Introduction au NLP

Charles Béniac

Définition du NLP et Etat de l'art

Qu’est ce que le NLP?

NLP signifie Natural Language Processing, il désigne toute tâche de machine learning qui s’appuie sur une donnée textuelle.

Ces tâches sont nombreuses et diverses:

- Classification et catégorisation de texte (analyse de sentiment)
- Reconnaissance d’entité nommée (personnes, lieux...)
- Analyse sémantique et réponse aux questions
- Détection de paraphrases
- Génération de langue et résumé multi-documents
- Traduction automatique
- Reconnaissance vocale
- Vérification orthographique

Qu’est ce qui rend le NLP spécifique?

La spécificité du NLP ne réside pas tellement dans ses tâches qui se réduisent souvent à des problèmes de classification classiques que nous traiterons avec des modèles connus (de la régression linéaire aux réseaux de neurones).

L’enjeu principal du NLP est de traduire un texte dans un langage compréhensible par l’ordinateur, c’est l’encodage. Nous avons déjà rencontré ce problème avec les variables catégorielles que nous avons encodé.

- Nous avons vu que cet encodage pouvait se contenter de traduire les catégories en chiffres avec le One Hot Encoding mais qu’il pouvait aussi incorporer des relations entre les catégories avec l'Ordinal Encoding.

C’est la même chose pour le NLP, ce n’est pas tout de traduire un texte en chiffres, il faut aussi traduire toutes les relations entre les mots..

Enfin, tout comme les variables catégorielles, l’encodage dépend lui-même de la qualité du pre-processing. Cependant le pré-processing d’un texte est plus complexe.

En résumé: les grandes étapes du NLP

Pré-processing → Encodage - Conversion → Modélisation

Objectif: à partir d'un texte brut obtenir une liste de mots propres

étapes successives:

- **Tokenisation**
- Gestion de la ponctuation, des emojis, et des urls
- Gestion des Stopwords
- Lemmatisation/steaming

Objectif: A partir d'une liste de mots propres, obtenir un vecteur numérique

Choisir une méthode d'encodage:

- **Embedding** (Word2vect, BERT)
- Vectorisation (Bag of Word, Countvectorizer)
- TF-IDF

Objectif: A partir d'un vecteur numérique, obtenir une prédiction

Le modèle dépend du problèmes:

- supervisé (détection de spam)
- non supervisé (clustering automatique de mail)

Il est important de comprendre que ces trois étapes sont toutes intrinsèquement reliées. En fonction du modèle choisi on va choisir un certain encodage et donc un certain preprocessing.

Etat de l'art du NLP

Il y a encore quelques années, le NLP était un sujet très foisonnant techniquement, il fallait composer avec une multitude de librairies concurrentes et complémentaires pour composer une stack technique viable:

Cependant ce foisonnement à laisser la place ces dernières années a une sorte de consensus:

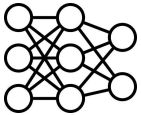
"Attention is all you need"

Le NLP aujourd'hui

Le consensus a rendu le NLP techniquement plus simple, du moins plus clair:

1. Je vais chercher dans **Hugging Face** un modèle pré-entraîné pour mon problème.

2. Je l'importe dans ma lirairie de deep-learning préférée **Pytorch** ou **TensorFlow**



3. J'ajoute si besoin des couches spécifiques aux problèmes que je veux résoudre.

La linéarité technique est associée paradoxalement à une grande complexité théorique. C'est donc ce point qui va particulièrement nous occuper. Une manière d'approcher cette théorie est d'en retracer l' évolution historique.

Histoire théorique du NLP

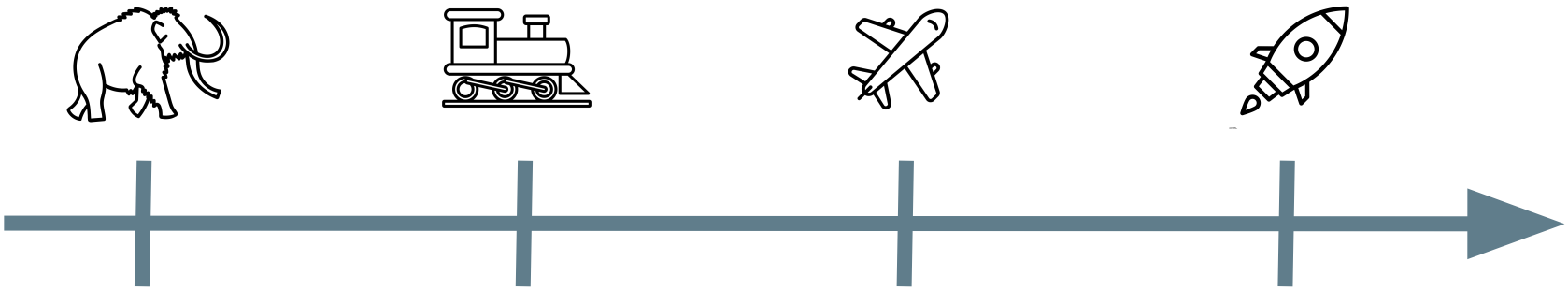
Pourquoi une histoire théorique du NLP?

L'histoire théorique du NLP c'est l'histoire des réponses successives apportées à une même et seule question:

"Quelle est la meilleure façon d'encoder un texte pour saisir l'ensemble des relations qui le composent et celles qui le lient aux autres textes?"

C'est une question à la fois **technique** et **linguistique**. Les réponses apportées ont suivi des pistes différentes, souvent en s'inspirant et en corrigeant des pistes passées, c'est pour cela qu'il est important de comprendre chaque étape et pourquoi et comment ces étapes ont été dépassées.

Histoire rapide du NLP



Bag of Words
TF-IDF

Embedding
Word2Vec

Recurrent
Neural
Networks

Transformers

Le vocabulary Space

L'objectif est donc d'encoder **un corpus** (texte d'une longueur allant du simple mot à l'encyclopédie) au format numérique. L'approche la plus naïve est d'utiliser quelque chose qui ressemble à ce que l'on connaît, le One Hot Encoding: chaque mot de notre vocabulaire correspond à une colonne.

On suppose le corpus suivant: ***"Le chemin longe le chemin de fer. Je chemine dessus. "***

On a donc comme **Vocabulary Space**:

Le	chemin	longe	le	de	fer	.	Je	chemine	dessus

Pour chaque chaîne de caractère différente, on a créé une colonne (une dimension de mon vecteur qui appartient à l'espace de vocabulaire).

Le token

Le	chemin	longe	le	de	fer	.	Je	chemine	dessus

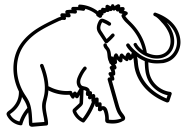
La division obtenue s'appuie sur les chaînes de caractères, elle peut sembler étrange et on aurait pu faire d'autres choix:

- Ignorer la ponctuation et les lettres majuscules
- Considérer que le mot 'le' est très générique et qu'il n'apporte aucune information. (Gestion des Stopwords)
- Considérer que le mot 'chemin' et le verbe 'cheminer' sont proches donc associable (Lemmatisation/steaming)
- Considérer que "chemin de fer" est un mot unique et donc différent de "chemin"

Définir ce qui constitue une unité textuelle est une étape clé qu'on appelle la **tokenisation**.

Ce choix se fait en fonction d'un arbitrage "dimension du Vocabulary Space / richesse des informations enregistrée". Cet arbitrage va dépendre du modèle visé.

Le bag of words (count-vectorizer)

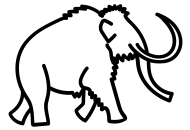


L'idée du bag of words ou du count vectorizer est très simple. **On représente chaque phrase d'un corpus par un vecteur** dans l'espace du Vocabulary Size en comptant le nombre d'occurrence de chaque mot.

	le	chemin	longe	chemin de fer	je	chemine	dessus
"chemin de fer"	0	0	0	1	0	0	0
"Le chemin longe le chemin de fer."	2	1	1	1	0	0	0
"Je chemine dessus."	0	0	0	0	1	1	1

On obtient donc un dataframe qu'on peut utiliser pour un problème supervisé (classification de sentiment des phrases). On peut même utiliser un modèle de ML classique à condition que la dimension du Vocabulary Space ne soit pas trop élevée

Le bag of words (count-vectorizer)

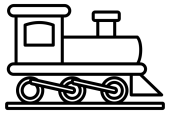


C’est le moment d’introduire notre grille d’évaluation pour les modèles de NLP:

Dimensionnalité	Grande dimension: Vocabulary Spce	
Pré-processing	Très important pour réduire le vocabulary space	
Rapidité calcul	Très Rapide	
Richesse syntaxique	Nulle, l’ordre des mots est perdu dans le process	
Contextualisation	Correcte, on traite une phrase en entier	

On peut améliorer le type de comptage avec la méthode du TF-IDF, mais ça ne change pas les problèmes intrinsèques de cette approche. Il faut donc pour s’améliorer explorer une nouvelle piste

Le word embedding



Contrairement au Bag of words, le word embedding ne cherche pas à représenter des phrases par un vecteur mais les mots eux-mêmes.

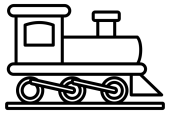
Pour le moment la seule représentation de mot que nous avons ce sont les One-Hot vecteur qui sont à la fois très long et très peu informatif. Le but du word embedding serait d'avoir de vecteur qui incorpore le sens des mots un peu comme cela:

	Roi	Reine	Lion	Lionne	Capucine	Lys
Femme	-0,25	0,75	-0,15	0,80	0,13	0,05
Royal	0,90	0,90	0,50	0,45	0,02	0,51
Fleur	0,03	0,12	0,01	0,01	0,75	0,80
Orange	0,03	0,02	0,34	0,40	0,85	0,10
…						

Chaque mot est représenté par un vecteur de dimension 4, ces vecteurs ont un sens. ils forment ensemble la **matrice d'embedding.**

La grande question est de savoir comment obtenir un telle matrice

Le word embedding



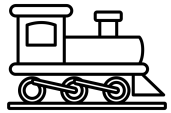
On va réussir à déterminer mathématiquement les paramètres de la **matrice d'embedding**, en en faisant **la matrice des poids d'une couche d'un réseau de neurones**. La connaissance d'un réseau de neurones réside en effet dans ces poids.

Ce qu'il faut donc c'est trouver une tâche dont la résolution nécessite la compréhension du sens des mots. On va ici s'appuyer sur une hypothèse linguistique: **l'hypothèse distributionnelle** (Harris, Firth) qui peut s'énoncer ainsi:

"On ne peut connaître un mot que par ses fréquentations"

Cela veut dire que deux mots qu'on retrouve souvent dans un même contexte doivent avoir un sens proche et donc que leurs vecteurs de représentation doivent être proches. (On calcule la proximité entre deux vecteurs par formule mathématique, la **Cosine similarity**)

Le word embedding: Word2Vec



De nombreuses tâches ont été proposées pour adapter cette hypothèse à un réseau de neurones. L'une d'elle est le **skip gram** proposée avec le modèle **Word2Vec** par les équipes de google 2013. On donne un mot en entrée et il faut réussir à deviner un mot de son contexte.

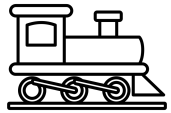
Pour le corpus : ***"La fleur de lys est l'emblème des rois et reines de France" .*** Si on prend une **fenêtre** de taille 1 (le contexte est constitué du mot de devant et de derrière). On obtient la tâche supervisée suivante:

Context (vecteur X)	Target (vecteur Y)
lys	emblème
roi	emblème
roi	reines
france	reines

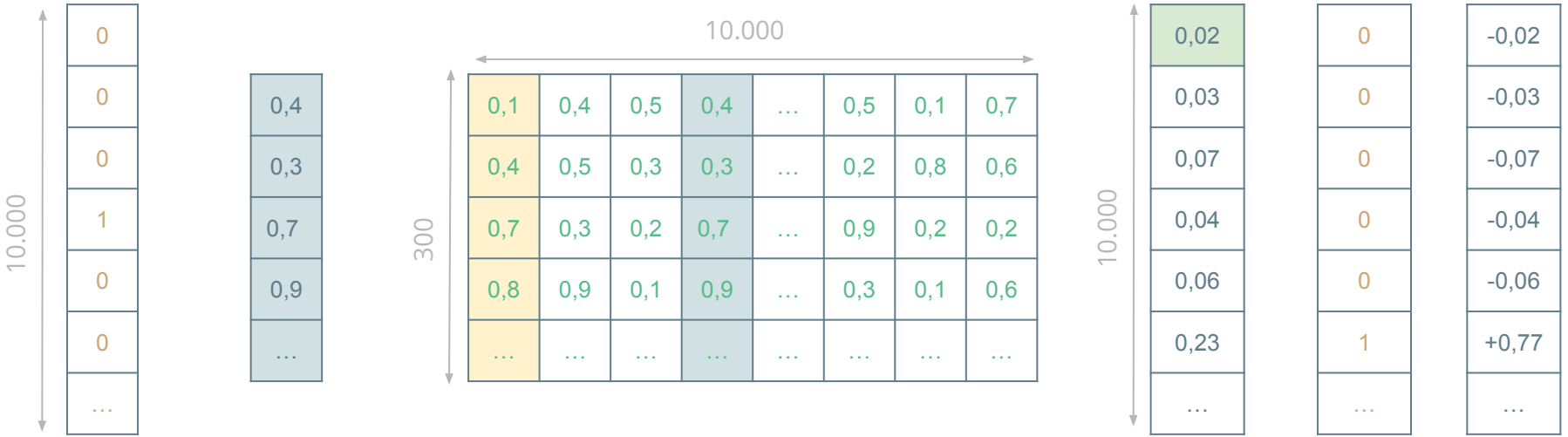
Remarques:

- On note que la tâche est étrange: un même input correspond à deux output différents. Le but n'est pas de réussir parfaitement cette tâche mais que les paramètres qui essayent d'y répondre soient optimisés.
- La fenêtre est souvent plus large (entre 2 et 5)
- C'est une **tâche auto-supervisée**, ce qui est super.

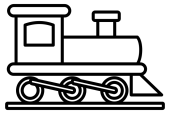
Le word embedding: Word2Vec



On suppose un vocabulaire de 10.000 mots. On veut obtenir pour chaque mot un vecteur de représentation de taille 300. A l'aide du contexte "roi" (3243), je veux prédire la target "reine" (5678)



Le word embedding



Le principe du word embedding est d'utiliser les vecteurs créés à la place des vecteurs One-Hot pour travailler en dimension plus petite. On peut créer un vecteur phrase en faisant la moyenne de ses vecteurs mots.

Dimensionnalité	Plus faible, de la taille du vecteur embedding	
Pré-processing	Défini par la matrice d'embedding	
Rapidité calcul	Long à entraîner mais rapide à utiliser	
Richesse syntaxique	Pas d'ordre des mots, problèmes avec les homonymes	
Contextualisation	Faible, de la taille de la fenêtre (réduit avec Glove)	

Il est possible de trouver de meilleures tâches et de meilleures méthodes pour passer du vecteur mot au vecteur phrase, mais les faiblesses de 'embedding persistent. Cependant c'est un bon point de départ.

Recurrent Neural Network

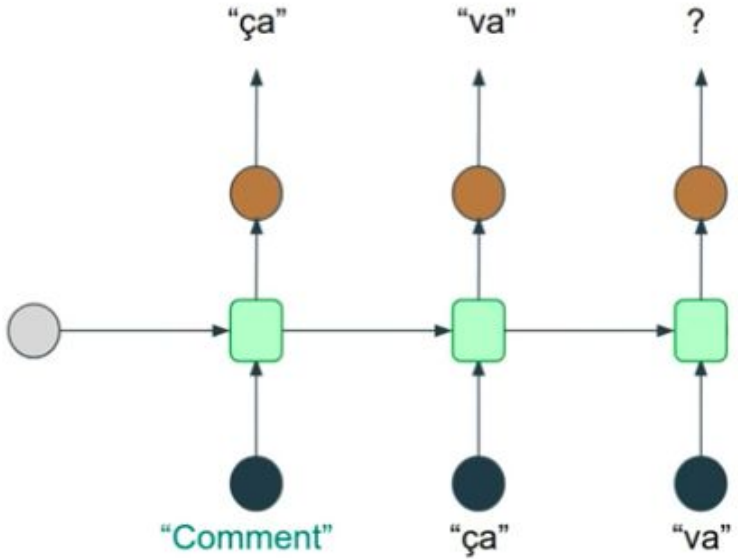
Un réseau de neurones récurrent est un réseau de neurones qui prend deux types d'entrées:

- un mot X
- les valeurs d'activation de ce même réseau à l'état précédent

Dans le cas simple il prédit un vecteur mot Y en sortie.

L'idée est d'adopter une démarche séquentielle qui permet d'approcher des notions syntaxiques.

On parle de **cellule** pour décrire le réseau (en vert) qui est utilisé à chaque fois. Cette cellule est déterminé par ce qu'elle prend en entrée et ce qu'elle renvoie en sortie mais a l'intérieur elle peut être plus ou moins complexe (en nombre de neurones, de couches ou même d'architecture)



Recurrent Neural Network

On reprend l’évaluation pour comprendre ce qu’on a gagné et perdu:

Dimensionnalité	Globalement la même que pour l’embedding	
Pré-processing	Défini par la matrice d’embedding également	
Rapidité calcul	Très long à entraîner car récurrent.	
Richesse syntaxique	Notion de syntaxe mais dans un seul sens	
Contextualisation	Faible, à cause de la disparition du gradient	

On a donc gagné sur la compréhension syntaxique mais c’est payé par un coût important en termes de temps de calcul. On peut prolonger la “mémoire” de ces réseaux à l’aide de neurones LSTM (Long Short Term Memory) mais cette mémoire reste restreinte et les calculs restent longs

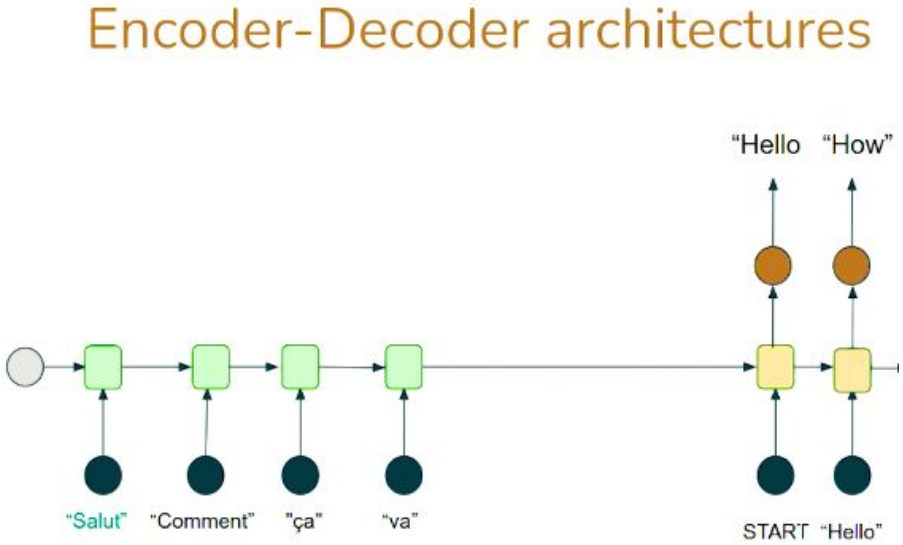
L’encoder-decoder

Avant de parler d’attention, il faut s'arrêter sur l’architecture encoder-decoder

L’encoder: C’est une cellule qui prend en entrée un mot et un état précédent et qui renvoie des valeurs d'activation. (en vert)

Le décodeur: Il prend entrée soit l’activation de l’encodeur, soit l’état précédent du décodeur. Il prend également un mot qui correspond à la bonne réponse et non à la prédiction précédente.

L’architecture encodeur/ décodeur permet de distinguer le réseau de neurone qui encode de celui qui prédit. Cependant le problème du vanishing gradient n’est pas résolu.



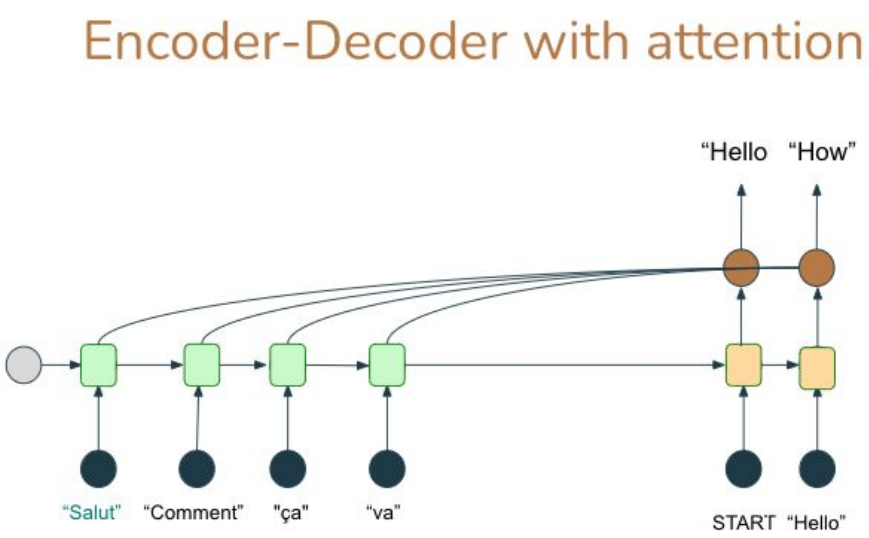
Le principe de l'attention

Le but de l'attention est d'éviter le phénomène du gradient vanishing.

Pour y parvenir, on va faire en sorte avant de faire la prédiction, de regarder à nouveau toutes les informations que l'on avait dans l'encodeur.

Ainsi, il aura un lien direct entre chaque étape de l'encodage et la prédiction finale (donc un seul gradient).

Si le principe a l'air clair, il faut comprendre comment on fait pour redonner toutes cette information avant la prédiction.



Le principe de l'attention

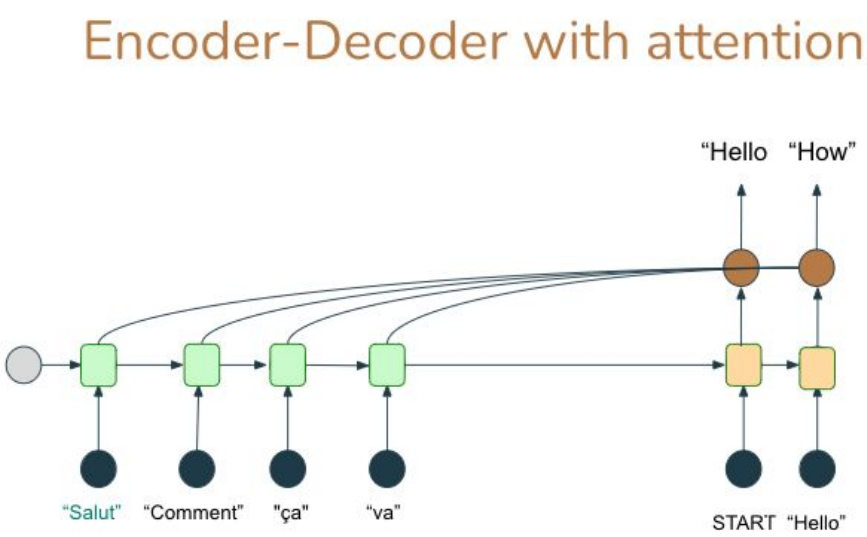
Le mécanisme de l'attention tourne autour de trois notions clés: La Query , la Key, et les Valeurs

La Query: correspond à la prédiction du décodeur. Elle s'appelle Query car c'est cette valeur qui va nous permettre d'interroger le décodeur pour savoir quelles valeurs sont pertinentes à retenir. dimension: nb_prediction, taille du vocabulaire

La Key: correspond aux 4 valeurs de l'encodeur. dimension: nb_input, taille du vocabulaire

En multipliant (matriciellement) la query par la Key et en réalisant un SoftMax on obtient une clé de répartition: quel pourcentage d'information que je récupère de chacune des cellules.

Je multiplie (produit scalaire) cette clé de répartition par les **valeurs** de l'encodeur. dimension: nb_input, taille du vocabulaire



Le transformer

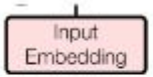
Voici l'architecture d'un transformateur.

Il est fondamental de comprendre que **les tokens ne sont pas transmis de manière séquentielle mais tous en même temps** à l'aide du mécanisme de multi-head self-attention.

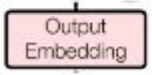
- **self attention**: la query et la key viennent du même input
- **multi-head:** on utilise plusieurs mécanisme d'attention en même temps pour capter différents phénomènes

L'out est encodé avec un mask multi-head-self-attention:

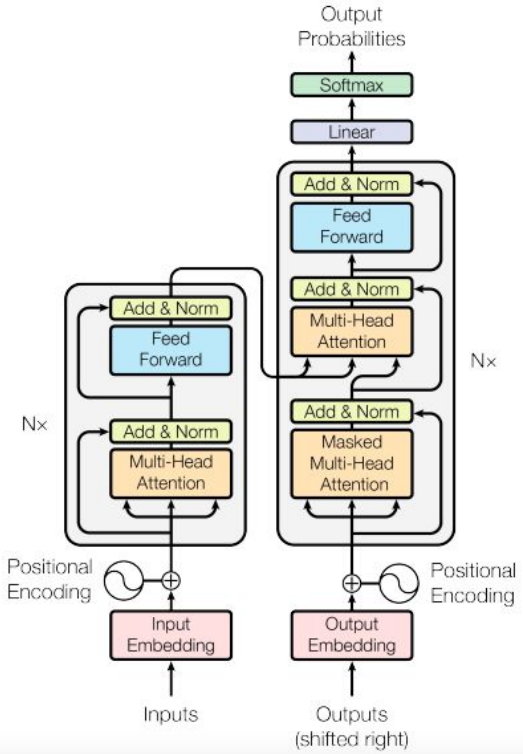
- **mask attention:** on a accès qu'aux tokens qui précèdent



["Salut", "comment", "ca", "va"]



[START, "Hello", "how", "are", "you"]

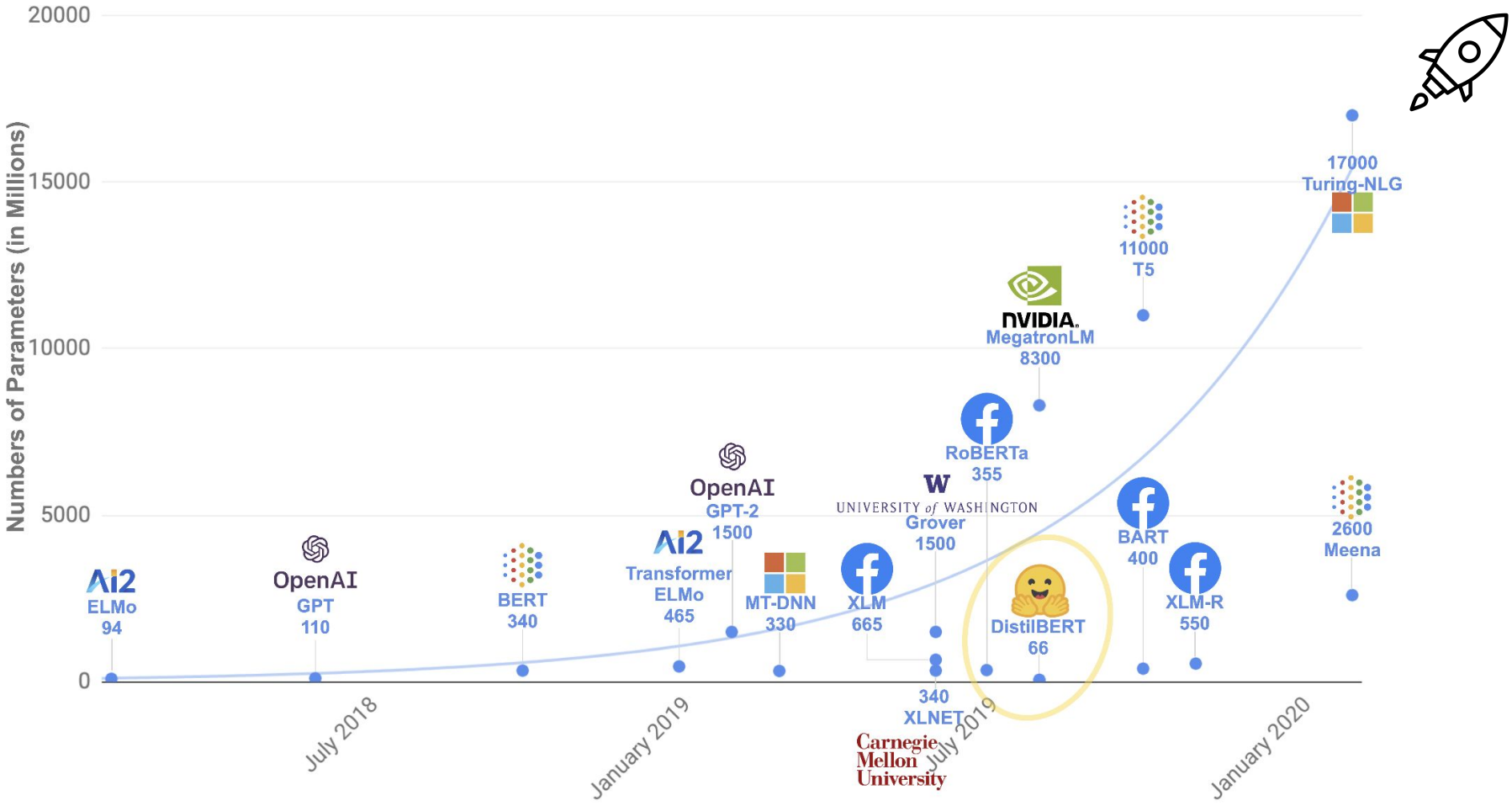


Le transformer

On reprend l'évaluation pour comprendre ce qu'on a gagné:

Dimensionnalité	Globalement la même que pour l'embedding	
Pré-processing	Défini par la matrice d'embedding également	
Rapidité calcul	Relativement rapide à entraîner car non séquentiel	
Richesse syntaxique	Notion de syntaxe dans les 2 sens (non séquentiel)	
Contextualisation	Bonne car plus de vanishing gradient	

On voit que l'architecture transformer ressemble aujourd'hui à l'architecture idéale. C'est pour cela que depuis 5-6 ans on est plus à une course au nombre de paramètres et à la largeur du corpus que dans la recherche de nouvelles architectures.


Numbers of Parameters (in Millions)
20000
15000
10000
5000
0
July 2018
January 2019
July 2019
January 2020
Ai2
ELMo
94
OpenAI
GPT
110
BERT
340
Ai2
Transformer
ELMo
465
OpenAI
GPT-2
1500
MT-DNN
330
XLM
665
340
XLNET
Carnegie Mellon University
W
UNIVERSITY of WASHINGTON
Grover
1500
RoBERTa
355
NVIDIA.
MegatronLM
8300
DistilBERT
66
11000
T5
BART
400
XLM-R
550
17000
Turing-NLG
2600
Meena