Table des matières

1 Introduction

1.1 Contexte et enjeux du stage

D'une part, depuis quelques années, l'apprentissage profond (*Deep Learning* en anglais) et les réseaux de neurones artificiels ou réseaux de neurones (*Neural Networks* en anglais) ont connu une explosion de popularité. Ce qui se cache derrière cet engouement est la combinaison de théories relativement anciennes et d'avancées technologiques permettant la mis en œuvre desdites théories.

Un des domaines exploitant les performances de ces outils est le traitement automatique des langues (TAL, *Natural Language Processing* en anglais). L'application au TAL des réseaux de neurones qui nous intéressera particulièrement dans ce rapport est la création de modèle de la langue.

D'autre part, nous somme à l'ère du « *Big Data* », et les quantités de donnéess produites de nos jours est bien au delà de ce que nous pouvons gérer sans outils spécialisés. Afin de produire des outils adaptés aux échelles actuelles, nous nous intéresserons dans ce rapport à des grands volumes de donnéess bien au delà des volumes habituellement utilisés en apprentissage profond.

Ainsi, l'axe principal de ce rapport est l'application des méthodes de l'apprentissage profond du point de vue du TAL sur de grands volumes de donnéess. Cela implique des problématiques relativement classiques en développement de réseau de neurones artificiels : le choix de l'architecture du réseau, de l'algorithme d'entraînement, mais aussi des questions plus pragmatique d'optimisation lié au volume de donnéess.

1.2 Objectifs du stage

Deux objectifs successifs se distinguent dans le stage.

L'objectif initial du stage était d'explorer une idée d'architecture de réseau de neurones artificiels innovante, imaginée par notre maître de stage Mr. Cerisara. Il s'agit du projet GMSNN.

Cependant, à l'issue du deuxième mois du stage, notre mission à évolué.

Le nouvel objectif était la réalisation d'un réseau de neurones artificiels et des outils nécessaires à son utilisation, en mettant à profit les connaissances acquises durant la première partie du stage. Cette réalisation servira de base technique pour une partie du projet PAPUD.

Les tenants et aboutissants des deux objectifs, ainsi que l'explication des noms des projets, seront expliqués en détails dans le chapitre 3 et le chapitre 9.

1.3 Terminologie et concepts fondamentaux

Cette section est dédié à la présentation et l'explication des théories, termes et concepts nécessaires à la compréhension du présent rapport.

Ces termes sont répertoriés dans le glossaire, et mais aucun rappel de leur présence dans celui-ci ne sera donné au long du texte.

L'objectif n'est pas de fournir des explications approfondies, mais de fournir les connaissances minimales à la compréhension du contenu et des enjeux du rapport.

1.3.1 Apprentissage automatique, modèle, entraînement et données

Apprentissage automatique

L'apprentissage automatique (*Machine Learning* en anglais) est un ensemble de « méthodes [statistiques] permettant à une machine (au sens large) d'évoluer par un processus systématique, et ainsi de remplir des tâches difficiles ou problématiques par des moyens algorithmiques plus classiques ». D'après Wikipédia [1].

Modèle

Un modèle en apprentissage automatique (*Machine Learning* en anglais) est la représentation du monde construite afin de répondre au problème à résoudre.

Pour un modèle, on parle d'entrées pour désigner les donnéess fournies, et de sorties pour désigner les donnéess produites.

Entraînement du modèle

L'entraînement du modèle, aussi appelé apprentissage, est le processus par lequel on adapte le modèle de façon à mieux résoudre le problème.

Données d'entraînement

Pour entraîner un modèle, il faut lui fournir des donnéess. Voici quelques termes courants se référant aux donnéess d'entraînement :

- le corpus : l'ensemble des donnéess d'entraînement ;
- un exemple : un fragment du corpus utilisé pour entraîner un modèle ;
- une époque : un cycle complet d'entraînement sur le corpus.

Généralement, on effectue un prétraitement des données (*preprocessing* en anglais) pour les préparer. Cela peut être retirer les données erronées, en adapter le format, les anonymiser, ou associer le résultat attendu aux données correspondantes.

1.3.2 Apprentissage profond et réseaux de neurones

Apprentissage profond

L'apprentissage profond représente un ensemble de techniques de apprentissage automatique, basés sur les techniques appelées réseaux de neurones.

Faisant partie des méthodes du apprentissage automatique, l'apprentissage profond regroupe à la fois les méthodes de création, d'entraînement, d'optimisation et d'utilisation des modèles basés sur des réseaux de neurones.

Réseau de neurones artificiels

Dans le cadre de l'apprentissage profond, un modèle correspond généralement au réseau de neurones artificiels.

Un réseau de neurones artificiels ou réseau de neurones (*Neural Network* en anglais) est un modèle mathématique composé d'éléments interconnectés nommés neurones formels, dont le comportement est vaguement semblable aux neurones biologiques.

Un réseau de neurones artificiels prend en entrée un tenseur (*tensor* en anglais) (un type de matrice spécifique utilisé en apprentissage profond), et produit un tenseur en sortie. Toutes les valeurs contenues dans ces tenseurs sont des nombres.

Architecture et modules

Pour des raisons de concision, nous considéreront les réseaux de neurones comme des modules. C'est-à-dire que nous les considérerons comme des boites noires, sans nous intéresser à leur conception interne.

Nous parlerons d'architecture pour désigner à la fois la façon dont sont conçus les réseaux de neurones et la façon dont sont assemblés les modules pour former un modèle.

Pour décrire les architectures, nous utiliserons des diagrammes considérant les modules comme des blocs.

Paramètre Les paramètres pour un module ou un modèles sont des valeurs qui varient au cours de l'entraînement. Généralement, plus un modèles possède de paramètres, plus son entraînement consomme de ressources mais plus la qualité de l'apprentissage est élevé.

Principales architectures de réseau de neurones artificiels

Réseau de neurones artificiels intégralement connecté ou module linéaire Il s'agit d'un des types de réseaux de neurones les plus simples. Il est emblématique des réseaux de neurones.

RNN Le RNN est une architecture de réseaux de neurones particulièrement adaptée au traitement de séquences. Il traite des éléments les uns après les autres, et stocke dans une « mémoire » des informations au fur-et-à-mesure. Il utilise les informations stockées pour améliorer la façon dont il traite les prochains éléments.

LSTM Le LSTM est un RNN particulier équipé d'une mémoire supplémentaire.

On peut noter qu'un réseaux de neurones intégralement connecté consomme moins de ressources qu'un RNN qui en consomme moins qu'un LSTM.

1.3.3 TAL et modèle de la langue

TAL

Le traitement automatique des langues (TAL, *Natural Language Processing* en anglais) est une discipline qui s'intéresse au traitement des informations langagières par des moyens formels ou informatiques.

Modèle de la langue

Un modèle de la langue est une « distribution de probabilité sur une séquence de mots [ou de caractères] », utilisé pour estimer la probabilité d'apparition du prochain mot ou caractère. D'après Wikipédia [2].

Autrement dit, c'est une représentation servant à prédire le prochain mot à partir des mots précédents.

Contexte et dépendances

Dans le cadre d'un modèle de la langue, le contexte est l'ensemble des informations disponible hors du mot ou caractère à prédire.

On parle aussi de dépendances entre les mots ou caractères et l'élément du contexte correspondant.

Les informations contenues dans le contexte peuvent autant être les mots alentours que la structure de la phrase ou des informations comme le genre du héro d'un livre.

On considère que plus on à d'informations contextuelles plus le modèle de la langue est précis. Par exemple : si on nous dit « un chat », on aura plus de difficultés à prédire la couleur du chat que si on nous dit « un chat noir ».

1.3.4 Performance et mesure

Le dernier concept important du rapport est celui de performance des modèles produits.

La performance d'un modèle est évalué par trois choses :
— la qualité maximale des résultats produits ; dans le cas d'un modèle de la langue, il s'agit de la qualité de la prédiction ;
— le temps d'entraînement nécessaire pour atteindre cette qualité (nombre d'époque, nombre d'heures, …) ;
— la consommation de ressources nécessaire pour atteindre cette qualité (mémoire, puissance de calcul, …).

Afin d'évaluer les performances, des mesures ont été définies pour :
— la qualité du résultat : l'écart entre le résultat produit et le résultat attendu ; une mesure définie dans la littérature est le BPC, plus elle est proche de 0 mieux c'est ;
— le temps d'entraînement : le nombre d'époque et le temps nécessaire par époque, le nombre total d'heures, … ;
— la consommation de ressources : l'espace mémoire occupée (en MiB ou GiB), la puissance de calcul utilisée, … ;

Un modèle optimal serait un modèle qui atteint des résultats parfaits (aucun écart entre ce qui est attendu et ce qui est produit), le plus rapidement possible, en consommant le moins de ressources possibles.

1.4 Plan du rapport

Dans un premier temps, nous avons présenté à la fois le contexte, les enjeux, et les objectif généraux du stage. Nous avons aussi présenté les principaux termes et concepts utilisés au long du rapport.

Dans un troisième et un quatrième temps nous développerons les deux parties du stage, le projet GMSNN et le projet PAPUD.

Notamment, pour chacun d'entre eux, nous présenterons le contexte, les enjeux, et les entités impliquées dans le projet. Nous décrirons ensuite le modèle implémenté, avant de dérouler le travail effectué. Enfin, une conclusion résumera les points majeurs du projet.

Dans un cinquième et dernier temps, nous ferons une rétrospective sur l'ensemble du travail réalisé.

Première partie

Projet GMSNN

2 Présentation du laboratoire et de l'équipe

2.1 Généralités

C'est dans le laboratoire du LORIA que le stage s'est déroulé, au sein de l'équipe SYNALP dirigée par M. Christophe Cerisara, notre maître de stage.

2.2 Le LORIA

Le Laboratoire Lorrain d'Informatique et ses Applications (LORIA) « est une Unité Mixte de Recherche (UMR 7503), commune à plusieurs établissements : le Centre National de la Recherche Scientifique (CNRS), l'Université de Lorraine (UL) et l'Institut National de Recherche en Informatique et en Automatique (INRIA). » D'après le site du LORIA [3]. Depuis sa création en 1997, le LORIA se concentre sur les science informatiques, que ce soit par la recherche fondamentale ou appliquée.

2.2.1 Structure administrative du LORIA

Il est dirigé par quatre instances, d'après le site du LORIA [4] :

- **l'équipe de direction** : composée du directeur, de son adjoint, de la responsable administrative, et de l'assistante de direction ; ils assistent le directeur dans la prise et la mise en œuvre des décisions ;
- **le conseil scientifique** : composé du directeur du laboratoire, des deux directeurs adjoints et des scientifiques responsables des cinq départements du laboratoire composée de membres élus pour 4 ans et de membres nommés ; ils assistent le directeur dans la prise et la mise en œuvre des décisions ;
- **le conseil de laboratoire** : composé de membres élus pour 4 ans et de membres nommés ; ils émettent des avis et conseillent le directeur sur toutes les questions concernant le LORIA ;
- **l'Assemblée des Responsables des Équipes (AREQ)** : composée des scientifiques responsables des 28 équipes du laboratoire ; ils se réunissent tout les deux mois.

2.2.2 Recherche au sein du LORIA

Le LORIA est l'établissement qui héberge l'équipe SYNALP (*SYmbolic and statistical NAtural Language Processing*), parmi de nombreuses autres équipes.

Ce laboratoire regroupe 28 équipes de recherche, structurées en 5 départements en fonction de leur domaine d'étude.

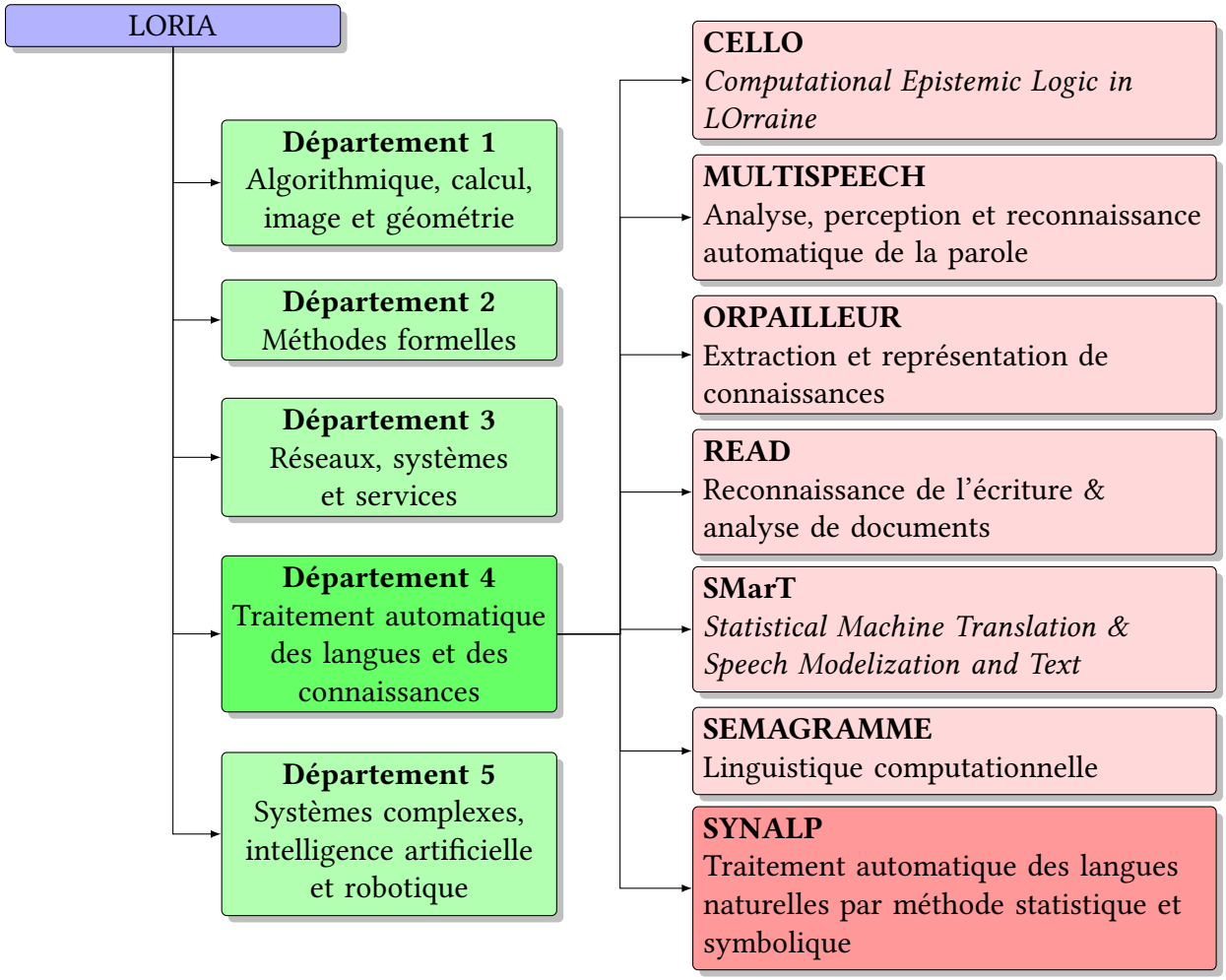


FIGURE 2.1 – Organigramme des départements du LORIA, et des équipes du département 4

La structure générale du LORIA en départements et plus en détail du département 4 est représentée sur l'organigramme de la Figure 2.1. Les thématiques générales de chaque département, ainsi que les thèmes de recherche des équipes du département 4, y sont présentées brièvement. Un organigramme complet du LORIA est disponible sur le site du laboratoire [5].

2.3 Équipe SYNALP

L'équipe SYNALP (*SYmbolic and statistical NAtural Language Processing*) est une équipe de recherche affiliée à la fois au CNRS et à l'Université de Lorraine, deux établissements dont nous avons parlé plus haut. Elle fait partie, avec 6 autre équipes, du département 4, dédié au traitement automatique des langues (TAL) et des connaissances.

2.3.1 Membres

L'équipe SYNALP est sous la direction de M. Christophe Cerisara, et comporte actuellement 12 membres permanents, une dizaine de doctorants et d'ingénieurs, et approximativement 6 stagiaires à l'heure de l'écriture de ce mémoire.

2.3.2 Thématiques de recherche

La recherche dans SYNALP se concentre sur les approches hybrides, symboliques et statistiques du TAL, ainsi que sur les applications de ces approches.

Ainsi, les principaux sujets de recherche de l'équipe sont les modèle de la langues, les grammaires formelles, la sémantique computationnelle, le traitement de la parole, et les outils et ressources utilisés en TAL.

Ce stage s'inscrit en particulier dans la réalisation de modèle de la langue, et l'élaboration d'outils et ressources utilisés en TAL. Nous verrons en détail pourquoi dans le chapitre 3.

2.4 Pour en savoir plus

Des informations plus détaillées sur le LORIA sont disponible sur le site du laboratoire [3]. Par ailleurs, la liste complète des membres de l'équipe, ainsi que des informations plus détaillées sont disponible sur le site de l'éqipe SYNALP (en anglais) [6].

3 Architecture innovante de réseau de neurones pour l'élaboration d'un modèle du langage

3.1 Contexte

Les modèles neuronaux actuellement utilisés en TAL, généralement basés sur les RNN, atteignent de très bonnes performances similaires dans certains cas à celles des humains [7, 8, 9].

Les modèles basés sur les caractères se montrent particulièrement flexibles, car ces modèles « apprennent » les mots. Au contraire, les modèles basés sur les mots se reposent sur des dictionnaires, qui sont très volumineux et ont des difficultés à gérer les fautes et les mots nouveaux.

Ces performances sont obtenues entre autres grâce à une gestion du contexte des exemples, typique des modèle de la langue.

3.1.1 Manque d'utilisation des gros volumes de données

Cependant, ces modèles sont souvent développés et entraînes avec peu de donnéess. Les raisons envisageables sont principalement le manque de donnéess brutes ou préparées, et le peu d'amélioration de performance malgré des coûts largement augmentés.

3.1.2 Problèmes de mémoire

Une des raison du manque d'augmentation de performance, typique des RNN, est la limite de rappel d'informations en mémoire.

Pour avoir un ordre d'idée, on peut considérer qu'un RNN basique conserve en mémoire des informations datant d'au plus 20 entrées auparavant ; d'autres architectures peut se rappeler d'informations vielles d'une 100$^{\text{aine}}$ d'entrées ; et un LSTM dépasse difficilement les 200 entrées.

Il est donc difficile d'apprendre des dépendances entre des éléments très écartés.

De nombreuses tentatives ont été faite de résoudre ce problème, par exemple en changeant l'architecture du réseau de neurones artificiels (ex : LSTM), ou en augmentant le réseau avec des mécanismes comme de la mémoire explicite (mémoire plus performante).

3.2 Solution proposée

L'architecture proposée par notre maître de stage vise à la fois à tirer partie des grands volumes de donnéess, et à permettre au modèle d'établir des dépendances de haut niveau, voir des connaissance contextuelles externes (c'est-à-dire étendre le contexte au-delà des informations directement accessibles, inférer des vérités générales).

L'architecture et ses caractéristiques sont décrits en détail dans le chapitre 5.

Nous avons nommé cette architecture GMSNN. Ainsi, nous désignerons ce projet par « projet GMSNN » dans le reste du rapport.

3.3 Projet GMSNN

La mission qui nous à été confiée est la création d'un modèle de la langue basé sur l'architecture GMSNN.

L'implémentation devait se réaliser à partir d'une base de code sur laquelle notre maître de stage avait commencé à travailler (plus de détails sont disponibles dans la sous-section 6.2.1).

À cela s'ajoutait l'exploration du potentiel de l'architecture en améliorant le modèle créé, par le biais d'optimisations classiques et de changements de l'architecture.

Enfin, la réintégration des optimisations déjà contenues dans la base de code devait conclure le stage.

3.4 Organisation du travail

Durant ce projet, nous avons travaillé individuellement.

Un fonctionnement en rapport réguliers (disponibles dans l'annexe A, page 79), complémentés d'une occasionnelle correspondance électronique, à permis de tenir notre maître de stage informé de l'avancement du stage.

À cela s'ajoutent des réunions hebdomadaires avec M. Cerisara, afin de faire le point sur les résultats obtenus et de décider de la marche à suivre.

3.4.1 Organisation initiale du travail

Dès la prise de connaissance du sujet définitif du stage, nous avons pu prévoir l'organisation temporelle du travail.

La première semaine était dédiée l'acquisition des connaissances nécessaires, à la lecture d'articles et à la prise en main des outils. Ensuite, 3 semaines étaient consacrées à la prise en main de la base de code fournie et à l'implémentation d'un prototype. Les 4 semaines suivantes devaient permettre d'améliorer l'architecture et d'intégrer de nouvelles fonctionnalités. Enfin, les optimisations état de l'art contenue dans la base de code devaient être intégrée durant les 4 dernières semaines .

La Figure 3.1 représente cette répartition prévue du travail.

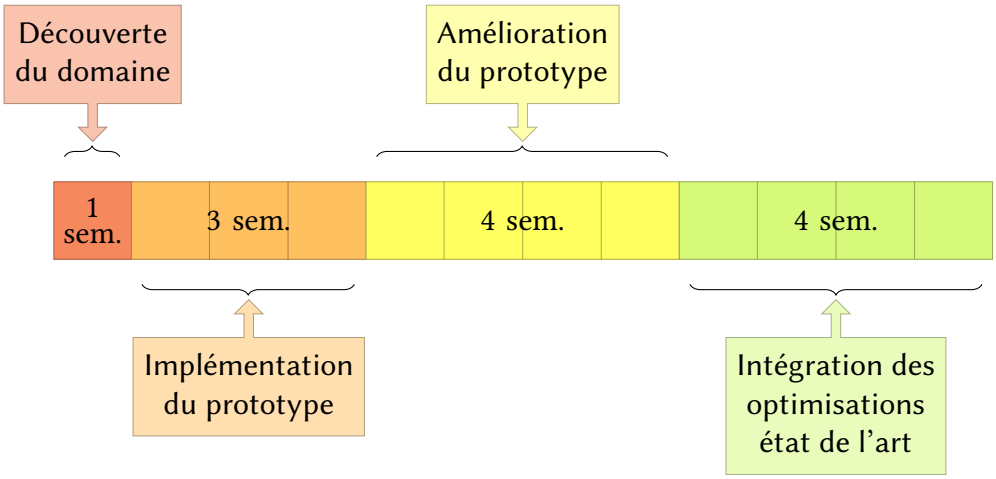


FIGURE 3.1 – Répartition prévue du travail. Une case correspond à une semaine de travail.

3.4.2 Déroulement réel du projet

Le projet s'est déroulé comme prévu jusqu'à la fin de la période d'amélioration.

Cependant, comme décrit section 6.7, nous avons décidé d'interrompre ce projet pour nous consacrer au projet PAPUD.

La Figure 3.2 représente la répartition réelle du travail.

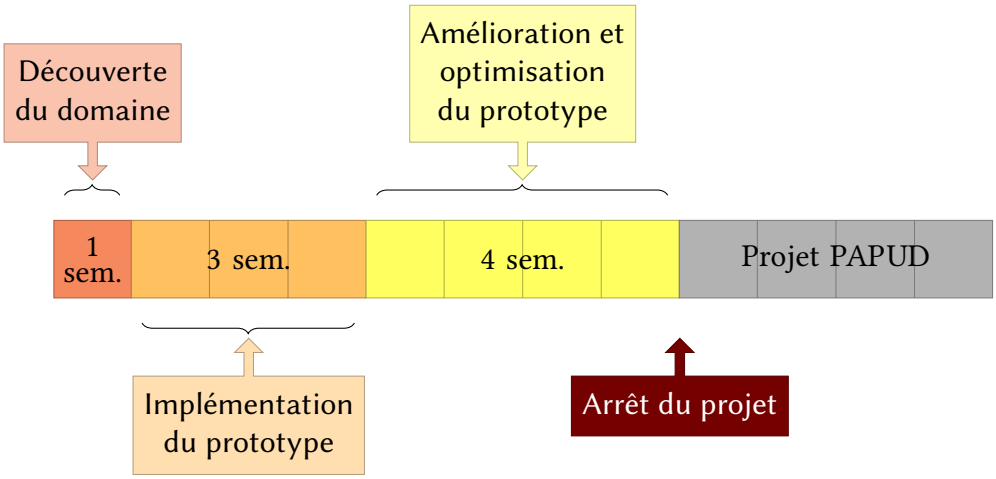


FIGURE 3.2 – Répartition réelle du travail. Une case correspond à une semaine de travail.

3.5 Outils

3.5.1 Gestion du code

Le code et les rapports sont stockés sur les serveurs Gitlab de l'Inria, avec le système de gestion de version Git.

3.5.2 Langage et librairie

Le langage choisi pour l'implémentation est Python, qui largement fourni en outils et librairies d'apprentissage profond.

Parmi ces librairies, notre choix c'est porté sur PyTorch, qui contrairement à d'autres librairies telles que Caffe ou Keras, permet de moduler l'architecture du réseau au cours de l'apprentissage. Cette propriété est très importante, étant donné la nature « croissante » de l'architecture proposée. De plus, cette librairie est particulièrement bien documentée.

3.5.3 Grid5000 et les machines distantes

Pendant le déroulement du projet, le modèle à été testé et entraîné sur des machines distantes.

Ces machines font partie du réseau Grid5000. « Grid'5000 est un banc d'essai à grande échelle et polyvalent pour la recherche expérimentale dans tous les domaines de l'informatique, avec un accent sur l'informatique parallèle et distribuée, y compris Cloud, HPC et Big Data. » D'après le site de Grid5000 [10].

Un des avantage de cet outil est la présence de machines spécialisé dans les calculs GPU, qui sont celles que nous avons utilisées. En effet, l'utilisation de GPU pour l'entraînement des réseau de neurones artificiels est une pratique fréquente en apprentissage profond, car elle permet d'accélérer les calculs effectués.

4 Données disponibles

Les données d'entraînement utilisées proviennent du Wikipédia anglais. Elles sont tirée du fichier « enwik8 » utilisé pour le prix Hutter [11, 12]. Ce fichier est composé d'environ 100 000 000 caractères.

Ces données sont composées de texte balisé structuré en paragraphes. Quelques fragment de XML [1] sont aussi présents, mais ils sont minoritaires dans les données. Il est donc peu probable de les retrouver dans les données apprises.

Deux versions alternatives de ce corpus ont été utilisées.

1. la première est composée des 10 000 000 premiers caractères de « enwik8 » ; cette version à servi aux entraînements et à la plupart des tests du modèle ;
2. la seconde est composée des 1 000 000 premiers caractères de « enwik8 » ; elle à servi pour le débogage du modèle.

4.1 Extrait des données d'entraînent

Le Fragment de code 4.1 un extrait des données brutes avant le découpage en caractères.

```
1 While anarchism is most easily defined by what it is against, anarchists also
    offer positive visions of what they believe to be a truly free society.
    However, ideas about how an anarchist society might work vary considerably
    , especially with respect to economics; there is also disagreement about
    how a free society might be brought about.
2
3 == Origins and predecessors ==
4
5 [[Peter Kropotkin|Kropotkin]], and others, argue that before recorded [[
    history]], human society was organized on anarchist principles.&lt;ref&gt
    ;[[Peter Kropotkin|Kropotkin]], Peter. ''&quot;[[Mutual Aid: A Factor of
    Evolution]]&quot;'', 1902.&lt;/ref&gt; Most anthropologists follow
    Kropotkin and Engels in believing that hunter-gatherer bands were
    egalitarian and lacked division of labour, accumulated wealth, or decreed
    law, and had equal access to resources.&lt;ref&gt;[[Friedrich Engels|
    Engels]], Freidrich.
6 [[Image:WilliamGodwin.jpg|thumb|right|150px|William Godwin]]
```

Fragment de code 4.1 – Extrait des premières lignes du fichier enwik8, correspondant à l'article Wikipédia sur l'ansarchisme.

1. XML (*eXtensible Markup Language* en anglais), « est un langage informatique qui sert à enregistrer des données textuelles. [...] Ce langage , grosso-modo similaire à l'HTML de par son système de balisage, permet de faciliter l'échange d'information sur l'internet. » D'après le glossaire sur *infowebmaster* [13]. Il s'agit du format sous lequel sont stockés les articles Wikipédia.

4.2 Prétraitement des données

Le prétraitement des données est composé du découpage du document en caractères et du remplacement des caractères par des nombres, à l'aide d'un dictionnaire.

Comme mentionné dans la sous-section 6.6.2, un défaut dans le prétraitement à mené à la disparition des espaces du texte.

En effet, le prétraitement d'origine du corpus utilisait les espaces en tant que séparateurs pour le stockage des données. Par la suite, au moment d'utiliser les données pré-traitées, l'intégralité des espaces étaient supprimés, y compris ceux du texte d'origine.

Malheureusement, ce défaut à été découvert à la fin du projet, et n'à pas pu être corrigé à temps.

5 Description de l'architecture proposée

5.1 Propriétés du modèle

Pour rappel, l'architecture proposée à pour but d'établir un modèle de la langue.

Elle est caractérisée par trois propriétés majeures :
— la structure récurrente ;
— l'utilisation de plusieurs échelles ;
— la croissance du modèle.
Nous avons nommé cette architecture réseau de neurones récurrents multi-échelles croissant (*Growing Multi-Scale Recurrent Neural Network* en anglais, GMSNN) en considérant ses principales caractéristiques.

5.1.1 Récurrence du modèle

Comme souvent dans la réalisation de modèle de la langue, on peut considérer les données sous forme de séquence.

Dans notre cas, le caractère à prédire est dépendant de la suite de tous les caractères précédents.

Pour rappel, en apprentissage profond, le type de réseau de neurones artificiels considéré le plus adapté à la manipulation de séquences est le RNN.

C'est pour ces raisons que l'architecture à été conçue à partir de RNN.

5.1.2 Passer à l'échelle

Comme décrit sous-section 3.1.2, les RNNs ont un problème inhérent de capacité mémoire, qui limite la distance des dépendances apprises par le modèle.

Afin de compenser ce défaut, l'architecture réseau de neurones récurrents multi-échelles croissant (*Growing Multi-Scale Recurrent Neural Network* en anglais, GMSNN) s'appuie sur des couches de plus en plus vastes, appelées « échelles » dans ce rapport. Chacune de ces couches est un RNN.

Chaque échelle supplémentaire permet de modéliser des dépendances sur de plus grandes distances.

De plus, chaque échelle tire ses informations de l'échelle précédente. L'exemple suivant explique ce mécanisme de transfert de l'information, également illustré sur la Figure 5.1, page 24.

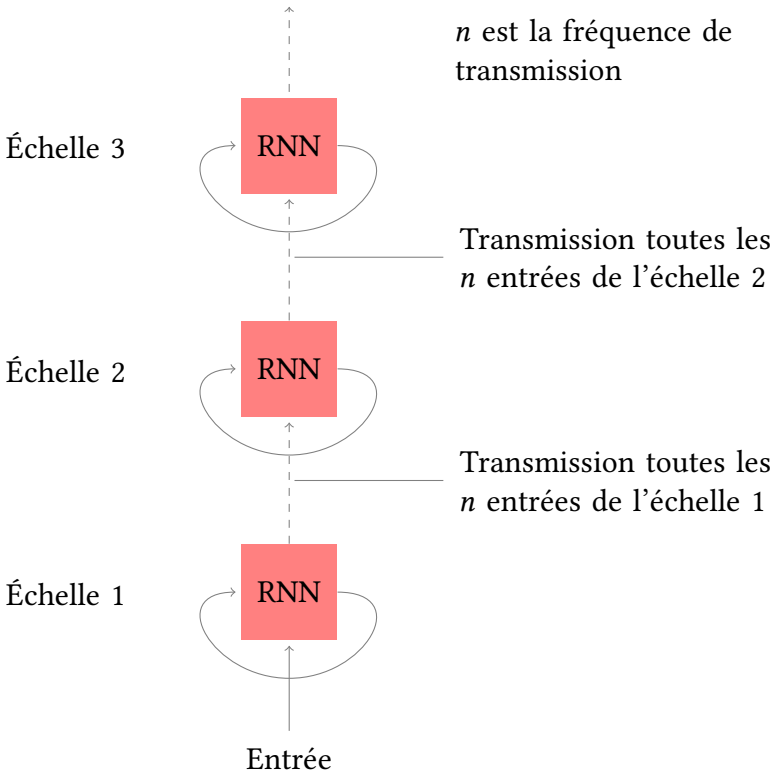


FIGURE 5.1 – Principe de transmission de l'information d'une échelle à la suivante. Ici, la fréquence de transmission est notée n.

Admettons que la capacité de mémoire d'un RNN est de 9 entrées (nombre arbitrairement défini pour l'exemple). Si l'échelle supérieure récupère des informations toutes les 3 entrées de la couche inférieure, ca capacité de mémoire devient $9 * 3 = 27$ entrées. L'échelle encore au dessus aura une capacité mémoire de $9 * 3 * 3 = 81$ entrées, et ainsi de suite. Ici le nombre 3 représente la fréquence à laquelle une échelle transmet des informations. On parlera par la suite de « fréquence de transmission ».

Plusieurs niveaux d'abstraction de l'information

Une autre caractéristique importante du GMSNN est qu'une échelle prend en entrée les informations abstraites par la couche précédente. Ainsi, on peut s'attendre à ce que chaque échelle ajoute un niveau d'abstraction supplémentaire au modèle, comme sur la figure 5.2, page 25.

5.1.3 Adapter le modèle au volume de données et croissance du modèle

Une propriété de l'architecture dérivée du principe d'échelles est de s'adapter au nombre d'entrées.

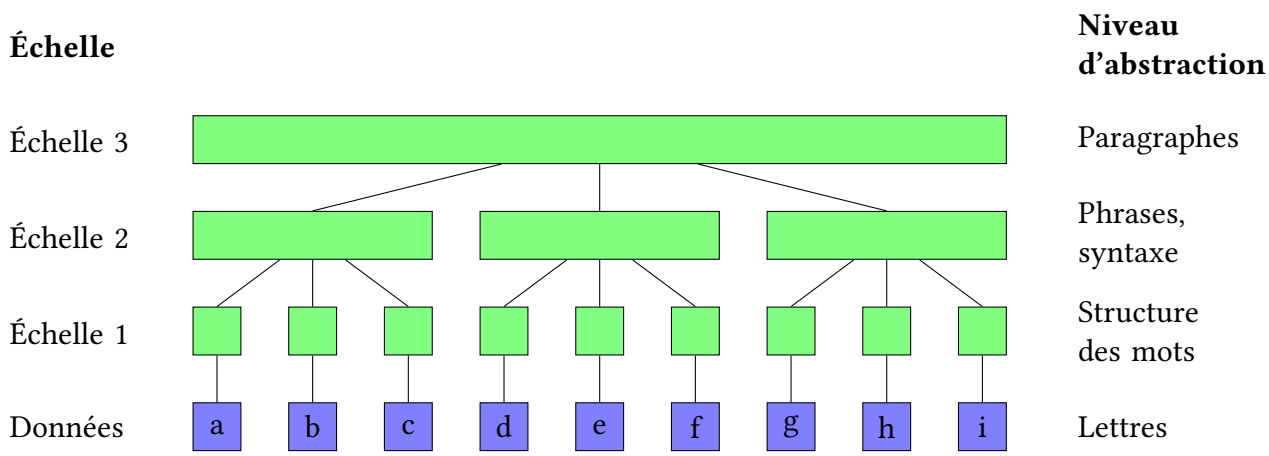


FIGURE 5.2 – Différentes échelles, et les niveaux d'abstraction que l'on pourait attendre de celles-ci. Chaque bloc vert correspond à une entrée pour l'échelle correspondante. Les blocs bleus en bas du graphique correspond aux caractères qui sont fourni en entrée au modèle. Ici la fréquence de transmission vaut 3.

En effet, comme décrit dans la sous-section 5.1.2, page 23, le nombre d'échelles est dépendant du nombre d'entrées totales.

On peut considérer que tant que aucune entrés ne lui est fournie, une échelle reste dans son état initial, elle n'« existe » pas ; par conséquent, les échelles qui en sont dépendantes n'« existent » pas non plus.

Ainsi, au fur et à mesure de l'entraînement, le modèle croit.

Il existe une formule pour déterminer le nombre de couches « existantes » en fonction du nombre d'entrées présentées et de la fréquence de transmission :

$$n = \lfloor \log_f i \rfloor + 1$$

n : nombre de couches, i : nombre d'entrées, f : fréquence de transmission

Croissance potentiellement infinie du modèle

Il est envisageable d'adapter le modèle au nombre d'entrées *durant l'entraînement*, en créant réellement les échelles au fur et à mesure que l'on fournit les données.

Dans ce cas, tant que l'on lui fournit des données, la croissance du modèle est potentiellement infinie.

Comme décrit dans la sous-section 6.5.1, page 31, cette propriété à été abandonnée.

6 Réalisation

6.1 Recherche documentaire

La première partie du projet à été la recherche documentaire et la prise en main des outils. Ce travail à été effectué à partir des documents fournis par notre maître de stage, de la documentation de PyTorch [14, 15, 16, 17] et de Grid5000 [19, 18], complétés par nos recherches personnelles.

6.2 Étude et ré-implémentation simplifiée du modèle état de l'art

6.2.1 Travail effectué

La deuxième partie du projet à été la prise en main de la base de code fournie. Il s'agit d'une implémentation état de l'art d'un modèle de la langue au niveau du caractère, sur laquelle notre maître de stage avait commencé à travailler. Le code d'origine provient du dépôt « awd-lstm-lm »[20], qui contenait un modèle état de l'art de modèle de la langue au niveau du caractère.

Au début du stage, la base de code contenait :
- la version d'origine du dépôt ;
- un début de ré-implémentation simplifiée du modèle de la version d'origine ; cette version devait servir de base pour développer le modèle du GMSNN, ainsi que de comparaison pour les performances du nouveau modèle ; elle comportait quelques bogues et ne fonctionnait pas en l'état ;
- un début de travail sur l'architecture du GMSNN.

L'objectif de cette étape était de faire fonctionner la ré-implémentation simplifiée du modèle.

Pour cela, nous avons déchiffré et re-documenté le code, qui contenait des fragments obsolètes et peu documentés. Après le déchiffrage, il à fallu comprendre et réparer les fragments défectueux.

6.2.2 Modèle ré-implémenté simplifiée

Le modèle simplifiée produit est composé d'un module encodant les caractères, d'un RNN particulier (un LSTM), et d'un module produisant une distribution de probabilité sur les caractères connus. La figure 6.1, page 28, représente cette architecture.

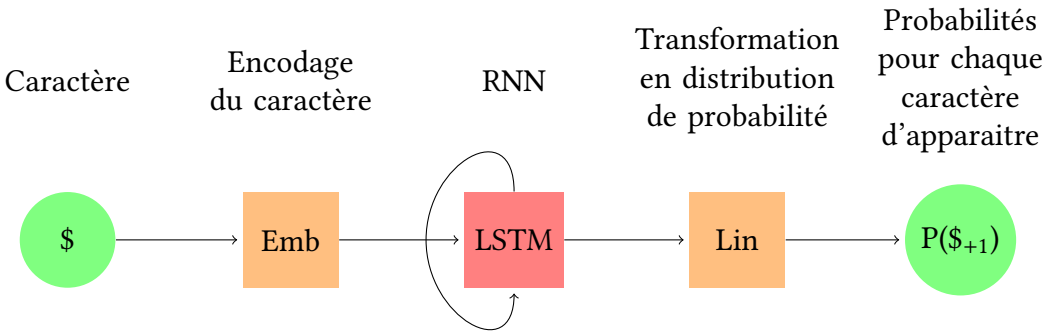


FIGURE 6.1 – Architecture du modèle réimplémenté. Le modèle prend en entrée des caractères, et produit des probabilités sur quel caractère apparaîtra ensuite.

Le module d'encodage des caractères, appelé *embeding layer* en anglais (littéralement « couche d'inclusion »), produit une représentation apprise de chaque caractère sous forme de tenseur. Ce tenseur est appelé *embedding*. Ce module, entraîné, peut apprendre des propriétés spécifiques à chaque caractère. Par exemple ce module peut apprendre que tel caractère est une consonne et que tel autre est un caractère de ponctuation.

Le RNN traite les caractères sous forme de séquence, et peut ainsi apprendre la structure des mots, la syntaxe et d'autres propriétés du langage.

Le module produisant la distribution de probabilité est un module linéaire [1]. Il transforme les informations produites par le réseau de neurones en probabilité pour chaque caractère d'être le prochain caractère de la séquence.

Voir les annexes A.8, A.9 et A.10 pour les rapports sur le modèle ré-implémenté.

1. « Module linéaire » est le nom donné aux modules composé d'un réseau de neurones artificiels intégralement connecté. Un réseau de neurones intégralement connécté signifie que chaque neurone d'une couche est connecté avec tous les neurones de la couche précédente. Il s'agit de l'architecture de réseau de neurones artificiels la plus simple.

6.3 Implémentation du nouveau modèle

6.3.1 Travail effectué

La troisième partie du projet à été la réalisation d'un prototype de l'architecture GMSNN, basé sur la ré-implémentation simplifiée du modèle état de l'art.

L'architecture du GMSNN est identique à celle du modèle ré-implémenté, mis à part le RNN qui est remplacé par le module GMSNN (voir figure 6.2). C'est sur ce nouveau module GMSNN que le reste du travail au cours du projet GMSNN à été effectué.

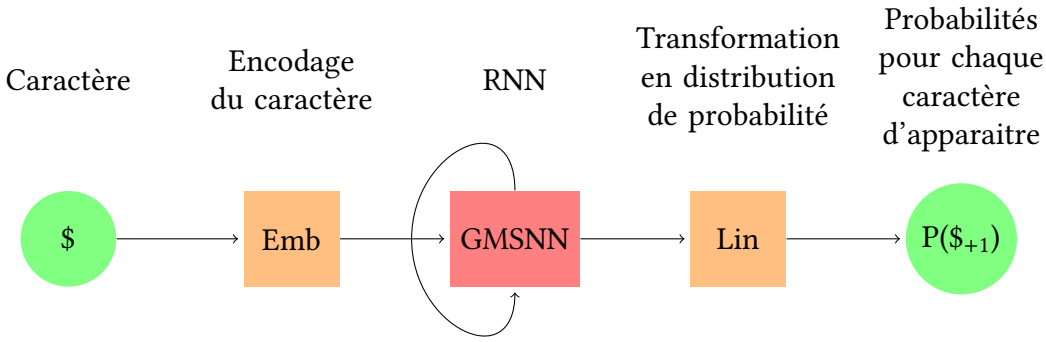


FIGURE 6.2 – Architecture du modèle réimplémenté. Le modèle prend en entrée des caractères, et produit des probabilités sur quel caractère apparaîtra ensuite.

Ce prototype à permis de mettre en place les mécanismes de base du modèle.

Durant cette étape, nous avons mis en place l'architecture multi-échelle avec deux mécanismse fondamentaux : la transmission de l'information d'une échelle à l'autre, et l'agrégation de l'information de toutes les échelles.

Chaque échelle qui compose le module GMSNN est un LSTM, qui est le RNN utilisé dans le modèle d'origine.

6.3.2 Transmission d'information

Pour rappel, la transmission d'information se fait d'une couche à la couche supérieure. Cette transmission se fait périodiquement, en fonction d'un nombre appelé fréquence de transmission.

Dans un premier temps, il à fallu choisir quelle information transmettre d'une échelle à l'échelle supérieure. En effet, les RNNs produisent à la fois une sortie, et un tenseur contenant leur mémoire. L'utilisation de l'*embedding* à été écarté initialement, car elle n'est pas en accord avec le principe d'abstraction de l'architecture proposée.

Le choix s'est porté sur le tenseur contenant la mémoire, qui contient don les informations abstraites par l'échelle, contrairement à la sortie qui contient uniquement les informations pour prédire le caractère suivant.

Voir l'annexe A.11 pour le rapport sur le prototype.

6.4 Intégration de systèmes de visualisation

Afin d'évaluer les performances du modèle dans la suite du projet, il à été nécessaire d'établir un système de visualisation des performances.

6.4.1 Utilisation de librairies

Dans un premier temps, diverses librairies permettant de visualiser l'état du réseau de neurones artificiels ont étés testées, en particulier VisualDL [21, 22].

Malheureusement, ces librairies ont des difficultés à supporter les architectures complexes (en particulier celles qui impliquent des RNN).

Ainsi, aucune des librairies testées n'a fonctionné avec notre modèle.

6.4.2 Création d'un outil personnalisé

Nous avons donc réalisé un outil capable d'enregistrer des données et de réaliser des graphiques. Nous nous sommes basés sur le module « matplotlib » de Python, et sur une variante française du format CSV. Il s'agit d'un format simple à manipuler qui permet de stoker facilement des lignes de données, et de définir le nom de chaque colonne du tableau ainsi obtenu.

L'outil à évolué tout au long du projet pour s'adapter à nos besoins.

Il nous a permis de réaliser les graphiques produits dans les divers rapports du projet (disponibles en annexes).

6.5 Optimisation et amélioration du nouveau modèle

Une fois le prototype fonctionnel, nous avons amélioré ses performances. Par performances, nous entendons principalement le temps nécessaire pour que la qualité prédictive du modèle dépasse un certain seuil.

Pour améliorer ce temps d'entraînement, il est possible de travailler sur deux dimensions :
- la *quantité de donnéess* traitées en un laps de temps ; pour cela on peut optimiser les algorithmes et le modèle pour réduire le temps nécessaire pour traiter les exemples ; c'est une *stratégie quantitative* ;
- la *qualité* de l'apprentissage pour une quantité fixée de donnéess ; pour cela on peut améliorer le modèle en réglant les paramètres (comme la fréquence de transmission) ou en implémentant de nouvelles mécaniques ; c'est une *stratégie qualitative*.

Les deux stratégies ont été utilisées. Il faut noter que certaines améliorations qualitatives ont un impact quantitatif négatif.

Principalement, le travail effectué pendant cette partie du projet est un travail de débogage, d'analyse et d'optimisation, avec peu d'implémentation de nouvelles mécaniques dans le modèle.

6.5.1 Agrégation des sorties des couches : d'une stratégie additive à une concaténation

La première optimisation à été de changer la façon de regrouper les informations de toutes les « échelles » avant de les transmettre au module produisant la distribution de probabilité.

Initialement, les sorties de toutes les « échelles » étaient sommées. Cela permettait de maintenir des tenseurs de dimensions uniformes quel que soit le nombre d'« échelle » (voir figure 6.3a).

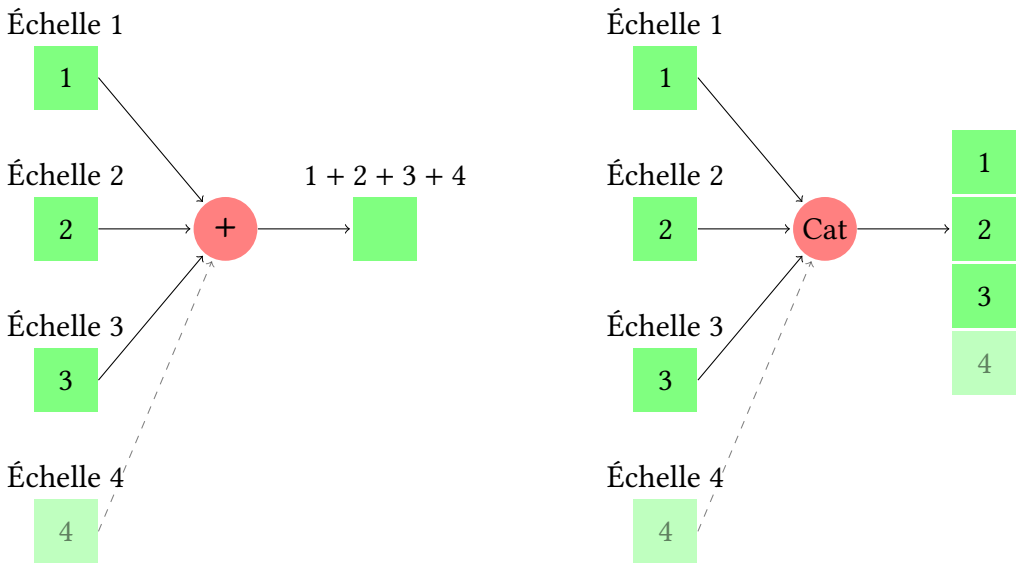
Après discussion avec notre maître de stage, la stratégie d'agrégation à été changé en une concaténation des sorties.

Comme montré dans la figure 6.3b, la taille du tenseur concaténé change en fonction du nombre d'entrées. La manipulation de tenseurs de taille non fixée est très ardue dans ce cas précis, bien que nous ne développerons pas plus avant les raisons de cette difficulté.

Cela à nécessité l'abandon de la propriété de croissance à l'infini de l'architecture (décrite sous-section 5.1.3), au profit d'un nombre maximal d'échelles défini à l'avance ou déterminée à l'aide d'une formule en fonction des données disponibles (décrite sous-section 5.1.3).

La stratégie par concaténation est plus lente en terme de temps de calcul que la stratégie additive, cependant pour le même temps de calcul elle permet d'obtenir de meilleurs résultats (voir figure 6.4).

Voir l'annexe A.12, page 109, pour plus de détail sur le choix de la stratégie d'agrégation.



(a) Stratégie d'agrégation additive. Les sorties sont additionnées afin de former un nouveau tenseur.

(b) Stratégie d'agrégation par concaténation. Les sorties sont mises côte-à-côte afin de former un nouveau tenseur.

FIGURE 6.3 – Stratégies d'agrégation

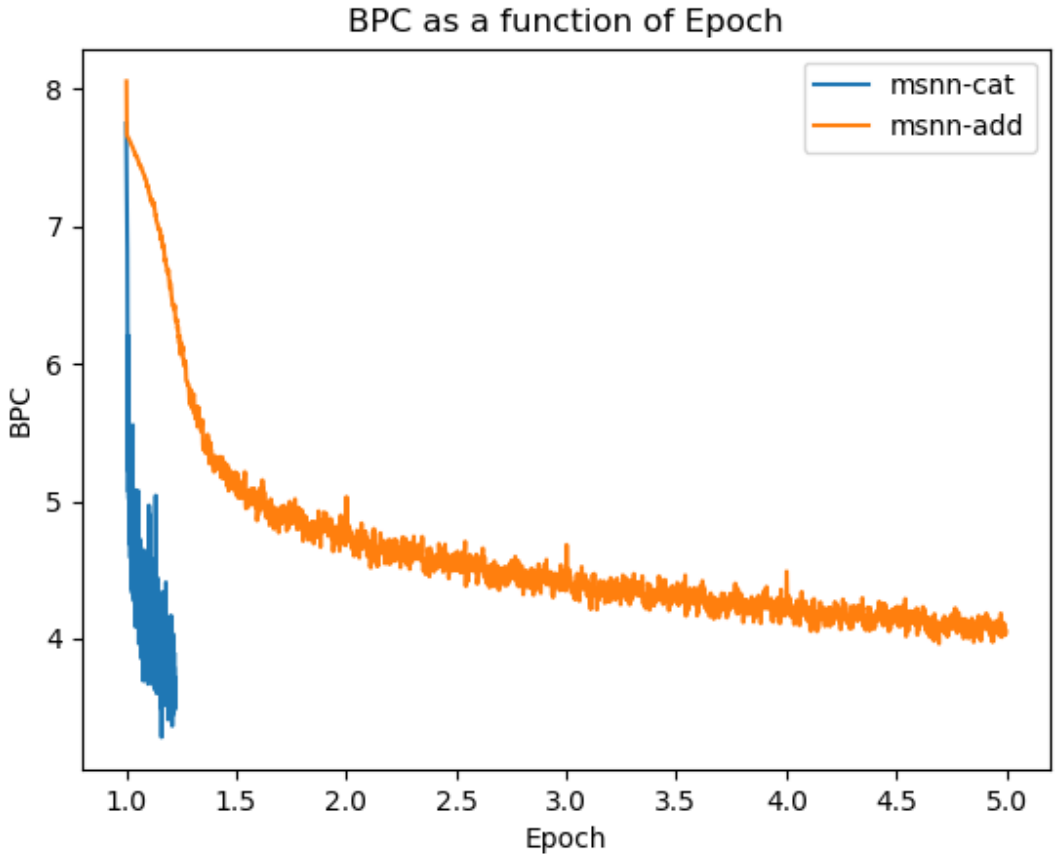


FIGURE 6.4 – Performances comparées des stratégies par concaténation *(msnn-cat)* et additive *(msnn-add)*. Le temps de calcul alloué est identique. Avec la concaténation on entraîne le modèle sur 1/4 des données, avec l'addition on l'entraîne 5 fois sur l'ensemble des données. Avec la concaténation, on obtient une BPC de 3.5, alors qu'on obtient une BPC de 4 avec l'addition.

6.5.2 Sauvegarde, interruption et reprise de l'entraînement

Une fonctionnalité s'est très vite détachée comme essentielle : la sauvegarde du modèle, et l'interruption et la reprise de l'entraînement.

En effet, avec des entraînements très lents et donc longs, il était nécessaire de pouvoir suspendre l'entraînement afin de répartir le temps d'entraînement sur plusieurs sessions de plusieurs heures. De plus, les sauvegardes permettaient de conserver le modèle une fois qu'il est entraîné.

Le système implémenté permet d'effectuer cycliquement des sauvegarde du modèle ainsi que de l'état de l'entraînement, permettant ainsi une reprise en l'état de l'entraînement.

Pour la réalisation du système, le principal obstacle à été le malfonctionnement initial des outils fournis par PyTorch. Cela à poussé à la conception d'un système de sauvegarde personnalisé mais malheureusement assez complexe. Cependant, la mise-à-jour majeure de la librairie qui c'est déroulé à point nommé à résolu le problème, et c'est avec les outils de PyTorch que le système de sauvegarde à finalement été implémenté.

Voir l'annexe A.3 pour un rapport contenant plus de détail sur le système de sauvegarde.

6.5.3 Tentatives d'optimisations, fuites mémoires et lenteur de l'entraînement

Les optimisations tentées par la suite ont révélé des fuites mémoires et mis en avant une lenteur excessive de l'apprentissage.

Les optimisations en questions ont été mises en suspens le temps de la résolution de ces deux problèmes. Il s'agit de :
— l'utilisation de *batchs* simultanés (voir sous-section 6.5.4, page 34) ;
— l'augmentation du nombre de paramètres du modèle (voir sous-section 6.5.5, page 35) ;

Consommation de mémoire et de temps de calcul accrue

Un effet direct des optimisations tentées est l'augmentation de la consommation mémoire.

Cette consommation accrue à causé le plantage [2] de plusieurs tests, révélant la présence de fuites mémoires critiques. Un ralentissement progressif de l'entraînement à aussi été découvert pendant l'analyse du problème. Le plus surprenant a été la corrélation forte découverte entre le temps de calcul et la consommation de mémoire.

Un premier correctif à fourni une amélioration notable mais insuffisante. Il remplaçait le LSTM de chaque couche (voir sous-section 6.3.1) par un RNN basique, moins gourmand. Cela permettait aussi d'éliminer une redondance entre l'architecture LSTM et l'architecture GMSNN.

2. Un plantage en informatique est l'arrêt d'un programme lié à un dysfonctionnement.

Estimation de la consommation normale du modèle

La première étape, qui est détaillée dans l'annexe A.2, à été d'estimer l'usage normal de la mémoire (sans fuite), et d'isoler les paramètres qui ont le plus d'impact sur la consommation mémoire. Cela à confirmé que l'explosion de la consommation n'était pas du à l'architecture en elle même, et qu'il s'agissait bien d'une anomalie.

Résolution des fuites

L'analyse et la résolution des fuites mémoires s'est révélée ardue. Si quelques fuites mineures ont été simple à détecter et réparer, la principale fuite était due à une spécificité non documenté de PyTorch.

En effet, PyTorch utilise la différentiation automatique pour mettre à jour les poids du réseau de neurones artificiels. Pour cela, PyTorch à besoin de connaître la suite d'opérations et l'implication des différents paramètres du modèle et se base sur un « graphe de computation ». C'est la façon dont est gérée ce graphe, couplée aux spécificités de l'architecture GMSNN qui est la cause de la principale fuite mémoire.

L'annexe A.4 contient le début de la résolution du problème.

Conclusion

Le problème de la fuite mémoire à été résolu, et avec lui celui de la lenteur de l'entraînement. On peut déplorer de ne pas avoir analysé plus avant cet étrange lien entre la mémoire et le temps d'entraînement. Cependant, la résolution des fuites mémoires et de la lenteur de l'entraînement était l'objectif principal de cette étape, et l'optimisation du module GMSNN à pu reprendre.

On peut noter l'ampleur de l'optimisation par rapport à la version initiale :
- le temps de d'entraînement à été réduit par un facteur 5 000 (de plus de 400 h à environ 5 min pour une époque) ;
- la consommation mémoire est passée d'une utilisation en constante augmentation, dépassant les 6 GiB par époque, à une consommation constante inférieure à 200 MiB.

6.5.4 Entraînement par exemples simultanés

Une fois le problème des fuites mémoires résolu, la première optimisation mise en place est l'utilisation de *batchs* parallèles.

Batch

Un *batch* (anglais pour lot), est un groupe d'exemples successifs.

Le découpage des données en *batchs* permet de répartir l'apprentissage tout au long de l'étude des données. Cela permet d'atteindre de meilleurs performances. L'algorithme basé sur ce principe est appelé *mini-batch* [23].

Il s'agit d'une optimisation rependue pour l'entraînement de réseaux de neurones [23]. Elle est souvent couplée à un entraînement simultané sur plusieurs *batchs*, décrit dans la partie suivante.

Batchs parallèles

Un entraînement par *batchs* parallèles permet de calculer le résultat de plusieurs exemples simultanément. On calcule ensuite la différence de chaque résultat avec le résultat attendu correspondant. Enfin, on met à jours le modèle en fonction de l'ensemble des exemples. Au final, les calculs des résultat sont parallélisé, et le coût de la mise à jour est mis en commun entre plusieurs exemples.

Cela permet de réduire drastiquement le temps de calcul et d'améliorer la qualité de l'entraînement, au prix d'une plus grande utilisation de la mémoire et de la puissance de calcul.

La version de l'algorithme de parallélisation utilisé est similaire à celle décrite dans l'article [24]. Elle est gérée de base par PyTorch.

Conflit entre les *batchs* parallèles et l'architecture GMSNN

Cependant, le découpage en *batchs* pose un problème majeur avec l'architecture GMSNN : elle est basée sur la continuité des exemples fournis, et l'utilisation de *batchs* brise la continuité en introduisant un parallélisme.

Une analyse approfondie à permit d'établir une méthode pour résoudre ou écarter la majorités des aspect du problème. Après consultation avec notre maître de stage, nous avons décidé d'utiliser l'entraînement par *batchs* malgré les problèmes restants.

Voir le rapport de l'annexe A.5 pour les détails de l'analyse des problèmes théoriques de l'utilisation de *batchs* avec l'architecture GMSNN.

Voir l'annexe A.2 pour les détails sur l'impact de la taille des *batchs* et du nombre de *batchs* sur la consommation mémoire.

Voir les annexesA.13, A.16, A.17 et A.18 pour les rapports des tests sur la rotation des *batchs*.

6.5.5 Augmentation du nombre de paramètres

Pour rappel, les paramètres du modèle sont des valeurs qui varient au long de l'entraînement du modèle.

Le fait d'augmenter ces paramètres augmente la qualité de l'apprentissage, et la précision du modèle appris. Mais cela se fait au prix d'un volume plus important du modèle, et d'une augmentation des calculs nécessaires pour utiliser et entraîner le modèle. Cela se manifeste par un entraînement plus lent et une consommation mémoire plus élevée.

Cependant, grâce aux optimisations mises en place durant la résolution des fuites mémoires (voir sous-section 6.5.3), ces coûts restent raisonnables.

Il existe plusieurs façon de mettre en place cette optimisation :
— augmenter le nombre de neurones par couches ;

— augmenter le nombre de couches dans le RNNs qui compose chaque « échelle ».

Ces deux pratiques ont été testées, et aucune n'a apporté d'amélioration de qualité de l'apprentissage, tout en multipliant le temps d'entraînement. En résumé, ces améliorations apportaient des coûts supplémentaires sans aucun bénéfices. Par conséquent, aucune de ces améliorations n'a été conservée.

6.5.6 Entraînement couche par couche

La dernière optimisation mise en place est un nouvel algorithme d'entraînement.

Cet algorithme est une implémentation naïve d'un entraînement couche par couche appliqué à l'architecture GMSNN. Cette algorithme s'apparente aux algorithmes HM.

L'intuition à l'origine de l'algorithme est : « il semble que pour apprendre des représentations de haut niveau, le modèle doit en premier lieu apprendre les représentations de bas niveau ; en effet, sans mots, il est difficile de faire des phrases cohérentes ».

Cela sous-entend que les échelles les plus proches des données doivent apprendre avant que les échelles supérieures puisse le faire à leur tour. Aussi, il semble inutile d'augmenter la charge de l'algorithme d'entraînement en entraînant des couches qui n'apprennent pas.

Le fonctionnent général de cette algorithme est d'entraîner successivement, une à une, les échelles du modèle en commençant par celle la plus proche des données.

Le fonctionnement détaillé de l'algorithme est disponible dans l'annexe A.7.

Performances

L'algorithme remplis sa fonction d'alléger la charge calculatoire. En effet, on obtient une réduction notable du temps nécessaire pour l'entraînement.

De plus, il n'y à aucune variation notable de la qualité de l'entraînement.

Les performances de l'algorithme sont disponible dans le rapport dans l'annexe A.19.

Remise en cause de l'intérêt de l'architecture

Justement, comme seule une échelle apprend, on pourrait s'attendre à une baisse des performances. En effet, en entraînant une seule couche on réduit le nombre de paramètre du modèle, ce qui influence de façon néfaste la performance (Voir sous-section 6.5.5 pour l'impact du nombre de paramètres).

Comme le modèle muni d'une seule échelle apprend aussi bien que le modèle multi-échelles, cela remet en question l'utilité de l'architecture GMSNN et de ses échelles multiples.

6.5.7 Conclusion

Cette partie du projet GMSNN, dédiée à l'optimisation, à permis d'améliorer notablement les performances du modèle, tout en réduisant drastiquement le coût d'entraînement.

De plus, l'algorithme présenté sous-section 6.5.6 à démontré une faiblesse majeure de l'architecture GMSNN.

On peut aussi noter la mise-à-jour majeure de PyTorch, qui en plus de résoudre certains dysfonctionnements à nécessité le remaniement d'une partie de la base de code.

6.6 Production des exemples et découverte du problème d'encodage

Une fois le modèle fonctionnel, une partie importante de la compréhension et de l'évaluation du modèle est la production d'exemple.

Nous avons retardé cette étape principalement à cause des problèmes de mémoire.

Le principe de cette étape est d'utiliser notre modèle de la langue pour produire du langage, afin d'avoir une idée plus concrète qu'un score de la performance du modèle.

6.6.1 Exemples

Voici quelques exemples produits par le modèle. La version du modèle choisie est celle avec le meilleur score, parmi celle enregistrées.

Cette version atteignait un score BPC de 1.839 après un entraînement de 465 époques. Pour comparaison, le modèle état de l'art avait un score BPC de 1.255 en 50 époques.

Pour rappel, les données sont issues d'une version filtrée de Wikipédia en anglais.

```
YeoMMDF|Ph#elementat[[Damous]]
    thatureoftenusevoirbeexpounderstatesandanumberofhisworkformembersothan
    novelwasmethecebylementorfromthelastPreenancenoldWarInstartedbythe
    Philosophy''theTayita(
    amsmethouspeopleamingshelebelobesinthesatietheuniversalistscientis
    educationof[[Lakingforts]].
```

Fragment de code 6.1 – Exemple 1 : une suite de caractères à priori incompréhensibles.

```
+EDrFuergCases,areinlesssuchasthesthealterplains.
*In[[Stefapes]]
*[[AcademyAwards===

ANASA)asLASCIIRunder,andas.
    MatthebusipenclearsandpresidenthaveaquelfuelsifthesearchfromAwarerLievol
    ofany30020.

It''[[Anim]]
*[[UnitedStates|raphicsiteDirection]]

Theplant-gainheditsreturnedtoaseethewarinsteast&quot;Oneofthe[
    ectlywouldnotbytheIntegrationscapianland](ora''[[schology]]
|published:
```

Fragment de code 6.2 – Exemple 2 : des termes balisé comme dans le corpus d'origine, les crochets ouverts sont refermés.

```
60447-toNewHarry}}
Thenmainst.Rand'sintereststhe&quot;toinpassingtheEarth''s(''[[par]].

:'''maimals,anackreloquedoutwidthofgrawwithluteframedapproyingtoundernverby[[
    hebesination]]of&lt;/smalkan,
    instablishedacondorttodevelopedframesbeforestatedwinkingaroundinrational
    hicarefartoredonaftercanbeagainsthatgroupswouldnear,
    notwhatwasthatisstillastructionCenter,toDagnythat

On[[ëÃ£teleofAirej]]
[[cy:Alaska]]
[[no:Arni-Anchorages]]
```

Fragment de code 6.3 – Exemple 3 : des termes balisé comme dans l'exemple 2, et une autre suite de carractères.

6.6.2 Manque d'espaces

Parmi ces exemples, on remarque immédiatement le manque d'espace.

La source de ce phénomène n'est autre qu'un problème dans le corpus source. La version prétraitée de ce corpus ne contenait aucun espace, donc le modèle à appris une langue dans laquelle l'espace n'existe pas.

C'est un problème majeur, qui à probablement eu un impact élevé sur les performances du modèle. En effet, en anglais comme dans beaucoup de langues occidentales, l'espace est un élément fondamental dans la structuration du langage écrit. Le modèle à ainsi apprit une langue moins structurée, donc plus difficile à apprendre, que l'anglais.

6.6.3 Quelques éléments qui ressortent

Cependant, si on regarde plus en détail les exemples produits, des structures apparaissent.

Si on prend *[[UnitedStates | raphicsiteDirection]] de l'exemple 2 (Fragment de code 6.2, ligne 8) ou [[cy:Alaska]] et [[no:Arni–Anchorages]] de l'exemple 2 (Fragment de code 6.2, lignes 7 et 8), on a des doubles crochets, qui sont correctement ouverts puis fermés. On remarque aussi la présence de séparateurs (: et |). De plus, ces structure sont similaire aux annotations présentes dans le code Wikipédia. On peut les voir dans l'Fragment de code 4.1, page 21.

Enfin, malgré l'absence d'espaces, on discerne de nombreux mots :

- la suite de mots statesandanumberofhisworkformembersothannovelwasmethe qui ne contient en fait que des mots bien formés en anglais : states and a number of his work for member so than novel was me the (Fragment de code 6.1, ligne 1) ;
- de même pour la suite de mots oldWarInstartedbythePhilosophy : old War In started by the Philosophy (Fragment de code 6.1, ligne 1) ;
- on trouve aussi des noms propres comme le nom de pays : UnitedStates (Fragment de code 6.2, ligne 8) et Alaska (Fragment de code 6.3, ligne 7).

6.7 Analyse des résultats et arrêt du projet

6.7.1 Analyse des résultats

Avec notre maître de stage, nous avons étudié attentivement les résultats des dernières optimisations sur les performances du modèle (voir annexe A.19). Le résultat le plus dérangeant était l'absence d'apprentissage des couche supérieures.

À partir des connaissances de la littérature possédées par notre maître de stage et de ce résultat, nous avons conclu que 90% de l'information nécessaire est apprise par la première échelle du modèle. Les autres échelles ne font qu'améliorer ce résultat, et sont peu utiles tant que la première échelle n'est pas complètement entraînée.

À cela s'ajoute la disparition des espaces du corpus, qui nécessiterait non seulement de remanier une partie du code tenue pour acquise, mais aussi de refaire la plupart des tests effectués.

6.7.2 Conclusions de l'analyse

La conclusion à laquelle nous sommes arrivés est qu'il aurait fallu recommencer le développement avec un système de gestion des données maîtrisé et changer le processus de développement du modèle.

La première étape aurait été de développer un modèle simple, avec une seule échelle, et de le pousser au maximum de ses capacités. Seulement à ce moment là nous aurions pu l'augmenter d'autres échelles.

Il aurait aussi été intéressant de revoir l'architecture pour utiliser un modèle sans récurrences.

Cette conclusion impliquait de recommencer le projet, ou à défaut de le remanier en grande partie.

6.7.3 Arrêt du projet et début du projet suivant

Au moment de cette analyse, notre maître de stage revenait d'une réunion décisive sur le projet PAPUD.

Celle-ci avait permis de définir les objectifs du projet ITEA3-PAPUD, cas d'utilisation BULL (voir chapitre 9), qui ont été influencés par nos conclusions.

La nécessité de recommencer le projet GMSNN, couplée à l'opportunité de mettre les conclusions et les compétences acquises en pratique dans un projet à grande échelle, nous ont mené à interrompre projet GMSNN pour consacrer la fin du stage au projet PAPUD.

C'est d'un commun accord avec notre maître de stage que nous avons décidé de basculer sur le projet PAPUD.

7 Conclusions sur le projet GMSNN

7.1 Retour sur le travail effectué

Ce projet nous à permit d'implémenter une architecture innovante de réseau de neurones artificiels, à partir du squelette d'un modèle état de l'art. Nous avons pus élaborer un prototype suivant les concepts clés de l'architecture proposée, avant de l'améliorer et de l'optimiser.

Pour cela nous avons étudié un domaine technique dans lequel nous avions peu de connaissances ; nous avons manipulé une librairie qui nous était inconnue ; nous avons géré des tests durant de plusieurs heures à plusieurs jours sur des machines distantes ; nous avons, enfin, affronté un des obstacles les plus importants dans le développement de réseau de neurones artificiels, le problème de l'optimisation.

Bien que le l'architecture GMSNN n'ai pas atteint les performances espérées, le modèle produit est robuste, rapide, et peu volumineux. De plus, l'algorithme présenté sous-section 6.5.6 à démontré une faiblesse majeure de l'architecture GMSNN. Enfin, les problèmes rencontrés dans ce projet ont permit de tirer des conclusions très utiles pour de prochains projets :

- les RNN sont très lents à entraîner ;
- la maîtrise du pré-traitement est fondamentale pour obtenir des bons résultats ;
- pour utiliser une architecture multi-échelle comme celle proposée, il vaut mieux entraîner un modèle simple en premier lieu.

En conclusion, le projet à abouti sur le rejet de l'architecture proposée. Néanmoins, ce résultat à permit de cerner les principaux écueils de la réalisation d'un modèle de la langue multi-échelle, et à ainsi permit un meilleur déroulement du projet suivant.

7.2 Apport personnel du projet

La réalisation de ce projet nous à permit d'approfondir largement nos connaissances en apprentissage profond, et de nous habituer aux problématique de la création et de l'utilisation de réseaux de neurones.

7.3 Discussion et perspectives

7.3.1 Pertinence des choix et possibilités d'exploration

Il est important de relever que dans beaucoup des situations rencontrées, nous avons dû faire des choix. De même dans l'ordre de priorité des optimisations à effectuer. Il est normal de douter de la pertinence de ces choix, d'autant plus que notre niveau d'expertise du domaine est faible.

Si à posteriori nous sommes capables d'envisages d'autres pistes pour poursuivre ce projet, en aucun cas nous ne regrettons les choix effectués, en particulier la décision d'abandonner le projet.

Par exemple, nous aurions pu optimiser le taux d'apprentissage, comme fait dans le projet PAPUD (voir section 12.7).

Parmi les très nombreuse possibilités envisagées, on trouve aussi :
- l'utilisation d'un RNN pour interpréter les informations des différentes échelles ;
- la poursuite de l'utilisation de l'algorithme couche par couche, en poussant chaque échelle au maximum de ses capacités avant d'intégrer de nouvelles échelle ;
- le changement de l'architecture de pyramidale à parallèle, c'est à dire que chaque échelle serait indépendante, similairement à l'article [25].

7.3.2 Travaux similaires

Dans un article datant du 27 juillet 2018 [26], soit quelques jours avant la fin du stage, une architecture extrêmement similaire à celle du GMSNN est développée.

Contrairement a notre stage, le modèle de l'article montre des performances supérieures à celles des autres architectures auxquelles il est comparé.

En écho à la partie précédente, cela peut être dû à des choix de développement différent, comme à un travail plus poussé et plus expert sur le sujet.