

Mémoire de fin d'études
Pour l'obtention du diplôme d'Ingénieur d'État en Informatique
Option : Systèmes Informatiques

Création d'un corpus de l'aphasie de Broca et
développement d'un système Speech-to-speech de
réhabilitation de la parole

Réalisé par :
BELGOUMRI Mohammed
Djameleddine
im_belgoumri@esi.dz

Encadré par :
Pr. SMAILI Kamel
smaili@loria.fr
Dr. LANGLOIS David
david.langlois@loria.fr
Dr. ZAKARIA Chahnez
c_zakaria@esi.dz

Table des matières

Page de garde	i
Table des matières	ii
Table des figures	iii
Liste des algorithmes	iv
Sigles et abréviations	v
Introduction générale	1
1 Apprentissage séquence-à-séquence	3
1.1 Énoncé du problème	3
1.2 Perceptrons multicouches	4
1.2.1 Généralités	4
1.2.2 Application à la modélisation de séquence	5
1.2.3 Avantages et inconvénients	6
1.3 Architecture encodeur-décodeur	6
1.4 Réseaux de neurones récurrents	7
1.4.1 Réseaux de neurones récurrents simples	8
1.4.2 Portes, gated recurrent unit et long short-term memory . . .	10

1.5	Réseau de neurones à convolutions	12
1.6	Transformeurs	15
1.6.1	Architecture générale	15
1.6.2	Encodage positionnel	16
1.6.3	Couches attention	17
1.6.4	Autres éléments de l'architecture	19
1.6.5	Analyse comparative de la performance	19
1.7	Conclusion	20
	Conclusion générale	21
	Bibliographie	22

Table des figures

1.1	Architecture sous-jacente d'un MLP de profondeur 4.	5
1.2	Architecture encodeur-décodeur	7
1.3	RNN v.s FFN	8
1.4	Dépliage temporel d'un RNN sur une entrée de longueur 4. . . .	9
1.5	Dépliage temporel d'un encodeur-décodeur récurrent.	9
1.6	Forme générale d'un RNN à portes.	10
1.7	Architecture interne d'un GRU	11
1.8	Architecture interne d'un LSTM	12
1.9	Couche convolutive unidimensionnelle.	13
1.10	Recurrent Continuous Translation Model	14
1.11	Architecture de ByteNet.	14
1.12	Architecture de ConvS2S.	14
1.13	L'architecture de transformeur.	16
1.14	Matrice d'encodage de la position pour $r = 10^4$ et $d = 512$	17
1.15	Schéma d'une couche attention multitête.	18

Liste des algorithmes

1.1	Passe d'un MLP	5
1.2	Passe d'un RNN	7
1.3	Encodeur-décodeur récurrent.	10

Sigles et abréviations

ASR	reconnaissance automatique de la parole
AVC	accident vasculaire cérébral
BPTT	rétro-propagation dans le temps
CNN	réseau de neurones à convolutions
FFN	réseau de neurones feed-forward
GRU	gated recurrent unit
LSTM	long short-term memory
ML	apprentissage automatique
MLP	perceptron multicouches
MT	traduction automatique
NLP	traitement automatique du langage
RNN	réseau de neurones récurrent
S2S	séquence-à-séquence

Introduction générale

L'aphasie est un trouble linguistique qui complique un grand pourcentage d'accidents vasculaires cérébraux, une condition médicale qui touche plus de 12.2 millions de personnes par an. Ce chiffre est susceptible d'augmenter avec l'augmentation de l'espérance de vie (FEIGIN et al., 2022).

L'aphasie de Broca est une forme d'aphasie qui affecte la capacité de s'exprimer oralement ou par écrit. Elle résulte d'une lésion dans l'aire de Broca, une région du cerveau qui est responsable de la production de la parole. Les personnes qui ont l'aphasie de Broca ont des difficultés à produire des mots, mais peuvent comprendre ce qui est dit (CHAPEY, 2008).

Ces difficultés peuvent avoir des conséquences néfastes sur plusieurs aspects de la vie quotidienne. Ceci peut inclure la communication avec les proches, la participation à des activités sociales, l'exercice d'un emploi ou même la demande d'aide en cas d'urgence (HALLOWELL, 2017).

L'utilisation de techniques d'apprentissage automatique et du traitement automatique du langage au bénéfice des personnes atteintes de l'aphasie est une piste de recherche qui a capturé l'attention de plusieurs chercheurs (MISRA et al., 2022; QIN et al., 2022; SMAÏLI et al., 2022).

Dans ce mémoire, notre objectif est de fournir une revue de l'état de l'art sur les travaux qui ont été faits dans cette direction. Nous portons une attention particulière à la traduction automatique et à la reconnaissance automatique de la parole appliquées à l'aphasie de Broca.

Pour ce faire, nous avons organisé notre travail en trois chapitres :

1. ??.

Dans ce chapitre, nous présentons en général les domaines de recherche qui nous intéressent. À cette fin, le chapitre est divisé en trois sections :

- (1) La première section présente l'aphasie de Broca,
- (2) la deuxième introduit la traduction automatique
- (3) et la troisième présente la reconnaissance automatique de la parole.

2. Apprentissage séquence-à-séquence.

Ce chapitre sert à familiariser le lecteur avec le cadre d'étude général dans lequel s'inscrivent la traduction automatique, la reconnaissance automatique de la parole

et la majorité des tâches de traitement automatique du langage. Il s'agit de la modélisation de séquences. Nous y présentons l'énoncé du problème et les différentes architectures neuronales qui ont été proposées pour le résoudre.

3. ??.

Ce chapitre part de l'étude générale faite dans le chapitre 1. Il détaille l'application des meilleures architectures neuronales qui y sont présentées dans le cadre de la traduction automatique et de la reconnaissance automatique de la parole.

Chapitre 1

Apprentissage séquence-à-séquence

Les modèles séquence-à-séquence (S2S) sont une famille d’algorithmes d’apprentissage automatique (ML, de l’anglais : machine learning) dont l’entrée et la sortie sont des séquences (MARTINS, 2018). Plusieurs tâches de ML, notamment en traitement automatique du langage (NLP, de l’anglais : natural language processing), peuvent être formulées comme tâches d’apprentissage S2S. Parmi ces tâches, nous citons : la création de chatbots, la réponse aux questions et — crucialement pour ce travail — la traduction automatique (MT, de l’anglais : machine translation) et la reconnaissance automatique de la parole (ASR, de l’anglais : automatic speech recognition) (FATHI, 2021).

Dans ce chapitre, nous commençons par formuler le problème de modélisation de séquences. Ensuite, nous présentons les architectures neuronales les plus utilisées pour cette tâche. Enfin, nous concluons avec une étude comparative de celles-ci.

1.1 Énoncé du problème

Formellement, le problème de modélisation S2S est celui de calculer une fonction partielle $f : X^* \rightarrow Y^*$, où :

- X est un ensemble dit d’entrées.
- Y est un ensemble dit de sorties.
- Pour un ensemble A , $A^* = \bigcup_{n \in \mathbb{N}} A^n$ est l’ensemble de suites de longueur finie d’éléments de A .

f prend donc en entrée un $x = (x_1, x_2, \dots, x_n) \in X^n$ et renvoie un $y = (y_1, y_2, \dots, y_m) \in Y^m$. Dans le cas général, $n \neq m$ et aucune hypothèse d’alignement n’est admise. On prend souvent $X = \mathbb{R}^{d_e}$ et $Y = \mathbb{R}^{d_s}$ avec $d_e, d_s \in \mathbb{N}$. Dans ce cas, $x \in \mathbb{R}^{d_e \times n}$ et $y \in \mathbb{R}^{d_s \times m}$. Les indices peuvent représenter une succession temporelle ou un ordre plus abstrait comme celui des mots dans une phrase (MARTINS, 2018).

La majorité des outils mathématiques historiquement utilisés pour ce problème viennent de la théorie du traitement de signal numérique. Cependant, l’approche actuellement dominante et celle qui a fait preuve de plus de succès, est de les combiner avec les réseaux

de neurones.

1.2 Perceptrons multicouches

Les réseaux de neurones profonds sont parmi les modèles les plus expressifs en ML. Leur succès pratique est incomparable à ceux des modèles qui les ont précédés, que se soit en termes de qualité des résultats ou de variété de domaines d’application (SEBASTIAN & MIRJALILI, 2017). De plus, grâce aux théorèmes dits d’approximation universelle, ce succès empirique est formellement assuré (CALIN, 2020).

1.2.1 Généralités

Dans cette section, nous introduisons les perceptrons multicouches (MLP, de l’anglais : multi-layer perceptron), l’architecture neuronale la plus simple et la plus utilisée. Il s’agit d’une simple composition de couches affines avec des activations non affines (voir Définition 1).

Définition 1 (MLP).

Soient $k, w_0, w_1, \dots, w_{k+1} \in \mathbb{N}$, un réseau de neurones feed-forward de profondeur $k + 1$, à w_0 entrées et w_{k+1} sorties, est défini par une fonction :

$$\begin{cases} \mathbb{R}^{w_0} \rightarrow \mathbb{R}^{w_{k+1}} \\ x \mapsto \varphi_{k+1} \circ A_{k+1} \circ \varphi_k \circ A_k \circ \dots \circ \varphi_1 \circ A_1(x) \end{cases} \quad (1.1)$$

où les A_i sont des fonctions affines $\mathbb{R}^{w_{i-1}} \rightarrow \mathbb{R}^{w_i}$ et les φ_i sont des fonctions quelconques, typiquement non affines $\mathbb{R}^{w_i} \rightarrow \mathbb{R}^{w_i}$, dites *d’activations*. La fonction $\varphi_i \circ A_i$ est appelée la $i^{\text{ème}}$ couche du réseau (MUKHERJEE, 2021).

À un tel réseau de neurones, on peut associer un graphe orienté acyclique qu’on appelle son “architecture sous-jacente” (KEARNS & VAZIRANI, 1994). La Figure 1.1 illustre l’architecture d’un MLP de profondeur 4 avec $(w_0, w_1, w_2, w_3, w_4) = (4, 5, 7, 5, 4)$.

Deux MLP peuvent avoir la même architecture sous-jacente, en effet, cette dernière ne dépend que des dimensions de leurs couches respectives. De ce fait, une méthode de trouver, pour une architecture et une fonction cible données, le meilleur MLP est nécessaire (MUKHERJEE, 2021).

Pour ce faire, nous exploitons le fait que les A_i sont des applications affines sur des espaces de dimensions finies. Nous pouvons donc les écrire comme combinaisons de produits matriciels et de translations. Le problème se réduit donc à régler¹ les paramètres des matrices en question. Cela nécessite une façon de quantifier la qualité d’approximation d’une fonction f par une autre \hat{f} . L’analyse fonctionnelle nous en donne plusieurs. Les équations (1.2) et (1.3) sont deux exemples récurrents de fonctions dites *de perte*.

1. En ML, le terme “entraîner” est plutôt utilisé.

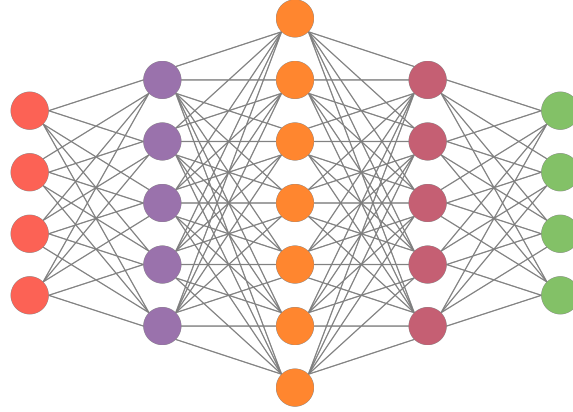


FIGURE 1.1 – Architecture sous-jacente d’un MLP de profondeur 4.

$$L_1(f, \hat{f}) = \|f - \hat{f}\|_1 = \int |f - \hat{f}| \quad (1.2)$$

$$L_2(f, \hat{f}) = \|f - \hat{f}\|_2^2 = \int |f - \hat{f}|^2 \quad (1.3)$$

Ayant fixé une fonction de perte L , l’entraînement revient à un problème d’optimisation comme représenté par l’équation 1.4.

$$f^* = \underset{\hat{f}}{\operatorname{argmin}} L(f, \hat{f}) \quad (1.4)$$

Dans le cas particulier où L est différentiable, l’algorithme du gradient peut être utilisé pour trouver un minimum local. Les gradients sont calculés en utilisant une méthode de dérivation automatique comme la rétro-propagation.

1.2.2 Application à la modélisation de séquence

Les réseaux de neurones opèrent sur des vecteurs. Afin de les utiliser dans le contexte de la modélisation S2S, il faut utiliser une représentation vectorielle des entrées. Une telle représentation s’appelle un *plongement* (embedding en anglais). Le plongement peut-être appris ou prédéfini (SEBASTIAN & MIRJALILI, 2017).

Dans le cas des MLP, la séquence d’entrée est d’abord décomposée en sous-séquences. Ensuite, les plongements de ces sous-séquences sont traités un par un par le réseau de neurones, ce qui produit une séquence de vecteurs en sortie. (voir Algorithme 1.1).

Algorithme 1.1 Passe d’un MLP

```

1 def mlp_s2s(xs: Sequence, k: int) -> Sequence:
2     ys = [] # Initialiser la sortie
3     for x in blocks(xs, k): # Parcourir les blocs
4         y = mlp(x) # Traiter chaque bloc

```

```

5     ys.append(y)    # Concatener les resultats
6     return ys

```

1.2.3 Avantages et inconvénients

Les MLP présentent deux avantages par rapport aux architectures discutées dans le reste de ce chapitre. Le premier est leur simplicité. Elle les rend plus simples à comprendre et à implémenter. Le deuxième est le fait qu'ils traitent indépendamment les sous-séquences. Cela rend très facile la tâche de les paralléliser, ce qui accélère considérablement leur entraînement.

Cependant, ce dernier point pose un grand problème. Comme ils traitent indépendamment les blocs de la séquence, les MLP ne peuvent pas modéliser les dépendances inter-blocs. Par conséquent, leur performance sur les séquences composées de plusieurs blocs est très médiocre. La solution de ce problème est d'augmenter la taille du bloc (et donc aussi la dimension d'entrée). Les MLP dépendent également d'une hypothèse d'alignement par blocs entre les deux séquences, une hypothèse invalide selon la Section 1.1.

1.3 Architecture encodeur–décodeur

Les lacunes des MLP sont en large partie le résultat du traitement séparé des parties des séquences. Dans la production de l'élément (ou bloc) courant de la sortie, un MLP se base uniquement sur l'élément (ou bloc) correspondant de l'entrée. L'équation 1.5 l'illustre pour une entrée $x = (x_1, x_2, \dots, x_n)$ et une sortie $y = (y_1, y_2, \dots, y_m)$.

$$y_j = f(x_i, x_{i+1}, \dots, x_{i+\ell}) \quad 1 \leq j \leq m \quad (1.5)$$

En plus de l'hypothèse implicite de l'existence d'une telle correspondance, cela suppose que les éléments d'une séquence sont complètement indépendants les uns des autres. Cette dernière hypothèse n'est presque jamais vérifiée.

Une façon naturelle de combler ces lacunes est d'abandonner le traitement par bloc de l'entrée. Tout élément de la séquence de sortie est considéré comme fonction de la séquence d'entrée toute entière. L'équation 1.6 montre cette approche sur l'exemple précédent (STAHLBERG, 2020).

$$y_j = f(x) = f(x_1, x_2, \dots, x_n) \quad 1 \leq j \leq m \quad (1.6)$$

L'architecture encodeur–décodeur y parvient en combinant deux modules : un encodeur et un décodeur. L'encodeur consomme la suite donnée en entrée et produit un vecteur de dimension fixe qui la représente². Le décodeur consomme ce vecteur et produit la sortie (voir Figure 1.2). L'équation 1.7 le montre sur le même exemple.

$$\begin{aligned} c &= \text{encoder}(x) \\ y &= \text{decoder}(c) \end{aligned} \quad (1.7)$$

2. Ce vecteur est appelé un vecteur de contexte, vecteur de pensée ou encore un encodage.

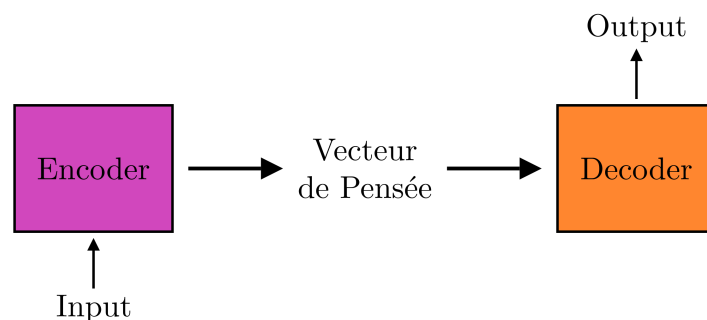


FIGURE 1.2 – Architecture encodeur-décodeur

L'équation 1.7 ne dépend ni du fonctionnement interne de l'encodeur ni de celui du décodeur. Les deux modules peuvent avoir deux architectures quelconques, qui peuvent ou non être les mêmes (YANG et al., 2020). Dans le reste de ce chapitre, nous examinons les architectures communes en apprentissage S2S.

1.4 Réseaux de neurones récurrents

L'un des principaux défauts que nous avons observés avec les MLP et qui nous ont poussés à introduire l'architecture encodeur-décodeur, est leur incapacité de représenter la dépendance entre les éléments d'une séquence. Une incapacité qui résulte de leur traitement indépendant des éléments.

Les réseaux de neurones récurrents (RNN, de l'anglais : recurrent neural network) tentent de résoudre ce problème en utilisant un état interne persistant. Chaque élément de la séquence modifie cet état lors de son traitement. Cela permet aux premiers éléments d'affecter le traitement des éléments qui les suivent, ainsi permettant à l'information de se propager vers le futur, ce qui donne lieu à une *mémoire*. Pour implémenter ce type

Algorithme 1.2 Passe d'un RNN

```

1 def rnn_pass(xs: Sequence, h=0) -> Vector:
2     for x in xs: # pour chaque element de la sequence
3         # traitement de l'element + mise a jour de l'etat
4         h = process(x, h)
5     return h # retourne l'etat final
  
```

de comportement, l'état d'un RNN est décalé d'une unité et réinjecté dans l'entrée (voir Figure 1.3a et Algorithme 1.2). Cela est très différent des MLP qui n'ont pas de boucle de rétroaction, il s'agit de réseaux de neurones feed-forward (FFN, de l'anglais : feed-forward network) (voir Figure 1.3b). Par conséquent, ils n'ont pas d'état ni de mémoire (FATHI, 2021).

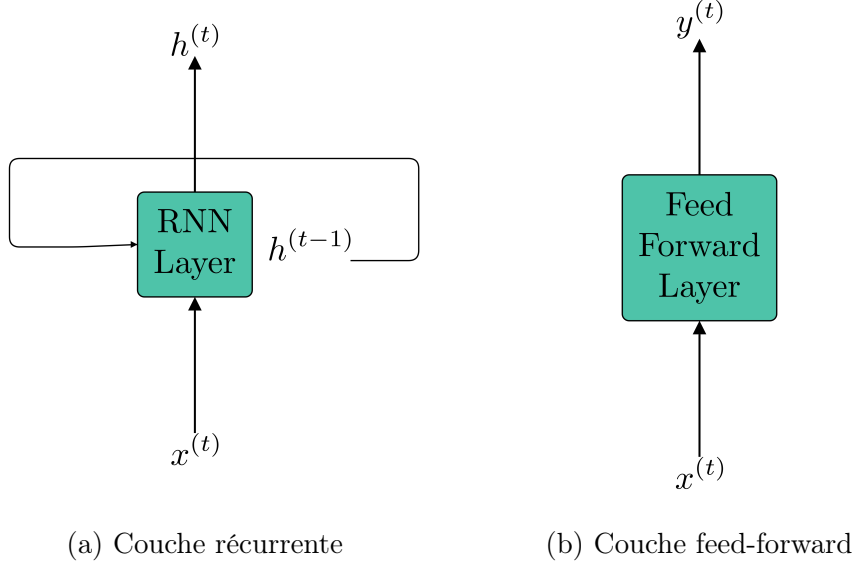


FIGURE 1.3 – RNN v.s FFN

1.4.1 Réseaux de neurones récurrents simples

Le RNN le plus simple à imaginer se réduit à une combinaison affine de l'état passé et de l'entrée. Il permet à l'information sur le passé de se propager sans contrôle particulier. Mathématiquement, une couche d'un tel RNN prend la forme suivante

$$h^{(t)} = \varphi \left(U h^{(t-1)} + W x^{(t)} + b \right) \quad 1 \leq t \leq n \quad (1.8)$$

où t est le temps³, $x^{(t)}, h^{(t)}$ sont respectivement l'entrée et la sortie à l'instant t , n est la longueur de la séquence et φ est la fonction d'activation (FATHI, 2021). Dans le cas où φ est l'identité, la transformée en z de l'équation 1.8 est donnée par

$$H(z) = z(zI - U)^{-1}(WX(Z) + b) \quad (1.9)$$

il s'agit donc d'un système à réponse impulsionnelle infinie (FATHI, 2021).

Dépliage temporel et encodeur–décodeur récurrent

Le traitement d'une séquence x par un RNN (\mathcal{R}), est équivalent à son traitement par un FFN (\mathcal{F}) dont la profondeur est égale à la longueur de x ⁴. \mathcal{F} est donc appelé *dépliage temporel* de \mathcal{R} pour x . La Figure 1.4 montre le dépliage temporel de la Figure 1.3a pour une séquence de longueur 4 (LECUN et al., 2015).

Chaque état caché $h^{(i)}$ contient de l'information sur tous les $x^{(j)}$, $j \leq i$. En particulier, $h^{(n)}$ contient de l'information sur toute la séquence x . Un encodeur récurrent peut donc retourner son dernier état caché comme vecteur d'encodage (YANG et al., 2020).

3. Il peut être continu ou discret, réel ou abstrait.

4. Et dont toutes les couches sont identiques à \mathcal{R} .

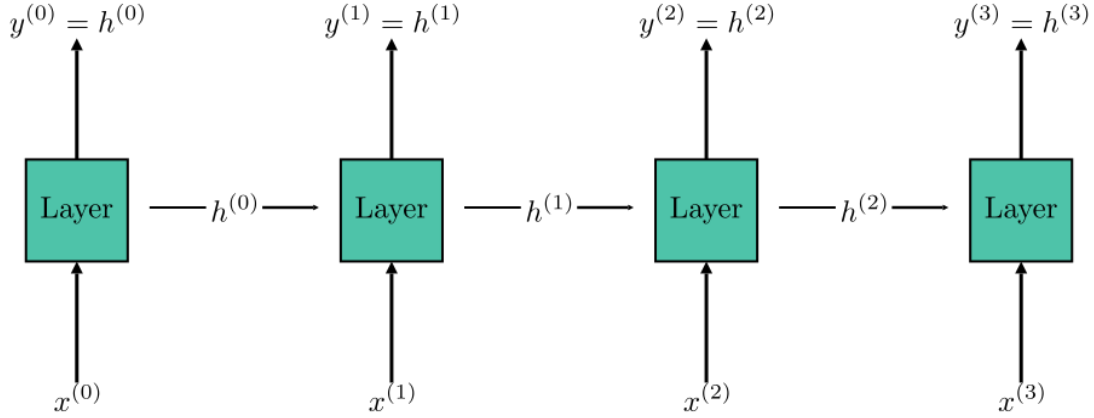


FIGURE 1.4 – Dépliement temporel d’un RNN sur une entrée de longueur 4.

De sa part, un décodeur récurrent peut conditionner sur l’encodage et passer ses états cachés à une couche supplémentaire qui les interprète comme plongements des éléments de la sortie y (FATHI, 2021). Un tel décodeur n’a pas besoin d’entrée séquentielle (voir Figure 1.5 et Algorithme 1.3).

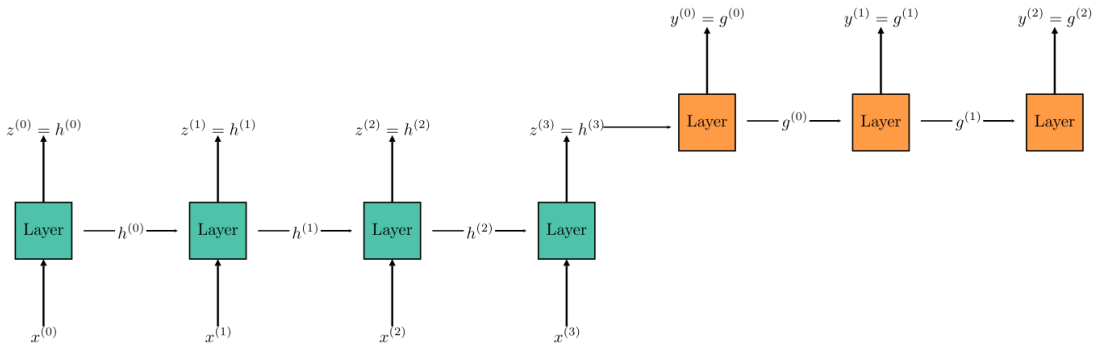


FIGURE 1.5 – Dépliement temporel d’un encodeur-décodeur récurrent sur une entrée de longueur 4 et une sortie de longueur 3.

Rétro-propagation dans le temps et mémoire à court terme

L’entraînement d’un RNN sur un exemple se fait en le dépliant, puis en entraînant le FFN qui résulte par rétro-propagation. L’algorithme résultant s’appelle la rétro-propagation dans le temps (BPTT, de l’anglais : back-propagation through time). Cela est problématique, car la taille de l’entrée n’est théoriquement pas bornée. Par conséquent, la profondeur effective d’un RNN ne l’est pas non plus (FATHI, 2021).

Or, dans l’entraînement d’un réseau de neurones trop profond, les modules des gradients peuvent atteindre des valeurs trop grandes ou trop petites. Il s’agit respectivement des problèmes de “l’explosion du gradient” et de “la disparition du gradient” (BASODI et al., 2020). Une conséquence de ce phénomène est que les RNN simples ont du mal à traiter

Algorithme 1.3 Encodeur–décodeur récurrent.

```
1 def rnn_s2s(xs:Sequence) -> Sequence:
2     encoder = RNN() # encodeur récurrent
3     decoder = RNN() # decodeur récurrent
4     ys = [] # sequence de sortie initialement vide
5     h = encoder(xs) # encode la sequence d'entree
6
7     while True:
8         # decoder sequentiellement
9         h = decoder(0, h)
10        ys.append(h)
11        if h == eos: # si on a genere le symbole de fin
12            # on arrete la generation
13            break
14    return ys
```

les entrées pour lesquelles le dépliement temporel est profond, (i.e les longues entrées). Cette incapacité à modéliser les corrélations à long terme est appelée “mémoire à court terme” (BENGIO et al., 1994; INFORMATIK et al., 2003).

1.4.2 Portes, gated recurrent unit et long short-term memory

Pour une grande partie des problèmes d’apprentissage S2S, les séquences peuvent être très longues. Le mémoire à court terme constitue donc un véritable obstacle pour l’utilisation des RNN simples en pratique. Une approche de le contourner qui a eu un énorme succès expérimental, est l’introduction d’un mécanisme de contrôle sur la boucle de rétroaction (voir Figure 1.6). Ce mécanisme est généralement implémenté avec des *portes*, des unités entraînables qui peuvent réguler le flux d’information dans la couche récurrente. On parle alors d’*RNN à portes*. Dans cette section, nous explorons les deux variants les plus utilisés d’RNN à portes : le gated recurrent unit (GRU) et le long short-term memory (LSTM).

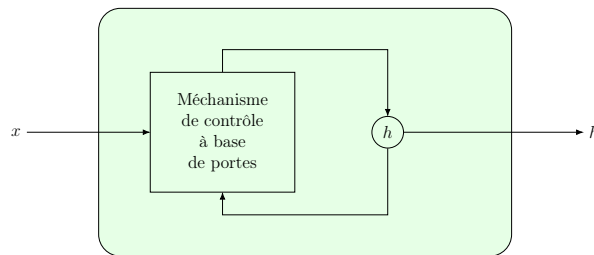


FIGURE 1.6 – Forme générale d’un RNN à portes.

Gated recurrent unit

Le GRU, introduit par (CHO et al., 2014), est une architecture récurrente à portes très simple (voir Figure 1.7). Elle utilise deux portes. La première est la porte de réinitialisation ((r dans la figure 1.7)). Elle détermine le point auquel l’information sur le passé peut se

propager (quand $r = 0$, pas de propagation et quand $r = 1$, propagation totale). Sa sortie s'appelle *l'état candidat* (\tilde{h} dans la figure). La deuxième est la porte de mise à jour (z dans la figure). Elle détermine les contributions respectives de l'état candidat et l'état passé (si $z = 0$, seul l'état candidat contribue et si $z = 1$, seule l'état courant contribue). Sa sortie est la sortie globale du GRU (CHO et al., 2014).

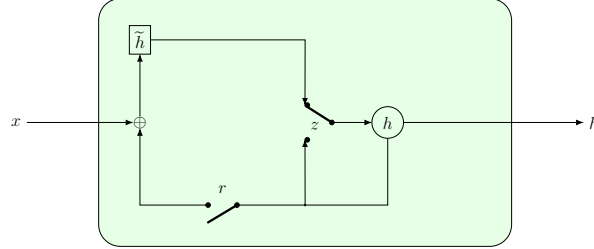


FIGURE 1.7 – Architecture interne d'un GRU (CHUNG et al., 2014, Fig. 1b)

Le fonctionnement des portes d'un GRU est simple. Leurs valeurs sont calculées à partir de l'entrée et de l'état courants par les équations 1.10 et 1.11, où σ est la fonction *sigmoïde*⁵ et \odot et le produit d'Hadamard⁶.

$$z^{(t)} = \sigma(W_z x^{(t)} + U_z h^{(t-1)} + b_z) \quad (1.10)$$

$$r^{(t)} = \sigma(W_r x^{(t)} + U_r h^{(t-1)} + b_r) \quad (1.11)$$

$$\tilde{h}^{(t)} = \varphi(W_h x^{(t)} + U_h (r^{(t)} \odot h^{(t-1)}) + b_h) \quad (1.12)$$

$$h^{(t)} = z^{(t)} \odot h^{(t-1)} + (1 - z^{(t)}) \odot \tilde{h}^{(t)} \quad (1.13)$$

L'état candidat est calculé à partir de l'entrée et l'état pondéré par la porte de réinitialisation par l'équation 1.12, où φ est la fonction d'activation. Finalement, l'état futur (la sortie) est la moyenne pondérée par z de l'état courant et l'état candidat (CHO et al., 2014). Notons que le GRU devient un RNN simple si les portes de réinitialisation et de mise à jour sont respectivement fixés à 1 et 0 (FATHI, 2021).

Long short-term memory

Il s'agit de l'une des premières architectures récurrentes à portes (CHUNG et al., 2014). Elle a été introduite par (HOCHREITER & SCHMIDHUBER, 1997). Un LSTM implémente trois portes : une porte d'entrées (i), une porte d'oublie (f) et une porte de sortie (o) (voir Figure 1.8).

Le fonctionnement des portes est similaire à celui du GRU. Les équations 1.14–1.19 le montrent en détails⁷ (HOCHREITER & SCHMIDHUBER, 1997).

5. $\sigma : \mathbb{R} \rightarrow \mathbb{R}, x \mapsto \frac{1}{1+e^{-x}}$

6. Pour $u, v \in \mathbb{R}^n$, $u \odot v \in \mathbb{R}^n$ et $(u \odot v)_i = u_i v_i$

7. $\tanh : \mathbb{R} \rightarrow \mathbb{R}, x \mapsto \frac{e^x - e^{-x}}{e^x + e^{-x}}$

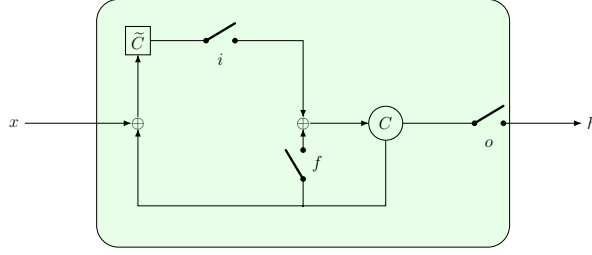


FIGURE 1.8 – Architecture interne d’un LSTM (CHUNG et al., 2014, Fig. 1a)

$$f^{(t)} = \sigma(W_f x^{(t)} + U_f h^{(t-1)} + b_f) \quad (1.14)$$

$$i^{(t)} = \sigma(W_i x^{(t)} + U_i h^{(t-1)} + b_i) \quad (1.15)$$

$$o^{(t)} = \sigma(W_o x^{(t)} + U_o h^{(t-1)} + b_o) \quad (1.16)$$

$$\tilde{c}^{(t)} = \tanh(W_c x^{(t)} + U_c h^{(t-1)} + b_c) \quad (1.17)$$

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t)} \quad (1.18)$$

$$h^{(t)} = o^{(t)} \odot \varphi(c^{(t)}) \quad (1.19)$$

Évaluation des réseaux de neurones récurrents

Nous avons établi que les RNN simples souffrent du mémoire à court terme (BENGIO et al., 1994 ; PASCANU et al., s. d.). En utilisant les portes pour contrôler le flux d’information, les GRU et LSTM résolvent le problème au prix d’architectures plus complexes et par conséquent plus lourdes à entraîner. Ils ont des performances similaires est largement meilleures que celle des RNN (CHUNG et al., 2014).

Cependant, toutes les architectures récurrentes présentent un problème fondamental : elles fonctionnent séquentiellement. Bien que cela les rend naturellement mieux adaptées à la modélisation de séquences, il les rend aussi quasi impossibles à paralléliser pour exploiter des architectures parallèles (GPU). Par conséquent, l’entraînement des RNN est extrêmement lent (STAHLBERG, 2020).

1.5 Réseau de neurones à convolutions

En dépit d’être une architecture naturelle pour le traitement des séquences, les RNN sont trop lents pour la majorité des cas d’utilisation pratiques. Cela est dû à leur nature séquentielle qui à son tour, est due à leur utilisation de boucles de rétroaction (voir Section 1.4). Une architecture sans telles boucles (i.e une architecture d’FFN) est donc préférable.

Les réseaux de neurones à convolutions (CNN, convolutional neural network) sont une famille d’FFN typiquement utilisés en traitement d’images. Ils ont été introduits par (FUKUSHIMA, 1980) et popularisés par (LECUN et al., 1989 ; LECUN et al., 1998).

Les CNN atteignent des performances comparables à celles des RNN sans mémoire explicite. Ils y parviennent en exploitant un outil mathématique appelé produit de *convolution* (CALIN, 2020).

Principes mathématiques de fonctionnement

Le produit de convolution de deux signaux f et g est le signal $f * g$ donné par

$$u \mapsto \int_{\mathbb{R}} f(u-t)g(t)dt \quad (1.20)$$

il s'agit d'une opération commune en probabilité, analyse fonctionnelle, traitement de signaux et traitement d'images (BARBE & LEDOUX, 2012; OPPENHEIM & SCHAFER, 2013). À partir d'elle, une autre opération appelée *l'inter-corrélation* est définie. L'inter-corrélation de f et g est notée $f \star g$. Elle est définie par :⁸

$$u \mapsto \int_{\mathbb{R}} f(t)g(t-u)dt = (f * g^-)(u) \quad (1.21)$$

Intuitivement, elle mesure la similarité entre les deux signaux en question. Les CNN utilisent l'inter-corrélation à la place des applications linéaires quelconques d'un MLP. Une couche de convolution unidimensionnelle calcul donc la fonction suivante

$$x \mapsto \varphi \left(b + \sum_{i=1}^n x \star w_i \right) \quad (1.22)$$

ou x est l'entrée et les vecteurs w_i sont les paramètres entraînaables de la couche, aussi appelés ses *noyaux* ou *masques* (voir Figure 1.9). La sortie de cette couche est une combinaison de tous les éléments de la séquence d'entrée. En composant suffisamment de telles couches, un CNN apprend une représentation de l'entrée beaucoup plus structurée qu'un MLP. On parle par fois de représentation *hiérarchique*.

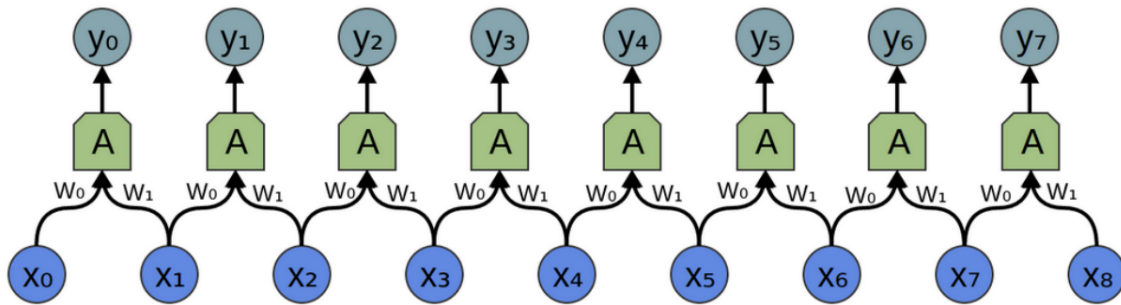


FIGURE 1.9 – Couche convolutive unidimensionnelle.

8. $g^- : t \mapsto g(-t)$.

Application à l'apprentissage S2S

Leur tendance naturelle à produire des représentations synthétiques des entrées, fait des CNN des encodeurs très puissants pour une grande variété de tâches, y compris des tâches de transduction de séquences. Plusieurs travaux ont combiné un encodeur convolutif avec un décodeur récurrent (voir Figure 1.10, KALCHBRENNER et BLUNSOM, 2013, Fig.3) (YANG et al., 2020).

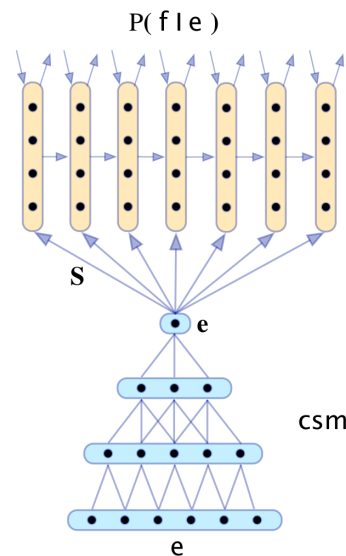


FIGURE 1.10 – Recurrent Continuous Translation Model

Cependant, des architectures totalement convolutives pour l'apprentissage S2S existent également. ByteNet (voir Figure 1.11 KALCHBRENNER et al., 2017) et ConvS2S (voir Figure 1.12 KAMEOKA et al., 2020) sont deux exemples de telles architectures.

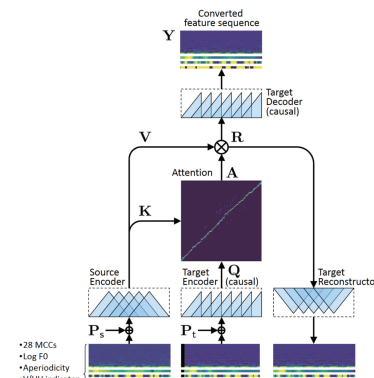
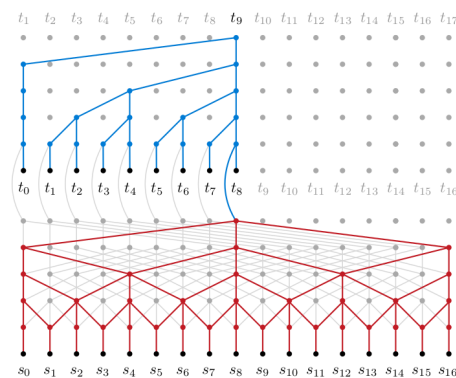


FIGURE 1.11 – Architecture de ByteNet. FIGURE 1.12 – Architecture de ConvS2S.

Avantages et inconvénients

Le produit de convolution (et par conséquent, inter-corrélation) est une opération très facile à paralléliser. Il est donc possible d'exploiter des architectures matérielles parallèles

(GPU) pour accélérer l’entraînement des CNN. À cet effet, les CNN sont beaucoup plus efficaces que les RNN pour les entrées longues. Ils peuvent être employés dans des cas pratiques de tailles raisonnables (LI et al., 2016).

Or, la nature locale du produit de convolution rend difficile aux CNN d’apprendre les corrélations globales⁹. En effet, pour représenter toutes les relations dans une séquence de longueur n , un CNN dont les masques sont de taille $k < n$ doit avoir $\log_k n$ couche (VASWANI et al., 2017). Le traitement de séquences très longues nécessite alors des CNN trop profonds, ce qui donne lieu aux problèmes de disparition et d’explosion des gradients.

1.6 Transformeurs

Les architectures discutées dans les sections 1.2 à 1.5 (notamment les encodeurs-décodeurs à base de CNN et RNN) forment la colonne vertébrale des modèles classiques d’apprentissage S2S (YANG et al., 2020). Cependant, leur mise en place est inhibée par des problèmes de performance (voir sections 1.4, 1.5 et 1.6.5).

Le transformeur (aussi appelé réseau de neurones auto-attentif) est une architecture performante pour le traitement de séquences (SHIM & SUNG, 2022). Introduite par (VASWANI et al., 2017), elle est basée sur le mécanisme d’attention (LAROCHELLE & HINTON, 2010). Une opération mathématique parallélisable à l’instar du produit de convolution, mais dont la complexité ne dépend pas de la distance dans les séquences.

1.6.1 Architecture générale

Le transformeur a une architecture encodeur-décodeur (voir Figure 1.13). L’encodeur et le décodeur ont des architectures très similaires, comprenant chacun une couche d’encodage positionnel, une couche d’auto-attention et un MLP. Le décodeur se distingue de l’encodeur en ce qu’il a une couche d’attention croisée entre l’auto-attention et l’MLP. Chacune des couches susmentionnées est suivie d’une couche de normalisation (BA et al., 2016) qui reçoit également une connexion directe à l’entrée de la couche¹⁰. L’encodeur et le décodeur peuvent être empilés pour former un transformeur profond. VASWANI et al., 2017 proposent 6 couches d’encodeur et 6 couches de décodeur.

Étant donné une séquence d’entrée $x = (x_1, \dots, x_n)$, l’encodeur calcule un vecteur de pensée z qu’il passe au décodeur. Le décodeur prédit le prochain élément y_i de la séquence de sortie à partir de z et les éléments précédents (y_1, \dots, y_{i-1}) . Pour produire y_0 , le décodeur commence avec une séquence de sortie vide.

9. Dépendances entre des couples d’éléments éloignés dans la séquence

10. On parle de connexion résiduelle ou de connexion de saut (skip connection) (HE et al., 2016)

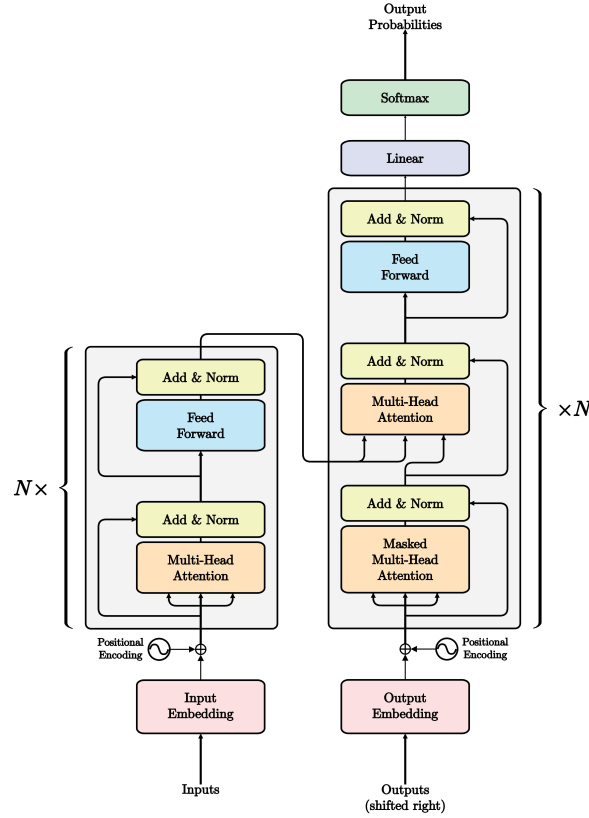


FIGURE 1.13 – Architecture de transformeur (VASWANI et al., 2017, Fig 1).

1.6.2 Encodage positionnel

La première couche de l'encodeur aussi bien que du décodeur est une couche d'encodage positionnel. Elle permet de calculer une représentation vectorielle de la séquence en question. Elle est composée d'une couche de plongement et d'une couche d'encodage de la position.

La couche de plongement mappe chaque élément de la séquence à un vecteur de dimension fixe d , ainsi transformant une séquence de longueur n en une matrice de $\mathbb{R}^{n \times d}$. Elle peut avoir une architecture quelconque en fonction de la nature des données. Il peut s'agir d'une couche de plongement lexical dans le cas d'un texte, d'une couche de convolution dans le cas de l'audio ou d'une couche récurrente dans le cas d'une série chronologique.

Le transformeur est un invariant aux permutations¹¹. La sortie de la couche de plongement est donc insuffisante pour représenter la séquence. Il lui faut ajouter une information sur l'ordre des éléments. C'est ce que fait la couche d'encodage de la position. Les auteurs (VASWANI et al., 2017) proposent l'encodage suivant pour la position i :

$$\text{PE}_k(i) = \begin{cases} \sin\left(\frac{i}{r^{2k/d}}\right) & \text{si } k \text{ est pair} \\ \cos\left(\frac{i}{r^{2k/d}}\right) & \text{si } k \text{ est impair} \end{cases} \quad 1 \leq k \leq d \quad (1.23)$$

11. Pour une séquence $x = (x_1, \dots, x_n)$ et une permutation $\pi \in S_n$, x et $(x_{\pi(i)})_{1 \leq i \leq n}$ ont deux encodages identiques.

où d est la dimension de plongement et r est un hyperparamètre fixé à 10^4 par les auteurs. La figure 1.14 montre la matrice d’encodage des 256 premières positions avec $r = 10^4$ et $d = 512$.

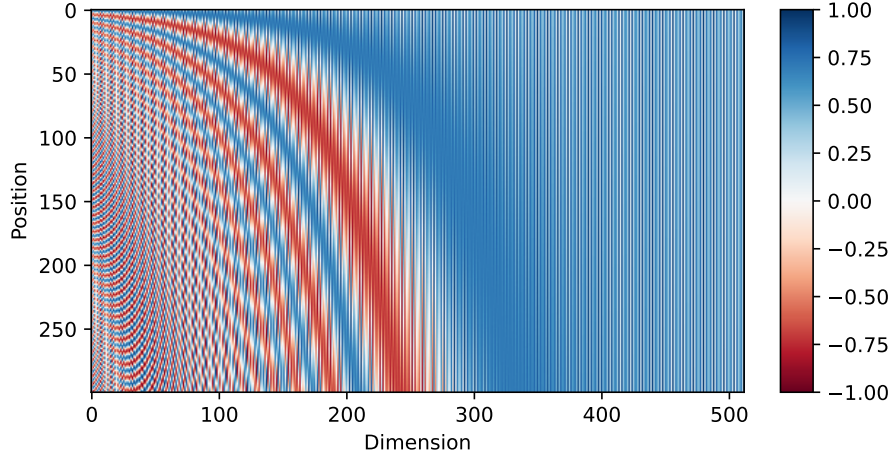


FIGURE 1.14 – Matrice d’encodage de la position pour $r = 10^4$ et $d = 512$.

La représentation finale de la séquence est la somme de la sortie de la couche de plongement et celle de la couche d’encodage de la position. Une couche de plongement lexical peut être substituée à l’encodage donné par l’équation (1.23) avec des résultats similaires (VASWANI et al., 2017).

1.6.3 Couches attention

Le mécanisme d’attention est la partie centrale du transformeur qui réalise sa fonctionnalité. Tous les autres modules sont une forme de prétraitement de son entrée ou de post-traitement de sa sortie. Chaque couche de l’encodeur implémente un module d’attention, tandis que chaque couche du décodeur en implémente deux.

Plusieurs types de mécanismes d’attention ont été proposés dans la littérature, notamment l’attention additive (BAHDANAU et al., 2016) et l’attention multiplicative (LUONG et al., 2015). Le transformeur utilise une variante de l’attention multiplicative appelée “*scaled dot-product attention*” (VASWANI et al., 2017). Elle opère sur trois matrices Q, K, V ¹² de dimensions respectives $d_Q \times l, d_K \times m, d_V \times n$, qui représentent chacune l’encodage d’une séquence¹³. Le résultat est donné par l’équation (1.24).

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_K}} \right) V \quad (1.24)$$

Intuitivement, la scaled dot-product attention peut être vue comme une forme de recherche floue dans une base de données. Le terme QK^T est un produit scalaire entre les lignes de

12. De la terminologie anglophone : “*Query*”, “*Key*”, “*Value*”, empruntée aux systèmes de recherche de l’information.

13. d_Q, d_K, d_V sont les dimensions de plongement et l, m, n sont les longueurs des séquences. Notez que l’équation (1.24) impose les contraintes $d_Q = d_K$ et $m = n$, ce qui est vérifié pour les trois modules d’attention du transformeur.

la requête Q et la clé K . Il représente la similarité entre les deux. La fonction softmax¹⁴ le normalise pour obtenir des poids positifs de somme 1. Enfin, le résultat est une somme des valeurs V pondérées par ces poids. C'est dans ce sens-là qu'il s'agit d'une recherche floue : les valeurs correspondantes aux clés qui répondent le mieux à la requête sont sélectionnées, mais au lieu de renvoyer discrètement la meilleure valeur, toutes les valeurs contribuent dans la mesure de leur pertinence (« CS480/680 Lecture 19 : Attention and Transformer Networks », 2019). La division par $\sqrt{d_K}$ est une forme de régularisation. Elle permet d'éviter que le module de l'entrée du softmax devienne trop grand, ce qui rend son gradient trop petit ainsi ralentissant l'apprentissage¹⁵ (VASWANI et al., 2017).

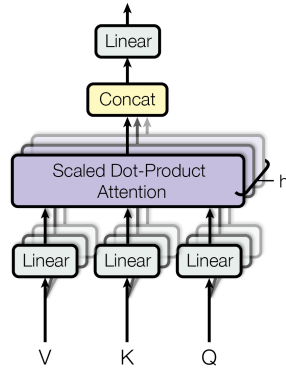


FIGURE 1.15 – Schéma d'une couche attention multi-tête (VASWANI et al., 2017, Fig 2).

Les auteurs de (VASWANI et al., 2017) ont observé une amélioration de la performance en utilisant plusieurs modules (ou têtes) de scaled dot-product attention, projetant les entrées différemment avant de les passer à chaque tête. Les sorties des têtes sont concaténées et passées à une couche linéaire qui produit la sortie finale. Les projections sont réalisées par des couches linéaires entraînaables et le nombre h de têtes est un hyperparamètre fixé à 8 par les auteurs (voir la Figure 1.15).

La couche attention de l'encodeur et la première couche attention du décodeur sont des couches d'*auto-attention* i.e leurs requêtes, clés et valeurs sont les mêmes. L'*auto-attention* du décodeur se distingue de celle de l'encodeur par la présence du *masque*. En produisant un élément de la séquence de sortie, le décodeur ne doit faire attention qu'aux éléments qui le précèdent. Toutes les autres valeurs doivent avoir un poids nul. Pour ce faire, la matrice M

$$M_{ij} = \begin{cases} 0 & \text{si } j > i \\ -\infty & \text{sinon} \end{cases} = \begin{bmatrix} -\infty & -\infty & \cdots & -\infty \\ 0 & -\infty & \cdots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -\infty \end{bmatrix} \quad (1.25)$$

est ajoutée à l'entrée de la fonction softmax.

La deuxième couche attention du décodeur est une couche d'*attention croisée*. Les requêtes viennent de la sortie et les clés et valeurs viennent de l'entrée. Cette couche calcul

14. $\text{softmax} : \mathbb{R}^p \rightarrow \mathbb{R}^p, x = (x_1, x_2, \dots, x_p) \mapsto \frac{e^{x_i}}{\sum_{i=1}^p e^{x_i}}$

15. Le choix de la valeur de $\sqrt{d_K}$ n'est pas arbitraire. Sous l'hypothèse que les valeurs de Q et K sont centrées et réduites, les valeurs de QK^T sont centrées de variance d_K . La division par $\sqrt{d_K}$ les ramène à une variance de 1 (VASWANI et al., 2017).

les dépendances entre l'entrée et la sortie, tandis que les deux autres couches calculent les dépendances à l'intérieur de l'entrée et de la sortie. C'est cette couche qui permet au décodeur de conditionner sa sortie sur l'entrée.

1.6.4 Autres éléments de l'architecture

L'encodage positionnel et l'auto-attention sont les deux innovations clés du transformeur. Les autres éléments de l'architecture existent comme portes d'entrée et de sortie pour elles. Ces éléments sont les suivants :

- Les couches feed forward : Il s'agit d' MLP qui exploitent la représentation calculée par le mécanisme d'attention.
- Les couches de normalisation : Introduites par (BA et al., 2016), elles sont appliquées après chaque module. Elles assurent que leur sortie est centrée et réduite. Cela permet de stabiliser l'apprentissage.
- Les connexions résiduelles (HE et al., 2016) : Appliquées à chaque couche de normalisation, elles ont la forme $\text{LayerNorm}(x + \text{Module}(x))$. Elles permettent aux gradients de se propager plus facilement.
- La couche de sortie : Dans le cas où les éléments de la sortie sont des variables continues (par exemple pour amplitudes d'un signal audio), une couche linéaire est appliquée à la sortie du dernier module. Dans le cas opposé (par exemple pour les mots d'une phrase), la fonction softmax est aussi appliquée pour donner la loi de probabilité de la sortie.

1.6.5 Analyse comparative de la performance

Dans le début de cette section, nous affirmons la supériorité du transformeur aux autres architectures S2S. Ce constat repose sur la comparaison du Tableau 1.1 (VASWANI et al., 2017). On y voit que le transformeur est le seul à avoir une longueur de chemin constante, c'est-à-dire qu'elle possède un majorant indépendant de la taille de l'entrée.

On note également que le transformeur exige un nombre d'opérations séquentielles constant, ce qui est le cas pour les autres architectures à l'exception des RNN. En effet, tous les réseaux feed-forward ont cette propriété.

Pour la complexité par couches, l'auto-attention est la plus efficace pour les séquences courtes (pour lesquelles $n \ll d$). Ces séquences sont les plus fréquentes en pratique pour les dimensions de représentation usuelles. Dans le cas de très longues séquences, elle est moins efficace que les autres architectures. Cependant, le transformeur toujours de meilleurs résultats pour ces séquences (SHIM & SUNG, 2022).

Type du module	Complexité	Nombre d'Opérations Séquentielles	Longueur du Chemin Emprunté par le Gradient
Auto-Attention	$\mathcal{O}(n^2 \cdot d)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Recurrent	$\mathcal{O}(n \cdot d^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Convolutif	$\mathcal{O}(k \cdot n \cdot d^2)$	$\mathcal{O}(1)$	$\mathcal{O}(\log_k n)$
MLP	$\mathcal{O}(n \cdot d^2)$	$\mathcal{O}(1)$	$+\infty$

TABLE 1.1 – Analyse comparative de la performance des différents types d’architectures S2S. n est la longueur de la séquence, d est la dimension de la représentation vectorielle des éléments et k est la taille du noyau des convolutions (VASWANI et al., 2017, Tab. 1).

1.7 Conclusion

Dans ce chapitre, nous avons introduit l’apprentissage S2S, une tâche générale en ML dont la MT et la ASR sont des cas particuliers (SZABO et al., s. d.). Nous avons défini le problème traité en modélisation S2S, expliqué les différentes architectures neuronales utilisées pour le résoudre et donné une comparaison de leurs performances théoriques. Dans le chapitre suivant, nous étudions les deux exemples de MT et ASR dans la mesure où ils peuvent être appliqués à notre problématique.

Conclusion générale

L’aphasie de Broca est un trouble de communication qui touche une partie grandissante de la population mondiale. Souvent, le résultat d’un accident vasculaire cérébral (AVC), l’aphasie de Broca affecte le décodage des mots et la production de la parole. Elle diminue ainsi mesurablement la qualité de vie des personnes qui en sont atteintes (CHAPEY, 2008 ; FEIGIN et al., 2022 ; ROSS & WERTZ, 2010).

Le traitement le plus courant de l’aphasie de Broca est la rééducation de la parole par un orthophoniste. En dépit d’être un traitement efficace, il est gourmand en temps, en argent et en ressources humaines, ce qui en fait un traitement peu accessible pour la majorité des personnes atteintes. Ce manque d’accessibilité, combiné à la gravité de certaines conséquences de l’aphasie de Broca, rend urgent le développement de solutions alternatives (da FONTOURA et al., 2012 ; FLOWERS et al., 2016).

Les méthodes informatiques, notamment les techniques de ML et de NLP, semblent avoir le potentiel de réduire les coûts matériels et humains associés à la rééducation de la parole. Elles peuvent ainsi faciliter l’accès au traitement de l’aphasie de Broca (MISRA et al., 2022 ; QIN et al., 2022 ; SMAÏLI et al., 2022).

Dans ce mémoire, nous avons exploré la possibilité d’utiliser la MT et l’ASR, deux techniques de NLP basées sur l’apprentissage S2S pour aider les personnes atteintes de l’aphasie de Broca.

Nous avons d’abord introduit l’aphasie pour familiariser le lecteur avec ses causes, sa portée, ses effets, les traitements disponibles et les défis auxquels ils sont confrontés. Après cela, nous avons présenté la modélisation S2S, le cadre général de la MT et de l’ASR. Nous avons posé le problème et présenté les solutions que nous avons évaluées et comparées. Le résultat de cette comparaison est une supériorité nette du transformeur sur les autres modèles. Nous avons alors exploré dans le troisième chapitre les différentes publications qui ont étudié l’utilisation du transformeur pour la résolution de problèmes de MT et d’ASR.

Bibliographie

- BA, J. L., KIROS, J. R., & HINTON, G. E. (2016). Layer Normalization [arXiv :1607.06450 [cs, stat]], (arXiv :1607.06450). <http://arxiv.org/abs/1607.06450>
- BAHDANAU, D., CHO, K., & BENGIO, Y. (2016). Neural Machine Translation by Jointly Learning to Align and Translate [arXiv :1409.0473 [cs, stat]], (arXiv :1409.0473). <https://doi.org/10.48550/arXiv.1409.0473>
- BARBE, P., & LEDOUX, M. (2012). *Probabilité (L3M1)*. EDP Sciences.
- BASODI, S., JI, C., ZHANG, H., & PAN, Y. (2020). Gradient amplification : An efficient way to train deep neural networks. *Big Data Mining and Analytics*, 3(3), 196-207. <https://doi.org/10.26599/BDMA.2020.9020004>
- BENGIO, Y., SIMARD, P., & FRASCONI, P. (1994). Learning long-term dependencies with gradient descent is difficult. 5(2), 157-166. <https://doi.org/10.1109/72.279181>
- CALIN, O. (2020). *Deep Learning Architectures*. Springer. <https://link.springer.com/book/10.1007/978-3-030-36721-3>
- CHAPEY, R. (2008). *Language Intervention Strategies in Aphasia and Related Neurogenic Communication Disorders*. Wolters Kluwer Health/Lippincott Williams & Wilkins.
- CHO, K., van MERRIENBOER, B., BAHDANAU, D., & BENGIO, Y. (2014). On the Properties of Neural Machine Translation : Encoder-Decoder Approaches [arXiv :1409.1259 [cs, stat]], (arXiv :1409.1259). <http://arxiv.org/abs/1409.1259>
- CHUNG, J., GULCEHRE, C., CHO, K., & BENGIO, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling [arXiv :1412.3555 [cs]], (arXiv :1412.3555). <http://arxiv.org/abs/1412.3555>
- CS480/680 Lecture 19 : Attention and Transformer Networks. (2019). https://www.youtube.com/watch?v=OyFJWRnt_AY
- da FONTOURA, D. R., RODRIGUES, J. d. C., CARNEIRO, L. B. d. S., MONÇÃO, A. M., & de SALLES, J. F. (2012). Rehabilitation of language in expressive aphasia : a literature review. *Dementia & Neuropsychologia*, 6(4), 223-235. <https://doi.org/10.1590/S1980-57642012DN06040006>
- FATHI, S. (2021). *Recurrent Neural Networks*. <https://link.springer.com/book/10.1007/978-3-030-89929-5>
- FEIGIN, V. L., BRAININ, M., NORRVING, B., MARTINS, S., SACCO, R. L., HACKE, W., FISHER, M., PANDIAN, J., & LINDSAY, P. (2022). World Stroke Organization (WSO) : Global Stroke Fact Sheet 2022. *International Journal of Stroke : Official Journal of the International Stroke Society*, 17(1), 18-29. <https://doi.org/10.1177/17474930211065917>
- FLOWERS, H., SKORETZ, S., SILVER, F., ROCHON, E., FANG, J., FLAMAND-ROZE, C., & MARTINO, R. (2016). Poststroke Aphasia Frequency, Recovery, and Outcomes :

- A Systematic Review and Meta-Analysis. *Archives of Physical Medicine and Rehabilitation*, 97, 2188-2201. <https://doi.org/10.1016/j.apmr.2016.03.006>
- FUKUSHIMA, K. (1980). Neocognitron : A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4), 193-202. <https://doi.org/10.1007/BF00344251>
- HALLOWELL, B. (2017). *Aphasia and Other Acquired Neurogenic Language Disorders : A Guide for Clinical Excellence*. Plural Publishing.
- HE, K., ZHANG, X., REN, S., & SUN, J. (2016). Deep Residual Learning for Image Recognition, 770-778. https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html
- HOCHREITER, S., & SCHMIDHUBER, J. (1997). Long short-term memory. 9(8), 1735-1780.
- INFORMATIK, F., BENGIO, Y., FRASCONI, P., & SCHMIDHUBER, J. (2003). Gradient Flow in Recurrent Nets : the Difficulty of Learning Long-Term Dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*.
- KALCHBRENNER, N., & BLUNSOM, P. (2013). Recurrent Continuous Translation Models. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 1700-1709. <https://aclanthology.org/D13-1176>
- KALCHBRENNER, N., ESPEHOLT, L., SIMONYAN, K., OORD, A. v. d., GRAVES, A., & KAVUKCUOGLU, K. (2017). Neural Machine Translation in Linear Time [arXiv :1610.10099 [cs]], (arXiv :1610.10099). <http://arxiv.org/abs/1610.10099>
- KAMEOKA, H., TANAKA, K., KWAŚNY, D., KANEKO, T., & NOBUKATSU, H. (2020). ConvS2S-VC : Fully Convolutional Sequence-to-Sequence Voice Conversion. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28, 1849-1863. <https://doi.org/10.1109/TASLP.2020.3001456>
- KEARNS, M. J., & VAZIRANI, U. (1994). *An Introduction to Computational Learning Theory* [Google-Books-ID : vCA01wY6iywC]. MIT Press.
- LAROCHELLE, H., & HINTON, G. E. (2010). Learning to combine foveal glimpses with a third-order Boltzmann machine. *Advances in Neural Information Processing Systems*, 23. <https://proceedings.neurips.cc/paper/2010/hash/677e09724f0e2df9b6c000b75b5da10d-Abstract.html>
- LECUN, Y., BOSER, B., DENKER, J. S., HENDERSON, D., HOWARD, R. E., HUBBARD, W., & JACKEL, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4), 541-551. <https://doi.org/10.1162/neco.1989.1.4.541>
- LECUN, Y., BOTTOU, L., BENGIO, Y., & HAFFNER, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324. <https://doi.org/10.1109/5.726791>
- LECUN, Y., BENGIO, Y., & HINTON, G. (2015). Deep learning. *Nature*, 521(7553), 436-444. <https://doi.org/10.1038/nature14539>
- LI, X., ZHANG, G., HUANG, H. H., WANG, Z., & ZHENG, W. (2016). Performance Analysis of GPU-Based Convolutional Neural Networks. *2016 45th International Conference on Parallel Processing (ICPP)*, 67-76. <https://doi.org/10.1109/ICPP.2016.15>
- LUONG, M.-T., PHAM, H., & MANNING, C. D. (2015). Effective Approaches to Attention-based Neural Machine Translation [arXiv :1508.04025 [cs]], (arXiv :1508.04025). <http://arxiv.org/abs/1508.04025>
- MARTINS, A. (2018). Lecture 9 : Machine Translation and Sequence-to-Sequence Models.
- MISRA, R., MISHRA, S. S., & GANDHI, T. K. (2022). Assistive Completion of Agrammatic Aphasic Sentences : A Transfer Learning Approach using Neurolinguistics-based

- Synthetic Dataset [arXiv :2211.05557 [cs, q-bio]], (arXiv :2211.05557). <http://arxiv.org/abs/2211.05557>
- MUKHERJEE, A. (2021). A Study of the Mathematics of Deep Learning [arXiv :2104.14033 [cs, math, stat]], (arXiv :2104.14033). <http://arxiv.org/abs/2104.14033>
- OPPENHEIM, A. V., & SCHAFER, R. W. (2013). *Discrete-time Signal Processing*. Pearson.
- PASCANU, R., MIKOLOV, T., & BENGIO, Y. (s. d.). On the difficulty of training recurrent neural networks.
- QIN, Y., LEE, T., KONG, A. P. H., & LIN, F. (2022). Aphasia Detection for Cantonese-Speaking and Mandarin-Speaking Patients Using Pre-Trained Language Models. *2022 13th International Symposium on Chinese Spoken Language Processing (ISCSLP)*, 359-363. <https://doi.org/10.1109/ISCSLP57327.2022.10037929>
- ROSS, K., & WERTZ, R. (2010). Quality of life with and without aphasia. *Aphasiology*. <https://doi.org/10.1080/02687030244000716>
- SEBASTIAN, R., & MIRJALILI, V. (2017). *Python Machine Learning : Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow* (2^e éd., T. 1). Packt Publishing.
- SHIM, K., & SUNG, W. (2022). A Comparison of Transformer, Convolutional, and Recurrent Neural Networks on Phoneme Recognition [arXiv :2210.00367 [cs, eess]], (arXiv :2210.00367). <http://arxiv.org/abs/2210.00367>
- SMAÏLI, K., LANGLOIS, D., & PRIBIL, P. (2022). Language rehabilitation of people with BROCA aphasia using deep neural machine translation. *Fifth International Conference Computational Linguistics in Bulgaria*, 162.
- STAHLBERG, F. (2020). Neural Machine Translation : A Review. *Journal of Artificial Intelligence Research*, 69, 343-418. <https://doi.org/10.1613/jair.1.12007>
- SZABO, V., PLESIAK, M., YANG, Y., & HEUMANN, C. (s. d.). *Modern Approaches in Natural Language Processing*. https://slds-lmu.github.io/seminar_nlp_ss20/index.html
- VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, & POLOSUKHIN, I. (2017). Attention is All you Need. *Advances in Neural Information Processing Systems*, 30. <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>
- YANG, S., WANG, Y., & CHU, X. (2020). A survey of deep learning techniques for neural machine translation. *arXiv preprint arXiv :2002.07526*.