

Large Language Models

Redha Moulla

29 - 31 janvier 2025

Plan de la formation

- Introduction au LLMs
- Eléments de deep learning
- Large Language Models
- Retrieval Augmented Generation (RAG)
- Agents avec LangChain
- Alignement et fine tuning des LLMs
- Déploiement des LLMs et état de l'art

Qu'est-ce qu'un modèle de langage ?

Modèles de langage

- **Définition :** Un modèle de langage (LM) est une représentation de la structure du langage (comment les mots apparaissent les uns avec les autres).
- **En pratique :**
 - Génère du texte cohérent (complète des phrases, rédige des résumés, etc.).
 - Évalue la plausibilité d'une séquence de mots.
 - Sert de base pour des tâches comme la traduction, la classification, la Q&R, etc.
- **Exemples simples :**
 - Saisie prédictive (smartphone).
 - Correction orthographique et grammaticale.

Formulation mathématique d'un modèle de langage

Objectif : apprendre la distribution de probabilité d'une séquence de mots.

Soit une séquence de T mots :

$$X = (x_1, x_2, \dots, x_T).$$

- **Modèle de langage** : un LM vise à approximer

$$p(X) = p(x_1, x_2, \dots, x_T).$$

- **Règle de la chaîne (chain rule)** :

$$p(x_1, \dots, x_T) = \prod_{t=1}^T p(x_t \mid x_1, x_2, \dots, x_{t-1}).$$

- Dans la pratique, on apprend un modèle paramétré p_θ :

$$p_\theta(x_t \mid x_1, x_2, \dots, x_{t-1}),$$

grâce à de vastes corpus textuels.

Exemple de modèle de langage

Exemple : Considérons la phrase :

Le chat mange une souris .

Un LM estime

$$p(\text{ Le }, \text{ chat }, \text{ mange }, \text{ une }, \text{ souris }).$$

Il calcule donc :

$$p(\text{ Le }) \times p(\text{ chat } | \text{ Le }) \times p(\text{ mange } | \text{ Le }, \text{ chat }) \times \dots$$

Brève histoire des modèles de langage

- **Modèles n-gram (années 1980–1990) :**

- Calcul de la probabilité d'un mot en fonction des (n-1) mots précédents.
- Approches efficaces sur de grands corpus, mais peu adaptées aux dépendances longues.

- **Réseaux de neurones récurrents (années 1990–2010) :**

- Introduction des RNN, LSTM, GRU pour modéliser les séquences.
- Difficultés avec les dépendances longues (problèmes de gradients).

- **Word Embeddings (dès 2013) :**

- Apparition de Word2Vec et GloVe : représentations vectorielles des mots.
- Nouvelle façon de capturer la similarité sémantique.

- **Transformers et LLMs (depuis 2017) :**

- Mécanisme d'attention (modèle Transformer) : traitement parallèle et capture des dépendances globales.
- BERT, GPT, Claude, Llama, Mistral...
- Nouvelles performances en génération et compréhension du langage.

Les modèles n-gram

- **Principe général :**

$$p(w_1, w_2, \dots, w_T) \approx \prod_{t=1}^T p(w_t | w_{t-1}, \dots, w_{t-n+1}).$$

On limite le contexte à ($n-1$) mots pour chaque prévision.

- **Exemple :** Considérons un modèle bigram ($n = 2$) et la phrase

"Le chat mange une souris."

La probabilité est approximée par :

$$p("Le") \times p("chat" | "Le") \times p("mange" | "chat") \times p("une" | "mange") \times p("souris")$$

- **Limites :**

- Incapacité à capturer des dépendances longues.
- Le modèle devient volumineux (beaucoup de paramètres) et sujet à la rareté des données (d'où l'intérêt des techniques de lissage).

Fondamentaux du deep learning

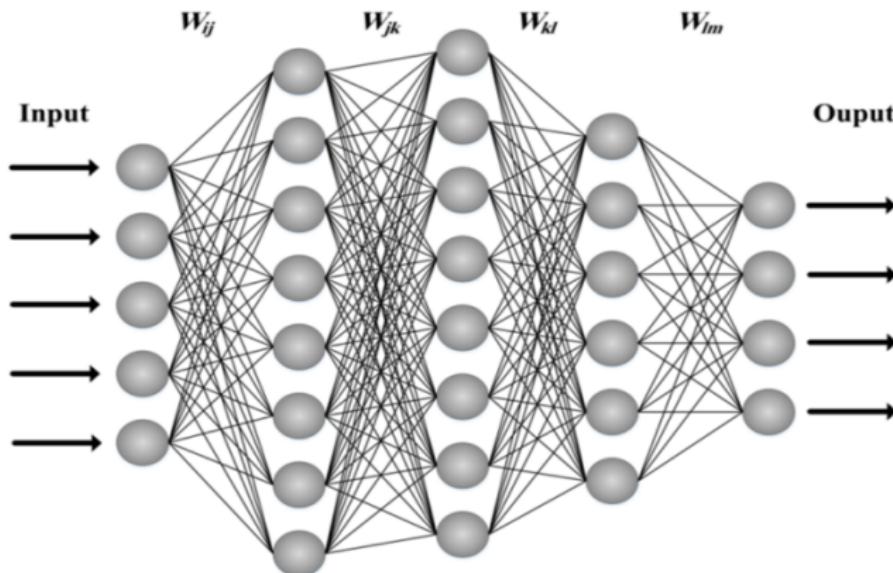
Introduction aux réseaux de neurones

Définition : Les réseaux de neurones sont des modèles computationnels inspirés par le fonctionnement des neurones dans le cerveau humain. Ils sont capables d'apprendre des tâches complexes en modélisant des relations non linéaires entre les entrées et les sorties.

Caractéristiques :

- **Extraction automatique des features** : Capacité d'adaptation et d'extraction des features à partir des données sans programmation explicite.
- **Modélisation non linéaire** : Aptitude à capturer des relations complexes dans les données.
- **Flexibilité** : Applicables à un large éventail de tâches et de typologies de données (images, langage naturel, données graphiques, etc.).

Illustration d'un réseau de neurones classique



Le neurone artificiel

Modèle Mathématique : Chaque neurone artificiel effectue une somme pondérée de ses entrées, ajoute un biais, et passe le résultat à travers une fonction d'activation pour obtenir la sortie.

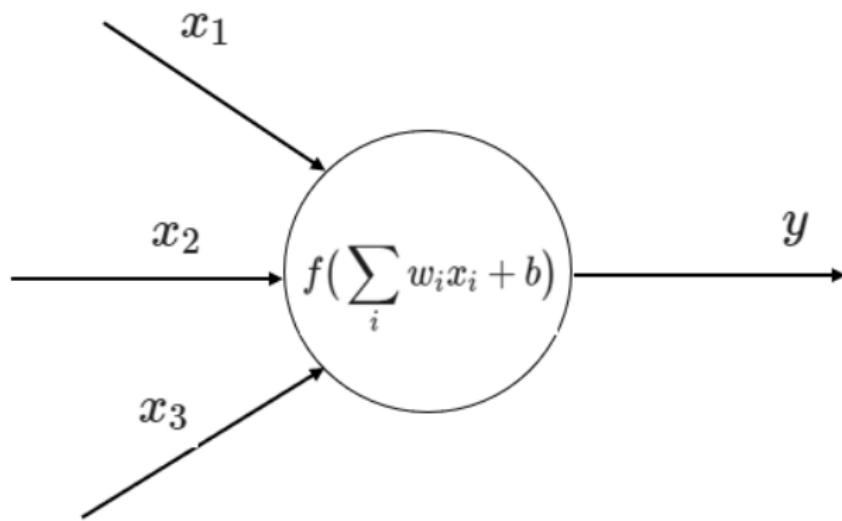
$$z_i = \sum_{j=1}^m w_{ij}x_j + b_i$$

$$a_i = f(z_i)$$

où :

- x_j représente l'entrée du neurone,
- w_{ij} est le poids associé à l'entrée x_j ,
- b_i est le biais du neurone,
- f est la fonction d'activation,
- z_i est le potentiel d'action pré-synaptique,
- a_i est la sortie activée du neurone.

Illustration d'un neurone artificiel



Fonctions d'activation

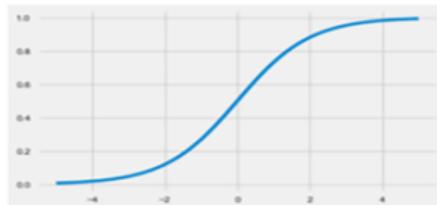
Les fonctions d'activation permettent aux modèles d'apprendre des relations plus complexes en capturant les non-linéarités dans les données.

- **Sigmoïde** : $\sigma(z) = \frac{1}{1+e^{-z}}$, plage de sortie (0, 1), utilisée pour la probabilité dans la classification binaire.
- **Tangente Hyperbolique (tanh)** : $\tanh(z)$, plage de sortie (-1, 1), version centrée et normalisée de la sigmoïde.
- **ReLU (Unité Linéaire Rectifiée)** : $f(z) = \max(0, z)$, non saturante, favorise la convergence rapide et permet d'éviter le problème de disparition des gradients.
- **Leaky ReLU** : $f(z) = \max(\alpha z, z)$, variante de ReLU qui permet un petit gradient lorsque $z < 0$.
- **Softmax** : Utilisée pour la couche de sortie des problèmes de classification multi-classes.

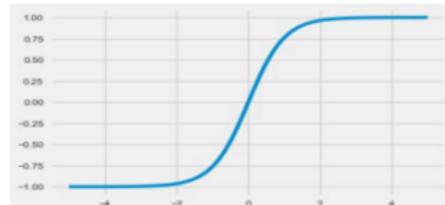
$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Illustration des fonctions d'activation

Sigmoïde



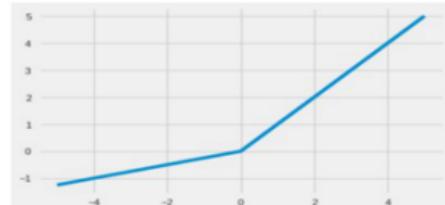
Tanh



ReLU



ReLU paramétrique



Entraînement d'un réseau de neurones artificiel

Principe d'Entraînement : L'entraînement d'un réseau de neurones consiste à ajuster ses poids pour minimiser une fonction de coût qui mesure l'erreur entre les prédictions et les vraies valeurs.

Descente de gradient stochastique (SGD) :

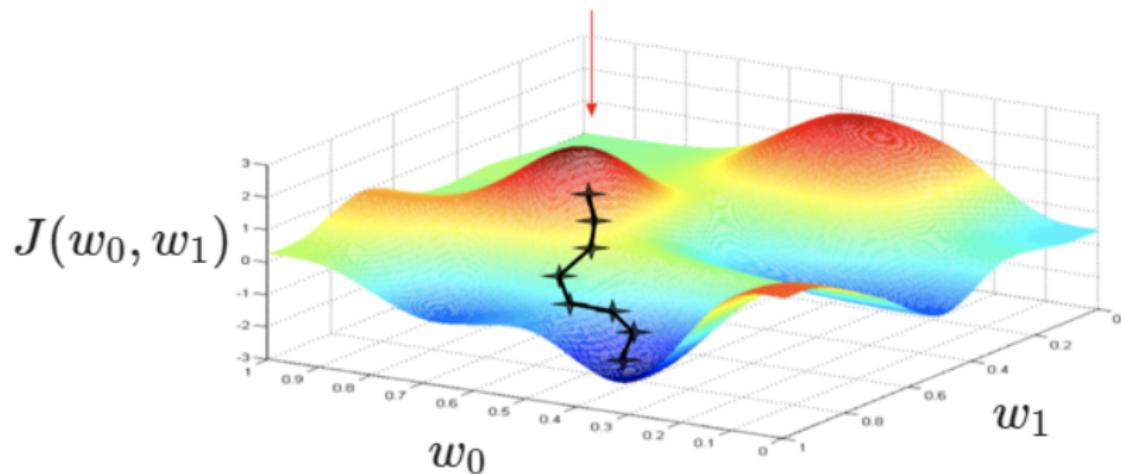
- Méthode d'optimisation utilisée pour mettre à jour les poids du réseau de manière itérative.
- À chaque itération, un sous-ensemble (batch) de données est utilisé pour calculer le gradient de la fonction de coût.
- Les poids sont mis à jour dans la direction opposée du gradient pour réduire l'erreur.

$$w_{new} = w_{old} - \eta \cdot \nabla_w J(w)$$

où :

- w_{old} et w_{new} sont les valeurs des poids avant et après la mise à jour,
- η est le taux d'apprentissage,
- $\nabla_w J(w)$ est le gradient de la fonction de coût par rapport aux poids.

Illustration graphique de la SGD



Entraînement des réseaux de neurones profonds

Dans les réseaux de neurones profonds, l'erreur calculée à la sortie du réseau dépend indirectement des poids des couches cachées. Ce lien indirect rend difficile de savoir comment ajuster ces poids pour réduire l'erreur.

Implications pour l'Entraînement :

- **Propagation de l'erreur** : Sans un mécanisme pour propager l'erreur de la sortie vers les couches antérieures, il est impossible de déterminer l'impact de chaque poids sur l'erreur finale.
- **Complexité de l'ajustement des Poids** : Chaque poids dans les couches cachées affecte l'erreur de sortie de manière complexe, nécessitant une méthode précise pour leur ajustement.



Rétropropagation du gradient

Il est facile de calculer les gradients correspondant à la dernière couche $\frac{\partial J}{\partial W^{(n)}}$ car l'erreur J dépend immédiatement de $W^{(n)}$.



Examinons maintenant le cas de la couche $n - 1$:

$$\frac{\partial J}{\partial W^{(n-1)}} = \frac{\partial J}{\partial O^{(n)}} \frac{\partial O^{(n)}}{\partial O_{n-1}} \frac{\partial O_{n-1}}{\partial W^{(n-1)}}$$

Surapprentissage dans les réseaux de neurones

Le surapprentissage (overfitting) se produit lorsqu'un réseau de neurones apprend trop bien les détails et le bruit des données d'entraînement, au détriment de sa capacité à généraliser sur de nouvelles données.

Le surapprentissage dans les réseaux de neurones peut se produire pour différentes raisons :

- Complexité excessive du modèle
- Manque de diversité dans les données d'entraînement
- Entraînement prolongé

Stratégies pour atténuer le surapprentissage :

- **Régularisation** : Techniques de régularisation telles que L1/L2, Dropout, pour limiter la complexité du modèle.
- **Dropout** : "Eteindre" un ensemble de neurones choisis aléatoirement pendant l'entraînement.
- **Early Stopping** : Arrêt de l'entraînement lorsque la performance sur un ensemble de validation cesse de s'améliorer.
- **Augmentation de Données** : Augmenter la variété des données d'entraînement pour améliorer la robustesse du modèle.

Représentations distribuées

Introduction aux Embeddings

• Qu'est-ce qu'un Embedding ?

- Un embedding est une représentation vectorielle d'un mot qui capture le contexte du mot dans un document, des relations sémantiques et syntaxiques avec d'autres mots.
- Ils transforment des mots en vecteurs de nombres pour que les algorithmes de machine learning puissent les traiter efficacement.

• Pourquoi sont-ils importants ?

- Les embeddings permettent de capturer non seulement l'identité d'un mot mais aussi ses aspects sémantiques et contextuels.
- Ils facilitent des tâches telles que la classification de texte, la traduction automatique, et la détection des sentiments.

• Évolution des embeddings

- Historiquement, les mots étaient représentés comme des indices ou des vecteurs one-hot, où chaque mot est indépendant des autres.
- Les embeddings modernes, tels que Word2Vec et GloVe, représentent les mots dans des espaces vectoriels continus où les mots de sens similaires sont proches les uns des autres.

Différence entre One-hot Encoding et Embeddings

• One-hot Encoding

- Chaque mot est représenté par un vecteur avec un '1' dans la position qui lui est propre et des '0' partout ailleurs. Ce type de représentation est simple mais très inefficace en termes d'espace et ne capture pas les relations entre les mots.
- Exemple : pour un vocabulaire de dimension 100 000 :

Véhicule = [0, 0, 1, 0, 0, 0, 0, 0, ..., 0, 0]

Voiture = [0, 0, 0, 0, 0, 1, 0, 0, 0, ..., 0, 0]

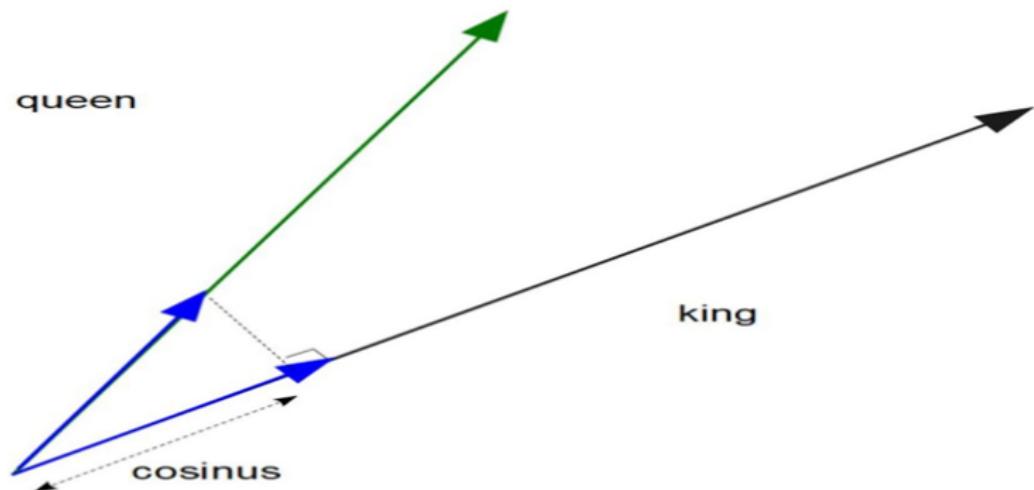
Cette représentation ne permet pas de prendre en compte la dimension sémantique

• Embeddings

- Les embeddings représentent les mots comme des vecteurs denses de nombres flottants (généralement entre 50 et 300 dimensions).
- Cette représentation est beaucoup plus riche et peut capturer des relations complexes entre les mots, comme la similarité sémantique.

Similarité sémantique

Nous sommes intéressés par des représentations de mots qui capturent la distance sémantique, sous forme de produit scalaire par exemple (ou distance cosiné).



Représentations distribuées : Principes

“You shall know a word by the company it keeps”

— J.R. Firth (1957)

Les mots sont similaires s'ils apparaissent fréquemment dans le même contexte.

- Il conduit son *véhicule* pour rentrer à la maison.
- Il conduit sa *voiture* pour rentrer chez lui.

Construction d'une représentation distribuée

Considérons le corpus suivant à titre d'exemple : **cnn in crop analysis**

cnn and svm are widely used.

linear_regression performed along with svm

linear_regression for crop and farm

svm being used for farm monitoring

Do cnn, svm and linear_regression appear in the same context?

Le vocabulaire est alors :

[cnn, in, crop, analysis, and, svm, are, widely, used,
linear_regression, performed, along, with, for, farm, being,
monitoring, do, appear, the, same, context]

$$\dim(\text{vocabulaire}) = 22$$

Similarité sémantique

La matrice de co-occurrence représente la fréquence à laquelle les mots apparaissent ensemble deux à deux.

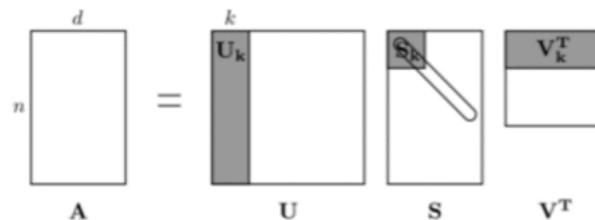
```
cnn in crop ...  
cnn [[0, 2, 1, 1, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1],  
in [2, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1],  
crop [1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
... [1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[2, 1, 0, 0, 1, 0, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
[1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[1, 0, 0, 0, 1, 2, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
[1, 1, 0, 1, 2, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1],  
[0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 2, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],  
[1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0]]]
```

Réduction de la dimension

La décomposition en valeurs singulières est une technique permettant de réduire la dimension des données (similaire à l'ACP).

Étant donné une matrice $A \in \mathbb{R}^{n \times d}$

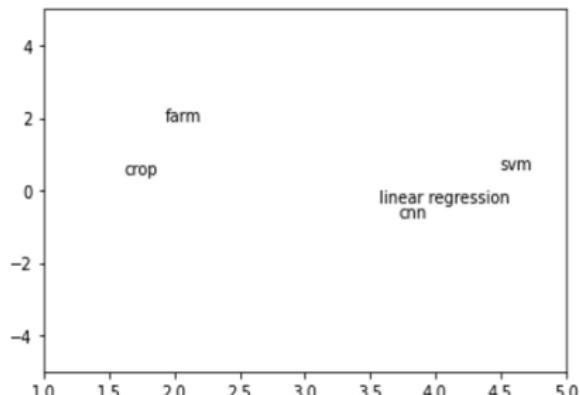
$$A = UDV^T \quad \text{où} \quad U \in \mathbb{R}^{n \times r}, D \in \mathbb{R}^{r \times r}, V \in \mathbb{R}^{d \times r}.$$



U est la matrice contenant les représentations (vecteurs de mots)

Visualisation des représentations distribuées

```
cnn :[ [ 3.72113518, -0.73233585],  
ln [ 2.7940387 , -1.40864784],  
crop [ 1.61178082, 0.44786021],  
... [ 0.77784994, -0.31230389],  
[ 2.12245048, 1.29571556],  
[ 4.49293777, 0.58090417],  
[ 1.33312748, 0.66758743],  
[ 1.33312748, 0.66758743],  
[ 2.25882809, 1.80738544],  
[ 3.56593605, -0.31006976],  
[ 0.95394179, 0.079159 ],  
[ 0.95394179, 0.079159 ],  
[ 0.95394179, 0.079159 ],  
[ 0.95394179, 0.079159 ],  
[ 1.92921553, 1.94783497],  
[ 1.92921553, 1.94783497],  
[ 1.12301312, 1.42126096],  
[ 1.12301312, 1.42126096],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175]]
```



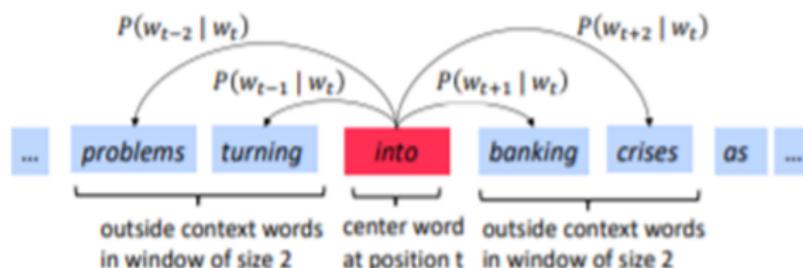
Word2Vec : Principles

- **Principes de base :**

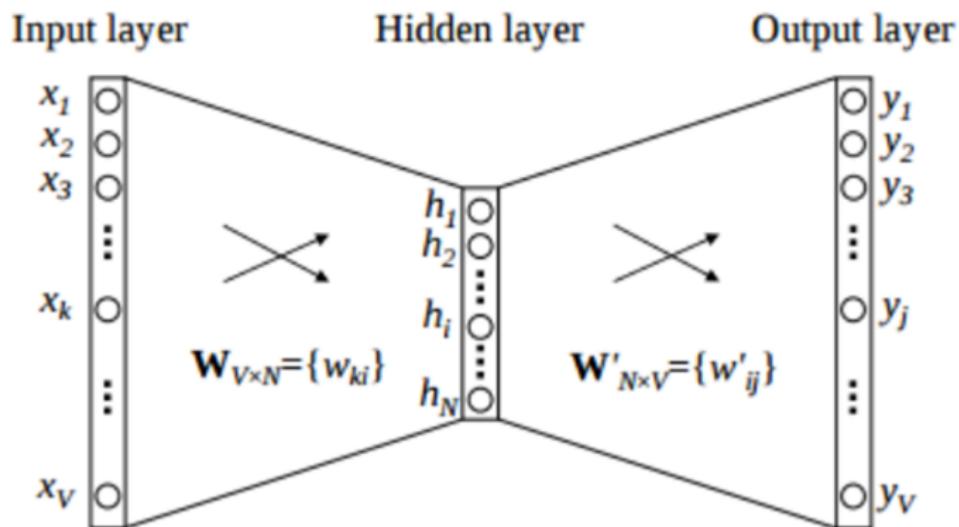
- Word2Vec est basé sur l'hypothèse distributionnelle : les mots qui apparaissent dans des contextes similaires ont des significations similaires.
 - Utilise un réseau de neurones peu profond pour apprendre des embeddings de mots à partir de grands corpus.

- Deux architectures principales :

- ① *Continuous Bag of Words (CBOW)* : Prédit le mot cible à partir du contexte.
 - ② *Skip-Gram* : Prédit le contexte à partir du mot cible.



Architecture du modèle CBOW



Modèle CBOW : Formulation mathématique

- **Objectif** : Prédire le mot cible w_t en fonction du contexte C .
- **Fonction de prédiction** :

$$P(w_t|C) = \frac{e^{\mathbf{v}_{w_t}^T \cdot \mathbf{h}}}{\sum_{w \in W} e^{\mathbf{v}_w^T \cdot \mathbf{h}}} \quad (1)$$

- **Où** :
 - \mathbf{v}_w est le vecteur de l'embedding du mot w .
 - \mathbf{h} est le vecteur du contexte, la moyenne des embeddings des mots du contexte.
 - W est l'ensemble de tous les mots du vocabulaire.
- **Optimisation** : Minimisation de la fonction de coût, souvent une forme de cross-entropy ou log-likelihood négatif.

Modèle Skip-Gram : Formulation mathématiques

- **Objectif** : Prédire les mots de contexte à partir du mot cible w_t .
- **Fonction de prédiction** :

$$P(C|w_t) = \prod_{w_c \in C} \frac{e^{\mathbf{v}_{w_c}^T \cdot \mathbf{h}}}{\sum_{w \in W} e^{\mathbf{v}_w^T \cdot \mathbf{h}}} \quad (2)$$

- **Où** :
 - \mathbf{v}_{w_c} est le vecteur de l'embedding du mot de contexte w_c .
 - \mathbf{h} est le vecteur de l'embedding du mot cible w_t .
- **Optimisation** : Minimisation de la fonction de coût, souvent une forme de cross-entropy ou log-likelihood négatif.

Optimisation

- **Negative sampling :**
 - Pour chaque paire de mots (cible, contexte), des paires négatives sont générées en échantillonnant des mots aléatoires du vocabulaire.
 - Améliore l'efficacité de l'entraînement en réduisant le nombre de calculs nécessaires pour chaque étape de mise à jour.
- **Sous-échantillonnage des mots fréquents :**
 - Les mots très fréquents (par exemple, "le", "est") sont sous-échantillonnés pour améliorer la qualité des embeddings et accélérer l'entraînement.
 - Réduit la dominance des mots fréquents dans la formation des embeddings.
- **Fonction de coût :**
 - Utilisation typique de la log-likelihood négative comme fonction de coût.
 - L'objectif est de maximiser la probabilité des mots réels (positifs) tout en minimisant celle des mots échantillonnés négativement.
- **Mise à jour des poids :**
 - Les poids du réseau sont mis à jour en utilisant des méthodes de descente de gradient stochastique.
 - Les gradients sont calculés par backpropagation à travers le réseau.

Negative sampling : Formulation mathématique

- **Objectif du Negative Sampling :**

- Simplifier l'optimisation de la fonction de coût en remplaçant le problème de classification multinomiale par une série de classifications binaires.

- **Formule du Negative Sampling :**

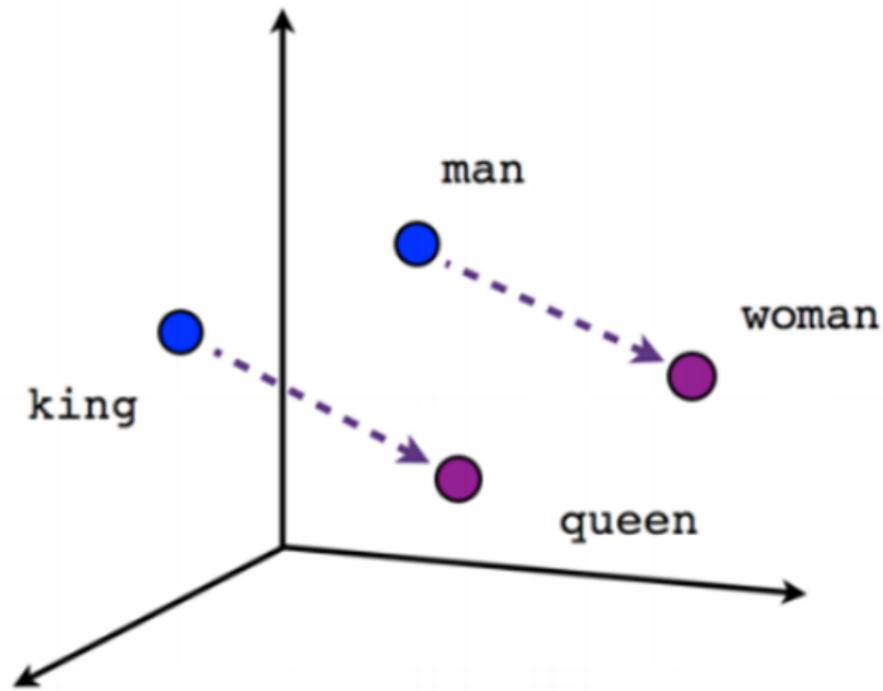
$$L(\theta) = \log \sigma(\mathbf{v}_{w_O}^T \mathbf{h}) + \sum_{i=1}^k \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-\mathbf{v}_{w_i}^T \mathbf{h})] \quad (3)$$

- Où $\sigma(x) = \frac{1}{1+e^{-x}}$ est la fonction logistique.
- \mathbf{v}_{w_O} est le vecteur du mot cible et \mathbf{h} est le vecteur du mot d'entrée (pour Skip-Gram) ou le vecteur du contexte (pour CBOW).
- k est le nombre de mots négatifs à échantillonner, tirés selon une distribution de probabilité $P_n(w)$.

- **Distribution de probabilité des mots négatifs ($P_n(w)$) :**

- Généralement, les mots négatifs sont échantillonnés selon une distribution qui favorise les mots fréquents, mais pas autant que la distribution de fréquence des mots dans le corpus.
- Une pratique courante est d'utiliser $P_n(w) \propto (U(w))^{3/4}$, où $U(w)$ est la fréquence brute du mot w .

Représentations Word2Vec



Applications pratiques de Word2Vec

- **Analogie de mots :**

- Word2Vec est célèbre pour capturer des relations complexes, comme "homme est à femme ce que roi est à reine".
- Permet de résoudre des analogies en utilisant des opérations arithmétiques simples sur les vecteurs de mots.

- **Clustering sémantique :**

- Les embeddings peuvent être utilisés pour regrouper des mots sémantiquement similaires, facilitant l'analyse de textes volumineux.

- **Amélioration des systèmes de recommandation :**

- Les vecteurs de mots de Word2Vec peuvent être utilisés pour améliorer la précision des recommandations en comprenant mieux les préférences des utilisateurs.

Autres représentations distribuées

Il existe d'autres représentations distribuées :

- Glove (2014) : proposé par une équipe de Stanford, il combine à la fois les techniques modernes de deep learning et les techniques statistiques (co-occurrences entre les mots). Il permet d'avoir des représentations des mots plus globales que Word2Vec.
- Fasttext (à partir de 2016) : la librairie a été créée par Facebook. Le modèle est entraîné sur des subwords (n-grams de caractères). Il est ainsi plus efficace pour traiter les mots inconnus (out of vocabulary).
- Représentations contextuelles (Transformers), etc.

Réseaux de neurones récurrents

Introduction aux RNN

Les Réseaux de Neurones Récurrents (RNN) sont une classe de réseaux de neurones artificiels spécialement conçus pour traiter des séquences de données, tels que des séries temporelles ou des séquences textuelles.

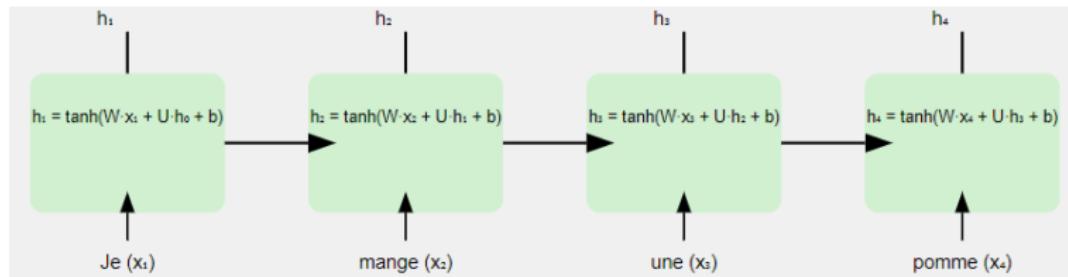
Caractéristiques:

- **Mémoire à court terme** : Les RNN ont la capacité de se "souvenir" d'informations passées grâce à leurs connexions récurrentes.
- **Traitement de séquences** : Ils sont particulièrement adaptés pour des tâches où les données sont séquentielles et où le contexte est important (ex: langage naturel, musique).
- **Modélisation de dépendances temporelles** : Les RNN peuvent capturer des dépendances temporelles et contextuelles dans les données.

Architecture des RNN

Cette architecture est dépliée pour montrer la séquence complète:

- Chaque bloc A est le même RNN à différents instants temporels.
- Le bloc A prend une entrée x_t et produit un état caché h_t , qui est alors passé à la même cellule au pas de temps suivant.
- L'état initial h_0 est souvent initialisé à zéro ou une petite valeur aléatoire.



Formulation mathématique des RNN

Un RNN est un réseau de neurones conçu pour traiter des séquences de données, caractérisé par sa capacité à maintenir une 'mémoire' des entrées antérieures dans ses connexions récurrentes.

Formulation mathématique d'un RNN simple pour une séquence de temps t :

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

où :

- x_t est l'entrée à l'instant t
- h_t est l'état caché à l'instant t
- y_t est la sortie à l'instant t
- W et b sont les paramètres du réseau
- σ est une fonction d'activation non-linéaire

Problème de disparition du gradient

Le problème de disparition du gradient survient lors de la rétropropagation dans les RNN traditionnels. Les gradients des paramètres peuvent devenir très petits, rendant l'apprentissage des dépendances à long terme difficile.

Mathématiquement, pendant la rétropropagation, si les valeurs de $\frac{\partial h_t}{\partial W}$ sont petites, le gradient total peut tendre vers zéro à mesure que T augmente.

Solutions au problème de disparition du gradient :

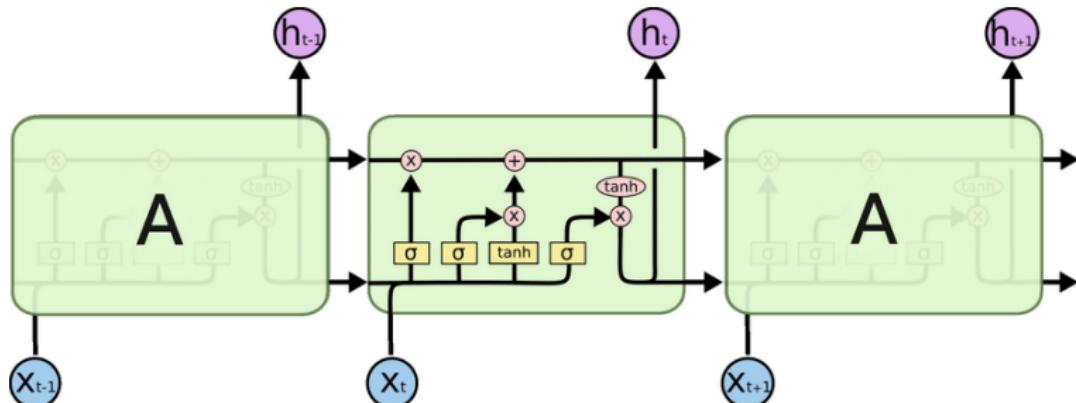
Pour atténuer le problème de disparition du gradient, différentes solutions ont été proposées :

- Utilisation de fonctions d'activation comme ReLU au lieu de sigmoïde ou tanh.
- Introduction d'architectures RNN avancées comme LSTM et GRU.
- Techniques de clipping de gradient pour éviter l'explosion des gradients.

RNN avec Long Short-Term Memory (LSTM)

Les LSTM sont une variante des RNN conçus pour mieux capturer les dépendances à long terme. Ils introduisent des 'cellules mémoire' avec des portes de régulation :

- Porte d'oubli : contrôle la quantité d'informations à retenir de l'état précédent.
- Porte d'entrée : contrôle la quantité d'informations à ajouter de l'entrée actuelle.
- Porte de sortie : détermine la quantité d'informations à transmettre à l'état suivant.



Formulation mathématique des LSTM

La formulation d'un LSTM pour un instant t :

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

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

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

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

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

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

où :

- C_t est l'état de la cellule à l'instant t
- h_t est l'état caché à l'instant t
- f_t, i_t, o_t sont respectivement les états des portes d'oubli, d'entrée, et de sortie
- Les W et b représentent les poids et biais
- σ est la fonction sigmoïde
- \tanh est la tangente hyperbolique

RNN avec Gated Recurrent Unit (GRU)

Les GRU sont une autre variante des RNN qui simplifient l'architecture des LSTM en combinant la porte d'oubli et la porte d'entrée en une seule porte de mise à jour.

Formule d'un GRU pour un instant t :

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

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

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

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

où :

- z_t est l'état de la porte de mise à jour
- r_t est l'état de la porte de réinitialisation
- \tilde{h}_t est l'état candidat pour h_t
- h_t est l'état caché final à l'instant t

Applications des RNN

Les RNN sont largement utilisés dans divers domaines tels que :

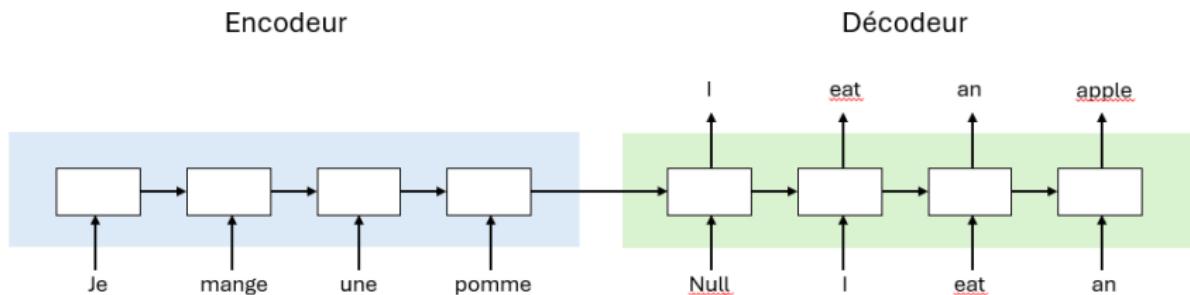
- Traitement du langage naturel : traduction automatique, génération de texte.
- Reconnaissance vocale : conversion de la parole en texte.
- Prévision de séries temporelles : prévision météorologique, analyse boursière.

Modèles Seq2Seq

Architecture de base

L'architecture seq-to-seq comprend deux composants principaux :

- **Encodeur** : Un réseau de neurones qui lit et encode la séquence d'entrée en un vecteur de contexte (une représentation dense de la séquence).
- **Décodeur** : Un autre réseau de neurones qui lit le vecteur de contexte pour générer la séquence de sortie, un élément à la fois.



Fonctionnement de l'Encodeur

- L'encodeur transforme la séquence d'entrée, souvent textuelle, en une série d'états cachés représentant l'information contenue dans la séquence.
- Utilise généralement des réseaux de neurones récurrents (RNN), LSTM ou GRU pour traiter les dépendances temporelles.

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$$

où \mathbf{x}_t est l'entrée à l'instant t , et \mathbf{h}_t est l'état caché.

Fonctionnement du décodeur

- Le décodeur commence par le vecteur de contexte fourni par l'encodeur pour commencer la génération de la séquence de sortie.
- À chaque étape, le décodeur est alimenté par son propre état caché précédent et parfois par une partie de la sortie précédemment générée.
- Le processus continue jusqu'à ce qu'un symbole de fin de séquence soit généré.

$$s_t = f(y_{t-1}, s_{t-1}, C)$$

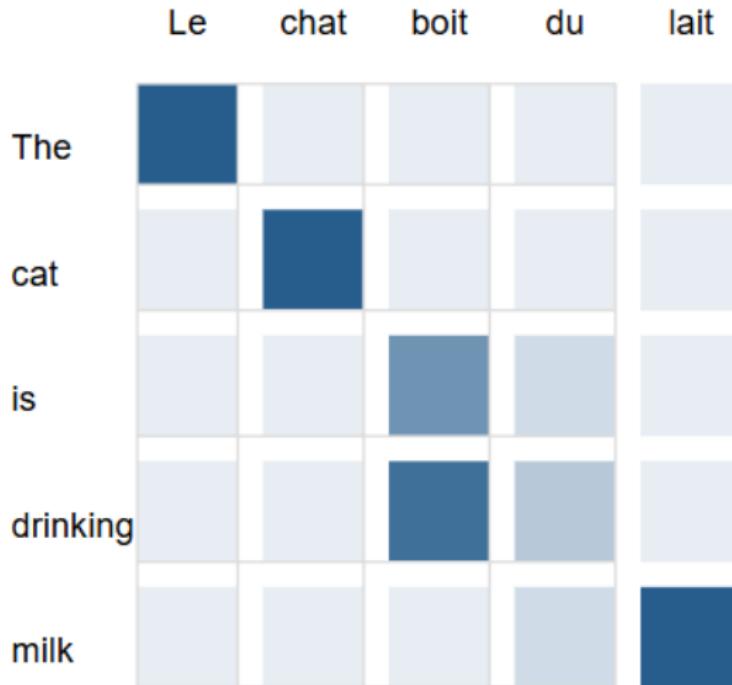
où y_{t-1} est la sortie générée à l'instant $t - 1$, s_t est l'état caché du décodeur, et C est le vecteur de contexte.

Introduction au mécanisme d'Attention

Le mécanisme d'attention est une amélioration clé apportée aux modèles seq-to-seq traditionnels, permettant au modèle de se "concentrer" sur différentes parties de la séquence d'entrée lors de la génération de la séquence de sortie.

- **Objectif** : Surmonter les limitations des encodeurs seq-to-seq qui compressent toute l'information d'une séquence d'entrée dans un vecteur de contexte fixe.
- **Avantage** : Améliore la capacité du modèle à gérer de longues séquences d'entrée, en rendant le processus de génération de la séquence de sortie plus dynamique et contextuellement informé.

Illustration du mécanisme d'Attention



Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

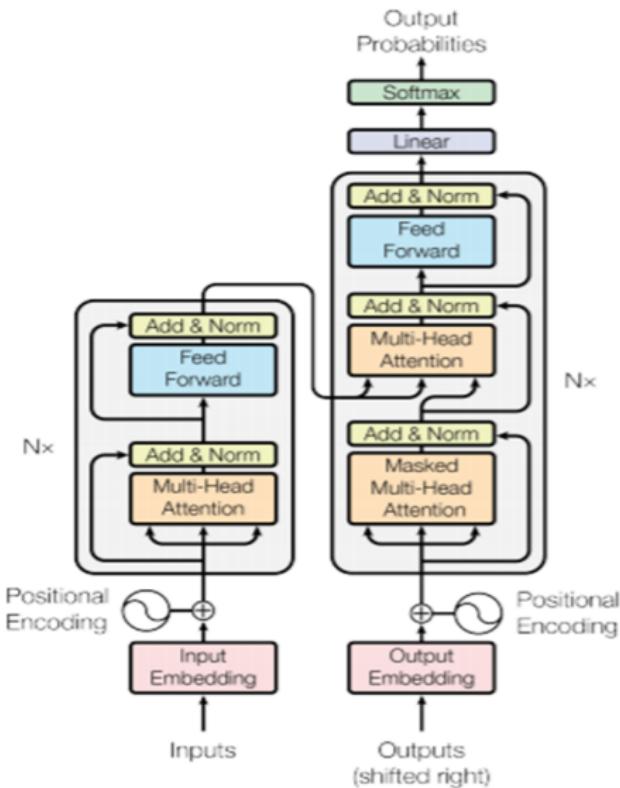
Lukasz Kaiser*
Google Brain
lukasz.kaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

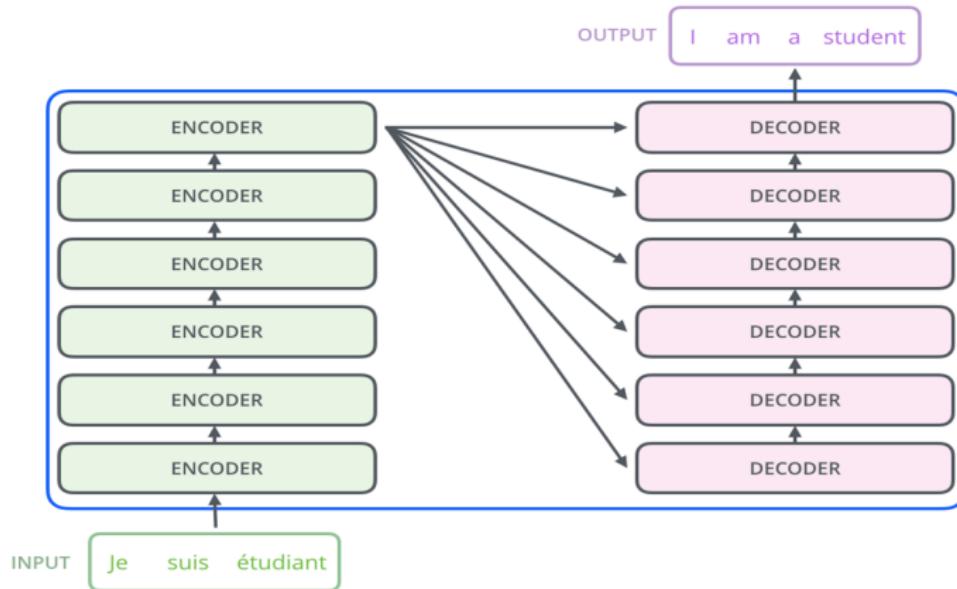
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to

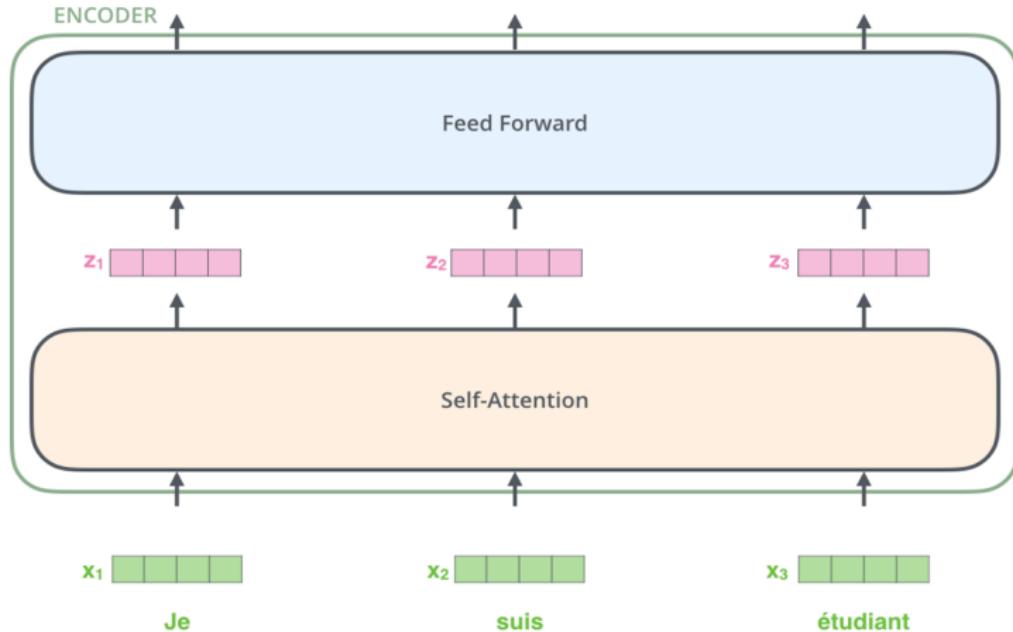
Transformer originel



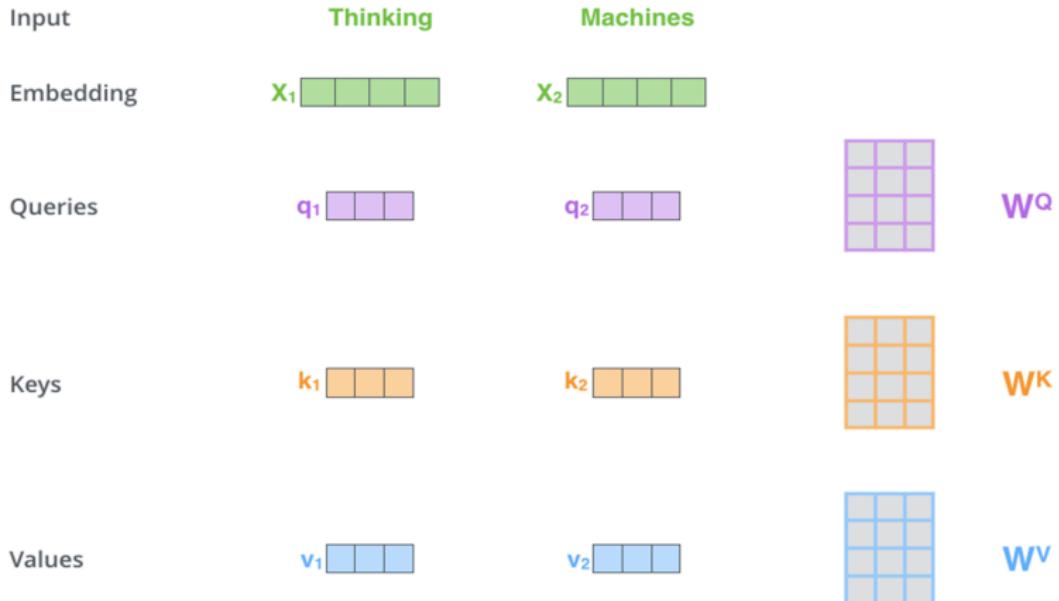
Mécanisme de cross-attention



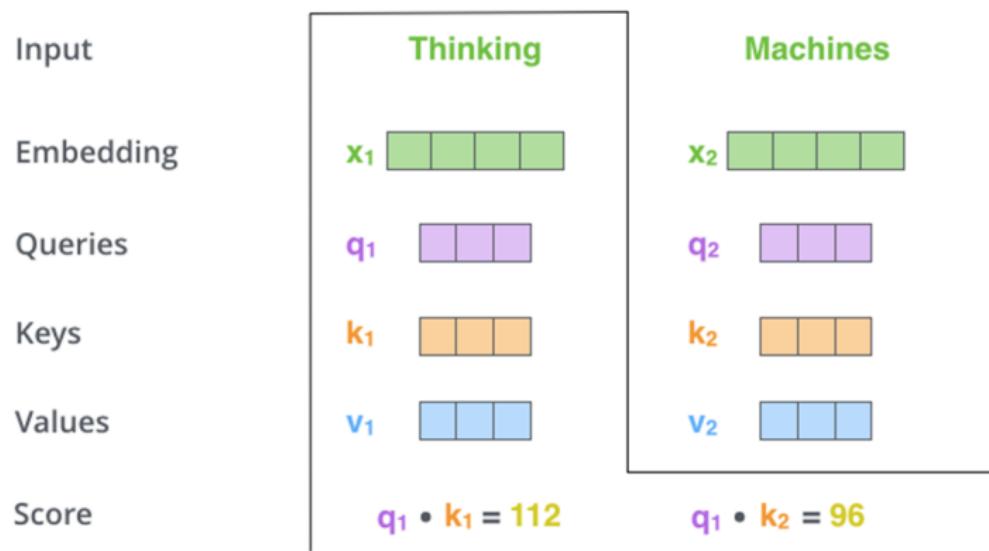
Mécanisme de self-attention



Fonctionnement du mécanisme de self-attention



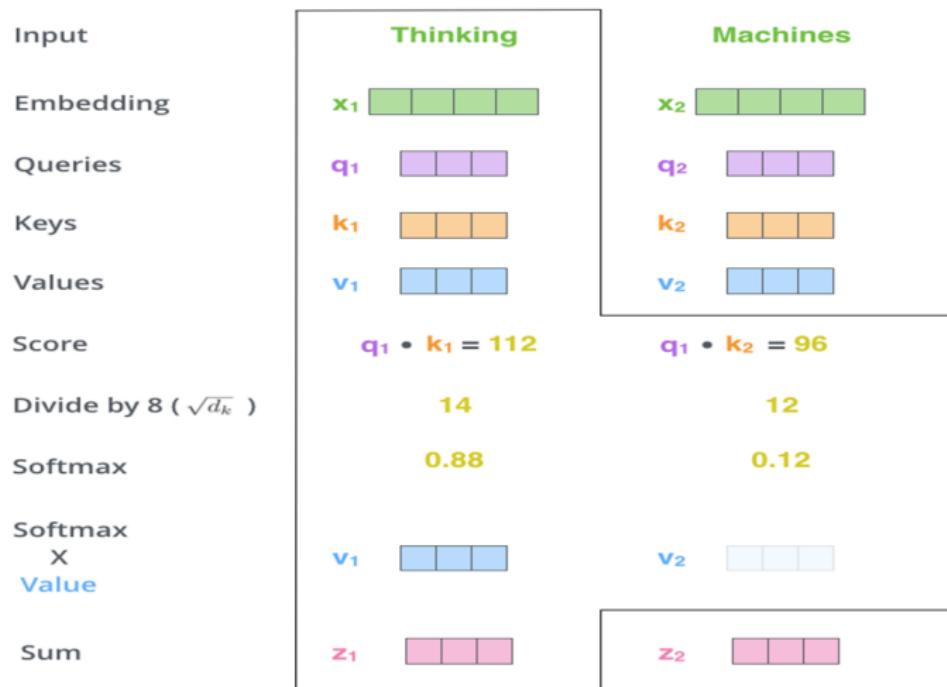
Calcul des scores de self-attention



Calcul des scores de self-attention

| Input | Thinking | | Machines | |
|------------------------------|-------------------------|--|------------------------|--|
| Embedding | x_1 | | x_2 | |
| Queries | q_1 | | q_2 | |
| Keys | k_1 | | k_2 | |
| Values | v_1 | | v_2 | |
| Score | $q_1 \bullet k_1 = 112$ | | $q_1 \bullet k_2 = 96$ | |
| Divide by 8 ($\sqrt{d_k}$) | 14 | | 12 | |
| Softmax | 0.88 | | 0.12 | |

Calcul de la nouvelle représentation



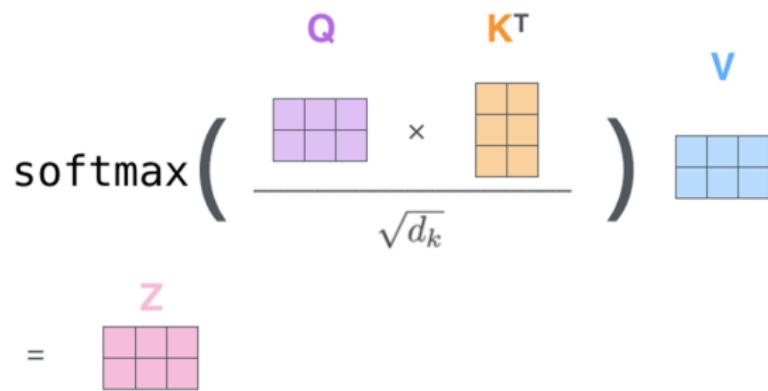
Formulation matricielle

$$\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$$

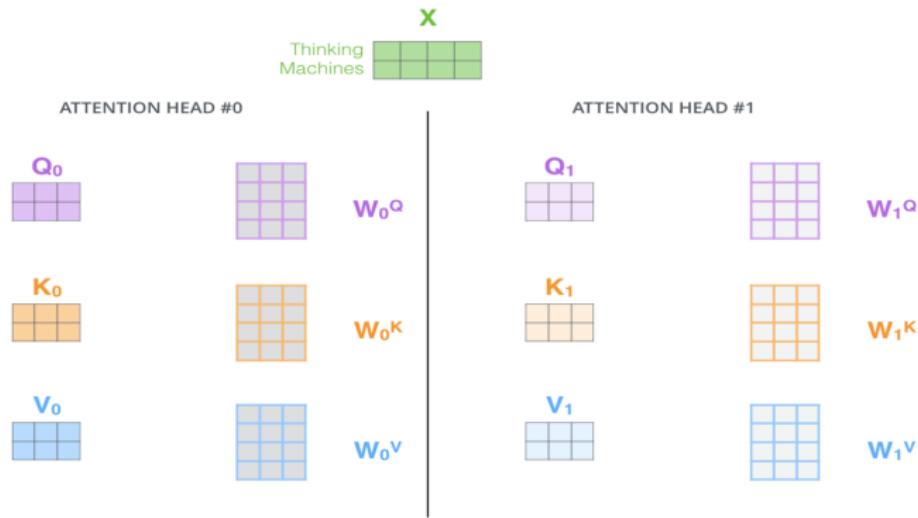
$$\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$$

$$\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$$

Formulation matricielle du mécanisme de self attention

$$\text{softmax} \left(\frac{\begin{matrix} \mathbf{Q} & \times & \mathbf{K}^T \\ \begin{matrix} \text{---} \end{matrix} & \begin{matrix} \text{---} \end{matrix} & \begin{matrix} \text{---} \end{matrix} \end{matrix}}{\sqrt{d_k}} \right) \mathbf{V}$$
$$= \mathbf{Z}$$


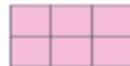
Multihead attention



Multihead attention

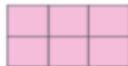
ATTENTION HEAD #0

Z₀



ATTENTION HEAD #1

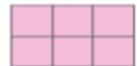
z₁



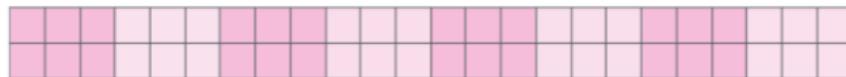
• • •

**ATTENTION
HEAD #7**

z7



$z_0 \quad z_1 \quad z_2 \quad z_3 \quad z_4 \quad z_5 \quad z_6 \quad z_7$



X



Les embeddings de position dans le Transformer

- Le modèle Transformer ne fonctionne pas de manière récurrente et ne sait pas, **a priori**, à quelle position se trouve chaque token dans la séquence.
- Pour fournir cette information d'ordre, on ajoute **un vecteur de position** à chaque embedding de mot.
- Dans la version d'origine, ces vecteurs sont **codés par des fonctions sinus et cosinus** :

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

- **pos** : position du token dans la séquence (de 0 à la longueur du texte).
- **i** : index d'une dimension particulière dans l'espace d'embedding.
- **d_model** : dimension du vecteur d'embedding.
- Cette méthode apprend au modèle à **distinguer** les positions et à capturer des informations de distance de manière efficace.

Tokénisation

- **Définition :** La tokénisation consiste à découper un texte en unités élémentaires appelées **tokens**. Ces tokens peuvent représenter des mots, des sous-mots, des caractères ou même des morceaux de mots.
- **Pourquoi est-ce important ?**
 - Les modèles de langage manipulent des indices ou des représentations numériques plutôt que des mots bruts.
 - Une bonne tokénisation permet de mieux gérer la morphologie (préfixes, suffixes), la gestion des mots rares, et de réduire la taille du vocabulaire.
- **Méthodes courantes :**
 - Découpage par espaces et ponctuation (simpliste).
 - Sous-mots (Byte-Pair Encoding, WordPiece, SentencePiece) : on segmente les mots en fragments fréquents pour mieux gérer les mots inconnus.
 - Caractères purs (dans certaines applications spécifiques).
- **Exemple (WordPiece) :**
 - Phrase : "J'adore le traitement automatique du langage"
 - Tokens : ["J", "'", "adore", "le", "traitement", "auto", "matique", "du", "langage"]

Byte Pair Encoding (BPE) : procédure

- **Objectif :**

- Réduire la taille du vocabulaire tout en gérant les mots inconnus.
- Capturer les éléments récurrents (préfixes, racines, suffixes) sous forme de sous-mots.

- **Principe :**

- ➊ Découper chaque mot en caractères au départ.
- ➋ Repérer la paire de symboles la plus fréquente dans tout le corpus.
- ➌ Fusionner cette paire pour former un nouveau symbole (sous-mot).
- ➍ Répéter les fusions jusqu'à la taille de vocabulaire souhaitée.

- **Avantages :**

- **Évite** les énormes vocabulaires composés de mots entiers.
- **Gère** de façon flexible les mots rares ou inconnus (en les segmentant en sous-unités).
- **Capture** la structure morphologique (préfixes, suffixes) via la répétition.

Byte Pair Encoding (BPE) : exemple

- **Phrase (mini-corpus)** : "la souris rit sous la pluie"

Au départ, on segmente en caractères (y compris les espaces) :

```
[ "l", "a", " ", "s", "o", "u", "r", "i", "s", " ", "r", "i", "t", " ",  
  "s", "o", "u", "s", " ", "l", "a", " ", "p", "l", "u", "i", "e"]
```

- **Étapes de fusion itératives** :

- ➊ On compte la fréquence de chaque paire de symboles (par ex. "l"+ "a" qui forme la, "s"+ "o", "o"+ "u", etc.).
- ➋ On **fusionne** la paire la plus fréquente pour former un nouveau **token**.
- ➌ On répète jusqu'à atteindre la taille de vocabulaire souhaitée (ici, on fera seulement quelques fusions pour l'exemple).

- **Exemple de fusions probables** :

- Fusion de "l" et "a" → "la" (vu deux fois dans la phrase).
- Fusion de "s" et "o" → "so".
- Fusion de "so" et "u" → "sou" (présent dans "souris" et "sous").

- **Résultat final** :

```
[ "la", "sou", "ri", "i", "s", "i", "t", "la", "p", "l", "u", "e"]
```

Performances du tranformer originel

Entraîné sur WMT 2014 English-German dataset, comprenant près de 4.5 millions de phrases sentence, le WMT 2014 English-French dataset, comprenant près de 36 millions de phrases.

- 8 NVIDIA P10 GPUs
- **Base** : 12 heures
- **Large** : 3.5 jours

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

| Model | BLEU | | Training Cost (FLOPs) | |
|---------------------------------|-------------|--------------|---------------------------------------|---------------------|
| | EN-DE | EN-FR | EN-DE | EN-FR |
| ByteNet [18] | 23.75 | | | |
| Deep-Att + PosUnk [39] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [38] | 24.6 | 39.92 | $2.3 \cdot 10^{19}$ | $1.4 \cdot 10^{20}$ |
| ConvS2S [9] | 25.16 | 40.46 | $9.6 \cdot 10^{18}$ | $1.5 \cdot 10^{20}$ |
| MoE [32] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [39] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [38] | 26.30 | 41.16 | $1.8 \cdot 10^{20}$ | $1.1 \cdot 10^{21}$ |
| ConvS2S Ensemble [9] | 26.36 | 41.29 | $7.7 \cdot 10^{19}$ | $1.2 \cdot 10^{21}$ |
| Transformer (base model) | 27.3 | 38.1 | $3.3 \cdot 10^{18}$ | |
| Transformer (big) | 28.4 | 41.8 | $2.3 \cdot 10^{19}$ | |

BERT

BERT (Bidirectional Encoder Representations from Transformers) représente une avancée majeure dans le NLP, introduisant une approche novatrice pour la modélisation de langage.

- **Contextualisation profonde** : Première architecture à utiliser pleinement la bidirectionnalité pour comprendre le contexte des mots, permettant une compréhension plus fine du langage.
- **Pré-entraînement et fine-tuning** : Méthodologie en deux étapes offrant une flexibilité pour l'adaptation à diverses tâches de TALN sans architecture spécifique à la tâche.
- **Impact sur le NLP** : Avant BERT, les approches de modélisation de langage étaient limitées par la compréhension unidirectionnelle ou des représentations statiques des mots. BERT a changé la donne en permettant des représentations contextuelles dynamiques.
- **Performances révolutionnaires** : À son introduction, BERT a établi de nouveaux standards de performance sur une gamme de benchmarks de NLP, y compris GLUE, SQuAD, etc.

Pré-entraînement de BERT

Objectifs de pré-entraînement :

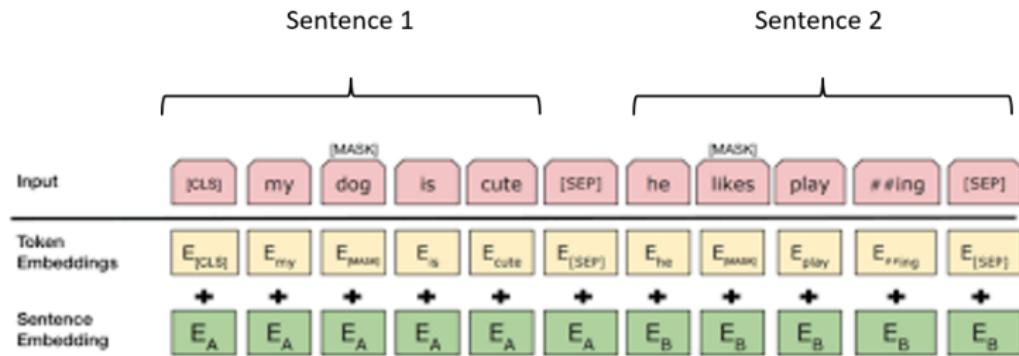
- **Masquage de token (MLM)** : 15% des tokens masqués aléatoirement, prédits par le modèle.

$$L_{\text{MLM}} = - \log P(\text{token masqué} | \text{contexte}) \quad (4)$$

- **Prédiction de la prochaine phrase (NSP)** : Déterminer si une phrase B suit logiquement une phrase A.

$$L_{\text{NSP}} = - [y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \quad (5)$$

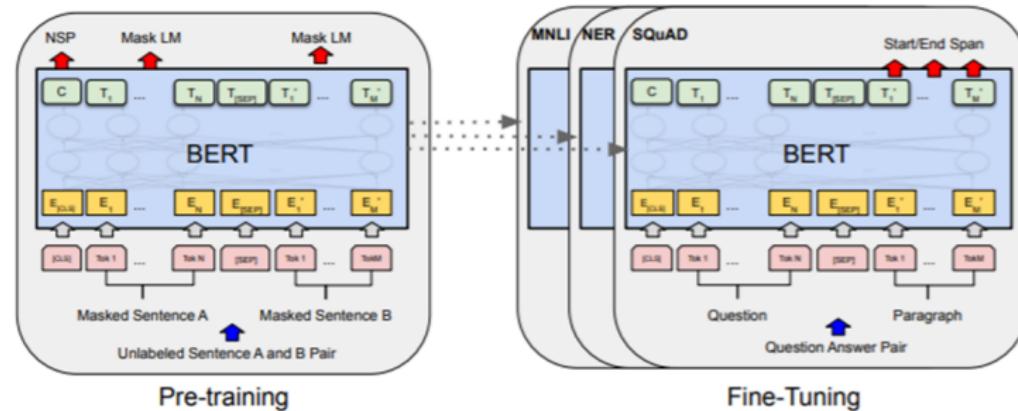
- Combinaison des pertes pour l'optimisation.



Fine-tuning de BERT pour des tâches spécifiques

Après pré-entraînement, BERT est affiné pour des tâches spécifiques:

- Ajout d'une couche de sortie spécifique à la tâche (classification, NER, QA).
 - Fine-tuning de tous les paramètres du modèle pré-entraîné sur le corpus de la tâche.



Performances de BERT

BERT a été pré-entraîné sur le BookCorpus (800 millions de mots) et English Wikipedia (2,500 millions de mots).

Deux modèles :

- **BERT Base** : 12 couches avec 110 millions de paramètres.
- **BERT Large** : 24 couches avec 340 millions de paramètres.

| 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 _{BASE} | 84.6/83.4 | 71.2 | 90.5 | 93.5 | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| BERT _{LARGE} | 86.7/85.9 | 72.1 | 92.7 | 94.9 | 60.5 | 86.5 | 89.3 | 70.1 | 82.1 |

Modèles autorégressifs - Generative Pretrained Transformer

Improving Language Understanding by Generative Pre-Training

Alec Radford

OpenAI

alec@openai.com

Karthik Narasimhan

OpenAI

karthikn@openai.com

Tim Salimans

OpenAI

tim@openai.com

Ilya Sutskever

OpenAI

ilyasu@openai.com

Abstract

Natural language understanding comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification. Although large unlabeled text corpora are abundant, labeled data for learning these specific tasks is scarce, making it challenging for discriminatively trained models to perform adequately. We demonstrate that large gains on these tasks can be realized by *generative pre-training* of a language model on a diverse corpus of unlabeled text, followed by *discriminative fine-tuning* on each specific task. In contrast to previous approaches, we make use of task-aware input transformations during fine-tuning to achieve effective transfer while requiring minimal changes to the model architecture. We demonstrate the effectiveness of our approach on a wide range of benchmarks for natural language understanding. Our general task-agnostic model outperforms discriminatively trained models that use architectures specifically crafted for each task, significantly improving upon the state of the art in 9 out of the 12 tasks studied. For instance, we achieve absolute improvements of 8.9% on commonsense reasoning (Stories Cloze Test), 5.7% on question answering (RACE), and 1.5% on textual entailment (MultiNLI).

GPT-3

- Développé par OpenAI, publié en 2020.
- Modèle auto-régressif basé sur l'architecture Transformer.

| Caractéristique | Valeur / Détail |
|------------------------------|--|
| Date | 2020 |
| Paramètres | 175 milliards |
| Corpus d'entraînement | Environ 300 milliards de tokens |
| Contexte maximum | 2048 tokens |
| Ressources GPU | Plusieurs centaines de GPU (type V100) |
| Coût d'entraînement (estim.) | 4,6 M\$ à 12 M\$ (selon les sources) |

- **Applications** : génération de texte, Q&R, résumé, traduction, assistance à la programmation.

Le few-shot learning

- Les grands modèles (comme GPT-3) peuvent réaliser des tâches **sans être explicitement réentraînés** dessus.
- Le principe repose sur **l'utilisation de quelques exemples** (exemples étiquetés) directement dans le prompt ou l'entrée.
- **Zero-shot** : aucune démonstration fournie, le modèle doit comprendre la tâche à partir de sa connaissance générale.
- **One-shot / Few-shot** : on inclut un nombre très réduit d'exemples (un ou quelques-uns) pour guider le modèle dans la résolution de la tâche.
- Exemple :

English: "cat" → French: "chat"

English: "house" → French: "maison"

English: "car" → French: "voiture"

English: "tree" → French:

ChatGPT

ChatGPT, développé par OpenAI, est un modèle de langage basé sur l'architecture GPT (Generative Pre-trained Transformer) optimisé pour comprendre et générer des dialogues naturels. L'entraînement de ChatGPT se décline selon les étapes suivantes :

- ① Pré-entraînement sur un corpus volumineux :** Comme GPT-3, ChatGPT est d'abord pré-entraîné sur un vaste ensemble de données textuelles, englobant un large éventail de la littérature disponible sur Internet, pour apprendre une compréhension générale du langage.
- ② Fine-tuning supervisé :** Ensuite, ChatGPT est affiné sur des dialogues spécifiques pour améliorer ses compétences conversationnelles. Cette étape utilise des paires question-réponse et des conversations pour enseigner au modèle des structures de dialogue et des réponses contextuellement appropriées.
- ③ Reinforcement Learning from Human Feedback (RLHF):** Utilisation de techniques de renforcement pour ajuster les réponses du modèle basées sur les préférences et les corrections fournies par des évaluateurs humains, raffinant davantage la pertinence et la naturalité des réponses.

Retrieval Augmented Generation (RAG)

Problèmes des modèles génératifs

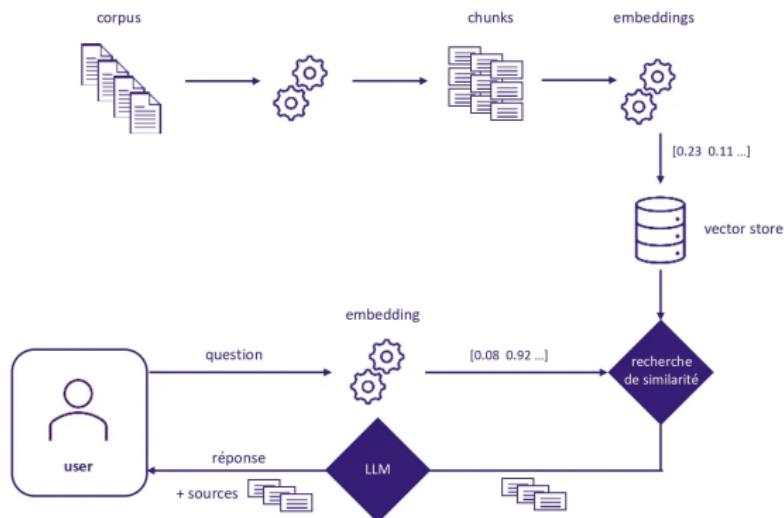
- **Hallucinations** : Les modèles peuvent générer des informations incorrectes ou inventées, rendant difficile la validation des réponses.
- **Mémoire limitée** : Ils ne conservent pas le contexte au-delà d'une fenêtre de tokens, ce qui peut entraîner des incohérences dans les échanges prolongés.
- **Biais et manque de contrôle** : Apprentissage sur des données biaisées, risque de réponses inappropriées ou non alignées avec les attentes utilisateur.
- **Problèmes d'explicabilité** : Difficile de comprendre comment une réponse est générée et sur quelles sources elle s'appuie.
- **Coût computationnel élevé** : Besoin en calcul et mémoire important, complexité du déploiement en production.

Qu'est-ce qu'un Retrieval-Augmented Generation (RAG) ?

- **Définition** : Un modèle RAG combine la recherche d'information (*retrieval*) avec la génération de texte pour améliorer la précision et la pertinence des réponses.
- **Principe** : Le modèle récupère d'abord des documents pertinents à partir d'une base de données, puis les utilise pour générer une réponse enrichie.
- **Avantages** : Réduction des hallucinations, accès à des connaissances actualisées et contrôlées, amélioration de l'explicabilité des réponses.
- **Cas d'usage** : Chatbots spécialisés, assistants de recherche, génération de résumés, support client automatisé.

Principes du RAG

RAG basique



Stratégies de chunking

- **Définition** : Le chunking consiste à segmenter un document en morceaux (chunks) pour optimiser la récupération et la génération d'informations dans un système RAG.
- **Principales stratégies** :
 - **Fixed-size chunking** : découpage en segments de longueur fixe (ex : 512 tokens), simple mais peut fragmenter le contexte.
 - **Overlapping chunks** : chevauchement entre les segments pour conserver le contexte et éviter la perte d'informations critiques.
 - **Semantic chunking** : segmentation basée sur la structure du texte (paragraphes, titres, sections) ou l'analyse sémantique.
 - **Recursive chunking** : division progressive en fonction de la granularité des informations pertinentes.
- **Exemples d'outils** : LangChain, NLTK, SpaCy, tiktoken.
- **Avantages** : Améliore la récupération d'informations, réduit le bruit et optimise l'usage des modèles génératifs.

Bases de données vectorielles

- **Définition** : Stockent et indexent des représentations numériques (embeddings) pour effectuer des recherches sémantiques efficaces.
- **Principe** : Chaque document est transformé en un vecteur dense dans un espace de grande dimension. La recherche repose sur la similarité (cosinus, ANN, etc.) plutôt que sur des mots-clés.



Approches avancées du RAG

- **Reranking** : Amélioration de la sélection des documents récupérés en utilisant un modèle plus puissant (ex : cross-encoder, Cohere Rerank, ColBERT).
- **Fine-tuning du retrieveur** : Optimisation de l'indexation et du scoring des documents avec un modèle entraîné spécifiquement sur le domaine d'application.
- **Graph RAG** : Utilisation d'un graphe de connaissances pour structurer les relations entre documents et améliorer la récupération d'information.
- **Agent RAG** : Intégration d'agents conversationnels capables d'exécuter des actions (recherches multiples, vérification des réponses) avant de générer une réponse finale.

Reranking dans le RAG

- **Définition** : Le reranking consiste à réordonner les documents récupérés pour améliorer la pertinence des résultats avant de les passer au modèle génératif.
- **Méthodes principales** :
 - **BM25 + Reranker** : recherche lexicale suivie d'un modèle neuronal qui affine le classement des documents.
 - **Cross-encoder** : modèle BERT-like qui évalue chaque document en prenant en compte à la fois la requête et le contenu.
 - **Dense Reranking** : ajuste les scores des documents en utilisant des embeddings appris sur des données spécifiques.
- **Exemples d'outils** : Cohere Rerank, ColBERT, MonoT5, Rank-BERT.
- **Avantages** : Meilleure précision des résultats, réduction du bruit, amélioration des performances du modèle génératif.

Fine-tuning dans le RAG

- **Définition** : Le fine-tuning consiste à ajuster un modèle pré-entraîné sur un ensemble de données spécifique pour améliorer ses performances dans un domaine précis.
- **Types de fine-tuning** :
 - **Fine-tuning du retrieveur** : amélioration de la récupération des documents en entraînant un modèle dense (ex : Contriever, ColBERT) sur un corpus spécialisé.
 - **Fine-tuning du modèle génératif** : adaptation d'un LLM pour mieux exploiter les documents récupérés et générer des réponses plus pertinentes.
 - **Fine-tuning hybride** : entraînement simultané du retrieveur et du modèle génératif pour une synergie optimale.
- **Avantages** : L'avantage principal est une meilleure spécialisation du modèle.

Graph RAG

- **Définition** : Le Graph RAG utilise une structure de graphe pour organiser et récupérer l'information de manière plus contextuelle et structurée.
- **Principes clés :**
 - **Relations entre documents** : les nœuds représentent des documents, concepts ou entités, et les arêtes définissent leurs connexions.
 - **Recherche basée sur la connectivité** : la récupération d'informations se fait en explorant les relations entre les nœuds pour trouver les réponses les plus pertinentes.
 - **Augmentation du contexte** : le modèle peut utiliser des chemins sémantiques dans le graphe pour enrichir le prompt avec des données plus cohérentes.
- **Exemples d'outils** : Neo4j, NetworkX, Weaviate.
- **Avantages** : Meilleure structuration des connaissances, récupération d'information plus précise, réduction des erreurs contextuelles.

Agent RAG

- **Définition** : L'Agent RAG intègre un agent autonome qui orchestre la récupération et l'exploitation des documents pour affiner la génération de réponses.
- **Principes clés :**
 - **Multi-recherches adaptatives** : l'agent effectue plusieurs requêtes en fonction du contexte et ajuste dynamiquement le retrieveur.
 - **Vérification et correction** : l'agent peut analyser la réponse générée, détecter d'éventuelles erreurs et reformuler la requête si nécessaire.
 - **Actions spécifiques** : l'agent peut interagir avec des bases de données, des API externes ou exécuter du code pour enrichir la réponse finale.
- **Exemples d'outils** : LangChain Agents, CrewAI, etc.
- **Avantages** : Plus grande autonomie, récupération d'informations plus dynamique, réduction des erreurs génératives.

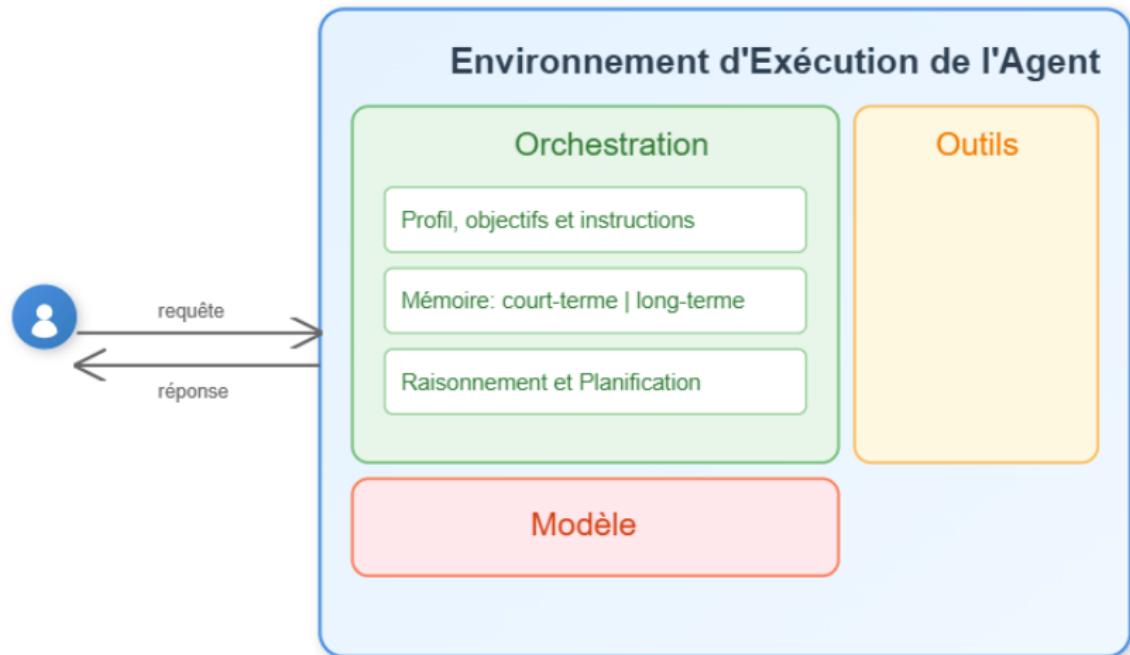
Agents basés sur les LLM

Introduction aux Agents

- **Définition** : Les agents basés sur les LLMs sont des systèmes capables de prendre des actions de manière autonome en utilisant des outils mis à leur disposition.
- **Pourquoi les agents ?**

Les modèles de langage seuls sont limités par leurs données d'entraînement.
Les agents étendent ces capacités en utilisant des outils externes.
- **Objectif de la présentation :**
 - Expliquer le rôle des agents en IA.
 - Présenter leur architecture et leurs composants.
 - Explorer les cas d'utilisation et les techniques modernes d'orchestration.

Principe d'un Agent



Chain-of-Thought (CoT)

- **Qu'est-ce que CoT ?**

Une technique de raisonnement permettant aux modèles de langage de diviser une tâche complexe en étapes intermédiaires.

- **Fonctionnement :**

- Le modèle génère une séquence de raisonnements explicites avant de fournir une réponse.
- Permet d'améliorer la précision et la transparence des décisions prises par un agent.

- **Applications :**

- Résolution de problèmes mathématiques complexes.
- Compréhension de texte avec inférences logiques.
- Réponses plus précises et cohérentes dans les dialogues IA.

ReAct : Reasoning + Acting

Définition : ReAct est une approche qui combine le raisonnement et l'action pour améliorer la prise de décision des modèles de langage.

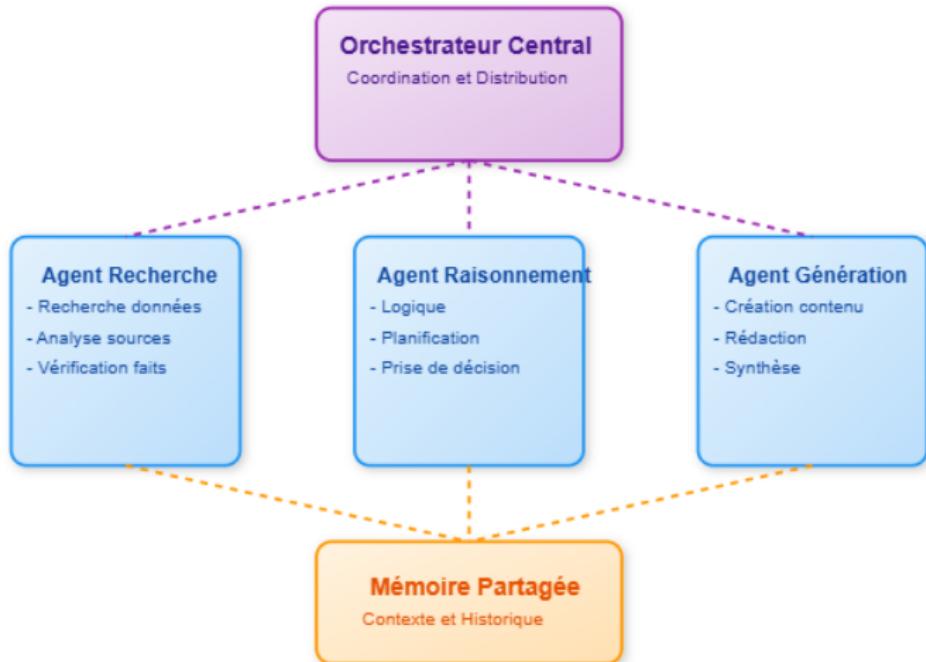
Principe :

- Le modèle observe une entrée utilisateur et génère une pensée explicite (**reasoning**).
- Il sélectionne une action (**acting**) basée sur cette pensée et l'exécute.
- Une observation est obtenue en réponse à l'action, servant de contexte pour la prochaine itération.
- Ce cycle de raisonnement et d'action continue jusqu'à atteindre une réponse satisfaisante.

Avantages :

- Réduction des hallucinations en intégrant des informations en temps réel.
- Amélioration de l'interprétabilité grâce aux traces de raisonnement.
- Meilleure capacité d'adaptation dans des environnements dynamiques.

Systèmes multiagents LLMs



Alignement et réentraînement des LLMs

Alignment des LLMs : Enjeux et défis

Qu'est-ce que l'alignment ?

- Adapter le comportement du modèle aux intentions humaines et aux valeurs sociétales.
- Réduire les biais, assurer la sécurité et améliorer l'utilité.

Pourquoi est-ce un défi ?

- Les LLMs optimisent une fonction de vraisemblance qui ne capture pas toujours les attentes humaines.
- Risques d'hallucinations et de comportements non désirés.
- Contradiction entre maximisation de l'utilité et minimisation des risques.

Méthodes d'alignment

- **Fine-tuning supervisé (SFT)** : entraînement sur des exemples annotés.
- **RLHF (Reinforcement Learning from Human Feedback)** : ajustement basé sur des retours humains.
- **Filtrage et post-traitement** : exclusion des sorties problématiques.

Supervised Fine-Tuning (SFT) : Principes

Définition

- Affinement d'un modèle pré-entraîné via un apprentissage supervisé sur un corpus annoté spécifique.
- Utilise la descente de gradient standard avec rétropropagation pour minimiser une fonction de perte (ex : cross-entropy pour la classification, log-likelihood pour le NLP).

Motivation

- Réduction du décalage entre les distributions de données d'entraînement et d'inférence (*domain adaptation*).
- Incorporation de connaissances expertes via des annotations humaines de qualité.
- Amélioration des performances sur des tâches spécifiques par rapport aux modèles génériques.

Pipeline du Supervised Fine-Tuning

1. Prétraitement des données

- Nettoyage et normalisation des données textuelles.
- Tokenization avec un vocabulaire adapté (*WordPiece*, *BPE*, *SentencePiece*).
- Gestion des déséquilibres de classes dans le dataset.

2. Configuration de l'entraînement

- Choix d'un optimiseur adapté (**AdamW**, etc.).
- Utilisation de techniques de type Parameter-Efficient Fine-Tuning pour limiter le nombre de paramètres à ajuster.
- Régularisation : **dropout**, **weight decay**, **gradient clipping**.

3. Évaluation et validation

- Utilisation de **métriques spécifiques** : *accuracy*, *perplexity*, *BLEU*, *ROUGE*, *F1-score*.
- Détection d'overfitting : **early stopping**, **validation croisée**.

Défis et limitations du Supervised Fine-Tuning

Problèmes liés aux données

- **Biais des données** : un corpus non représentatif entraîne des préjugés dans les générations.
- **Taille du dataset** : un SFT efficace nécessite des millions d'exemples annotés.
- **Distribution shift** : le modèle peut être inefficace si la distribution en production diffère trop.

Problèmes liés à l'optimisation

- **Oublis catastrophiques** : risque d'effacement des connaissances pré-entraînées.
- **Instabilité du fine-tuning** : gradients explosifs, mode collapse.
- **Coût computationnel** : nécessité d'accélérateurs matériels (*TPUs, GPUs A100/H100*).

Quantization des LLMs : Principes

Pourquoi quantifier un modèle ?

- Réduction de la consommation mémoire et des besoins en calcul.
- Accélération de l'inférence sur CPU et GPU.
- Permet l'exécution sur du matériel contraint (ex. : edge computing).

Principe

- Représentation des poids et activations avec une précision plus faible (ex : FP32 → INT8).
- Équilibrage entre compression du modèle et préservation des performances.

Types de quantization

- **Post-Training Quantization (PTQ)** : Quantification après l'entraînement.
- **Quantization-Aware Training (QAT)** : Le modèle est entraîné en intégrant la quantization.

Techniques de quantization

Post-Training Quantization (PTQ)

- Conversion des poids après l'entraînement.
- Méthodes :
 - Static quantization : calibration des poids avec un jeu de données de calibration.
 - Dynamic quantization : quantification des activations à la volée.
- Avantage : Facile à appliquer, rapide.
- Inconvénient : Peut dégrader les performances si la calibration est mal faite.

Quantization-Aware Training (QAT)

- Intègre la quantization dans le processus d'entraînement.
- Simule les erreurs dues à la quantization et ajuste les poids en conséquence.
- Avantage : Meilleure robustesse.
- Inconvénient : Entraînement plus coûteux en calcul.

Défis de la quantization

- Perte de précision : certains modèles sont plus sensibles à la quantization que d'autres.
- Choix du niveau de quantization : INT8, INT4, FP8, BF16.
- Compatibilité matérielle : certains formats sont optimisés pour des architectures spécifiques (TensorRT, XNNPACK).

Outils et bibliothèques

- Hugging Face Transformers : supporte quantization avec bitsandbytes.
- TensorRT, ONNX Runtime : optimisation pour l'inférence.
- Intel Neural Compressor : quantization pour CPU.
- GPTQ (GPT Quantization) : quantization optimisée pour LLMs.

Parameter-Efficient Fine-Tuning (PEFT)

Pourquoi PEFT ?

- Le fine-tuning complet des LLMs est coûteux en calcul et en mémoire.
- PEFT permet d'adapter un modèle en ajustant un nombre réduit de paramètres.
- Réduit le besoin en ressources tout en maintenant des performances compétitives.

Principe

- Congélation des poids du modèle pré-entraîné.
- Ajout de couches ou modifications légères sur certaines parties du réseau.
- Optimisation de seulement quelques millions de paramètres au lieu de milliards.

Applications

- Adaptation de LLMs à des domaines spécifiques avec peu de données.
- Fine-tuning efficace sur des GPUs à mémoire limitée.
- Déploiement plus léger sans besoin de recharger l'ensemble des poids modifiés.

Low-Rank Adaptation (LoRA) : Introduction

Pourquoi LoRA ?

- Fine-tuner un LLM de manière efficace sans modifier tous ses poids.
- Réduction de la consommation mémoire et du nombre de paramètres à optimiser.
- Permet d'utiliser des GPU avec une VRAM limitée.

Principe

- Gèle les poids du modèle de base.
- Introduit des matrices de bas-rang pour ajuster les poids de certaines couches.
- Optimise uniquement ces matrices additionnelles, réduisant ainsi la charge de calcul.

Avantages

- Réduction drastique du nombre de paramètres ajustés (jusqu'à 10 000x moins que le fine-tuning complet).
- Compatible avec les modèles pré-entraînés sans perturber leurs capacités initiales.
- Facilement combinable avec d'autres techniques comme la quantization.

Fonctionnement de LoRA

Modification des couches linéaires

- Au lieu de modifier la matrice de poids W , LoRA apprend une correction en bas-rang.
- Décomposition de la mise à jour en deux matrices A et B de rang réduit :

$$W' = W + BA \tag{6}$$

- $A \in \mathbb{R}^{d \times r}$ et $B \in \mathbb{R}^{r \times d}$ avec $r \ll d$.

Pourquoi une approximation en bas-rang ?

- Les mises à jour du fine-tuning complet sont souvent redondantes.
- Une mise à jour en basse dimension capture l'essentiel des ajustements nécessaires.
- Conserve la robustesse du modèle pré-entraîné tout en ajustant ses sorties.

Implémentation de LoRA

Choix du rang r

- Un r trop petit entraîne une perte de capacité d'adaptation.
- Un r trop grand augmente la complexité sans gain significatif.
- Valeurs typiques : $r \in [4, 32]$ selon la taille du modèle et la tâche.

Compatibilité avec les frameworks

- Hugging Face 'peft' : implémentation standard pour les modèles Transformers.
- Support natif dans PyTorch et TensorFlow avec adaptation des couches linéaires.
- Intégration avec des méthodes de quantization pour minimiser l'empreinte mémoire.

Pourquoi RLHF ?

- Le fine-tuning supervisé (SFT) ne capture pas toujours les subtilités des préférences humaines.
- RLHF permet d'ajuster un modèle LLM en optimisant ses réponses en fonction de retours humains.
- Objectif : aligner le modèle avec les valeurs humaines et éviter les comportements indésirables.

Pipeline RLHF

- ① Fine-tuning supervisé initial (SFT) sur un corpus annoté.
- ② Recueil de préférences humaines via des paires de réponses.
- ③ Entraînement d'un **Reward Model** (RM) pour scorer les réponses.
- ④ Optimisation du LLM avec PPO en utilisant le Reward Model.

Entraînement du Reward Model

Objectif

- Modéliser la préférence humaine sous forme d'une fonction de récompense $R(x, y)$.
- Entraîné sur des paires de réponses (y_1, y_2) où un annotateur préfère y_1 à y_2 .

Fonction de perte

$$\mathcal{L}_{RM} = - \sum_{(y_1, y_2)} \log \sigma(R(x, y_1) - R(x, y_2)) \quad (7)$$

où σ est la fonction sigmoïde.

Enjeux

- Nécessite un grand volume de feedback humain.
- Risque de surapprentissage aux biais des annotateurs.

Optimisation avec Proximal Policy Optimization (PPO)

Pourquoi PPO ?

- Algorithme de renforcement efficace pour fine-tuner un LLM sans changer drastiquement sa distribution de sortie.
- Contraint les mises à jour du modèle pour éviter l'instabilité.

Fonction de perte PPO

$$\mathcal{L}_{PPO} = \mathbb{E} [\min(r_t A_t, \text{clip}(r_t, 1 - \epsilon, 1 + \epsilon) A_t)] \quad (8)$$

où :

- $r_t = \frac{\pi_\theta(y_t|x)}{\pi_{\theta_{\text{old}}}(y_t|x)}$ est le ratio d'importance.
- A_t est l'avantage estimé.
- ϵ est un hyperparamètre de clip (ex: 0.2).

Avantages

- Convergence plus stable que d'autres algorithmes de RL.
- Permet des mises à jour régulières du modèle sans dériver trop fortement.

Défis et risques du RLHF

Limitations

- Collecte des données humaines coûteuse et sujette à des biais.
- Risque d'optimisation excessive : le modèle peut sur-apprendre la fonction de récompense.
- Contradictions dans les préférences humaines entraînant des incohérences.

Stratégies d'atténuation

- Mélange de RLHF et de fine-tuning supervisé.
- Diversification des annotateurs et des scénarios d'évaluation.
- Introduction d'aléa dans les récompenses pour éviter l'overfitting.

Diverses considérations sur les LLMs

Répartition des données d'entraînement des LLMs

- **Corpus en anglais (1,189B tokens)**

- Web crawl : 889B (74.8%)
- Divers : 109B (9.2%)
- News : 94B (7.9%)
- Conversations : 59B (5.0%)
- Livres : 35B (2.9%)
- Scientifique : 33B (2.8%)

- **Corpus multilingue (596B tokens)**

- Web crawl : 540B (90.6%)
- Corpus parallèle : 56B (9.4%)

- **Corpus de code (212B tokens)**

- The Stack v1.2 : 212B (100%)

Sources principales : Common Crawl, ArXiv, Stack Exchange, Reddit, Wikipedia, The Stack v1.2

Évaluation des LLMs : Benchmarks standards

1. Compréhension et raisonnement général

- **MMLU (Massive Multitask Language Understanding)** : 57 tâches académiques (histoire, sciences, mathématiques, etc.).
- **HellaSwag** : Raisonnement sur des suites de phrases.
- **Winogrande** : Résolution d'ambiguités syntaxiques (Winograd Schema).
- **PIQA (Physical IQA)** : Raisonnement physique et compréhension du monde réel.
- **ARC (AI2 Reasoning Challenge)** : Questions à choix multiples de niveau scolaire.

2. Capacité de modélisation du langage

- **LAMBADA** : Prédiction du dernier mot dans un contexte long.
- **Wikitext-103** : Modélisation de texte sur un corpus de haute qualité.
- **Perplexité (PPL)** : Mesure la cohérence et la fluidité du modèle.

Évaluation des LLMs : Benchmarks spécialisés

3. Génération et manipulation du code

- **HumanEval** : Génération de code à partir de descriptions en langage naturel.
- **MBPP (Mostly Basic Python Problems)** : Résolution de problèmes en Python.
- **MultiPL-E** : Évaluation sur plusieurs langages de programmation.

4. Compréhension et traitement du langage naturel

- **SuperGLUE** : Tests avancés de compréhension du langage naturel.
- **DROP** : Raisonnement numérique et inférence logique.
- **BoolQ** : Questions-réponses sur des faits simples.

5. Alignement et sécurité

- **TruthfulQA** : Capacité du modèle à générer des réponses factuelles.
- **RealToxicityPrompts** : Détection et génération de texte toxique.
- **BiasBench** : Analyse des biais algorithmiques dans les LLMs.

Benchmarks avancés pour l'évaluation des LLMs

1. Raisonnement avancé et intelligence artificielle générale

- **ARC-AGI (Abstraction and Reasoning Corpus - Artificial General Intelligence)**

- Teste la capacité des modèles à généraliser des règles abstraites.
- Exige des compétences en raisonnement visuel et logique.

2. Résolution de problèmes algorithmiques et programmation compétitive

- **CodeForces**

- Évalue les LLMs sur des problèmes de programmation compétitive.
- Utilisé pour mesurer la capacité des modèles à générer des solutions optimisées.

- **APPS (Automated Programming Progress Standard)**

- Benchmark avec des exercices de programmation allant du niveau débutant à expert.
- Contient des problèmes issus d'examens et de concours de code.

3. Évaluation des compétences en ingénierie logicielle

- **SWE-Bench (Software Engineering Benchmark)**

- Conçu pour évaluer les LLMs sur des tâches complexes de développement logiciel.
- Mesure la capacité à comprendre et modifier du code existant dans un contexte industriel.

Benchmarks avancés : Raisonnement général et maths

1. Évaluation des connaissances générales et du raisonnement avancé

- **GPQA (General-Purpose Question Answering)**

- Teste la capacité des modèles à répondre à des questions ouvertes de culture générale.
- Évalue la compréhension contextuelle et le raisonnement factuel.

2. Benchmarks spécialisés en mathématiques

- **AIME (American Invitational Mathematics Examination)**

- Contient des problèmes mathématiques avancés de niveau olympique.
- Évalue les capacités de raisonnement symbolique et de résolution algébrique.

- **MATH (Mathematical Aptitude Test for Heuristics)**

- Benchmark dédié à l'évaluation des LLMs en mathématiques avancées.
- Comprend des problèmes issus de compétitions mathématiques.

3. Applications en raisonnement scientifique et formel

- **GSM8K (Grade School Math 8K)**

- Évalue les capacités en mathématiques élémentaires et en raisonnement numérique.
- Teste la compréhension des opérations de base et des problèmes en plusieurs étapes.

- **MMLU-Math (Massive Multitask Language Understanding - Math subset)**

1. Modes de déploiement

- **Cloud-based (SaaS)**

- Accès via des APIs (OpenAI, Anthropic, Mistral, Google AI).
- Scalabilité et maintenance assurées par le fournisseur.
- Coût basé sur l'usage (pay-per-token, abonnement).

- **On-premise (Self-hosted)**

- Hébergement sur serveurs dédiés ou clusters privés.
- Contrôle total sur les données et la sécurité.
- Nécessite une infrastructure GPU optimisée.

- **Edge AI et Inference locale**

- Déploiement sur appareils mobiles, microcontrôleurs (ex : Phi-3, Gemma).
- Optimisation via quantization et distillation.

2. Stratégies pour optimiser le déploiement

- **Quantization (INT8, FP16, GPTQ, AWQ)**

- Réduit la taille du modèle et accélère l'inférence.
- Compatible avec des frameworks comme TensorRT, ONNX Runtime.

- **Distillation**

- Transfert de connaissances vers un modèle plus petit.
- Exemples : DistilBERT, TinyLLaMA.

- **Batching et compilation**

- Utilisation de moteurs comme vLLM pour optimiser l'allocation mémoire.
- Amélioration du débit avec DeepSpeed et TensorRT-LLM.

Merci de votre attention

redha_moulla@yahoo.fr