République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديموقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي والبحث العلمي

المدرسة الوطنية للإعلام الآلي
(المعهد الوطني للتكوين في الإعلام الآلي سابقا)
École nationale Supérieure d'Informatique
ex. INI (Institut National de formation en Informatique)

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

18 janvier 2023

Table des matières

Table des figures

Sigles et abréviations

ML	apprentissage automatique
MLP	perceptorn multicouches
MT	traduction automatique
NLP	traîtement automatique du langage
RNN	réseau de neurones récurrent
S2S	séquence-à-séquence

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 learining) dont l'entrée et la sortie sont des séquences (Martins, 2018). Plusieurs tâches d'ML, notamment en traîtement 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, la traduction automatique (MT) et la reconnaissance automatique de la parole (Fathi, 2021).

Dans ce chapitre, nous commençons par formuler le problème de modélisation de séquences. En suite, nous présentons les architectures neuronales les plus utilisées pour cette tâche. En fin, nous terminons 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 $f : X^* \to Y^*$, 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 un $x = (x_1, x_2, \cdots, x_n) \in X^n$ et renvoie un $y = (y_1, y_2, \cdots, y_m) \in Y^m$. Dans le cas général, $n \neq m$ et aucune hypothèse d'alignement n'est supposée. Il est souvent de prendre $X = \mathbb{R}^{d_e}$ 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 utilisées 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 Perceptorns multicouches

Les réseaux de neurones profonds sont parmi les modèles les plus expressifs en ML. Leur succès pratique est incomparable aux modèles qui les ont précédés, que se soit en termes de qualité des résultats ou de variétés de domaines d'application (Raschka, 2015). 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 introduisant les perceptorns multicouches (MLP, de l'anglais : multi-layer perceptorn), 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 (voire Définition 1).

Definition 1 (MLP (Mukherjee, 2021))**.**
Soient $k, w_0, w_1, \cdots, w_{k+1} \in \mathbb{N}$, un réseau de neurones feed-forward de profondeur $k+1$, à w_0 entrées et w_{k+1} sorties, est défini par une fonction :

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

Où les A_i sont des fonctions affines $\mathbb{R}^{w_{i-1}} \to \mathbb{R}^{w_i}$ et les φ_i sont des fonctions quelconques, typiquement non affines $\mathbb{R}^{w_i} \to \mathbb{R}^{w_i}$, dites *d'activations*. La fonction $\varphi_i \circ A_i$ est appelée la i^{eme} couche du réseau.

Un tel réseau de neurones est souvent représenté par un graphe orienté acyclique $(k+1)-$partie appelé son "architecture sous-jacente" (Kearns & Vazirani, 1994). La Figure 1.1 illustre l'architecture d'un MLP de profondeur 4 avec $(w_0, w_1, w_2, w_3, w_4) = (4, 5, 7, 5, 4)$.

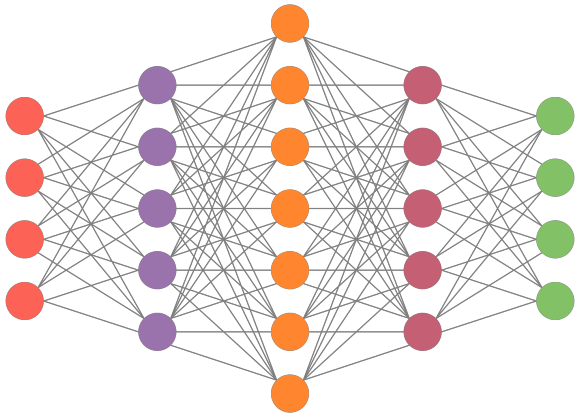

Figure 1.1 – Architecture sous-jacente d'un MLP de profondeur 3.

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.

Pour ce faire, nous exploitons le fait que les A_i soient 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 régler [1] 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*.

$$L_1(f, \hat{f}) = \int \left\| f - \hat{f} \right\| \tag{1.2}$$

$$L_2(f, \hat{f}) = \int \left\| f - \hat{f} \right\|^2 \tag{1.3}$$

Ayant fixé une fonction de perte L, l'entraînement revient à un problème d'optimisation. 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. À fin de les utiliser dans le contexte de la modélisation S2S, il faut donc utiliser une représentation vectorielle des entrées. Une telle représentation s'appelle un *plongement* (embedding en anglais). Le plongement peut-être apprit ou prédéfini.

Dans le cas des MLP, la séquence d'entrée est d'abord décomposée en sous-séquences. En suite, 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. (Voire l'Algorithme 1.1).

ALGORITHME 1.1 : Passe d'un MLP

Input : input_sequence `// Séquence d'entrée`

```
1 begin
2     y ← ∅
3     foreach x ∈ sub_sequences(input_sequence) do
4         x̃ ← embedding(x)
5         y ← y∪FFN(x̃)
6     end
7 end
```

Output : y `// Séquence de sortie`

1. En ML, le terme "entraîner" est plutôt utilisé.

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, et par conséquent, il accélère considérablement leur entraînement.

Cependant, ce dernier point pose un grand problème. Comme ils traitent indépendamment les bloques de la séquence, les MLP ne peuvent pas modéliser les dépendances inter-bloc. 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 (est donc aussi la dimension d'entrée). Or, cela suppose un 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 due au 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 1.4 l'illustre pour une entrée $x = (x_1, x_2, \cdots, x_n)$ et une sortie $y = (y_1, y_2, \cdots, y_m)$.

$$y_j = f(x_i, x_{i+1}, \cdots, x_{i+\ell}) \qquad 1 \leq j \leq m \tag{1.4}$$

En plus de l'hypothèse implicite de l'existence d'une telle correspondance, cela suppose que les éléments d'une séquence sont complétement indépendant l'un de l'autre. 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 1.5 montre cette approche sur le même exemple.

$$y_j = f(x) = f(x_1, x_2, \cdots, x_n) \qquad 1 \leq j \leq m \tag{1.5}$$

1.4 Réseaux de neurones récurrents

Comme nous l'avons établi en Section 1.2, les MLP sont très mal adaptés à l'apprentissage S2S. D'autres architectures doivent donc être utilisées. Le réseau de neurones récurrent (RNN, de l'anglais : recurrent neural network) est une telle architecture. Étant conçues spécifiquement pour ce type de tâche, les RNN peuvent capturer les dépendances séquentielles.

Bibliographie

Calin, O. (2020). *Deep Learning Architectures*. Springer. https://link.springer.com/book/10.1007/978-3-030-36721-3

Fathi, S. (2021). *Recurrent Neural Networks*. https://link.springer.com/book/10.1007/978-3-030-89929-5

Kearns, M. J., & Vazirani, U. (1994). *An Introduction to Computational Learning Theory* [Google-Books-ID : vCA01wY6iywC]. MIT Press.

Martins, A. (2018). Lecture 9 : Machine Translation and Sequence-to-Sequence Models.

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

Raschka, S. (2015). *Python Machine Learning* [Google-Books-ID : GOVOCwAAQBAJ]. Packt Publishing Ltd.