Modèles de langage

Julien Romero

Introduction

Brève histoire de la linguistique

- -6.000 avant J.C.: Invention de l'écriture en Mésopotamie
- -2.000 avant J.C.: Listes lexicales (glossaires, dictionnaires, traductions) en Mésopotamie
- -1.200 avant J.C.: Étude de la morphologie (structure des mots) et de la phonétique pour les textes religieux en Inde
- -600 avant J.C.: Pānini crée les premières règles de grammaire en Inde
- -300 avant J.C.: Aristote étudie la rhétorique
- 18ème : Début de la linguistique moderne (liens avec psychologie, biologie)
- 20ème : La linguistique devient une discipline avec une division en sous-disciplines: phonologie, morphologie, syntaxe, sémantique, ...

Traitement automatique du langage naturel

- 1950-1990: approche symbolique, inspirée de la linguistique. On donne des règles, et on laisse l'ordinateur les exploiter.
- 1990-2010: approche statistique. Nous avons suffisamment de données et de puissance pour utiliser des outils statistiques et du machine learning.
- 2010-: approche neuronale. Basée sur des réseaux de neurones profonds.

Comment représenter un texte ?

- La manière dont nous allons représenter les textes aura un impact sur la modélisation.
  - Nous allons décomposer notre texte en une suite de d'objets (appelés tokens) : c'est la **tokenization**
  - L'ensemble des tokens possibles est appelé le **vocabulaire**
- Tokenization naturelle en informatique : les caractères
  - On sépare tous les caractères de la phrase
  - "Le renard" devient ["L", "e", " ", "r", "e", "n", "a", "r", "d"]
  - Avantages : Facile, on peut tout représenter
  - Inconvénients : Séquences longues, peu d'informations sémantiques utiles au niveau des lettres

Comment représenter un texte ?

- La manière dont nous allons représenter les textes aura un impact sur la modélisation.
  - Nous allons décomposer notre texte en une suite de d'objets (appelés tokens) : c'est la **tokenization**
  - L'ensemble des tokens possibles est appelé le **vocabulaire**
- Tokenization naturelle pour les humains : les mots
  - On sépare tous les mots d'une phrase
  - ["Le renard mange la poule"] devient ["Le", "renard", "mange", "la", "poule"]
  - Avantages : Séquences courtes, chaque token a une signification
  - Inconvénients : Large vocabulaire (on doit le limiter), non flexible avec les fautes, variantes d'un mot, beaucoup de mots hors du vocabulaire, modèle très larges

Comment représenter un texte ?

- La manière dont nous allons représenter les textes aura un impact sur la modélisation.
  - Nous allons décomposer notre texte en une suite de d'objets (appelés tokens) : c'est la **tokenization**
  - L'ensemble des tokens possibles est appelé le **vocabulaire**
- À partir de maintenant, nous représenterons un texte comme une succession de tokens $w_1 w_2 ... w_n$
  - Les w_i peuvent représenter des mots, ou des lettres suivant la tokenization choisie

Modélisation

Qu'est-ce qu'un modèle de langage?

Étant donné une séquence composée de plusieurs mots $w_1 w_2 ... w_n$, nous voulons savoir à quel point cette phrase est probable. Autrement dit, nous voulons :

$$P(w_1, w_2, \ldots, w_n)$$

Exemples:

- "Le renard mange la poule." a une plus forte probabilité que "La renard le poule"
- "Il vient de Paris. Son train est en retard." a une plus forte probabilité que "Il vient de Paris. Le renard mange la poule.". (Le contexte est important).

Applications des modèles de langage

- Reconnaissance vocale
  - Les données sont bruitées, un modèle de langage aide à désambiguïser
  - Ex: “La vache mange l’herbe.” et “La hache vange l’air” sont phonétiquement proches, mais la première est plus probable.
- Correction d’orthographe/de grammaire
  - Ex: “Les vaches mangent de l’herbe” vs “Les vaches mange de l’herbe”
- Traduction
- Chatbot
  - Ex: “- Où puis-je manger ce soir? - Il y a un restaurant japonais au bout de la rue.” vs “- Où puis-je manger ce soir? La capitale de la France est Paris.”

Décomposition d'un modèle de langage

$P(w_1, w_2, ..., w_n)$ est trop compliqué à calculer d'un coup. On doit donc la décomposer.

- Décomposition causale:

$$P(w_1, w_2, ..., w_n) = P(w_1)*P(w_2|w_1)*P(w_3|w_1w_2) ...P(w_n|w_1...w_{n-1})$$

(On cherche la probabilité du mot suivant étant donné les mots précédents)

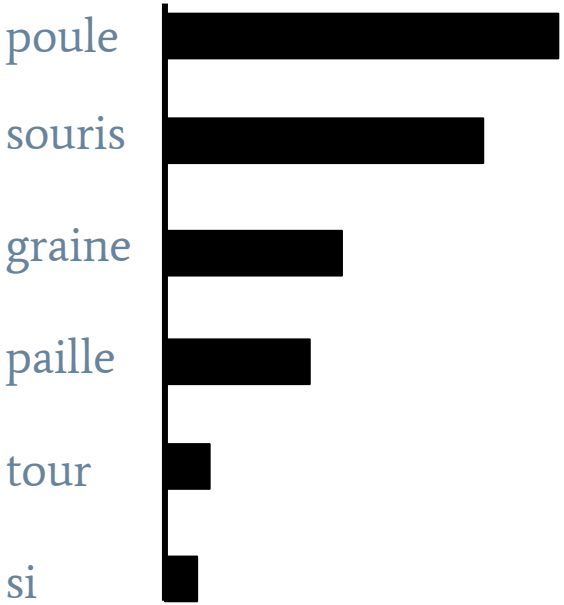
- Décomposition avec un masque

$$P(w_1, w_2, ..., w_n) = P(w_k|w_1...w_{k-1}w_{k+1}...w_n) * P(w_1...w_{k-1}w_{k+1}...w_n)$$

Que représentent ces probabilités ?

- On a la probabilité d'un mot en fonction du context

P(w|Le renard mange la)



Quand utiliser quelle décomposition?

- Décomposition causale: Marche bien quand on doit générer du texte.
  - Chatbot, résumé de texte, traduction
  - La probabilité des premiers mots prend peu de contexte.
- Décomposition avec un masque: Quand on a besoin d'une représentation contextuelle (avant le mot et après le mot) pour chaque mot.
  - Assignation d'un label à chaque mot (verbe, nom, adjectif, ...), classification de phrases
  - Ne permet pas de générer du texte de manière convenable (on apprend à prédire un mot en ayant déjà tout le reste de la phrase, pas juste le début).

$$P(w_1, w_2, ..., w_n) = P(w_k | w_1 ... w_{k-1} w_{k+1} ... w_n) * \underbrace{P(w_1 ... w_{k-1} w_{k+1} ... w_n)}_{?}$$

Comment calculer les probabilités?

$$P(w_1, w_2, ..., w_n) = P(w_1)*P(w_2|w_1)*P(w_3|w_1w_2) ...P(w_n|w_1...w_{n-1})$$

On va vouloir estimer chacune des probabilités $P(w_k|w_{<k})$

Cette décomposition est très utile pour générer du texte. Dans certaines applications comme la traduction, on peut s'intéresser à d'autres probabilités.

Première solution : statistiques pures (1990)

$$P(w_k \mid w_{<k}) = \frac{Nbr\ observations\ w_1 w_2 \ldots w_k}{Nbr\ observations\ w_1 w_2 \ldots w_{k-1}}$$

Problème : Plus k est élevé, moins on a d'observations (voire pas du tout).

On doit donc faire des approximations N-grams:

- Bigram: $P(w_k \mid w_{<k}) \approx P(w_k \mid w_{k-1})$
- Trigram: $P(w_k \mid w_{<k}) \approx P(w_k \mid w_{k-1}, w_{k-2})$

Augmenter trop le nombre de mots considérés donne des probabilités imprécises et difficiles à stocker (Google s'est arrêté à environ 1 milliard de 5-grams).

Deuxième solution: Word2Vec, un vecteur par mot, sans ordre (2013)

- On voudrait un vecteur (appelé **embedding**) par mot qui "contiendrait" les informations nécessaires pour approximer nos probabilités.
- Pour faciliter les calculs, on suppose que ***l'ordre des mots n'est pas important*** .
- Deux approches principales:
  - Continuous Bag Of Words: À partir du contexte autour d'un mot, on veut prédire le mot ciblé.

$$\frac{1}{T}\sum_{k=1}^{n} P(w_k \mid w_1, \ldots, w_{k-1}, w_{k+1}, \ldots, w_n)$$

  - Skip-gram: À partir d'un mot cible, on veut prédire tous les mots du contexte.

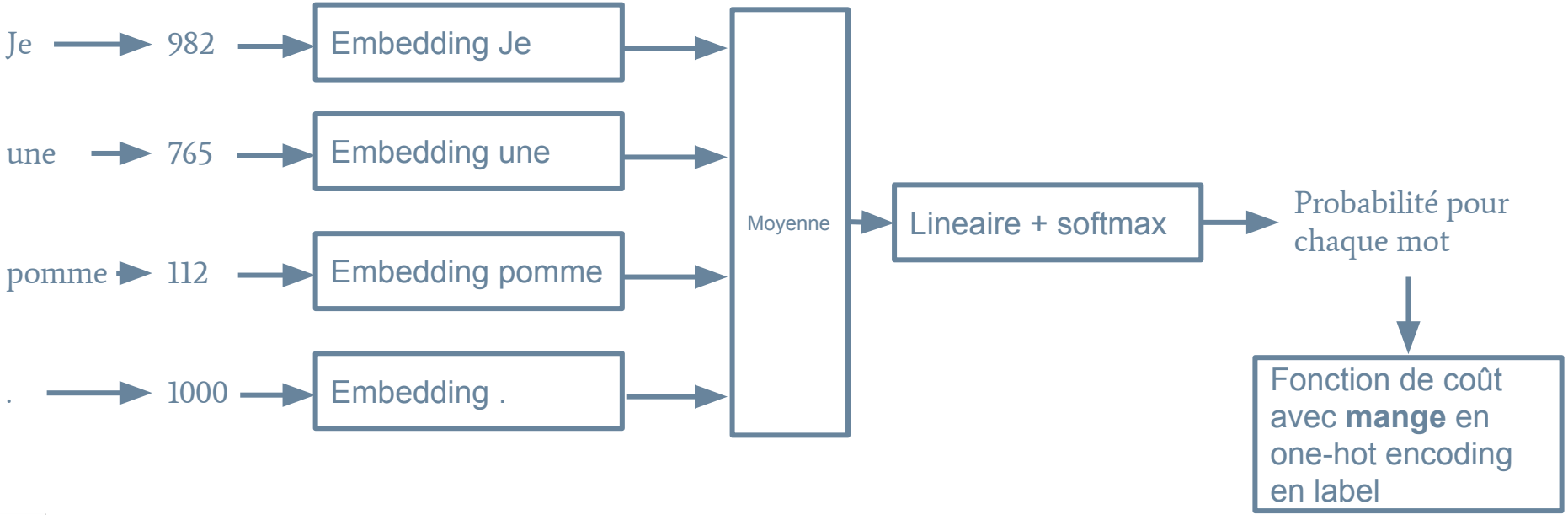
$$\frac{1}{T}\sum_{i=1}^{n}\sum_{j\neq i} P(w_j \mid w_i)$$

Les embeddings

- Embedding = encodage de tokens sous forme de vecteurs de réels
- En pratique :
  - Chaque token du vocabulaire a un indice unique. Tous les indices sont continus en partant de 0.
  - Chaque embedding a une taille E, et tous les embeddings sont stockés dans une matrice de taille VxE où V est la taille du vocabulaire
  - On transforme les indices en une séquence de one-hot encoding, puis on multiplie avec la matrice des embeddings

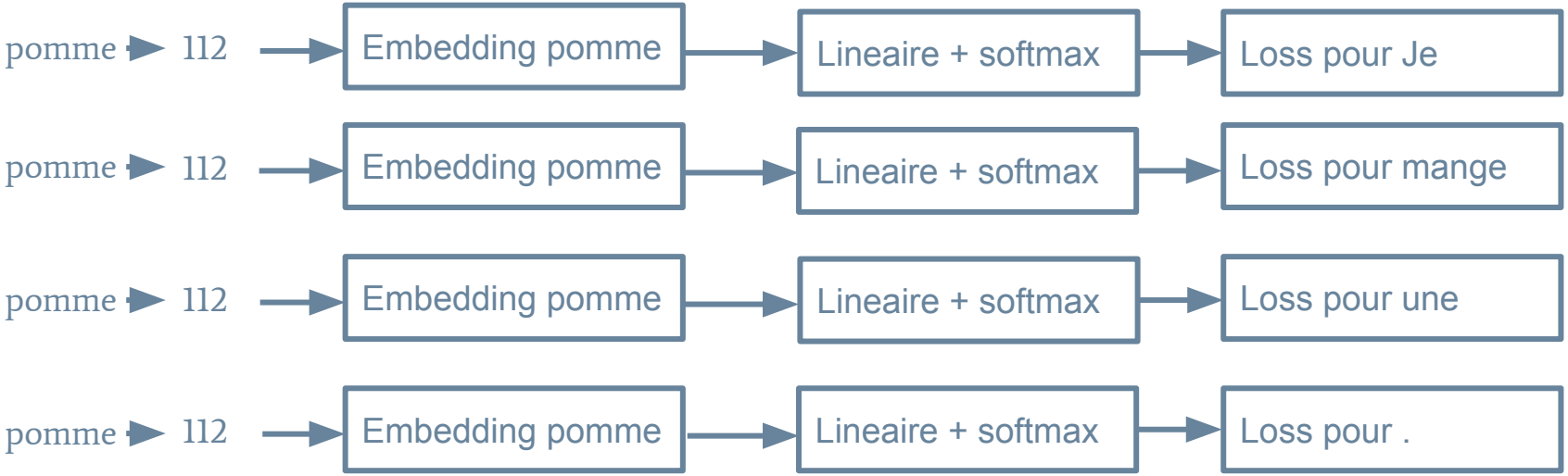
Deuxième solution: un vecteur par mot - Exemple CBoW

Sentence: Je **mange** une pomme.



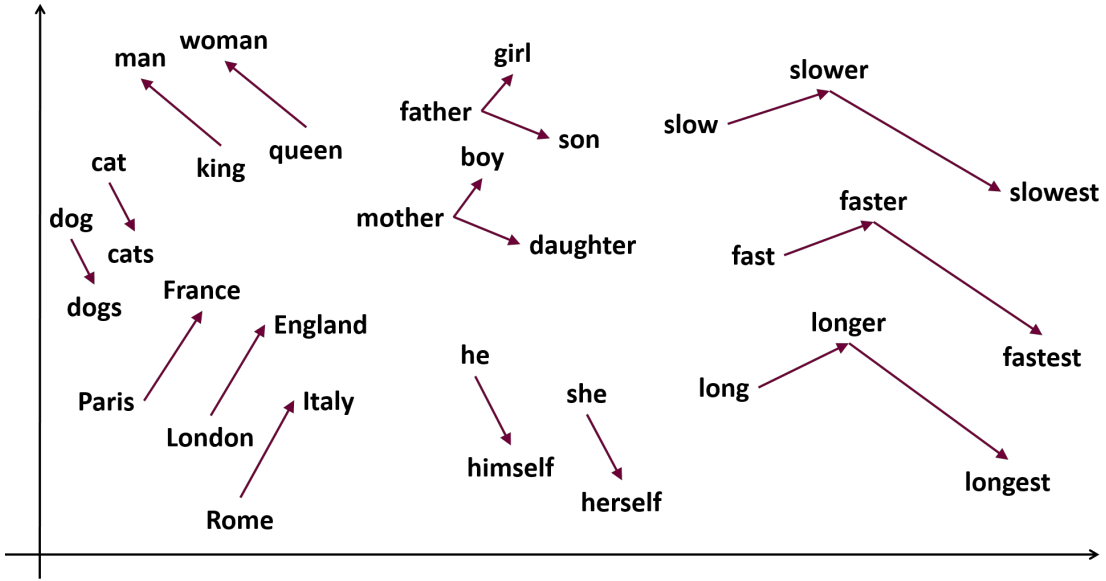
Deuxième solution: un vecteur par mot - Exemple Skip-Gram

Sentence: Je **mange** une pomme.



Interprétation de Word2Vec

On peut faire de l'arithmétique avec les mots : king - man + woman = queen



Word2Vec toujours utile ?

- Représentation vectoriel de mots, même hors contexte
- Léger
- Pas besoin de GPU pour l'utiliser
- Nécessite relativement peu de données pour l'entraînement

Quelle est la taille de ces modèles ?

- Taille des embeddings : V*E
  - V = taille du vocabulaire, E dimension des embeddings
- Couche linéaire : E * V

Quelle est la taille de ces modèles ?

Dans le Word2Vec de Gensim, V = 662109 et E = 300

Taille des embeddings = $2*10^8$ paramètres, soit 1.5Gb

Même chose pour la couche linéaire, et on doit encore rajouter :

- Les résultats intermédiaires du forward pass (proportionnel à la batch size)
- Les calculs intermédiaires de Adam (2 floats par paramètre)
- Les gradients (nombre de paramètre)

Soit au moins environ 3Gb * 5 = 15Gb !

Et nous n'avons qu'une seule couche linéaire !

Comment gérer les vocabulaires trop grands ?

- On veut :
  - Suffisamment de tokens pour que chaque token puisse avoir une interprétation sémantique
  - Pas trop de tokens pour ne pas faire exploser la mémoire
  - Pouvoir tout écrire et prévoir les fautes de frappe, les mots rares, les variations des mots, ...
- En gros, quelque chose entre que des caractères et que des mots
- Solution : Byte-Pair Encoding (BPE)

Byte-Pair Encoding

- Idée : On part des caractères uniques, puis on ajoute petit à petit les paires de tokens les plus fréquentes dans le vocabulaire

Algorithme :

Input : un corpus de textes, taille de vocabulaire V
Output : une liste des transformations à appliquer pour tokenizer le texte

1. Récupérer les caractères uniques dans le corpus. Ce sont les tokens initiaux.
2. Tokeniser tous les mots du vocabulaire avec ces tokens
3. Tant que le nombre de tokens < V
   a. Trouver la paire de tokens t_1t_2 la plus fréquente dans le corpus. La rajouter aux tokens.
   b. Fusionner t_1 et t_2 dans tous les mots du corpus
4. Retourner la liste de tokens (dans l'ordre)

Byte-Pair Encoding - Exemple

Corpus = [("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)]

Au début, tokens = ["b", "g", "h", "n", "p", "s", "u"]

Première tokenization = ("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)

Paire la plus fréquente : ("u", "g") -> "ug"

Retokenization = ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)

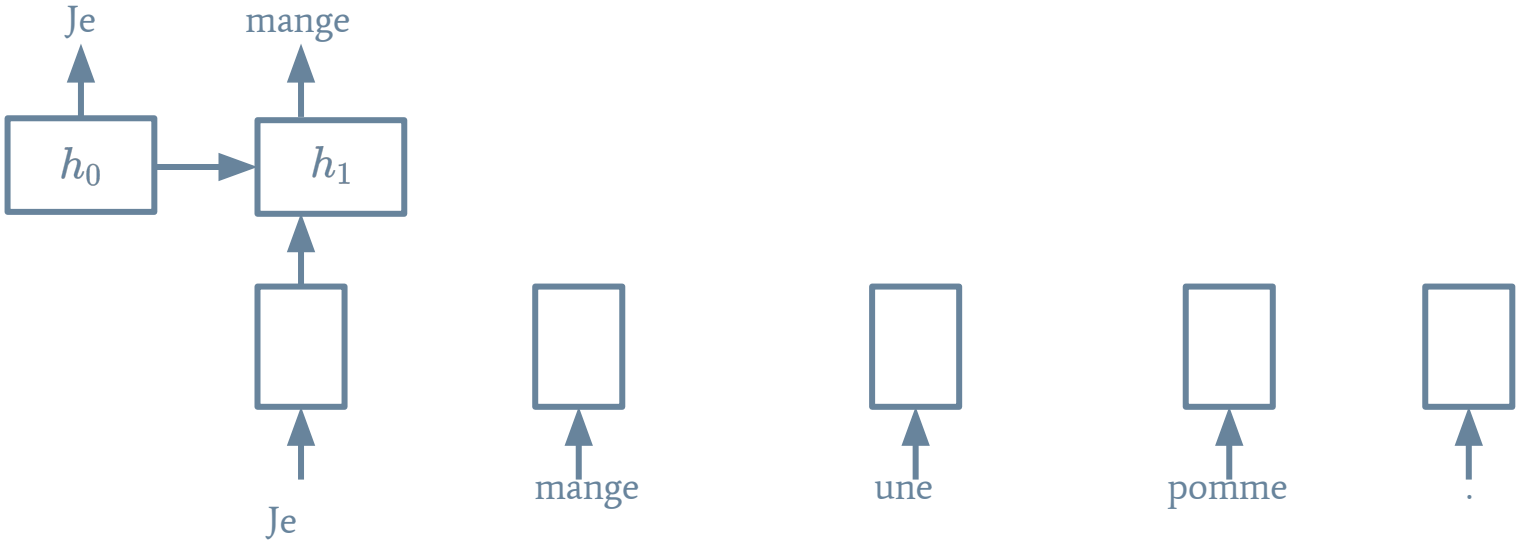
Les tokens spéciaux

Souvent, les modèles de langage ajoutent des tokens supplémentaires pour encoder certaines situations :

- [EOS], <|end_of_text|> : fin du texte
- <|begin_of_text|> : début du texte
- <|start_header_id|> : donner un rôle dans un chat (user vs assistant)
- [MASK] : cacher un mot

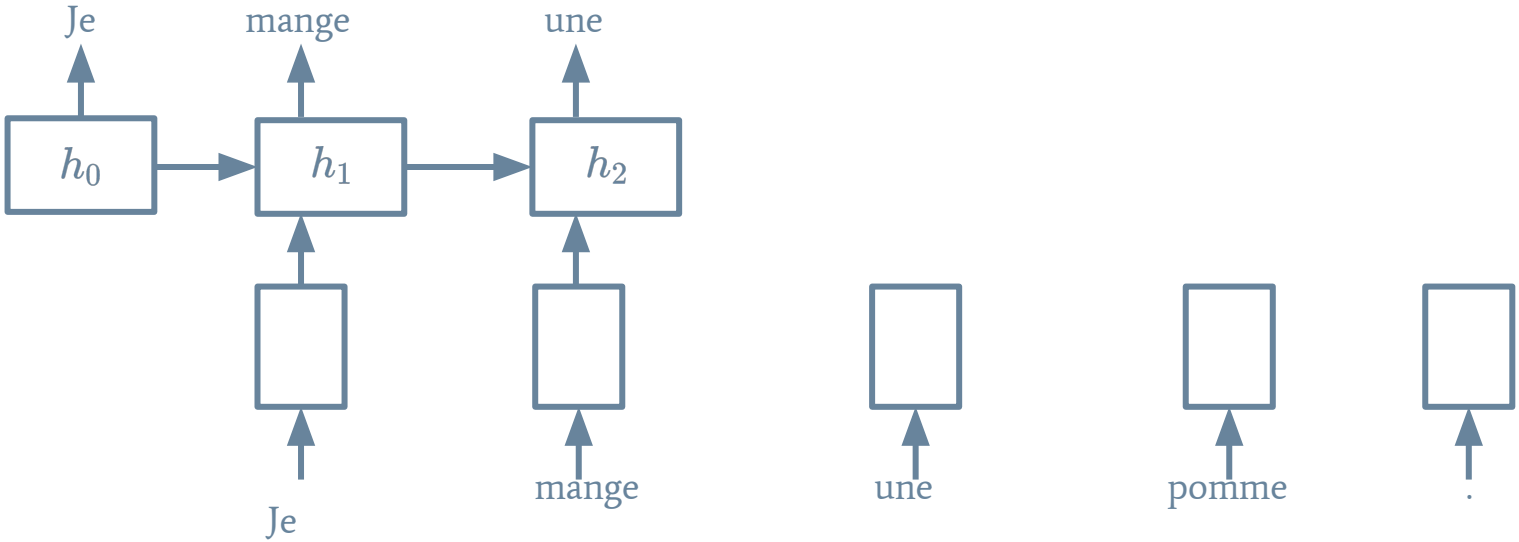
Troisième solution: réseau de neurones récurrents

- On veut prendre en compte l'ordre des mots.
  - On va représenter le contexte précédent avec un vecteur $P(w_k \mid w_{<k}) \approx P(w_k \mid w_{k-1}, h_{k-1})$
  - On garde toujours un vecteur par mot



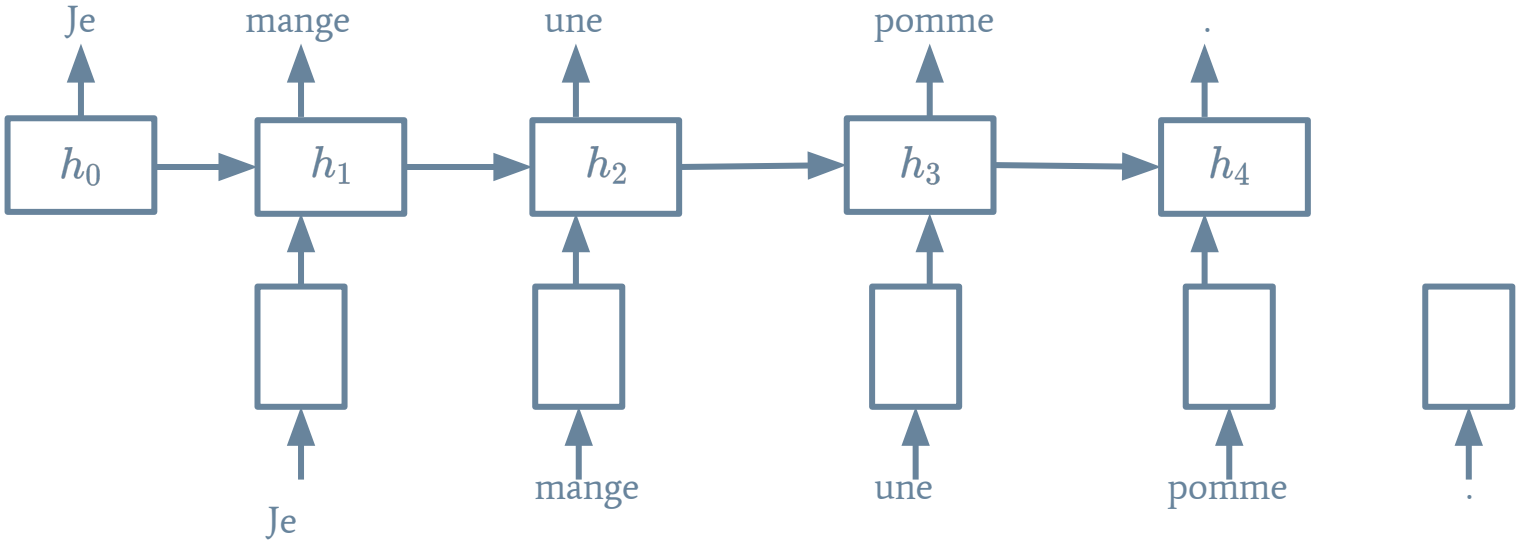
Troisième solution: réseau de neurones récurrents

- On veut prendre en compte l'ordre des mots.
  - On va représenter le contexte précédent avec un vecteur $P(w_k \mid w_{<k}) \approx P(w_k \mid w_{k-1}, h_{k-1})$
  - On garde toujours un vecteur par mot



Troisième solution: réseau de neurones récurrents

- On veut prendre en compte l'ordre des mots.
  - On va représenter le contexte précédent avec un vecteur $P(w_k \mid w_{<k}) \approx P(w_k \mid w_{k-1}, h_{k-1})$
  - On garde toujours un vecteur par mot

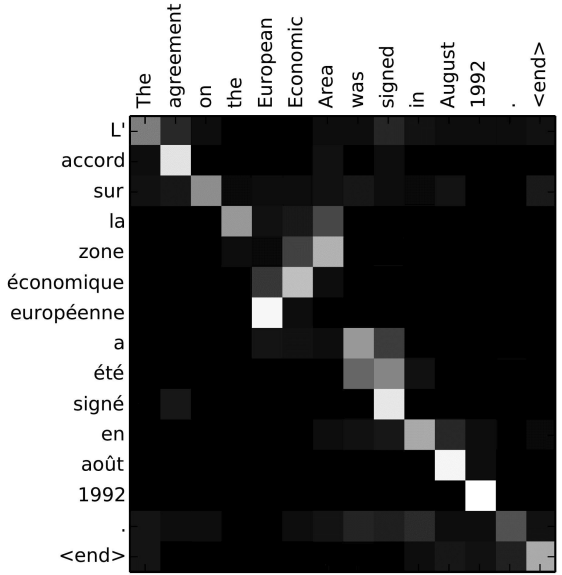


Quatrième solution: Transformers (2017)

- Problèmes des réseaux récurrents:
  - Difficile de se “souvenir” des mots très éloignés.
  - Difficile à entraîner dans le cas des grands contextes.
  - Difficile de paralléliser.
- Solution:
  - On donne tous les mots précédents + **une indication sur la position**
  - On veut que le modèle apprenne à choisir le contexte pertinent (mécanisme d’attention)

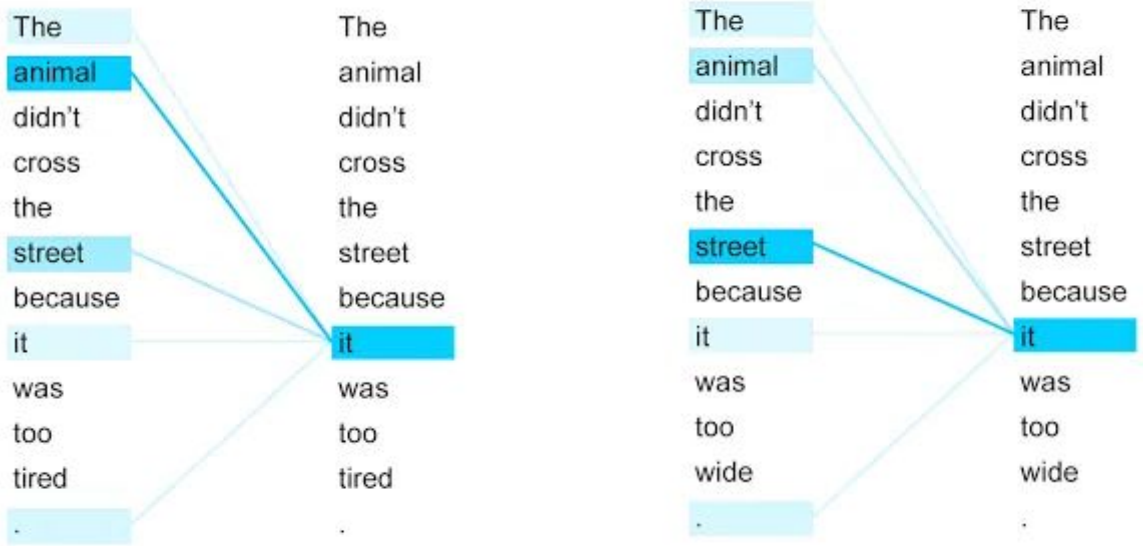
Quatrième solution: Transformers (2017) - Self-attention

- L'attention est un mécanisme associant un poids à chacune des entrées du réseau
  - Quand le poids est calculé uniquement à partir de l'entrée elle-même, on parle de self-attention.
- Exemple attention: Traduction
  - Neural Machine Translation by Jointly Learning to Align and Translate
  - L'attention utilise l'entrée + le début de la traduction séparément



Quatrième solution: Transformers (2017) - Self-attention

- L'attention est un mécanisme associant un poids à chacune des entrées du réseau
  - Quand le poids est calculé uniquement à partir de l'entrée elle-même, on parle de self-attention.
- Exemple self-attention



Positional Encoding

- Problème de l'attention : l'ordre de l'entrée n'a pas d'importance
  - Il faut donc encoder la position d'un mot directement dans l'entrée, en plus de l'embedding du mot
  - C'est du positional encoding
- Première solution : utiliser un entier indiquant la position
  - Peut devenir très grand
  - Variations trop petites après la normalisation (comment normaliser si la taille de la séquence d'entrée est variable ?)

Absolute Positional Encoding

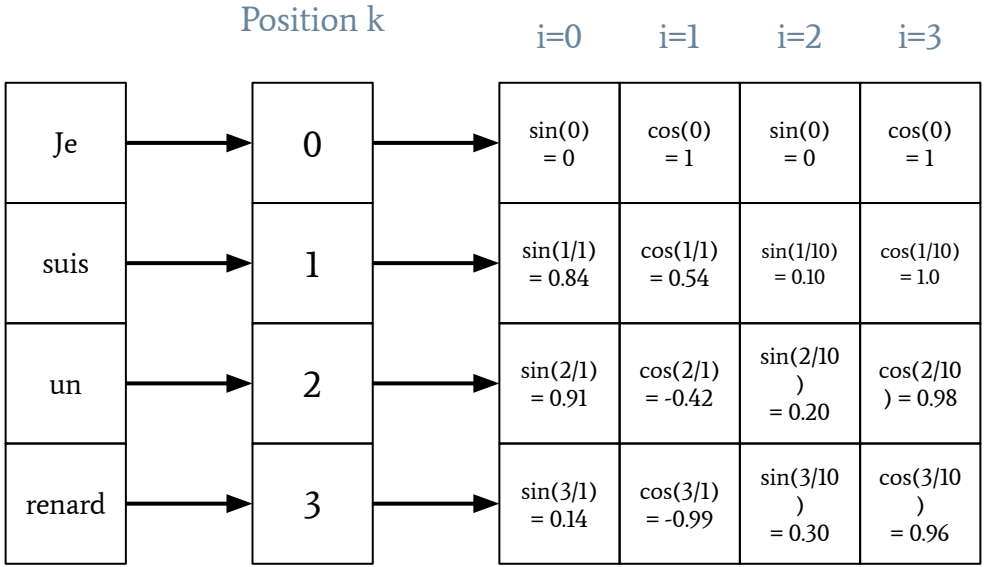
- Problème de l'attention : l'ordre de l'entrée n'a pas d'importance
  - Il faut donc encoder la position d'un mot directement dans l'entrée, en plus de l'embedding du mot
  - C'est du positional encoding
- Deuxième solution : Utiliser plusieurs dimensions
  - Tous les vecteurs doivent être uniques
  - On doit pouvoir comparer deux positions
  - Chaque dimension est entre -1 et 1 (normalisé naturellement entre -1 et 1)

Dans Transformers :

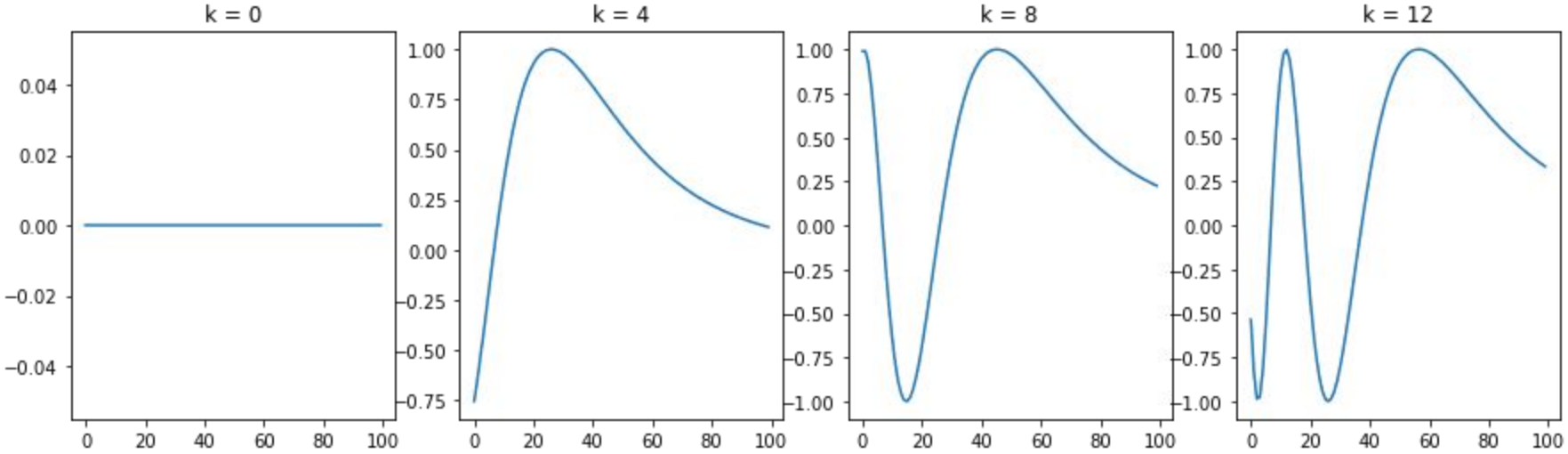
$$P(k, 2i) = sin(\frac{k}{n^{2i/d}}) \qquad P(k, 2i+1) = cos(\frac{k}{n^{2i/d}})$$

Avec k la position dans la phrase, 2i et 2i+1 la dimension dans le vecteur de position, et n un hyperparamètre (10k à l'origine), d = dimension de sortie

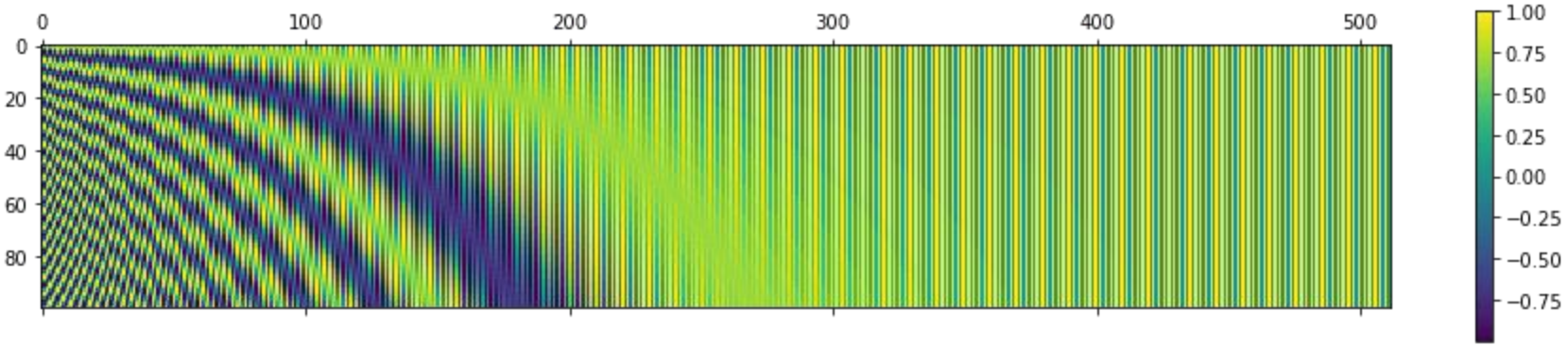
Absolute Positional Encoding - Example



Absolute Positional Encoding - Visualisation

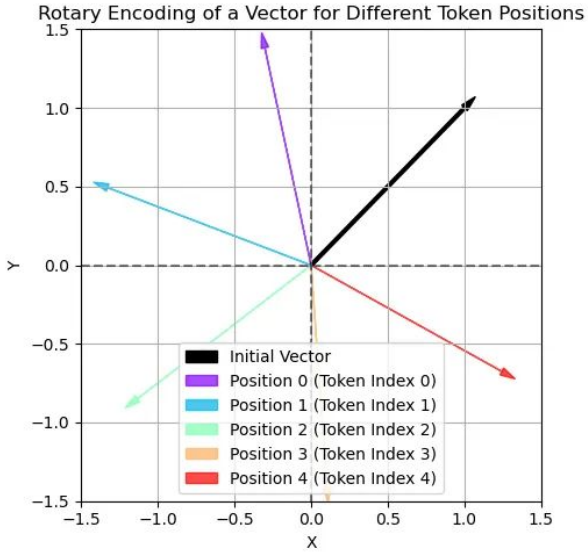


Absolute Positional Encoding - Visualisation



Rotary Position Embedding (RoPE)

Idée : Au lieu de rajouter des dimensions supplémentaires, on fait tourner l'embedding en fonction de sa position dans la phrase.



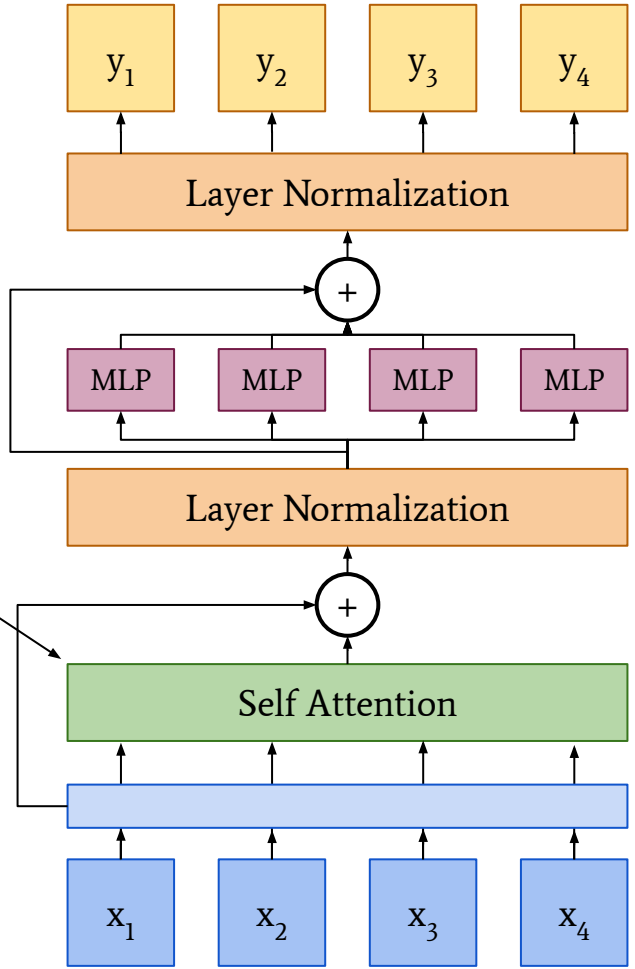
Les blocs Transformers

Un MLP indépendant par vecteur

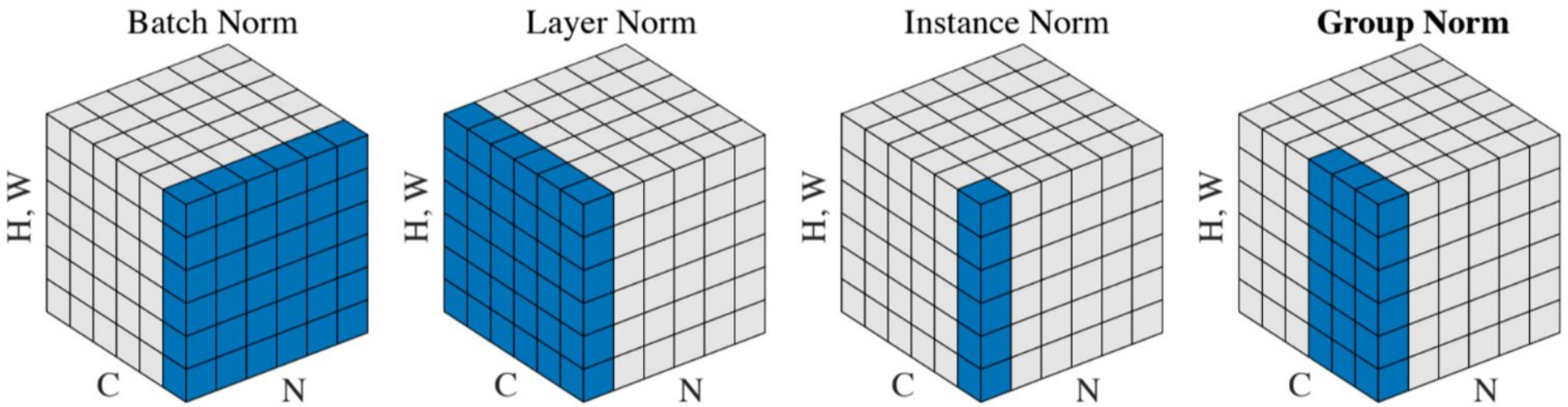
Interactions entre les entrées dans la self attention

Connexion résiduelle

Hautement parallélisable



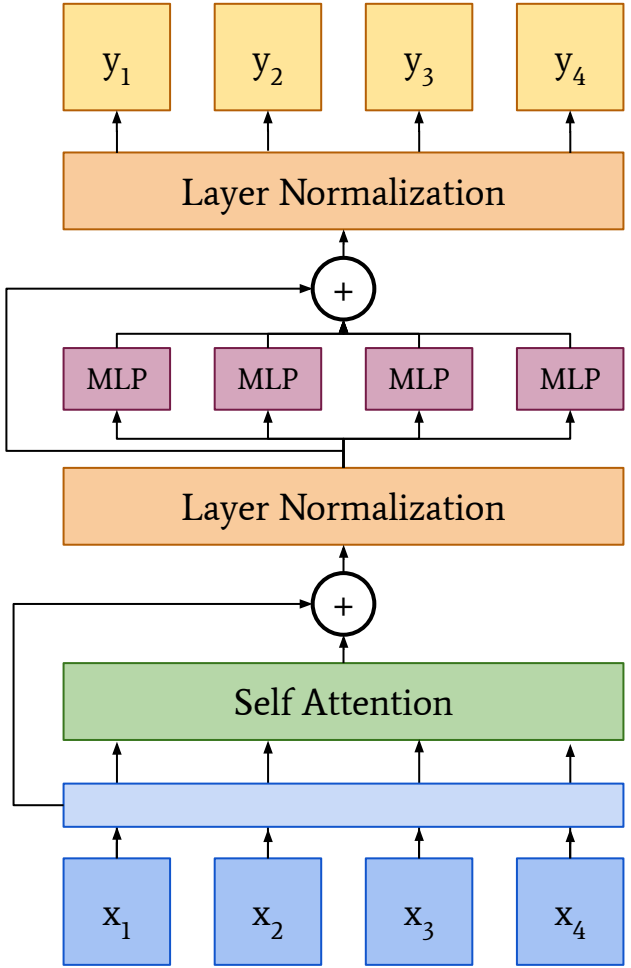
Variantes de la Batch Normalization



Wu and He, “Group Normalization”, ECCV 2018

Les blocs Transformers : Variantes

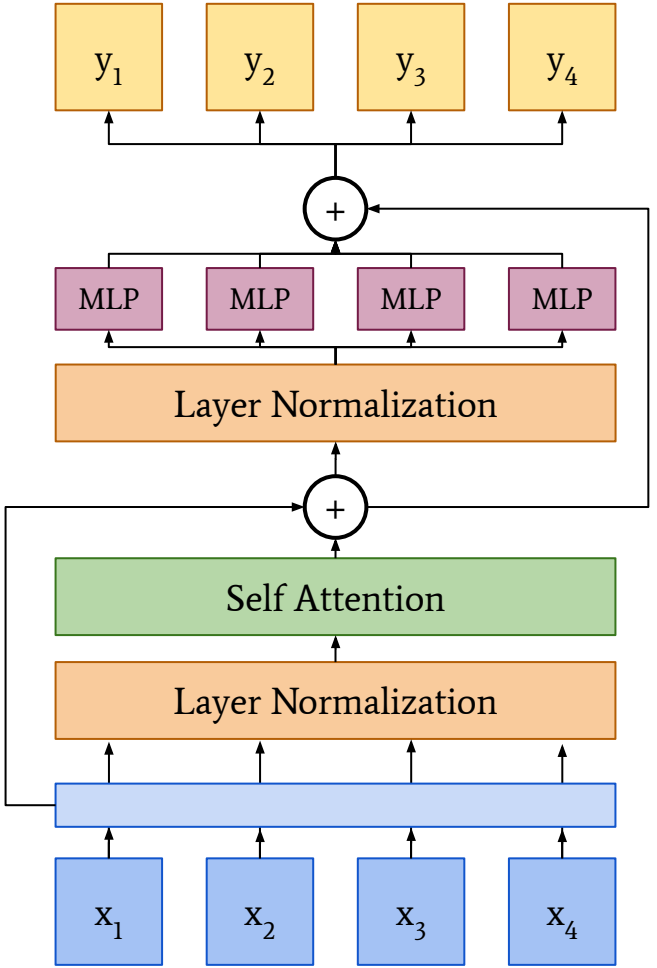
Post-Norm Transformers



Les blocs Transformers : Variantes

Pre-Norm Transformers

Entraînement plus stable, utilisé en pratique

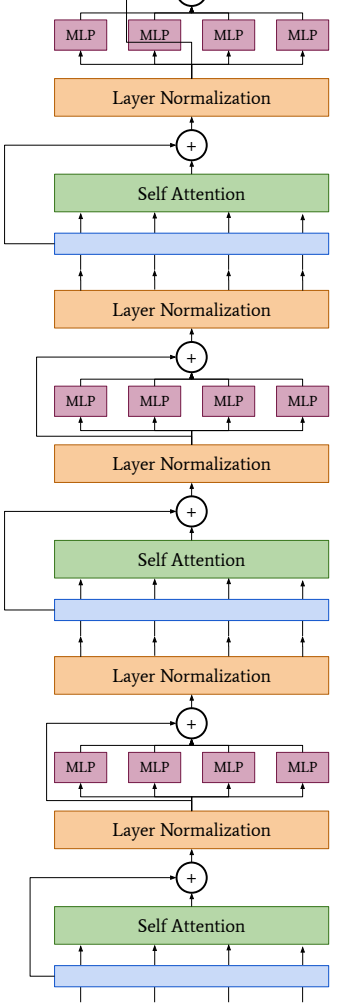


Transformers

Un Transformers est une suite de blocs Transformers

Dans le papier original :

- 12 blocs, MLP dim = 512, 6 heads

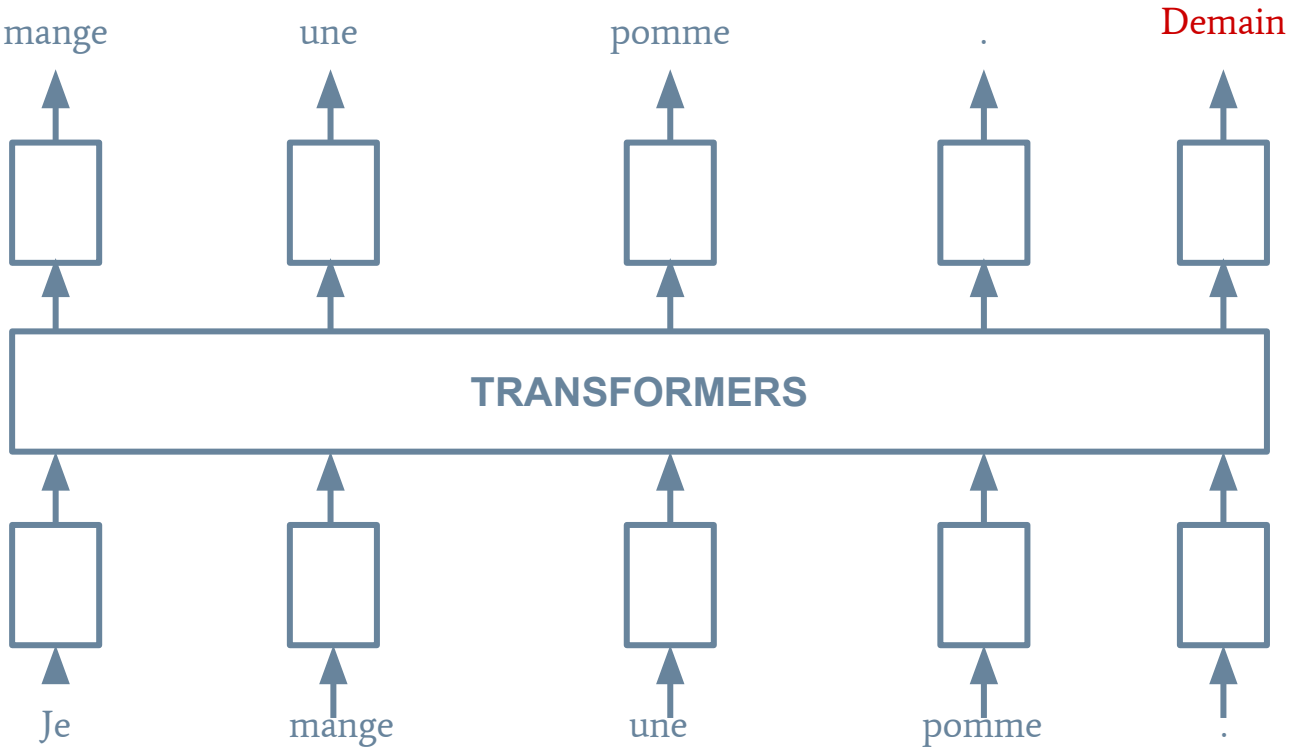


Passage à l'échelle de Transformers

Modèle	Blocs/couches	MLP Dimension	Heads	Paramètres	Données	Entraînement
Transformer-Base	12	512	8	60M		8xP100 (12h)
Transformer-Large	12	1024	16	213M		8xP100 (3.5j)
BERT-Base	12	768	12	110M	13GB	
BERT-Large	24	1024	16	340M	13GB	
XLNet-Large	24	1024	16	340M	126GB	512 TPUv3 (2.5j)
RoBERTa	24	1024	16	355M	160GB	1024xV100 (1j)
GPT-2	48	1600	25	1.5B	40GB	
Megatron-LM	72	3072	32	8.3B	174GB	512xV100 (9j)
Turing-NLG	78	4256	28	17B		256xV100
GPT-3	96	12,288	96	175B	694GB	
Gopher	80	16,384	128	280B	10.55TB	4096 TPUv3 (38j)
GPT4	~120			1800B?		
LLaMa 3	126	16,384	128	405B	~100TB	16k H100 (54j)
DeepSeekV3	61	7168	128	671B	~100TB	2048 H800 (<60j)

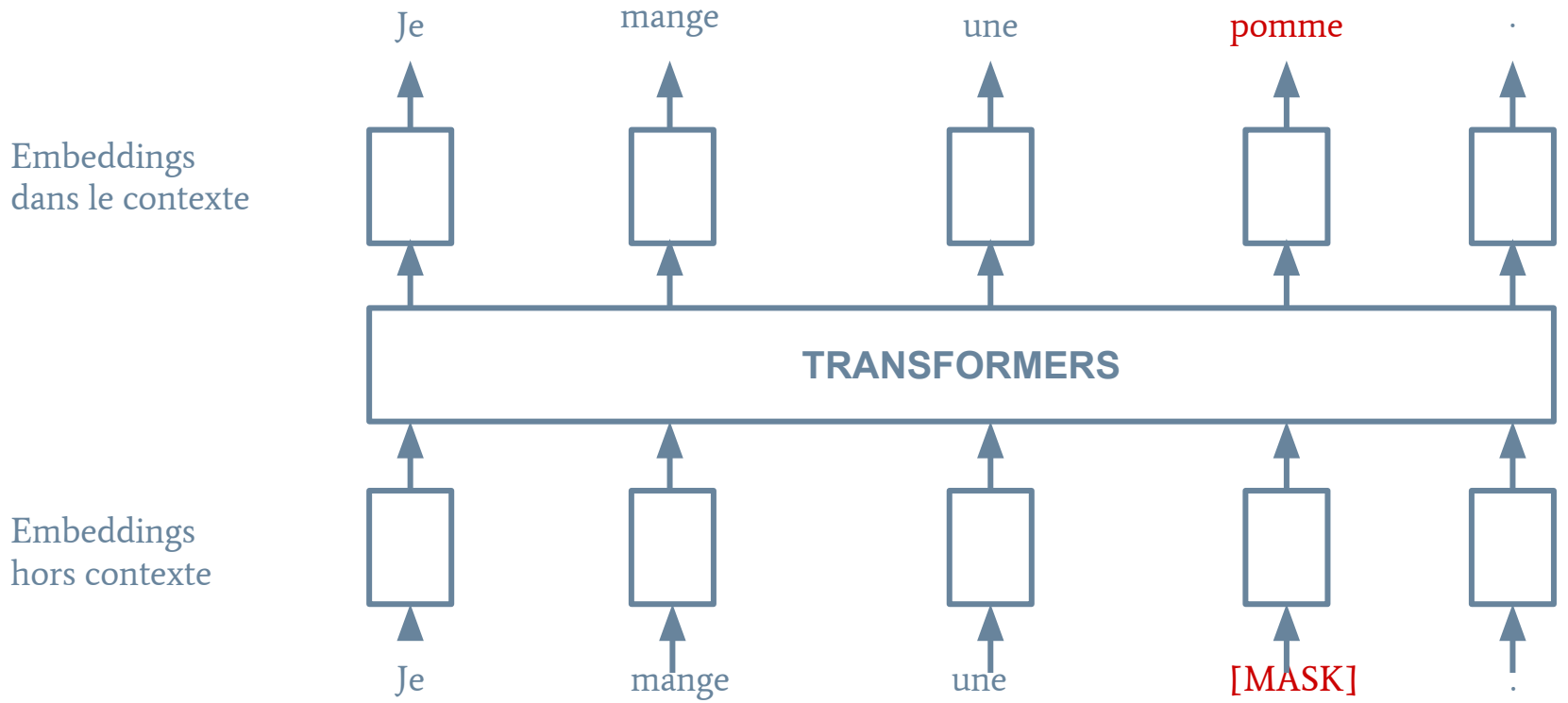
Quatrième solution: Transformers (2017) - GPT

$$P(w_k \mid w_{<k})$$



$$P(w_k \mid w_1, \ldots, w_{k-1}, w_{k+1}, \ldots, w_n)$$

Quatrième solution: Transformers (2017) - BERT



Quatrième solution: Transformers (2017)

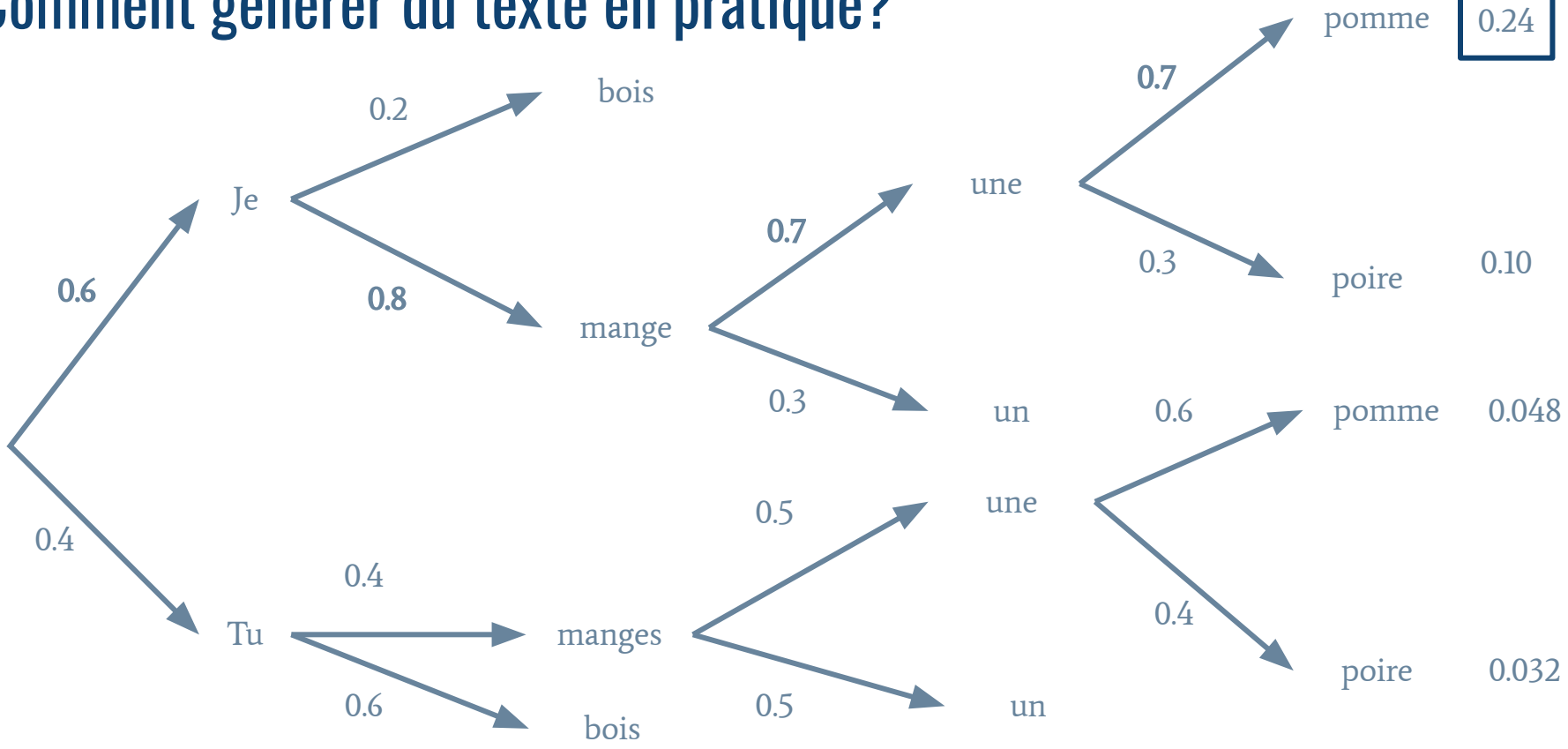
- Transformers est à la base de tous les modèles populaires aujourd'hui
  - GPT(1, 2, 3, 4), ChatGPT, BERT, CamemBERT, RoBERTa, T5, Bloom, LLaMa, DeepSeek
- Les différences entre ces modèles:
  - La tâche d'entraînement (causal/masked/mix)
  - Les données d'entraînement et la manière de les organiser
  - Le post-training
  - Quelques "tricks": normalisations intermédiaires, changement de fonctions d'activation, différents encodages de la position, ...
- Problème: un réseau beaucoup plus gros, très dépendants de la taille de l'entrée
  - RNN: Complexité par couche: $\mathcal{O}(n \cdot d^2)$
  - Self-attention: Complexité par couche: $\mathcal{O}(n^2 \cdot d)$
  - n=taille de l'entrée, d=taille des embeddings
  - Mais la self-attention peut être parallélisée !

Génération de texte

Comment générer du texte en pratique?

- Avec un modèle comme GPT, on obtient $P(w_k \mid w_{<k})$
- Comment trouver le $P(w_1, w_2, \ldots, w_n)$ maximal ?
  - Avec des algorithmes d'exploration

Comment générer du texte en pratique?



Comment générer du texte en pratique?

- Avec un modèle comme GPT, on obtient $P(w_k \mid w_{<k})$
- Comment trouver le $P(w_1, w_2, \ldots, w_n)$ maximal ?
  - Avec des algorithmes d'exploration
- La recherche exhaustive est impossible. On doit utiliser une heuristique.

Génération Greedy

À chaque étape, on prend le mot avec la plus haute probabilité

Avantage : Rapidité

Inconvénients : Non optimal, génère toujours la même chose

Sampling

Pour favoriser la diversité des générations, on prend au hasard un mot en suivant la probabilité donnée par le modèle.

Si P(poule|Le renard mange une) = 0.5, on a une chance sur deux de générer “poule”.

Variante : La température.

- On va modifier la probabilité en faisant p/T (puis on renormalise avec un softmax)
- Plus T est élevé, plus le modèle est “créatif”, voire aléatoire
- Plus T est faible (>0), plus le modèle est conservateur et favorise la probabilité max.

Top-K/Top-P

- L'approche sampling simple peut quand même générer des mots peu probables
  - Si P(poule|Le renard mange une) = 0.5, mais la probabilité des autres mots est <0.01, on a quand même une chance sur deux de générer un mot non probable
- Solution : Limiter le nombre de mots considérés
  - Top-K : On prend les K mots les plus probables
  - Top-P : On prend les N mots les plus probables jusqu'à atteindre une probabilité cumulée de P
- Avantage : Rapide à calculer, sortie variée
- Inconvénient : Pas assez optimal

(En pratique, c'est ça qu'on utilise car le coût de calcul est limitant)

Beam Search

Idée : On garde en permanence les B (beam size) séquences les plus probables (et donc on génère B probabilités différentes à chaque étape

- Avantage : Solution plus optimale que greedy
- Inconvénient : Long à calculer

À l'époque de GPT-2, on utilisait encore beam search, mais devenu trop coûteux maintenant.

Limitations des modèles de langage

- Le contexte est limité et cher à augmenter
  - Nécessaire pour écrire des textes longs et cohérents comme des romans
- On ne peut pas injecter de connaissances dans le modèle
  - On ne peut pas ajouter de manière pérenne des actualités et des faits divers
  - On pourrait énoncer tous les faits au début du texte, mais on est limité par la taille du contexte
- Raisonnements difficiles
  - Très mauvais aux échecs
  - Même des problèmes plus simples sont difficiles pour les modèles de langage
- Les modèles de langage sont biaisés
  - Difficile de les rendre politiquement corrects

En résumé

- Modélisation du langage avec des probabilités
- Word2Vec
- Transformers
- Génération de texte