

# Traitement Automatique des Langues Naturelles

Cours 6: Réseaux de neurones pour le TAL

---

Chloé Braud, Philippe Muller

Master IAFA 2024-2025

# Introduction : L'évolution historique du TAL

- TAL 'symbolique' ( 1950-1989)
  - modélisation du langage à base de règles
  - importance de ressources faites à la main : dictionnaires, grammaires, ...
- TAL 'statistique' NLP ( 1990-2014)
  - utilisation de méthode d'apprentissage automatique utilisant des vecteurs de caractéristiques de grande dimension, à base de catégories discrètes
  - modèles surtout linéaires, ou log-linéaires
  - grands corpus annotés, caractéristiques choisies "à la main"
- TAL 'neuronal' ( 2014-présent)
  - modèles neuronaux non linéaires sur entrée numériques (dense), de dimensions plus réduites
  - grands corpus annotés + utilisation de données non annotées
  - ± sélection automatique des caractéristiques / des architectures
  - grande modularité des approches

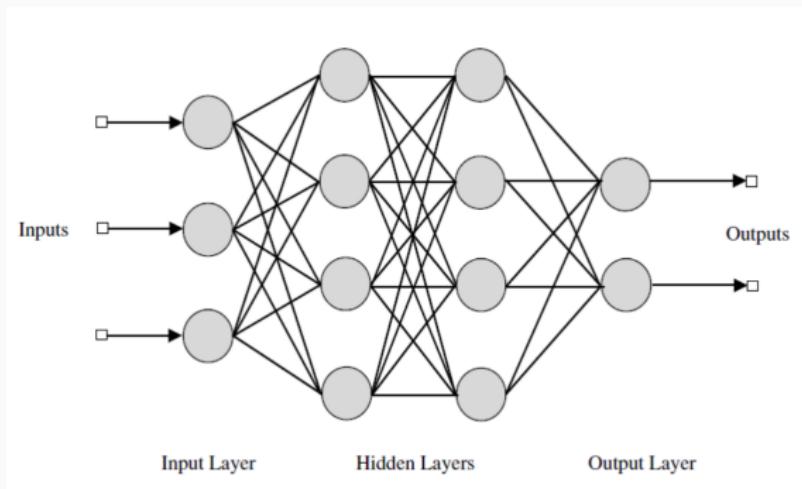
# Introduction : changement de paradigme en ML

- approches "classiques" : modèles linéaires entraînés sur des caractéristiques discrètes, en grandes dimensions → modèles "épars" (sparse)
- plus récemment : réseaux de neurones
  - modèles non-linéaires
  - entrées numériques : modèles "denses"
  - architectures modulaires
  - utilisation de données "brutes" / non supervisé
  - apprentissage de représentations intermédiaires

# Rappel : architectures de réseaux de neurone

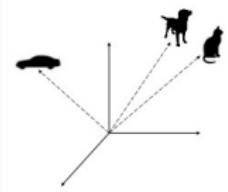
## Perceptron multi-couches / Feed-forward

- l'approche la plus simple de classification neuronale
- couches complètement connectées
- peut remplacer tel quel un modèle "classique", en remplaçant des entrées discrètes avec des entrées numériques "one-hot"



# Représentation des entrées : dense vs. one hot

- **One hot:** chaque trait/caractéristique/"feature" a son propre jeu de dimensions, un trait binaire 0/1 pour chaque valeur de catégorie
  - dimension d'entrée  $\propto$  nombre de traits
  - chaque trait est indépendant des autres
- **Dense:** chaque valeur de trait est un vecteur de dimension  $d$ 
  - dimension d'entrée : fixe ( $d$ )
  - des entrées "similaires" ont des représentations vectorielles similaires



On a vu des exemples du premier type d'approches au début du cours, puis des représentations vectorielles au niveau du mot dans le cours précédent.

# Représentation et combinaisons de traits

- TAL "traditionnel": il faut spécifier explicitement les interactions entre traits
  - E.g. avec des traits artificiels combinant des informations comme: le trait  $27 = 1$  si le mot considéré est *jump*, le tag est *V* et le mot précédent est *they*  
(dans un modèle de séquence par exemple)
- TAL "neuronal": on spécifie seulement des traits de base
- les non-linéarités du réseaux trouveront les combinaisons pertinentes pour optimiser le résultat



# Représentation des entrées

## Pourquoi avoir des représentations denses ?

- l'approche discrète marche souvent très bien pour beaucoup de tâches de TAL
  - modèle de langue n-grammes
  - POS-tagging, analyse syntaxique
  - classification "simple" : analyse de sentiment
- en général plus faible quand il faut représenter le sens des mots
  - pas de notion de similarité
  - généralisation / inférences limitées

# Représentation des entrées

## Pourquoi avoir des représentations denses ?

- l'approche discrète marche souvent très bien pour beaucoup de tâches de TAL
  - modèle de langue n-grammes
  - POS-tagging, analyse syntaxique
  - classification "simple" : analyse de sentiment
- en général plus faible quand il faut représenter le sens des mots
  - pas de notion de similarité
  - généralisation / inférences limitées

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 ]

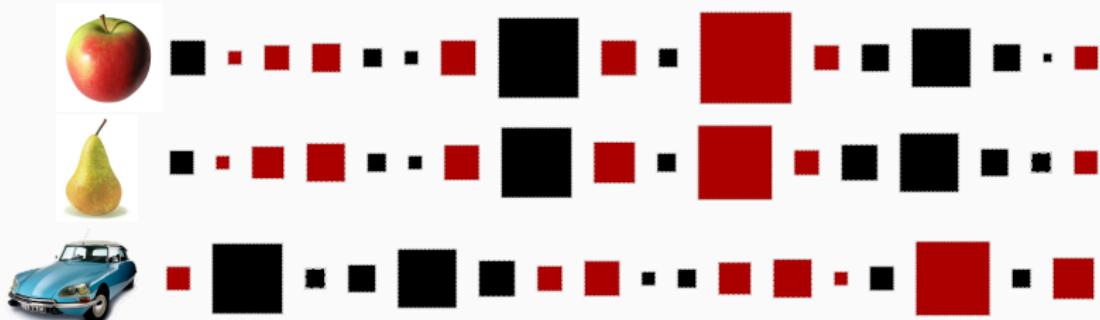


[ 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ]



# Représentation des entrées

On voudrait des représentations plus graduées ...



# Comment utiliser ces représentations ?

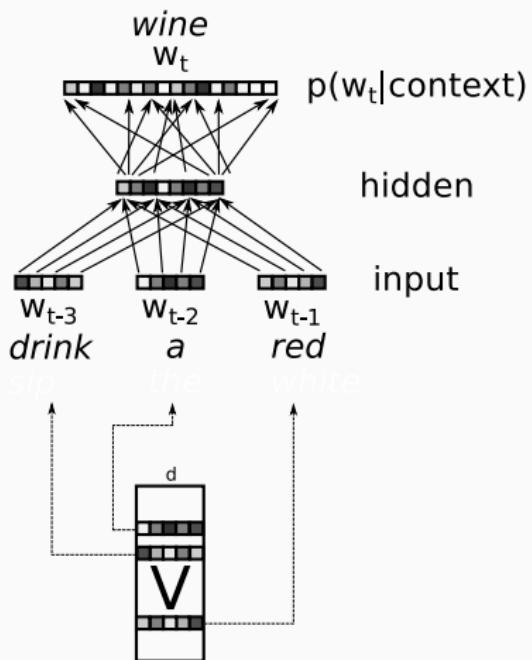
On a vu au cours précédent qu'on pouvait apprendre des représentations vectorielles de mots qui capturent en partie les similarités de sens / certains liens sémantiques

Mais en pratique, comment utiliser ces représentations pour :

- de la classification de texte ?
- de l'étiquetage de séquence ?
- de la génération de textes ?

# Cas simple : modèles de langue

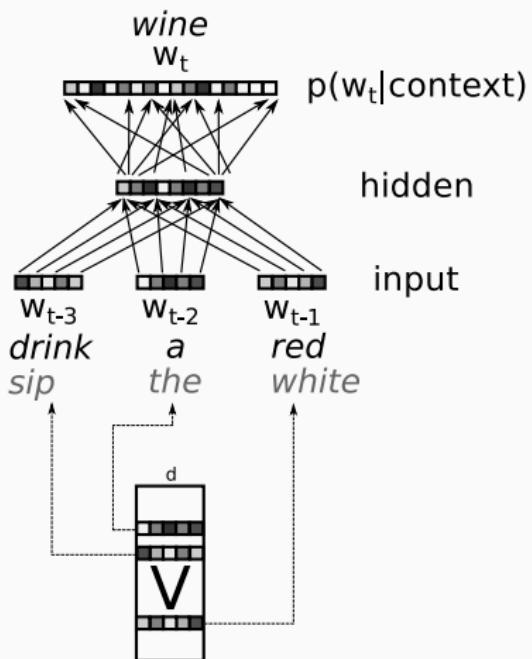
- prédire le mot suivant dans une séquence, en "regardant" juste le mot précédent, ou n mots précédents.
- une couche cachée non linéaire, softmax en sortie
- optimiser la probabilité du mot suivant correct



Bengio et al. 2003. *A Neural Probabilistic Language Model*.

# Cas simple : modèles de langue

- prédire le mot suivant dans une séquence, en "regardant" juste le mot précédent, ou n mots précédents.
- une couche cachée non linéaire, softmax en sortie
- optimiser la probabilité du mot suivant correct



Bengio et al. 2003. *A Neural Probabilistic Language Model.*

# Au delà du mot

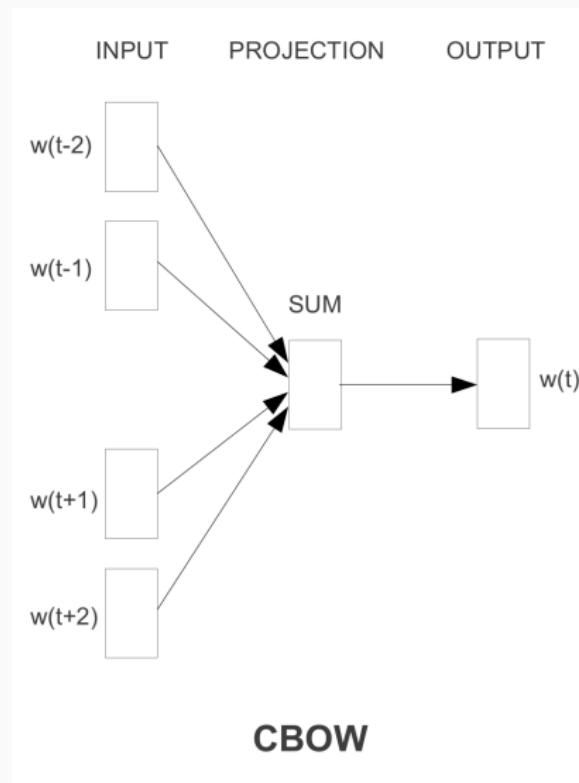
## Prédiction sur des phrases

Comment représenter des entrées de taille variables, comme une phrase ou un document avec un nombre variable de mots en entrée ?

Plusieurs options

- Continuous Bag of Words (CBOW): on somme les représentations des mots
  - mais : on perd l'information de l'ordre ("not good quite bad" = "not bad quite good")
- Réseau convolutif :
  - fenêtre glissante sur la phrase puis somme/moyenne, pour prendre au moins en compte interactions locales entre mots
- Réseau récurrent (RNN) : les représentations de mots sont composés dans l'état courant du RNN

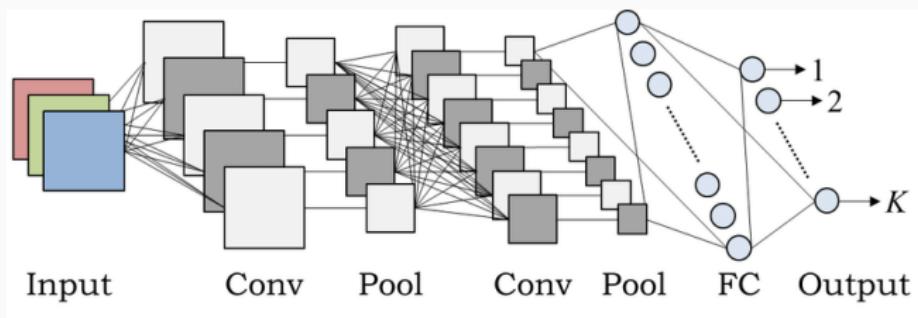
# Continuous bag of words



# Composition locale : Convolutional neural network

## Rappel

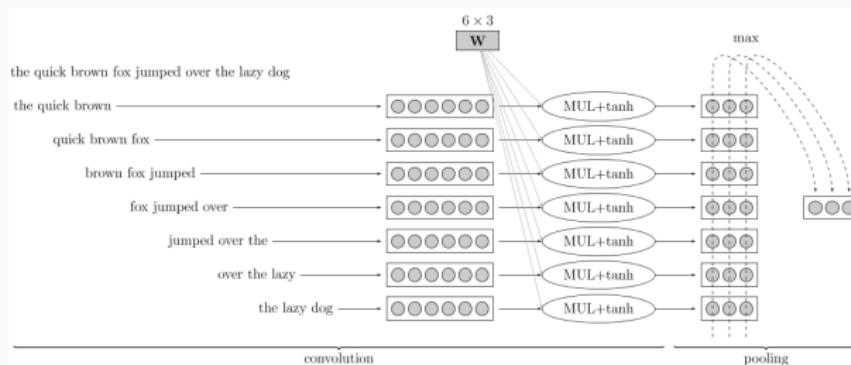
- utilise des filtres de convolutions
- des couches connectées **localement** seulement
- compose les informations locales (convolution) et agrègent cette info (pooling)
- apprennent des schémas qui reviennent souvent (très efficace en vision)



FC = fully connected

# CNN pour le TAL

- But: identifier des traits informatifs localement (liés à des n-grammes par exemple) et les combiner en un vecteur de taille fixe
- appliquer une convolution sur leurs représentations
- aggréger "Pooling" par exemple en prenant le maximum dans chaque dimensions



From Goldberg, 2017, Neural Network Methods in Natural Language Processing

# CNN pour le TAL

## Exemple détaillé

$\emptyset$	0.0	0.0	0.0	0.0
<b>tentative</b>	0.2	0.1	-0.3	0.4
<b>deal</b>	0.5	0.2	-0.3	-0.1
<b>reached</b>	-0.1	-0.3	-0.2	0.4
<b>to</b>	0.3	-0.3	0.1	0.1
<b>keep</b>	0.2	-0.3	0.4	0.2
<b>government</b>	0.1	0.2	-0.1	-0.1
<b>open</b>	-0.4	-0.4	0.2	0.3
$\emptyset$	0.0	0.0	0.0	0.0

$\emptyset, t, d$	-0.6
$t, d, r$	-1.0
$d, r, t$	-0.5
$r, t, k$	-3.6
$t, k, g$	-0.2
$k, g, o$	0.3
$g, o, \emptyset$	-0.5

Apply a **filter** (or **kernel**) of size 3

3	1	2	-3
-1	2	1	-3
1	1	-1	1

# CNN pour le TAL

Exemple détaillé: plus de filtres + pooling

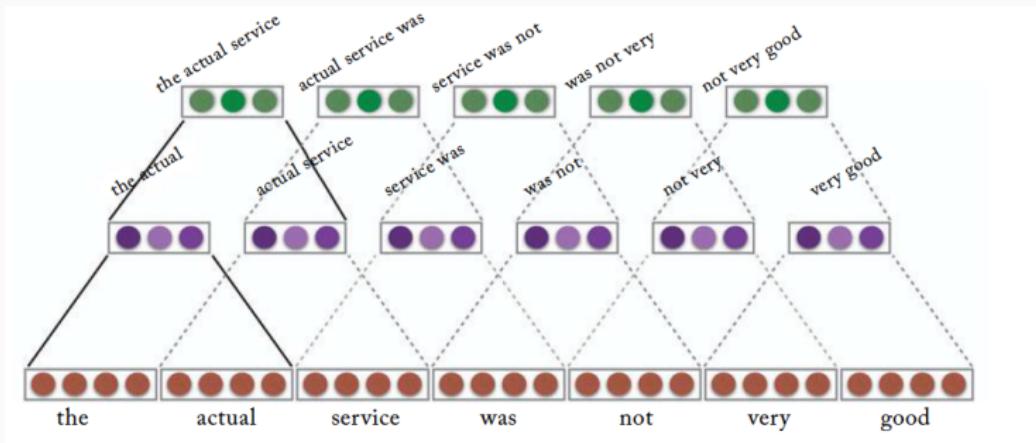
$\emptyset$	0.0	0.0	0.0	0.0
<b>tentative</b>	0.2	0.1	-0.3	0.4
<b>deal</b>	0.5	0.2	-0.3	-0.1
<b>reached</b>	-0.1	-0.3	-0.2	0.4
<b>to</b>	0.3	-0.3	0.1	0.1
<b>keep</b>	0.2	-0.3	0.4	0.2
<b>government</b>	0.1	0.2	-0.1	-0.1
<b>open</b>	-0.4	-0.4	0.2	0.3
$\emptyset$	0.0	0.0	0.0	0.0

$\emptyset, t, d$	-0.6	0.2	1.4
$t, d, r$	-1.0	1.6	-1.0
$d, r, t$	-0.5	-0.1	0.8
$r, t, k$	-3.6	0.3	0.3
$t, k, g$	-0.2	0.1	1.2
$k, g, o$	0.3	0.6	0.9
$g, o, \emptyset$	-0.5	-0.9	0.1
<b>max p</b>	0.3	1.6	1.4

Apply 3 filters of size 3

3	1	2	-3	1	0	0	1	1	-1	2	-1
-1	2	1	-3	1	0	-1	-1	1	0	-1	3
1	1	-1	1	0	1	0	1	0	2	2	1

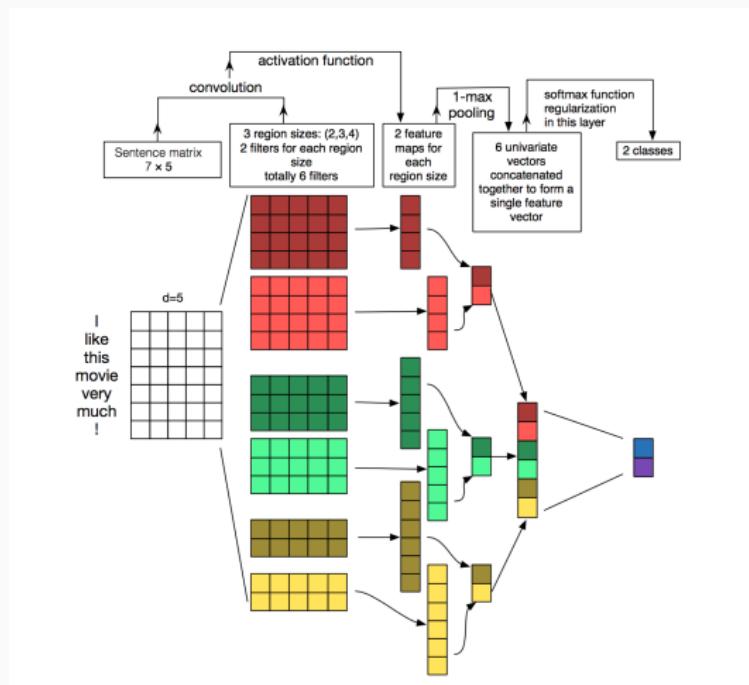
## CNN en cascade: structure hiérarchique



From Goldberg, 2017, Neural Network Methods in Natural Language Processing

# CNN pour le TAL

On peut combiner plusieurs échelles



From: Zhang and Wallace (2015) A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification

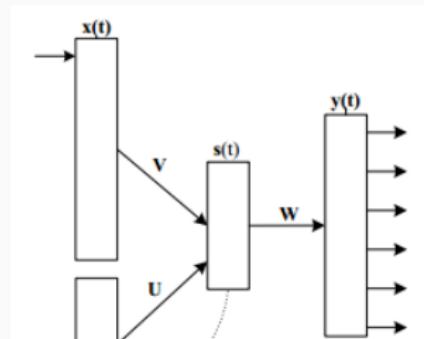
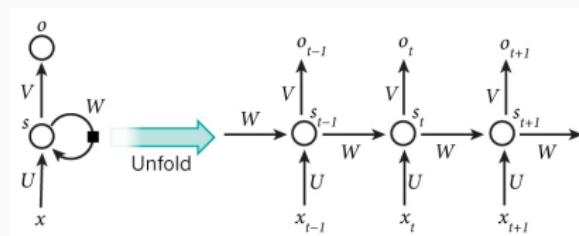
## Problèmes des CNN sur une phrase

- taille de contexte fixée
- ne correspond pas aux structures linguistiques
- interactions entre mots très locales

# Composition séquentielle: RNN

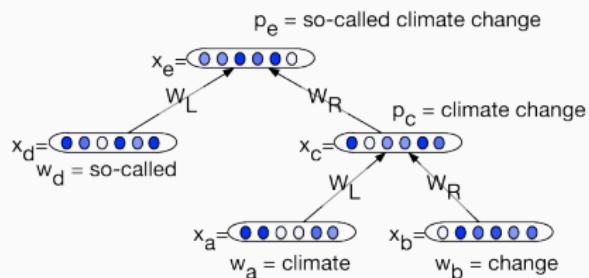
## Recurrent neural networks (RNN)

- gérer des données structurées de taille quelconque et variable
- Recurrent networks pour les **séquences**



# Composition en arbre: Recursive NN / Tree RNN

- gérer des données structurées de taille quelconque et variable
- Recursive networks pour les **arbres**



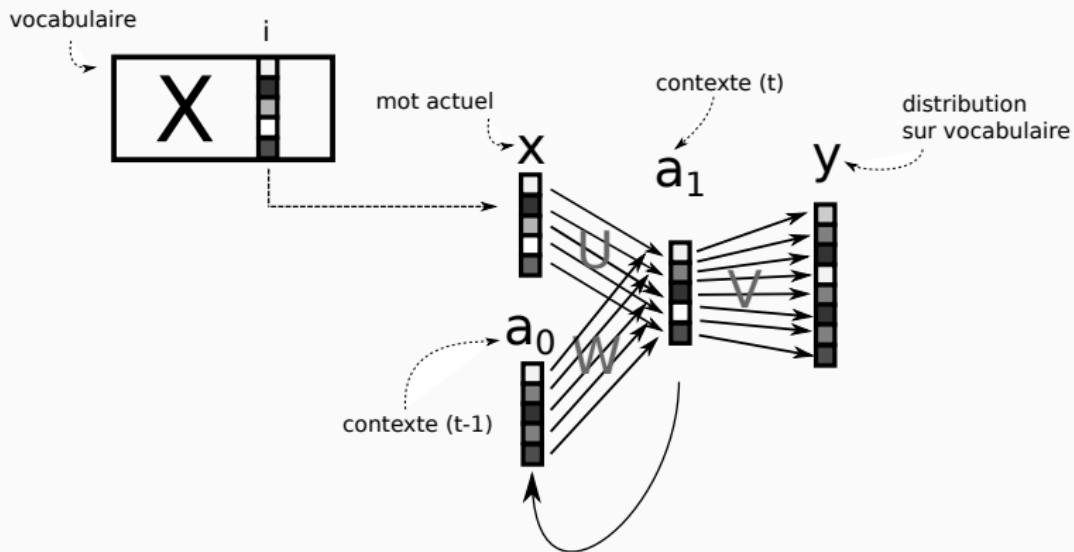
## Récapitulatif

- CBOW: pas d'ordre, pas de structure
- CNN: mieux mais seulement des schémas locaux
- RNN: représente les structures de taille arbitraire dans un vecteur de taille fixe, prend en compte les propriétés de structure.

# Réseau de neurone récurrent

Peut aussi servir de modèle de langue de taille infinie

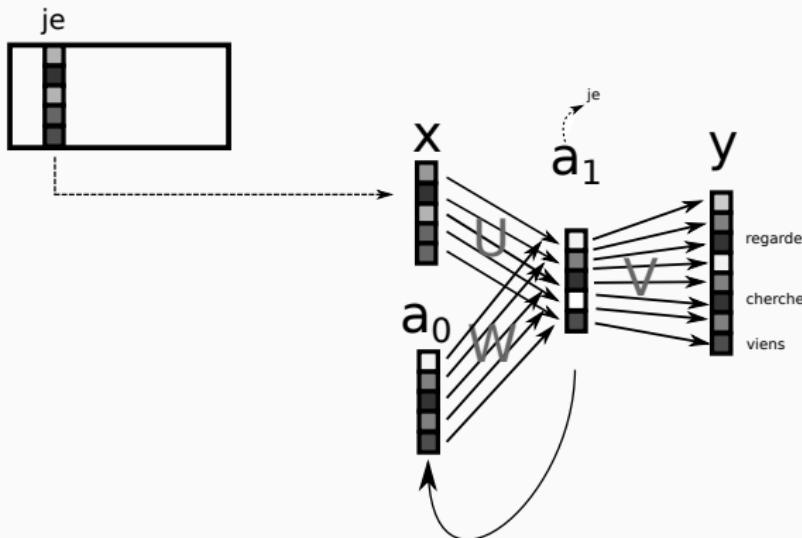
$\infty - gram$



# Réseau de neurone récurrent

Peut aussi servir de modèle de langue de taille infinie

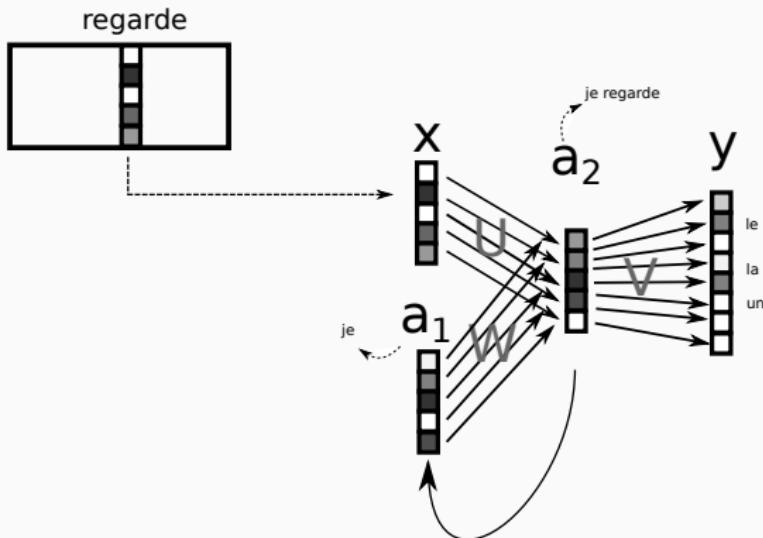
$\infty - gram$



# Réseau de neurone récurrent

Peut aussi servir de modèle de langue de taille infinie

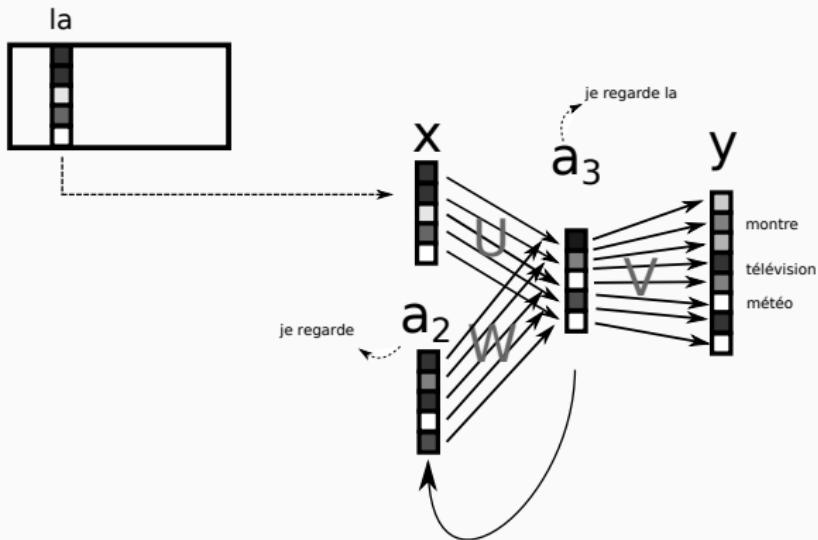
$\infty - gram$



# Réseau de neurone récurrent

Peut aussi servir de modèle de langue de taille infinie

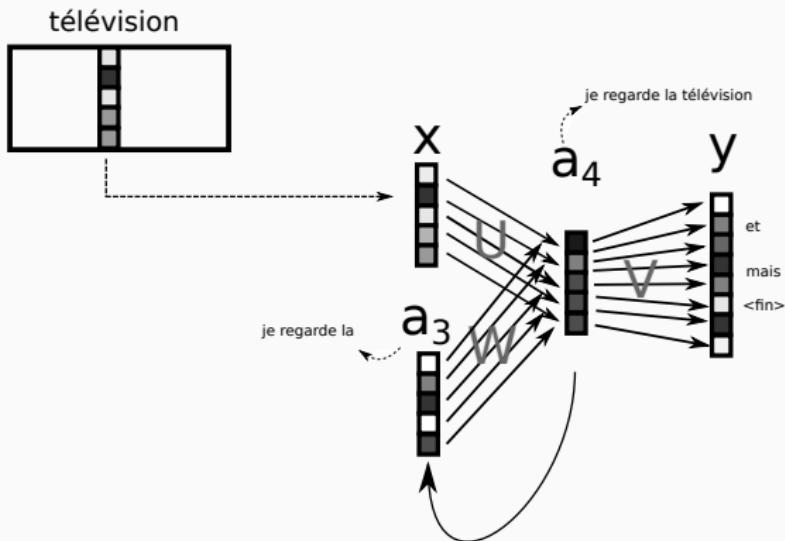
$\infty - gram$



# Réseau de neurone récurrent

Peut aussi servir de modèle de langue de taille infinie

$\infty - gram$

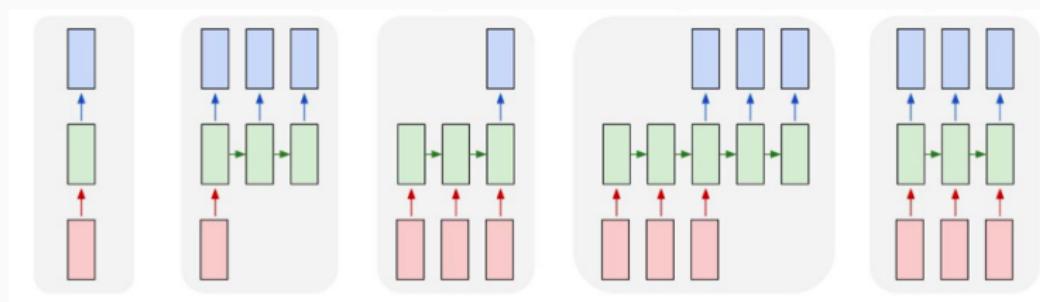


## Entrainement / type de tâches

- réseau récurrent  $\approx$  réseau très profond avec des paramètres partagés entre les éléments de la structure
- "Backpropagation through time"
- Peut être bidirectionnel
- Quel genre de supervision ?
  - Classification: seul l'état final compte
  - Etiquettage: une sortie pour chaque élément de l'input
  - Encodeur-décodeur: un RNN encode la séquence en un vecteur, un autre RNN le transforme en une nouvelle séquence (ex: traduction automatique)

# Réseau de neurone récurrent

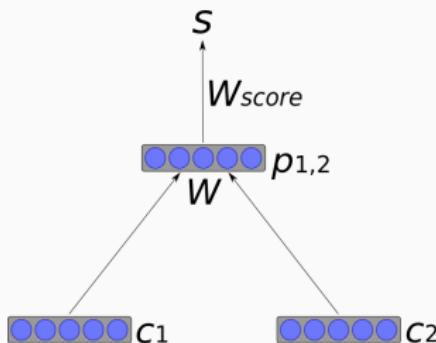
## Différentes architectures pour des problèmes de séquences



FF / Generation/ Classification / Encoder-Decoder / Tagging

# Recursive neural networks

- Généralisation des RNNs de séquence à arbre
- Transformation linéaire + activation non-linéaire appliquée récursivement en arbre
- Peut servir pour gérer des représentations à composer dans un arbre, comme un arbre donné par une analyse syntaxique par exemple



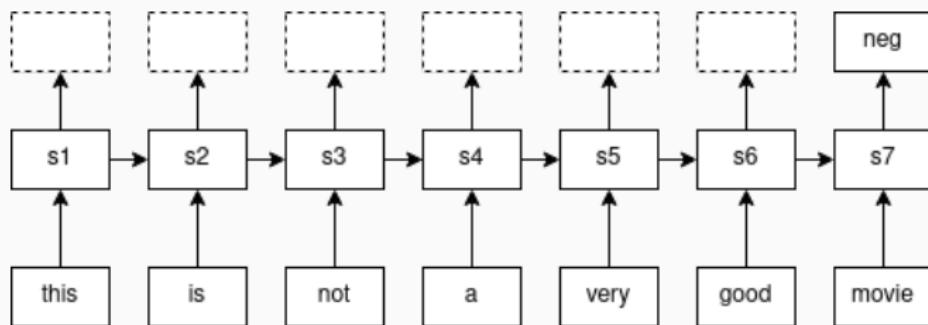
# Exemple de modélisation de problème de TAL

## Apprentissage / Analyse

- classification: sentiment, junk-mail,
- étiquetage de séquence : détection d'entités, question-answering
- structure/génération : modèle encodeur décodeur (résumé, traduction)

# Application

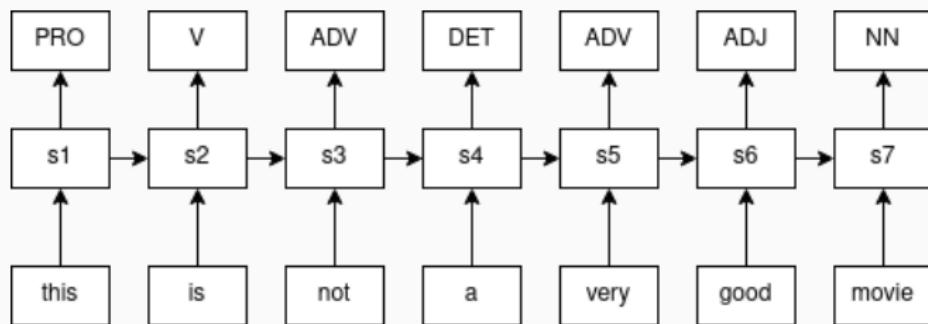
## Text classification



$s_i$  = état / unité récurrente (RNN/LSTM/GRU)

# Application

## Etiquetage de séquence/Sequence tagging

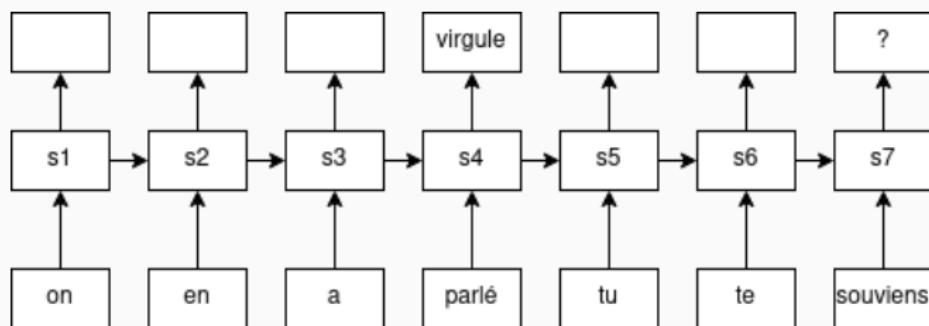


peut être combiné avec un classifieur structuré de séquence (eg HMM ou CRF: conditional random field)

# Application

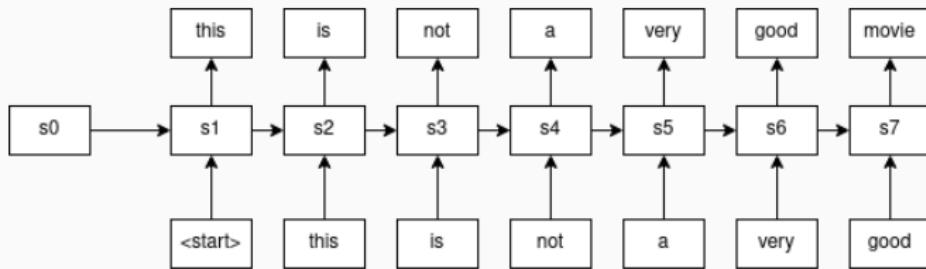
## Sequence tagging: couvre de nombreux cas

exemple : reponcuation de transcriptions orales



# Et la génération ?

Un générateur prédit l'élément suivant d'une séquence :



Par exemple générateur de langage naturel.

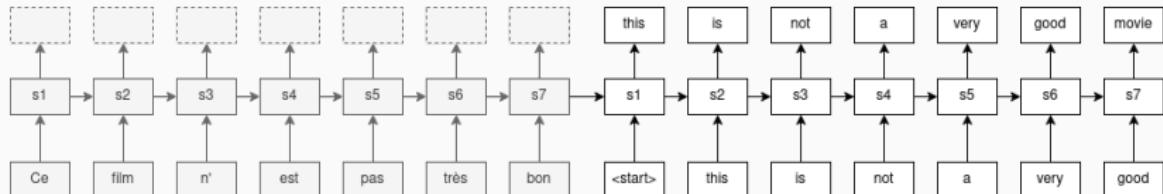
Comment peut-on l'entrainer ?

La représentation initiale peut être aléatoire ou venir d'une autre partie d'un système.

# On peut combiner !

On combine un RNN où on ne s'occupe pas des sorties + un générateur "conditionné" par la sortie du premier RNN ("encodeur")

→ Encodeur-décodeur séquentiel



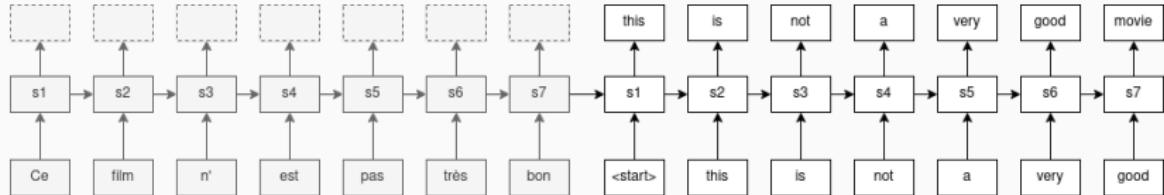
Encodeur-décodeur séquentiel ==

"sequence-to-sequence"

"modèle seq2seq"

**La longueur de la sortie peut être différente de la longueur de l'entrée**

# Exemple de seq2seq: traduction automatique



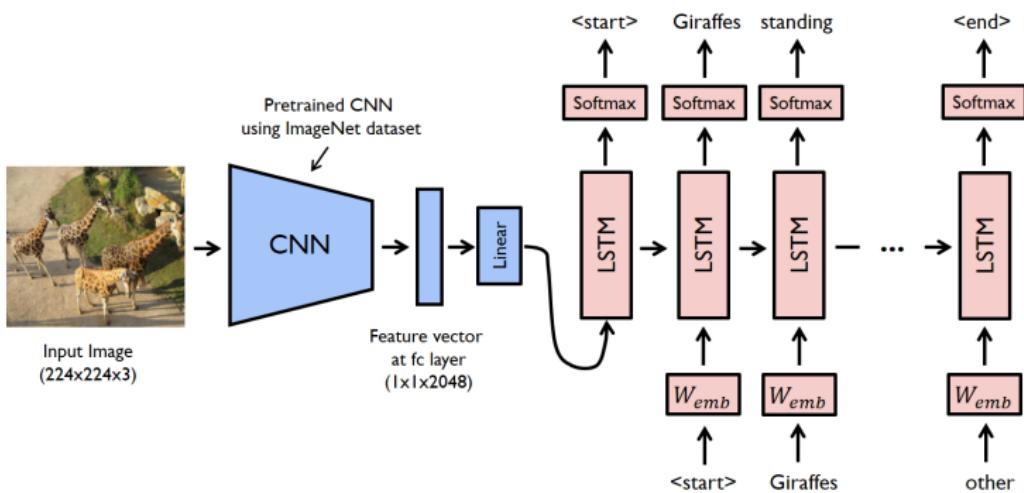
L'encodeur représente la phrase de la langue source pour conditionner le décodeur en langue cible.

Le décodeur est un générateur de langue cible.

# Application

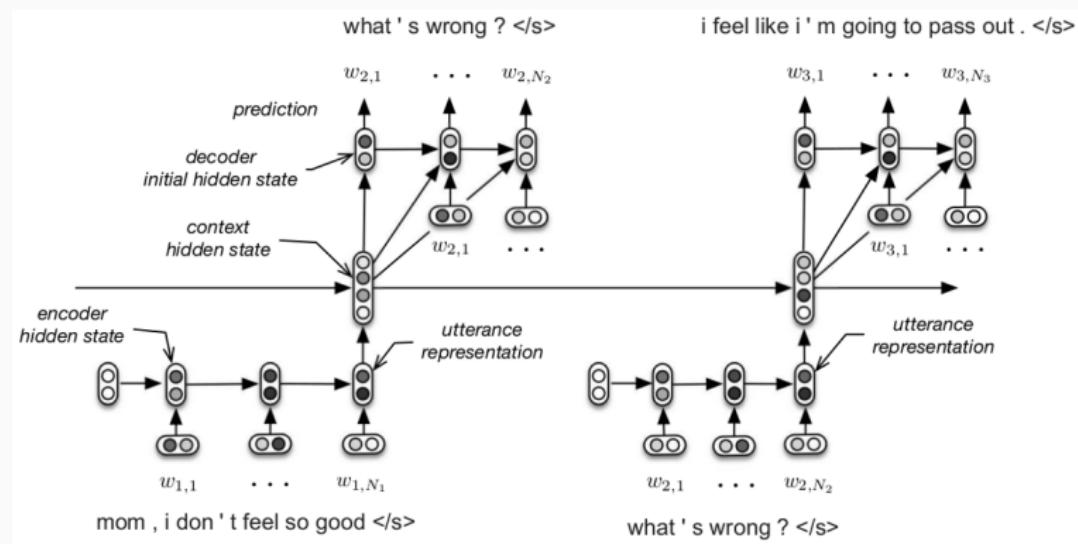
## Génération de légende d'image

Cette fois le générateur est conditionné par l'encodage d'une image, avec un réseau spécifique



# Application

## Génération de dialogue (chatbot)



**Un encodeur peut aussi conditionner des tâches de classification**

Exemple : Système de "Question-réponse"

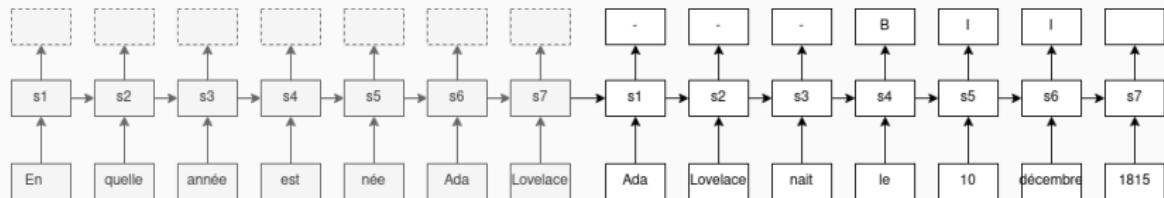
Comment ?

# Application

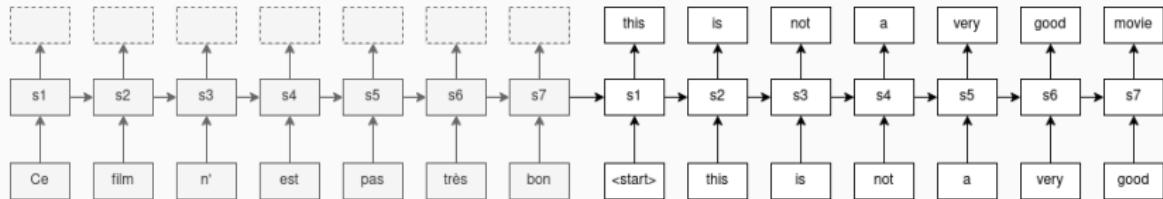
Un encodeur peut aussi conditionner des tâches de classification

Exemple : Système de "Question-réponse"

Comment ?



# Application: Traduction automatique



L'encodeur représente la phrase de la langue source pour conditionner le décodeur en langue cible.

Problème : goulot d'étranglement informationnel

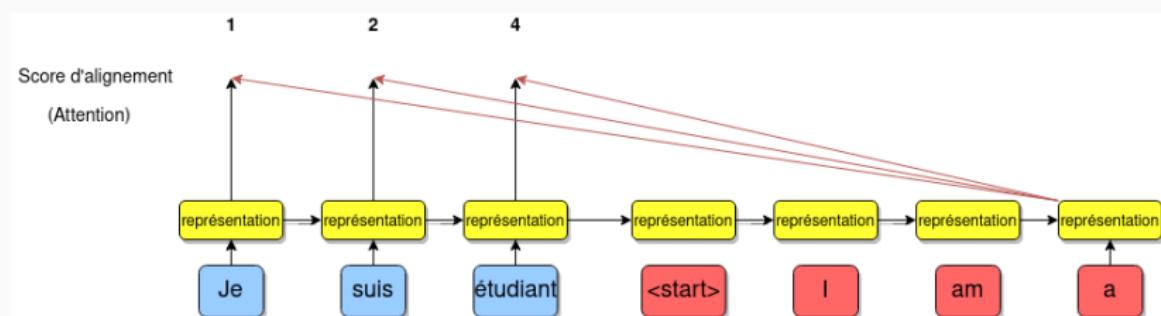
- Toute l'entrée est encodée en une seule représentation  
→ Perte d'information du début de la séquence
- Solution ? ajouter de l'information spécifique venant des éléments de l'entrée
- "Attention"

# Attention = score d'alignement

introduit par (Bahdanau, Cho, Bengio 2015) "Neural Machine Translation by Jointly Learning to Align and Translate"

Variante: (Luong, Pham, Manning, 2015) "Effective Approaches to Attention-based Neural Machine Translation"

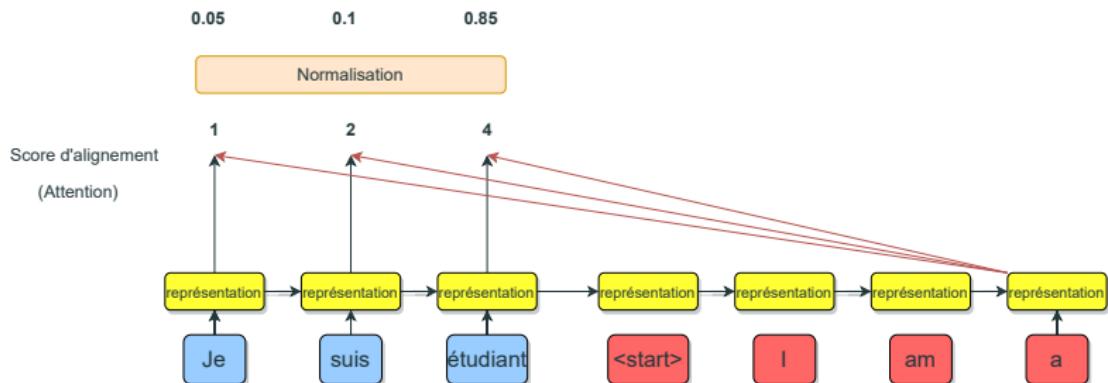
On va essayer d'"aligner" l'état courant du décodeur sur les états pertinents de l'entrée de l'encodeur.



# Vecteur de contexte

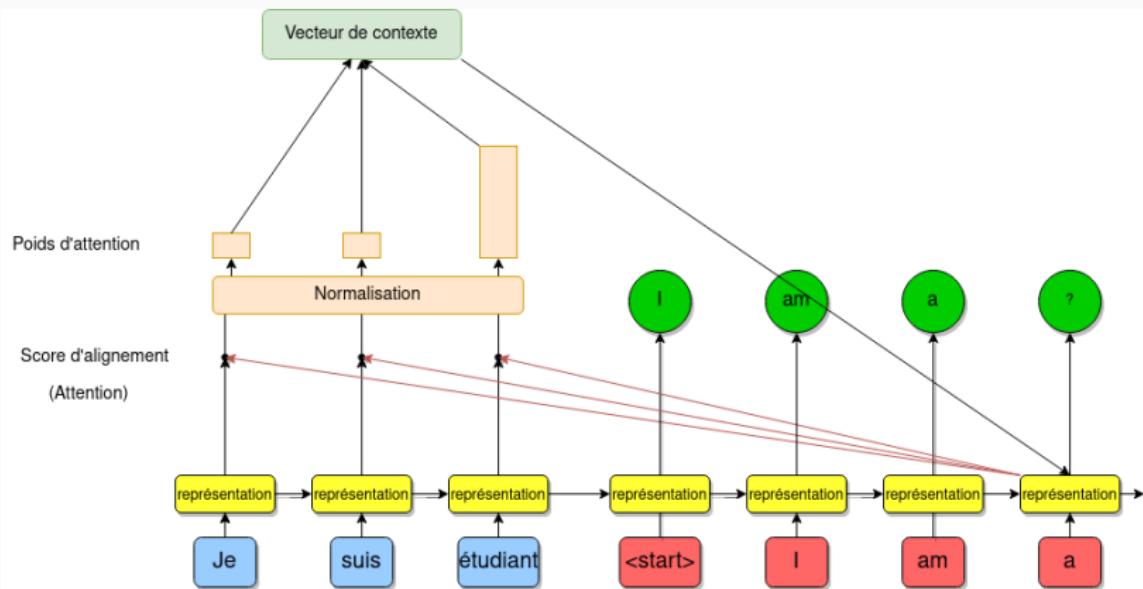
Les poids d'attention servent à faire une somme pondérée des états des entrées de l'encodeur (après normalisation).

Vecteur de contexte :  $0.05\text{"Je"} + 0.1\text{"suis"} + 0.85\text{"étudiant"}$



# Réseau d'attention

Le vecteur de contexte est finalement combinée à l'état courant du décodeur pour faire une prédiction



# Réseaux d'attention

## Formellement: le vecteur de contexte

Avec :

$h_t$  = état courant  $t$  du décodeur

$s_k$  = état de l'entrée  $k$  de l'encodeur

Scores d'alignement possibles:

- $score(x, y) = x \cdot y$
- $score(x, y) = w_2^T \cdot \tanh W1 \times [x; y]$  (Bahdanau)
- $score(x, y) = x^T \times W \times y$  (Luong)

Poids d'attention (normalisés):

$$a_{t,k} = \frac{\exp(score(h_t, s_k))}{\sum_{j=1}^S \exp(score(h_t, s_j))}$$

Vecteur de contexte:

$$c_t = \sum_k a_{t,k} \cdot s_k$$

## Formellement: la prise en compte du vecteur de contexte

- Représentation du décodeur / Bahdanau

$$h_t = f(h_{t-1}, c_t)$$

et  $c_t$  est en fait calculé à partir de  $h_{t-1}$

- Représentation du décodeur / Luong

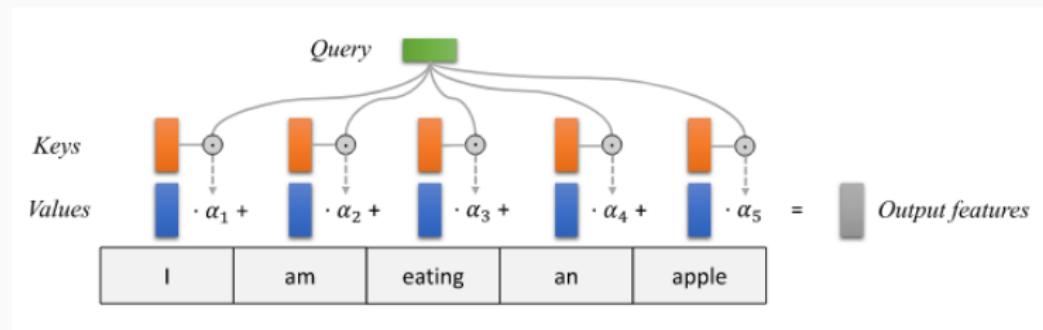
$$\hat{h}_t = f(c_t, h_t) = \tanh(W_c[c_t; h_t])$$

## Avantages de l'attention

- facilite l'entraînement
- aide à l'interprétation du résultat (un peu)

# Vue générale sur l'attention

On peut considérer que l'attention est une requête ("query") sur les entrées, que l'on fait correspondre avec une clef ("key") pour opérer sur une valeur de l'entrée avec une certaine ("value")



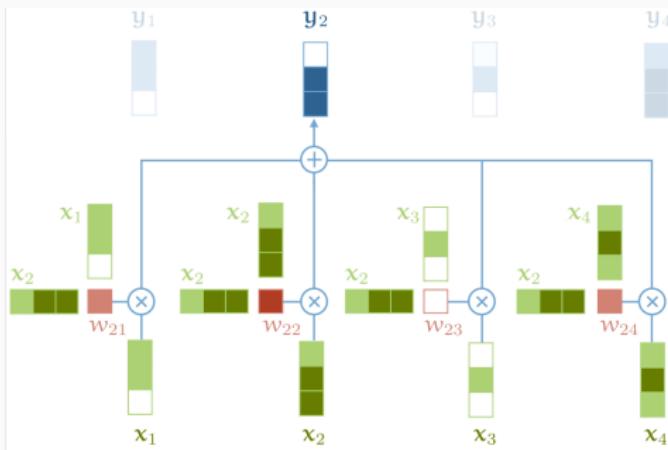
Dans le cas du seq2seq avec attention :

- la query vient de l'état courant du décodeur ( $h_t$ ),
- qui interroge chaque entrée  $k$  par sa représentation  $s_k$  (clef),
- la value, pondération du vecteur de contexte ( $c_t$ ), est aussi  $s_k$

# Réseaux de self-attention

En fait, on pourrait remplacer tout le réseau récurrent de l'encodeur ou du décodeur avec en entrée  $x_1, \dots, x_n$  et en sortie  $y_1, \dots, y_n$ ,

$$y_i = \sum_j w_{ij} x_j$$



“Transformers from scratch” <http://peterbloem.nl/blog/transformers>

## Réseaux de self-attention

Au lieu de prendre juste les dépendances entre entrées, une autre façon de faire: créer des représentations pour chacun des différents rôles de  $x_i$  par rapport au calcul de l'attention:

- **Query** son interaction avec les autres  $x_j$  pour calculer le score d'attention  $x_i, x_j$
- **Key** son rôle pour le calcul des poids pour la sortie d'un autre  $x_j$  jouant le rôle de Query
- **Value** son rôle dans la pondération finale pour calculer les  $y_j$

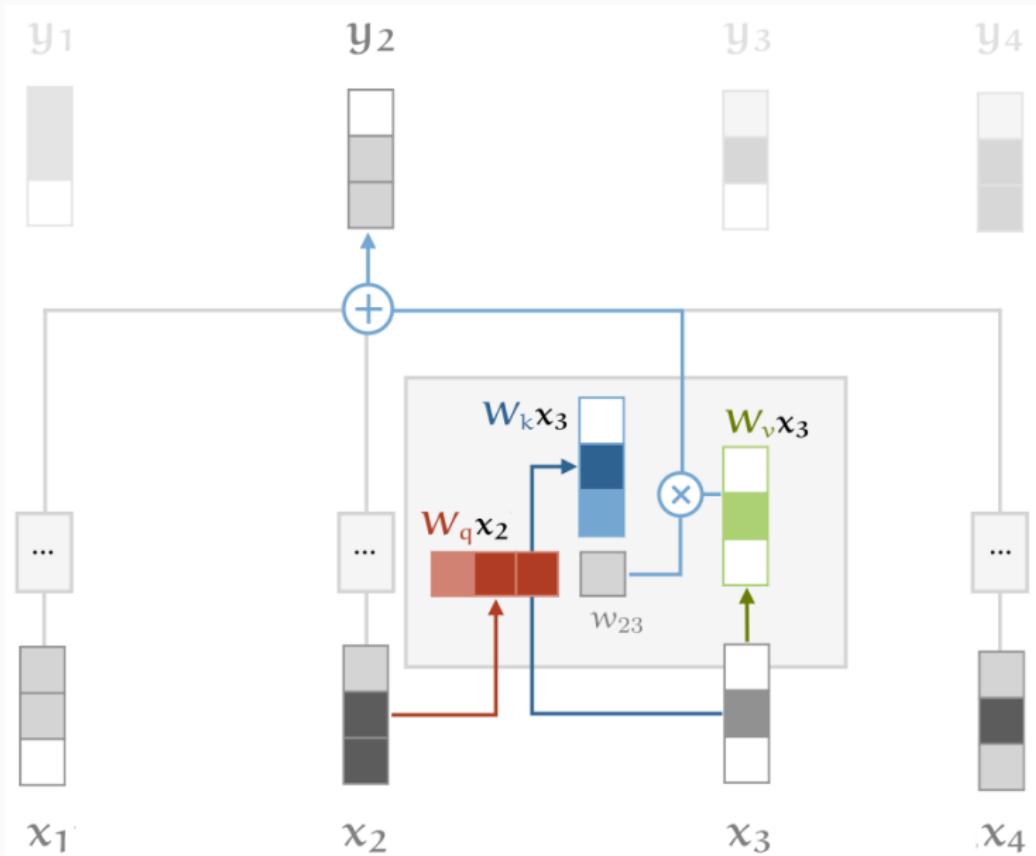
$q_i = W_q x_i$  paramètres pour créer une Query

$k_i = W_k x_i$  paramètres pour créer une Key

$v_i = W_v x_i$  paramètres pour créer une value

$w_{ij} = q_i \cdot k_j$  score d'attention  $x_i/x_j$  (normalisé ensuite avec softmax)

$y_i = \sum_j w_{ij} v_j$  sortie finale: somme des values, pondérées par les scores d'attention

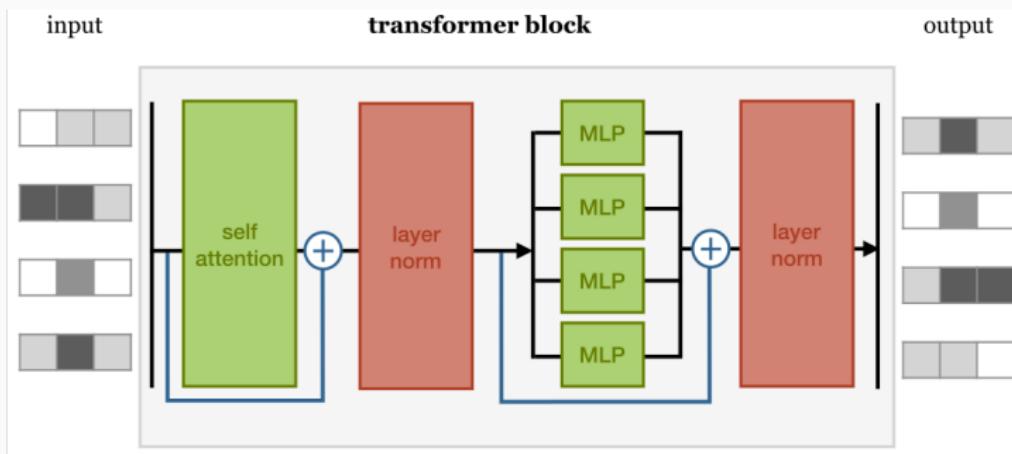


# Réseaux de self-attention

## “Transformer”

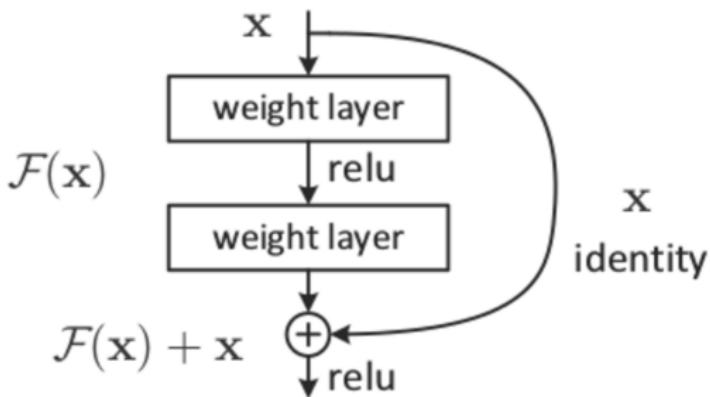
Eléments en plus :

- ajout de non-linéarités
- normalisation par couche
- connection résiduelle
- mais aucune information sur l'ordre de la séquence ?



## Parenthèse: connexion résiduelle

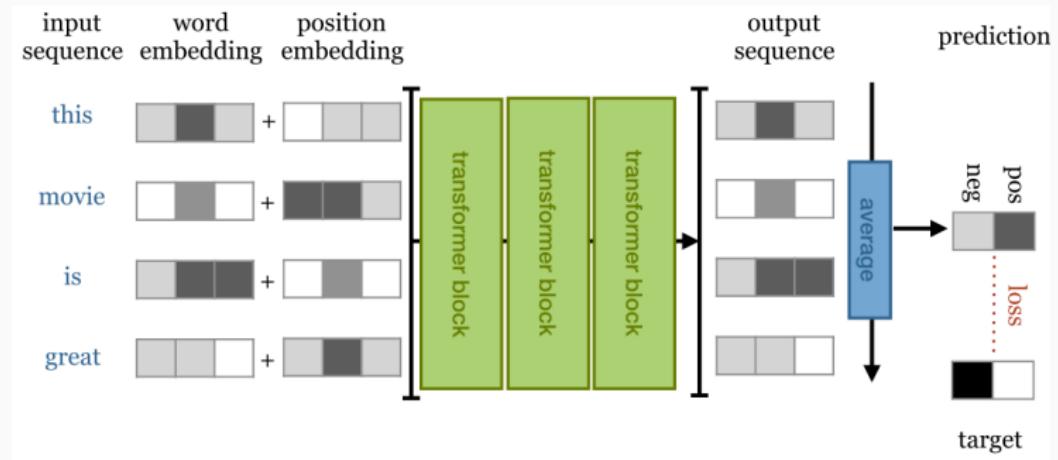
Préserve de l'information d'origine à travers des réseaux plus ou moins profonds



# Réseaux de self-attention

## Classifieur

on ajoute une représentation de la position en plus



"Transformers from scratch" <http://peterbloem.nl/blog/transfomers>

## Encodage de position

Plusieurs possibilités:

- représentation absolue, fixe (juste un ordinal)
- représentation fixe pour encoder des positions relatives
- représentation apprise en même temps par le réseau

Philipp Dufter, Martin Schmitt, and Hinrich Schütze. 2022. *Position Information in Transformers: An Overview*. Computational Linguistics, 48(3)

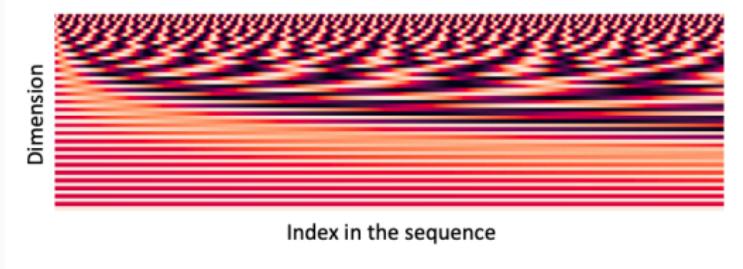
# Réseaux de self-attention

## Encodage de position

Un exemple de représentation fixe pour encoder des positions relatives:

ajout de vecteur périodique de période géométrique

chaque position est différente + permet de retrouver la différence relative entre 2 tokens (produit vectoriel ne dépend que de la différence de position)



En effet ce qui compte est le résultat des compositions de représentation: et  $x.y = \dots = K * \cos(px - py)$  ne dépend plus que de la différence des positions

## Encodage de position

En a-t-on vraiment besoin ?

certaines expériences montrent que les résultats expérimentaux se dégradent sans encodage de position, mais cela semble dépendre des tâches.

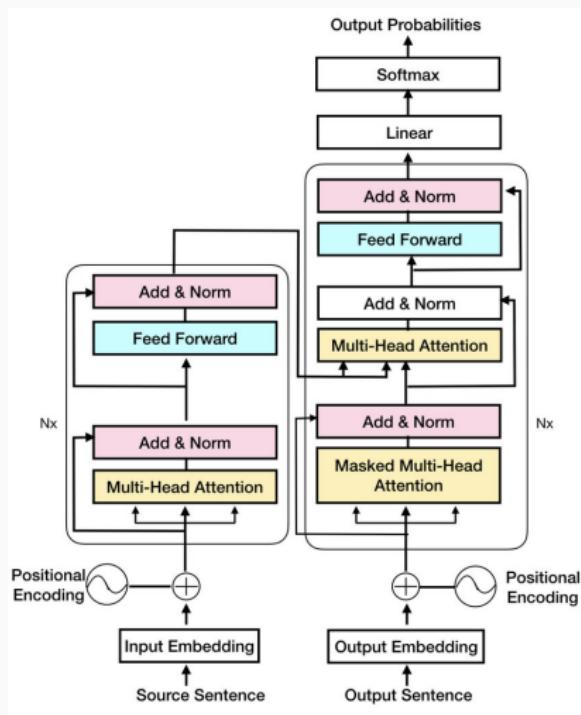
sur certaines tâches textuelles (anglais), pas besoin d'ordre à l'entraînement

Sinha et al. *Masked Language Modeling and the Distributional Hypothesis: Order Word Matters Pre-training for Little*, EMNLP 2021

En pratique les modèles courants actuels intègrent tous un encodage de position

# Transformer original

Encoder-decoder pour la traduction



## Pour quoi faire ?

---

- l'auto-attention est plus expressif que RNN: tout l'input peut utiliser directement des informations d'autres parties
- meilleure utilisation du contexte / moins de restrictions

Est-ce que ça peut aider aussi à contourner les problèmes des vecteurs d'embeddings "fixes" pour les mots ?

- comment pourrait-on entraîner des modèles différemment ?
- mais toujours sans supervision directe ?
- problème plus général d'**auto-supervision**  
"Self-Supervised Learning"
- CBOW, Skip-gram versions "simples" de SSL

# Self-Supervised Learning (SSL)

- utilisation de données "brutes", non annotées pour aider à construire des modèles avec des connaissances sur le langage, qui seront ensuite utilisées dans des tâches différentes  
**modèle préentraîné** : pour l'instant on a vu word2vec
- ou bien apprentissage sur des **pseudo-labels** —> générés automatiquement à partir de
  - attributs des données / méta-données
  - définition de tâches de préentraînement spécifiques

tâche de préentraînement = tâche artificielle/auxilliaire, pour laquelle on a des données naturelles.

## Pretrained language model (PLM)

# Self-Supervised Learning (SSL)

$\neq$  apprentissage non supervisé

- SSL demande une forme de supervision
- apprendre des représentations vs trouver des régularités "cachées"

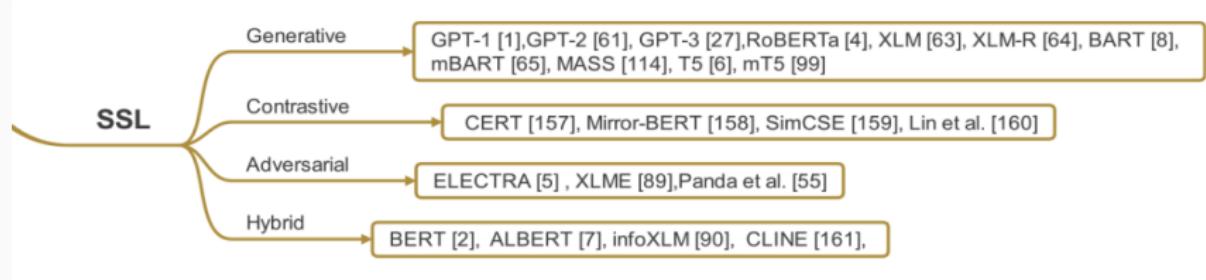
$\neq$  apprentissage semi-supervisé

- le semi-supervisé a besoin d'annotations vs on génère automatiquement des annotations, mais les 2 ont besoin de supervision.
- tache spécifique vs connaissances "universelles".

# Types de SSL

- *Generative SSL* —> prédire des tokens/partie de l'entrée en générant une sortie
  - prédire un mot en fonction des précédents : causal language model (CLM)
  - prédire un mot "masqué" dans une entrée: Masked Language Model (MLM)
- *Contrastive SSL* —> apprendre par comparaison
  - forcer des représentations plutôt au niveau "phrase"
- *Adversarial SSL* —> apprendre à distinguer des entrées corrompues (mots remplacés ou permutés).
- *Hybrid SSL* —> mélanger plusieurs méthodes.

# Types of SSL

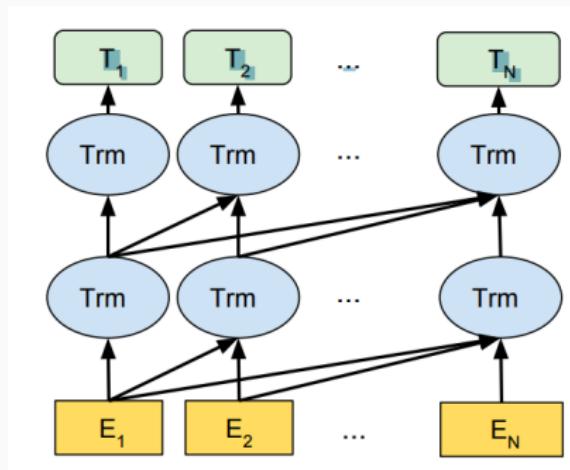


Kalyan et al. 2022. AMMUS : A Survey of Transformer-based Pretrained Models in Natural Language Processing. Arxiv

# Exemple de CLM: GPT<sub>n</sub>

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever.  
2018. Improving language understanding by Generative Pre-Training

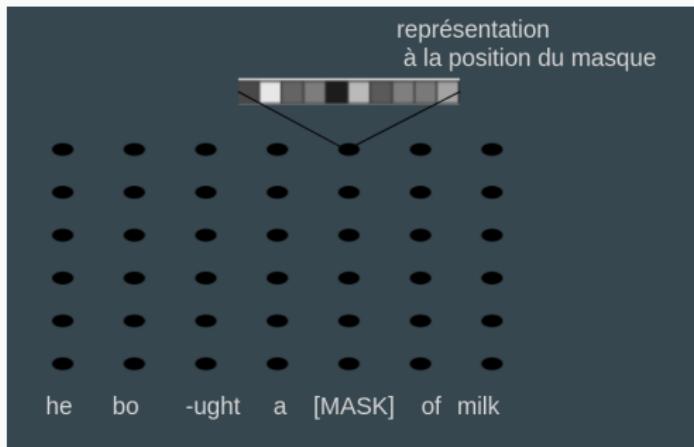
Modèle de langue avec transformers



lien source

# Exemple de MLM: BERT

Modèle de langue avec masquage

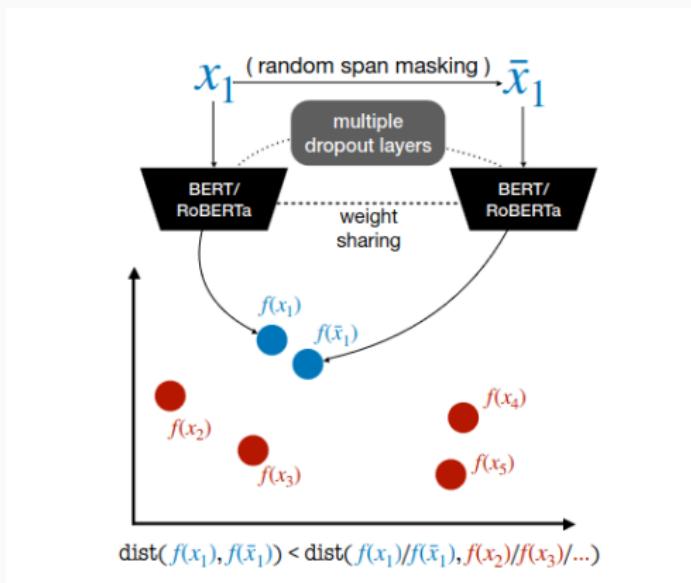


Devlin et al. 2019, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

<https://www.aclweb.org/anthology/N19-1423.pdf>

# Exemple de contrastive SSL: mirrorBERT

Modèle de langue avec masquage

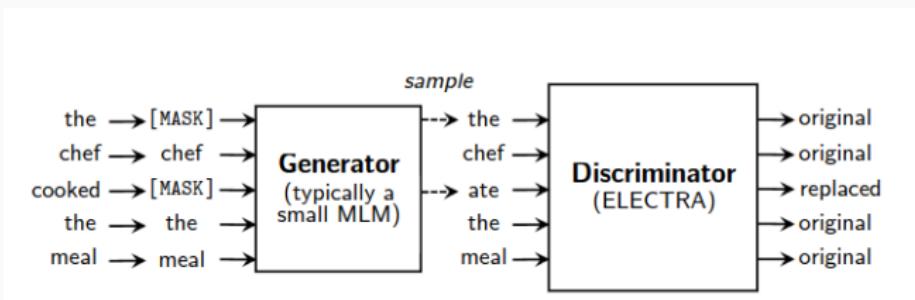


Liue et al. 2021, Fast, Effective, and Self-Supervised: Transforming  
Masked Language Models into Universal Lexical and Sentence Encoders

<https://aclanthology.org/2021.emnlp-main.109/>

# Exemple d'adversarial SSL: Electra

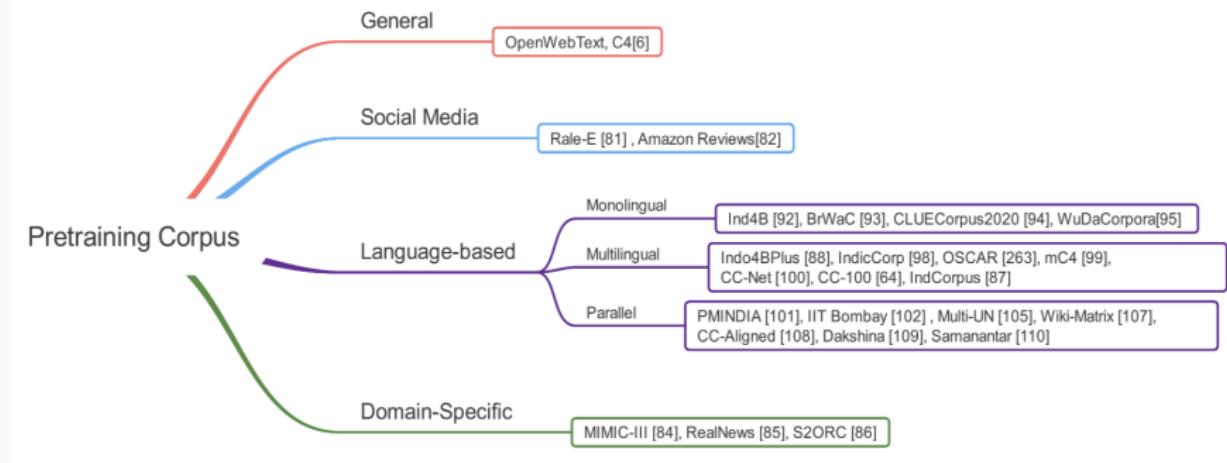
Modèle de langue avec masquage



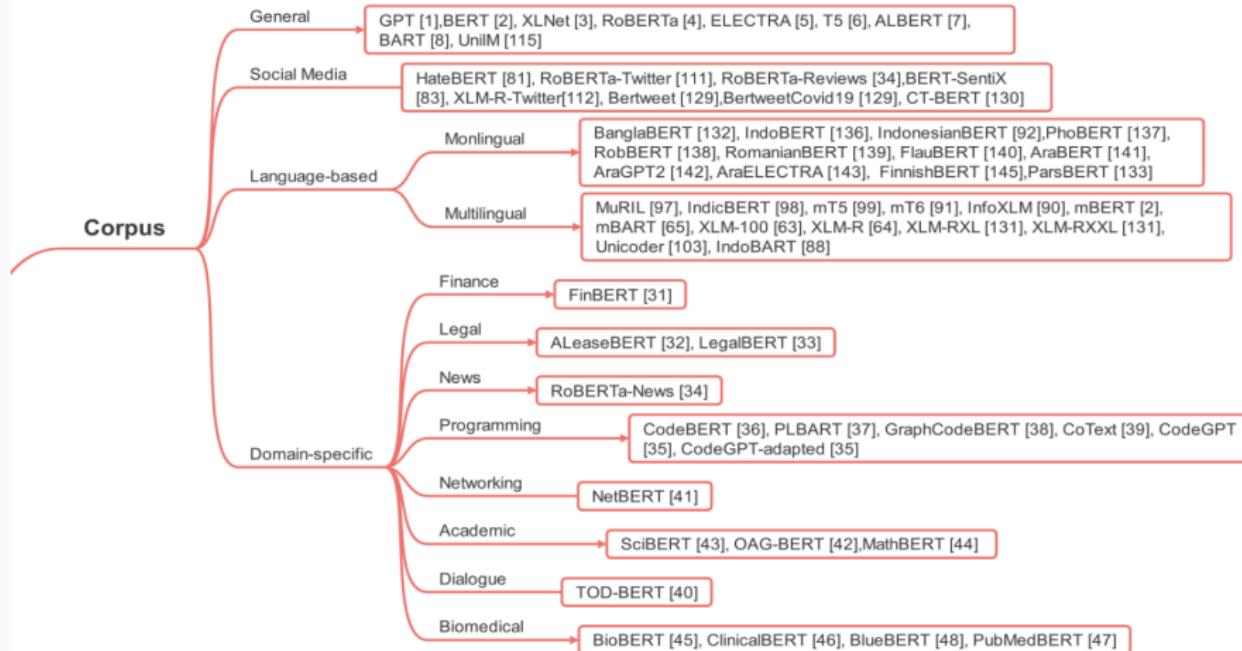
Clark et al. 2020: ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators.

<https://openreview.net/forum?id=r1xMH1BtvB>

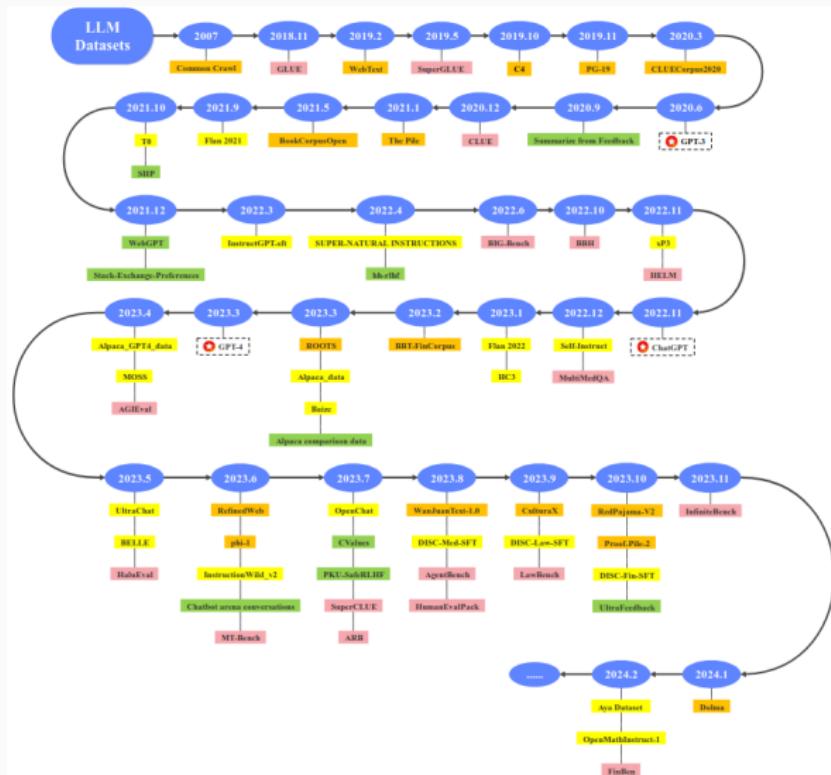
# Corpus de Préentraînement



# Modèles par types de corpus de préentraînement

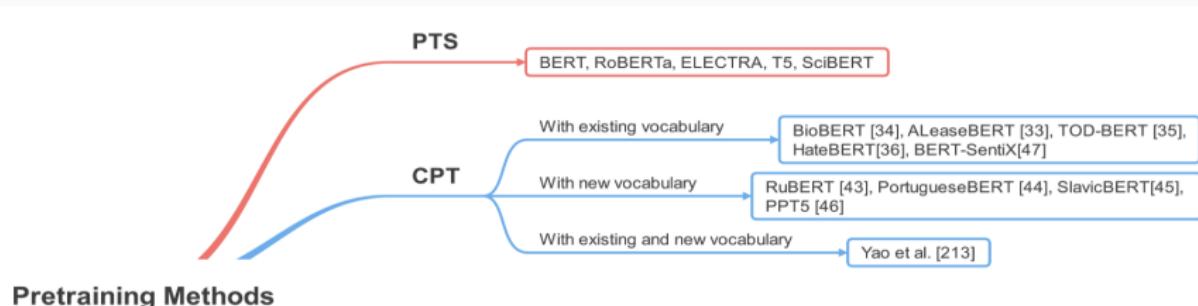


# Corpus de Préentraînement



# Types de pré-entraînement

- *Pretraining from Scratch* (PTS)
  - Encoder/decoder initialisés aléatoirement
  - Paramètres appris en minimisant les pertes sur une ou plusieurs tâches de pré-entraînement
  - intense en calcul / données
- *Continual Pretraining* (CPT) → Part d'un modèle déjà pré-entraîné et continue l'entraînement sur un domaine/corpus spécifique



# Tâches de pré-entraînement

On peut combiner plusieurs tâches

- Masked Language Modelling (MLM)
- Next sentence prediction (NSP) loss
- Sequence-to-Sequence LM (Seq2SeqLM) : modèle causal mais dans un encodeur-décodeur (ex modèles T5)
- perturbation de l'entrée : regénérer l'entrée initiale

# Taches de pré-entraînement

Operation	Element	Original Text	Corrupted Text
Mask	one token	Jane will move to New York .	Jane will [Z] to New York .
	two tokens	Jane will move to New York .	Jane will [Z] [Z] New York .
	one entity	Jane will move to New York .	Jane will move to [Z] .
Replace	one token	Jane will move to New York .	Jane will move [X] New York .
	two tokens	Jane will move to New York .	Jane will move [X] [Y] York .
	one entity	Jane will move to New York .	Jane will move to [X] .
Delete	one token	Jane will move to New York .	Jane move to New York .
	two token	Jane will move to New York .	Jane to New York .
Permute	token	Jane will move to New York .	New York . Jane will move to
Rotate	none	Jane will move to New York .	to New York . Jane will move
Concatenation	two languages	Jane will move to New York .	Jane will move to New York . [/s] 简将搬到纽约。

Kalyan et al. 2022

# Continual Pretraining: exemples

Name	Pretrained from	Pretraining tasks	Corpus	Evaluation
HateBERT [81]	BERT	MLM	RAL-E (dataset of 1.5M hateful Reddit comments)	Offensive tweets classification
RoBERTa-Twitter [111]	RoBERTa	MLM	Tweets (60M)	Tweet classification
RoBERTa-Reviews [34]	RoBERTa	MLM	Amazon reviews (24.75M) [128]	Review classification
BERT-SentiX [83]	BERT	SWP, WP, EP and RP	Amazon (233M) [82] and Yelp reviews (8M) Reviews	Cross domain sentiment analysis
XLM-R-Twitter [112]	XLM-R	MLM	Tweets in multiple languages (198M)	TweetEval [111] and UMSAB [112]
Bertweet [129]	Scratch	MLM	Tweets (845M English + 5M COVID tweets)	POS, NER and Tweets classification
BertweetCovid19 [129]	Bertweet	MLM	COVID tweets (23M)	Tweet classification
CT-BERT [130]	BERT	MLM, NSP	COVID tweets (160M)	Tweet classification

Kalyan et al. 2022

# Une étape préalable cruciale : la tokenisation

Vocabulaire complet d'une langue : trop grand ( 500k-1M)

→ techniques pour décomposer en unités plus fréquentes

"Word pieces": découpage des mots en sous-unités fréquentes  
(algorithme WordPiece)

ou appelés sous-tokens (subtokens)

- diminue la taille du vocabulaire
- limite les mots hors vocabulaire
- permet de généraliser par rapport à la morphologie

Dépend du modèle: "Tokenization is difficult"

- exemple modèle 1: 'token', 'ization', 'is', 'difficult' → BERT
- exemple modèle 2: 'to', 'ken', 'ization', 'is', 'difficult' → XLM

# Différents types de tokenisation

Pre-processing (vocabularies): WordPiece, Byte Pair Encoding (BPE), SentencePiece

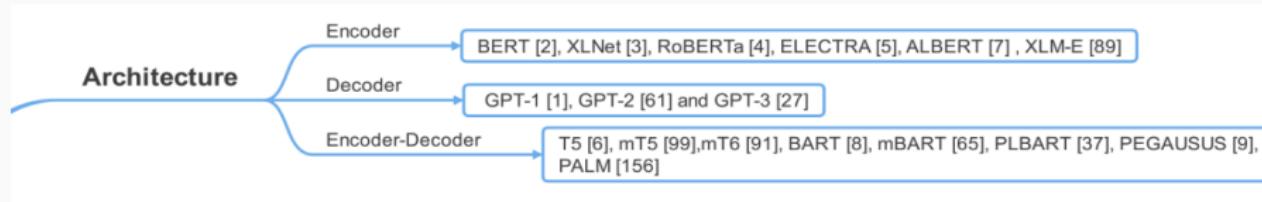
Name	Language	Pretrained from	Pretraining tasks	Corpus	Vocabulary
BanglaBERT [132]	Bangla	Scratch	RTD	Bangla Web text corpus	WordPiece (32k)
IndoBERT [136]	Indonesian	Scratch	MLM	Indonesian Wikipedia, News and Web corpus	WordPiece (32k)
IndonesianBERT [92]	Indonesian	Scratch	MLM	Indo4B	Sentencepiece (30k)
IndonesianBERT-Lite [92]	Indonesian	Scratch	MLM and SOP	Indo4B	Sentencepiece (30k)
PhoBERT [137]	Vietnamese	Scratch	MLM	Vietnamese Wikipedia and News corpus	BPE (64K)
RobBERT [138]	Dutch	Scratch	MLM	OSCAR corpus	bBPE (40K)
RomanianBERT [139]	Romanian	Scratch	MLM,NSP	OPUS, OSCAR and Wikipedia corpus	BPE (50k)
FlauBERT [140]	French	Scratch	MLM	French text corpus	BPE (50K)
AraBERT [141]	Arabic	Scratch	MLM and NSP	Arabic Wikipedia and News corpus	SentencePiece (64K)

# Types d'architectures

---

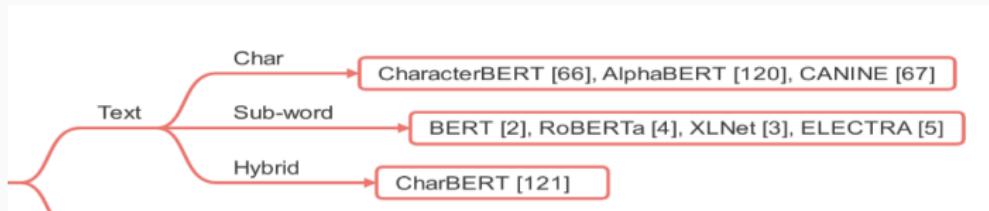
- *Encoder-only*: Embedding + une pile d'encodeurs → pour des tâches de classification/compréhension (NLU)
- *Decoder-only*: Embedding + pile de décodeurs/générateurs → Used in NLG tasks
- *Encoder-decoder based* → pour des tâches séquence-séquence : traduction, résumé etc.
- mais encodeur-décodeur peut aussi tout faire ! (prompting ... on verra plus tard)

# Types d'architectures



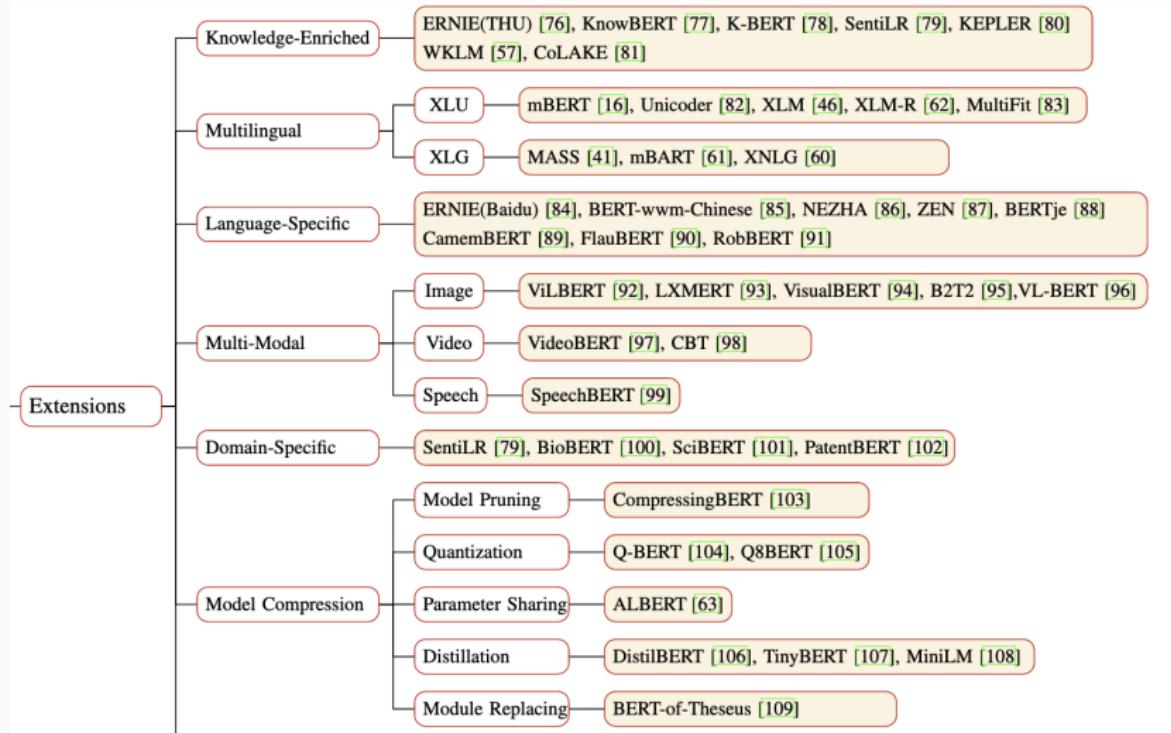
Kalyan et al. 2022

# Types d'Embeddings



Kalyan et al. 2022

# Extensions

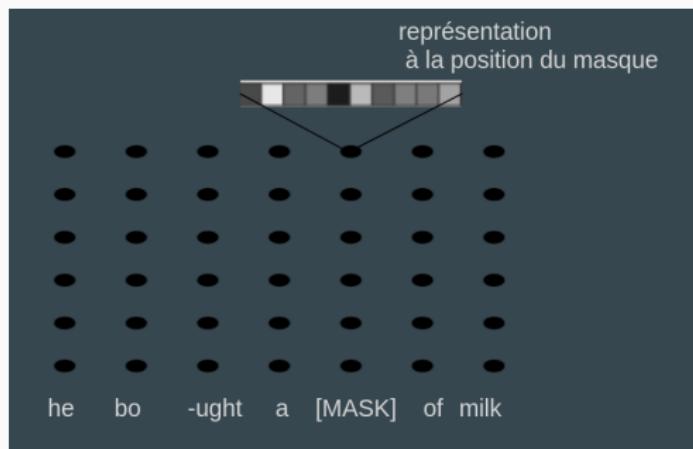


# Implémentation de Transformers

Library	Purpose	Description	Framework	Link
Transformers [278]	Training and Inference	State-of-the-art library for transformer based PTLMs.	Pytorch, Tensorflow and Jax	<a href="https://github.com/huggingface/transformers">https://github.com/huggingface/transformers</a>
SimpleTransformers	Training and Inference	Built on the top of transformers and lets you quickly train and evaluate models.	PyTorch	<a href="https://github.com/ThilinaRajapakse/simpletransformers">https://github.com/ThilinaRajapakse/simpletransformers</a>
HappyTransformer	Training and Inference	Built on the top of transformers and makes the use of state-of-the-art models easy.	PyTorch	<a href="https://github.com/EricFillion/happy-transformer">https://github.com/EricFillion/happy-transformer</a>
FairSeq [279]	Training and Inference	Library to train custom models for translation, summarization, language modeling and other text generation tasks.	PyTorch	<a href="https://github.com/pytorch/fairseq">https://github.com/pytorch/fairseq</a>
AdaptNLP	Training and Inference	Built on the top of Flair and Transformers library and makes the use of state-of-the-art models easy.	PyTorch	<a href="https://github.com/Novetta/adaptnlp">https://github.com/Novetta/adaptnlp</a>
SimpleT5	Training and Inference	Built on top of PyTorch-lightning and Transformers that lets you quickly train your T5 models.	PyTorch-Lightning	<a href="https://github.com/Shivanandroy/simpleT5">https://github.com/Shivanandroy/simpleT5</a>
SpacyTransformers	All NLP tasks	spaCy pipelines for pretrained BERT, RoBERTa XLNet, GPT-2 etc.	PyTorch	<a href="https://github.com/explosion/spacy-transformers">https://github.com/explosion/spacy-transformers</a>
TextBox	Text Generation	Library for building text generation systems based on models like GPT-2, BART, T5 etc.	PyTorch	<a href="https://github.com/RUCAIBox/TextBox">https://github.com/RUCAIBox/TextBox</a>

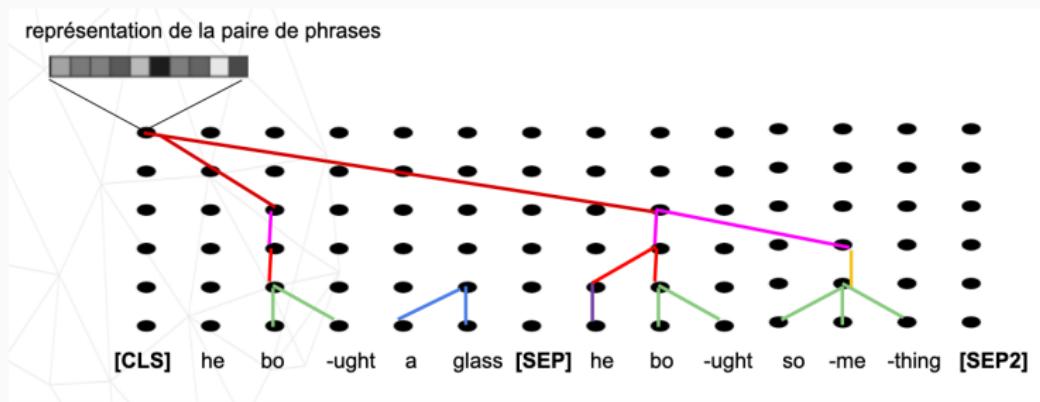
# Utilisation de PLM, exemple BERT

Modèle de langue avec masquage



## Utilisation de PLM, exemple BERT

L'entraînement intègre aussi deux phrases avec la tâche de prédire si elles se suivent ou si ce sont deux phrases choisies aléatoirement



Et un token spécial “CLS” qui sert pour les nouvelles tâches

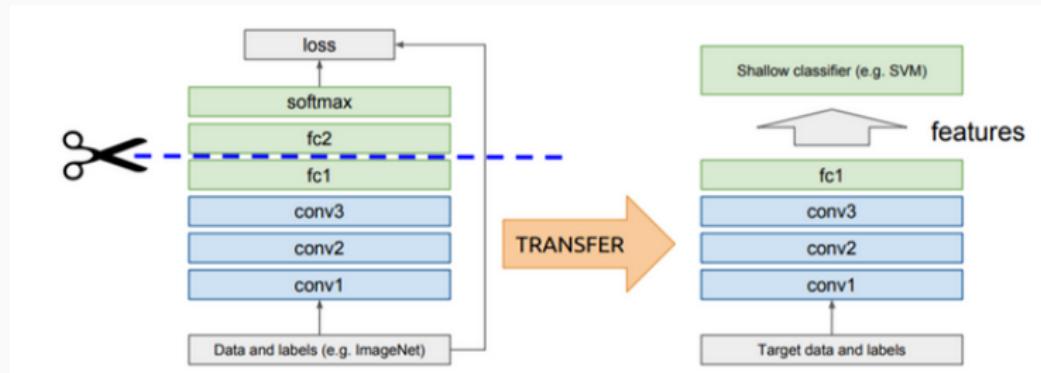
**ELMO, GPT, Bert** sont des modèles intrinsèquement prévu pour faire de l'apprentissage de nouvelles tâches par dessus le modèle préentraîné = transfert / fine-tuning

## Et une fois qu'on a un modèle préentraîné ?

---

- on l'adapte à une "vraie" tâche : **fine-tuning**
- adaptation de domaine / transfert
- combinaison multi-tâches

# Apprentissage par Transfert

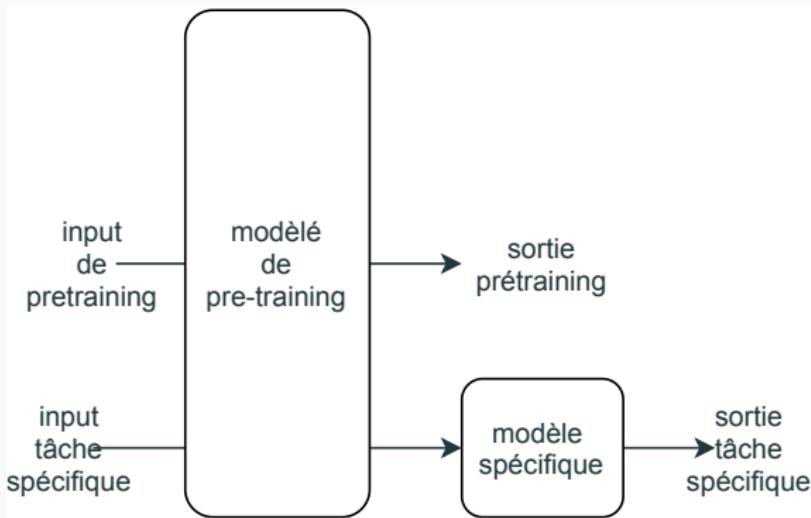


en TAL

- word2vec comme initialisation de word embeddings
- BERT/GPT/Elmo/... pour la classification de texte, l'étiquetage et (BERT seulement) les relations textuelles
- Modèles de langue unidirectionnel (GPT) ou architecture spécifique (BART) pour les tâches avec encodeur-décodeur

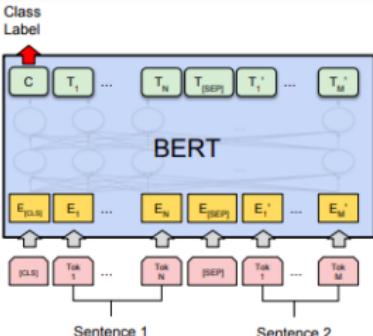
# Fine tuning

Cas particulier: on garde tout le modèle de base

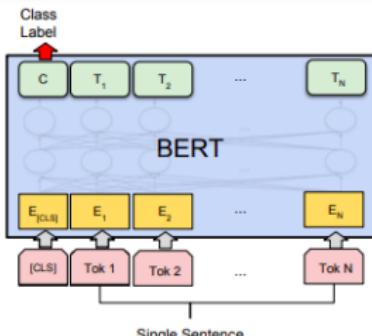


Et on spécialise pour une tâche spécifique.

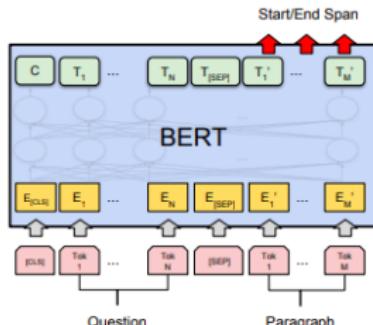
# Exemple: Fine-tuning avec BERT



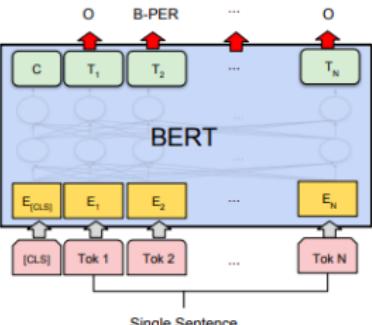
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(b) Single Sentence Classification Tasks:  
SST-2, CoLA



(c) Question Answering Tasks:  
SQuAD v1.1



(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

# Benchmark d'évaluation en transfert

Regroupement de tâches pour évaluation d'apprentissage par transfert

GLUE <https://gluebenchmark.com/>

SuperGLUE <https://super.gluebenchmark.com/>

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>92.7</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>82.1</b>

GLUE / (BERT 2018) → aujourd'hui ?  $\approx 93.3$  (variante d'Electra en 2022)

SuperGlue ? (Bert 2018) 69.4 → aujourd'hui ?  $\approx 91$  (modèle génératif aout 2024 / PLM Vega 2022)

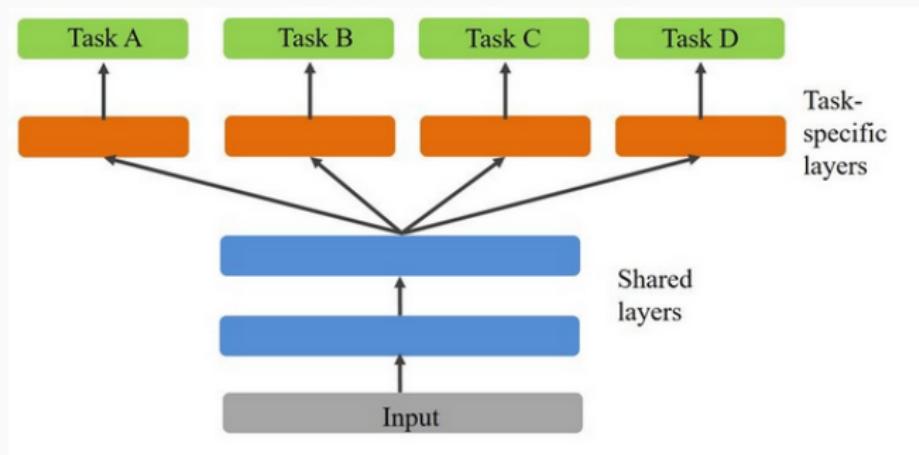
# Apprentissage Multi-tache

MTL: Multi-task learning

= apprendre ensemble des problèmes reliés

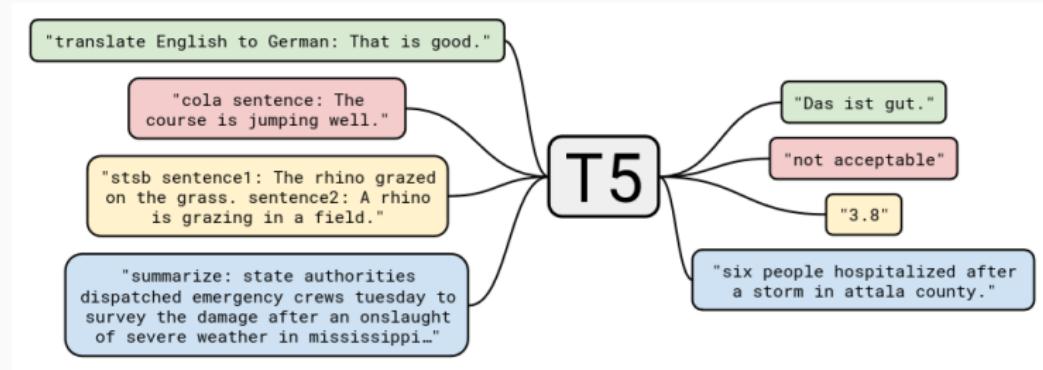
en TAL: combiner plusieurs tâches pendant le pré-entraînement

BERT: 2 ; T5  $\approx$  20; Flan-T5  $\approx$  2000



mais en TAL le "branchement" de la tâche est faite par l'entrée

## exemple: T5



T5: *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer* (2020)

Flan: *Scaling Instruction-Finetuned Language Models* (2022)

## Généralisation: le *prompting*

Au lieu de signaler la tâche dans l'entrée, on peut aussi la décrire explicitement dans l'entrée, avant la vraie instance : "prompting"

Soit avec uniquement une instruction (*0-shot*), soit avec quelques exemples (*few-shot*, aussi appelé In Context Learning)

On utilise un générateur de texte comme prédicteur :

**Choose the sentiment of the given text  
from Positive and Negative.**

**Text:** a feast for the eyes

**Sentiment:** Positive

...

**Text:** boring and obvious

**Sentiment:** Negative

**Text:** [Unlabeled Data]

**Sentiment:** [Label]

*Is GPT-3 a Good Data Annotator?* (Ding et al., ACL 2023)

*Want To Reduce Labeling Cost? GPT-3 Can Help* (Wang & al., EMNLP21)

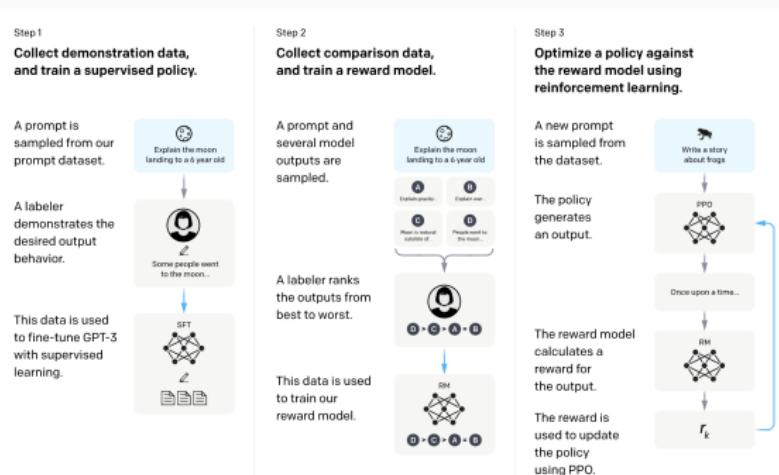
*Language models are few-shot learners* (2020) → GPT3

## Modèles de langue "conversationnels"

En fait "juste" des modèles génératifs qui laisse un utilisateur insérer des portions de texte

+ prompt (invisible)

+ correction des sorties avec un entraînement spécifique : "reinforcement learning with human feedback"



# La magie noire du prompting

La technique du prompting semble très puissante, mais très sensible à la formulation, et beaucoup d'améliorations empiriques ont été proposées, sans garantie.

"prompt engineering"

- chain-of-thought ("let's think step-by-step")
- retrieval/knowledge augmented generation
- multiple generations ("self-consistency")
- role giving ("you are a financial expert")
- décomposition de prompt ("prompt chaining")

cf <https://www.promptingguide.ai/fr>

aussi *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing* ACM Survey 2023

<https://dl.acm.org/doi/10.1145/3560815>

# Exemple: chain of thought

## Standard Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The answer is 27. X

## Chain-of-Thought Prompting

### Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had  $23 - 20 = 3$ . They bought 6 more apples, so they have  $3 + 6 = 9$ . The answer is 9. ✓

Chain-of-Thought Prompting Elicits Reasoning in Large Language Models (Neurips 2022)

# Evolution de l'évaluation

- transfert: fine-tuning sur tâches séparées
- benchmark regroupant plusieurs tâches (Glue, etc)
- approches génératives: FT mais aussi 0/k-shot learning
- évaluation des modèles génératifs par QCM (MMMU)
- ou benchmark à N tâches de plus en plus gros: BIGBench, MegaBench

# Premier Bilan

- pré-entraînement très efficace pour produire des représentations contextuelles
- très bon résultats en transfert
- pouvoir impressionnant de l'approche par génération

Limites:

- restrictions sur la taille de l'entrée; performances sur textes longs
- contrôle de la génération délicat
- manque de transparence sur les données de pré-entraînement ; questions sur la vraie généralisation
- gros problèmes de contrôle/biais (plus tard)