Chapitre 3 : NLP, transformers et LLM

Jordy Palafox

October 3, 2024

Intelligence Artificielle pour la Cybersécurité
CY Tech - Ing 3 CS - 2024/2025

Généralité sur le TAL

TAL : Traitement Automatique du Langage ou
NLP : Natural Language Processsing

Objectif : analyser du texte (écrit) par un ordinateur

- Fin des années 50 avec les travaux de Chomsky,
- Premier objectif : la traduction automatique
- Année 60 : quelques avancées et succès,
- Année 80 : IA formelle (symbolique) et systèmes experts, développements importants,
- Premières limites : la passage à l'échelle;
- Années 90 à 2010 : Modèles statistiques,
- Depuis 2010 : Approche connexionniste (NN)

Quelles tâches peut réaliser le NLP ?

1. Annotation
   - Etiquetage,
   - Reconnaissance d'entités nommées
2. Analyse syntaxique
3. Classification de textes
   - Anti-spam
   - Analyse de sentiments
4. Réécriture
   - Traduction automatique
   - Correction automatique
   - Résumé
5. Agent conversationnel
6. OCR
7. Reconnaissance de la parole.

Difficultés du NLP

- La langue est vivante et ambiguë,
- Homonymes (père, paire) : même prononciation mais sens différents, homographes (son, son), homophones (cette et sept),
- L'échantillonage : mots inconnus, nouveaux, noms propres,
- Ambiguïté de la grammaire (plus arbres pour une même phrase au sens théorie du langage),
- Enrichir le vocabulaire crée de l'ambiguité "la" l'article et "la" la note de musique.

Limites de la théorie du langage et probabilités

- Gestion difficile des irrégularités de la langue,
- Problèmes sémantiques,
- Possibilité de probabiliser certaines approches.

Un exemple d'approche probabiliste

Les réseaux de markov cachés !
Une phrase constitue une séquence d'évènement, on veut estimer :

$$\mathbb{P}(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$$

Pour une séquence (A, B, C, D), on obtient :

$$\mathbb{P}(A, B, C, D) = \mathbb{P}(A|B, C, D)\mathbb{P}(B|C, D)\mathbb{P}(C|D)\mathbb{P}(D)$$

Si X est une chaîne de Markov alors $\mathbb{P}(X_n|X_{n-1} \ldots X_1) = \mathbb{P}(X_n|X_{n-1})$.
Exemple d'algorithme : Viterbi.

Mais comment représenter un mot pour l'ordinateur ?

Première question : Un mot c'est quoi ?

- une entrée dans un dictionnaire ? (et un mot inconnu dans ce cas ?)
- Une suite de lettres ?

On peut aussi se poser la question du sens des mots !

Dictionnaires

WordNet est un graphe de mots disponible et très souvent utilisé pour les approches connexionnistes,
$\Rightarrow$ `https://www.nltk.org/howto/wordnet.html`
Il existe d'autres dictionnaires, voir `https://spacy.io/`

Approche naîve : le one-hot encoding

On prend la liste des mots disponibles puis on les numérote par rapport à leur position dans la liste :

$$mot_i \longrightarrow \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \; ieme\ position$$

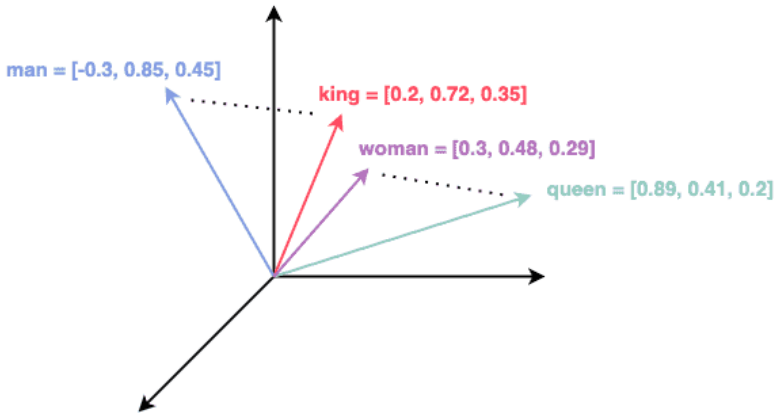
Factuellement cela marche **mais** :

- l'ordre des mots n'a pas forcément de logique,
- la distance entre deux mots est toujours la même,
- représentation sparse (beaucoup de 0!).

On peut aussi introduire des vecteurs pour des mots inconnus ($< UNKNOWN >$)et du padding ($< PADDING >$). On peut aussi remplacer un mot peu fréquent par $< UNKNOWN >$.

Word Embedding

- Représenter un mot sous forme d'un vecteur "plein" dans un espace numérique ($\mathbb{R}^n$ avec n entre 50 et 300 le plus souvent)
- Deux mots sémantiquement proches auront des vecteurs proches (dans un sens métrique, norme)
- Premiers travaux dans cette direction : Latent Semantic Analysis



1

Vocabulaire: on parle d'embedding, plongement ou représentation vectorielle de mots.

[1] `https://www.baeldung.com/cs/dimensionality-word-embeddings`

Word2Vec

Méthode d'apprentissage non supervisé d'embeddings de mots qui consiste en l'une des deux taches :

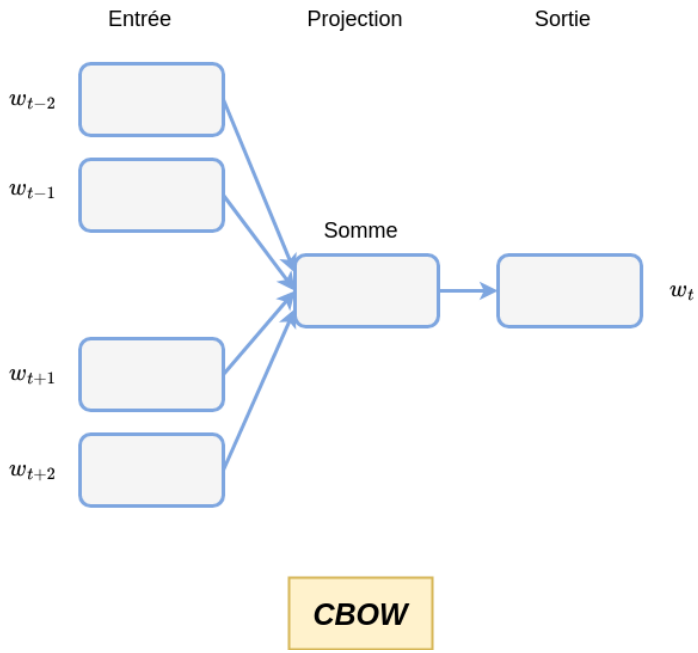
1. Prédire un mot connaissant ses mots environnants → **Continuous Bag Of Words** (ou CBOW),
2. Prédire un contexte à partir d'un mot → **Skip-gram**.

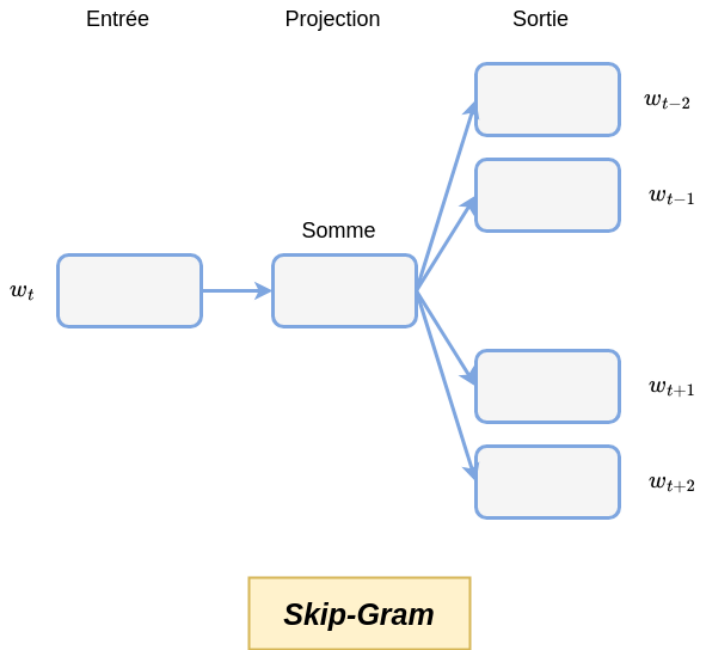
Chacun ses avantages :

1. CBOW pour les mots fréquents,
2. Skip-gram pour les mots rares.

Il suffit d'un corpus pour réaliser l'entraînement.

CBOW vs Skip-gram





Distance et géométrie

Distance cosinus

Soient mot_1 et mot_2 représentés resp. par les vecteurs v_1 et v_2 alors :

$$d_{cosinus}(mot_1,\ mot_2) = \frac{< v_1,\ v_2 >}{\|v_1\| \times \|v_2\|} = cos(\widehat{v_1 v_2})$$

- Plus l'angle est proche de 1 , plus v_1 et v_2 proches,
- Fonctionne mieux en grande dimension que la norme euclidienne,
- $d_{cosinus}$ petite $\Leftrightarrow$ sens sémantique proche.

Glove

Global Vectors for Word Representation

Respecte les propriétés géométriques de sommes et différences de vecteurs.
L'idée : sens des mots = rapports entre des probabilités conditionnelles

On va définir v_1 en essayant de trouver des relations simples entre $v_1 - v_2$, v_2 et $\frac{\mathbb{P}(mot_k|mot_1)}{\mathbb{P}(mot_k|mot_2)}$, la solution est obtenu par minimisation d'une fonction de coût et est **globale**. Meilleures informations sémantiques.

FastText

Proche de Word2Vec avec pour différences :

- La gestion des mots hors vocabulaires
  - Word2vec : ne sait pas représenter un mot hors vocabulaires,
  - FastText : décomposition en n-grams
- Représentation des mots
  - Word2Vec : plongement baser sur les mots, pas de structure interne
  - FastText : capture des informations de sous-mots (n-grams)
- Entraînement
  - Word2Vec : approche sur les mots plus lents
  - FastText : efficacité des n-grams.
- Cas d'usage
  - Word2vec : recherche de mots similaires, compréhension de relations entre des mots, capture de similitude sémantique
  - FastText : analyse de sentiments, identification de langue, tache où la morphologie est importante.

Séquences et séries temporelles

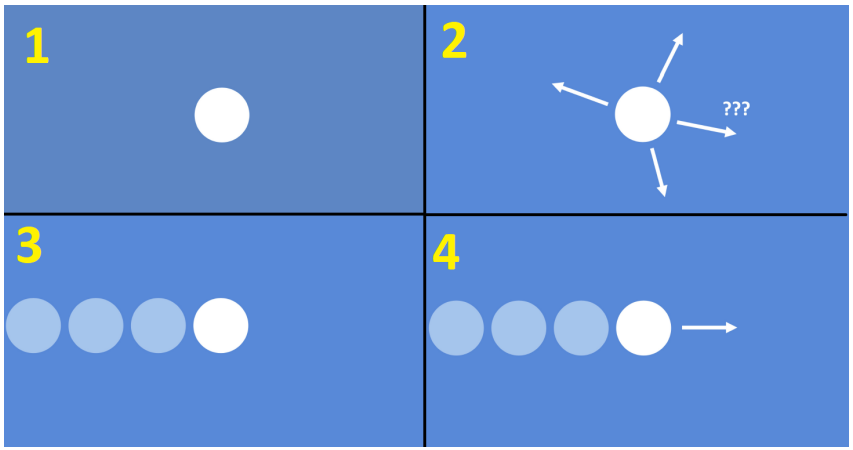
Définition

Une séquence est une suite d'observation $\{x_t\}_{t=1,\dots,T}$ avec $x_i \in \mathbb{R}^d$ avec généralement une notion de causalité x_{t_2} dépend de x_{t_1} pour $t_1 \leq t_2$.

Si $d = 1$, séquence univariée. Si $d \geq 2$, séquence multivariée.

Un texte est une séquence !
Les séries temporelles aussi, par exemple des flux financiers.

Les variables explicatives d'une séquence résident dans la taille de l'historique ! (que l'on prendra de longueur fixe). Permettant d'introduire une notion de mémoire.

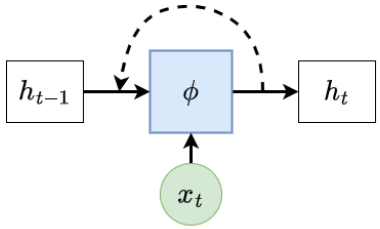


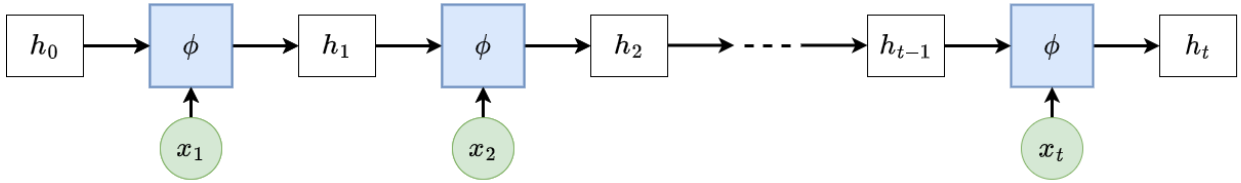
Définition

Soit $\{x_t\}$ une séquence d'entrée univariée ou multivariée.
Un **neurone récurrent** (ou cellule neuronale récurrente) est la donnée d'une séquence d'états internes $\{h_t\}_{t=1,\dots,T}$ tel que $h_i \in \mathbb{R}^l$ définie par :

$$h_t = \phi(x_t, h_{t-1})$$

L'état interne h_t dépend de la valeur de la séquence à ce pas t et de l'état interne au pas précédent h_{t-1}.





Formulation mathématique

L'état interne h_t est défini par :

$$h_t = \phi(\underbrace{U}_{\in \mathbb{R}^{l \times d}} x_t + \underbrace{W}_{\in \mathbb{R}^{l \times l}} h_{t-1} + \underbrace{b_h}_{\in \mathbb{R}^{l}}).$$

où ϕ fonction d'activation non linéaire (tanh ou sigmoide le plus souvent). Les matrices U et W sont partagées à tous les pas de temps.

Expression d'une sortie

Notons y_t la sortie. Pour l'obtenir il suffit d'appliquer une couche FC sur l'état interne de la cellule récurrente :

$$y_t = \psi(Vh_t + b_y)$$

où ψ est l'identité ou une activation non linéaire (softmax par exemple).

Optimisation d'un RNN

Les matrices U, V, W et le vecteur de biais ne dépendent pas du temps donc le nombre de paramètres de la cellule RNN s'exprime en fonction :

- la dimension des entrées
- la taille de l'état interne,
- la dimension des sorties.

On va chercher à minimiser une fonction de coût $\mathcal{L}$ telle que :

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t(y_t, y_t^*; \theta)$$

où y_t est la sortie du réseau, y_t^* la vraie label, θ les paramètres du réseau.

Optimisation d'un RNN

Comme les paramètres du réseau sont partagés par tous les états, on va explorer ce qu'il se passe en appliquant la descente de gradient ! On va prendre W mais c'est pareil pour U et V. En particulier on a :

$$\frac{\partial \mathcal{L}_t}{\partial W} = \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial W}.$$

Ici c'est le terme $\frac{\partial h_t}{\partial W}$ qui est le plus difficile à calculer car h_t dépend de W et h_{t-1} qui dépend aussi de W d'où :

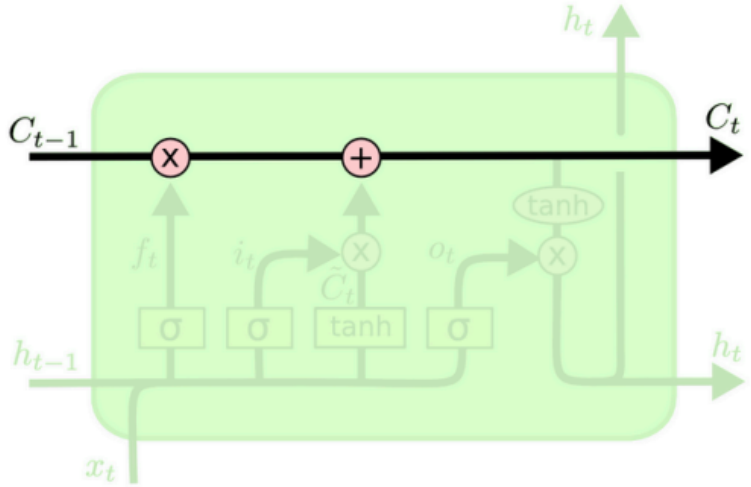
$$\frac{\partial h_t}{\partial W} = \frac{\partial \phi(x_t, h_{t-1}, W)}{\partial W} + \frac{\partial \phi(x_t, h_{t-1}, W)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W}$$

On peut continuer à détailler mais les formules sont lourdes, en pratique elles sont tronquées sur les pas de temps !

La forme explicite de la descente de gradient permet de voir que les RNN sont sensibles à l'explosion ou l'annulation du gradient ! Les LSTM sont censés résoudre ce problème.

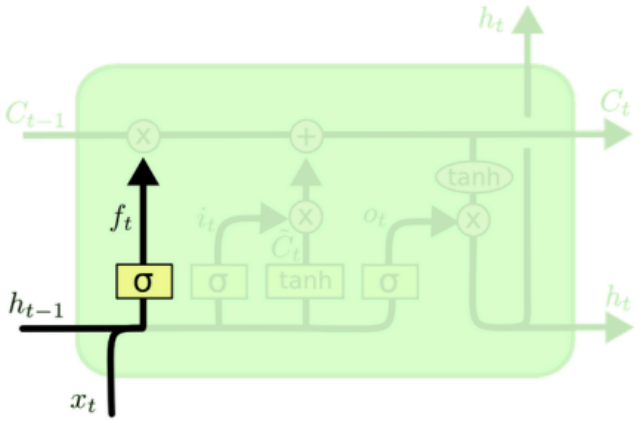
LSTM

- Modification principale : ajout d'un état de la cellule $\mathbf{C_t}$ passé directement au t suivant
- Simple à implémenter, permet à l'information (gradient) d'avoir un flux non-interrompu selon le chemin $\mathbf{C_t}$
- La cellule LSTM ajoute ou retire de l'information de son état interne $\mathbf{C_t}$



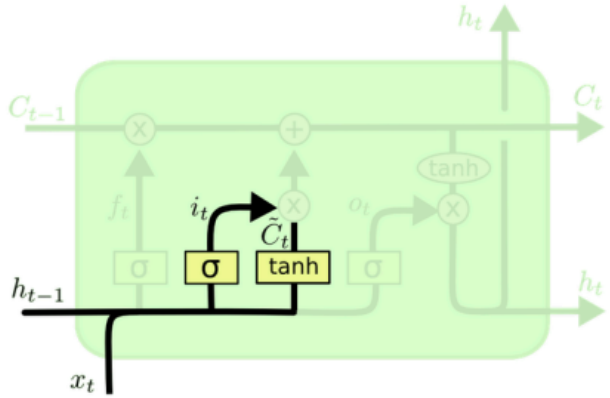
LSTM

- **Porte d'oubli** (*forget gate*) $\mathbf{f_t}$: contrôle la conservation ou l'effacement de la cellule



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

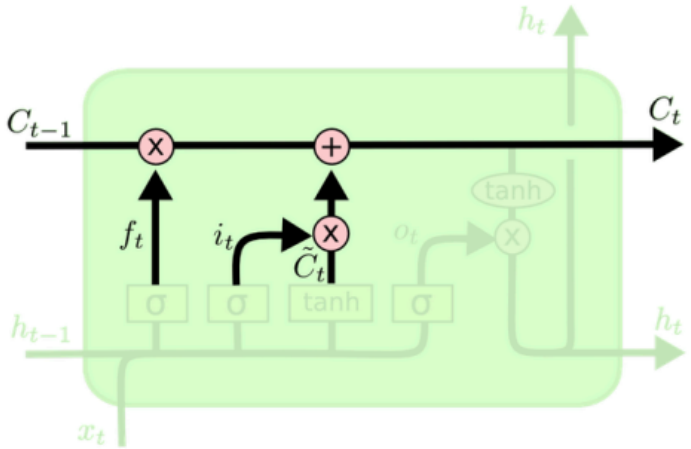
- **Portée d'entrée** (*input gate*) $\mathbf{i_t}$: contrôle si la cellule est écrite ou non
- $\mathbf{\tilde{C}_t}$: ce qui est écrit dans la cellule
  - $\sigma \in [0, 1]$ (contrôle/commutateur), $\tanh \in [-1, 1]$ (non-linéarité)



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

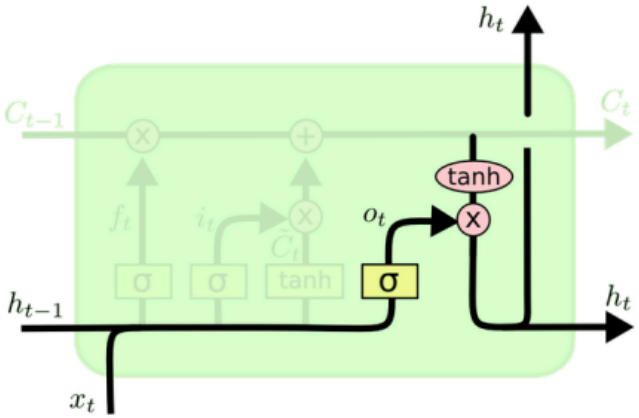
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- **Mise à jour** : efface la cellule si $f_t \approx 0$, ajoute le contenu de $\mathbf{i_t} \times \mathbf{\tilde{C}_t}$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- **Porte de sortie** $\mathbf{o_t}$ contrôle la proportion de l'état interne précédent remplacé par l'état actuel de la cellule

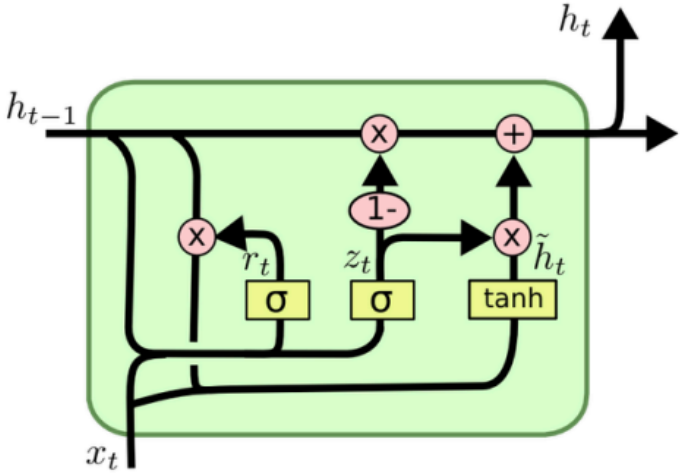


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

GRU

- GRU Cho et al. 2014 : Variante simplifiée des LSTM avec moins de paramètres
  - Combine les portes d'oubli et d'entrée dans une porte de *mise à jour*
  - Fusionne l'état de la cellule et l'état caché interne



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- ⊕ Plus simple que les LSTM, plus rapide, avec moins de paramètres
- ⊖ Performances généralement inférieures à architecture équivalente

Réseau récurrent bidirectionnel

On peut pour apporter plus de complexité en considérant que le temps est réversible (croissant et décroissant : lire de gauche à droite et de droite à gauche)
Ces réseaux sont un peu plus performants, à vous d'explorer !

Transformers : une nouvelle architecture

- Introduits par Vaswani et al. en 2017, les **Transformers** ont révolutionné le traitement du langage naturel (NLP).
- Basés sur le mécanisme d'**attention**, ils permettent de traiter efficacement les dépendances longues dans les séquences.
- Abandonnent les architectures séquentielles traditionnelles (RNN, LSTM) en faveur d'un traitement parallèle des données.
- Principaux composants :
  - **Attention multi-têtes**
  - **Encoders** et **Decoders**

Transformers : Architecture Générale

- Les Transformers sont basés sur des blocs d'encodeurs et décodeurs empilés.
- **Encoders** : composés de couches d'auto-attention et de feed-forward.
- **Decoders** : ajoutent un mécanisme d'attention croisée avec la séquence encodée.

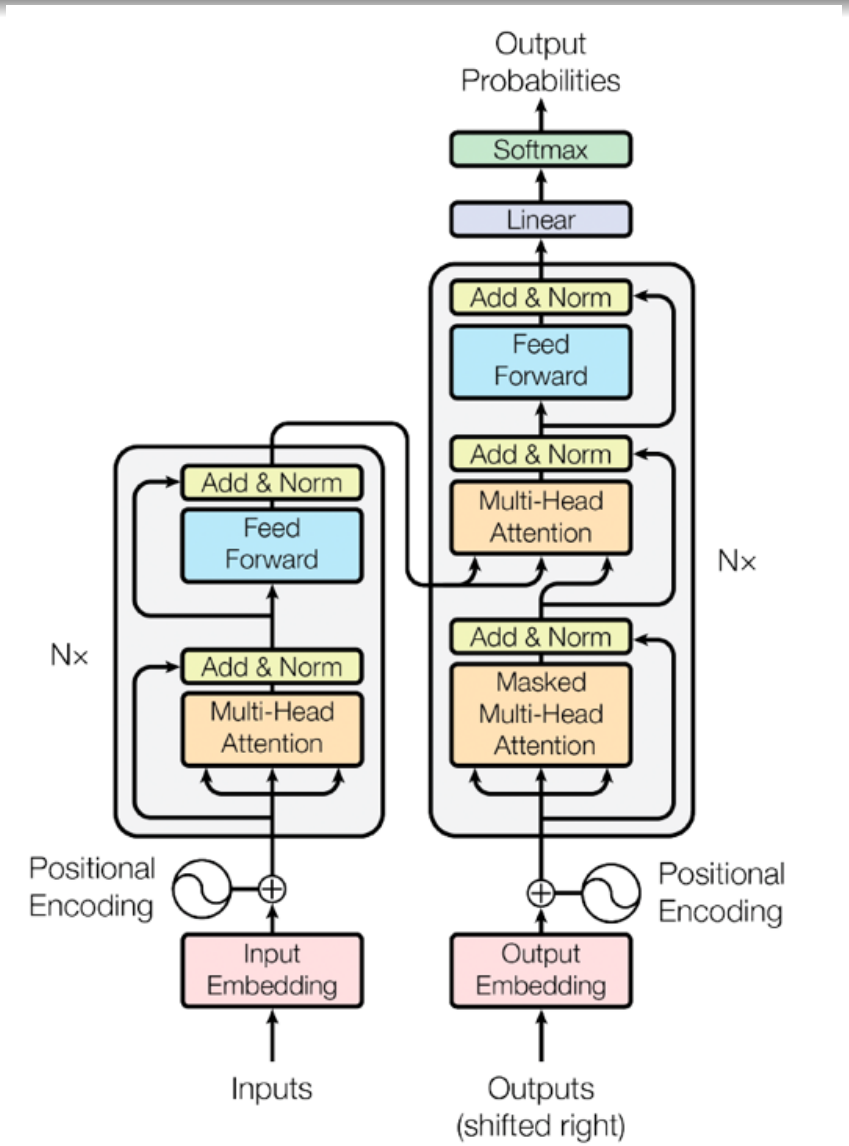


Schéma global d'un Transformer (Source : Vaswani et al., 2017)

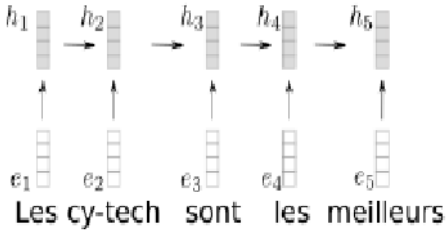
Attention : Scaled Dot-Product

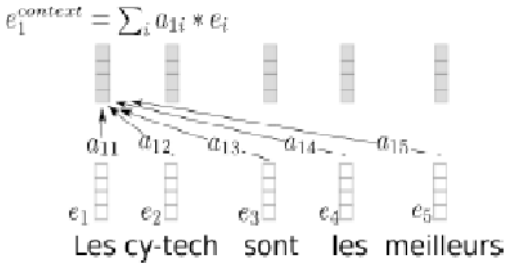
- Le cœur du Transformer est le mécanisme d'**attention**, qui permet de concentrer l'attention sur des parties spécifiques de la séquence.
- Chaque mot de la séquence est associé à trois matrices : **Query** (Q), **Key** (K) et **Value** (V).
- Le score d'attention est calculé via un produit scalaire entre les matrices Q et K :

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

- d_k : dimension des clés (key).
- Le terme $\frac{1}{\sqrt{d_k}}$ est un facteur de normalisation pour éviter des gradients trop importants.

Le mécanisme d'attention





Avantage 1: propagation de l'information

- Prise en compte directe de chaque mot indépendamment de sa position dans la phrase

=> Risque d'oubli d'information plus faible que pour le RNN

Inconvénient 1: Invariant à la permutation des mots

- Prise en compte directe de chaque mot indépendamment de sa position dans la phrase

Nous avons perdu l'information de position des mots dans $\sum_i a_{ji} e_i$

Avantage 2: Parallelisation

- Le calcul de chaque $e_i^{context}$ peut-être effectuée séparément.
  - Rappel : besoin des états précedents avec le RNN
  - Gain de temps pour ajouter des couches

Inconvénient 2: Besoin en mémoire quadratique

- Nous devons gérer la matrice des poids d'attention a_{ij} qui est de taille $n_{seq} \times n_{seq}$
- Généralement, limitation à des séquences de taille $n_{seq} = 512$ ou 1024

Figure: Mécanisme d'attention (Source : Vaswani et al., 2017)

Attention Multi-têtes

- Pour améliorer la capture de différentes relations entre les mots, les Transformers utilisent des mécanismes d'**attention multi-têtes**.
- Au lieu d'une seule attention, plusieurs "têtes" sont utilisées, chacune avec ses propres Q, K, et V.
- Formule de l'attention multi-têtes :

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$

- Chaque tête d'attention est calculée comme une attention individuelle :

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

L'Encoder du Transformer

- Un bloc d'**encodeur** est composé de :
  - Une couche d'attention multi-têtes.
  - Une couche feed-forward entièrement connectée.
- Les opérations sont accompagnées de normalisation couche par couche :

$$\text{Encoder} = \text{LayerNorm}(X + \text{Attention}(X, X, X))$$

$$\text{Encoder} = \text{LayerNorm}(X + \text{FeedForward}(X))$$

Le Decoder du Transformer

- Le bloc de décodeur est similaire à l'encodeur, mais avec une couche supplémentaire d'attention croisée.
- Cela permet au décodeur de se concentrer sur les informations de l'encodeur et de générer la séquence de sortie.

$$\text{Decoder} = \text{LayerNorm}(Y + \text{Cross-Attention}(Y, X, X))$$

- Le décodeur peut également fonctionner en auto-régression, où chaque mot prédit est utilisé comme entrée pour prédire le suivant.

Les Embeddings et l'Encodage Positionnel

- Les Transformers ne possèdent pas d'ordre séquentiel inhérent comme les RNNs.
- Un **encodage positionnel** est donc ajouté aux embeddings pour indiquer la position des mots dans la séquence.
- Encodage positionnel sinusoidal :

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

BERT : Bidirectional Encoder Representations from Transformers

- BERT (Devlin et al., 2018) est basé sur l'encodeur des Transformers.
- Utilise un pré-entraînement bidirectionnel sur un grand corpus de texte non étiqueté.
- Deux tâches de pré-entraînement :
  - **Masked Language Modeling (MLM)** : certaines parties des phrases sont masquées et BERT doit les prédire.
  - **Next Sentence Prediction (NSP)** : BERT apprend à prédire si une phrase suit une autre dans un texte.

`https://arxiv.org/pdf/1810.04805`

Avantages de BERT

- **Contexte bidirectionnel** : BERT capture une représentation plus riche du contexte.
- **Fine-tuning** : BERT peut être ajusté facilement pour des tâches spécifiques (classification, NER, QA).
- Résultats impressionnants sur des benchmarks comme **GLUE** et **SQuAD**.
- Modèle pré-entraîné : disponible en différentes tailles (BERT-Base, BERT-Large).

Large Language Models (LLM)

- Les **LLMs** sont des modèles massifs, souvent basés sur l'architecture Transformer.
- Entraînés sur d'énormes corpus de données pour générer des textes de manière cohérente.
- Exemples de LLMs : GPT, T5, et BERT.
- Taille des modèles : des milliards de paramètres (GPT-3 compte 175 milliards de paramètres).

Comparaison : GPT vs BERT

- **GPT (Generative Pre-trained Transformer)** :
  - Basé sur la partie **Decoder** du Transformer.
  - Unidirectionnel : prédit un mot à la fois à partir de gauche à droite.
  - Génération de texte.
- **BERT** :
  - Basé sur la partie **Encoder**.
  - Bidirectionnel : prend en compte les contextes des deux côtés.
  - Excellents résultats pour les tâches de compréhension.

Entraînement et Fine-tuning des LLMs

- **Pre-training** : les modèles sont pré-entraînés sur des corpus massifs, en utilisant des objectifs comme MLM (BERT) ou causal LM (GPT).
- **Fine-tuning** : ajustement des modèles sur des tâches spécifiques en fonction des besoins (classification, traduction, QA).
- Importance des ressources : l'entraînement des LLM nécessite des infrastructures massives (GPU/TPU, données, temps).

Applications des LLM

- **Classification de texte** : sentiment analysis, spam detection.
- **Génération de texte** : chatbots, résumé automatique, génération de code.
- **Traduction automatique** : amélioration des performances dans la traduction neuronale.
- **Question-Answering (QA)** : moteurs de recherche intelligents.

Conclusion

- Les Transformers et BERT ont redéfini le traitement du langage naturel grâce à l'attention.
- Les LLM, comme GPT et BERT, ont des capacités remarquables, mais nécessitent d'énormes ressources pour l'entraînement.
- L'avenir des LLM est prometteur avec des applications dans de nombreux domaines, du NLP à l'intelligence artificielle générale.

Références

1. Le cours très complet du CNAM : https://cedric.cnam.fr/vertigo/cours/ml2/coursDeep6.html
2. ChatGPT
3. Apprentissage profond de Francois Chollet
4. Cours de Deep Learning, Paul Gay et Yann Vernaz
5. https://arxiv.org/abs/1706.03762
6. Natural Language Processing with Transformers, Edition O'Reilly by Lewis Tunstall, Leandro von Werra, Thomas Wolf https://huggingface.co/