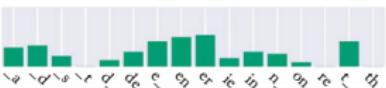
## Exemple détaillé : Identification de la langue des documents

Les paires de lettres consécutives (bi-grams de caractères) des mots d'un document forment une bonne caractéristique pour l'identification de la langue

## Document en anglais



## Document en allemand



# Exemple détaillé : Identification de la langue des documents

Le problème peut-être formulé comme une problème de classification multi-classes (6) :  $L = \{En, Fr, De, It, Sp, O\}$  ( $O = \text{other}$ )

## Formalisation

- ▶ Entrée :  $x = (x_{aa}, x_{ab}, \dots, x_{zz})^T \in \mathbb{R}^d$  où  $d$  est la taille de l'alphabet  $a, \dots, z$  au carré
- ▶ Sortie :  $y \in L$
- ▶ Ensemble d'apprentissage :  $\{(x^j, y^j) \in \mathbb{R}^d \times \mathbb{R}\}_{j=1}^n$
- ▶ Hypothèse du modèle linéaire :  
$$\hat{y} = softmax(f(x)) = softmax(x \cdot \mathbf{W} + \mathbf{b})$$
 où  $\hat{y} \in \mathbb{R}^6$ ,  $\mathbf{W} \in \mathbb{R}^{d \times 6}$  et  $\mathbf{b} \in \mathbb{R}^6$
- ▶ Prédiction (langue ayant le plus haut score) :  
$$prediction = \operatorname{argmax}_i \hat{y}[i]$$

# Exemple détaillé : Identification de la langue des documents

Cette exemple conduit à manipuler plusieurs représentations vectorielles

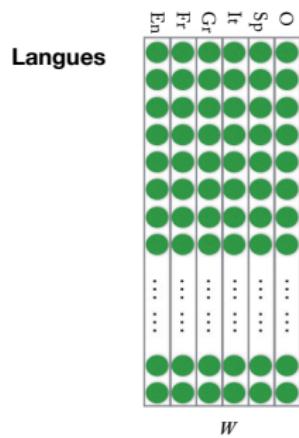
- ▶ Représentation du document : l'entrée  $x$  et la sortie  $\hat{y}$
- ▶ Représentation des langues : les colonnes de la matrice  $W$
- ▶ Représentation des bigrams : les lignes de la matrice  $W$

Document

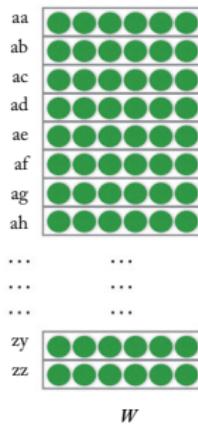


$$x = [2 \ 6 \ 2 \ 2 \ 1 \ 3 \ \dots \ 2 \ 1]$$

$$\hat{y}^{\wedge} = [0.02864, 0.07786, 0.21165, 0.5753, 0.02864, 0.07786]$$



Bigrams



# Apprentissage non supervisé

Apprentissage à partir de données non étiquetées.

## Données à fournir

- ▶ Spécification de l'entrée : une représentation  $x$  des données (caractéristiques des individus ou objets de l'étude)
- ▶ Pas de spécification de la sortie : pas de cible  $y$  précisée !
- ▶ Un ensemble d'observations :  $\{x^1, \dots, x^n\}$

## Objectif

Rechercher des structures sous-jacentes dans les données pour les organiser de façon logique :

- ▶ Analyser les relations entre les objets
- ▶ Former des groupes d'objets similaires
- ▶ Trouver de meilleures représentations des objets

On ne sait pas très bien ce que l'on cherche...

# Apprentissage non supervisé : formalisation

## Formalisation

- ▶ Entrée : un ensemble de données :  $\mathcal{D} = \{x^1, x^2, \dots, x^n\}$ 
  - ▶  $x^j$  est un vecteur  $(x_1^j, \dots, x_d^j)^T \in \mathbb{R}^d$  comprenant les  $d$  caractéristiques (features) d'une donnée  $j$
- ▶ Sortie : pas de cible précise, le résultat dépend du type de problème considéré
- ▶ Objectif général : Analyser les données :
  - ▶ Trouver des similarités entre les données
  - ▶ Trouver des corrélations entre les caractéristiques représentant les données

# Apprentissage non supervisé : utilisation

## Quand faire appel à l'apprentissage non supervisé ?

- ▶ Pas d'annotation possible des données :
  - ▶ Pas d'expert
  - ▶ Trop coûteux ou/et trop long (trop de données, de caractéristiques, de catégories...)
- ▶ Besoin d'une analyse exploratoire des données pour trouver une structure, un motif dans les données
  - ▶ Trouver les caractéristiques (features) les plus pertinentes
  - ▶ Prétraiter les données avant l'application d'un algorithme d'apprentissage supervisé

# Apprentissage non supervisé : problèmes cibles

## Le partitionnement (clustering)

- ▶ On cherche à construire des classes (clusters) automatiquement : classification non supervisée
- ▶ Exemple : Définition de groupes de clients

## La réduction de dimensions

- ▶ On cherche à réduire la dimension du jeu de données en se limitant aux caractéristiques (features) les plus pertinentes
- ▶ Réduction des coûts (calcul, stockage), Visualisation des données (projection en dimensions 2 ou 3)

## La découverte de règles d'association

- ▶ On recherche des relations entre les données : identification d'items qui apparaissent ensemble lors d'un événement
- ▶ Exemple : Secteur de la vente : recherche des produits fréquemment achetés ensemble

# Apprentissage non supervisé et NLP

## Partitionnement des textes d'un corpus

- ▶ Entrée : un ensemble de documents  $\mathcal{D} = \{d_1, \dots, d_i, \dots, d_n\}$  et  $k$  le nombre de partitions souhaitées
- ▶ Sortie : une partition  $\mathcal{P} = \{G_1, G_2, \dots, G_k\}$  de  $\mathcal{D}$

## Représentation latente de textes (mots, phrases, paragraphes, documents)

- ▶ Entrée : une représentation  $x = (x_1, x_2, \dots, x_d)^T \in \mathcal{R}^d$  du texte avec  $d$  caractéristiques
- ▶ Sortie : une nouvelle représentation  $x' = (x'_1, \dots, x'_{d'})^T \in \mathcal{R}^{d'}$  du texte avec un nombre de caractérisques plus faibles

## Découverte de thèmes dans une collection de textes

- ▶ Entrée : un ensemble de documents  $\mathcal{D} = \{d_1, \dots, d_i, \dots, d_n\}$  et  $k$  thèmes
- ▶ Sortie : distribution des  $k$  thèmes dans chaque document, distribution des mots dans chaque thème  $k$

# Exemple détaillé : Partitionnement de textes

## Objectif

Regrouper dans le même sous-ensemble les textes de la collection qui sont le plus similaires

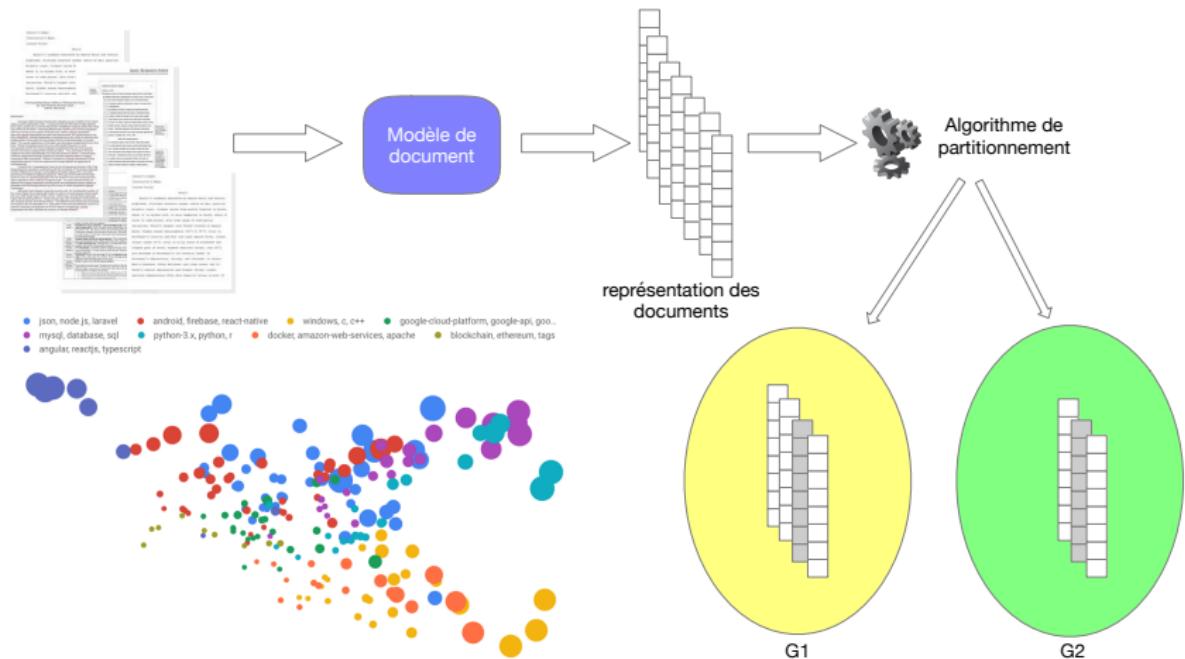
## Usages

- ▶ Obtenir une compréhension globale de la collection
- ▶ Créer une structure pour la navigation (regroupement de résultat de recherche)
- ▶ Lier les textes similaires pour filtrer (synthèse, obtenir un membre caractéristique)

## Tâches

- ▶ Déterminer une bonne représentation des documents
- ▶ Définir un critère de proximité entre documents (similarité)
- ▶ Appliquer un algorithme de partitionnement (K-Means, Clustering Hiérarchique)

# Exemple détaillé : Partitionnement de textes



# Exemple détaillé : Partitionnement de textes

Plusieurs possibilités pour représenter les documents :

- ▶ Représentation ensembliste : Ensemble de mots
- ▶ Représentation vectorielle (la plus commune) : vecteur des fréquences
- ▶ Représentation probabiliste : distribution de thèmes

d =  
... news of presidential campaign ...  
... presidential candidate ...

Représentation ensembliste

$\text{rep}_s(d) = \{\text{news, of, presidential, campaign, candidate, ...}\}$

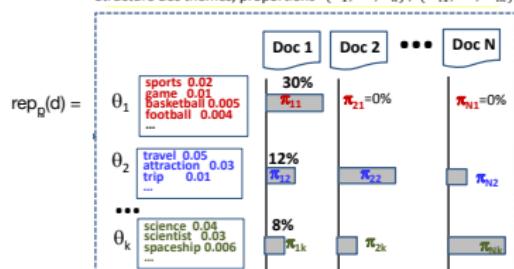
Représentation vectorielle

$\text{rep}_v(d) = \begin{bmatrix} \dots & 1 & 2 & \dots & 1 & 1 & 1 & \dots \end{bmatrix}$

news  
Présidentiel  
campaign  
candidate  
or

Représentation probabiliste

Structure des thèmes, proportions  $\{\theta_1, \dots, \theta_k\}, \{\pi_{11}, \dots, \pi_{Nk}\}$



# Exemple détaillé : Partitionnement de textes

## Proximité entre documents

- ▶ Fonction de la représentation choisie

- ▶ Ensembles :

- ▶ distance de jaccard :  $dist_{jaccard}(S_d, S_{d'}) = 1 - \frac{|S_d \cap S_{d'}|}{|S_d \cup S_{d'}|}$

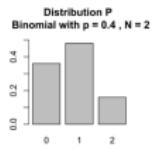
- ▶ Vecteurs :

- ▶ distance cosinus :  $dist_{cosinus}(V_d, V_{d'}) = 1 - \frac{\sum_{i=1}^{|V|} x_{id} x_{i d'}}{\sqrt{(\sum_{k=1}^{|V|} x_{id}^2)(\sum_{k=1}^{|V|} x_{i d'}^2)}}$

- ▶ Distribution de probabilités ( vecteur probabilistes)

- ▶ Divergence de Kullback-Leibler :

$$div\_kullback(P_d, P_{d'}) = \sum_{i=1}^k x_{id} \log\left(\frac{x_{id}}{x_{i d'}}\right)$$



$$\begin{aligned} D_{KL}(P \parallel Q) &= \sum_{x \in \mathcal{X}} P(x) \ln\left(\frac{P(x)}{Q(x)}\right) \\ &= 0.36 \ln\left(\frac{0.36}{0.333}\right) + 0.48 \ln\left(\frac{0.48}{0.333}\right) + 0.16 \ln\left(\frac{0.16}{0.333}\right) \\ &= 0.0852996 \end{aligned}$$

$$\begin{aligned} D_{KL}(Q \parallel P) &= \sum_{x \in \mathcal{X}} Q(x) \ln\left(\frac{Q(x)}{P(x)}\right) \\ &= 0.333 \ln\left(\frac{0.333}{0.36}\right) + 0.333 \ln\left(\frac{0.333}{0.48}\right) + 0.333 \ln\left(\frac{0.333}{0.16}\right) \\ &= 0.097455 \end{aligned}$$

x	0	1	2
Distribution P(x)	0.36	0.48	0.16
Distribution Q(x)	0.333	0.333	0.333

# Exemple détaillé : Partitionnement de documents

Représentation du corpus : matrice Termes x Documents

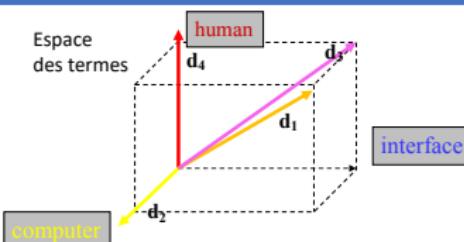
## Articles

- d1: **Human** machine **interface** for ABC **computer** applications
- d2: A **survey** of **user** opinion of **computer system response time**
- d3: The **EPS human interface** management **system**
- d4: **System** and **human system** engineering testing of **EPS**
- d5: Relation of **user** perceived **response time** to error measurement



	d1	d2	d3	d4	d5	d6	d7	d8	d9
human	1	0	1	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	0	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

- d6: The generation of random, binary, ordered **trees**
- d7: The intersection **graph** of paths in **trees**
- d8: **Graph minors** IV: Widths of **trees** and well-quasi-ordering
- d9: **Graph minors: A survey**



## Exemple détaillé : Partitionnement de documents

Pour obtenir un bon partitionnement, les algorithmes de clustering cherchent à :

- ▶ **minimiser la similarité intra-classe**, ie la distance entre les éléments d'une même classe, pour obtenir des clusters les plus homogènes possibles
- ▶ **maximiser la similarité inter-classe**, ie la distance entre les classes, pour obtenir des clusters bien différenciés

La méthode exacte consiste à essayer toutes les partitions possibles (beaucoup trop nombreuses !) :

- ▶ On fait appel à des **méthodes approchées, dites heuristiques**, pour trouver la solution optimale en un temps raisonnable.
- ▶ Principe de ces méthodes :
  - ▶ On commence par une partition initiale.
  - ▶ Puis on itère un processus qui optimise le partitionnement en déplaçant les données d'un cluster à un autre.

# Exemple détaillé : Partitionnement de documents

Deux grandes familles d'algorithmes :

## Méthodes non hiérarchiques

- ▶ Principales approches basées sur :
  - ▶ basé sur des centroïdes : K-means et ses variantes : K-medoids, K-medians...
  - ▶ basé sur des distributions de probabilité : mélange de Gaussiennes, algorithme EM...
  - ▶ la densité locale de points : DBSCAN...

## Méthodes hiérarchiques

- ▶ Principe : Construction d'un dendrogramme, arbre binaire représentant graphiquement les relations d'inclusion entre les classes.
- ▶ Approche ascendante (par agglomération) : Fusion successive des 2 classes les plus proches

# Exemple : L'algorithme K-means

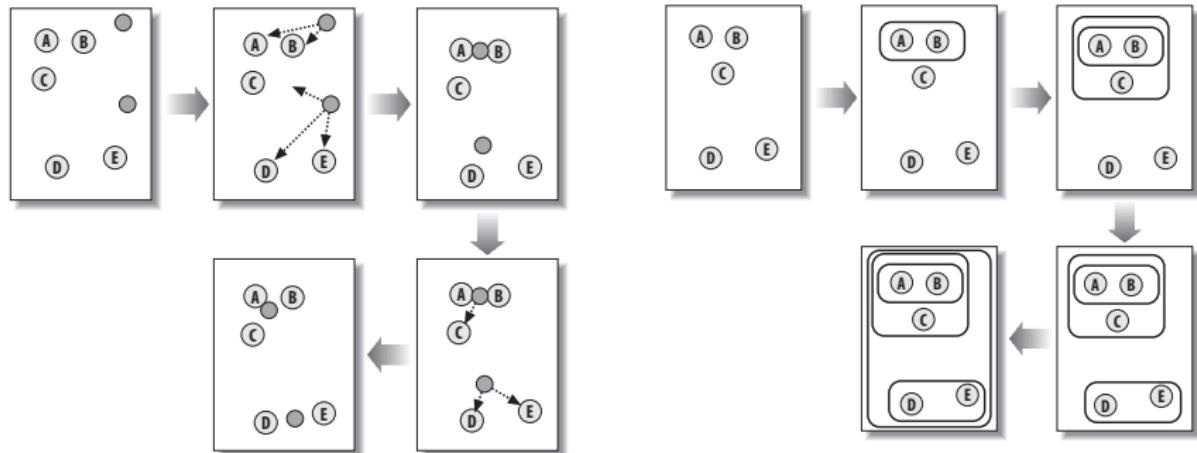
## Algorithme K-means

- ▶ Initialisation : Choix aléatoire des centres  $\mu^1, \dots, \mu^K$  des  $K$  clusters
- ▶ Tant qu'il n'y a pas convergence
  - ▶ Affecter chaque observation  $x^i$  au cluster dont le centre est le plus proche
  - ▶ Mettre à jour le centre de chaque cluster  $\mathcal{C}_k$  en fonction de la nouvelle partition obtenue :  $\mu^k = \frac{1}{\#\mathcal{C}_k} \sum_{x^i \in \mathcal{C}_k} x^i$

Convergence  $\leftrightarrow$  partition stable, plus de changement dans l'attribution des observations aux clusters

- ▶ Simple à coder et interpréter (donc très utilisé)
- ▶ Il faut fixer le nombre de clusters  $K$  a priori
- ▶ L'algorithme peut rester bloqué sur un minimum local

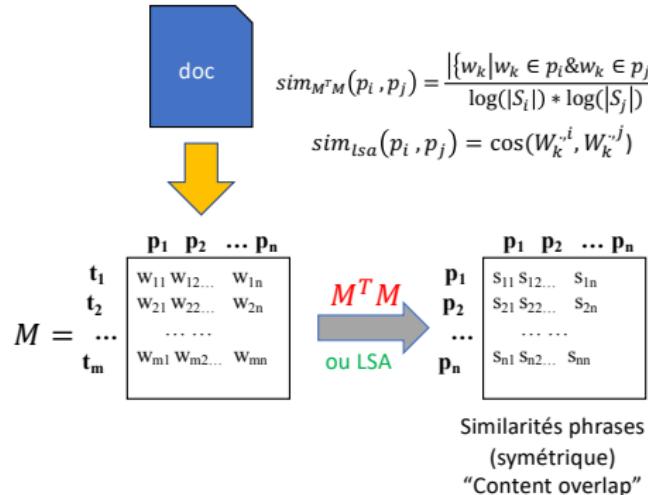
# Exemple détaillé : Partitionnement de documents



Evaluation de la performance d'un algorithme de clustering :

- ▶ Difficile à évaluer car on ne connaît pas les "bons" clusters
- ▶ Critères d'évaluation possibles :
  - ▶ qualitatifs : jugement d'un expert
  - ▶ quantitatifs : rapport de distances intra-cluster et inter-cluster

# Autre application : Résumé de documents

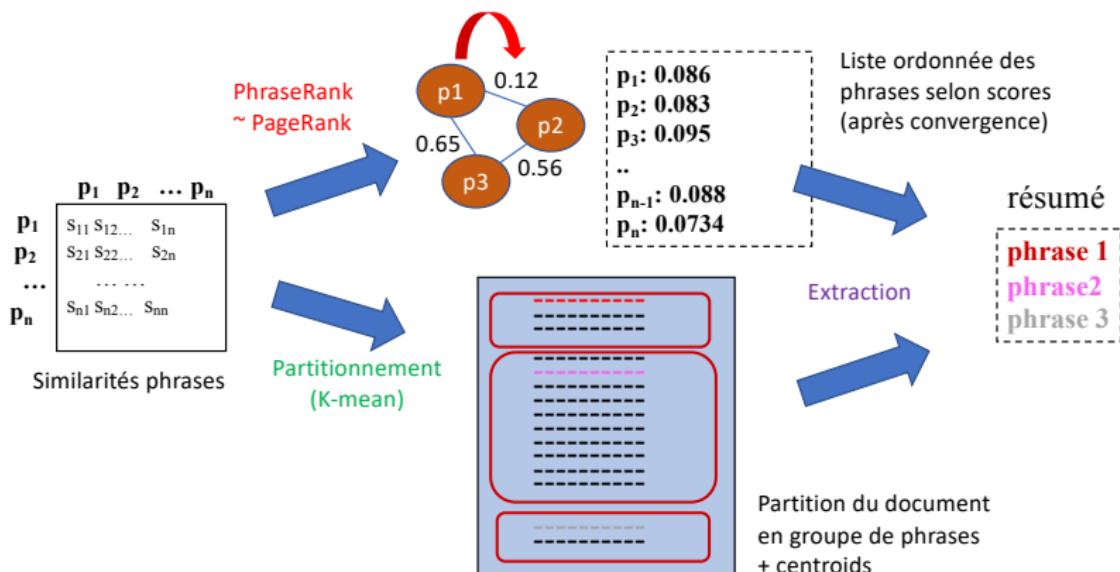


## Résumé de document

- Méthode extractive
- Ordonnancement des phrases (scores)
- Sélection des phrases

## Autre application : Résumé de documents

## Résumé de document



# Plan

Natural Language Processing (NLP)

Apprentissage automatique et NLP

Réseaux de Neurones

Représentations

Apprentissage Profond et NLP

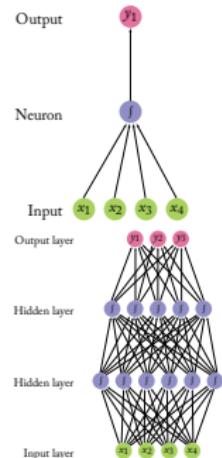
# Réseaux de neurones

## Un neurone formel (Perceptron)

- ▶  $output = activation(\sum_{weight}(input) + bias)$

## Une couche de neurones

- ▶ traitement en parallèle des neurones
- ▶ implantation par un multiplication de matrice-vecteur
- ▶  $y = f(xW + b), x \in R^{d_{in}}, W \in R^{d_{in} \times d_{out}}, b \in R^{d_{out}}$



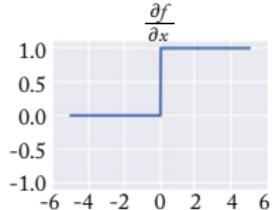
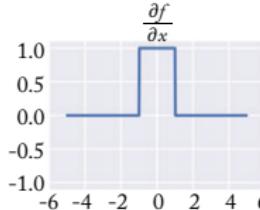
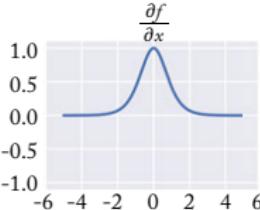
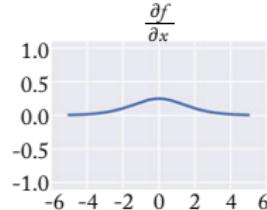
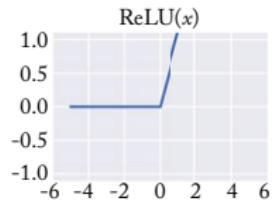
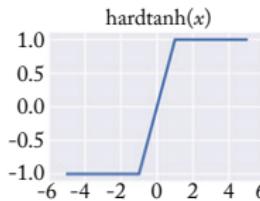
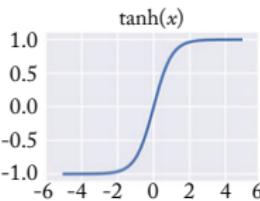
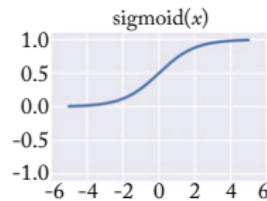
## Perceptron multi-couches

- ▶  $y = NN_{\Theta}(x) = (f_2(f_1(xW_1 + b_1)W_2 + b_2))W_3 + b_3$
- ▶  $\Theta = (W_1, b_1, W_2, b_2, W_3, b_3)$

# Non-linéarité

- ▶ Fonction d'activation
  - ▶ Si  $f$  est linéaire, la composition de fonctions est linéaire
  - ▶ Besoin de non linéarité ( $\tanh, \sigma, \dots$ )
- ▶ Non linéarité
  - ▶ Les réseaux de neurones peuvent approximer n'importe quel fonction continue (Hornik91)
- ▶ Réseaux de neurones profonds
  - ▶ Composition de beaucoup de fonctions non-linéaires
  - ▶ Rapide à calculer et pouvoir d'expression plus puissant que les autres modèles
  - ▶ Mais plus difficile à entraîner

# Fonctions non linéaires



# Apprentissage dans les réseaux de neurones

L'apprentissage se fait de la même manière que dans les modèles linéaires vue précédemment

- ▶ Minimisation de la fonction de perte
- ▶  $\hat{\Theta} = \operatorname{argmin}_{\Theta} L(\Theta) = \operatorname{argmin}_{\Theta} \frac{1}{n} \sum_{i=1}^n L(NN_{\Theta}(x), y_i)$
- ▶ Besoin de minimiser une fonction non linéaire, non convexe
- ▶ Recherche de minima locaux par descente de gradient
- ▶ Différentiation de composition de fonctions
- ▶  $g \circ f(x) = g(f(x))$
- ▶ Chaine Rule :  $\frac{\partial(g \circ f)}{\partial x} = \frac{\partial g}{\partial f} \frac{\partial f}{\partial x}$
- ▶ Généralisation :  $\frac{\partial(f_1 \circ \dots \circ f_n)}{\partial \Theta} = \frac{\partial f_1}{\partial f_2} \dots \frac{\partial f_{n-1}}{\partial f_n} \frac{\partial f_n}{\partial \Theta}$

# Apprentissage dans les réseaux de neurones

## Perceptron avec une couche cachée

- ▶  $L(\Theta) = \frac{1}{n} \sum_{i=1}^n L_{ce}(NN_\Theta(x)), y_i)$
- ▶  $NN_\Theta(x) = z_1(x) = \sigma(z_2(x)W_2) + b_2)$
- ▶  $z_2(x) = \sigma(xW_1 + b)$
- ▶  $\Theta = (W_1, b_1, W_2, b_2)$

## Calcul des gradients pour trouver les minimas locaux

- ▶  $\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial L_{ce}} \frac{\partial L_{ce}}{\partial z_1} \frac{\partial z_1}{\partial W_2}$
- ▶  $\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial L_{ce}} \frac{\partial L_{ce}}{\partial z_1} \frac{\partial z_1}{\partial b_2}$
- ▶  $\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial L_{ce}} \frac{\partial L_{ce}}{\partial z_1} \frac{\partial z_1}{\partial z_2} \frac{\partial z_2}{\partial W_1}$
- ▶  $\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial L_{ce}} \frac{\partial L_{ce}}{\partial z_1} \frac{\partial z_1}{\partial z_2} \frac{\partial z_2}{\partial b_1}$

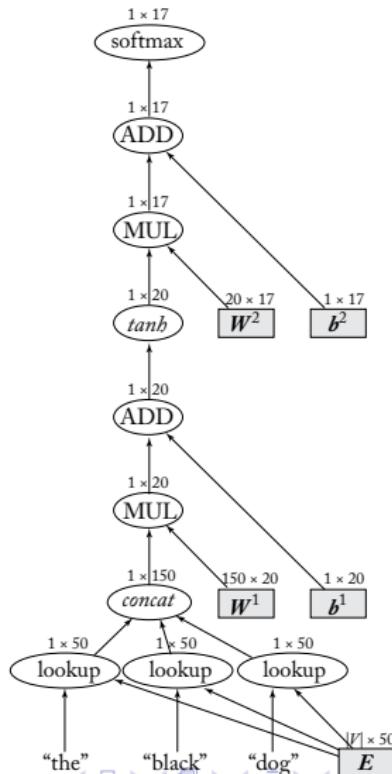
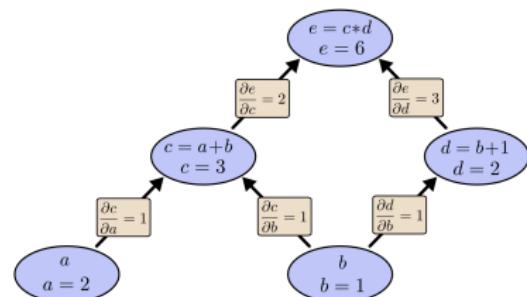
On observe une redondance des calculs !

# Propagation arrière

- ▶ Beaucoup de calculs sont partagés
- ▶ Les calculs des gradients peuvent être vus comme un calcul effectué en arrière dans le réseau
- ▶ Deux phases : forward-propagation et back-propagation
- ▶ Entrainement d'un réseau de neurones
  - ▶  $\Theta_0 = \text{random}$
  - ▶ while not converged
    - ▶ forward :  $L_{\Theta_t}$  (Predict  $\hat{y}$ , Compute Loss)
    - ▶ backward :  $\nabla L_{\Theta_t}$  (Calcul des dérivées partielles)
    - ▶ update :  $\Theta_{t+1} = \Theta_t - \lambda \nabla L_{\Theta_t}$

# Graphes de calculs

Les opérations effectuées dans le réseau peuvent être représentées sous la forme d'un graphe de calculs



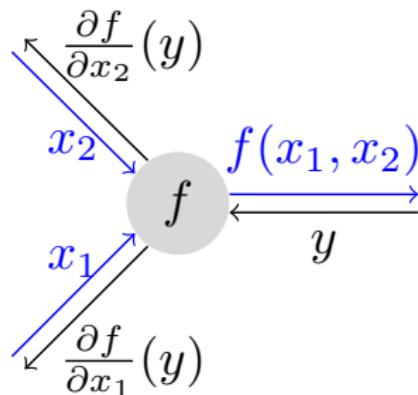
# Bloc de calcul

Les noeuds du graphe de calculs peuvent être assemblé comme des blocs de "légo"

- ▶ Chaque bloc a des entrées, des paramètres et des sorties
- ▶ Examples
  - ▶ Logarithm : forward  $y = \ln(x)$ , backward  $\frac{\partial \ln}{\partial x}(y) = 1/y$
  - ▶ Fonction linéaire
    - ▶ forward  $y = f_{W,b}(x) = W \cdot x + b$
    - ▶ backward  $\frac{\partial f}{\partial x}(y) = y^T \cdot x$ ,  $\frac{\partial f}{\partial W}(y) = y \cdot W$ ,  $\frac{\partial f}{\partial b}(y) = y$
  - ▶ Sum, Product, ...

Les boîtes à outil utilisent des blocs qui font automatiquement les calculs

(propagation avant et arrière)



# Boîte à outils pour les réseaux de neurones

## Bas-niveau

- ▶ Prévu pour prendre en compte les accélérations matérielles
- ▶ Permet de construire n'importe quels graphes de calculs
- ▶ Liaison avec de multiples langages : Java, Python, C++, ...
  - ▶ Tensorflow - <https://www.tensorflow.org> (Google)
  - ▶ mxnet - <http://mxnet.io> (Apache)

## Haut-niveau

- ▶ Généralement construit au-dessus des boîtes bas-niveau
- ▶ Implementation des couches standards
- ▶ Codage d'un réseau en quelques lignes, peu personnalisable
  - ▶ Keras - <http://keras.io>
  - ▶ Tflearn - <http://tflearn.org>

## Les deux

- ▶ Chainer - <http://chainer.org>
- ▶ Pytorch - <http://pytorch.org>



# Comparaison des boîtes à outils

## Framework Comparison: Basic information\*

Viewpoint	Torch.nn**	Theano***	Caffe	autograd (NumPy, Torch)	Chainer	MXNet	Tensor- Flow
GitHub stars	4,719	3,457	9,590	N: 654 T: 554	1,295	3,316	20,981
Started from	2002	2008	2013	2015	2015	2015	2015
Open issues/PRs	97/26	525/105	407/204	N: 9/0 T: 3/1	95/25	271/18	330/33
Main developers	Facebook, Twitter, Google, etc.	Université de Montréal	BVLC (U.C. Berkeley)	N: HIPS (Harvard Univ.) T: Twitter	Preferred Networks	DMLC	Google
Core languages	C/Lua	C/Python	C++	Python/Lua	Python	C++	C++/Python
Supported languages	Lua	Python	C++/Python MATLAB	Python/Lua	Python	C++/Python R/Julia/Go etc.	C++/Python

\* Data was taken on Apr. 12, 2016

\*\* Includes statistics of Torch7

# Comparaison des boîtes à outils

## Framework Comparison: Design Choices

Design Choice	Torch.nn	Theano	Caffe	Chainer	MXNet	Tensor-Flow	PyTorch
NN definition	Script (Lua)	Script* (Python)	Data (protobuf)	Script (Python)	Script (many)	Script (Python)	Script (Python)
Backprop	Through graph	Extended graph	Through graph	Through graph	Through graph	Extended graph	Through graph
Parameters	Hidden in operators	Separate nodes	Hidden in operators	Separate nodes	Separate nodes	Separate nodes	Separate nodes
Update formula	Outside of graphs	Part of graphs	Outside of graphs	Outside of graphs	Outside of graphs	Part of graphs	Outside of graphs
Graph construction	Static	Static	Static	Dynamic	Static	Static	Dynamic
Graph Optimization	-	Supported	-	-	-	Supported	-
Parallel computation	Multi GPU*	Multi GPU*	Multi GPU*	Multi GPU**	Multi node Multi GPU	Multi node Multi GPU	Multi GPU**

\* Third-party multi-node implementations exist

\*\* Planned to release multi-node training support

† Keras has the same capability as its

backend (Theano or TensorFlow)

# Plan

Natural Language Processing (NLP)

Apprentissage automatique et NLP

Réseaux de Neurones

Représentations

Apprentissage Profond et NLP

# Caractéristiques pour les problèmes NLP

## Caractéristiques pour les mots

### Caractéristiques lexicales

- ▶ taille du mot
- ▶ éléments constituant le mot : nombre de voyelles / consommes, n-gram de caractères, capitalisation, préfixe, suffixe, ...

### Caractéristiques sémantiques

- ▶ synonymes, antonymes
- ▶ hyponymes (+ général), hyperonymes (+ spécifiques)
- ▶ meronym (composé de), holonym (fait partie de)
- ▶ ressources : dictionnaires, thésaurus (Wordnet)

The noun “bass” has 8 senses in WordNet.

1. bass<sup>1</sup> - (the lowest part of the musical range)
2. bass<sup>2</sup>, bass part<sup>1</sup> - (the lowest part in polyphonic music)
3. bass<sup>3</sup>, basso<sup>1</sup> - (an adult male singer with the lowest voice)
4. sea bass<sup>1</sup>, bass<sup>4</sup> - (the lean flesh of a saltwater fish of the family Serranidae)
5. freshwater bass<sup>1</sup>, bass<sup>5</sup> - (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
6. bass<sup>6</sup>, bass voice<sup>1</sup>, basso<sup>2</sup> - (the lowest adult male singing voice)
7. bass<sup>7</sup> - (the member with the lowest range of a family of musical instruments)
8. bass<sup>8</sup> - (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

Relation	Also Called	Definition	Example
Hypernym	Superordinate	From concepts to superordinates	<i>breakfast</i> <sup>1</sup> → <i>meal</i> <sup>1</sup>
Hyponym	Subordinate	From concepts to subtypes	<i>meal</i> <sup>1</sup> → <i>lunch</i> <sup>1</sup>
Instance Hypernym	Instance	From instances to their concepts	<i>Austen</i> <sup>1</sup> → <i>author</i> <sup>1</sup>
Instance Hyponym	Has-Instance	From concepts to their instances	<i>composer</i> <sup>1</sup> → <i>Bach</i> <sup>1</sup>
Part Meronym	Has-Part	From wholes to parts	<i>table</i> <sup>2</sup> → <i>leg</i> <sup>3</sup>
Part Holonym	Part-Of	From parts to wholes	<i>course</i> <sup>7</sup> → <i>meal</i> <sup>1</sup>
Antonym		Semantic opposition between lemmas	<i>leader</i> <sup>1</sup> ↔ <i>follower</i> <sup>1</sup>
Derivation		Lemmas w/same morphological root	<i>destruction</i> <sup>1</sup> ↔ <i>destroy</i> <sup>1</sup>



# Caractéristiques pour les problèmes NLP

## Caractéristiques pour les mots

### Caractéristiques contextuelles

- ▶ contexte : fenêtre de mots, phrase, paragraphe, document
- ▶ statistiques : nombre d'occurrences dans le contexte, ...
- ▶ linguistiques
  - ▶ type : nom, verbe, adjectif, adverbe, ...
  - ▶ fonction du mot : sujet, verbe, cod
- ▶ distributionnelles : mots précédents, mots suivants, position des mots ...

Les caractéristiques distributionnelles des mots sont très utilisées aujourd'hui

### Hypothèse de sémantique distributionnelle

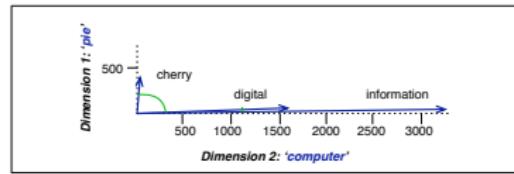
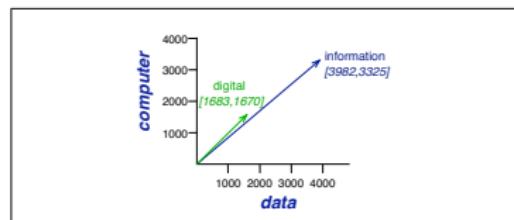
- ▶ Firth (1957), Harris (1954)
- ▶ La signification d'un mot dépend de sa compagnie
- ▶ Des mots qui apparaissent dans des contextes similaires (c-a-d avec des ensembles de mots proches) ont une signification similaire

# Caractéristiques pour les problèmes NLP

## Caractéristiques distributionnelles

is traditionally followed by **cherry** pie, a traditional dessert  
often mixed, such as **strawberry** rhubarb pie. Apple pie  
computer peripherals and personal **digital** assistants. These devices usually  
a computer. This includes **information** available on the internet

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	
strawberry	0	...	0	0	1	60	19	
digital	0	...	1670	1683	85	5	4	
information	0	...	3325	3982	378	5	13	



# Caractéristiques pour les problèmes NLP

Caractéristiques pour les textes (phrase, paragraphe, document)

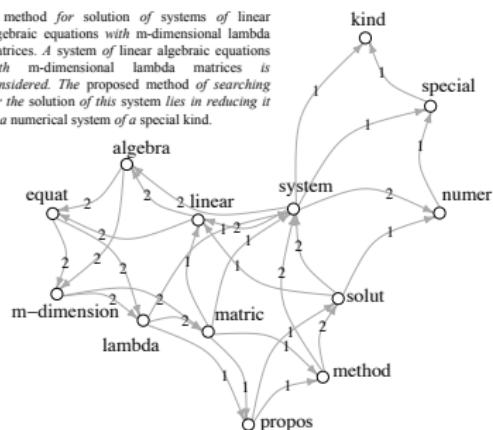
## Caractéristiques observés

- ▶ ensemble des mots du texte
- ▶ comptage des mots, poids des mots
- ▶ séquence de mots consécutifs : n-grams, colocalisation

## Caractéristiques inférées

- ▶ relations entre mots :
  - ▶ arbre syntaxique
  - ▶ arbre de dépendances
  - ▶ graphe de colocation

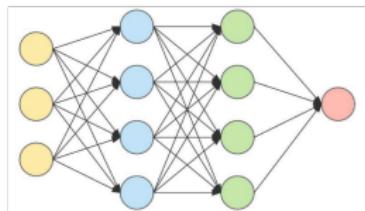
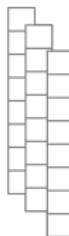
*A method for solution of systems of linear algebraic equations with m-dimensional lambda matrices. A system of linear algebraic equations with m-dimensional lambda matrices is considered. The proposed method of searching for the solution of this system lies in reducing it to a numerical system of a special kind.*



# Des caractéristiques NLP aux entrées

Date collected	Plot	Species	Sex	Weight
1/9/78	1	DM	M	40
1/9/78	1	DM	F	36
1/9/78	1	DS	F	135
1/20/78	1	DM	F	39
1/20/78	2	DM	M	43
1/20/78	2	DS	F	144
3/13/78	2	DM	F	51
3/13/78	2	DM	F	44
3/13/78	2	DS	F	146

transformation



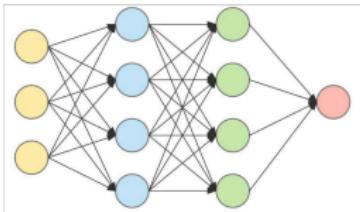
UNITED STATES

Obama tried to give Zuckerberg a wake-up call over fake news on Facebook

The Washington Post - By Adam Entous, Elizabeth...  
Economy September 25, 2017 at 7:45 AM Facebook CEO Mark Zuckerberg's company recently said it would turn over to Congress more than 3,000 politically themed

transformation

?



# Des caractéristiques NLP aux entrées

Les entrées des méthodes d'apprentissage sont généralement des vecteurs ou des matrices

Comment passer des caractéristiques à des représentations vectorielles ?

- ▶ Signification des dimensions ?
- ▶ Contenu des dimensions ?

La représentation vectorielle doit :

- ▶ être facile à manipuler
- ▶ préserver l'information utile : distance faible entre vecteurs  $\approx$  entités similaires
- ▶ limiter le coût de stockage et de calcul

# Représentations vectorielles

## Signification des dimensions

### Plusieurs possibilités

- ▶ Caractéristiques observées ou inférées (identifiées par ingénierie)
- ▶ Caractéristiques latentes (obtenues de façon automatique, non-interprétable)
  - ▶ Par calcul algébrique : Réduction Dimension
  - ▶ Par méthode d'apprentissage : Representation Learning
- ▶ Combinaison de plusieurs caractéristiques

# Représentations vectorielles

Contenu des dimensions pour des caractéristiques observées ou inférées

## Plusieurs possibilités

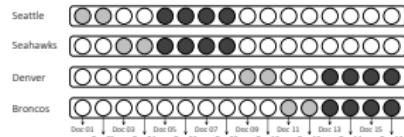
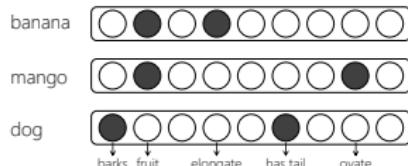
En prenant  $w$  un mot in  $V$  et  $c$  une caractéristique contextuelle dans  $C$  : autre mot  $w'$ , document  $d$ , ...

- ▶ valeur indicatrice (0 : absence, 1 : présence)
- ▶ valeur de comptage
  - ▶ comptage brute :  $\#_c(w)$
  - ▶ comptage normalisé (TF) :  $\frac{\#_c(w)}{\sum_{w' \in c} \#_c w'}$
  - ▶ comptage pondéré (TF-IDF) :  $\frac{\#_c(w)}{\sum_{w' \in c} \#_c w'} \log \frac{|C|}{|\{c \in C : w \in c\}|}$
- ▶ probabilités (estimation par comptage #) :
  - ▶ PPMI (Information mutuelle positive) :  
$$PPMI(w, c) = \max(\log\left(\frac{P(w, c)}{P(w)P(c)}\right), 0)$$
 (on ne tient pas compte des paires  $(w, c)$  telles que  $(w, c) = 0$  ou  $P(c|w) < P(c)$ )
  - ▶ smoothed PPMI (avec paramètre  $\alpha$ ) :  $PPMI_\alpha(w, c) = \log\left(\frac{P(w, c)}{P(w)P^\alpha(c)}\right)$   
où  $P^\alpha(c) = \frac{\#(c)^\alpha}{\sum c \#(c)^\alpha}$

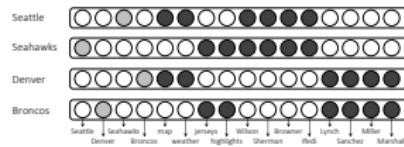
# Encodage One-Hot

Dans cette encodage chaque dimension correspond à une caractéristique distincte et indépendante des autres

Encodage one-hot de mots (Dimensions)

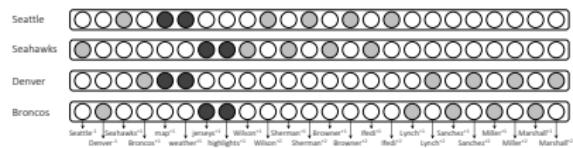


(a) “In-documents” features



(b) “Neighbouring terms” features

Sample documents	
doc 01	Seattle map
doc 02	Seattle weather
doc 03	Seahawks jerseys
doc 04	Seahawks highlights
doc 05	Seattle Seahawks Wilson
doc 06	Seattle Seahawks Sherman
doc 07	Seattle Seahawks Browner
doc 08	Seattle Seahawks Ifedi
doc 09	Denver map
doc 10	Denver weather
doc 11	Broncos jerseys
doc 12	Broncos highlights
doc 13	Denver Broncos Lynch
doc 14	Denver Broncos Sanchez
doc 15	Denver Broncos Miller
doc 16	Denver Broncos Marshall



(c) “Neighbouring terms w/ distances” features



# Encodage One-Hot

## Encodage one-hot de mots (Valeurs)

### Comptage

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

### Probabilités

$$P(w=information, c=data) = \frac{3982}{11716} = .3399$$

$$P(w=information) = \frac{7703}{11716} = .6575$$

$$P(c=data) = \frac{5673}{11716} = .4842$$

$$\text{ppmi}(information, data) = \log 2(.3399 / (.6575 * .4842)) = .0944$$

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

### PPMI

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

## Similarité avec l'encodage One-Hot

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\vec{v}}{|\vec{v}|} \cdot \frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

	large	data	computer
apricot	1	0	0
digital	0	1	2
information	1	6	1

Quelle paire de mots est la + similaire ?

cosine(apricot,information) =

$$\frac{1+0+0}{\sqrt{1+0+0} \sqrt{1+36+1}} = \frac{1}{\sqrt{38}} = .16$$

cosine(digital,information) =

$$\frac{0+6+2}{\sqrt{0+1+4} \sqrt{1+36+1}} = \frac{8}{\sqrt{38}\sqrt{5}} = .58$$

cosine(apricot,digital) =

$$\frac{0+0+0}{\sqrt{1+0+0} \sqrt{0+1+4}} = 0$$

$v_i$  le nombre d'occurrences du mot  $v$  dans le contexte  $i$   
 $w_i$  le nombre d'occurrences du mot  $v$  dans le contexte  $i$

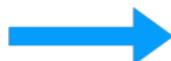


## Encodage One-Hot

### Encodage one-hot de phrases (ou documents)

Document 1	Love, love is a verb
Document 2	Love is a doing word
Document 3	Feathers on my breath
Document 4	Gentle impulsion
Document 5	Shakes me, makes me lighter
Document 6	Feathers on my breath

	id	word															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Document 1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	
Document 2	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0	1	
Document 3	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	
Document 4	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
Document 5	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0	0	
Document 6	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	



Document 1	[Love, love, is a verb]
Document 2	[Love, is, a, doing, word]
Document 3	[Feathers, on, my, breath]
Document 4	[Gentle, impulsion]
Document 5	[Shakes, me, makes, me lighter]
Document 6	[Feathers, on, my, breath]

a	breath	doing	feathers
gentle	impulsion	is	lighter
love	makes	me	my
on	shakes	verb	word

# Problèmes de l'encodage One-Hot

L'encodage One-Hot est facile à construire mais conduit à plusieurs problèmes :

## Problèmes

- ▶ Les vecteurs sont généralement de très grande taille : vocabulaire de 20000 mots => 20000 dimensions
- ▶ Le cout mémoire est très importante : la plupart des valeurs sont égales à 0
- ▶ Des mots synonymes ont des représentations différentes :
  - ▶ *voiture* et *automobile* sont synonymes mais correspondent à des dimensions distinctes
  - ▶ un mot avec *voiture* comme voisin et un mot avec *automobile* comme voisin devrait être similaire mais ce n'est pas le cas

# Vers des vecteurs de taille réduite

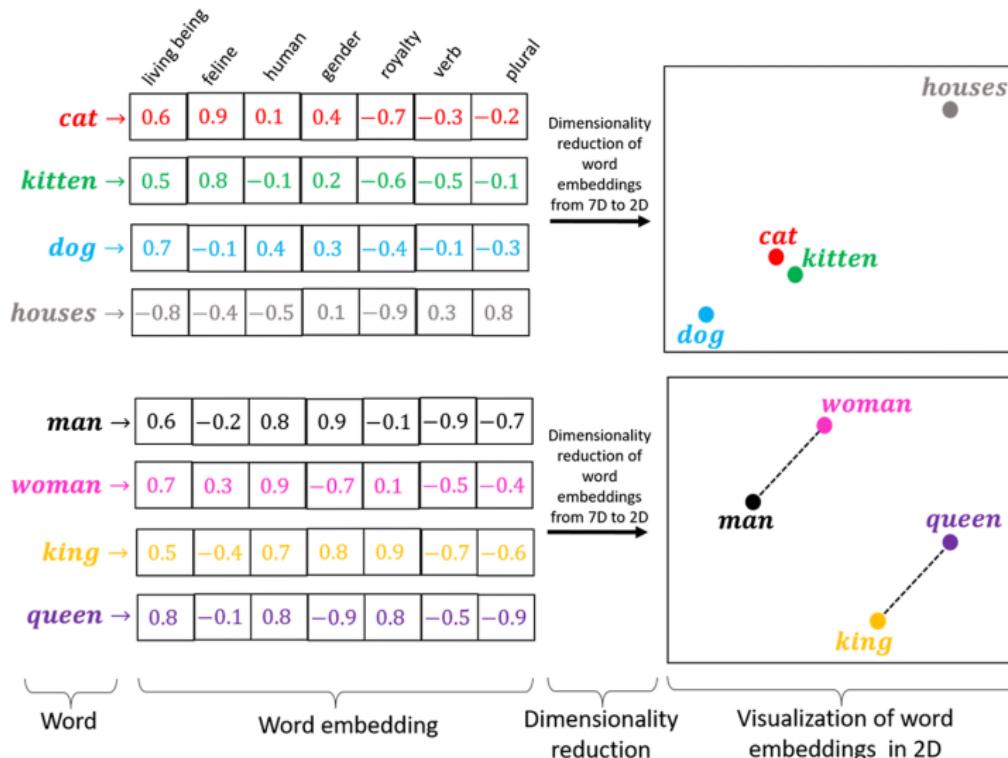
Il existe plusieurs solutions pour obtenir des vecteurs de taille réduite et qui préserve tant que possible les informations lexicales, syntaxiques, sémantiques des mots

## Solutions

- ▶ Faire une sélection des caractéristiques
- ▶ Réduire le nombre de dimensions :
  - ▶ Regrouper plusieurs caractéristiques en une seule : clustering des mots et choix du clustering comme caractéristique
  - ▶ Déterminer de nouvelles caractéristiques qui capturent la similarité des contextes
  - ▶ Par méthode de calcul algébrique : PCA, LSA, Random Indexing
  - ▶ Par méthode d'apprentissage : Word2vec, Glove, FastText, ...
  - ▶ On laisse l'algorithme comprendre les corrélations entre mots et généraliser avec les statistiques

# Encodage Dense

## Schematisation



## Encodage Dense : Analyse de sémantique latente (LSA)

A partir d'une matrice  $X$  des mots ( $t_i$ ) et leurs contextes (ici document  $d_j$ ), le principe est de construire une approximation de  $X$  au rank  $l$  à l'aide d'une méthode de décomposition SVD tel que  $X_l = U\Sigma V$

- ▶  $U_i$  est une représentation du mot  $i$  (dans un espace sémantique et de dimension  $l$ ) utilisable pour :
  - ▶ comparer la similarité de deux mots relativement au contexte
  - ▶ trouver les relations de synomynie et de polysémie entre les mots
  - ▶ regrouper les mots qui sont similaires avec du clustering

$$(t_i^T) \rightarrow \begin{matrix} X \\ (\mathbf{d}_j) \\ \downarrow \\ \left[ \begin{matrix} x_{1,1} & \dots & x_{1,j} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i,1} & \dots & x_{i,j} & \dots & x_{i,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,j} & \dots & x_{m,n} \end{matrix} \right] \end{matrix} = (\hat{\mathbf{t}}_i^T) \rightarrow \begin{matrix} U \\ \downarrow \\ \left[ \begin{matrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_l \end{matrix} \right] \dots \left[ \begin{matrix} \mathbf{u}_l \end{matrix} \right] \end{matrix} \cdot \begin{matrix} \Sigma \\ \downarrow \\ \left[ \begin{matrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_l \end{matrix} \right] \end{matrix} \cdot \begin{matrix} V^T \\ (\hat{\mathbf{d}}_j) \\ \downarrow \\ \left[ \begin{matrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_l \end{matrix} \right] \end{matrix}$$

## Encodage dense : Random Indexing

- ▶ Association à chaque mot d'un vecteur  $v$  One-Hot aléatoire de dimension réduite  $m$ 
  - ▶ Par ex : 4 composantes non nulles dans un vecteur de dimension 300
  - ▶ Il est peu probable que deux mots  $i$  et  $j$  possèdent le même vecteur de représentation
- ▶ Creation d'une matrice  $M$  de cooccurrence de taille  $(|vocab|, m)$
- ▶ Mise à jour de la matrice  $M$  en lisant le texte :
  - ▶ Quand le mot  $i$  co-occure avec le mot  $j$ , ajout du vecteur  $v_j$  à la ligne  $m_i$
- ▶ La matrice  $M$  est une approximation de taille réduite de la matrice de cooccurrence réel
- ▶ Après normalisation, la ligne  $m_i$  peut être utilisée comme nouvelle représentation du mot  $i$

John eats a **red** apple

$$\begin{aligned}V_{\text{john}} &\rightarrow (0,0,0,0,0,0,1,0,-1,0) \\V_{\text{eat}} &\rightarrow (1,0,0,0,-1,0,0,0,0,0) \\V_{\text{apple}} &\rightarrow (0,1,0,0,0,0,1,0,0,-1)\end{aligned}$$

$$\begin{aligned}M_{\text{red}} = V^{-2}_{\text{john}} + V^{-1}_{\text{john}} + V^{+1}_{\text{apple}} = \\(0,0,0,0,1,0,-1,0,0,0) \\+ (0,0,0,-1,0,0,0,0,0,1) \\+ (-1,0,1,0,0,0,0,0,0,1) \\= (-1,0,1,-1,1,0,-1,0,0,1)\end{aligned}$$

## Encodage dense : Apprentissage

L'idée d'utiliser l'apprentissage provient des travaux sur les "Neural Language Model" de Bengio(2003) et Collobert (2011)

La tâche est d'apprendre à prédire le prochain mot à partir des mots précédents

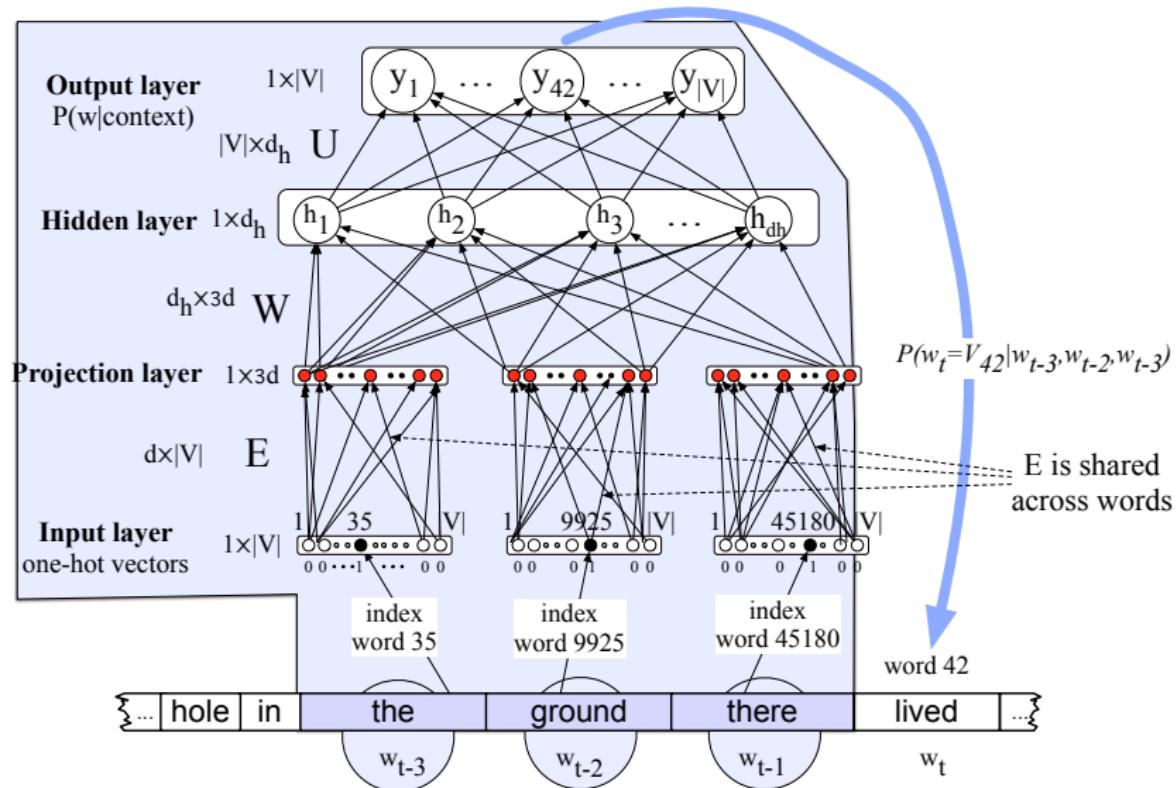
Intuitivement, soit un mot  $s$  près du mot *abricot* :

- ▶  $s$  peut être considéré comme une réponse correcte à la question :
- ▶ "Est-ce que le mot  $w$  est très probable près d'*abricot*"
- ▶ tâche de classification binaire : "predict rather than count"

Cela revient à implicitement à prendre le texte en entrée comme des données d'apprentissage supervisé

Ces travaux ont montré qu'on peut apprendre une représentation vectorielle dense des mots en appliquant cette tâche de prédiction

# Encodage Dense : Apprentissage



# Encodage Dense : Word2Vec

Word2Vec est une simplification très efficace de l'approche précédente proposée par Mikolov (2013)

Deux tâches :

- ▶ Continous Bag-of-Word (CBOW) : Etant donné des mots appartenant à une fenêtre de mots, prédire le mot central de celle-ci
- ▶ Skip-Gram : Etant donné le mot central dans un fenêtre, prédire les autres mots de celle-ci

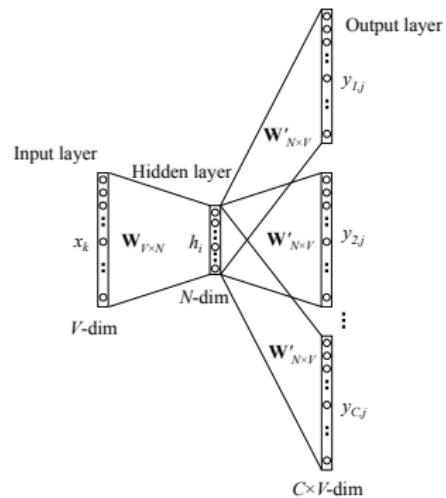
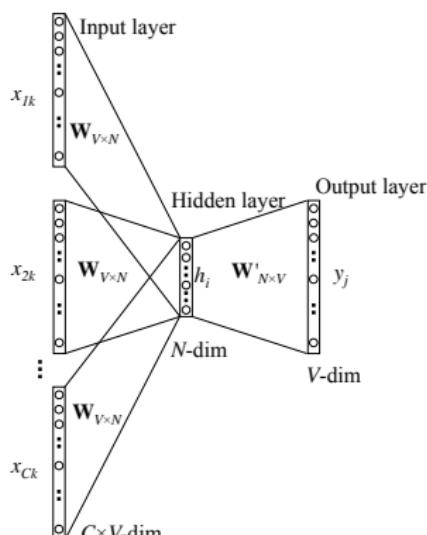


Figure 3: The skip-gram model.

# Encodage Dense : Word2Vec

L'objectif de Word2Vec est d'apprendre à distinguer les paires de mots vraisemblables des paires improbables en utilisant un corpus de textes

Par exemple : *(abricot, jambon)* est plus vraisemblable que  
*(abricot, aardvark)* dans un corpus

## Algorithme

1. Traiter le mot cible et les mot voisin dans le contexte comme des exemples positifs
2. Prendre de façon aléatoire dans le vocabulaire des mots comme des exemples négatifs
3. Utiliser la régression logistique pour entraîner un classifieur à distinguer les deux cas
4. Utiliser les paramètres appris pour obtenir les représentations vectorielles des mots

Remarque : plus le corpus de texte est grand, plus les vecteurs obtenus seront fiables

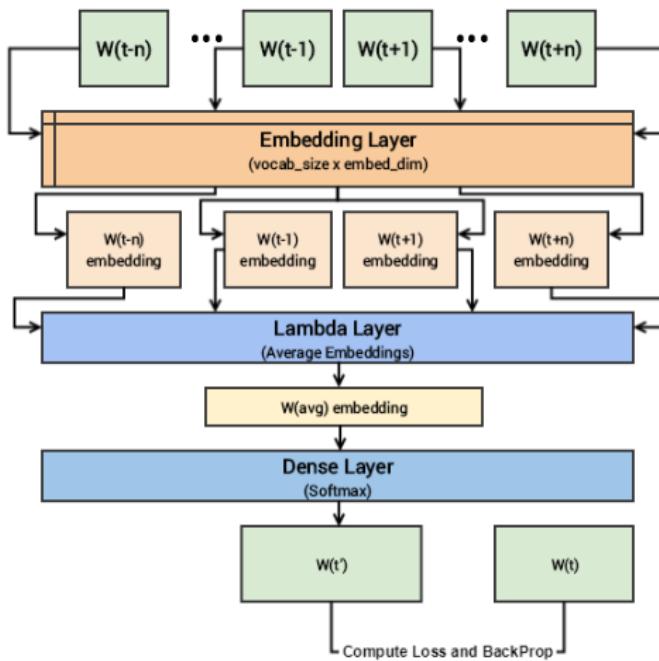
# Encodage Dense : Word2Vec

## Formalisation (Negative Sampling Objective)

- ▶ Soit un ensemble  $D$  de paires (mot  $w$ , mot du contexte  $c$ ) corrects et un ensemble  $\bar{D}$  de paires incorrectes
- ▶ On estime la probabilité que  $(w, c)$  soit une paire correcte par
$$P(D = 1|w, c) = \frac{1}{1 + \exp(-(w \cdot c))}$$
- ▶ L'objectif est de maximiser le log de vraisemblance sur les données  $D \cup \bar{D}$  :
- ▶  $\mathcal{L}(\Theta; D; \bar{D}) = \sum_{(w, c) \in D} \log P(D = 1|w, c) + \sum_{(w, c) \in \bar{D}} \log P(D = 0|w, c)$
- ▶ Les paires de  $D$  sont générées à partir du corpus et celles de  $\bar{D}$  sont déterminées aléatoirement en tenant compte de leur fréquence dans le corpus :  $\frac{\#(c)}{\sum_{c'} \#(c')}$
- ▶ Dans Word2Vec, la formule de probabilité est étendue à plusieurs mots du contexte de taille  $k$  (moyennage des vecteurs)
- ▶  $P(D = 1|w, c_{1:k}) = \frac{1}{1 + \exp(-(w \cdot c_1 + w \cdot c_2 + \dots + w \cdot c_k))}$

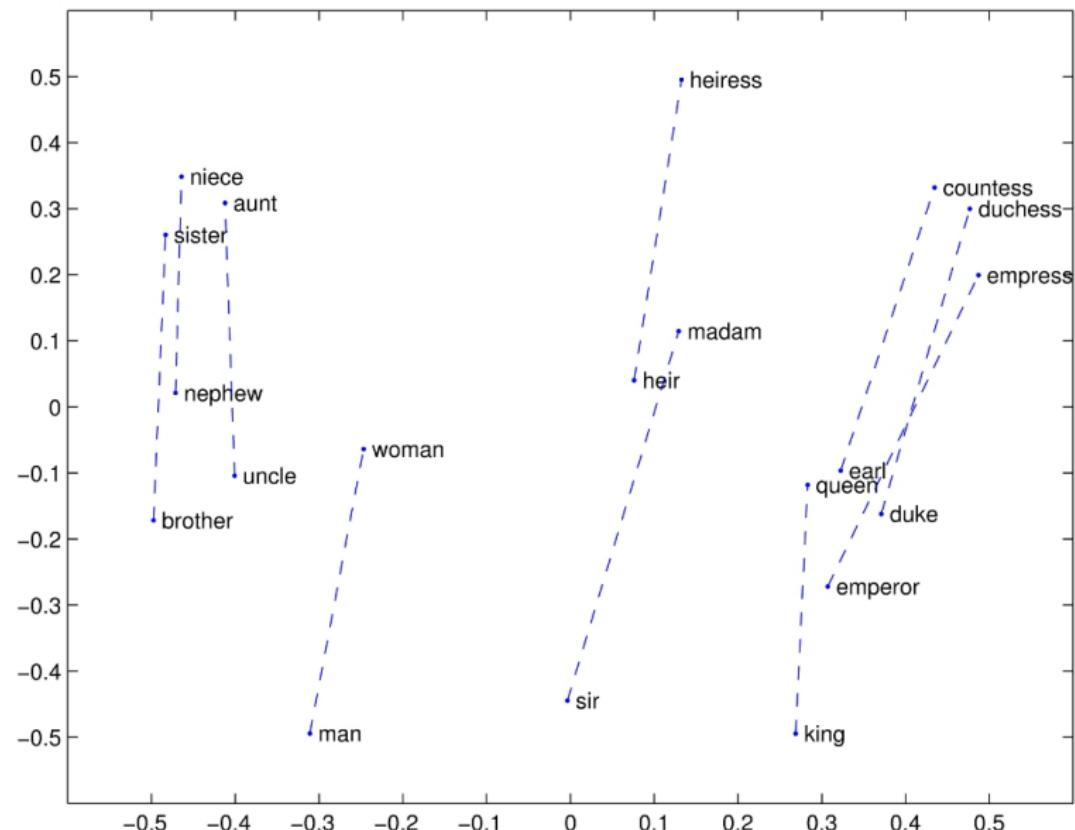
# Encodage Dense : Word2Vec

## Architecture détaillée



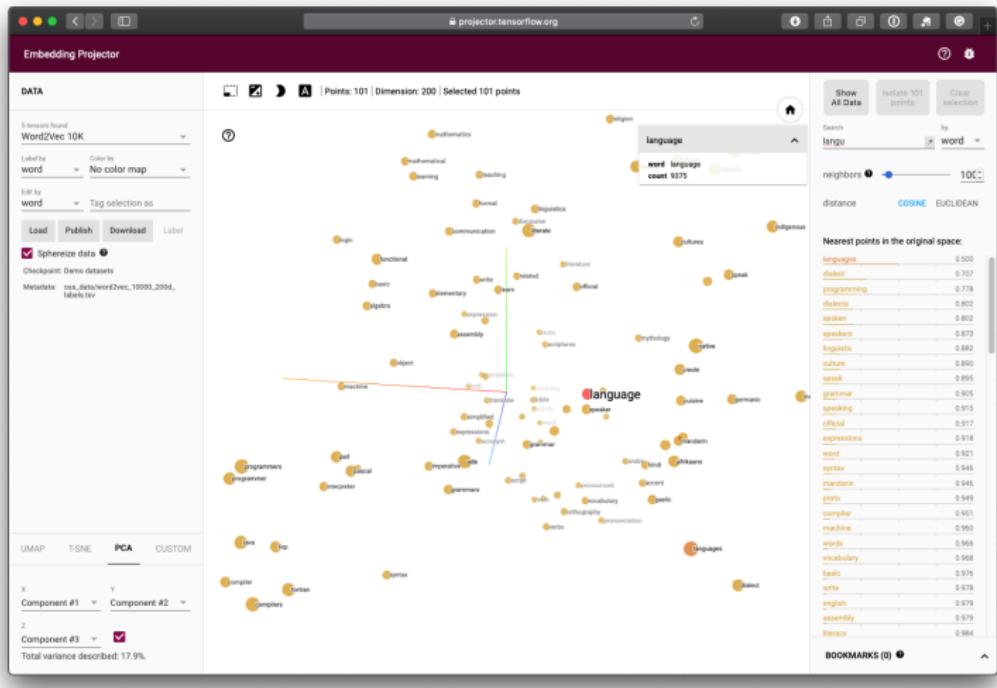
# Encodage Dense : Word2Vec

Exemple de proximité des mots (papier)



## Visualisation de vecteurs

## Demo : Embedding projector



# Encodage Dense : Word2Vec

Plusieurs régularités avec les vecteurs Word2Vec ont été observées :

## Inflection

- ▶ Pluriel, genre
- ▶ Superlatif
- ▶ Temps des verbes

## Relations sémantiques

- ▶ Capitale / Pays
- ▶ Chef / Group
- ▶ Analogies

## Relations linéaires

- ▶  $\text{vector('king')} - \text{vector('man')} + \text{vector('woman')} \approx \text{vector('queen')}$
- ▶  $\text{vector('Paris')} - \text{vector('France')} + \text{vector('Italy')} \approx \text{vector('Rome')}$

Il a aussi été montré que les vecteurs peuvent comporter des biais (culturels, ethniques, ...)

# Evaluation des vecteurs de mots

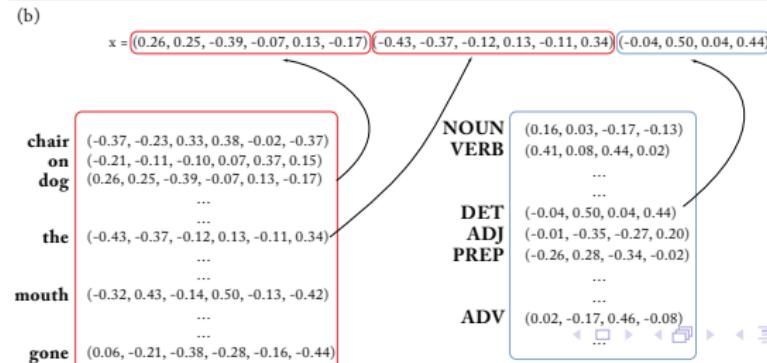
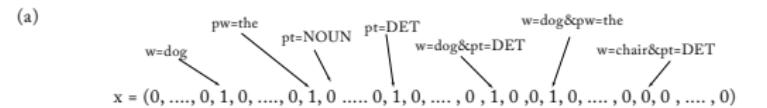
- ▶ Qu'est ce qu'un bon encodage de mots en vecteur dense ?
  - ▶ Les vecteurs doivent être suffisamment généraux pour être réutilisables
  - ▶ Les vecteurs doivent capturer des propriétés linguistiques
  - ▶ Les vecteurs doivent rendre compte de similarité "sémantique" entre les mots
  - ▶ Les biais liés au corpus doivent être limités
- ▶ Propriétés linguistiques
  - ▶ Les vecteurs doivent restituer des relations sémantiques comme celles des Wordnet
  - ▶ Les vecteurs doivent permettre des analogies
- ▶ Propriétés psychologiques
  - ▶ Les vecteurs doivent être en phase avec le sens attendu par l'humain

# Combinaisons de vecteurs denses

Plusieurs vecteurs denses correspondant à informations distinctes peuvent être combinés pour former les entrées

## Principaux modes de combinaisons

- ▶ Exemple : context du mot b : a et c
- ▶ Somme :  $v(a) + v(b) + v(c)$
- ▶ Somme pondérés :  $\frac{1}{2}v(a) + v(b) + \frac{1}{2}v(c)$
- ▶ Concaténation :  $[v(a); v(b); v(c)]$



## Choix du contexte

Le choix du contexte pour apprendre la représentation d'un mot a un effet important sur le vecteur résultat et les similarités qu'il encode

- ▶ fenêtre glissante d'une séquence de mot :  $2m + 1$  (m mots avant, m mots après, 1 mot focus)
- ▶ utilisation de la position des mots dans le contexte
- ▶ affectation de poids différents selon la position
- ▶ lemmatisation des mots avant et après
- ▶ taille dynamique de la fenêtre : la phrase, le paragraphe comme context
- ▶ utilisation des mots proches dans l'arbre de dépendance comme contexte (Dependency Embeddings)

# Vecteurs de mots et réseaux de neurones

La première couche d'un réseau est souvent dédiée à l'encodage des caractéristiques observées en vecteurs denses (embedding layer)

- ▶ Apprentissage de la représentation dense en même temps que la tâche par propagation arrière
- ▶ Encodage en réutilisant des vecteurs denses *pré-entraînés* indépendamment de la tâche : Modèle de langue

## Encode One-Hot vers Encodage Dense

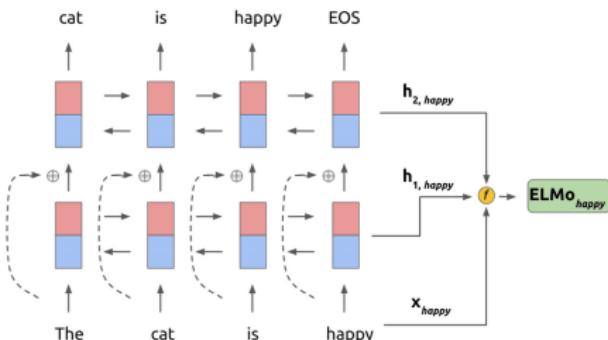
- ▶ Soit un vocabulaire de  $|V|$  mots, chaque mot ayant une représentation dense de dimension  $d$
- ▶ Collection de vecteurs dense de mots : une matrice  $E$  de taille  $|V| \times d$
- ▶ Chaque ligne de la matrice correspond à un mot du vocabulaire
- ▶ Soit  $w_i$  la représentation one-hot du  $i$ ème mot du vocabulaire  $V$
- ▶ Avec cette matrice  $E$ , la multiplication  $w_i E$  sélectionne le vecteur dense correspondant :  $v(w_i) = w_i E$
- ▶  $v(w_i)$  est donné en entrée du réseau
- ▶ Important : prévoir un vecteur dense pour les mots non inclus dans le vocabulaire : symbol **UNK**

# Encodage Dense : Autres approches

Un problème de Word2vec est de toujours associer la même représentation à chaque mot, peu importe le contexte  
En général, le sens d'un mot dépend de son contexte

## ELMO

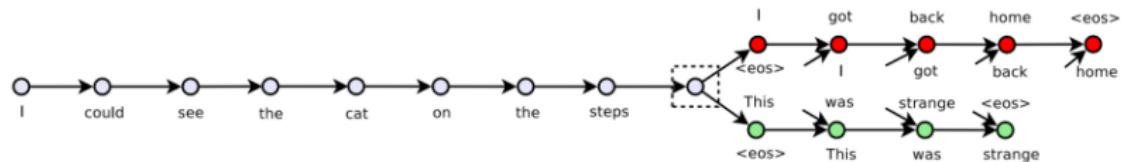
- ▶ Pour un mot, ELMO construit des représentations différentes selon le contexte
- ▶ Modèle de langue : prédiction du prochain mot
- ▶ Empilement de plusieurs couches LSTM bidirectionnel (forward et backward model)
- ▶ Représentation en concaténant le vecteur d'entrée et les états des couches cachés



# Encode Dense : représentation de phrases, paragraphes, documents

## Skip-Thought vectors

- ▶ Entrainement du réseau à générer la phrase précédente et phrase suivante de chaque phrase
- ▶ Encoder-Decoder
- ▶ Les phrases qui apparaissent dans le même contexte auront une représentation dense similaire



## Doc2vec

# Plan

Natural Language Processing (NLP)

Apprentissage automatique et NLP

Réseaux de Neurones

Représentations

Apprentissage Profond et NLP

# Apprentissage Profond

L'apprentissage profond est un sous-domaine de l'apprentissage automatique qui s'appuie généralement sur les réseaux de neurones  
Ce type d'apprentissage se caractérise par :

La construction de réseaux de neurones avec un grand nombre de couches

Jeff Dean : "When you hear the term deep learning, just think of a large deep neural net. Deep refers to the number of layers typically and so this kind of the popular term that's been adopted in the press. I think of them as deep neural networks generally."

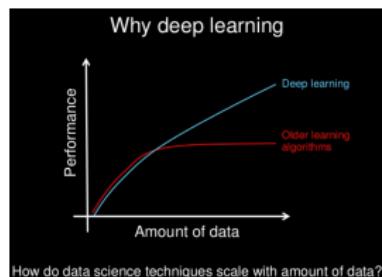
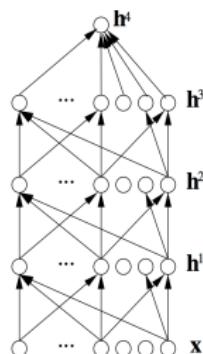
La capacité à apprendre des représentations hiérarchiques efficaces à partir des données

Yoashuo BenGio : "Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Automatically learning features at multiple levels of abstraction allow a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features."

# Apprentissage Profond

La capacité à passer à l'échelle et à améliorer les résultats quand le volume de donnée augmente

Andrew Ng : "... for most flavors of the old generations of learning algorithms ... performance will plateau. ... deep learning ... is the first class of algorithms ... that is scalable. ... performance just keeps getting better as you feed them more data"



## Important Property of Neural Networks

Results get better with

more data +  
bigger models +  
more computation

(Better algorithms, new insights and improved techniques always help, too!)



# Promesses de l'apprentissage profond appliqu  au NLP

L'apprentissage profond appliqu  au NLP :

$$DeepNLP = DeepLearning + NLP$$

Plusieurs promesses de cette application :

- ▶ Le remplacement b n fique des mod les existants
  - ▶ Les r sultats obtenus sur des probl mes classiques du NLP ont surpass  les mod les (lin aires) existants
  - ▶ capacit    apprendre des relations non lin aires entre les donn es
- ▶ L'apprentissage automatique des caract ristiques
  - ▶ Les caract ristiques int ressantes des donn es sont apprises automatiquement au lieu d' tre sp cifi es et extraites par un expert (souvent trop sp cifiques, incompl tes, longues   concevoir et   valider)

# Promesses de l'apprentissage profond au NLP

- ▶ Une amélioration continue des résultats
  - ▶ Les performances des architectures, la précision des résultats ne cessent de progresser depuis le début
- ▶ Des modèles de bout en bout
  - ▶ Au lieu de développer des pipelines de modèles spécialisés, on peut développer et former des modèles de bout en bout pour les problèmes de langage naturel
- ▶ Des blocs réutilisables pour construire des modèles nouveaux
  - Les couches standards sont composable (Embedding, CNN, LSTM) pour construire des architectures plus sophistiquées (CNN+LSTM, BiLSTM, ...)

# Architectures d'apprentissage profond pour le NLP

On rencontre souvent ces principes :

- ▶ Entrée
  - ▶ La séquence de mots est généralement transformée en séquence de vecteurs de mots
  - ▶ Les vecteurs de mots peuvent être issues d'un modèle pré-entraîné ou appris par l'architecture
  - ▶ On intègre ainsi les similarités sémantiques entre les mots
- ▶ Composants CNN, RNN, ...
  - ▶ Ces composants sont principalement utilisés comme extracteur de caractéristiques : il capture les aspects de l'entrée utile à la tâche : les n-grams les plus informatifs, les régularités dans les séquences
  - ▶ Ceux ne sont pas des composants autonomes : leurs résultats (vecteur ou séquence de vecteurs) sont introduits dans d'autres parties de l'architecture pour aboutir à des prédictions ou des générations
  - ▶ On peut les combiner, mélanger et assortir pour former une architecture de bout en bout et obtenir le comportement souhaité

## Réseaux convolutionnels pour le NLP

En général, l'ordre des mots joue un rôle important dans la compréhension du texte

- ▶ "It was not good, it was actually quite bad"
- ▶ "It avoids the obvious with humour and lightness"

Les représentations type sac de mots ou n-grams ne sont pas satisfaisantes pour interpréter correctement ces phrases

- ▶ Sac de mots : "It was not good, it was actually quite bad" = "It was not bad, it was actually quite good"
- ▶ L'utilisation des n-grams produit des matrices trop larges et trop ésparses

Les architectures de réseaux convolutionnels sont taillées pour ce type de problème :

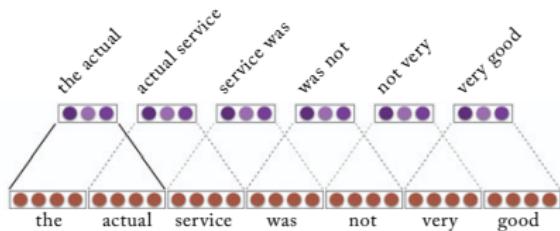
- ▶ ces réseaux identifient les n-grams de mots qui sont utiles à la tâche (sans besoin d'avoir besoin de spécifier des vecteurs denses pour chaque n-gram possible)
- ▶ ces réseaux permettent de partager le comportement prédictif des n-grams qui ont des constituants communs
- ▶ On parle de ces réseaux comme des "n-grams detectors"

## Principes

- ▶ Application d'une fonction non linéaire sur chaque fenêtre de mots du texte ayant une taille  $k$
- ▶ Application de filtres pour convertir chaque fenêtre de  $k$ -mots en une valeur réelle et obtenir un vecteur des résultats
  - ▶ Chaque dimension correspond à un filtre
  - ▶ Le vecteur capture les propriétés importantes des mots dans la fenêtre
- ▶ Application d'une opération (pooling) pour combiner les vecteurs des différentes fenêtres en un nouveau vecteur
  - ▶ Utilisation du max ou de la moyenne
  - ▶ Recherche des caractéristiques les plus importantes dans le texte sans tenir compte de leur localisation
- ▶ Transmission du vecteur au reste du réseau pour faire la prédiction
  - ▶ Les gradients des pertes sont propagés en arrière pour adapter les paramètres de filtres

## Réseaux convolutionnels pour le NLP

## Notion de fenêtre de mots



## Notion de filtre

I like this movie very much!

0.6	0.5	0.2	-0.1	0.4
0.8	0.9	0.1	0.5	0.1
0.4	0.6	0.1	-0.1	0.7
---	---	---	---	---
---	---	---	---	---
---	---	---	---	---
---	---	---	---	---

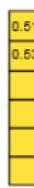
0.2	0.1	0.2	0.1	0.1
0.1	0.1	0.4	0.1	0.1



I  
like  
this  
movie  
very  
much  
!

0.6	0.5	0.2	-0.1	0.4
0.8	0.9	0.1	0.5	0.1
0.4	0.6	0.1	-0.1	0.7
---	---	---	---	---
---	---	---	---	---
---	---	---	---	---
---	---	---	---	---

0.2	0.1	0.2	0.1	0.1
0.1	0.1	0.4	0.1	0.1



## Formalisation de convolution 1D de taille $k$

- ▶ Soit un séquence de mots  $w_{1:n} = w_1, \dots, w_n$  avec un vecteur dense correspondant  $E_{[w_i]} = \mathbf{w}_i$
- ▶ On utilise l'opérator de concaténation  $\oplus(\mathbf{w}_{i:i+k-1})$
- ▶ Le vecteur concaténé de la  $i$ ème fenêtre est  
 $x_i = \oplus(\mathbf{w}_{i:i+k-1}) = [\mathbf{w}_i; \mathbf{w}_{i+1}; \dots : \mathbf{w}_{i+k-1}],$
- ▶ L'application d'un filtre  $u$  à chaque vecteur fenêtre a pour résultat une valeur  $p_i \in \mathcal{R}$ 
  - ▶  $p_i = g(x_i \cdot u)$ ,  $p_i \in \mathcal{R}$ ,  $\mathbf{x}_i \in \mathcal{R}^{k \cdot d_{emb}}$ ,  $u \in \mathcal{R}^{k \cdot d_{emb}}$
  - ▶  $g$  est une fonction d'activation non linéaire
- ▶ Il est fréquent d'utiliser  $\ell$  filtres  $u_1, \dots, u_\ell$  qui peuvent être agencés dans une matrice  $\mathbf{U}$  et un vecteur de biais  $\mathbf{b}$
- ▶ On obtient ainsi un vecteur  $p_i$  représentant la  $i$ ème fenêtre
  - ▶  $\mathbf{p}_i = g(x_i \cdot \mathbf{U} + \mathbf{b})$
  - ▶  $\mathbf{p}_i \in \mathcal{R}^\ell$ ,  $\mathbf{U} \in \mathcal{R}^{k \cdot d_{emb} \times \ell}$ ,  $\mathbf{b} \in \mathcal{R}^\ell$

# Réseaux convolutionnels pour le NLP

L'application de la convolution sur le texte produit  $m$  vecteurs  $\mathbf{p}_{1:m}$  chaque  $\mathbf{p}_i \in \mathcal{R}^\ell$

Ces vecteurs sont combinés (pooled) en un seul vecteur  $c \in \mathcal{R}^\ell$  représentant le texte et condensant l'information importante de celui-ci (n-grams informatifs d'un sujet, d'un sentiment par ex.)

## Max Pooling

- ▶ Valeur max de chaque dimension :  $c_{[j]} = \max_{1 < i < n} \mathbf{p}_{i[j]} \forall j \in [1, l]$

## Average Pooling

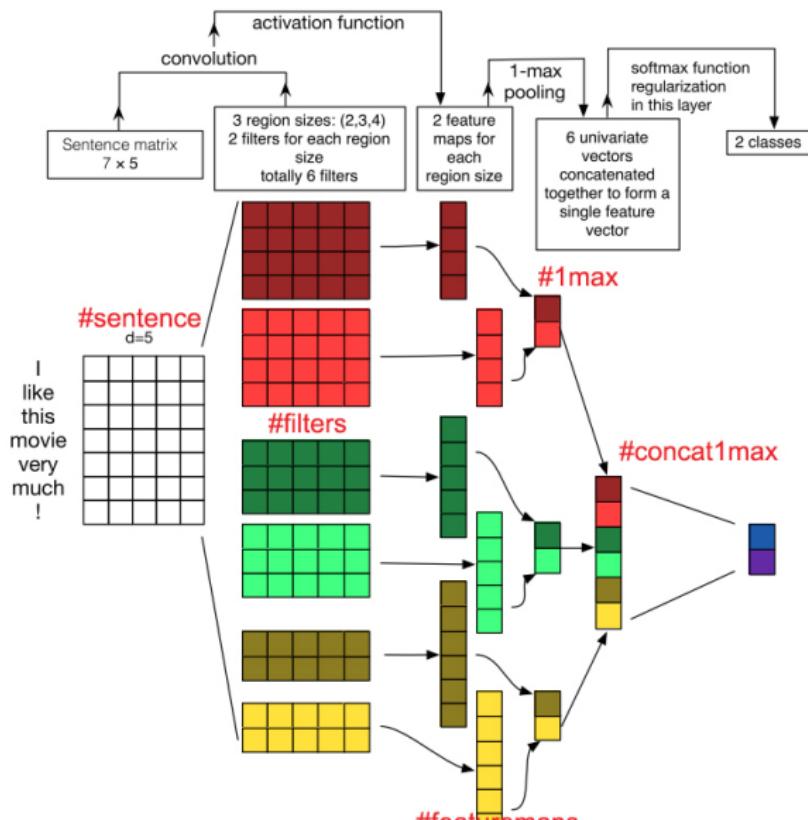
- ▶ Valeur moyenne de chaque indice :  $c = \frac{1}{m} \sum_{i=1}^m \mathbf{p}_i$

## K-max pooling

- ▶ K valeurs les plus élevées dans chaque dimension :  $\begin{bmatrix} 1 & 8 & 3 \\ 9 & 6 & 5 \end{bmatrix}$

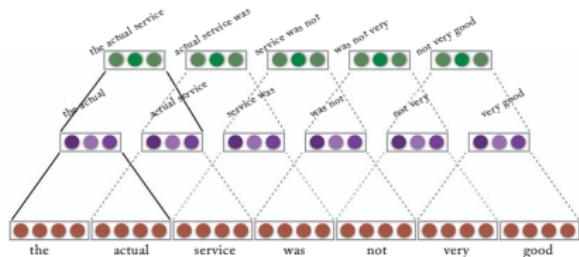
# Réseaux convolutionnels pour le NLP

Exemple d'architecture CNN pour la classification de documents :  
[Chang2015]

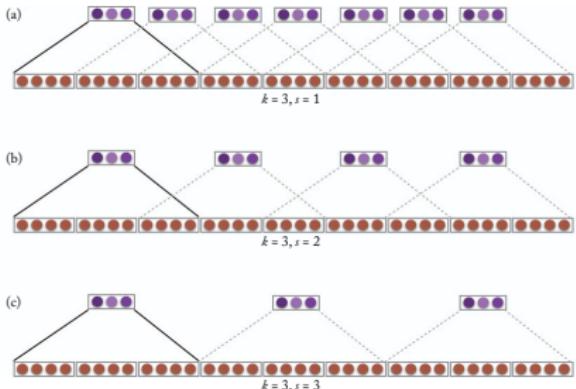


# Réseaux convolutionnels pour le NLP

## CNN Hiérarchique



## CNN avec strides



# Réseaux récurrents pour le NLP

Les réseaux CNN peuvent prendre en compte des séquences très locales mais ne sont pas efficaces pour capturer les caractéristiques globales d'une séquence de données

- ▶ mots : séquence des caractères, phrase : séquence des mots, documents : séquence des phrases

Les réseaux récurrents ont été conçus pour répondre à cette problématique

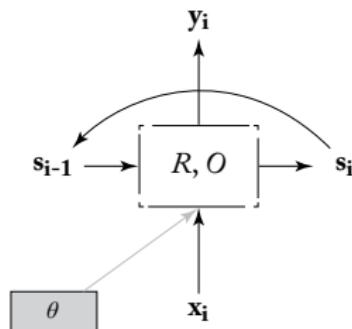
- ▶ Ils sont capables de représenter des séquences par un vecteur
- ▶ Ils sont capables de capturer des régularités dans des données séquentielles

En NLP, leurs applications sont multiples et ont permis d'atteindre de très bonnes performances sur des prédictions concernant des séquences

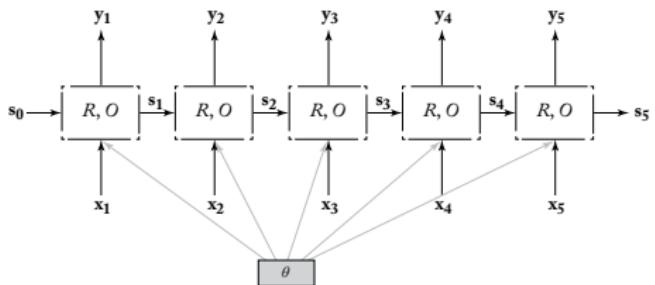
- ▶ Prédiction de la prochaine valeur de la séquence : Modèle de langue
- ▶ Classification de la séquence : Analyse de sentiment
- ▶ Génération de séquences : Génération de textes
- ▶ Prédiction de séquences à partir d'autres : Traduction de texte

# Réseaux récurrents pour le NLP

Représentation graphique  
réursive



Représentation graphique dépliée



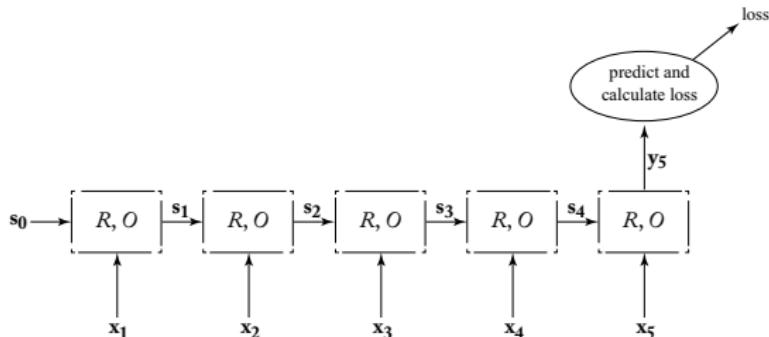
- ▶ Les fonctions  $R$  et  $O$  sont les mêmes pour toutes les positions de la séquence
- ▶ Différentes instantiations de  $R$  et  $O$  conduisent à des structures et des propriétés différentes

## Formalisation

- ▶ Soit une séquence de  $n$  vecteurs de dimension  $d_{in}$  :  $\mathbf{x}_{1:n} = \mathbf{x}_1, \dots, \mathbf{x}_n$
- ▶ On désigne par RNN une fonction qui prend en entrée une telle séquence et retourne un seul vecteur  $y_n$  de dimension  $d_{out}$  :
- ▶  $y_n = RNN(\mathbf{x}_{1:n})$ ,  $\mathbf{x}_i \in \mathcal{R}^{d_{in}}$  et  $y_i \in \mathcal{R}^{d_{out}}$
- ▶ Pour chaque préfixe  $\mathbf{x}_{1:i}$  de la séquence, cela définit implicitement un vecteur de sortie  $y_i$
- ▶ On désigne par  $RNN^*$  la fonction retournant cette séquence
- ▶  $y_{1:n} = RNN^*(\mathbf{x}_{1:n})$  avec  $y_i = RNN(\mathbf{x}_{1:i})$
- ▶ La fonction  $RNN^*$  peut être formulée de la façon suivante
  - ▶  $RNN^*(\mathbf{x}_{1:i}; \mathbf{s}0) = \mathbf{y}_{1:n}$
  - ▶  $y_i = O(s_i)$
  - ▶  $s_i = R(s_{i-1}, \mathbf{x}_i)$
- ▶  $\mathbf{x}_i \in \mathcal{R}^{d_{in}}$ ,  $y_i \in \mathcal{R}^{d_{out}}$ ,

# Réseaux récurrents pour le NLP

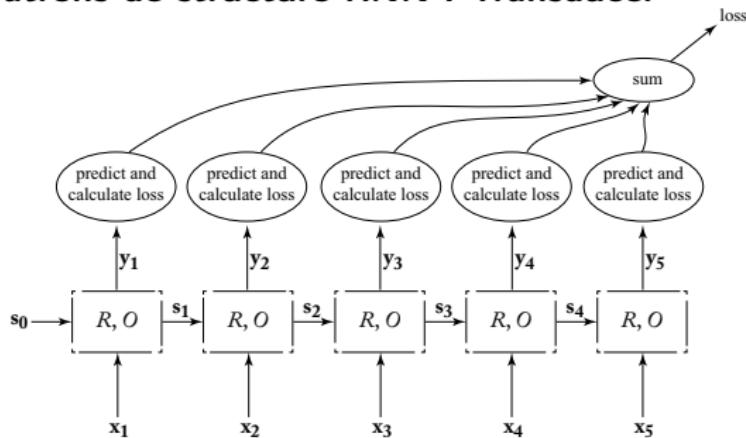
## Patrons de structure RNN : Acceptor et Encoder



- ▶ Acceptor : Prédiction à partir de la sortie
  - ▶  $p(\text{label} = k | w_{1:n}) = \hat{y}_{[k]}$
  - ▶  $\hat{y} = \text{softmax}(\text{MLP}(\text{RNN}(x_{1:n}))) = \text{softmax}(\text{RNN}(x) \cdot W + b)$
  - ▶  $\hat{y}_{[k]} = \text{softmax}(\text{RNN}(x) \cdot W + b)_{[k]}$
  - ▶  $x_{1:n} = E_{[w_1]} \dots E_{[w_n]}$ ,
  - ▶ Exemple : Classification de séquence de mots (Analyse sentiment), Modèle de langue (Prédiction du mot suivant)
- ▶ Encoder : Utilisation de la sortie comme encodage de la séquence et comme information supplémentaire avec d'autres signaux
  - ▶  $\hat{y} = \text{RNN}(x_{1:n})[n]$

# Réseaux récurrents pour le NLP

## Patrons de structure RNN : Transducer

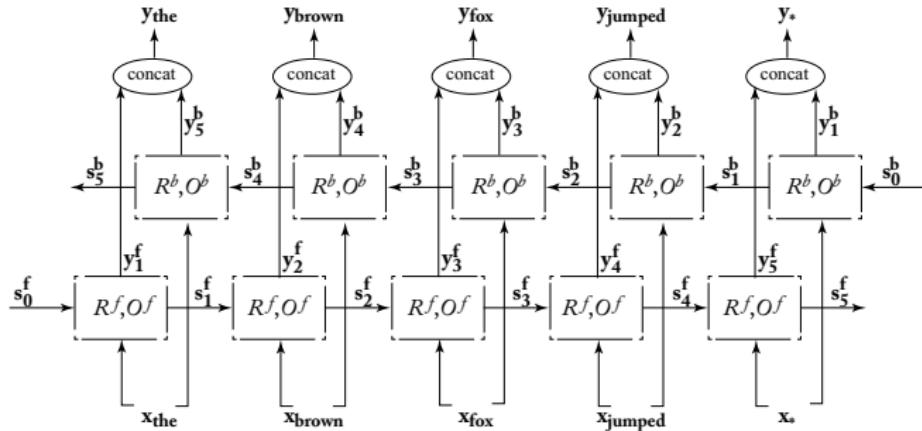


### ► Transducer : Prédiction

- ▶  $p(t_i = k | w_{1:n}) = \hat{y}_{i,[k]}$
- ▶  $\hat{y}_i = softmax(MLP(RNN(x_{1:i}, i)))$
- ▶  $\mathcal{L}(\hat{t}_{i:n}, t_{i:n}) = \sum_{i=1}^n \mathcal{L}_{local}(\hat{t}_{i:n}, t_{i:n})$  (ou une autre combinaison : moyenne, somme pondérée)
- ▶ Exemples : Etiquettagement de séquence de mots (POS tag, Entités nommés)

# Réseaux récurrents pour le NLP

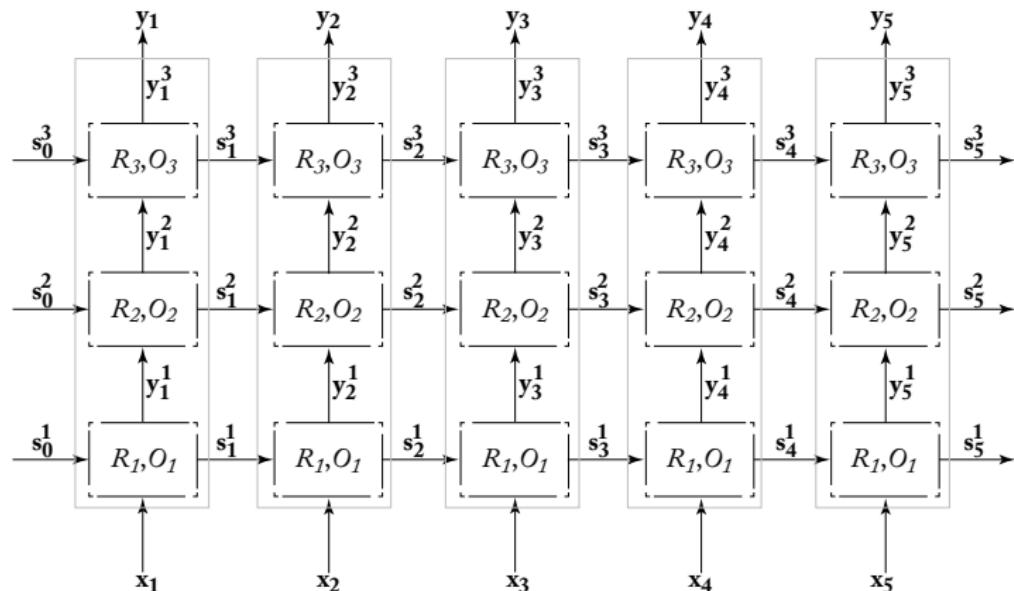
## Patrons de structure RNN : biDirectional RNNs



- ▶ La sortie à la position  $i$  est basée sur la concaténation de deux vecteurs de sortie représentant le passé (*backward*) et le futur (*forward*) de la séquence
- ▶  $biRNN(\mathbf{x}_{1:i}; i) = \mathbf{y}_i = [RNN^f(\mathbf{x}_{1:i}); RNN^b(\mathbf{x}_{n:i})]$
- ▶  $biRNN^*(\mathbf{x}_{1:n}) = \mathbf{y}_{1:n} = biRNN(\mathbf{x}_{1:n}; 1), \dots, biRNN(\mathbf{x}_{1:n}; n)$

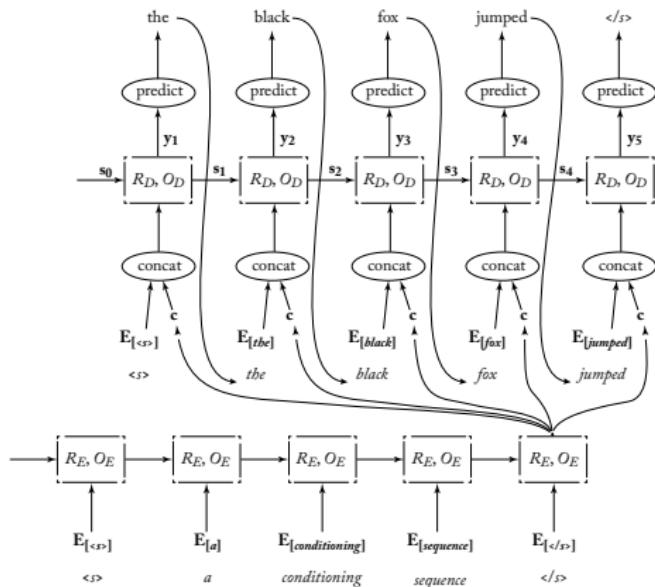
# Réseaux récurrents pour le NLP

**Patrons de structure RNN : Multi-layer (Stacked) RNNs**



# Réseaux récurrents pour le NLP

## Patrons de structure RNN : Encoder-Decoder (Sequence2Sequences)



- ▶  $c = RNN(x_{1:n})$ , encodage de la séquence source (forme de résumé)
- ▶ Génération de la séquence conditionné par  $c$
- ▶ La séquence source et la séquence générée peuvent être de longueur différente
- ▶ Exemple : Traduction d'une phrase en anglais vers le français

## Instantiations concrètes de réseaux RNN

- ▶ On attribue des définitions concrètes aux fonctions  $R$  et  $O$  du RNN
- ▶ Simple RNN
- ▶ Architecture à base de portes (gate) pour contrôler la mémorisation des états :
  - ▶ Long Short Term Memory (LSTM)
  - ▶ Gate Recurrent Unit (GRU)

## Simple RNN (Elman RNN)

- ▶  $\mathbf{s}_i = R_{sRNN}(\mathbf{x}_i, \mathbf{s}_{i-1}) = g(\mathbf{s}_{i-1}W^s + \mathbf{x}_iW^x + b)$
- ▶  $\mathbf{y}_i = O_{sRNN}(\mathbf{s}_i) = \mathbf{s}_i$
- ▶  $\mathbf{s}_i, \mathbf{y}_i \in \mathcal{R}^{d_s}, \mathbf{x}_i \in \mathcal{R}^{d_x}, W^x \in \mathcal{R}^{d_x \times d_s}, W^s \in \mathcal{R}^{d_s \times d_s}, \mathbf{b} \in \mathcal{R}^{d_s}$

# Réseaux récurrents pour le NLP

Les réseaux de type Simple RNN sont difficiles à entraîner à cause du "vanishing gradients problem"

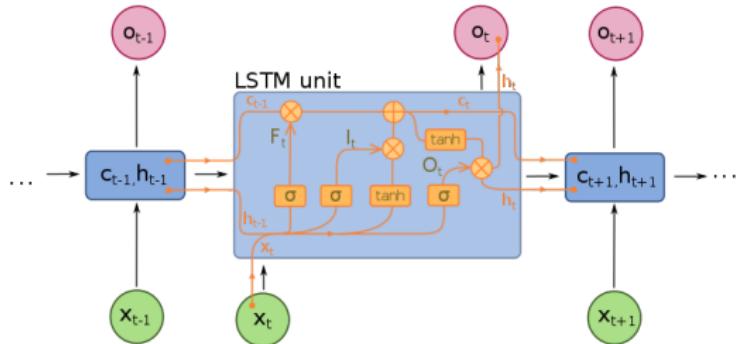
- ▶ Les erreurs dans les dernières étapes de la séquence diminuent rapidement en propagation arrière et n'atteignent pas les premières étapes de la séquence
- ▶ Ces réseaux ont des difficultés à capturer les dépendances plus lointaines
- ▶ Une solution est d'utiliser le mécanisme de porte (gate) pour donner le contrôle de l'accès à la mémoire des états
- ▶ Soit une mémoire  $s \in \mathcal{R}^d$ , une entrée  $x \in \mathcal{R}^d$  et une porte  $g \in [0, 1]^d$
- ▶ Mise à jour avec porte de la mémoire :  $s' \leftarrow g \odot x + (1 - g) \odot (s)$

$$\begin{bmatrix} 8 \\ 11 \\ 3 \\ 7 \\ 5 \\ 15 \end{bmatrix} \leftarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 8 \\ 9 \\ 3 \\ 7 \\ 5 \\ 8 \end{bmatrix}$$

$s'$        $g$        $x$        $(1-g)$        $s$

# Réseaux récurrents pour le NLP

## Long Short-Term Memory (LSTM) [Hochreiter1997]



$$s_j = R_{\text{LSTM}}(s_{j-1}, x_j) = [c_j; h_j]$$

$$c_j = f \odot c_{j-1} + i \odot z$$

$$h_j = o \odot \tanh(c_j)$$

$$i = \sigma(x_j W^{xi} + h_{j-1} W^{hi})$$

$$f = \sigma(x_j W^{xf} + h_{j-1} W^{hf})$$

$$o = \sigma(x_j W^{xo} + h_{j-1} W^{ho})$$

$$z = \tanh(x_j W^{xz} + h_{j-1} W^{hz})$$

$$y_j = O_{\text{LSTM}}(s_j) = h_j$$

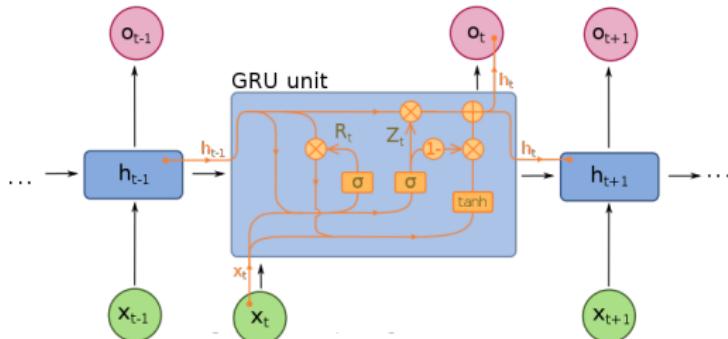
$$s_j \in \mathbb{R}^{2 \cdot d_h}, \quad x_t \in \mathbb{R}^{d_x}, \quad c_j, h_j, i, f, o, z \in \mathbb{R}^{d_h}, \quad W^{xo} \in \mathbb{R}^{d_x \times d_h}, \quad W^{ho} \in \mathbb{R}^{d_h \times d_h}$$

$c_j$  est un composant mémoire

$i, f, o$  sont des portes de contrôles (input, forget, output)  
dont les valeurs sont calculées par combinaison linéaire + sigmoid

# Réseaux récurrents pour le NLP

**Gate Recurrent Unit (GRU)** [Cho2014] : Alternative au LSTM, plus simple mais moins puissant



$$s_j = R_{\text{GRU}}(s_{j-1}, x_j) = (1 - z) \odot s_{j-1} + z \odot \tilde{s}_j$$

$$z = \sigma(x_j W^{xz} + s_{j-1} W^{sz})$$

$$r = \sigma(x_j W^{xr} + s_{j-1} W^{sr})$$

$$\tilde{s}_j = \tanh(x_j W^{xs} + (r \odot s_{j-1}) W^{sg})$$

$$y_j = O_{\text{GRU}}(s_j) = s_j$$

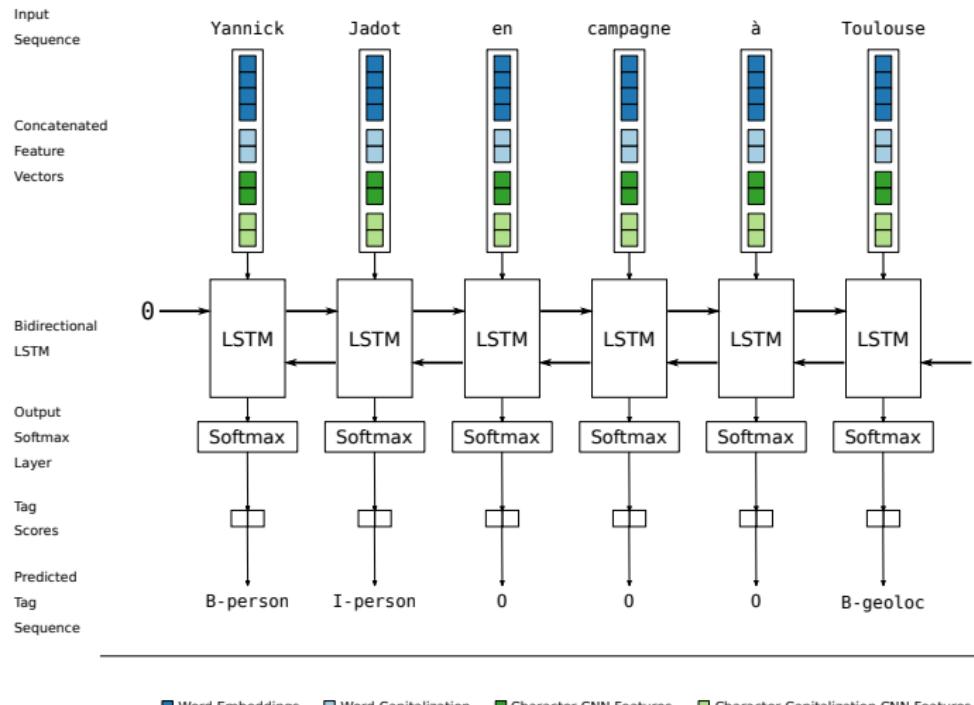
$$s_j, \tilde{s}_j \in \mathbb{R}^{d_s}, \quad x_i \in \mathbb{R}^{d_x}, \quad z, r \in \mathbb{R}^{d_s}, \quad W^{xo} \in \mathbb{R}^{d_x \times d_s}, \quad W^{so} \in \mathbb{R}^{d_s \times d_s}.$$

$r$  est une porte de contrôle utilisée avec l'état précédent

$z$  est un porte de contrôle l'état précédent et une interpolation du nouvel état

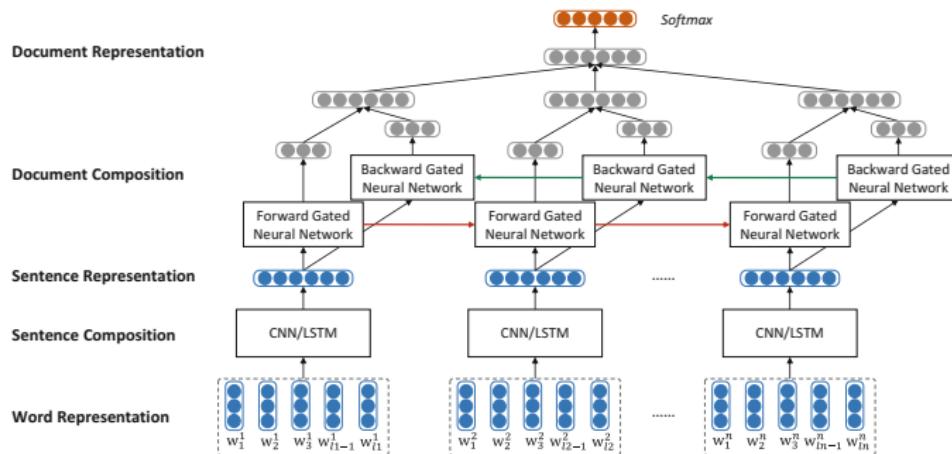
# Réseaux récurrents pour le NLP

## Détection d'entités nommés



# Réseaux récurrents pour le NLP

## Analyse de sentiments au niveau document



## Limites de réseaux récurrents

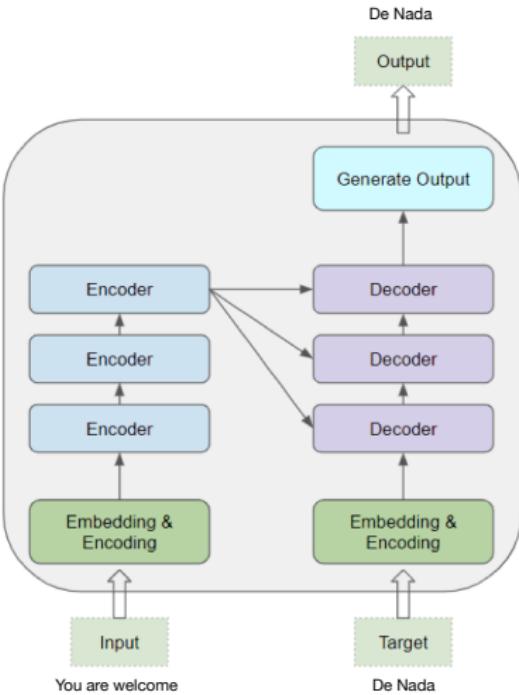
- ▶ Difficulté à traiter les dépendances à longue portée c-a-d entre des mots très éloignés dans une longue phrase
- ▶ Ils traitent la séquence d'entrée séquentiellement, un mot à la fois ce qui ralentit l'apprentissage et l'inférence (à l'inverse des réseaux CNN qui peuvent fonctionner en parallèle)

Ces limites ont conduit au développement des "Transformers" une catégorie d'architectures de type **Encoder-Decoder** pour le traitement des séquences qui utilise le **mécanisme d'attention**

- ▶ C'est devenu l'architecture de référence en NLP
- ▶ Papier séminal : Attention is all you need / Application à la traduction

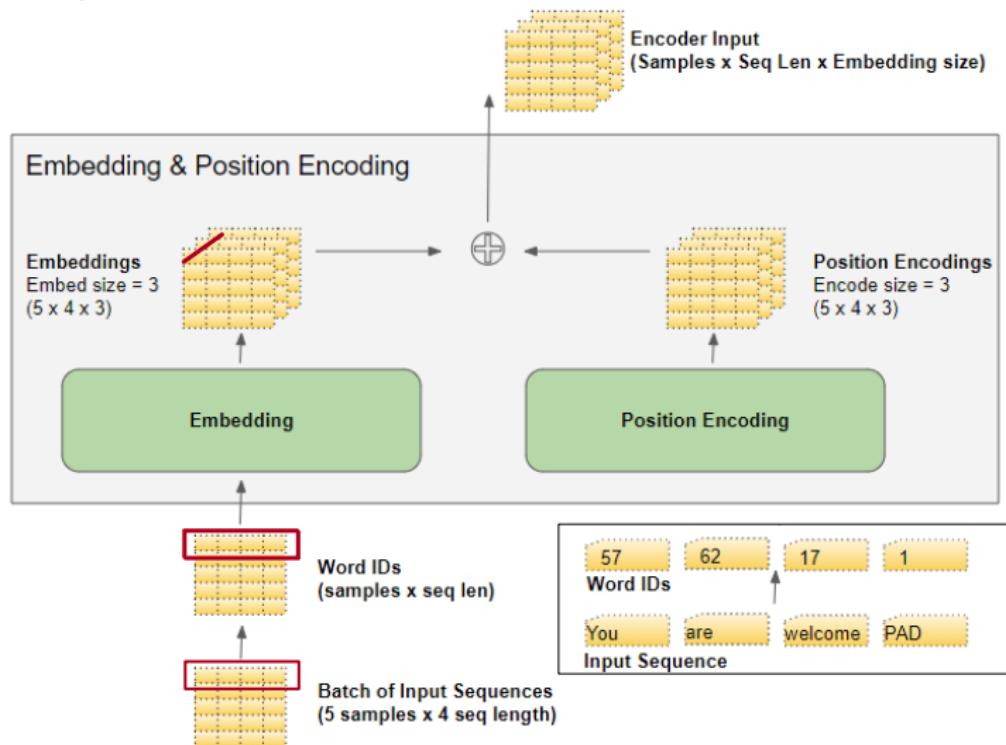
# L'architecture Transformeur

- ▶ Une architecture Transformeur prend en entrée une séquence (Encodeur) et produit une séquence en sortie (Decodeur)
- ▶ L'architecture est constituée d'une pile d'Encodeurs et d'une pile de Decodeurs
- ▶ Chaque pile possède sa couche de plongement (Embedding)
- ▶ La pile de Decodeurs se termine par une couche qui génère la sortie
- ▶ L'entraînement utilise une séquence cible en entrée des décodeurs



## Encodage des séquences en entrée

La couche de plongement encode chaque séquence de mots par une séquence de vecteurs. Chaque vecteur encode le mot et la position correspondante

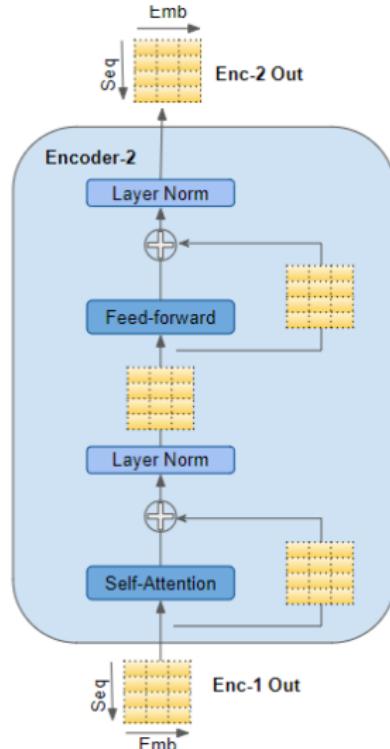


# Structure des Encodeurs

Les Encoders ont tous la même structure qui est composée :

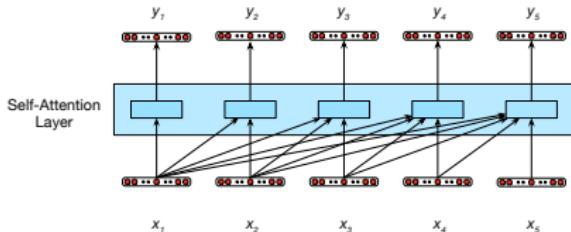
- ▶ d'un module d'auto-attention multi-têtes
- ▶ d'un module de propagation avant entièrement connecté (réduction à la dim. d'entrée)
- ▶ d'une connection résiduelle autour de chaque couche suivie d'un module de normalisation (amélioration performance d'entraînement)

Le premier Encoder reçoit une séquence de vecteurs produite par la couche de plongement. En sortie de l'Encoder, on obtient une séquence de vecteurs correspondant aux mots d'entrée



# Le mécanisme d'attention

- ▶ Ce mécanisme permet de comparer un élément d'intérêt à un ensemble d'autres éléments d'une manière qui révèle leur pertinence dans le contexte actuel
- ▶ Auto-attention : Comparaison de chaque élément avec les autres éléments de la séquence
- ▶ Utilisation du résultat de la comparaison pour calculer une sortie pour l'élément d'intérêt
- ▶  $score(x_i, x_j) = x_i \cdot x_j$
- ▶  $\alpha_{i,j} = softmax(score(x_i, x_j))$   
 $\forall j \leq i$
- ▶  $y_i = \sum_{j \leq i} \alpha_{i,j} x_j$

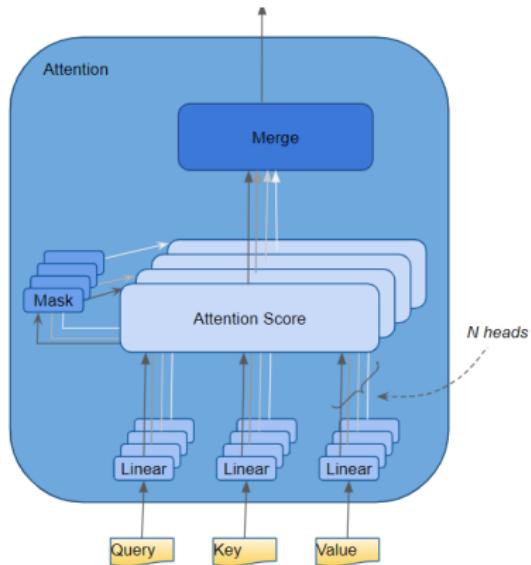


- ▶ A lui seul, le mécanisme d'attention ne permet d'apprendre la façon dont les mots contribuent à la sortie
- ▶ Il faut l'associer à des matrices de poids qui opèrent sur les entrées

# Le module d'auto-attention multi-têtes

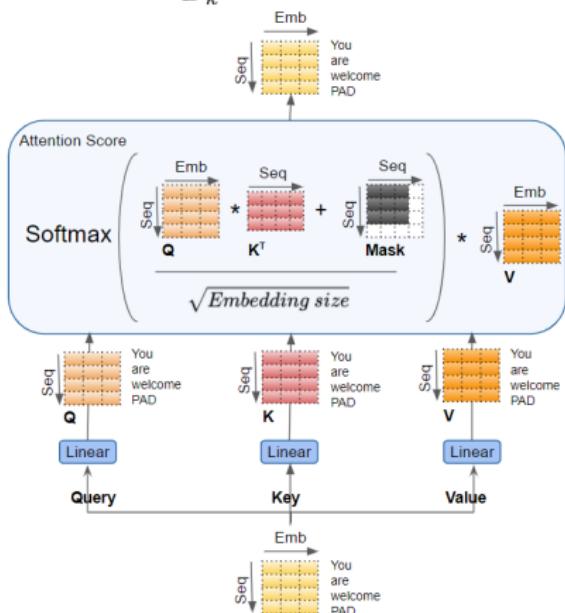
- ▶ Ce module reçoit ses données d'entrée sous la forme de trois paramètres, appelés *Query*, *Key* et *Value* (**identiques en auto-attention**)
- ▶ Il divise ses paramètres en  $N$  parties et fait passer chaque partie par un tête distincte
- ▶ Les calculs d'attention similaires des têtes sont combinés pour produire un score d'attention final
- ▶

$$\begin{aligned} \text{MultiHeadAttention}(Q, K, V) &= \\ \text{Concat}(\text{head}_1, \dots, \text{head}_H)W^0 \\ \text{avec } \text{head}_i &= \\ \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$

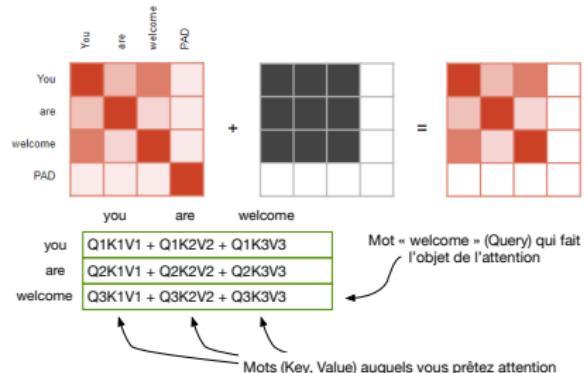


# Le calcul d'attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{D_k}}\right)V = AV$$



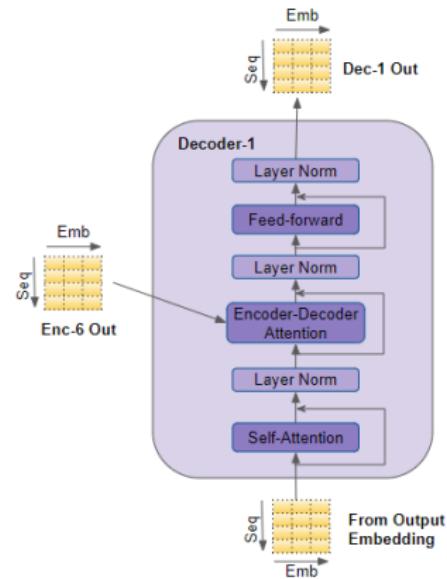
Le masquage sert à annuler les sorties d'attention lorsqu'il y a du padding dans les phrases d'entrée et éviter ainsi qu'il contribue à l'auto-attention



# Structure des Décodeurs

Les Décodeurs ont tous la même structure qui est composée :

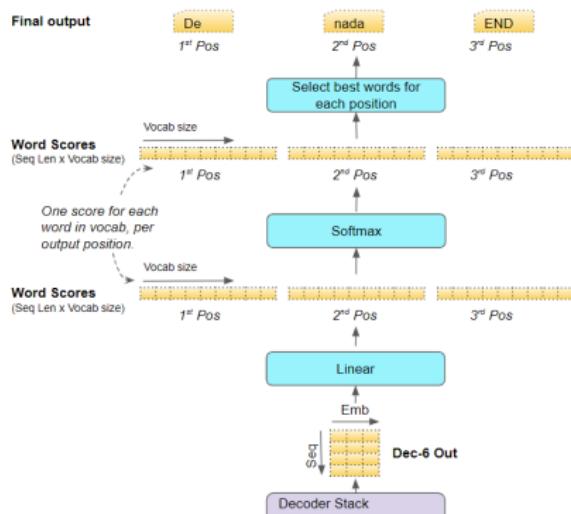
- ▶ d'un module d'auto-attention multi-tête avec masque
- ▶ d'un module d'attention croisée encodeur-décodeur
- ▶ d'un module de propagation avant en fonction de la position
- ▶ d'une connection résiduelle autour de chaque couche suivie d'un module de normalisation
- ▶



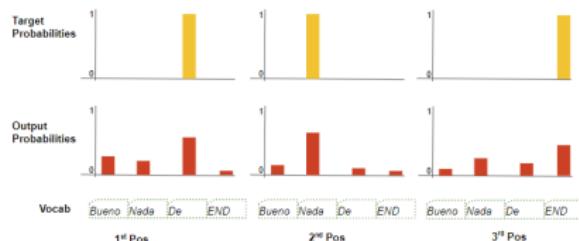
En phase d'entraînement, le premier Décodeur reçoit une séquence de vecteurs produite par la couche de plongement

# Génération de la sortie

- ▶ La couche linéaire projette le vecteur du décodeur en scores pour chaque mot du vocabulaire cible, à chaque position dans la phrase
- ▶ La couche Softmax transforme ensuite ces scores en probabilités



Utilisation d'une fonction de perte basée sur l'entropie croisée pour comparer la distribution de probabilité de sortie générée à celle de la séquence cible



# Entrainement et Prédition du Transformeur

## Entrainement

- Le Transformeur doit apprendre comment produire la séquence cible en utilisant la séquence d'entrée et la séquence cible

## Prédiction/Inférence

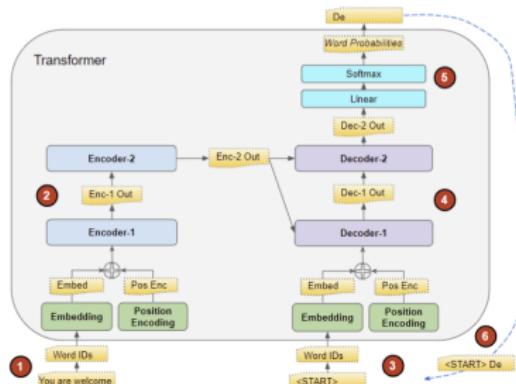
- Le Transformeur doit produire la séquence cible à partir de la seule séquence d'entrée

## Similitude avec un modèle Seq2Seq

- La sortie est générée en boucle de façon à ce que la séquence de sortie de l'étape précédente soit fournie au décodeur dans l'étape suivante jusqu'au jeton END

## Différence avec un modèle Seq2Seq

- Différence : A chaque pas de temps, la séquence de sortie entière ainsi générée est réinjectée (plutôt que le dernier mot - auto-regression)



# Usages de l'architecture Transformer

## Encoders-Decoders

- ▶ architecture complète typiquement utilisé dans la modélisation de séquence à séquence
- ▶ Méthode utilisée pour la traduction de langue neuronale

## Encoders (BERT)

- ▶ Seul les encodeurs sont utilisés et les sorties de l'encodeur final sont utilisées comme représentation de la séquence d'entrée
- ▶ Méthode utilisée pour les problèmes de classification ou d'étiquetage de séquences

## Décodeurs (GPT-i)

- ▶ Seul les décodeurs sont utilisés (les modules d'attention croisée sont éventuellement supprimés)
- ▶ Méthode utilisée pour la génération de séquences, comme la modélisation du langage

# Références

## Lectures

- ▶ Beaucoup de tutos simples sur '<https://towardsdatascience.com>'
- ▶ Foundations of Statistical Natural Language Processing, Christopher D.Manning, Hinrich Schutze
- ▶ Speech and Language Processing, An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Daniel Jurafsky, James H.Martin
- ▶ Neural Networks methods for Natural Language Processing, Yoav Goldberg
- ▶ Deep Learning in Natural Language Processing, Li Deng, Yang Liu  
Editors, Springer

## Code

- ▶ NLTK (Python) : <https://www.nltk.org>
- ▶ Spacy (Python) : <https://spacy.io>
- ▶ Stanford NLP (Java) : <https://stanfordnlp.github.io/CoreNLP/>
- ▶ Natural Language Processing with Python, O'Reilly
- ▶ Applied Text Analysis With Python, O'Reilly