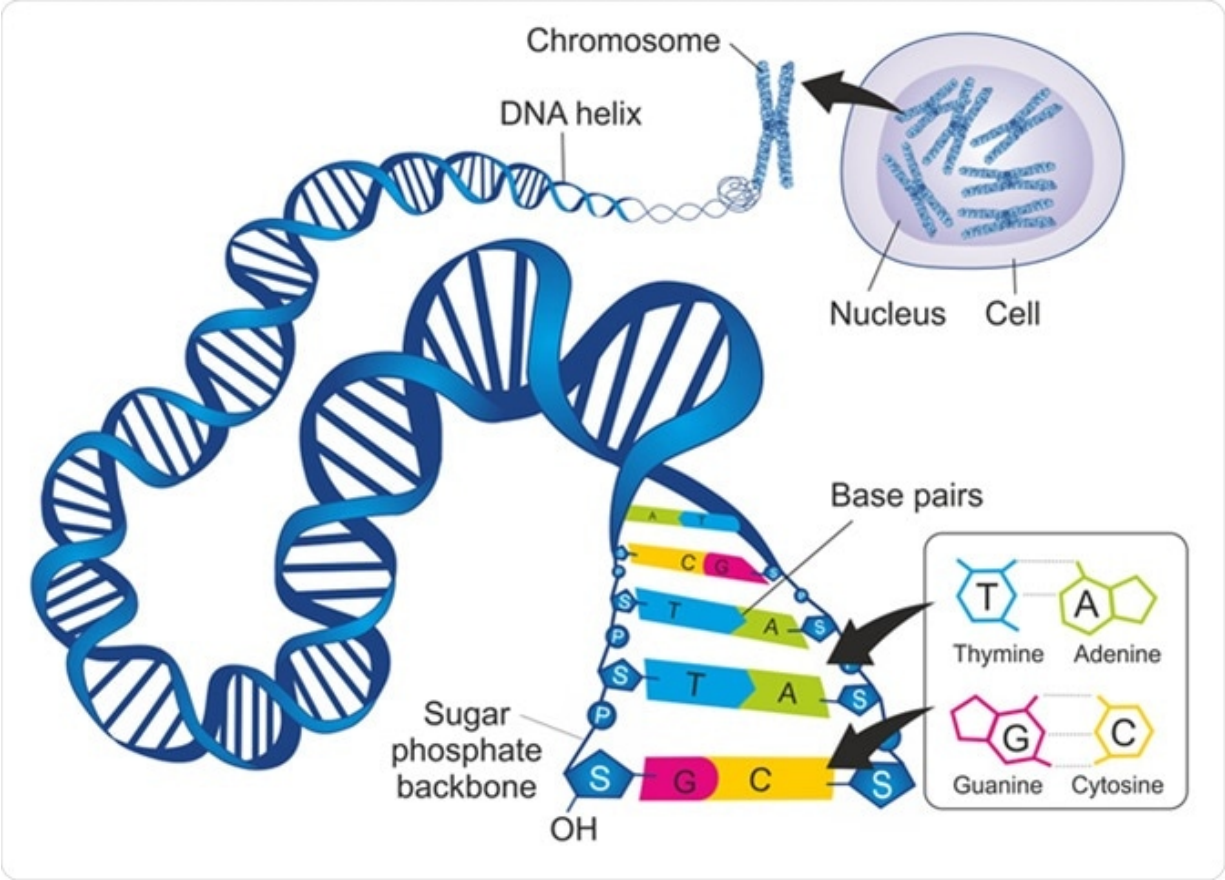
Deep Learning et séquences génomiques

Master parcours SSD - UE Apprentissage Statistique II

Pierre Mahé - bioMérieux & Université de Grenoble-Alpes
cours préparé en collaboration avec Meriem El Azami (bioMérieux)

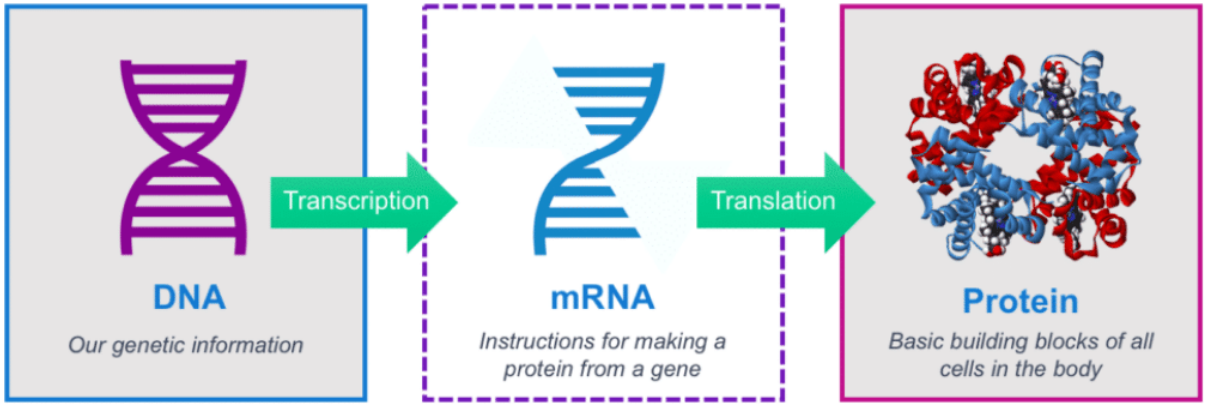
Introduction

Séquences biologiques - ADN



- molécule d'ADN $\sim$ une séquences de lettres `A,T,G,C`
  - nucléotides ou bases
- génome humain $\sim 3 \times 10^9$ bases

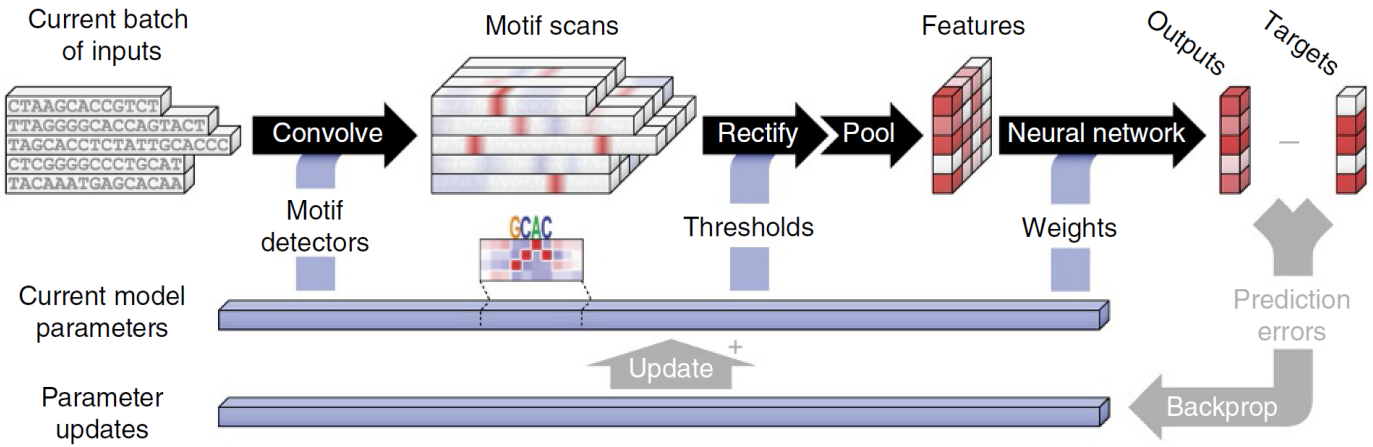
Séquences biologiques - ADN, ARN et protéines



- gène : portion d'ADN qui "code" pour un ARN messager
- ARNm $\sim$ séquences de bases `A,U,G,C`
- protéine : molécule obtenue à partir d'un ARNm
  - une séquence d'acides aminés : alphabet de taille 20
  - 1 triplet de nucléotides $\rightarrow$ 1 acide aminé

Quelques applications...

Régulation génétique :



- *Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning* (Alipanahi et al., 2015)

Quelques applications...

Annotation de protéines :

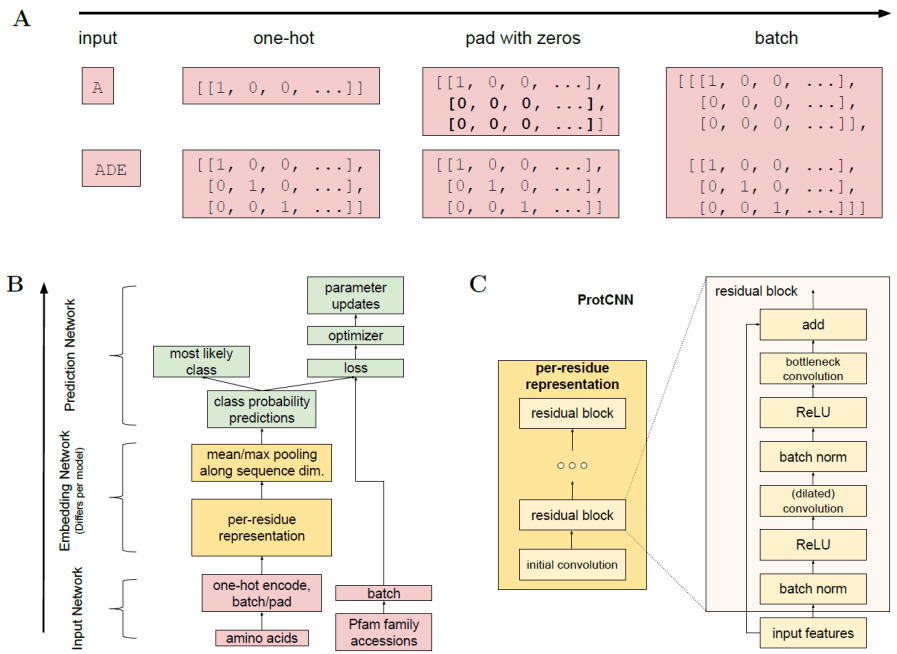


Figure 6: Architecture descriptions of neural networks. (A) Adding padding to a sequence for batching. (B) The model architecture shared by all neural networks includes the Input (red), Embedding (yellow), and Prediction (green) Networks. (C) ResNet architecture used by the ProtCNN models.

- *Using Deep Learning to Annotate the Protein Universe* (Bileschi et al., 2019).

Quelques applications...

Classification taxonomique :

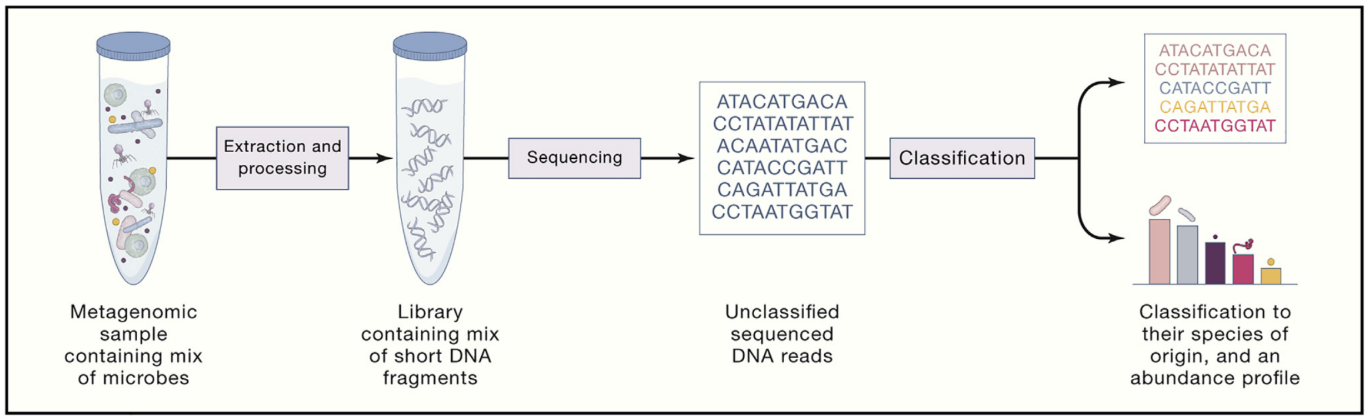
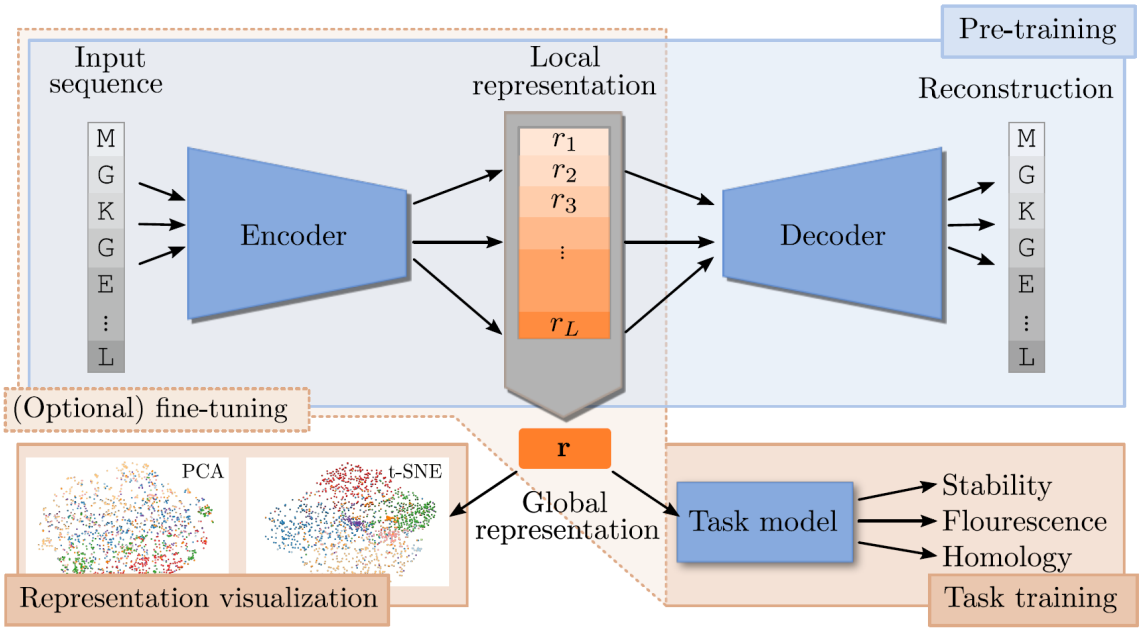


Figure 1. Processing Steps to Go from a Complex Metagenomic Sample to an Abundance Profile of Sample Content

- *DeepMicrobes : taxonomic classification for metagenomics with deep learning* (Liang et al., 2020).

Quelques applications...

"Design" de séquences biologiques :



- *Design by adaptive sampling* (Brookes and Listgarten, 2018).
- *What is a meaningful representation of protein sequences ?* (Detlefsen et al., 2021).

Représentation de séquences

Une question centrale : comment représenter ces séquences

Représentation de séquences

Une question centrale : comment représenter ces séquences

$\Rightarrow$ différentes stratégies tirées du NLP et/ou de l'imagerie :

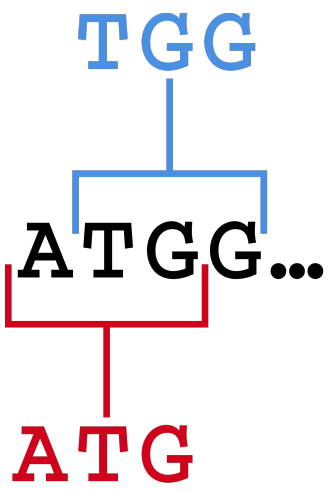
1. "bag of words" & profils de k-mers
   - + : simple à mettre en oeuvre
   - - : perte de l'information séquentielle
2. CNN pour la détection de motifs génomiques
   - + : expressivité et interprétabilité
   - - : difficile de capturer les interactions distantes
3. réseaux récurrents type LSTM
   - + : meilleure prise en compte de la nature séquentielle
   - - : plus coûteux à mettre en oeuvre
4. modèles d'attention et "transformers"

Architectures

- Bag-of-words & profils de k-mers
- CNNs & détection de motifs
- Réseaux récurrents, GRU & LSTM
- Transformers

k-mers

k-mer : une suite de k nucléotides dans une séquence d'ADN



- jusqu'à 4^k k-mers distincts
- $L - k + 1$ k-mers pour une séquence de longueur L

- un k-mer $\sim$ un mot de longueur k
- approche "bag of words" $\rightarrow$ "profil de k-mers"

Profils de k-mers

Principe : se ramener à une représentation vectorielle simple

- point d'entrée d'algorithmes "standards" ou MLP

Profils de k-mers

Principe : se ramener à une représentation vectorielle simple

- point d'entrée d'algorithmes "standards" ou MLP

Formellement, pour une séquence x de longueur L :

$$\boxed{x \in \{\mathtt{A},\mathtt{T},\mathtt{G},\mathtt{C}\}^L \Rightarrow \Phi(x) \in \mathbb{R}^{4^k}}$$

$\Rightarrow$ chaque dimension de $\Phi(x)$ correspond à un k-mer

- $\Phi_u(x)$ peut compter les occurrences du k-mer u
- $\Phi_u(x)$ peut simplement coder sa présence / absence
- on peut autoriser des mismatches dans les occurrences

Profils de k-mers

Avantage et inconvénient :

- + : très simple à mettre en oeuvre !
  - e.g., fenêtre glissante + dictionnaire python
- - : on perd toute l'information séquentielle

Challenge pour les séquences ADN : la taille du dictionnaire

- $k = 5$: 1.024
- $k = 8$: 65.536
- $k = 11$: 4.194.304

$\Rightarrow$ utilisation de k-mers "canoniques" pour exploiter la complémentarité des brins d'ADN

- même indice pour un k-mer et son reverse-complément
- vocabulaire de taille $4^k/2$ si k impair

Profils de k-mers

Limitation fondamentale de l'approche bag-of-words / profils de k-mers : pas de métrique de similarité entre k-mers

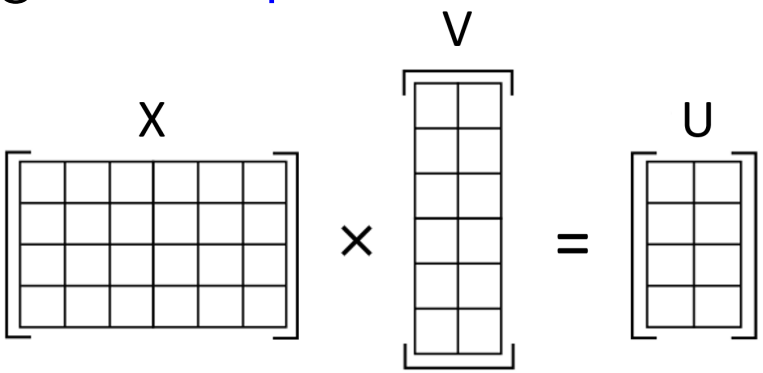
- soit parfaitement identiques ou complètement différents

Profils de k-mers

Limitation fondamentale de l'approche bag-of-words / profils de k-mers : pas de métrique de similarité entre k-mers

- soit parfaitement identiques ou complètement différents

Extension : intégrer des représentations continues des k-mers



- $X \in \mathbb{R}^{n \times 4^k}$: profils de k-mers, n séquences, 4^k variables
- $V \in \mathbb{R}^{4^k \times d}$: représentation ou "embeddings" des k-mers
- $U \in \mathbb{R}^{n \times d}$: profils d'"embeddings" de k-mers

Profils de k-mers

Embeddings de k-mers : comment construire la matrice V ?

- $[4^k \times d]$: les représentations en dimension d des k-mers

$\Rightarrow$ Deux stratégies :

- supervisée : apprentissage "end to end" de la matrice en lien avec la tâche de prédiction
- non-supervisée : apprentissage générique à partir de données externes + transfer learning

$\Rightarrow$ **un élément clé** des architectures plus complexes à venir !

- CNN, LSTM, transformers

Apprentissage d'embeddings : approche supervisée

Apprentissage end-to-end via la couche Keras Embedding() :

```
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding
```

```
# specify parameters
k = 5             # k-mer length
input_dim = 4**k  # length of input vectors
embed_dim = 64    # embedding dimension
n_classes = 10    # number of classes
# define model
mlp = Sequential()
mlp.add(Embedding(input_dim=input_dim, output_dim = embed_dim))
mlp.add(Dense(n_classes, activation='softmax'))
mlp.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 64)          65536
_________________________________________________________________
dense (Dense)                (None, None, 10)          650
=================================================================
Total params: 66,186
Trainable params: 66,186
Non-trainable params: 0
_________________________________________________________________
```

- embedding en dim. 64 des 5-mers + régression multinomiale
- $4^k \times d$ paramètres dans la couche Embedding
- entrée du réseau : **suite d'indices** dans le dico.
- ajouter une **couche de pooling** pour avoir une représentation dans $\mathbb{R}^d$

$\Rightarrow$ matrice de projection apprise par descente de gradient de la fonction de perte

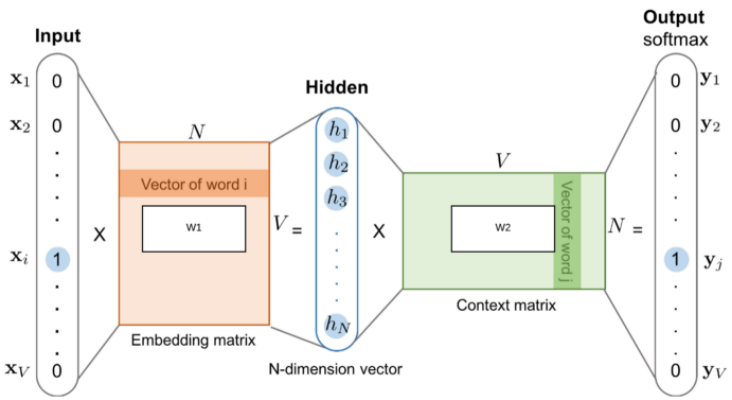
Apprentissage d'embeddings : approche non-supervisée

Approche non-supervisé : plusieurs algorithmes type word2vec

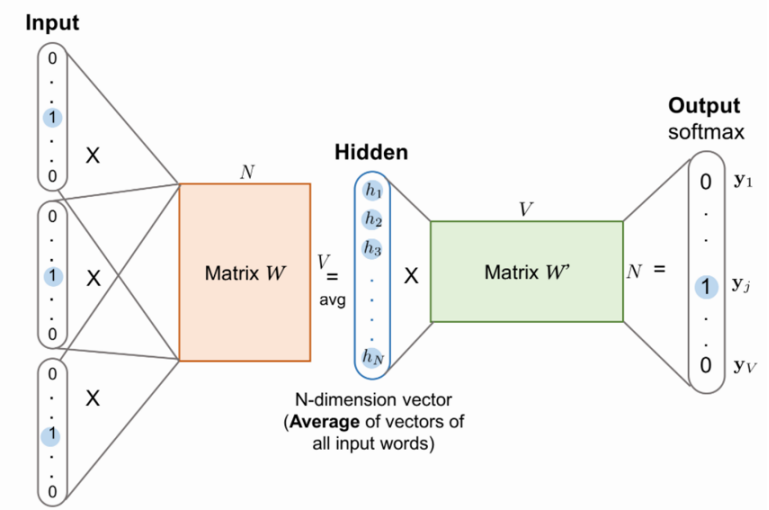
word2vec : un MLP à 2 couches appris sur une tâche de classification "virtuelle" pour capturer le sens des mots

"Quel chouette cours d'apprentissage statistique"

prédire le contexte à partir du mot central



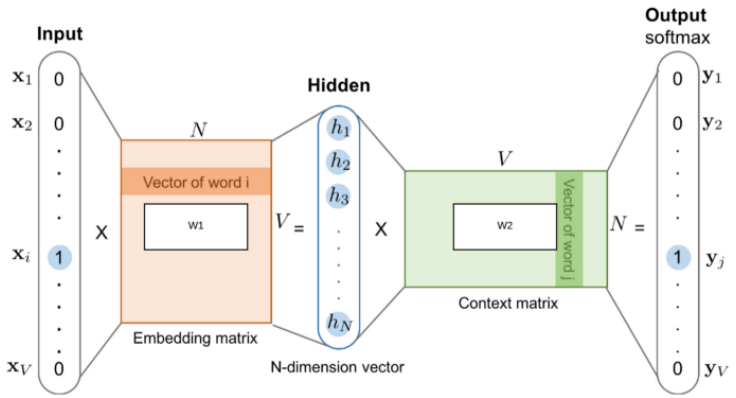
prédire le mot central à partir du contexte



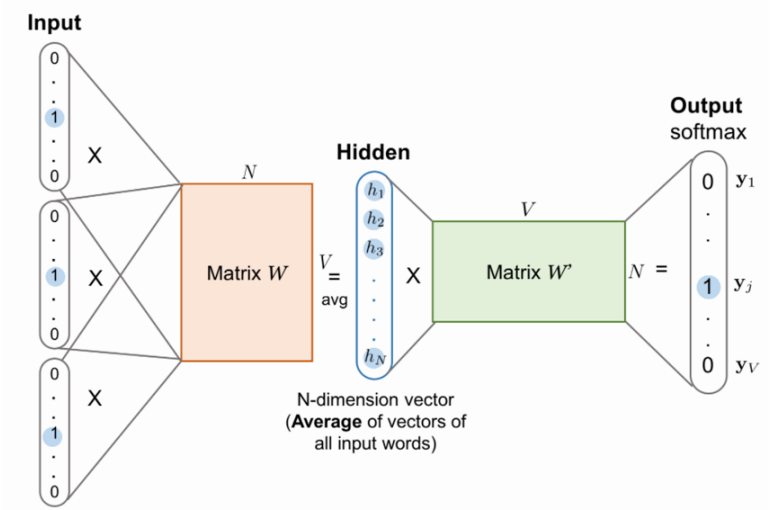
Word2vec in a nutshell...

"Quel chouette cours d'apprentissage statistique"

prédire le contexte à partir du mot central



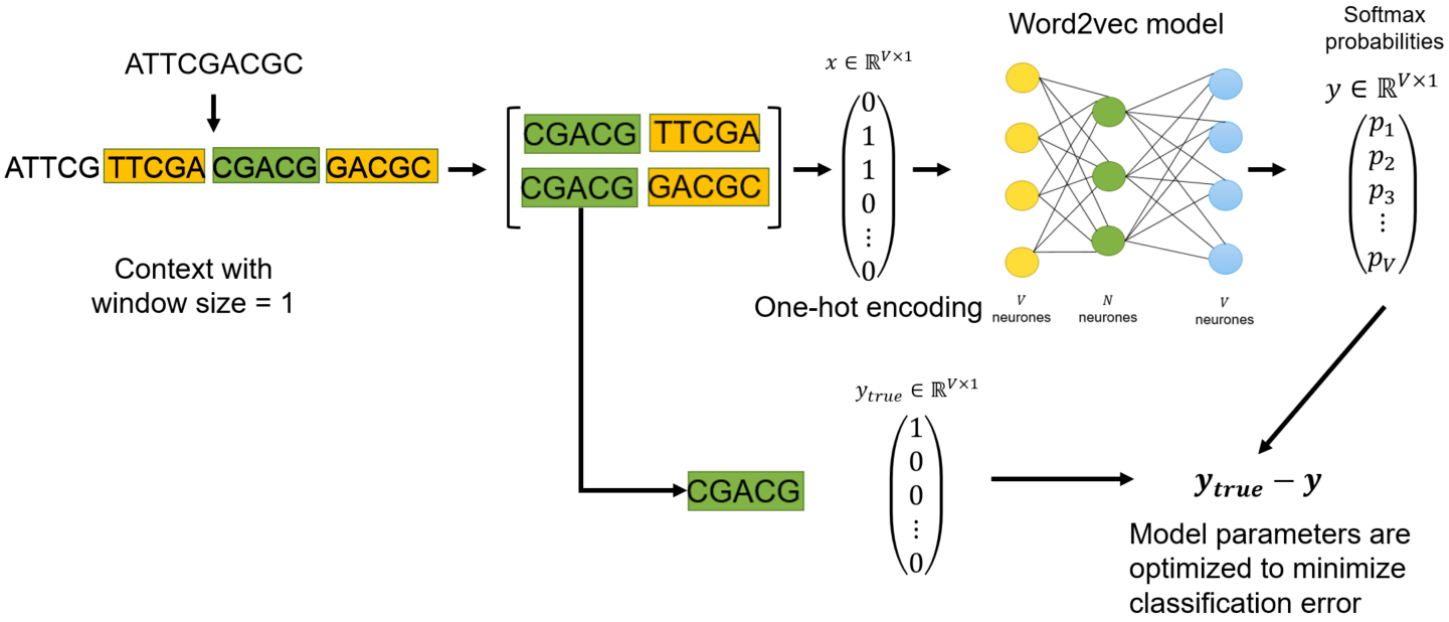
prédire le mot central à partir du contexte



- modèle de classification multiclasse : 1 mot = 1 classe
- 1 couche cachée avec activation linéaire
- deux matrices d'embeddings : en entrée et en sortie
  - mot central et contexte, peuvent être les mêmes
- appris sur de nombreux couples (mot central ; contexte)

Word2vec et k-mers

Apprentissage word2vec sur des k-mers :



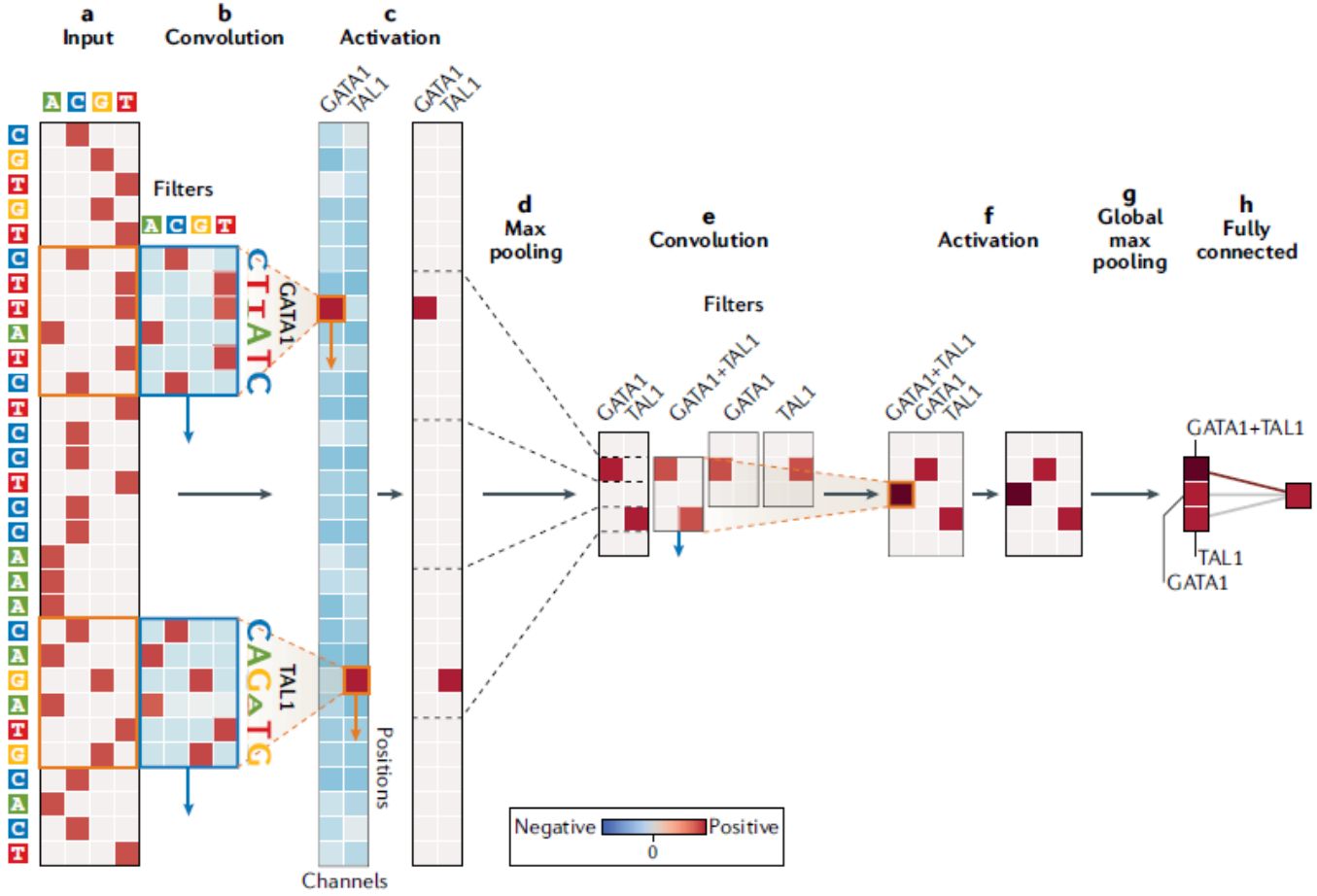
- on tire des k-mers (centraux + contexte) sur des bases de données externes (e.g., génomes complets ou gènes)
- on apprend le modèle word2vec
- on utilise les embeddings comme matrice de projection "statique" ou en initilisation d'une couche Embedding()

Architectures

- Bag-of-words & profils de k-mers
- CNNs & détection de motifs
- Réseaux récurrents, GRU & LSTM
- Transformers

CNN et séquences génomiques

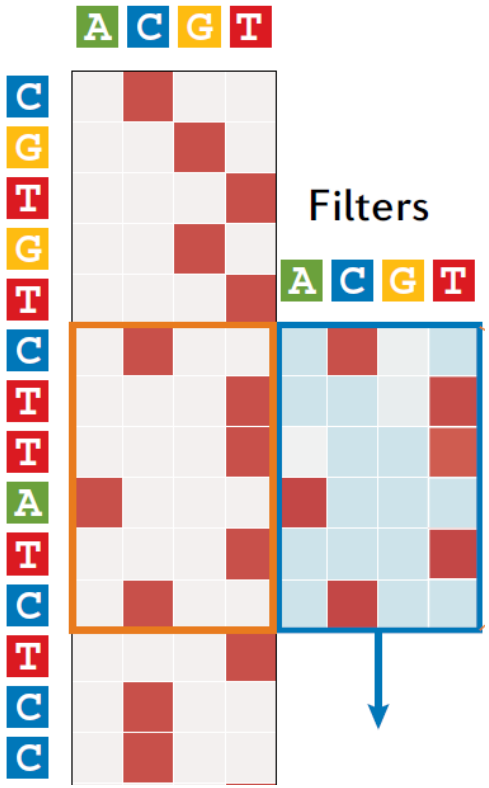
Illustration (tirée de Eraslan et al. (2019)) :



⇒ généralisation des CNNs à une information séquentielle

CNN et séquences génomiques

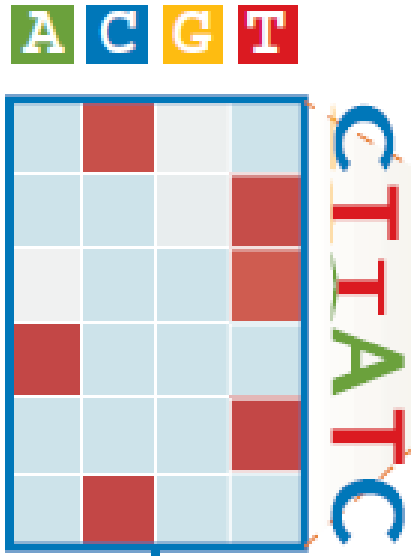
Transformation des séquences à une "image" 2D par one-hot encoding des bases `A,C,G,T` :

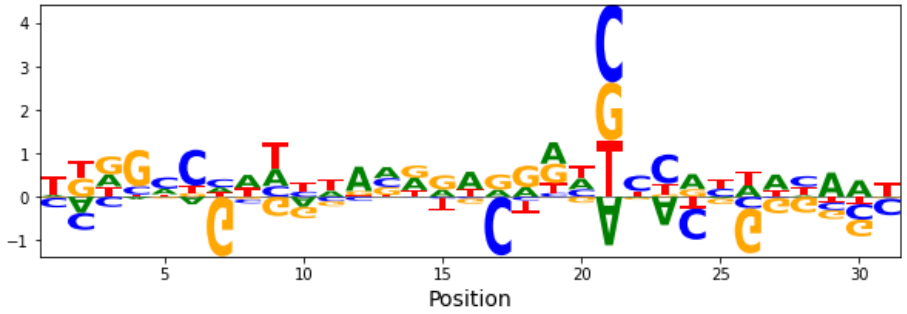


- 1 séquence de longueur $L \rightarrow$ une matrice $[L \times 4]$
- masques de convolution : matrices de dimension $[w \times 4]$

CNN et détection de motifs

Un masque de convolution $\sim$ un motif probabiliste :





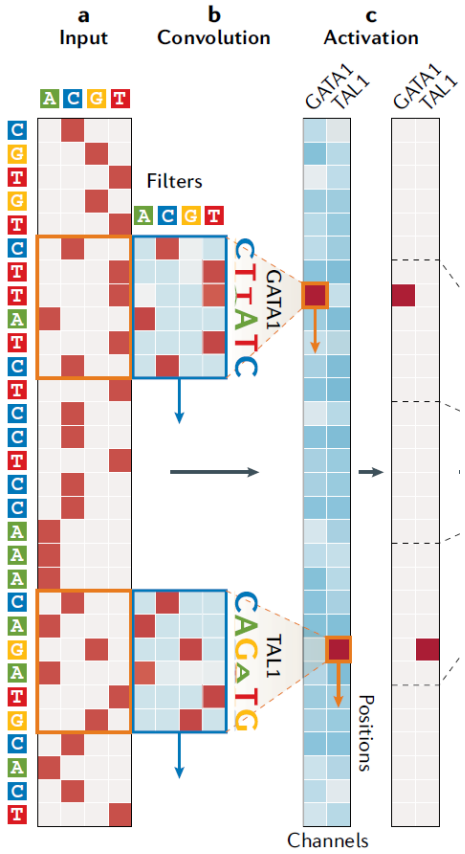
représentation en "sequence logo"

$\Rightarrow$ définit une fonction de score le long de la séquence

- contribution positive / négative d'une base à une position donnée en fonction de son poids dans le masque
- score élevé du masque à une position donnée si les bases de poids élevé sont présentes dans le bon ordre

CNN et détection de motifs

1ère couche de convolution : détection de motifs probabilistes



⇒ quelle stratégie suivre pour la suite de l'architecture ?

Deep vs shallow architectures ?

Stratégie "shallow" :

- 1 couche de convolution + couche(s) dense(s)
- (avec pooling global ou local pour réduire la dimension)

$\Rightarrow$ + : interprétabilité du modèle

- features = "motifs probabilistes" + interactions via MLP

Stratégie "deep" :

- cascade de convolution / pooling + couche(s) dense(s)

$\Rightarrow$ + : expressivité du modèle

- combinaison spatiale de "motifs probabilistes"

$\Rightarrow$ - : perte d'interprétabilité du modèle

En pratique...

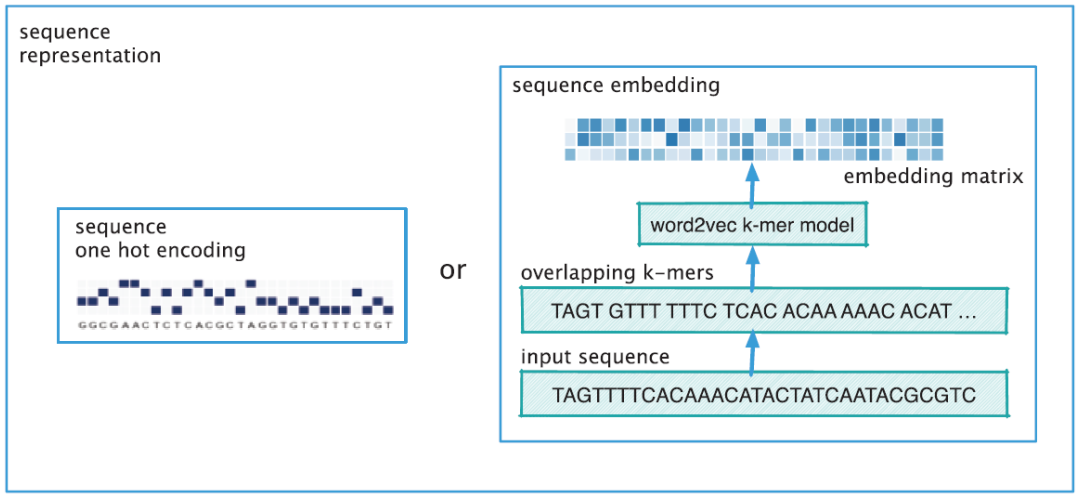
On préfère parfois travailler à partir d'un alignement multiple.

```
Sequence1   -TCAGGA-TGAAC----
Sequence2   ATCACGA-TGAACC---
Sequence3   ATCAGGAATGAATCC--
Sequence4   -TCACGATTGAATCGC-
Sequence5   -TCAGGAATGAATCGCM
```

- alphabet de taille 5 : $\{A, C, G, T, -\}$

On "padde" ou on tronque les séquences de taille variables.

On peut travailler à partir de k-mers et d'embeddings [1].



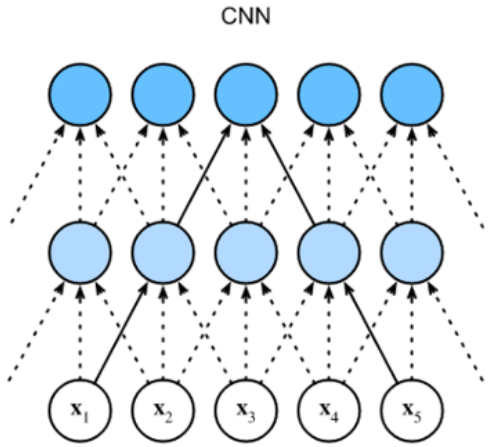
1. Illustration tirée de Trabelsi et al. (2019).

CNN et séquences génomiques

CNN et génomique : beaucoup d'applications depuis 2015.

- identification "end to end" de motifs génomiques

Une limitation fondamentale néanmoins : difficile de capturer des interactions distantes dans les séquences



⇒ utilisation d'architectures dédiées aux séquences :

- réseaux récurrents type GRU et LSTM
- mécanismes d'attention type "transformers"

(parfois combinées avec des CNNs)

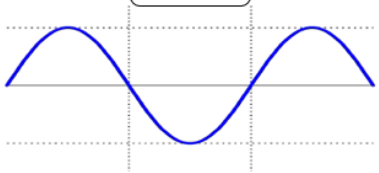
Architectures

- Bag-of-words & profils de k-mers
- CNNs & détection de motifs
- Réseaux récurrents, GRU & LSTM
- Transformers

Qu'est ce qu'une séquence ?

Exemples de séquences :

Signal



Texte

"Quel chouette cours d'apprentissage statistique"

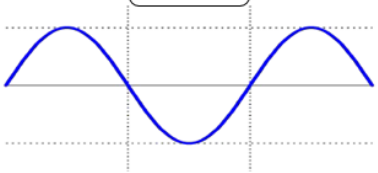
Sequence génomique

ACCTGCTGCCATGCT

Qu'est ce qu'une séquence ?

Exemples de séquences :

Signal



Texte

"Quel chouette cours d'apprentissage statistique"

Sequence génomique

ACCTGCTGCCATGCT

⇒ propriétés des séquences :

- observations non i.i.d
- observations suivent un ordre séquentiel
- processus non-stationnaire

Modélisation des séquences

Comment "bien" modéliser des séquences ?

- Conserver la notion d'ordre
- Capturer des dépendances long-terme
- Gérer les séquences de longueur variable
- Limiter le nombre de paramètres

Modélisation des séquences

Objectif général : estimer x_t sachant les observations passées

$$\boxed{x_t \sim p(x_t | x_1, x_2, \ldots, x_{t-1})}$$

Modélisation des séquences

Objectif général : estimer x_t sachant les observations passées

$$\boxed{x_t \sim p(x_t | x_1, x_2, \ldots, x_{t-1})}$$

$\Rightarrow$ 2 stratégies pour faire cela :

1. Les modèles auto-régressifs
   - Le principe :
     - Considérer une fenêtre τ
     - Utiliser uniquement les observations dans cette fenêtre pour estimer x_t

Modélisation des séquences

Objectif général : estimer x_t sachant les observations passées

$$\boxed{x_t \sim p(x_t | x_1, x_2, \ldots, x_{t-1})}$$

$\Rightarrow$ 2 stratégies pour faire cela :

1. Les modèles auto-régressifs
   - Le principe :
     - Considérer une fenêtre τ
     - Utiliser uniquement les observations dans cette fenêtre pour estimer x_t
   - Les limites :
     - Comment choisir τ ?
     - Les erreurs s'accumulent au fil des itérations

$\Rightarrow$ peut-être modélisé par des approches "standards".

- e.g., réseau de neurones MLP ou modèle de régression

Modélisation des séquences

Objectif général : estimer x_t sachant les observations passées

$$\boxed{x_t \sim p(x_t | x_1, x_2, \ldots, x_{t-1})}$$

$\Rightarrow$ 2 stratégies pour faire cela :

2. Les modèles auto-régressifs avec représentation latente
   - Le principe :
     - Construire un résumé des observations passées h_t
     - Utiliser le résumé h_t et x_{t-1} pour estimer x_t :

$$p(x_t | x_1, x_2, \ldots, x_{t-1}) \approx p(x_t | x_{t-1}, h_t)$$

     - Mettre à jour le résumé :

$$h_{t+1} = g(h_t, x_t)$$

$\Rightarrow$ peut être modélisé par des réseaux de neurones récurrents.

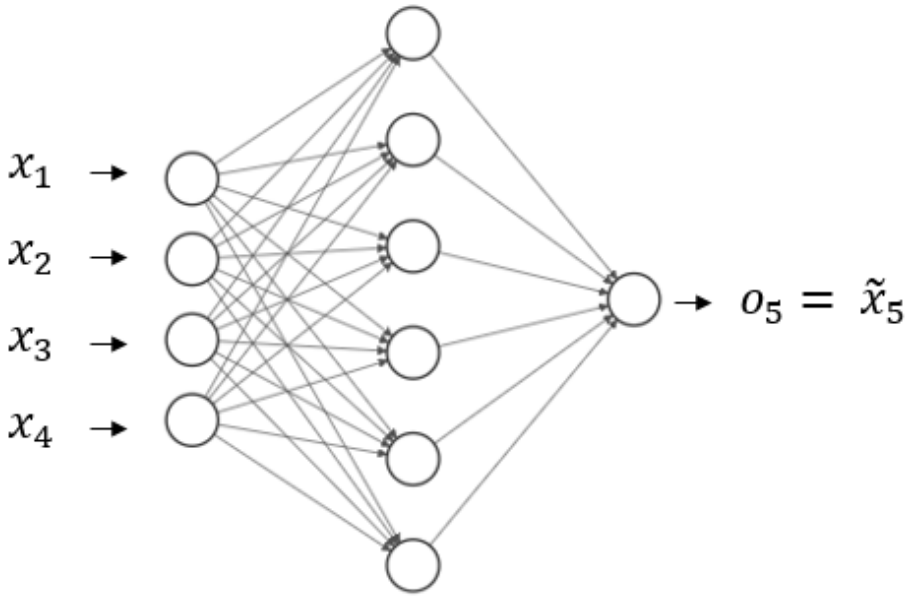
Architectures

- Bag-of-words & profils de k-mers
- CNNs & détection de motifs
- Réseaux récurrents, GRU & LSTM
  - Réseaux de neurones récurrents
  - GRU : Gated Recurrent Units
  - LSTM : Long Short Term Memory
- Transformers

RNN sans mémoire : MLP revisité

On considère un modèle auto-regressif :

- Un mini-batch de données $X \in \mathbb{R}^{n \times d}$
  - NB : d= taille de la fenêtre temporelle, signal univarié
- Un MLP avec une seule couche cachée



$$\boldsymbol{H} = \Phi(\boldsymbol{X}\boldsymbol{W}_{x,h} + \boldsymbol{b}_h)$$

$$\boldsymbol{O} = \boldsymbol{H}\boldsymbol{W}_{h,q} + \boldsymbol{b}_q$$

RNN avec mémoire

On considère un modèle auto-régressif avec mémoire :

- Un mini-batch de données $X_t \in \mathbb{R}^{n \times d}, t = 1, \ldots, T$
  - NB : ici $d = \#$ features, le signal peut être multivarié
- Un réseau de neurones récurrent :

$$\boldsymbol{H} = \Phi(\boldsymbol{X}\boldsymbol{W}_{x,h} + \boldsymbol{b}_h) \rightarrow \boldsymbol{H}_t = \Phi(\boldsymbol{X}_t\boldsymbol{W}_{x,h} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{h,h} + \boldsymbol{b}_h)$$

$$\boldsymbol{O} = \boldsymbol{H}\boldsymbol{W}_{h,q} + \boldsymbol{b}_q \rightarrow \boldsymbol{O}_t = \boldsymbol{H}_t\boldsymbol{W}_{h,q} + \boldsymbol{b}_q$$

RNN avec mémoire

On considère un modèle auto-régressif avec mémoire :

- Un mini-batch de données $X_t \in \mathbb{R}^{n \times d}, t = 1, \ldots, T$
  - NB : ici $d = \#$ features, le signal peut être multivarié
- Un réseau de neurones récurrent :

$$\boldsymbol{H} = \Phi(\boldsymbol{X}\boldsymbol{W}_{x,h} + \boldsymbol{b}_h) \rightarrow \boldsymbol{H}_t = \Phi(\boldsymbol{X}_t\boldsymbol{W}_{x,h} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{h,h} + \boldsymbol{b}_h)$$

$$\boldsymbol{O} = \boldsymbol{H}\boldsymbol{W}_{h,q} + \boldsymbol{b}_q \rightarrow \boldsymbol{O}_t = \boldsymbol{H}_t\boldsymbol{W}_{h,q} + \boldsymbol{b}_q$$

Remarques :

- $\boldsymbol{H}_t =$ l'état caché du RNN
- Les paramètres du modèle $(\boldsymbol{W}_{h,h}, \boldsymbol{W}_{x,h}, \boldsymbol{b}_h, \boldsymbol{b}_q)$ ne dépendent pas de t

Représentation graphique d'un RNN

$$\boldsymbol{H}_t = \phi([\boldsymbol{X}_t \quad \boldsymbol{H}_{t-1}]\begin{bmatrix} \boldsymbol{W}_{xh} \\ \boldsymbol{W}_{hh} \end{bmatrix} + \boldsymbol{b}_h)$$
$$\boldsymbol{O}_t = \boldsymbol{H}_t \boldsymbol{W}_{hq} + \boldsymbol{b}_q$$

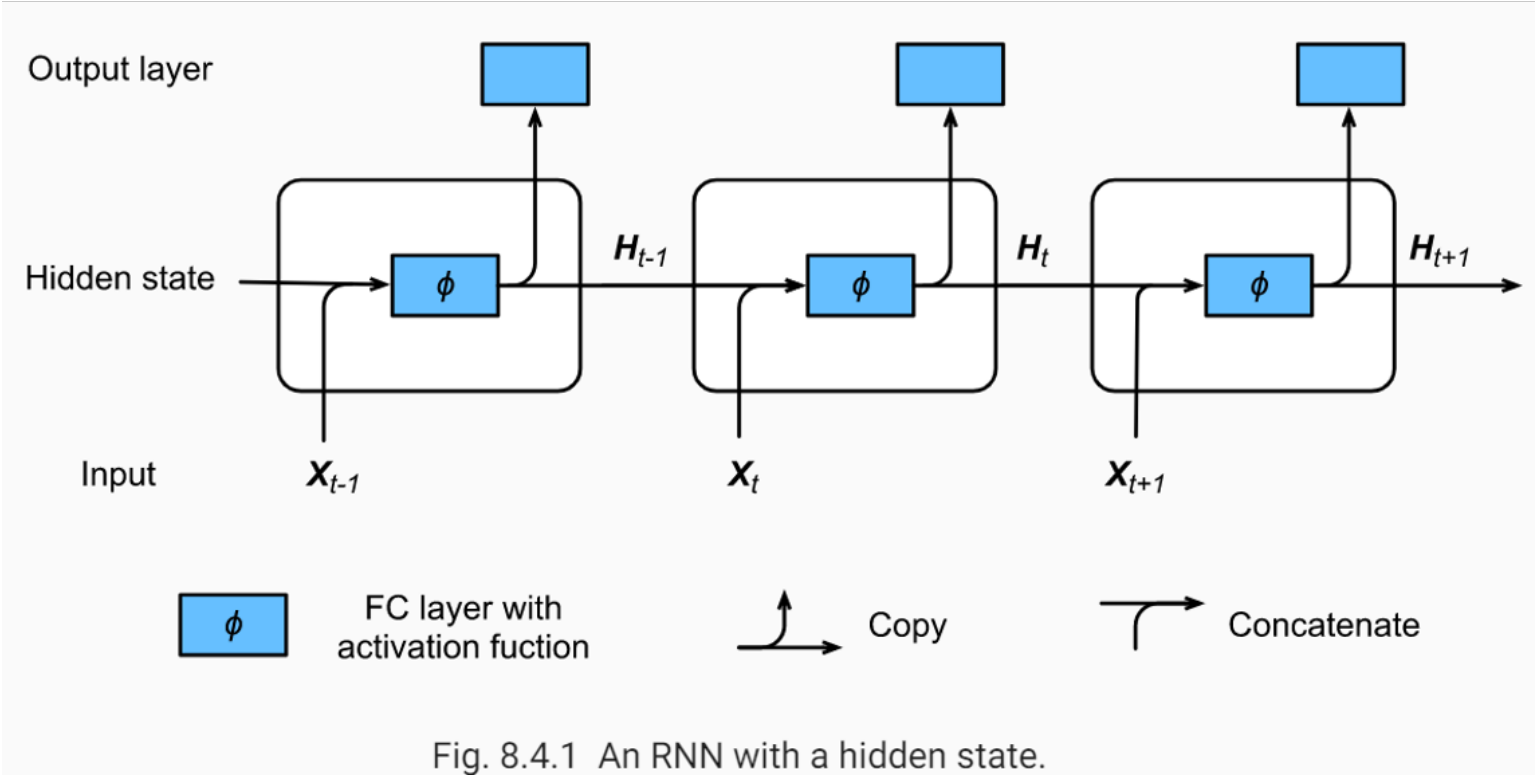


Fig. 8.4.1 An RNN with a hidden state.

Apprentissage des paramètres d'un RNN

BPTT : rétro-propagation du gradient au cours du temps

$\Rightarrow$ 2 étapes clés :

1. calculer les dérivés de la fonction de coût par rapport à chaque paramètre du modèle.
2. mettre à jour les paramètres dans la direction opposée du gradient pour minimiser la fonction de coût.

Apprentissage des paramètres d'un RNN

BPTT : rétro-propagation du gradient au cours du temps

$\Rightarrow$ 2 étapes clés :

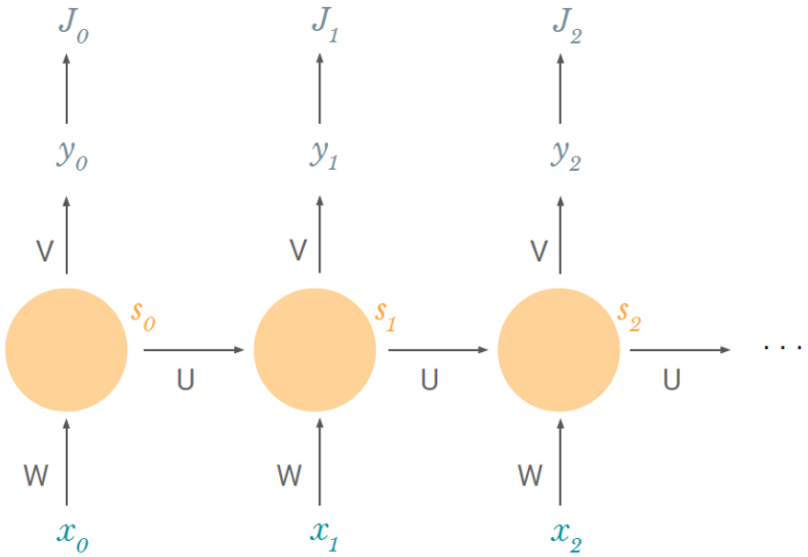
1. calculer les dérivés de la fonction de coût par rapport à chaque paramètre du modèle.
2. mettre à jour les paramètres dans la direction opposée du gradient pour minimiser la fonction de coût.

En pratique :

- On calcule la fonction de coût à chaque temps t
- On somme pour les différents temps t pour avoir le coût total
- Pour chaque paramètre, on calcule et accumule les gradients obtenus pour chaque temps t

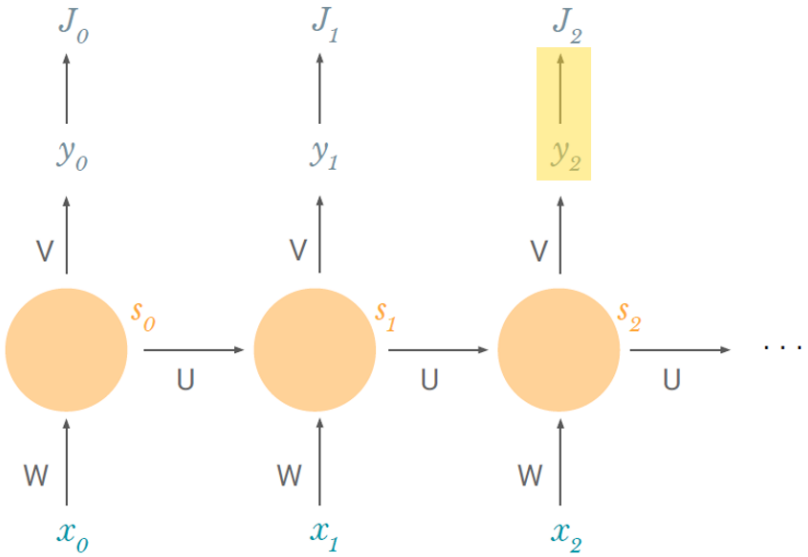
L'étape de rétro-propagation du gradient est plus compliquée à cause de la récurrence

Apprentissage des paramètres d'un RNN [2]



2. Adaptée de http ://introtodeeplearning.com/2017/index.html

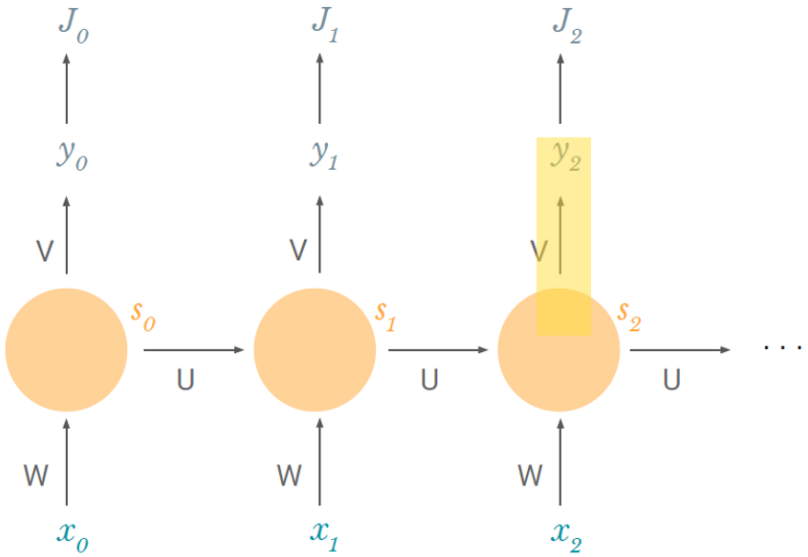
Apprentissage des paramètres d'un RNN [2]



$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}$$

2. Adaptée de http ://introtodeeplearning.com/2017/index.html

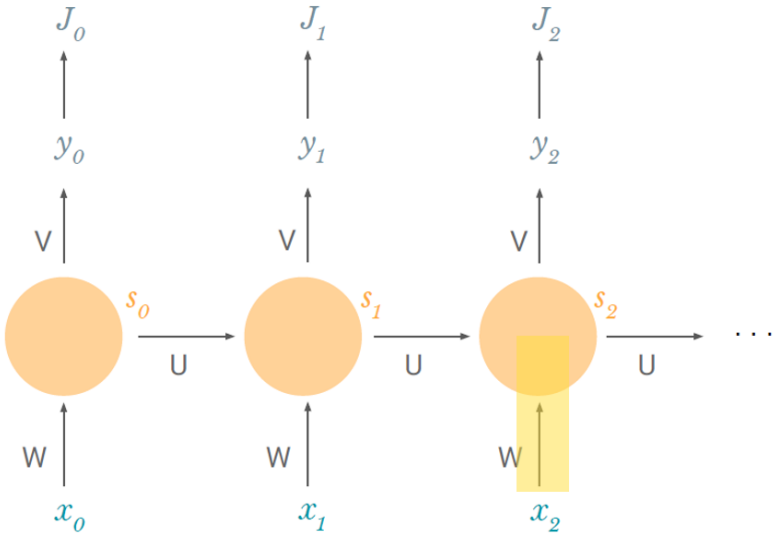
Apprentissage des paramètres d'un RNN [2]



$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial s_2}$$

2. Adaptée de http ://introtodeeplearning.com/2017/index.html

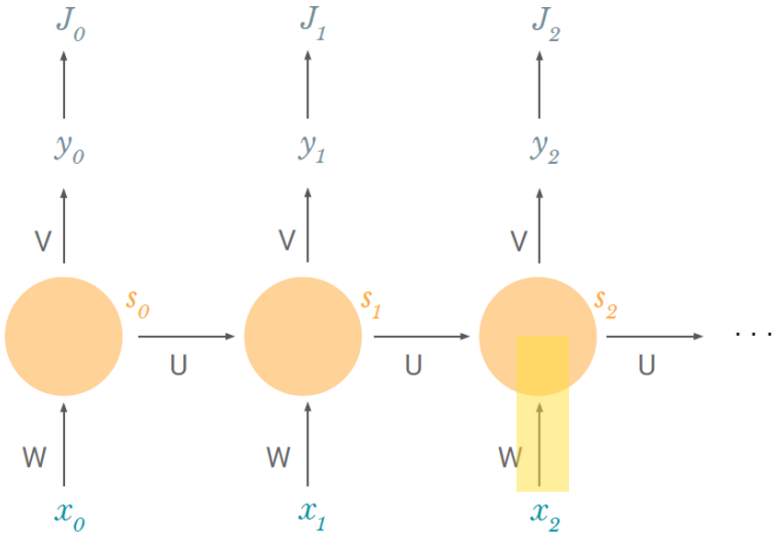
Apprentissage des paramètres d'un RNN [2]



$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial s_2}\frac{\partial s_2}{\partial W}$$

2. Adaptée de http ://introtodeeplearning.com/2017/index.html

Apprentissage des paramètres d'un RNN [2]

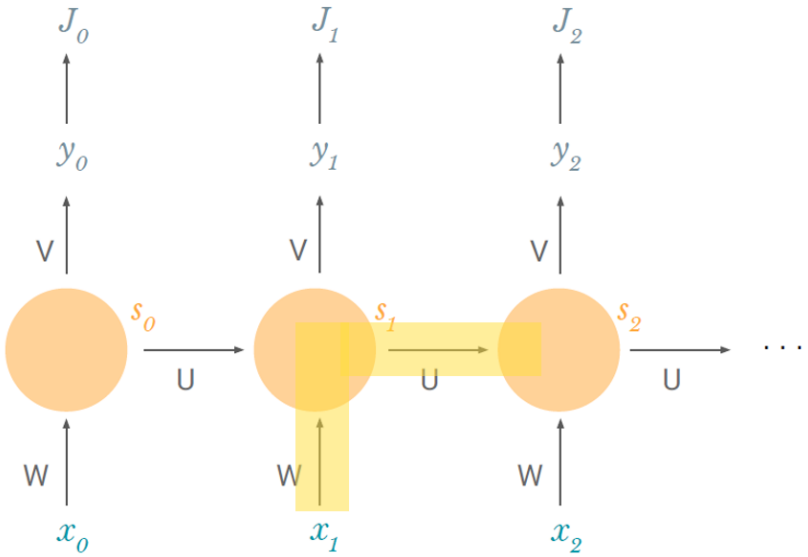


$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial s_2}\frac{\partial s_2}{\partial W}$$

$$s_2 = tanh(U s_1 + W x_2)$$

2. Adaptée de http ://introtodeeplearning.com/2017/index.html

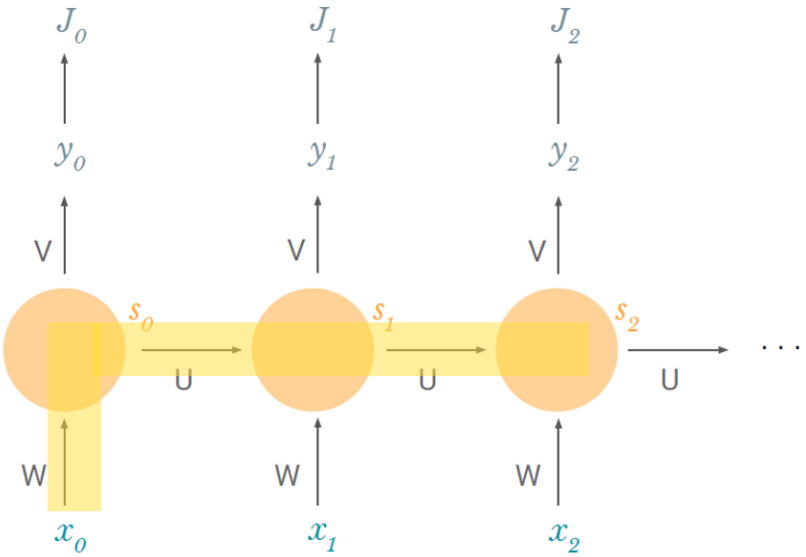
Apprentissage des paramètres d'un RNN [2]



$$\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W}$$

2. Adaptée de http ://introtodeeplearning.com/2017/index.html

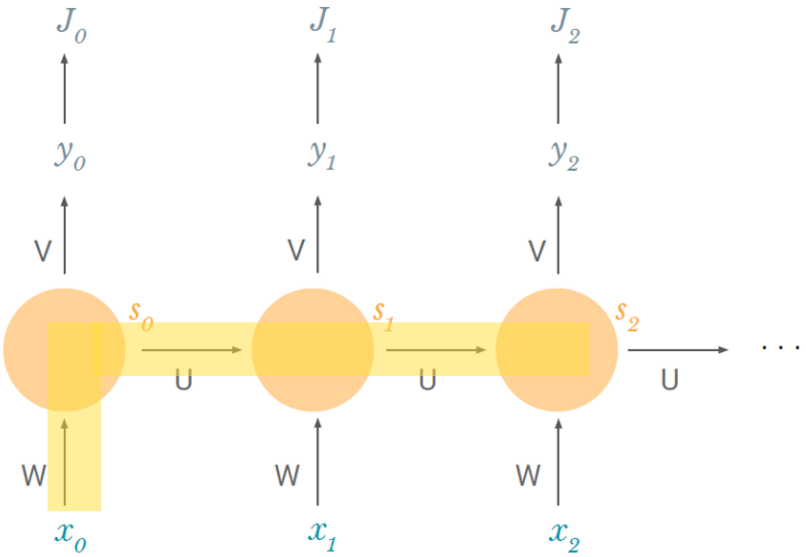
Apprentissage des paramètres d'un RNN [2]



$$\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial s_2}{\partial s_0}\frac{\partial s_0}{\partial W}$$

2. Adaptée de http ://introtodeeplearning.com/2017/index.html

Apprentissage des paramètres d'un RNN [2]



$$\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W} + \frac{\partial s_2}{\partial s_0}\frac{\partial s_0}{\partial W}$$

$$\frac{\partial s_n}{\partial s_{n-1}} = W^T diag[f'(W_{s_{j-1}} + Ux_j)]$$

$\Rightarrow$ On fait plusieurs multiplications de ce type de termes

$\Rightarrow$ Plus on revient dans le temps, plus on fait de multiplications, plus le gradient diminue

$\Rightarrow$ On se focalise sur des dépendances court-terme

2. Adaptée de http ://introtodeeplearning.com/2017/index.html

Les limites des réseaux de neurones récurrents

L'apprentissage des réseaux récurrents est complexe :

Les limites des réseaux de neurones récurrents

L'apprentissage des réseaux récurrents est complexe :

- Problèmes d'exploding gradients :
  Multiplication de valeurs $>1 \Rightarrow$ Les gradients $\rightarrow \infty$

Les limites des réseaux de neurones récurrents

L'apprentissage des réseaux récurrents est complexe :

- Problèmes d'exploding gradients :
  Multiplication de valeurs $>1 \Rightarrow$ Les gradients $\rightarrow \infty$

$\rightarrow$ Tronquer la récurrence et borner la norme du gradient

Les limites des réseaux de neurones récurrents

L'apprentissage des réseaux récurrents est complexe :

- Problèmes d'exploding gradients :
  Multiplication de valeurs $>1 \Rightarrow$ Les gradients $\rightarrow \infty$

$\rightarrow$ Tronquer la récurrence et borner la norme du gradient

- Problèmes de vanishing gradients :
  Multiplication de valeurs $< 1 \Rightarrow$ Les gradients $\rightarrow 0$

Les limites des réseaux de neurones récurrents

L'apprentissage des réseaux récurrents est complexe :

- Problèmes d'exploding gradients :
  Multiplication de valeurs $>1 \Rightarrow$ Les gradients $\rightarrow \infty$

$\rightarrow$ Tronquer la récurrence et borner la norme du gradient

- Problèmes de vanishing gradients :
  Multiplication de valeurs $< 1 \Rightarrow$ Les gradients $\rightarrow 0$

$\rightarrow$ Tronquer la récurrence, choix de la fonction d'activation, Initialisation des poids et changement d'architecture

Les limites des réseaux de neurones récurrents

L'apprentissage des réseaux récurrents est complexe :

- Problèmes d'exploding gradients :
  Multiplication de valeurs $>1 \Rightarrow$ Les gradients $\rightarrow \infty$

$\rightarrow$ Tronquer la récurrence et borner la norme du gradient

- Problèmes de vanishing gradients :
  Multiplication de valeurs $< 1 \Rightarrow$ Les gradients $\rightarrow 0$

$\rightarrow$ Tronquer la récurrence, choix de la fonction d'activation, Initialisation des poids et changement d'architecture

- Instabilité numérique

Les limites des réseaux de neurones récurrents

L'apprentissage des réseaux récurrents est complexe :

- Problèmes d'exploding gradients :
  Multiplication de valeurs $>1 \Rightarrow$ Les gradients $\rightarrow \infty$

$\rightarrow$ Tronquer la récurrence et borner la norme du gradient

- Problèmes de vanishing gradients :
  Multiplication de valeurs $< 1 \Rightarrow$ Les gradients $\rightarrow 0$

$\rightarrow$ Tronquer la récurrence, choix de la fonction d'activation, Initialisation des poids et changement d'architecture

- Instabilité numérique

$\rightarrow$ Tronquer la récurrence

Architectures modernes de RNNs

Finalement, un RNN se résume à :

$$\boldsymbol{H}_t = \Phi(\boldsymbol{X}_t \boldsymbol{W}_{x,h} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{h,h} + \boldsymbol{b}_h)$$

$$\boldsymbol{O}_t = \boldsymbol{H}_t \boldsymbol{W}_{h,q} + \boldsymbol{b}_q$$

Architectures modernes de RNNs

Finalement, un RNN se résume à :

$$\boldsymbol{H}_t = \Phi(\boldsymbol{X}_t \boldsymbol{W}_{x,h} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{h,h} + \boldsymbol{b}_h)$$

$$\boldsymbol{O}_t = \boldsymbol{H}_t \boldsymbol{W}_{h,q} + \boldsymbol{b}_q$$

...mais la mémoire $\boldsymbol{W}_{h,h}$ est figée !

Architectures modernes de RNNs

Finalement, un RNN se résume à :

$$\boldsymbol{H}_t = \Phi(\boldsymbol{X}_t \boldsymbol{W}_{x,h} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{h,h} + \boldsymbol{b}_h)$$

$$\boldsymbol{O}_t = \boldsymbol{H}_t \boldsymbol{W}_{h,q} + \boldsymbol{b}_q$$

...mais la mémoire $\boldsymbol{W}_{h,h}$ est figée !

- certaines observations du passé sont très importantes
  $\Rightarrow$ à garder absolument
- certaines observations du passé sont inutiles
  $\Rightarrow$ à supprimer
- le contenu de l'état caché peut devenir obsolète
  $\Rightarrow$ remise à zéro

Architectures modernes de RNNs

Finalement, un RNN se résume à :

$$\boldsymbol{H}_t = \Phi(\boldsymbol{X}_t \boldsymbol{W}_{x,h} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{h,h} + \boldsymbol{b}_h)$$

$$\boldsymbol{O}_t = \boldsymbol{H}_t \boldsymbol{W}_{h,q} + \boldsymbol{b}_q$$

...mais la mémoire $\boldsymbol{W}_{h,h}$ est figée !

- certaines observations du passé sont très importantes
  $\Rightarrow$ à garder absolument
- certaines observations du passé sont inutiles
  $\Rightarrow$ à supprimer
- le contenu de l'état caché peut devenir obsolète
  $\Rightarrow$ remise à zéro

$\Rightarrow$ Introduction de mécanismes de "gating" avec les :

GRUs et LSTMs

Architectures

- Bag-of-words & profils de k-mers
- CNNs & détection de motifs
- Réseaux récurrents, GRU & LSTM
  - Réseaux de neurones récurrents
  - GRU : Gated Recurrent Units
  - LSTM : Long Short Term Memory
- Transformers

GRU : Gated Recurrent Unit

Architecture introduite en 2014 (Cho et al., 2014) :

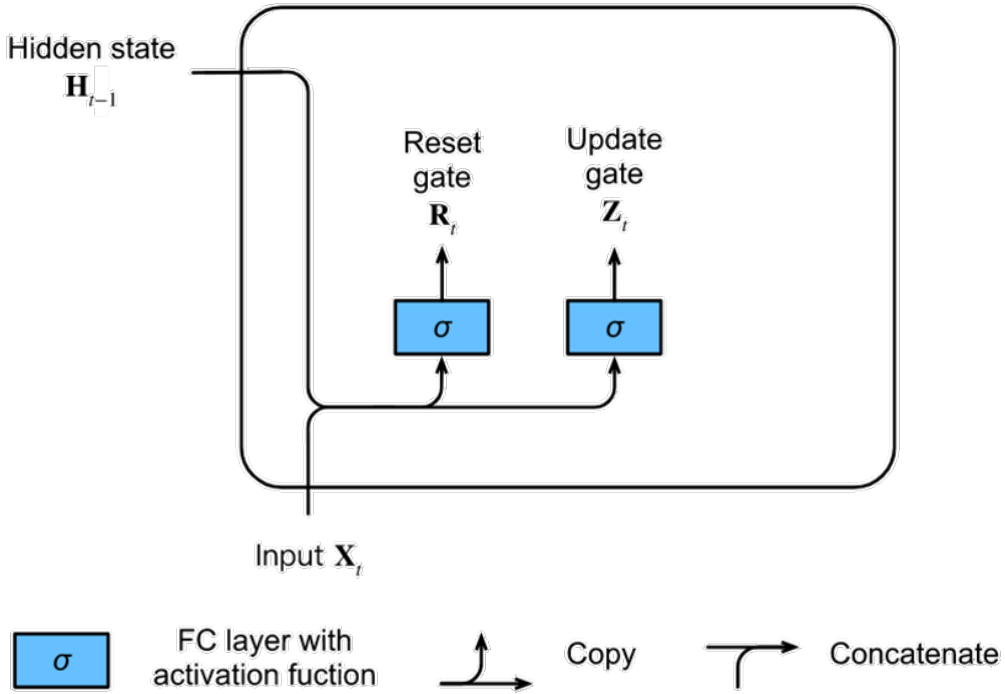


Fig. 9.1.1 Reset and update gate in a GRU.

- portes Reset et Update :

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z).$$

GRU : Gated Recurrent Unit

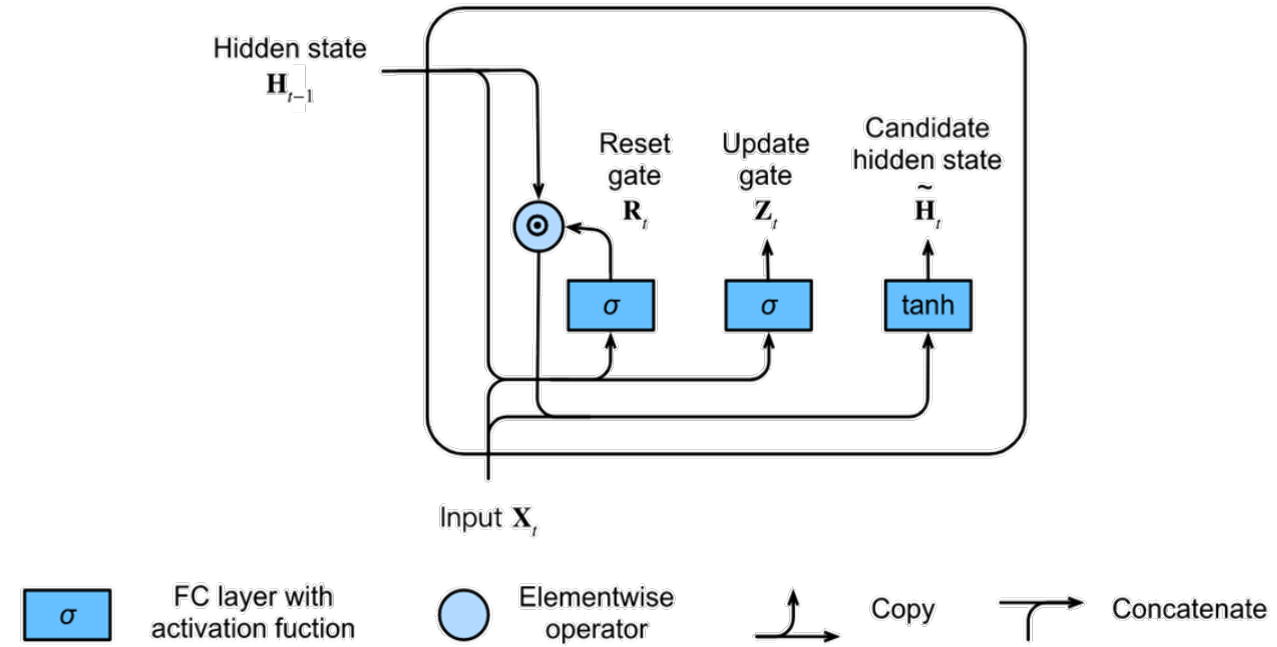


Fig. 9.1.2 Candidate hidden state computation in a GRU. The multiplication is carried out elementwise.

- portes Reset et Update :

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z).$$

- Etat caché candidat :

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h)$$

GRU : Gated Recurrent Unit

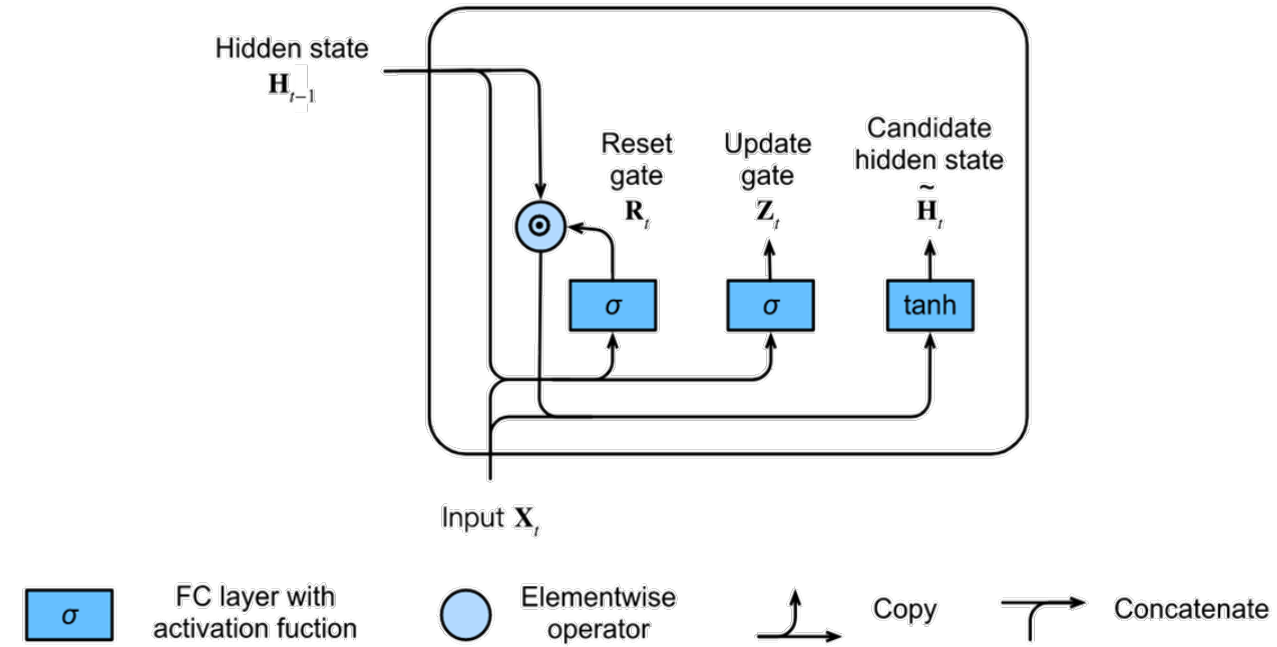


Fig. 9.1.2 Candidate hidden state computation in a GRU. The multiplication is carried out elementwise.

- portes Reset et Update :

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z).$$

- Etat caché candidat :

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h)$$

⇒ rappel pour un RNN :

$$\mathbf{H}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h)$$

GRU : Gated Recurrent Unit

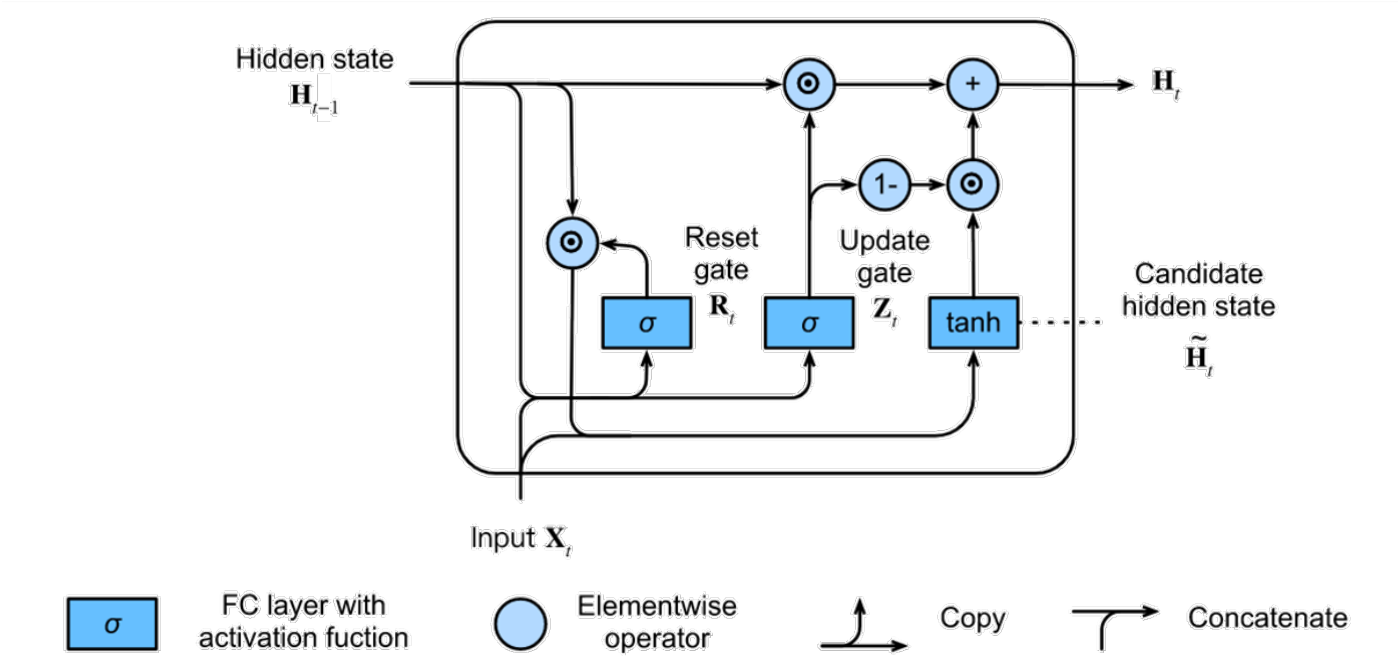


Fig. 9.1.3 Hidden state computation in a GRU. As before, the multiplication is carried out elementwise.

- portes Reset et Update :

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z).$$

- Etat caché candidat :

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h)$$

- Etat caché final :

$$\boxed{\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t}$$

Implémentation des GRUs sous Keras

La classe GRU existe déjà sous keras :

GRU class

```
tf.keras.layers.GRU(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
```

Implémentation des GRUs sous Keras

La classe GRU existe déjà sous keras :

GRU class

```
tf.keras.layers.GRU(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
```

Utilisation dans un exemple :

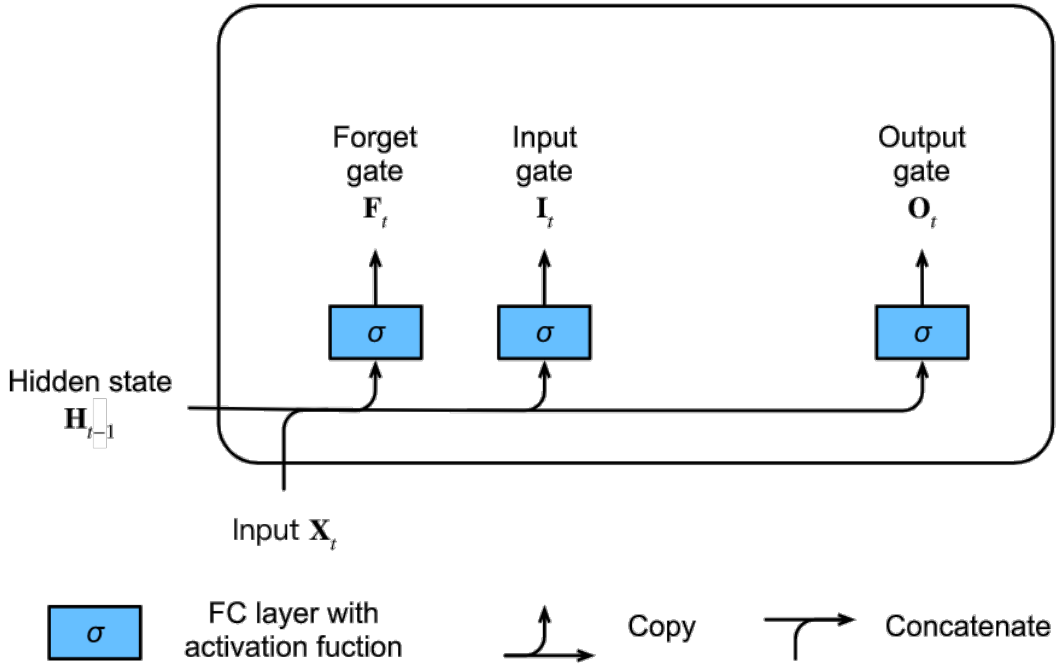
```
>>> inputs = tf.random.normal([32, 10, 8])
>>> gru = tf.keras.layers.GRU(4)
>>> output = gru(inputs)
>>> print(output.shape)
(32, 4)
>>> gru = tf.keras.layers.GRU(4, return_sequences=True, return_state=True)
>>> whole_sequence_output, final_state = gru(inputs)
>>> print(whole_sequence_output.shape)
(32, 10, 4)
>>> print(final_state.shape)
(32, 4)
```

Architectures

- Bag-of-words & profils de k-mers
- CNNs & détection de motifs
- Réseaux récurrents, GRU & LSTM
  - Réseaux de neurones récurrents
  - GRU : Gated Recurrent Units
  - LSTM : Long Short Term Memory
- Transformers

LSTM : Long Short Term Memory

Architecture introduite en 1997 [3] avec 3 mécanismes :

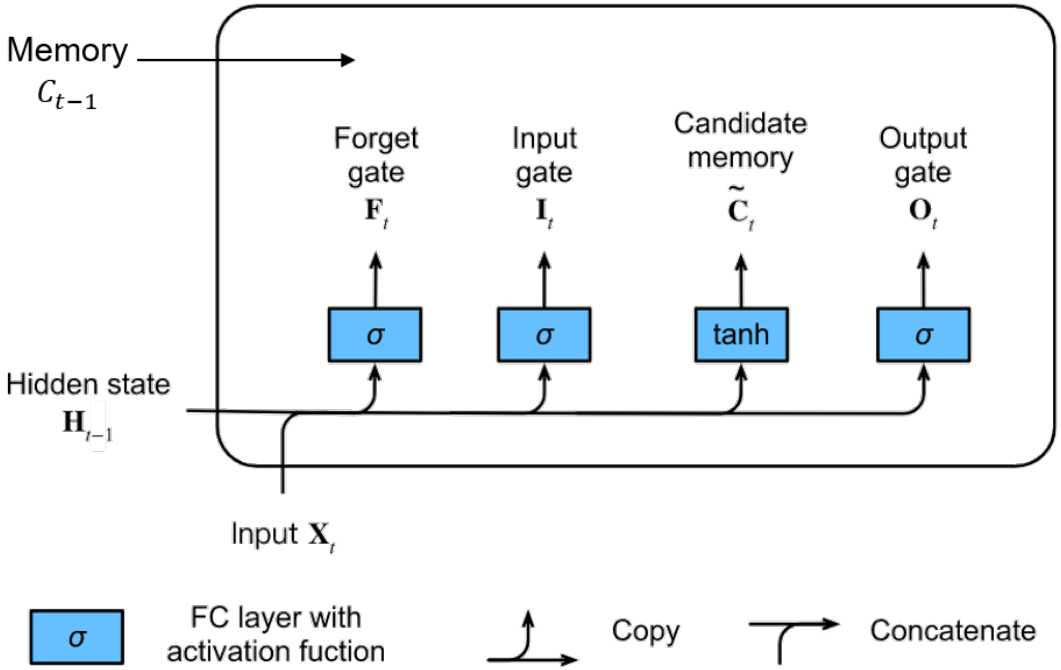


- Input / Forget / Output :

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i),$$
$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f),$$
$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o),$$

3. Hochreiter and Schmidhuber (1997)

LSTM : Long Short Term Memory



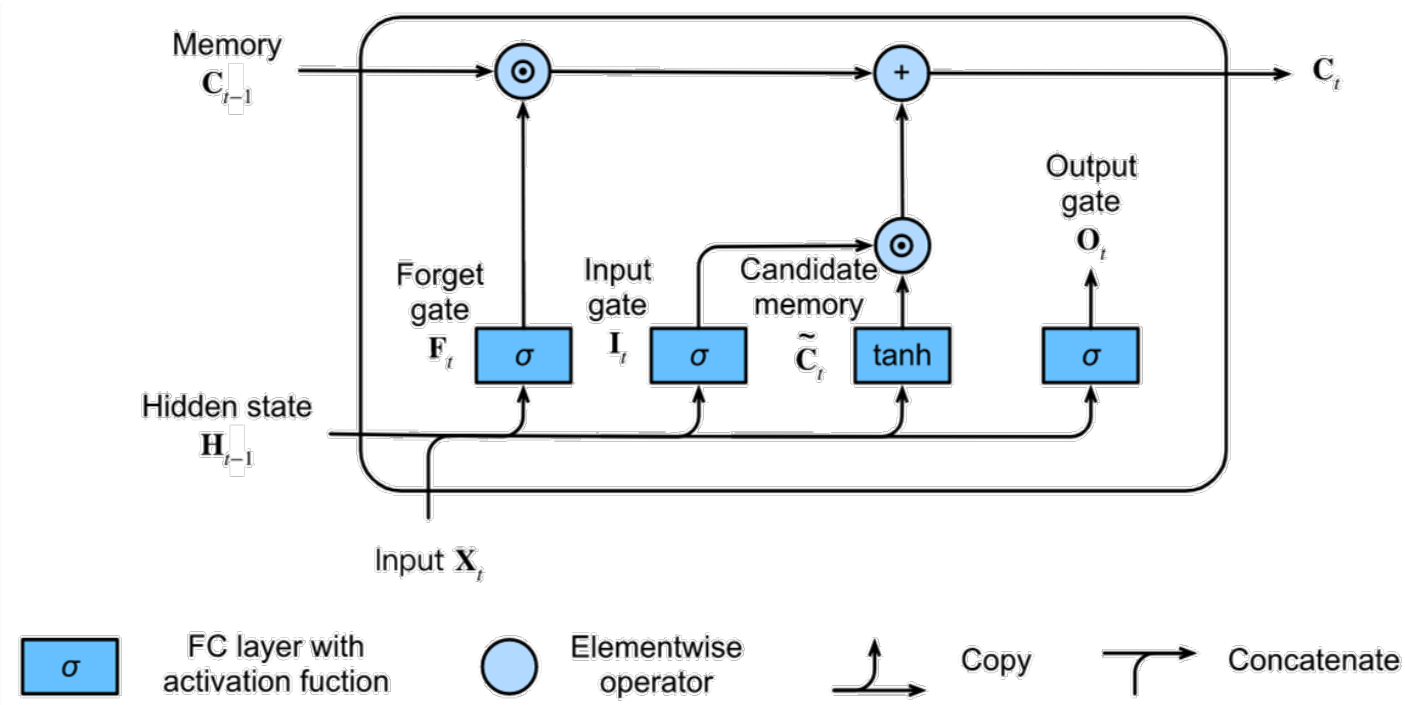
- Input / Forget / Output :

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i),$$
$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f),$$
$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o),$$

- Mémoire candidate :

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c)$$

LSTM : Long Short Term Memory



- Input / Forget / Output :

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i),$$
$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f),$$
$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o),$$

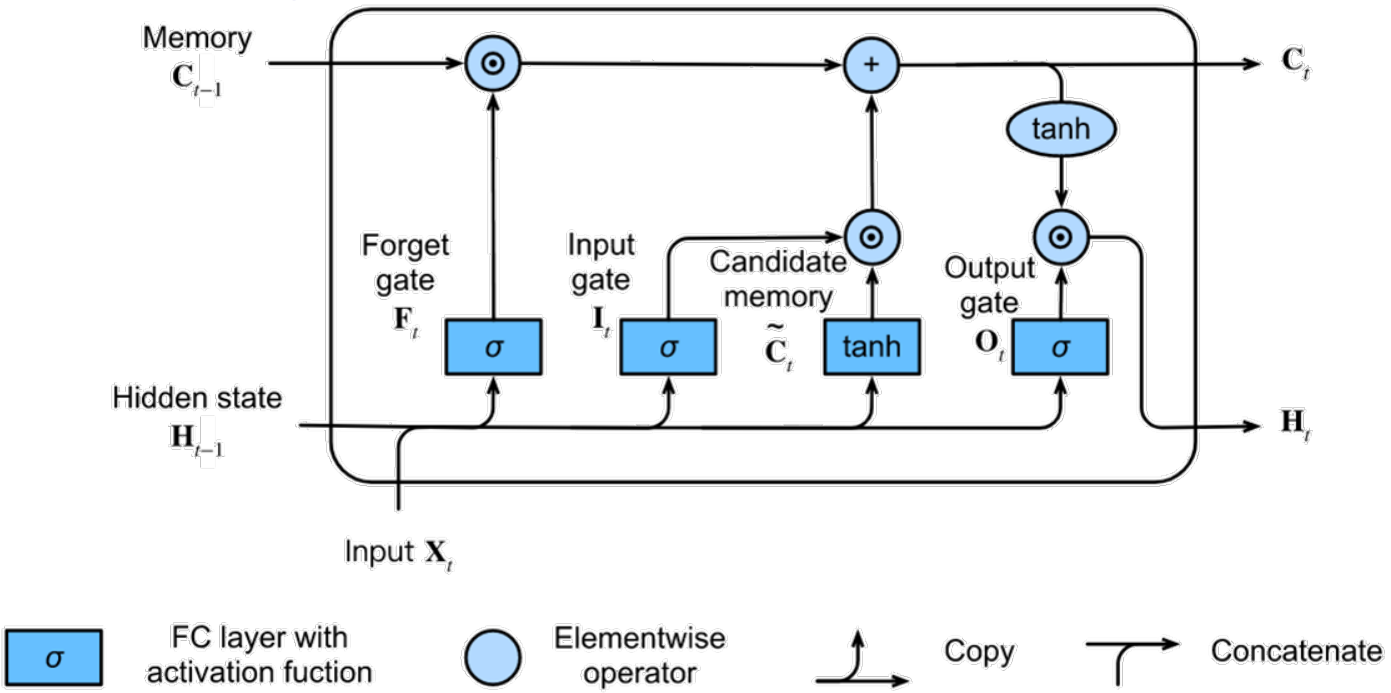
- Mémoire candidate :

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c)$$

- Mémoire finale :

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t$$

LSTM : Long Short Term Memory



- Input / Forget / Output :

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i),$$
$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f),$$
$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o),$$

- Mémoire candidate :

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c)$$

- Mémoire finale :

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t$$

- Etat caché final :

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t)$$

Implémentation des LSTM sous Keras

La classe LSTM existe déjà sous keras :

LSTM class

```
tf.keras.layers.LSTM(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    unit_forget_bias=True,
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
```

Implémentation des LSTM sous Keras

La classe LSTM existe déjà sous keras :

LSTM class

```
tf.keras.layers.LSTM(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    unit_forget_bias=True,
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
```

Utilisation dans un exemple :

```
>>> inputs = tf.random.normal([32, 10, 8])
>>> lstm = tf.keras.layers.LSTM(4)
>>> output = lstm(inputs)
>>> print(output.shape)
(32, 4)
>>> lstm = tf.keras.layers.LSTM(4, return_sequences=True, return_state=True)
>>> whole_seq_output, final_memory_state, final_carry_state = lstm(inputs)
>>> print(whole_seq_output.shape)
(32, 10, 4)
>>> print(final_memory_state.shape)
(32, 4)
>>> print(final_carry_state.shape)
(32, 4)
```

LSTM : Long Short Term Memory

On peut empiler plusieurs couches LSTMs :

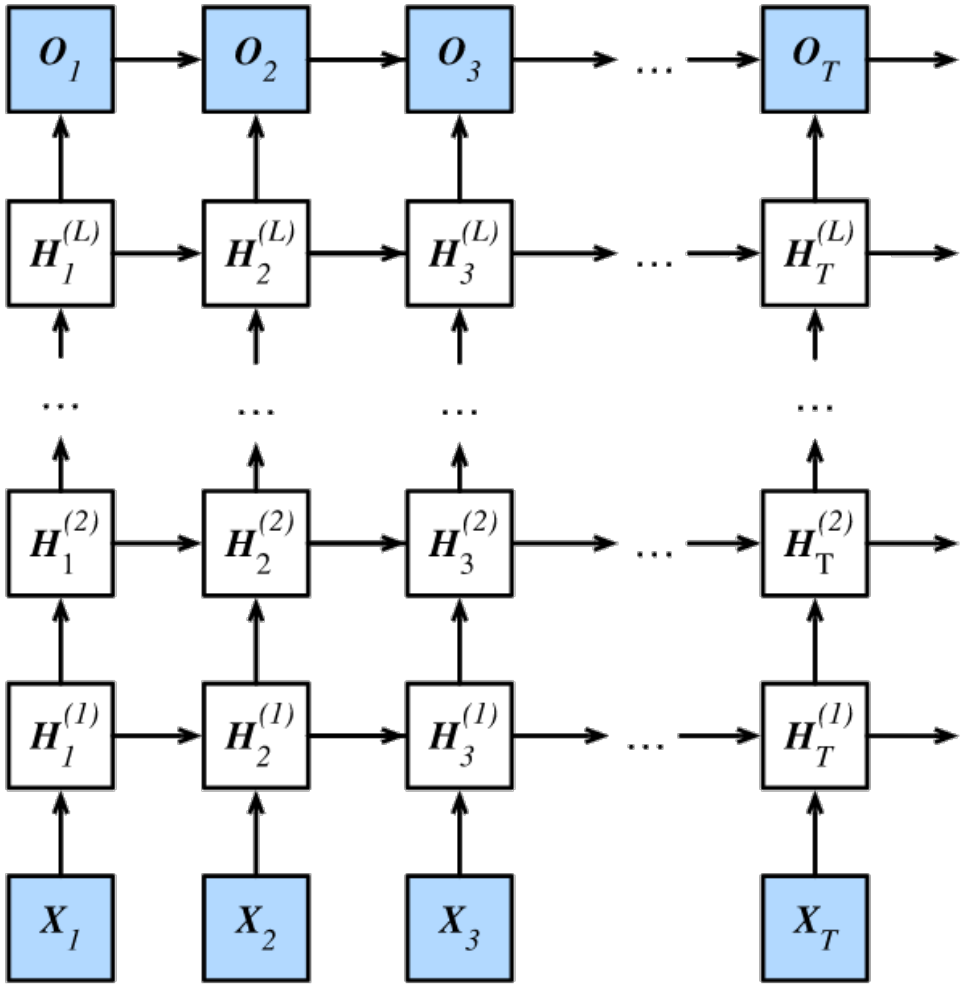


Fig. 9.3.1 Architecture of a deep recurrent neural network.

LSTM : Long Short Term Memory

On peut aussi faire des LSTM bi-directionels avec les deux sens de lecture des séquences :

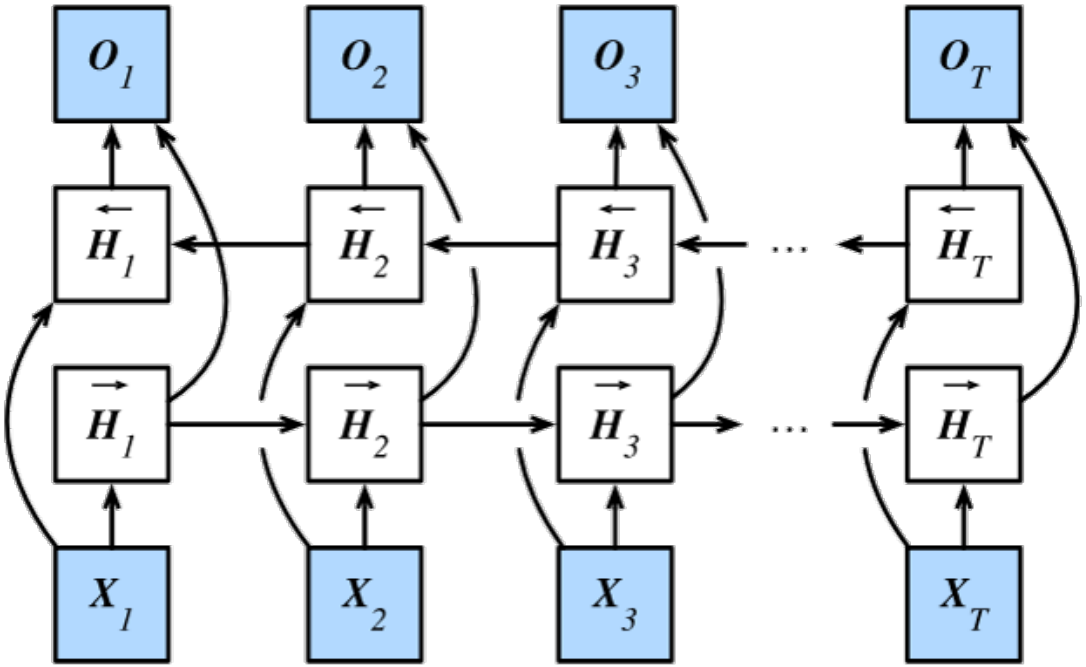
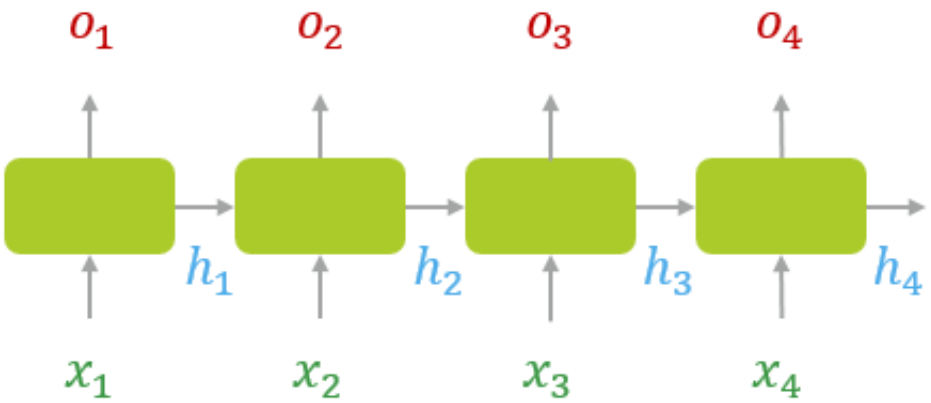


Fig. 9.4.2 Architecture of a bidirectional recurrent neural network.

- NB : si on a accès à toute la séquence en entrée...

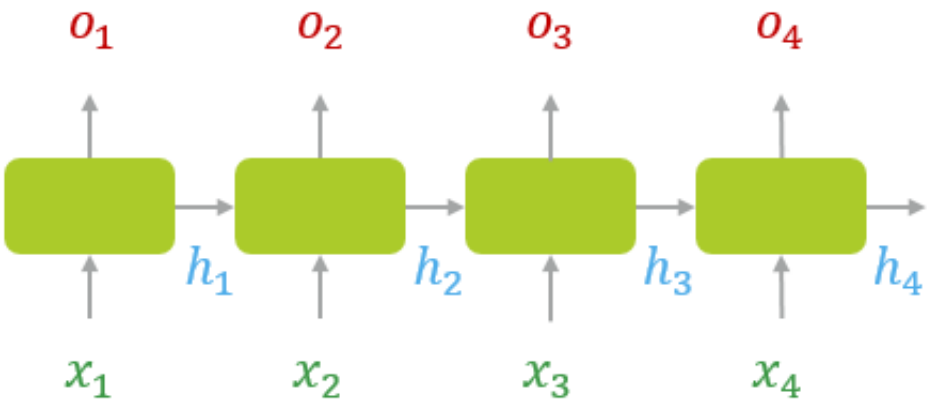
Utilisation dans un workflow global

Plusieurs sorties sont possibles : le dernier état caché, tous les états cachés, la dernière sortie, toutes les sorties ...



Utilisation dans un workflow global

Plusieurs sorties sont possibles : le dernier état caché, tous les états cachés, la dernière sortie, toutes les sorties ...



- En général, on se ramène à une dimension fixe pour pouvoir utiliser une couche de classification
- Plusieurs stratégies de pooling sont possibles : max pooling, average pooling ou pooling via un mécanisme d'attention

Conclusion

Conclusion

Deep-Learning et génomique : beaucoup d'intérêt dans la communauté scientifique

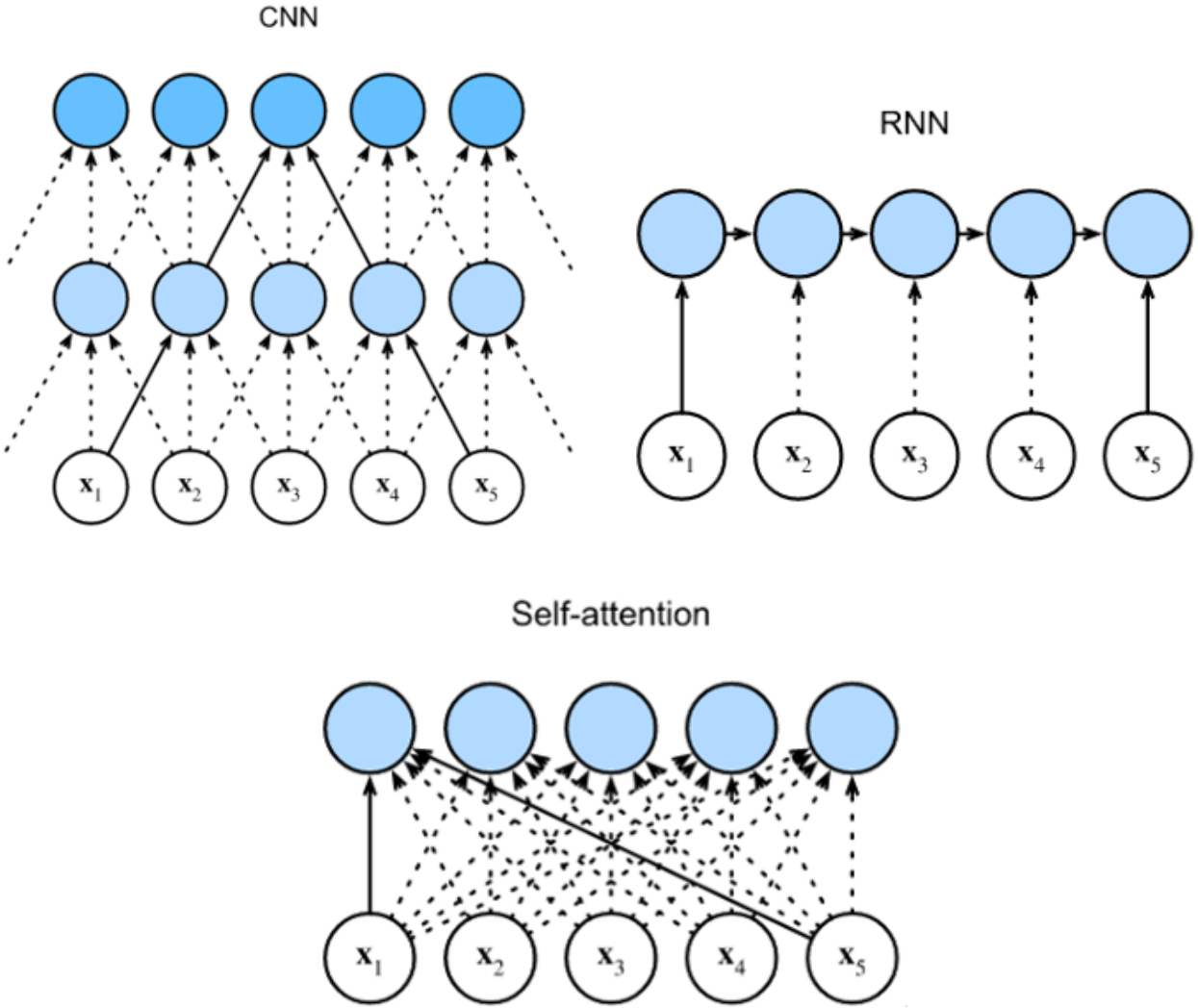
Ré-utilisation d'approches développées en NLP et imagerie :

- méthodes d'"embeddings" pour apprendre des représentation de k-mers et de séquences
- réseaux à convolution pour l'identification "end to end" de motifs génomiques ("probabilistes")
- réseaux récurrents pour tirer parti pleinement de l'information séquentielle (avec potentiellement une couche de convolution et/ou d'embedding en amont)

$\Rightarrow$ Perspective : mécanismes d'attention et transformers

Mécanismes d'attention

Comparaison des trois types d'architecture :



Mécanismes d'attention

- Entrée : une requête $\mathbf{q}$
- Mémoire : des couples de clés et valeurs $(\mathbf{k}_i, \mathbf{v}_i)_{i \in \{1...n\}}$
  1. Calculer la similarité entre l'entrée et les clés :

$$a_i = \alpha(\mathbf{q}, \mathbf{k}_i)$$

  2. Calculer les poids : $\mathbf{b} = softmax(\mathbf{a})$, avec $b_i = \frac{exp(a_i)}{\sum_j exp(a_j)}$
  3. Calculer la sortie : $\mathbf{o} = \sum_{i=1}^{n} b_i \mathbf{v}_i$

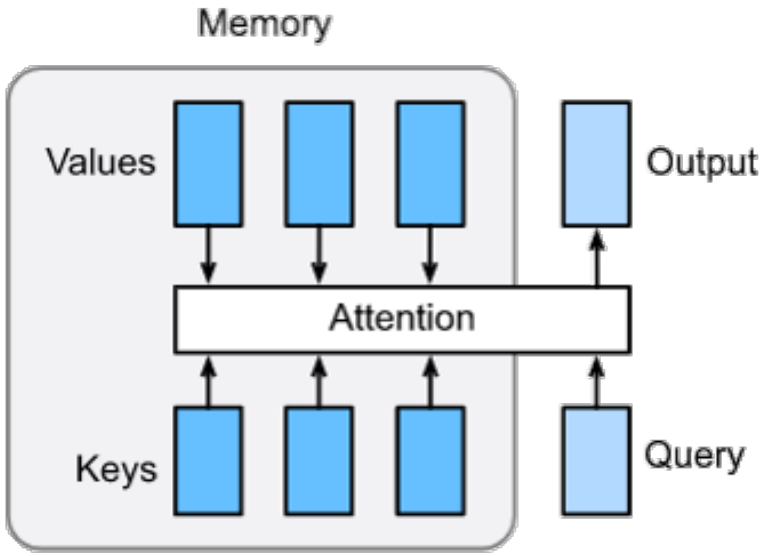
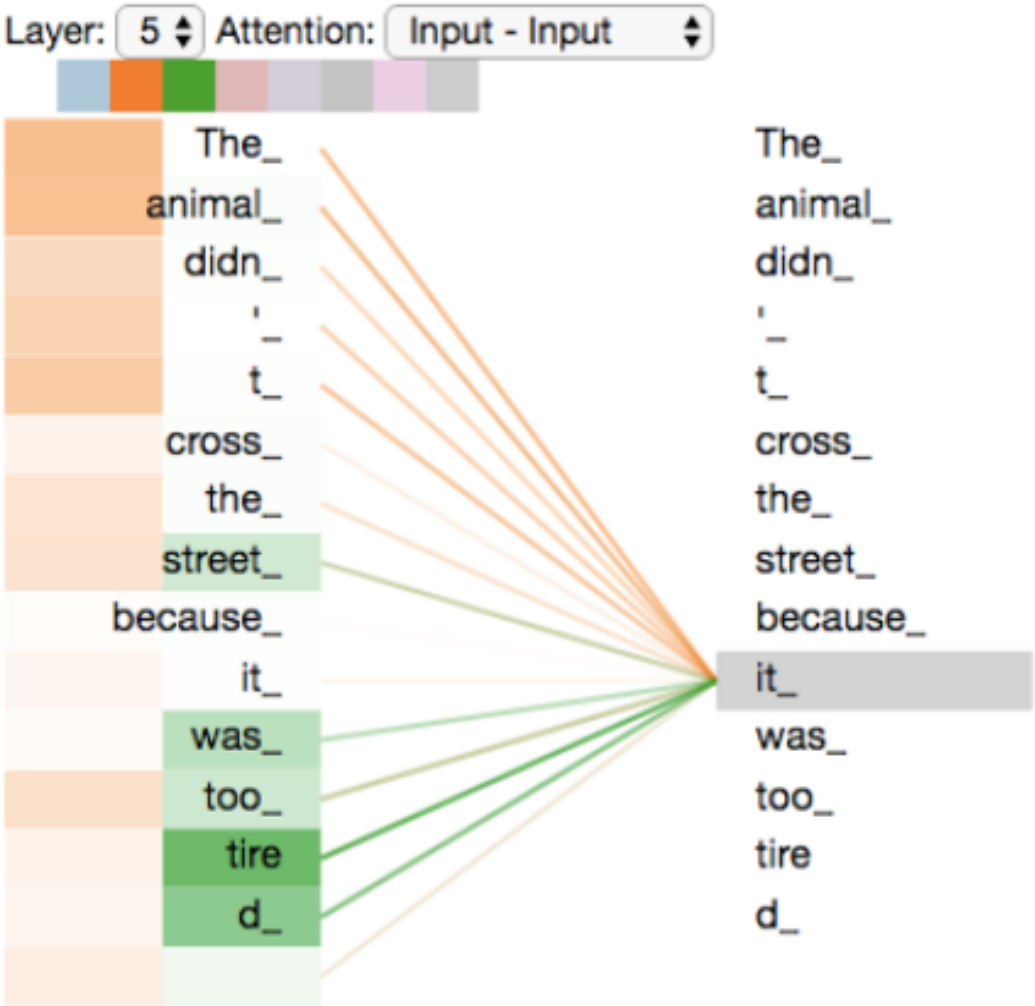


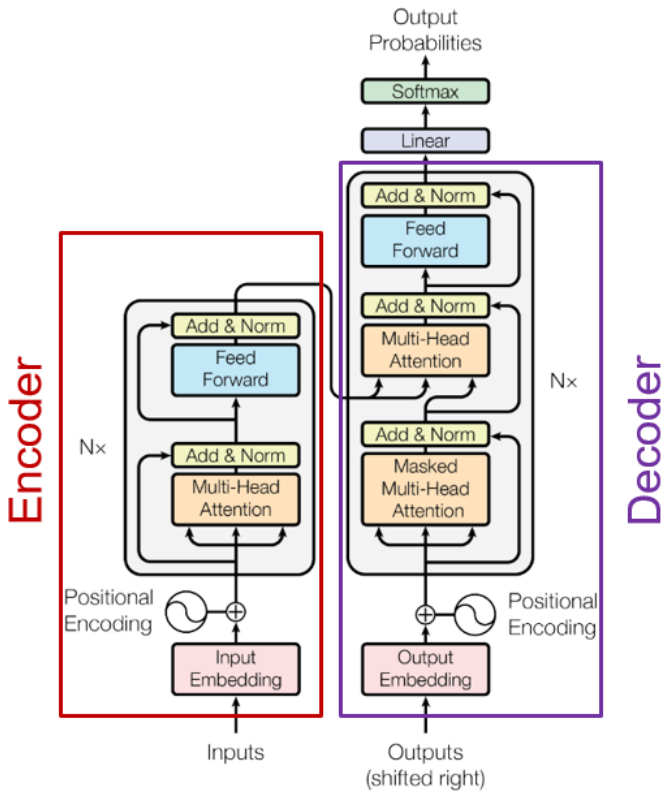
Illustration du mécanisme d'attention multi-têtes [4]

Visualisation des poids d'attention pour le mot "it" calculés dans les têtes d'attention 2 et 3



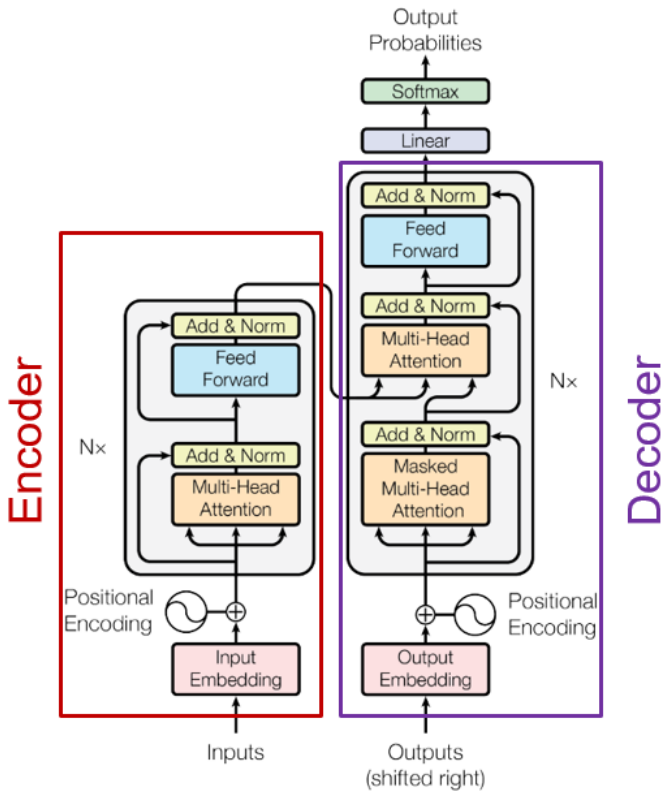
4. Vaswani et al. (2017).

Transformers



- Architecture introduite en 2017 (Vaswani et al., 2017)
- Architecture de type encodeur-décodeur avec différents mécanismes d'attention

Transformers



- Architecture introduite en 2017 (Vaswani et al., 2017)
- Architecture de type encodeur-décodeur avec différents mécanismes d'attention

Applications :

- Traitement du langage (traduction)
- Prédiction de séries temporelles
- Génération d'images
- Détection d'objets

Références

Babak Alipanahi, Andrew Delong, Matthew Weirauch, and Brendan Frey. Predicting the sequence specificities of dna- and rna-binding proteins by deep learning. *Nature biotechnology*, 33, 07 2015. doi : 10.1038/nbt.3300.

Maxwell L. Bileschi, David Belanger, Drew Bryant, Theo Sanderson, Brandon Carter, D. Sculley, Mark A. DePristo, and Lucy J. Colwell. Using Deep Learning to Annotate the Protein Universe. *bioRxiv*, pages 1–29, 2019. doi : 10.1101/626507.

David H. Brookes and Jennifer Listgarten. Design by adaptive sampling. oct 2018. URL http://arxiv.org/abs/1810.03714.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014. URL http://arxiv.org/abs/1406.1078. cite arxiv :1406.1078Comment : EMNLP 2014.

Nicki Skafte Detlefsen, Søren Hauberg, and Wouter Boomsma. What is a meaningful representation of protein sequences ?, 2021.

Gökcen Eraslan, Žiga Avsec, Julien Gagneur, and Fabian Theis. Deep learning : new computational modelling techniques for genomics. *Nature Reviews Genetics*, 20 :1, 04 2019. doi : 10.1038/s41576-019-0122-6.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9 (8) :1735–1780, 11 1997. ISSN 0899-7667. doi : 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Qiaoxing Liang, Paul W Bible, Yu Liu, Bin Zou, and Lai Wei. DeepMicrobes : taxonomic classification for metagenomics with deep learning. *NAR Genomics and Bioinformatics*, 2 (1) :1–13, 2020. doi : 10.1093/nargab/lqaa009.

Ameni Trabelsi, Mohamed Chaabane, and Asa Ben-Hur. Comprehensive evaluation of deep learning architectures for prediction of DNA/RNA sequence binding specificities. *Bioinformatics*, 35(14) :i269–i277, 07 2019. ISSN 1367-4803. doi : 10.1093/bioinformatics/btz339. URL https://doi.org/10.1093/bioinformatics/btz339.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.