Introduction au Sequence to Sequence

Sandratra Rasendrasoa

ITI

Plan

1. Qu'est-ce qu'un chatbot ?
2. Réseaux de neurones
3. Représenter une séquence de mots
4. Réseaux récurrents
   - Feed-Forward vs Réseaux récurrents
   - RNN
   - LSTM
5. Encodeur-Décodeur
   - Description
   - Exemple
6. Attention-based Networks
   - Vision d'ensemble
   - Représenter l'entrée
   - Pile d'encodeurs
   - Pile de décodeurs
7. Conclusion

Introduction

Chatbot ?

- Agent conversationnel, ex. Siri, Cortana, Google assistant
- Modalité unique : texte

Types de chatbot

- Spécifique à un domaine
- Dialogue ouvert

Approches existantes

"Retrieval based"

- Basé sur un ensemble de règles (expert du domaine par ex.)
- Réponses fixes
- Meilleure performance si la réponse existe
- pas très extensible

Génératif

- Approches statistiques (markov caché, LMMs)
- Scalable
- Performance dépend de la quantité de données

Basic Example

Propagation avant

- $z^{(2)} = \theta_1 x$, $z^{(3)} = \theta_2 a^{(2)}$
- Exemple de fonction d'activation : $a = \frac{1}{1+\exp^{-z}}$

Fonction de coût

Mesurer la performance du réseau pendant l'entraînement.
Exemple : erreur quadratique $J(\theta) = \frac{1}{2}(y - a^{(3)})^2$

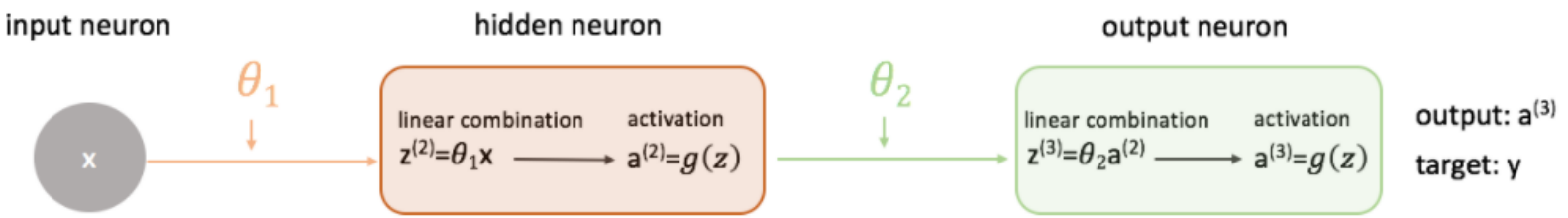


Figure – https://www.jeremyjordan.me/neural-networks-training

Basic Example

Rétropropagation

Comment la performance du modèle évolue-t-elle en fonction de ses paramètres ?

Exemple : $\frac{\delta J(\theta)}{\delta \theta_2} = (\frac{\delta J(\theta)}{\delta a^{(3)}})(\frac{\delta a^{(3)}}{\delta z})(\frac{\delta z}{\delta \theta_2})$

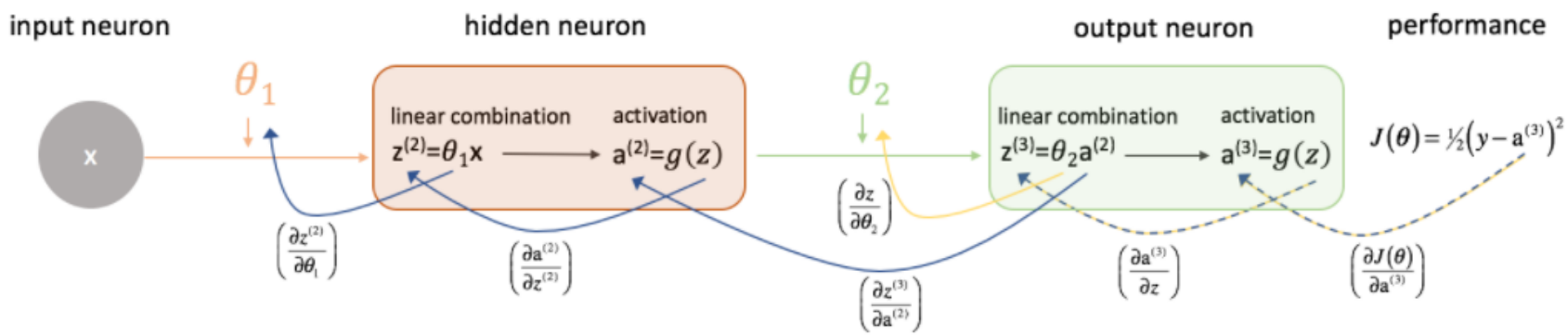
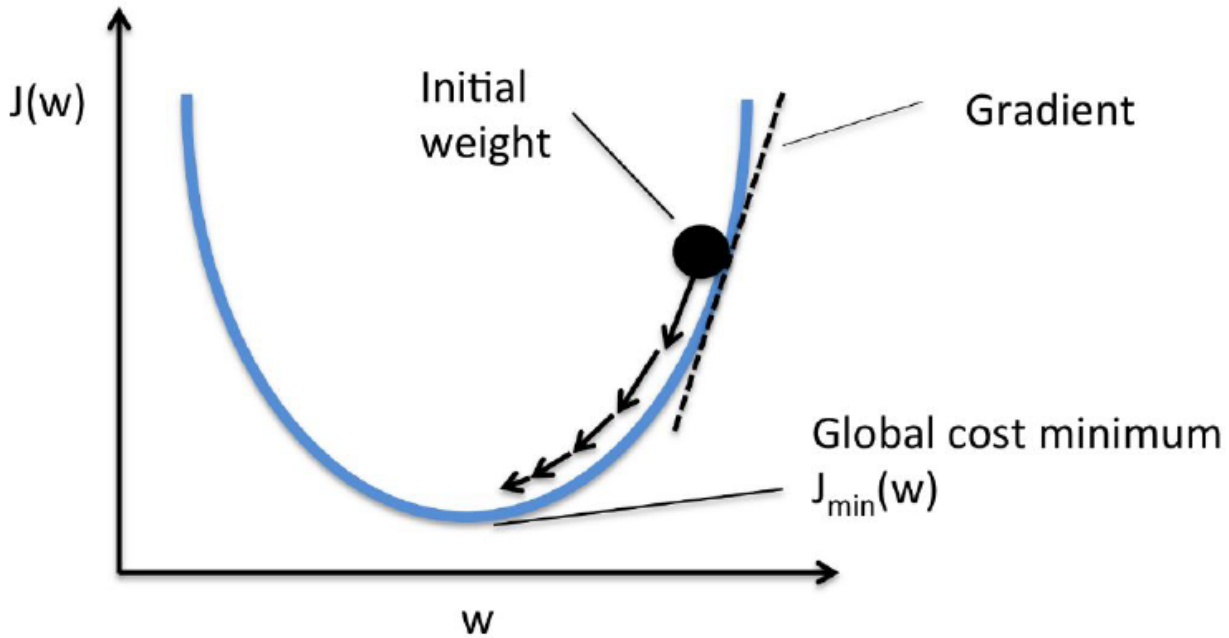


Figure – `https://www.jeremyjordan.me/neural-networks-training`

Descente de gradient

Apprentissage

- Minimiser l'erreur entre la sortie et la cible.
- Le gradient est estimé par rétropropagation. La descente de gradient stochastique est ensuite utilisée pour la formation..



Représenter la modalité textuelle

Problématique

- Algorithmes statistiques travaillent avec des nombres
- Qu'en est-il des données textuelles ?
- Représentation numérique des mots

Approches (non-exhaustif)

- Bag of Words
- N-Grams
- Word embeddings

Bag of Words

Avantages

Ne nécessite pas d'énormes ensembles de données

Inconvénients

- Information contextuelle n'est pas conservée
- Génération de matrices éparses

	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	2	1	1

Figure – Bag of Word (3 documents)

N-Grams

Example

2-grams for la séquence : "I hate dogs and knitting"
"I hate", "hate dogs", "dogs and", "and knitting"

Avantages

- Ne nécessite pas d'énormes ensembles de données
- Information contextuelle est conservée

Inconvénients

- Gaspillage d'espace
- Coût de calcul

Vecteurs de caractéristiques (embeddings)

Avantages

- + performant
- Contexte préservé

Inconvénients

- Quantité massive de données

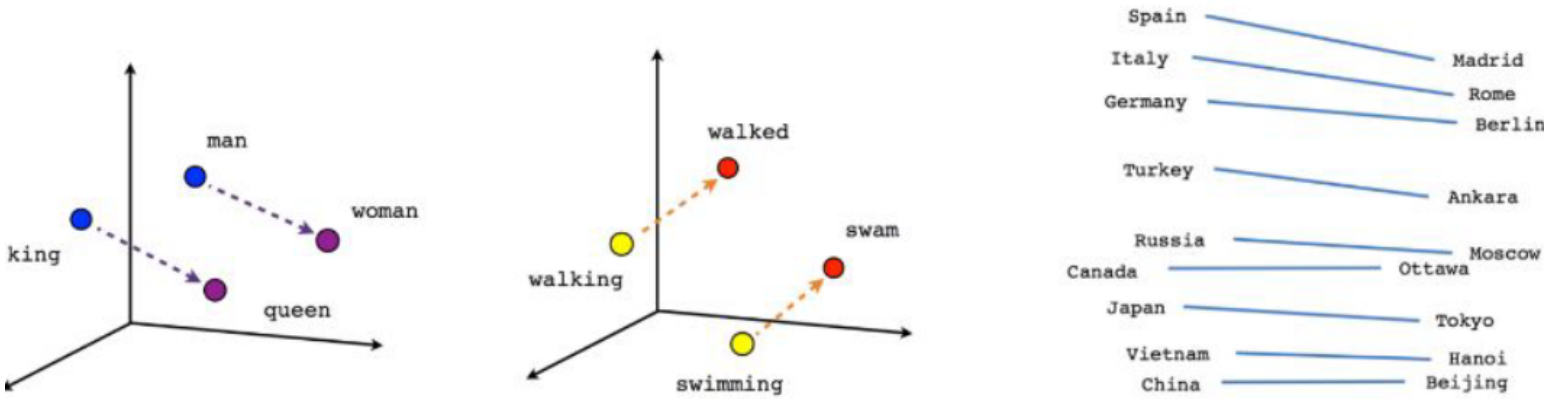


Figure – Examples of word embeddings

Tokenization et Padding

Tokenization

La tokenisation est utilisée pour créer un ensemble de jetons à partir d'un ensemble de données. Elle est utilisée pour construire un vocabulaire, qui est l'ensemble des tokens uniques. Les tokens peuvent être des mots, des sous-mots ou des caractères.

Padding

Hello how are you	---------->	62	12	57	78	0	0	0	0
I called you last night, you were busy.	------->	62	12	57	78	21	57	42	29

Résumé modalité textuelle

A retenir

D'une part, des approches telles que BoW, N-Grams ou TF-IDF (non présentées ici) :

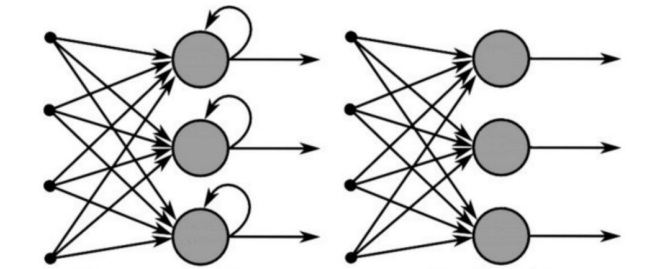
- s'appuient sur le nombre de mots dans une phrase, mais ne fournissent aucune information sémantique ou syntaxique.
- peut créer des matrices très éparses.

Les word embeddings, quant à eux :

- tentent de préserver les informations syntaxiques et sémantiques.
- nécessitent un ensemble de données considérable pour être performants.

Les deux types d'approches tentent de représenter les mots et les documents dans un espace numérique.

Feed-Forward vs Réseaux récurrents



Feed-forward Network

- Pas de mécanisme de feedback : indépendence entre les entrées.
- Entrée de taille fixe

Réseaux récurrents

- La sortie dépend de l'élément précédent dans la séquence.
- Le partage des paramètres au sein du modèle permet une généralisation à différentes longueurs de séquences.

Vue d'ensemble

Soit une séquence d'entrée x t.q. $x = x_1, .., x_N$, où N est la longueur de séquence.

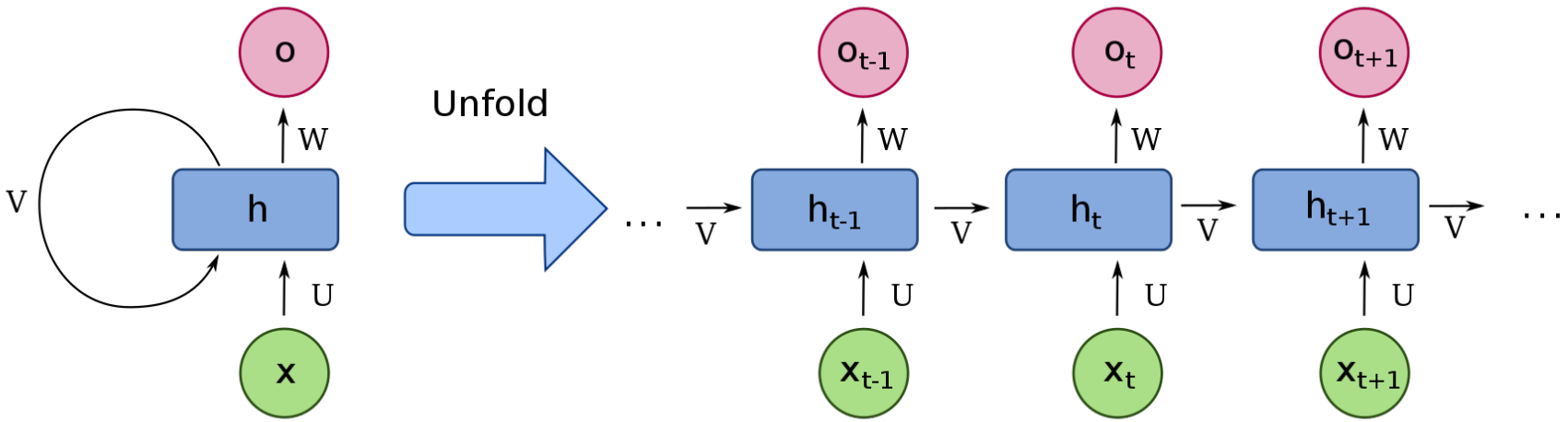


Figure – Example of a recurrent neural network

Problèmes

- Mémoire à court terme
- "Vanishing Gradient"

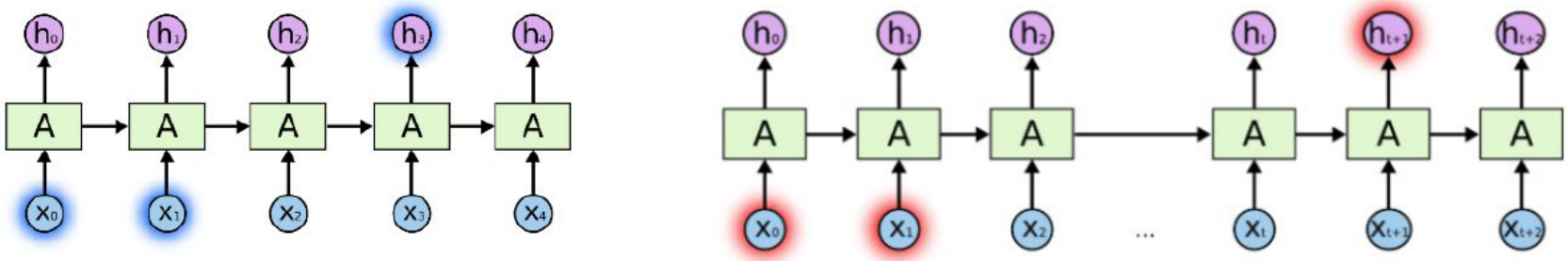
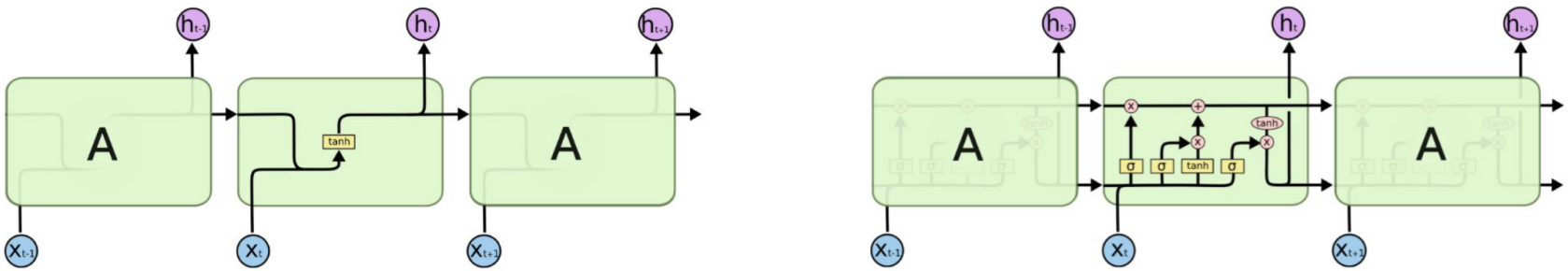


Figure – Short vs Long sequence

Long Short Term Memory

LSTM

"Gating architecture" : solution au problème de disparition du gradient avec des gradients additifs



(a) RNN (b) LSTM

Figure – LSTM vs RNN (Single Input - Single Output)

idées principales

- Reset from memory : on "oublie" une partie de l'état précédent
- Write to memory : on "écrit" en mémoire ce qui est important (état précédent + entrée courante)
- Read from memory : on envoie à la prochaine cellule l'information

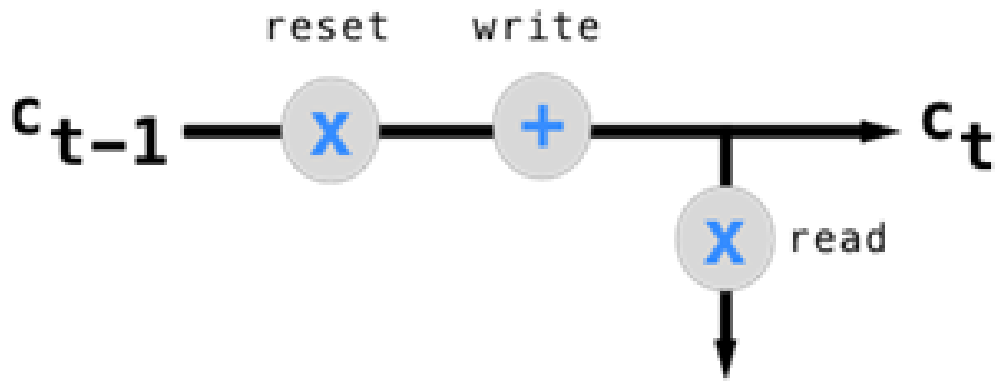


Figure – Core idea behind LSTM

Applications

Tâches diverses

Marquage de séquences, traduction automatique, chatbot, etc.

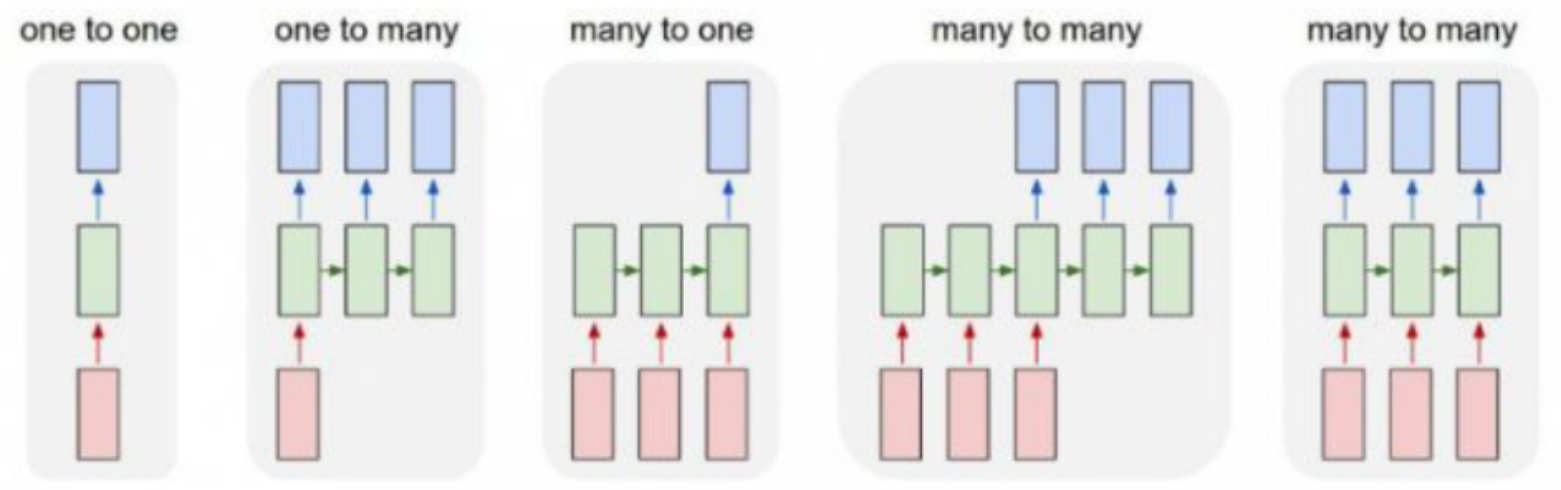


Figure – Variations of the LSTM architecture

Architecture

Many to many

- L'entrée initiale du décodeur est la sortie de l'encodeur.
- L'encodeur lit la séquence d'entrée.
- Le décodeur lit la séquence de sortie cible.

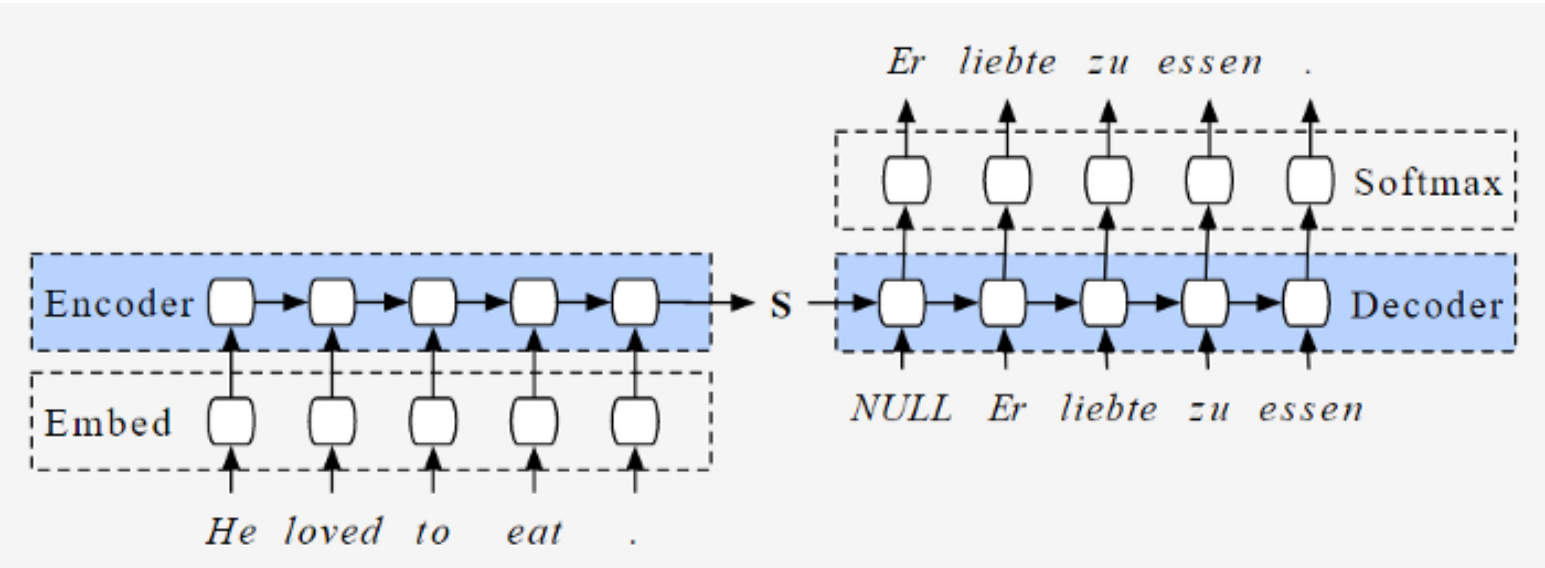


Figure – https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/

Seq2Seq

2 Phases

- Apprentissage
- Prédiction

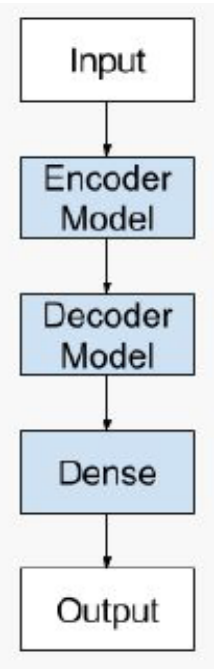
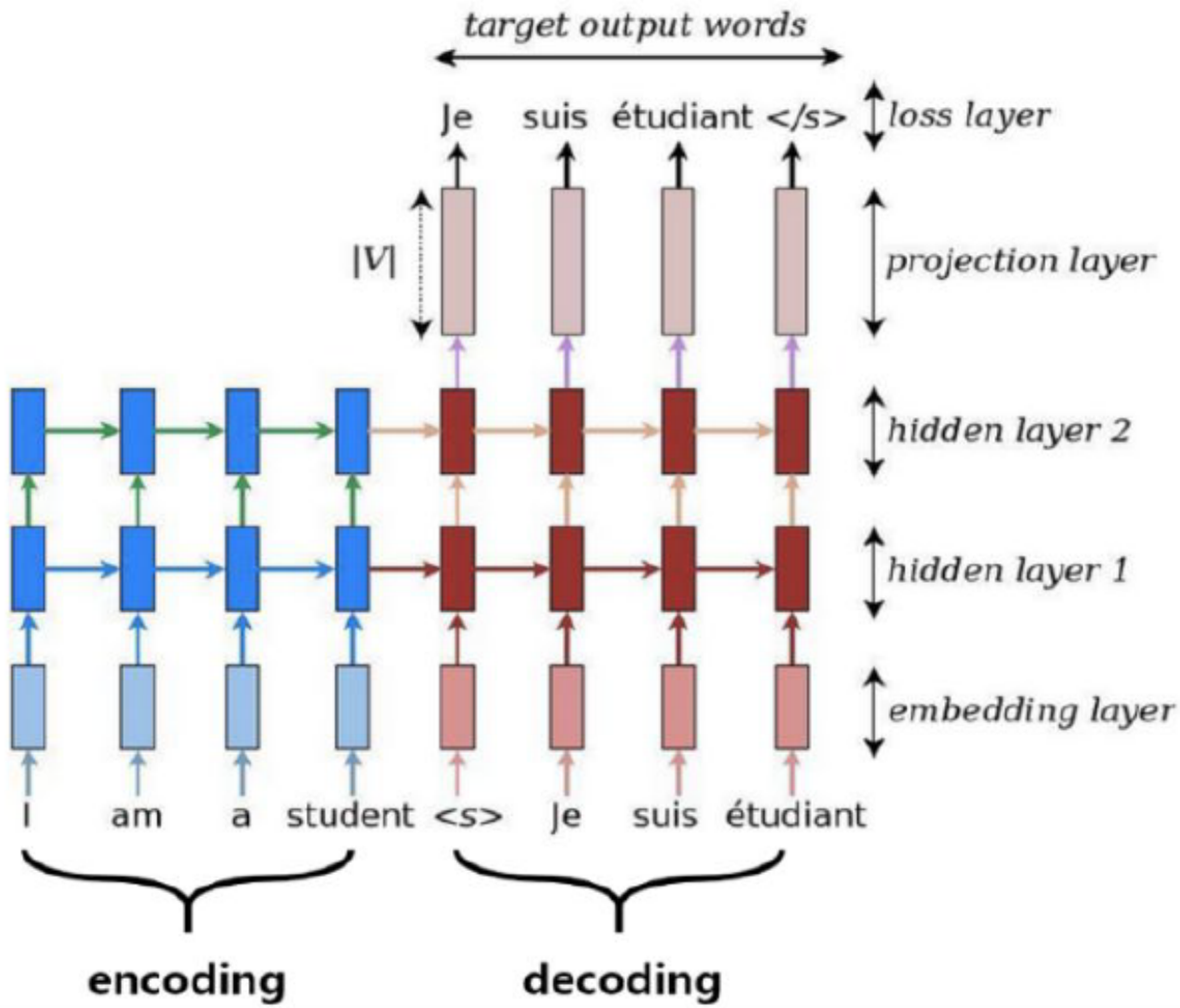


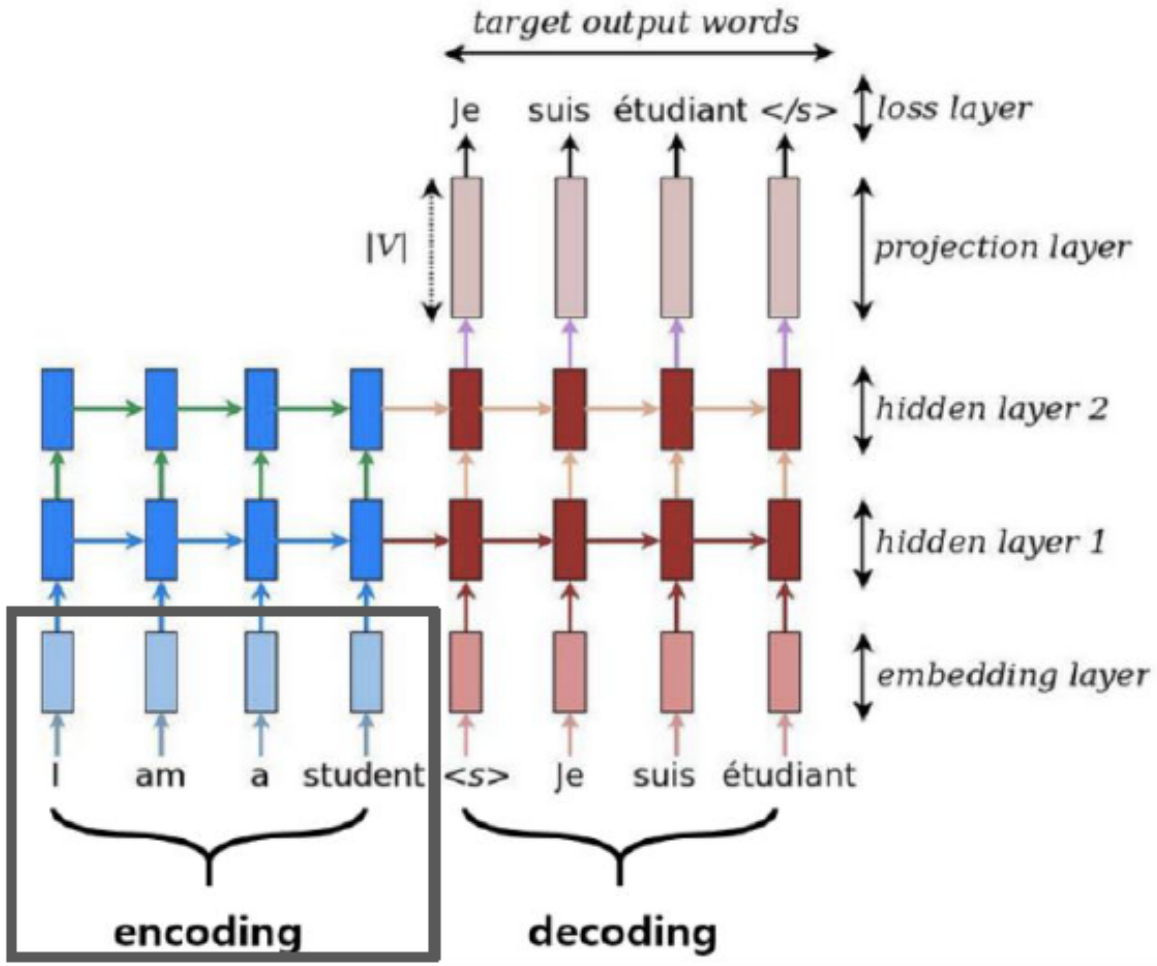
Figure – Seq2Seq summary

Exemple


target output words
Je
suis
étudiant
</s>
loss layer
|V|
projection layer
hidden layer 2
hidden layer 1
embedding layer
I
am
a
student
<s>
Je
suis
étudiant
encoding
decoding

Etape 1

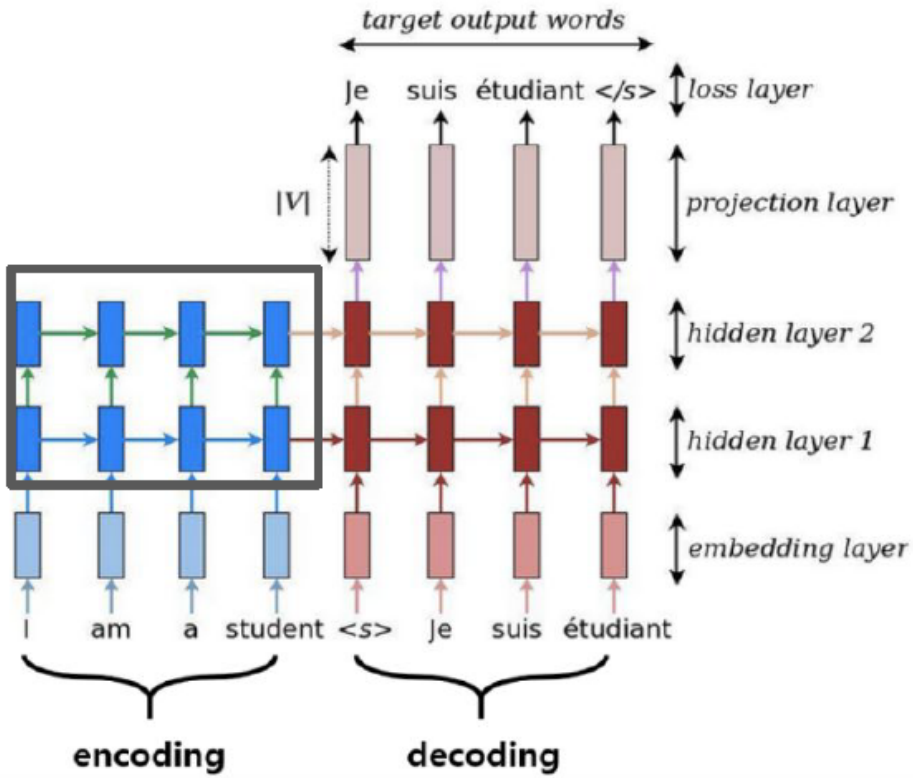
Convertir un texte en vecteurs de caractéristiques



Apprentissage

Etape 2

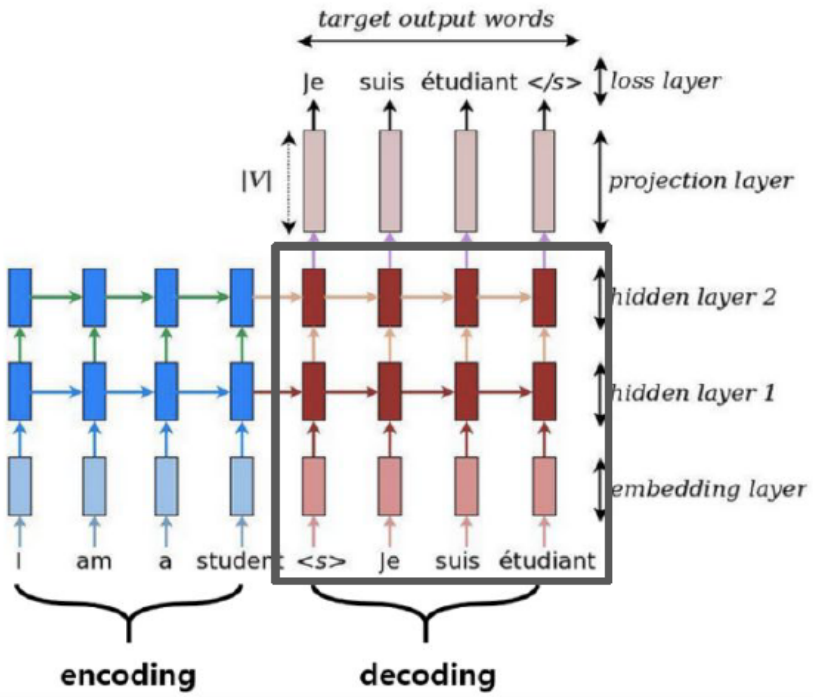
- Les sorties de l'encodeur ne sont pas prises en compte.
- On ne conserve que les états internes.



Apprentissage

Etape 3

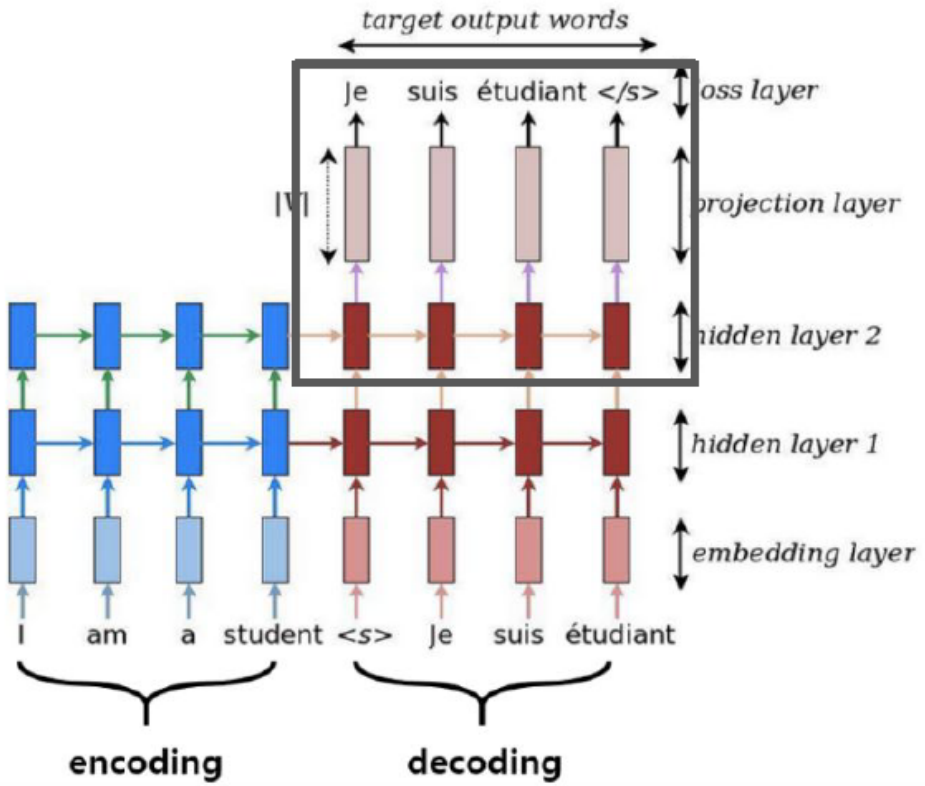
- Les états initiaux du décodeur sont les états internes de l'encodeur.
- Token spécial pour indiquer le début de l'entrée de la séquence cible.



Apprentissage

Etape 4

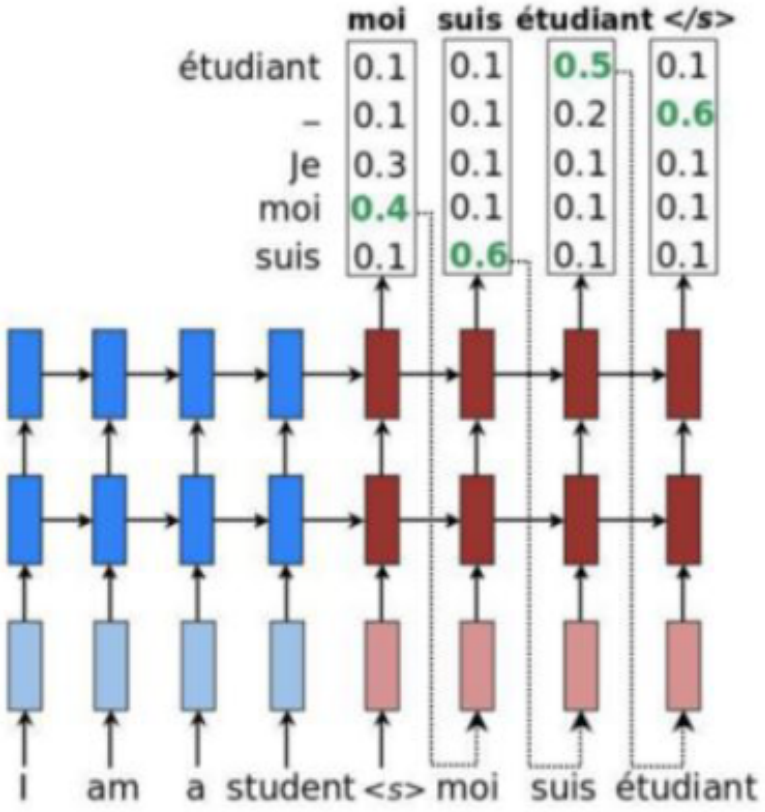
Générer les mots candidats les plus probables



Apprentissage

Etape 5

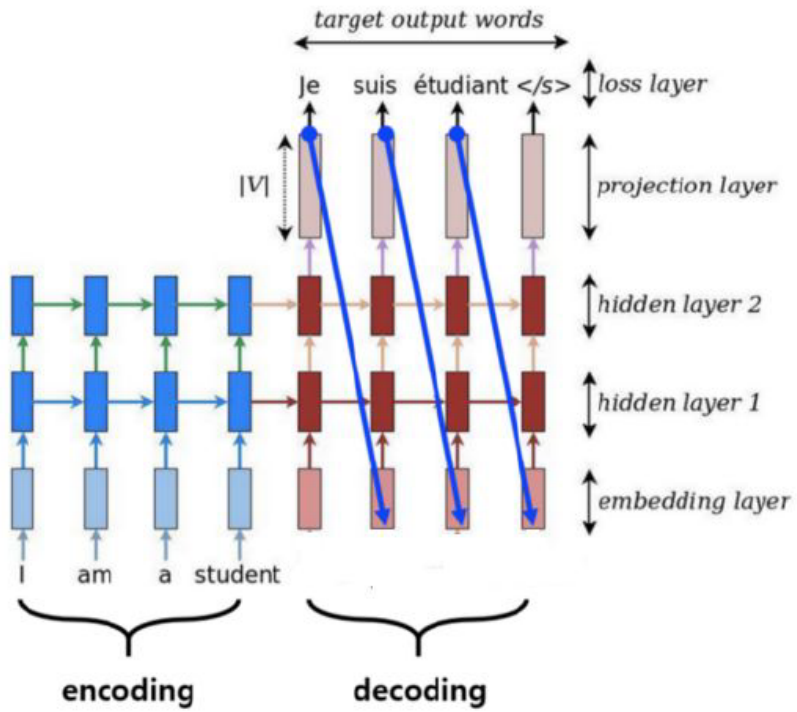
On met à jour les poids en rétropropageant l'erreur.



Prediction

Process

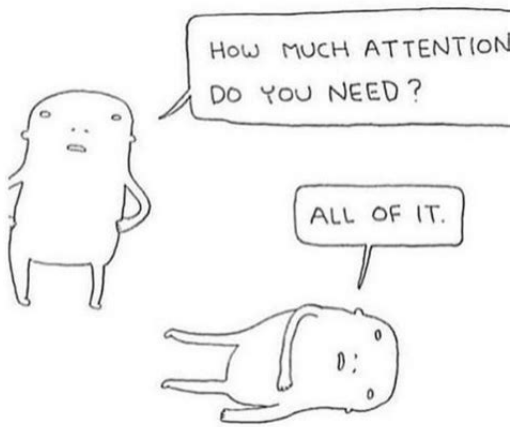
- La sortie de l'unité précédente est utilisée comme entrée dans l'unité suivante.
- L'inférence se termine lorsque le décodeur prédit le token de fin de séquence.



Mécanisme d'attention en TALN (Traitement Automatique du Langage Naturel)

Description

- Les réseaux basés sur l'attention sont à la pointe de la plupart des tâches de TALN (NLP).
- Ils se focalisent sur chaque élément de la séquence en même temps ! Ainsi, ils ne souffrent pas de la perte de mémoire à long terme : ils fonctionnent avec des séquences plus longues et sont plus rapides à entraîner !



Mécanisme d'attention en TALN (Traitement Automatique du Langage Naturel)

Intuition derrière le mécanisme d'attention

En traitant un mot, l'attention permet au modèle de se concentrer sur d'autres mots dans l'entrée qui sont étroitement liés à ce mot.

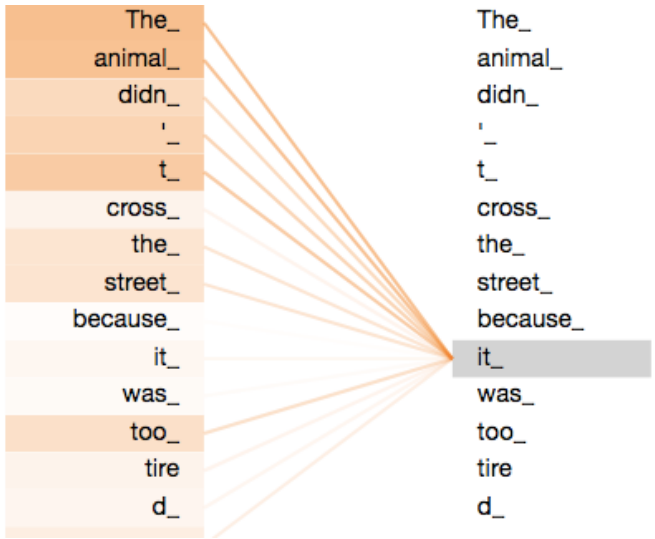


Figure – Illustration du mécanisme d'attention

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.

Contexte

- Il s'agissait d'une contribution dans le domaine de la modélisation de séquences.
- Des expériences ont été réalisées sur des tâches de traduction automatique.
- Les approches classiques pour ces tâches sont basées sur une architecture encodeur-décodeur composée de réseaux récurrents.
- Ils proposent un réseau basé sur l'attention qui permet un entraînement plus rapide en supprimant la récurrence.

Architecture

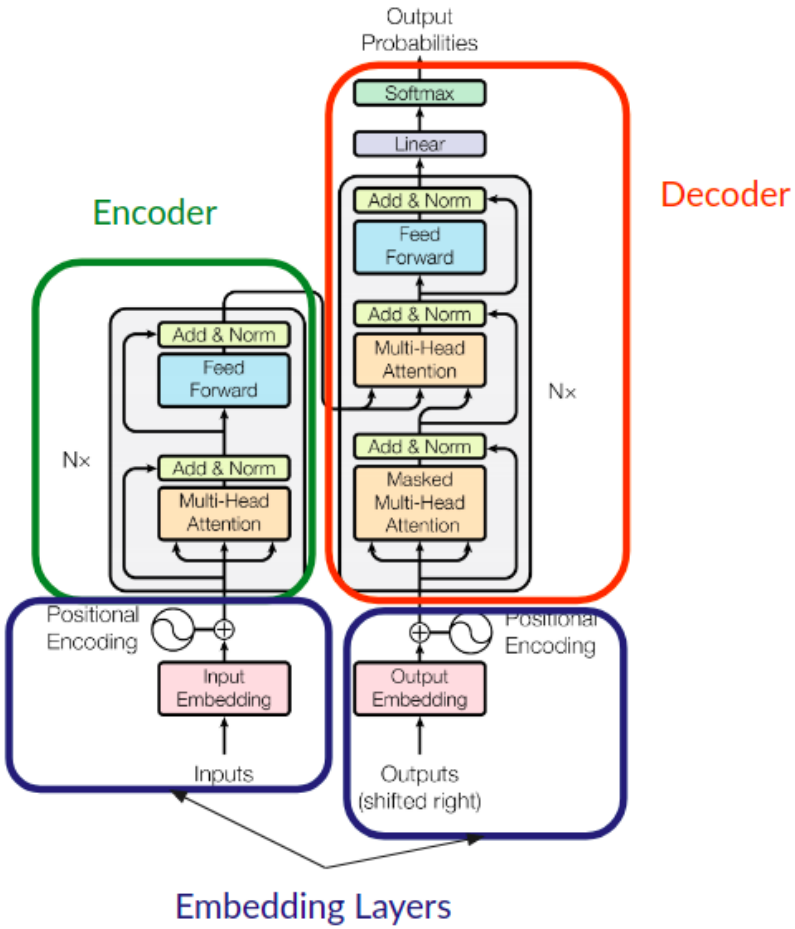


Figure – 3 main components : (1) Embedding Layer (one for each stack), (2) Encoder Stack, (3) Decoder Stack

Embedding Layer

Description

La couche d'incorporation comprend deux éléments :

- Input embeddings : convertit les tokens d'entrée en une représentation apprise : un vecteur de dimension d_{model}.
- Encodage de position : injecte des informations de position dans l'entrée, car les Transformers n'apprennent pas la position de chaque mot implicitement ! (contrairement aux RNN)

0.3

Encodage de position (1)

Pourquoi en avons-nous besoin ?

Chaque mot dans une phrase traverse simultanément une pile de Transformers, de sorte que le modèle n'a pas de sens de l'ordre pour chaque mot. Nous devons ajouter une information à chaque token, d'où l'idée de l'encodage de position !

- encodage unique pour chaque position
- les valeurs doivent être bornées (pour pouvoir généraliser à des séquences plus longues)
- doit être déterministe

Encodage de position (2)

Méthode proposée

Un vecteur de taille d_{model} code l'information concernant une position spécifique dans une phrase.

In this work, we use sine and cosine functions of different frequencies:

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

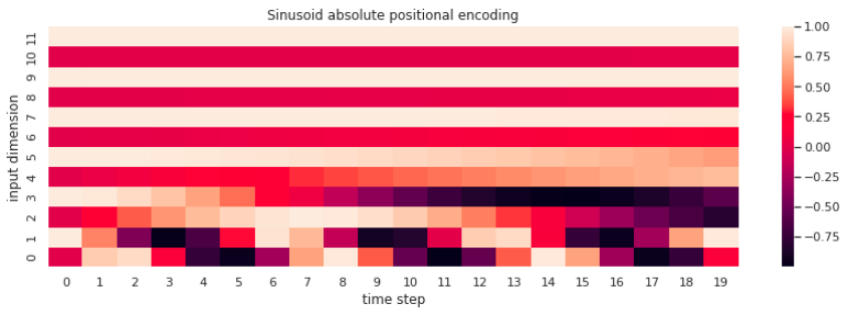
where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

Figure – Extrait de l'article original

Encodage de position (3)

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

(a) Représentation binaire des nombres



(b) Encodage de position avec une séquence de longueur 20 et $d_{model} = 12$

Figure – Intuition pour l'encodage de position

Aperçu

La pile est composée de N couches d'encodeurs, où la première reçoit l'incorporation de la couche d'incorporation, et chaque autre encodeur dans la pile reçoit la sortie de l'encodeur précédent.

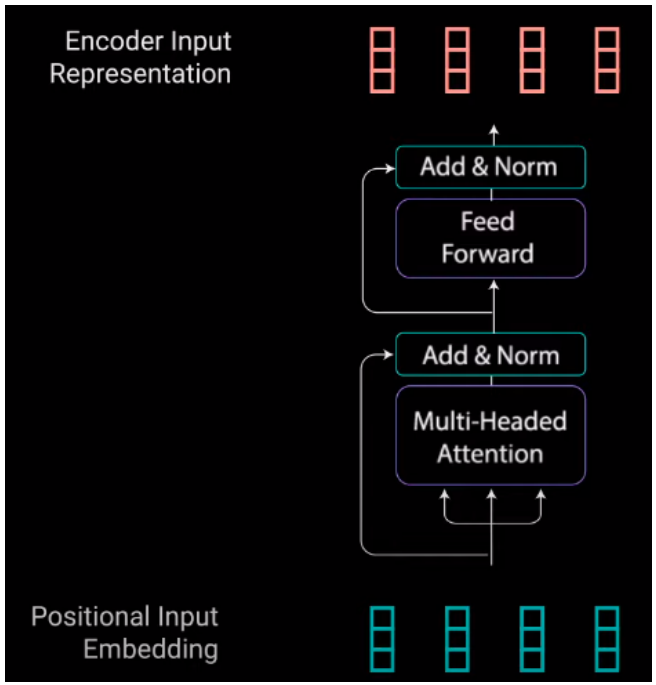
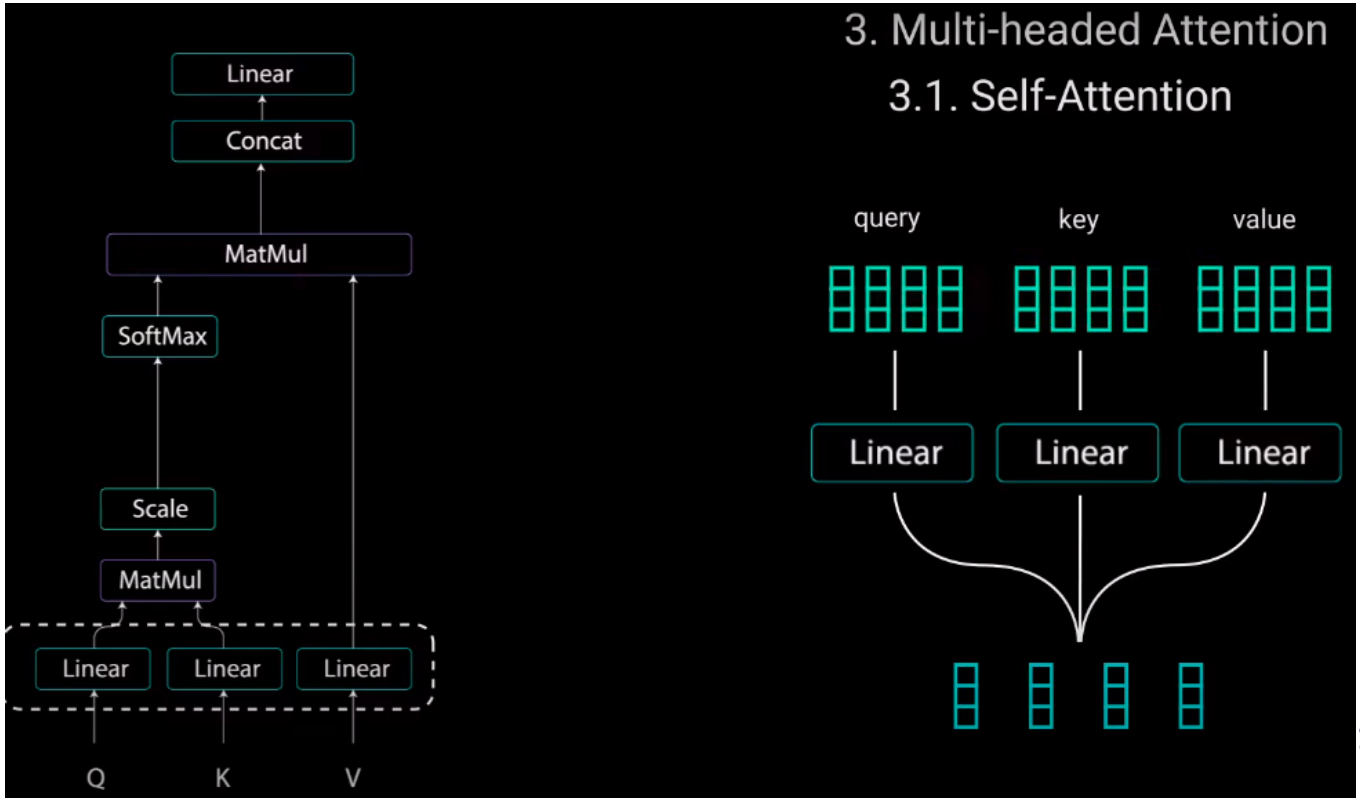


Figure – De ce site web

Attention Multi-têtes (1)

Inspiré des systèmes de recherche, l'idée est de **mettre en correspondance une requête avec un ensemble de clés** pour présenter les **meilleures valeurs correspondantes**. Dans l'auto-attention, le triplet est simplement constitué de 3 projections différentes apprises de l'entrée.



Attention Multi-têtes (2)

La fonction d'attention proposée s'appelle "Scaled Dot Product".

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

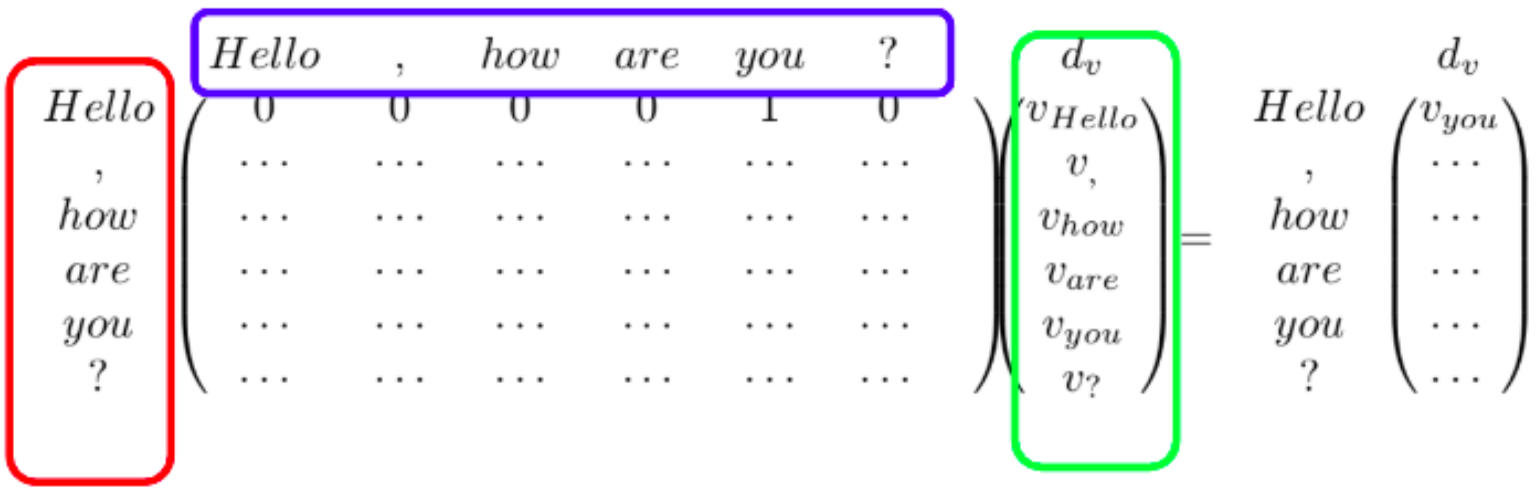
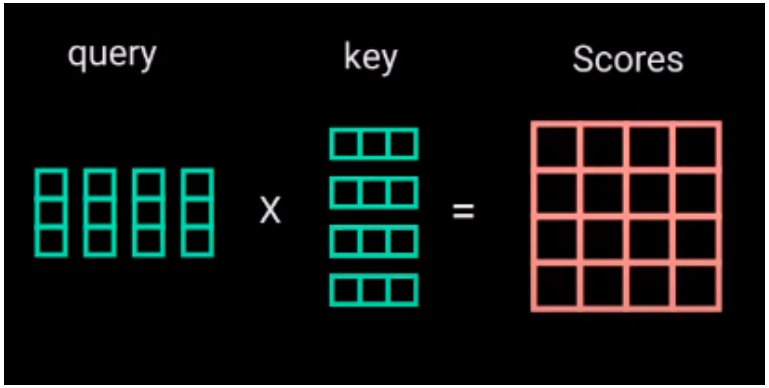
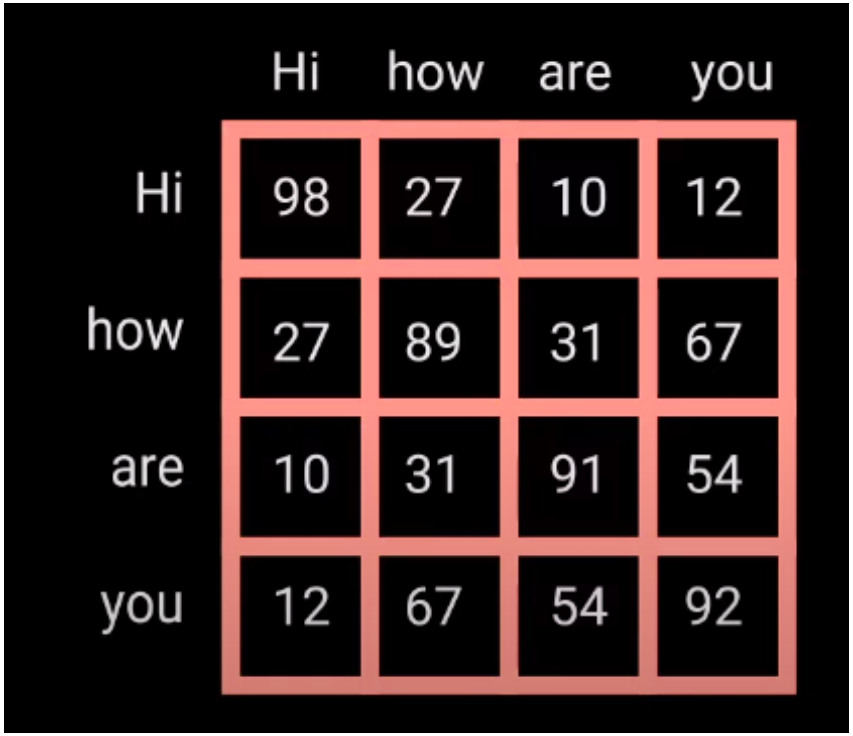


Figure – Rouge : requête Q, Violet : clé K, Vert : valeur V

Attention Multi-têtes (3)



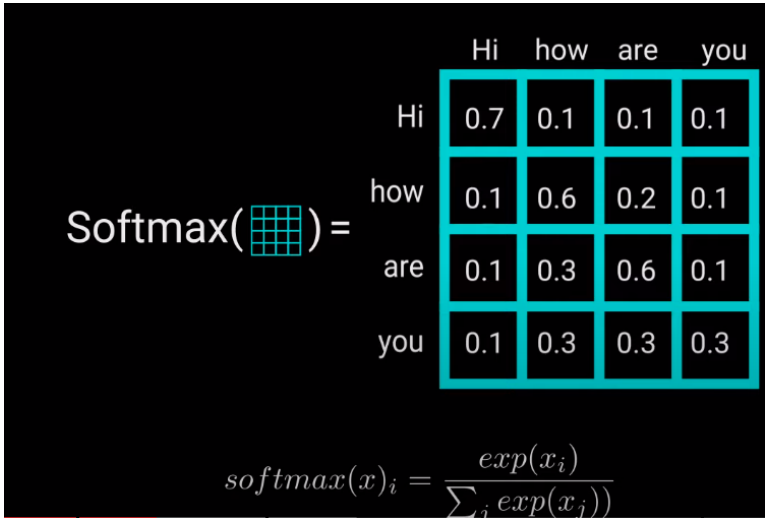
(a) Multiplication matricielle entre Q et K pour obtenir une matrice de scores



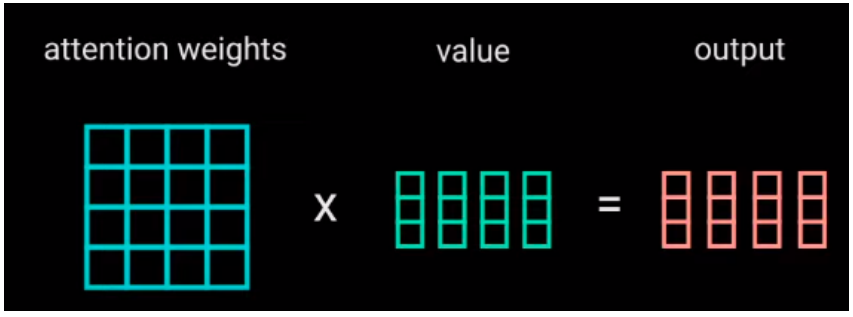
(b) Matrice de scores : quelles sont les mots pertinents les uns par rapport aux autres ?

Figure – Composant MatMul

Attention Multi-têtes (3)



(a) Après mise à l'échelle par $\sqrt{d_k}$, nous appliquons une fonction softmax pour obtenir des poids d'attention entre 0 et 1



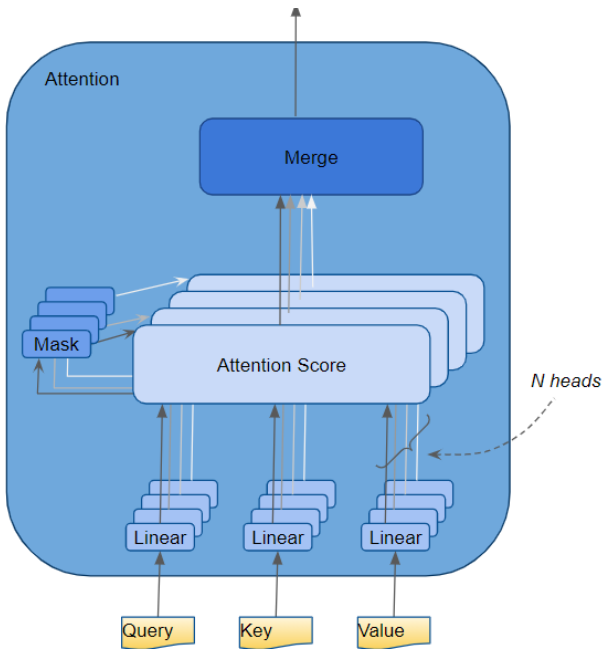
(b) Les poids d'attention permettent de mettre de côté les mots non pertinents et de mettre en évidence les mots importants

Figure – Sortie de l'Attention par produit scalaire mis à l'échelle

Attention Multi-têtes (3)

Alors pourquoi multi-têtes ?

Au lieu d'une seule fonction d'attention, les Transformers combinent la sortie de plusieurs fonctions d'attention qui fonctionnent en parallèle ! En pratique, cela signifie qu'une **tête d'attention travaille sur une plus petite portion de l'entrée.**



Qu'est-ce qui reste dans la pile d'encodeurs alors ?

Tout ce qui reste ...

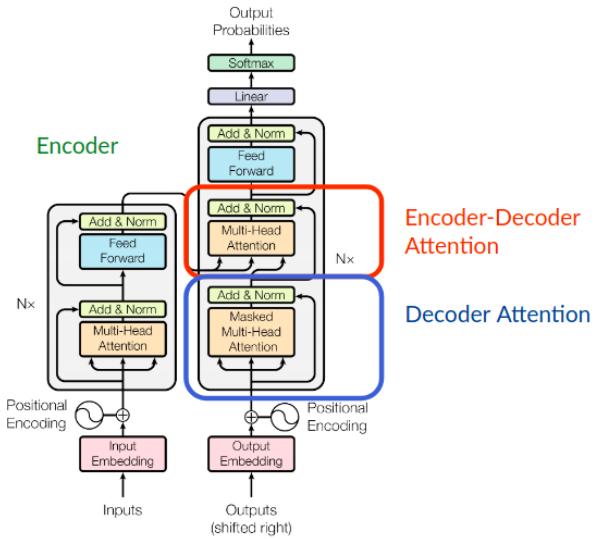
- La connexion résiduelle aide le réseau à s'entraîner en permettant au gradient de circuler directement à travers le réseau (tout comme ResNet).
- Une normalisation de couche est appliquée, car il a été démontré qu'elle aide à l'entraînement mieux que la normalisation par lots. Ba, J. L., Kiros, J. R., Hinton, G. E. (2016). Normalisation de couche.
- La rétropropagation raffine la représentation de sortie.

Des masques d'attention sont appliqués à l'entrée pour ne pas tenir compte des tokens de padding !

Pile de décodeurs

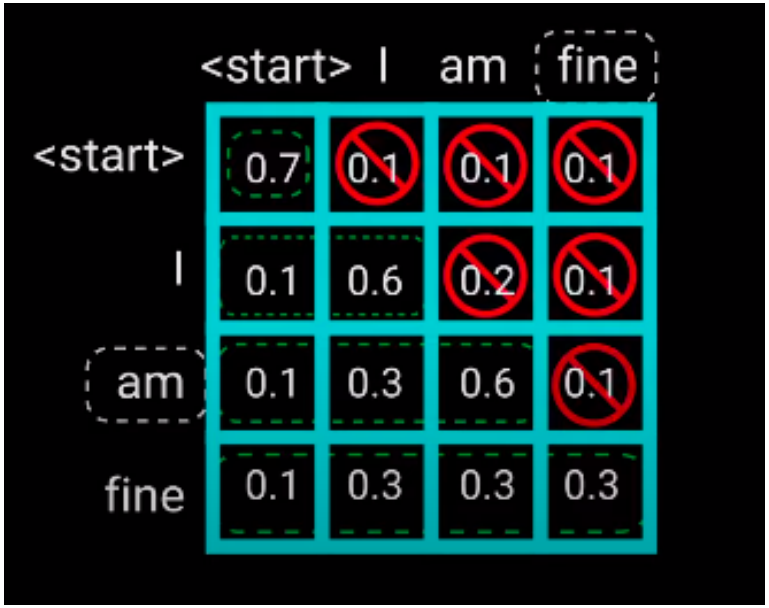
Description

- Très similaire à la pile d'encodeurs : les mêmes opérations sont utilisées !
- Les entrées sont différentes dans la configuration d'entraînement et d'inférence !
- Deux types d'attention multi-têtes : **Attention du décodeur** et **Attention encodeur-décodeur** !

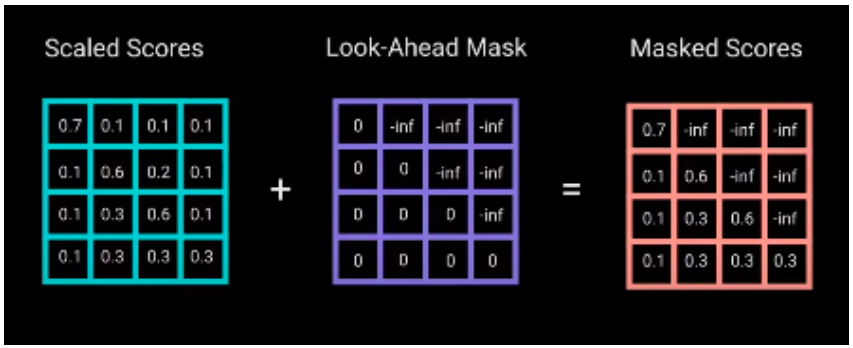


Attention du décodeur

Un schéma de masquage est utilisé pour s'assurer que **le modèle n'attende pas les mots futurs** !



(a) Masque d'attention : par exemple, lors du calcul du score d'attention pour "am", vous ne devez pas avoir accès au mot "fine"

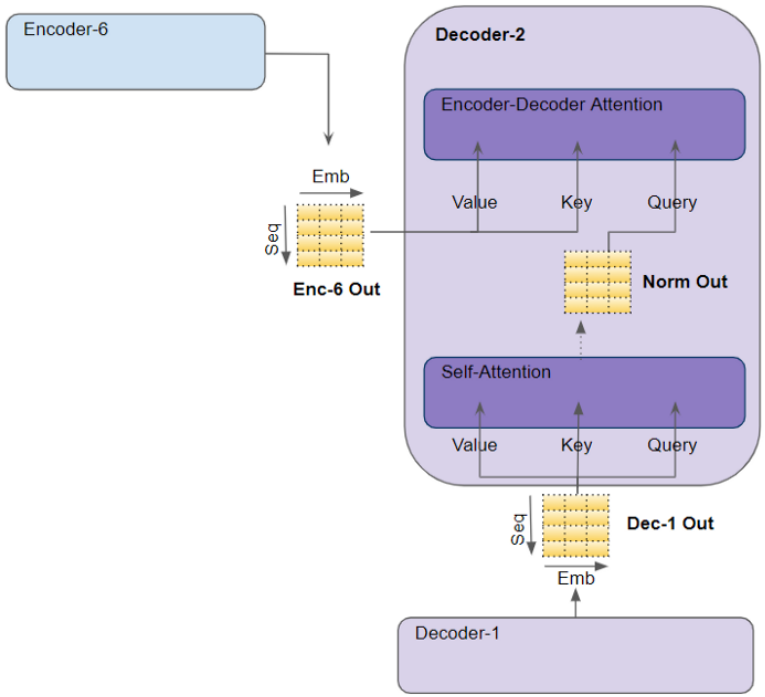


(b) Le masque d'attention et les scores d'attention ont la même taille et sont combinés pour obtenir les scores d'attention masqués

Figure — Schéma de masquage

Attention Encodeur-Décodeur

L'attention Encodeur-Décodeur utilise la même opération que dans l'Attention MultiHead.
Comme dans la configuration classique encodeur-décodeur, la sortie de la dernière couche d'encodeur est donnée comme Clé et Valeur, tandis que la Requête provient de la couche de décodeur précédente.



Conclusion

- L'architecture Encodeur-Décodeur est utilisée pour l'apprentissage seq2seq.
- Certaines architectures de réseau sont mieux adaptées à l'apprentissage seq2seq (Réseaux récurrents ou basés sur l'attention).
- Les réseaux récurrents souffrent d'une perte de mémoire à mesure que leur séquence d'entrée devient plus longue.
- Les architectures basées sur les Transformers sont désormais l'architecture privilégiée pour l'apprentissage de séquences (plus rapides à entraîner et meilleures performances dans de nombreuses tâches).
- Le texte doit être pré-traité pour être utilisable (tokenisation). Les vecteurs de caractéristiques (embeddings) jouent toujours un rôle important pour la performance d'un modèle de génération de séquence.

Merci pour votre **"attention"** !

Vue d'ensemble

Soit une séquence d'entrée x t.q. $x = x_1, .., x_N$, où N est la longueur de séquence.

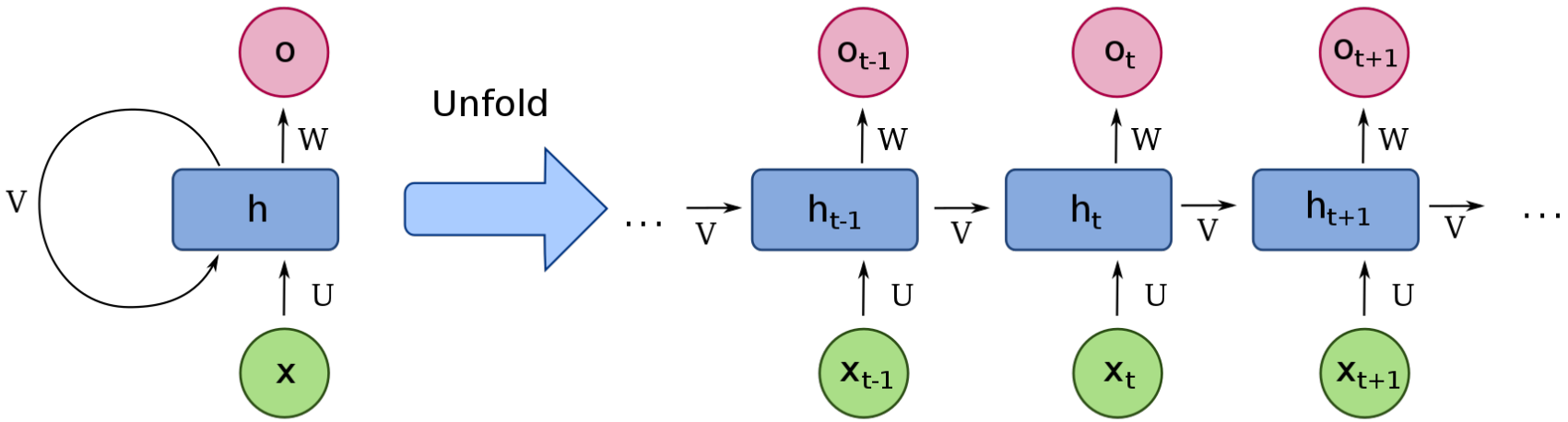


Figure – Example of a recurrent neural network

Analysis of gradient(1)

Forward propagation

Very straightforward : simply loop through (x_t, h_t, o_t) at each time step, with :

- $h_t = f(Ux_t, Vh_{t-1})$
- $o_t = g(Wh_t)$

Loss function

Given the input sequence x and target y, loss function L can be defined as

$$L(x, y) = \frac{1}{T} \sum_{t=1}^{T} l(y, o_t)$$

Analysis of gradient(2)

Chain rule (example)

$$\begin{aligned}\frac{\delta L}{\delta V} &= \frac{1}{T}\sum_{t=1}^{T}\frac{\delta l(y,o_t)}{\delta V} \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{\delta l(y,o_t)}{\delta o_t}\frac{\delta g(Wh_t)}{\delta h_t}\frac{\delta h_t}{\delta V}\end{aligned} \tag{1}$$

$$\frac{\delta h_t}{\delta V} = \frac{\delta f(Ux_t, Vh_{t-1})}{\delta V} + \frac{\delta f(Ux_t, Vh_{t-1})}{\delta h_{t-1}}\frac{\delta h_{t-1}}{\delta V} \tag{2}$$

Analysis of gradient(3)

Backpropagation through time (BPTT)

$$a_t = \frac{\delta h_t}{\delta V},$$
$$b_t = \frac{\delta f(Ux_t, Vh_{t-1})}{\delta V},$$
$$c_t = \frac{\delta f(Ux_t, Vh_{t-1})}{\delta h_{t-1}}$$

Given $a_0 = 0$ and $a_t = b_t + c_t a_{t-1}$, we can prove that for $t >= 1$

$$a_t = b_t + \sum_{i=1}^{t-1}(\prod_{j=i+1}^{t} c_j)b_i \tag{3}$$

idées principales

- Reset from memory : on "oublie" une partie de l'état précédent
- Write to memory : on "écrit" en mémoire ce qui est important (état précédent + entrée courante)
- Read from memory : on envoie à la prochaine cellule l'information

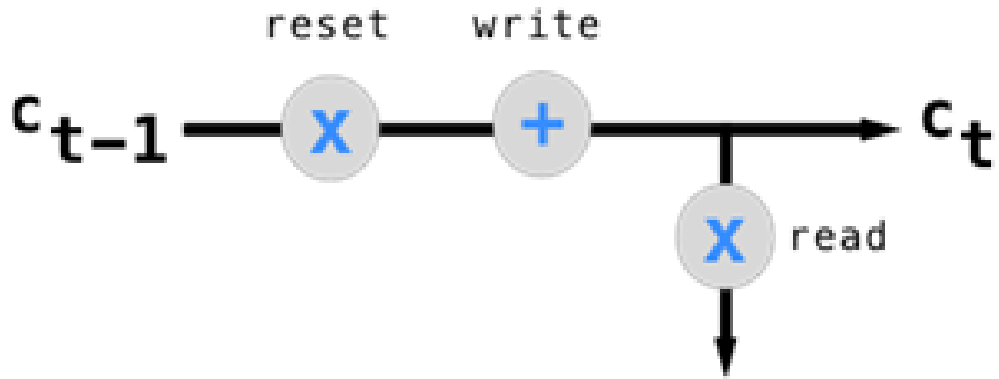


Figure – Core idea behind LSTM

Forget Gate

$$F_t = \sigma(W_F x_t + U_F h_{t-1} + b_F) \tag{4}$$

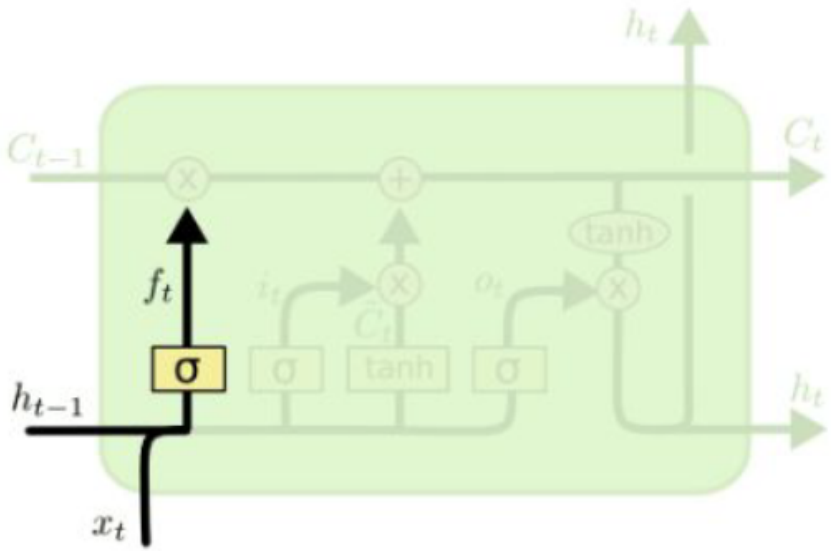


Figure – Forget Gate : "reset from memory"

Input Gate & Candidate Cell

$$I_t = \sigma(W_I x_t + U_I h_{t-1} + b_I)$$
$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

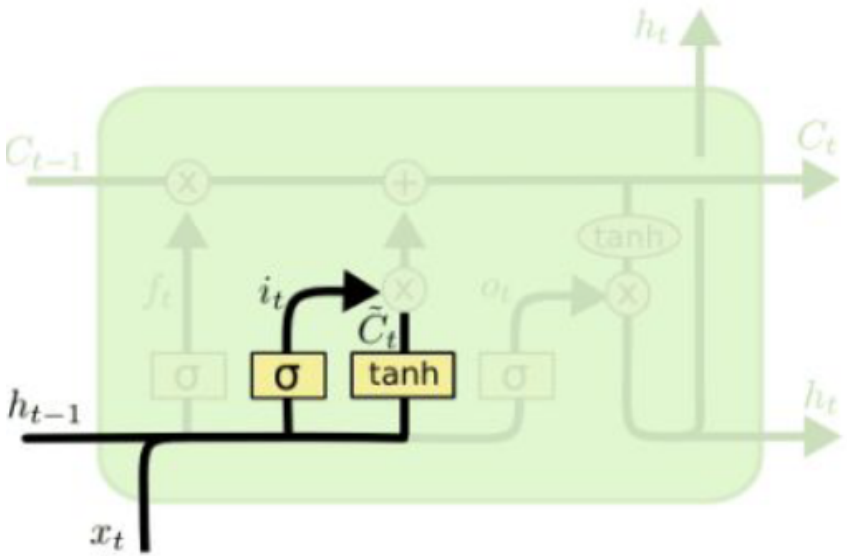


Figure – Input Gate : "write from memory"

Update Gate

$$c_t = F_t \circ c_{t-1} + I_t \circ \tilde{C}_t \tag{5}$$

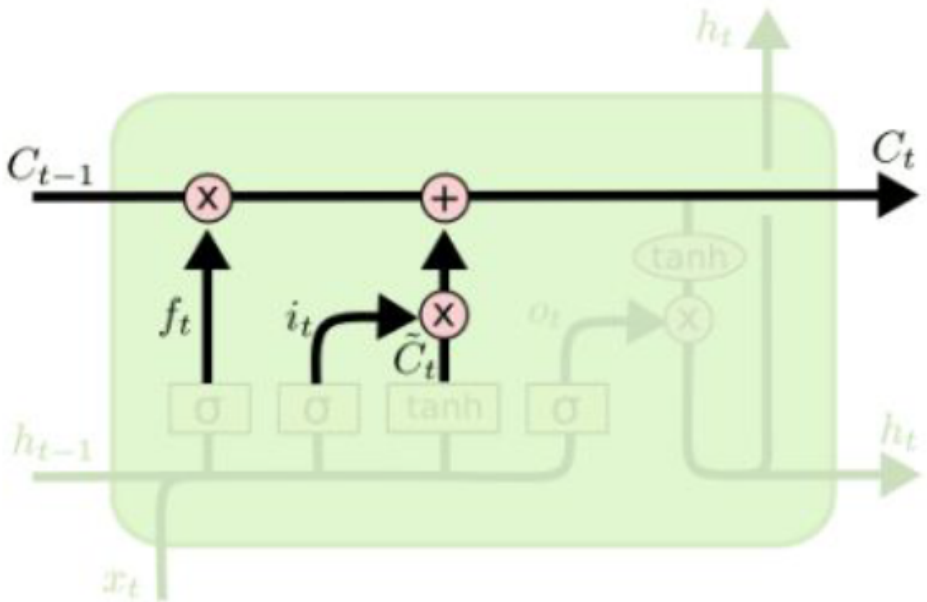


Figure – Update Gate

Generating the output

$$O_t = \sigma(W_O x_t + U_O h_{t-1} + b_O) \text{(output gate)}$$
$$h_t = O_t \circ \tanh(c_t)$$
$$o_t = f(W_o h_t + b_o)$$

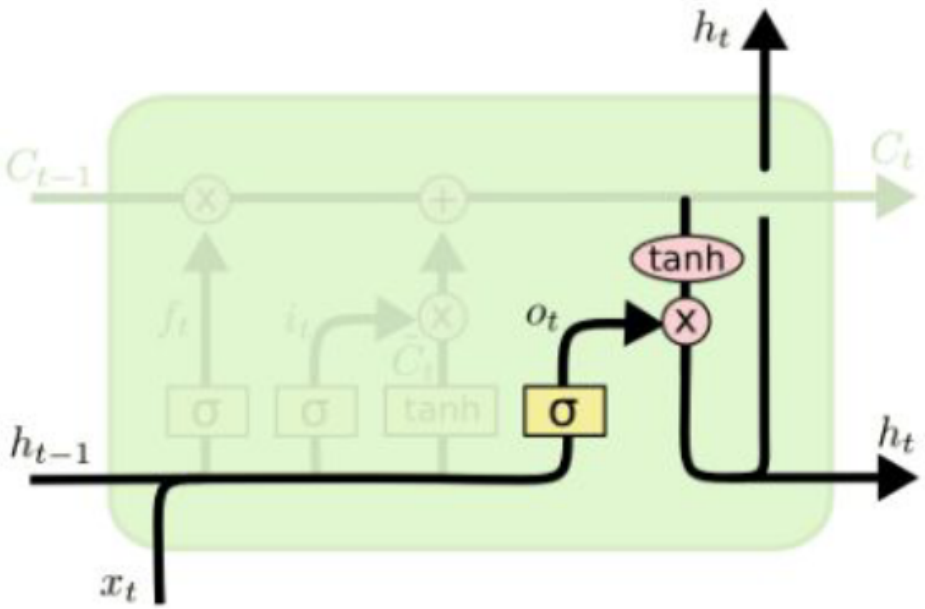


Figure – "Read from memory"