Stage de recherche

Réseaux de Neurones Multi-Échelles pour la Modélisation de la Langue à partir de Grands Volumes de Données

Esteban Marquer

Année 2017–2018

Projet réalisé pour l'équipe SYNALP du laboratoire LORIA

Maître de stage : Christophe Cerisara

Parrain universitaire : Jeanine Souquières

Stage de recherche

Réseaux de Neurones Multi-Échelles pour la Modélisation de la Langue à partir de Grands Volumes de Données

Esteban Marquer

Année 2017–2018

Projet réalisé pour l'équipe SYNALP du laboratoire LORIA

Esteban Marquer
marquer.esteban@etu.univ-lorraine.fr

Institut des Sciences du Digital Management & Cognition
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 72 74 16 18
idmc-contact@univ-lorraine.fr

LORIA
Campus scientifique
BP 239
54506, Vandoeuvre-lès-Nancy Cedex
+33 (0)3 83 59 20 00

Encadrant : Christophe Cerisara

Remerciements

Je tiens à remercier M. Christophe Cerisara, qui a élaboré un sujet passionnant pour ce stage, et qui a su m'accompagner tout au long de cette aventure, malgré un emploi du temps chargé et des responsabilités nombreuses en tant que chef d'équipe.

Je remercie Mme. Nadia Bellalem, Mme. Christine Fay-Varnier et M. Samuel Cruz-Lara avec qui j'ai brièvement collaboré au sein du projet PAPUD.

Je remercie aussi que Mme. Jeanine Souquières, qui m'a conseillé lors de l'élaboration de ce rapport.

Je tiens tout particulièrement à remercier M. Maxime Amblard, sans qui je n'aurai pas obtenu ce stage.

Mais je remercie aussi tous les stagiaires et doctorants qui ont supporté mon humour pendant plus de 3 mois, ainsi que les stagiaires de l'IDMC qui le supportent depuis bien plus longtemps.

Enfin, je remercie Annick Jacquot, qui s'est chargé de toutes les questions administratives de mon stage ; Caroline et toute l'équipe de la cafétéria, qui m'ont offert un cadre inoubliable ; et tous les gens du LORIA qui m'ont offert un accueil chaleureux et qui font vivre ce laboratoire.

Une mention spéciale pour mon PC agonisant, qui m'a supporté tout au long du stage et de la rédaction de ce rapport.

Avant-propos

La lecture du présent mémoire ne nécessite aucune connaissance préalable en traitement automatique de la langue, en apprentissage automatique ou en apprentissage profond.

Il est cependant préférable d'avoir des connaissances en informatique pour comprendre les enjeux du stage.

Par ailleurs, de nombreux termes techniques sont utilisés.

Bien que habituellement utilisés sous leur forme anglaise dans la littérature du domaine, dans ce rapport tous les termes qui possèdent une traduction française ont été traduits. Les seules exceptions sont certains sigles, qui sont utilisés tels quels dans les textes français.

C'est pourquoi tous les termes techniques seront toujours introduits en français, accompagnées d'une explication, de la traduction anglaise et du sigle anglais si nécessaire.

D'autre part, une partie du rapport est dédiée à l'explication des termes et concepts utilisés, et un glossaire est présent en fin d'ouvrage.

Ce rapport a été réalisé avec LaTeX et les diagrammes présentés ont été produits par nos soins avec l'outil Ti*k*Z.

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 plus simplement 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 mise en œuvre desdites théories.

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

D'autre part, nous somme à l'ère du « *Big Data* », et les quantités de données produites de nos jours sont bien au-delà de ce que nous pouvons gérer sans outils spécialisés. Afin de produire des outils adaptés aux échelles actuelles, nous nous intéressons dans ce rapport à des grands volumes de données 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 données. 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 pragmatiques d'optimisation liées au volume de données.

1.2 Objectifs du stage

Deux objectifs se sont succédé durant le stage.

L'objectif initial du stage était d'explorer une idée d'architecture innovante de réseau de neurones artificiels, imaginée par Mr. Cerisara, le maître de stage. Il s'agit du projet GMSNN (réseau de neurones récurrents multi-échelles croissant, *Growing Multi-Scale Recurrent Neural Network* en anglais, voir chapitre 3, page 21).

Cependant, à l'issue du deuxième mois du stage, notre objectifs a changé.

Le nouvel objectif a été 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 doit servir de base technique pour une partie du projet PAPUD (*Profiling and Analysis Platform Using Deep Learning*, voir chapitre 9, page 53).

1.3 Terminologie et concepts fondamentaux

Cette section est dédiée 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 n'est donné au long du texte.

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

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.

On parle d'entrées pour désigner les données fournies au modèle, et de sorties pour désigner les données produites par ce modèle.

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 données. Voici quelques termes courants se référant aux données d'entraînement :
- le corpus : l'ensemble des données 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 consister à retirer les données erronées, à en adapter le format, à les anonymiser, ou encore à 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 d'apprentissage automatique consacrées aux réseaux de neurones.

Faisant partie des méthodes d'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

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 considérons les réseaux de neurones comme des modules, c'est-à-dire des boites noires, sans nous intéresser à leur conception interne.

Nous parlons 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 utilisons 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 réseaux de neurones

La liste suivante présente les principales architectures de réseaux de neurones utilisées dans ce rapport, classées en ordre croissant de complexité et de consommation de ressources.

— Le réseau de neurones artificiels intégralement connecté ou module linéaire est un des types de réseaux de neurones les plus simple, qui est emblématique des réseaux de neurones.
— Le réseau de neurones récurrents (*Recurrent Neural Network* en anglais, RNN) est une architecture de réseaux de neurones particulièrement adaptée au traitement de séquences. Le RNN 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 éléments suivants.
— Le réseau récurent à mémoire à court et long terme (*Long Short Term Memory* en anglais, LSTM) est un RNN particulier équipé d'une mémoire supplémentaire, plus puissant mais plus coûteux à entraîner qu'un RNN classique.

1.3.3 Traitement automatique des langues (TAL) et modèle de la langue

Traitement automatique des langues (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 (*Language Model* en anglais, LM) est une « distribution de probabilité sur une séquence de mots [ou de caractères] » utilisée 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 mot suivant à partir des mots précédents (ou le caractère suivant à partir des caractères précédents).

Contexte et dépendances

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

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

Parmi les information contenues dans le contexte d'un mot, on peut trouver aussi bien le sens des mots environnants que la structure syntaxique de la phrase, ou encore des informations plus générales comme le nom du dernier roi de France.

On considère que, plus on a d'informations contextuelles, plus le modèle de la langue est précis. Par exemple, si on nous dit « un chat », il sera plus difficile de 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 composantes :
— 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'époques, durée, etc.) ;
— la consommation de ressources nécessaire pour atteindre cette qualité (mémoire, puissance de calcul, …).

Afin d'évaluer la performance, des mesures ont été définies :
— pour mesurer la qualité du résultat, on utilise l'écart entre le résultat produit et le résultat attendu ; une mesure définie dans la littérature et utilisée dans les annexes est le BPC [1] ;
— pour mesurer le temps d'entraînement, on le nombre d'époque et le temps nécessaire par époque, la durée totale en heures, etc. ;
— pour mesurer la consommation de ressources, on évalue l'espace mémoire occupée (en MiB ou GiB), la puissance de calcul utilisée (pourcentage de la puissance disponible), etc.

Un modèle optimal serait un modèle qui atteint d'excellents résultats (l'écart entre ce qui est attendu et ce qui est produit le plus faible possible), 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 second temps nous allons développer les deux parties du stage, c'est-à-dire le projet GMSNN et le projet PAPUD.

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

Dans un dernier temps, nous ferons un bilan de l'ensemble du travail réalisé pur en tirer les apports principaux.

1. Le BPC (*Bits Per Character* en anglais) [50] est originalement une mesure de la qualité de compression de texte, mais plusieurs papiers ont détourné cette mesure en s'en servant d'estimation de la qualité d'un modèle de la langue au niveau du caractère. Dans cet usage, le BPC est assimilable à une mesure de la précision des résultats du modèle de la langue. Plus le BPC est proche de 0 plus la qualité du modèle de la langue est élevé.

Première partie

Projet GMSNN

Réseau de neurones récurrents multi-échelles croissant (*Growing Multi-Scale Recurrent Neural Network* en anglais, abrégé en GMSNN)

2 Présentation du laboratoire et de l'équipe

2.1 Généralités

C'est dans le Laboratoire Lorrain d'Informatique et ses Applications (LORIA) que le stage s'est déroulé, au sein de l'équipe SYNALP (*SYmbolic and statistical NAtural Language Processing*) dirigée par M. Christophe Cerisara, le maître de stage.

2.2 Le LORIA

Le 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). » [3]. Depuis sa création en 1997, le LORIA se concentre sur les sciences informatiques, que ce soit par la recherche fondamentale ou appliquée.

2.2.1 Structure administrative du LORIA

Le LORIA est dirigé par quatre instances [4] :

- **l'équipe de direction** : elle est composée du directeur, d'un directeur adjoint, de la responsable administrative, et de l'assistante de direction ; ils assistent le directeur dans la pris de décision et leur mise en œuvre ;
- **le conseil scientifique** : il est composé du directeur du laboratoire, des deux directeurs adjoints et des scientifiques responsables des cinq départements du laboratoire ; ils assistent le directeur dans la pris de décision et leur mise en œuvre ;
- **le conseil de laboratoire** : il est 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)** : elle est composée des scientifiques responsables des 28 équipes du laboratoire ; ils se réunissent tous les deux mois.

2.2.2 Recherche au sein du LORIA

Le LORIA est l'établissement qui héberge l'équipe SYNALP, une des 28 équipes de recherche du laboratoire. Le laboratoire est composé de 5 départements, chacun orienté vers un domaine d'étude particulier.

La structure générale du LORIA en départements et, plus en détail, du Département 4 est représentée sur l'organigramme de la figure 2.1 (page 18). 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 internet du laboratoire [5].

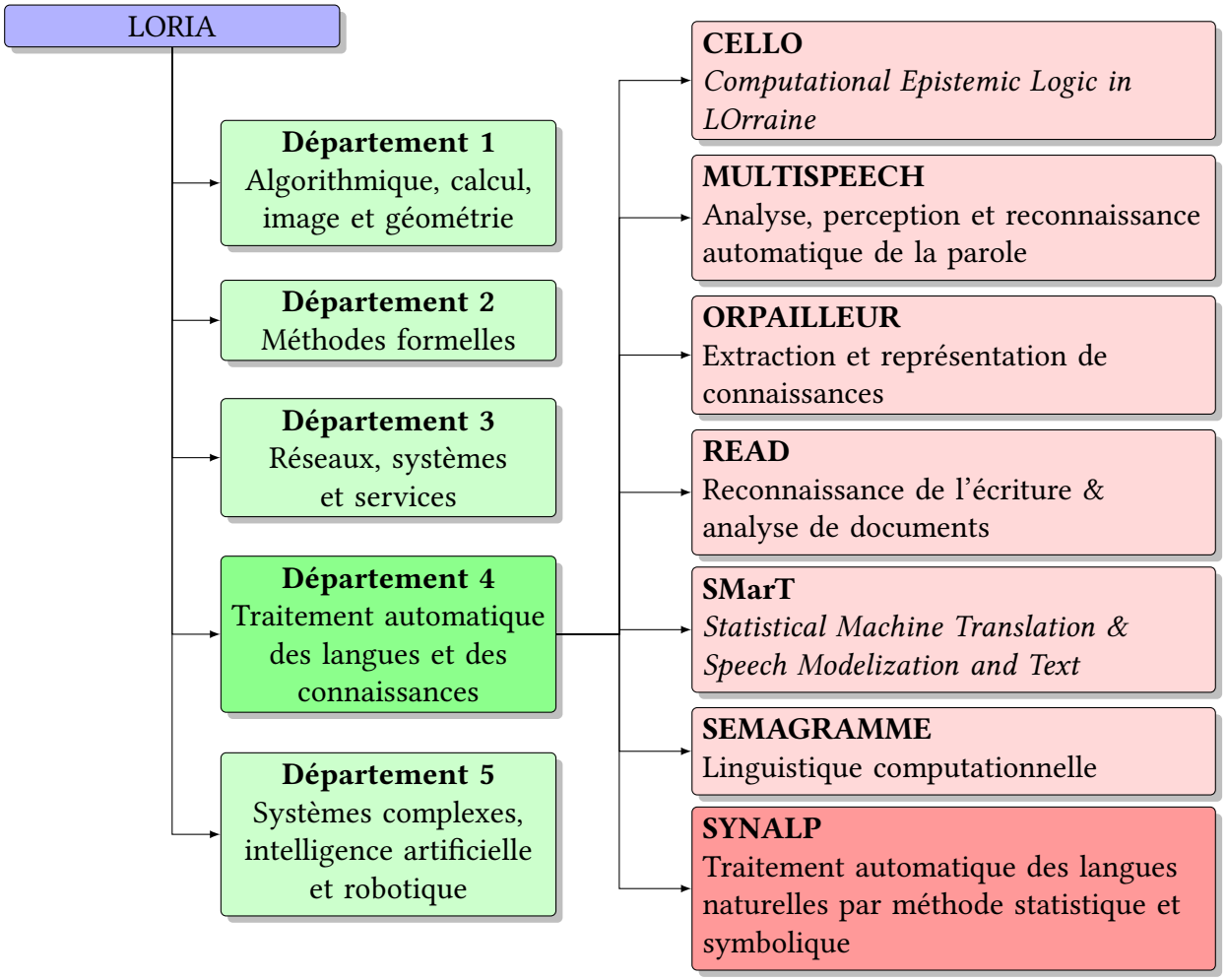


FIGURE 2.1 – Organigramme des départements du LORIA, et des équipes du Département 4

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. 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 dirigée par M. Christophe Cerisara, et comporte actuellement 12 membres permanents, une dizaine de doctorants et d'ingénieurs, et 5 stagiaires à l'heure de la rédaction de ce rapport.

2.3.2 Thématiques de recherche

La recherche au sein de l'équipe 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 modèles de la langue, 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 modèles de la langue, et l'élaboration d'outils et ressources utilisés en TAL. Nous verrons en détail pourquoi dans le chapitre 3 (page 21).

2.4 Pour en savoir plus

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

3 Architecture innovante de réseaux 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 gèrent difficilement les fautes et les mots nouveaux.

Ces performances sont obtenues, entre autres, grâce à une gestion du contexte des exemples.

3.1.1 Manque d'utilisation des gros volumes de données

Cependant, ces modèles sont souvent développés et entraînés avec peu de données. Les raisons envisageables sont principalement le manque de données brutes ou préparées, et le peu d'amélioration de performance malgré des coûts supplémentaires importants.

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 de RNN peunent se rappeler d'informations vielles d'une centaine d'entrées ; et un LSTM dépasse difficilement les 200 entrées.

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

De nombreuses tentatives ont été faite pour 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 le sujet de stage vise à la fois à tirer partie des grands volumes de données, et à permettre au modèle d'établir des dépendances de haut niveau, voire des connaissance contextuelles externes (c'est-à-dire à étendre le contexte au-delà des informations directement accessibles, à inférer des vérités générales).

Cette architecture, nommée GMSNN (voir chapitre 5, page 27), a donné son nom au projet qui l'utilises.

3.3 Projet GMSNN

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

La mise en œuvre devait se réaliser à partir d'une base de code sur laquelle le maître de stage avait commencé à travailler (plus de détails sont disponibles dans la sous-section 6.2.1, page 31).

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

Cette tâche a été nourrie par un travail individuel.

Un fonctionnement en rapports de suivis (disponibles dans l'annexe A, page 87), complémentés d'une occasionnelle correspondance électronique, a permis de tenir le maître de stage informé de l'avancement du travail.

À cela s'ajoutent des réunions hebdomadaires pour faire le point sur les résultats obtenus et déterminer les priorités de travail.

3.4.1 Organisation initiale du travail

Dès la 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 (page 23) représente cette répartition prévue du travail. Une case correspond à une semaine de travail.

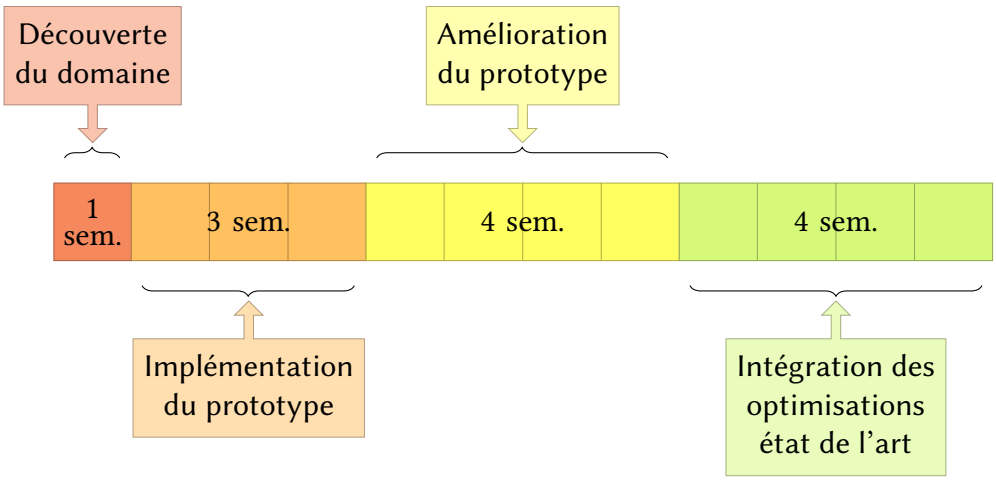


FIGURE 3.1 – Répartition prévue du travail

3.4.2 Déroulement réel du projet

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

Cependant, comme décrit section 6.7 (page 45), nous avons décidé d'interrompre ce projet pour nous consacrer au projet PAPUD. Une case de la figure correspond à une semaine de travail.

La figure 3.2 (Figure 3.2) représente la répartition réalisée du travail.

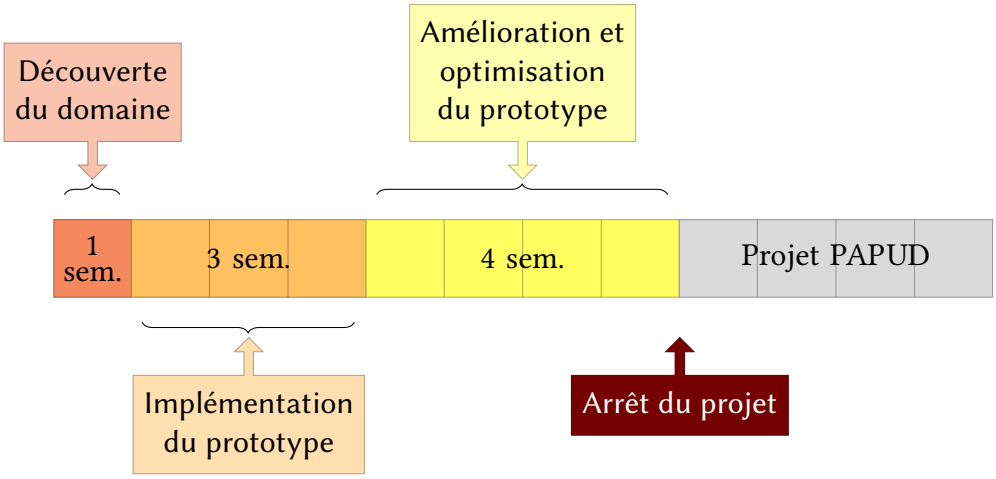


FIGURE 3.2 – Répartition réalisée du travail

3.5 Outils

3.5.1 Gestion du code

Le code et les rapports ont été gérés avec le système de gestion de version Git, et ont été stockés sur les serveurs Gitlab de l'INRIA.

3.5.2 Langage et librairie

Le langage choisi pour la mise en œuvre du projet est Python, qui est abondement fourni en outils et librairies d'apprentissage profond.

Parmi ces librairies, notre choix s'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 donnée 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 a 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. » [10]

Un des avantage de cet outil est la présence de machines spécialisées équipées de GPU [1], qui sont celles que nous avons utilisées.

1. Un processeur graphique (*Graphical Processing Unit* en anglais, GPU) est un composant d'ordinateur spécialisé, qui montre d'excellentes performances dans les calculs impliquant des matrices (ex. : images). Cette propriété s'applique aussi sur les tenseurs. L'utilisation de GPU pour l'entraînement des réseaux de neurones 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 du corpus ont été utilisées :

1. la première est composée des 10 000 000 premiers caractères de « enwik8 » ; cette version a 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 a servi pour le débogage du modèle.

4.1 Extrait des données d'entraînent

Le Fragment de code 4.1 (page 25) est un extrait des données brutes avant le découpage en caractères. Il correspond à l'article Wikipédia sur l'ansarchisme.

```
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

1. XML (*eXtensible Markup Language* en anglais), « est un langage informatique qui sert à enregistrer des données textuelles. [...] Ce langage , [...] 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 (page 44), un défaut dans le prétraitement a 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 prétraitées, l'intégralité des espaces était supprimé, y compris ceux du texte d'origine.

Malheureusement, ce défaut a été détecté à la fin du projet, et n'a pas pu être corrigé à temps pour entraîner le modèle sur une version propre du corpus.

5 Description de l'architecture proposée

5.1 Propriétés du modèle

Pour rappel, l'architecture proposée a 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.

Comme annoncé dans la section 3.2 (page 22), 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 modèles de la langue, on peut considérer les données sous forme de séquence.

Dans le cas présent, 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 a été conçue à partir de RNN.

5.1.2 Passer à l'échelle

Comme décrit dans la sous-section 3.1.2 (page 21), les RNN 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, identifiés par leur échelle dans ce rapport. Chacune de ces couches est un RNN.

Chaque échelle supplémentaire permet de modéliser des dépendances au sein de champs plus larges.

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 28). Sur cette figure, la fréquence de transmission est notée n.

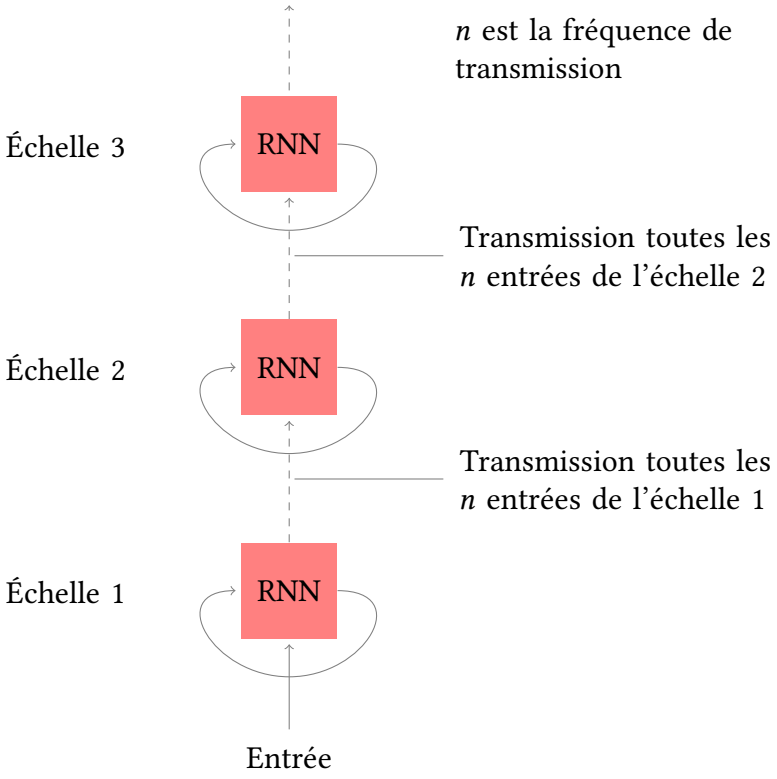


FIGURE 5.1 – Principe de transmission de l'information d'une échelle à la suivante

Admettons que la capacité de mémoire d'un RNN soit de 9 entrées (nombre arbitrairement défini pour l'exemple). Si l'échelle supérieure récupère des informations toutes les 3 entrées de la couche inférieure, sa 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. Nous parlerons 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. En effet, l'information stockée dans la mémoire d'un RNN peut d'apparenter à une représentation, donc à une abstraction, des données.

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 29). Sur cette figure, chaque bloc vert correspond à une entrée pour l'échelle correspondante ; les blocs bleus en bas du diagramme correspond aux caractères qui sont fourni en entrée au modèle. Ici la fréquence de transmission vaut 3.

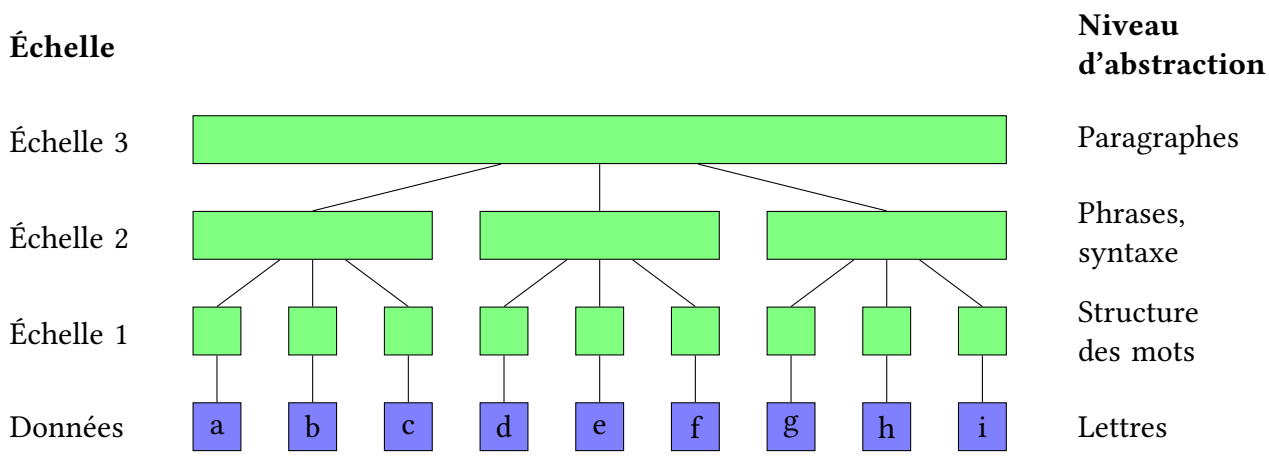


FIGURE 5.2 – Différentes échelles, et les niveaux d'abstraction attendus

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.

En effet, comme expliqué dans la sous-section 5.1.2 (page 27), le nombre d'échelles est dépendant du nombre d'entrées totales.

On peut considérer que, tant qu'aucune entrées 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 croît.

Une formule permet de déterminer le nombre de couches « existantes » n en fonction du nombre d'entrées présentées i et de la fréquence de transmission f :

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

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 36), l'utilisation de cette propriété a été abandonnée.

6 Réalisation

6.1 Recherche documentaire

La première partie du projet a été la recherche documentaire et la prise en main des outils. Ce travail a été effectué à partir des documents fournis par le maître de stage, de la documentation de PyTorch [14, 15, 16, 17] et de Grid5000 [18, 19], complétés par des 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 a été la prise en main de la base de code fournie. Elle contient une implémentation état de l'art [1] d'un modèle de la langue au niveau du caractère, sur laquelle le maître de stage avait commencé à travailler. Le code d'origine provient du dépôt « awd-lstm-lm »[20], qui contenait un d'un modèle de la langue au niveau du caractère état de l'art.

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 et 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 comportait des fragments obsolètes et peu documentés. Après le déchiffrage, il a fallu comprendre et corriger les fragments défectueux.

1. Par état de l'art nous entendons l'état actuel des connaissances et méthodes du domaine, ainsi que les logiciels et les résultats obtenus par ces méthodes.

6.2.2 Modèle ré-implémenté simplifiée

Le modèle simplifiée prend en entrée des caractères, et produit des probabilités sur quel caractère apparaîtra ensuite. Le modèle est composé d'un module encodant les caractères, d'un RNN particulier (un LSTM) et d'un module produisant une distribution de probabilités sur les caractères connus. La figure 6.1, (page 32) représente cette architecture.

Le module d'encodage des caractères, appelé *embedding 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 qu'un 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 [2]. Il transforme les informations produites par le réseau de neurones en probabilité qu'un des caractères soit le prochain caractère de la séquence.

Les rapports sur le modèle ré-implémenté sont disponibles aux annexes A.8 (page 106), A.9 (page 107) et A.10 (page 109).

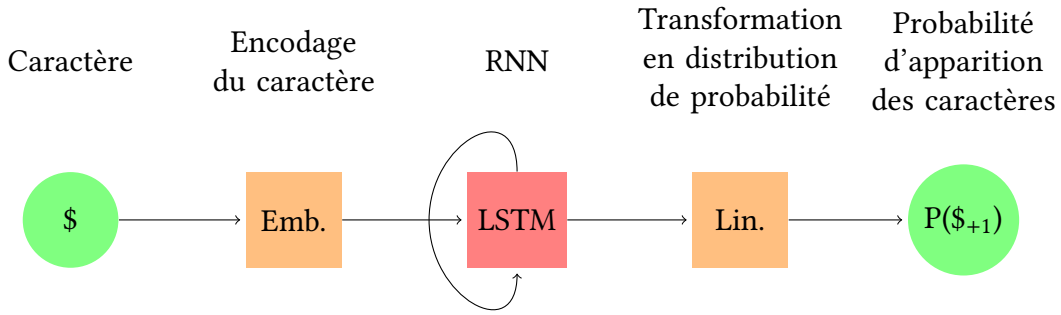


Figure 6.1 – Architecture du modèle ré-implémenté

2. « 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 a 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 celle du modèle ré-implémenté, mais le RNN y est remplacé par le module GMSNN (voir figure 6.2, page 33). C'est sur ce nouveau module GMSNN que le reste du travail au cours du projet GMSNN a été effectué.

De la même façon qu'avec la version ré-implémentée, le modèle prend en entrée des caractères, et produit des probabilités sur quel caractère apparaîtra ensuite.

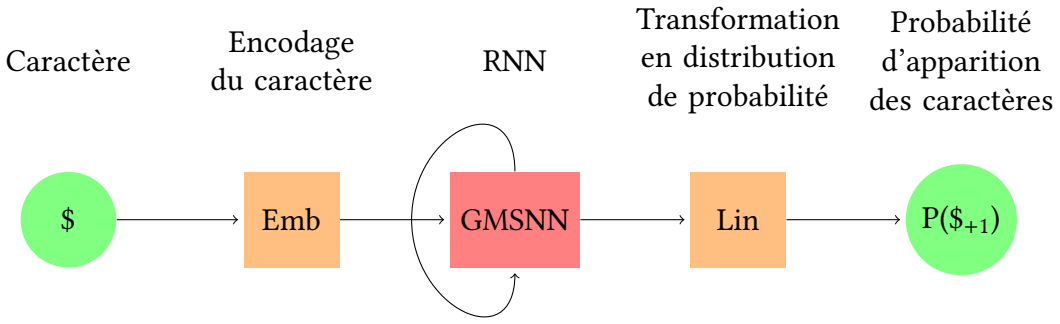


FIGURE 6.2 – Architecture du modèle ré-implémenté

Ce prototype a 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 mécanismes 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 donnée à la couche immédiatement supérieure. Cette transmission se fait périodiquement, en fonction d'un nombre appelé fréquence de transmission.

Dans un premier temps, il a fallu choisir quelle information transmettre d'une échelle à l'échelle supérieure. En effet, les RNN produisent à la fois une sortie et un tenseur contenant leur mémoire. L'utilisation de l'*embedding* a été écartée 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 donc les informations abstraites par l'échelle, contrairement à la sortie qui contient uniquement les informations pour prédire le caractère suivant.

L’annexe A.11 (page 113) présente 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 a 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 pu fonctionner 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 a 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 données* 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 données ; 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 a 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, page 37).

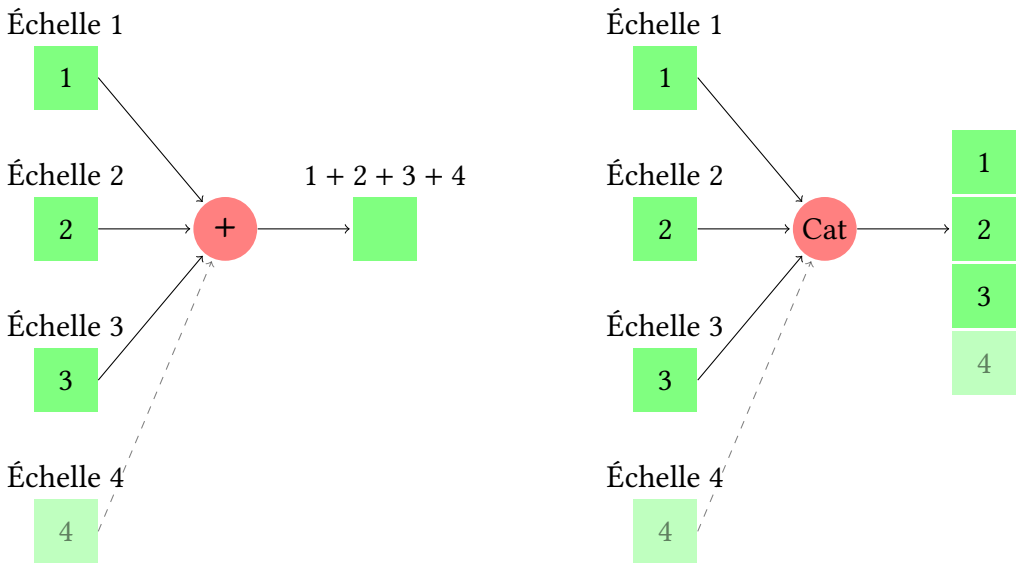
Après discussion avec le maître de stage, la stratégie d'agrégation à été changé en une concaténation des sorties.

Comme montré dans la figure 6.3b (page 37), les sorties sont mises côte-à-côte afin de former un nouveau tenseur et 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 a nécessité l'abandon de la propriété de croissance à l'infini de l'architecture (décrite sous-section 5.1.3, page 29), 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, page 29).

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. La figure 6.4 (page 37) représente ces résultats. Le temps de calcul alloué à l'entraînement des deux modèles 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 un BPC de 3.5, alors qu'on obtient un BPC de 4 avec l'addition.

Plus de détail sur le choix de la stratégie d'agrégation sont disponibles dans l'annexe A.12 (page 117).



(a) Stratégie d'agrégation additive

(b) Stratégie d'agrégation par concaténation

FIGURE 6.3 – Stratégies d'agrégation

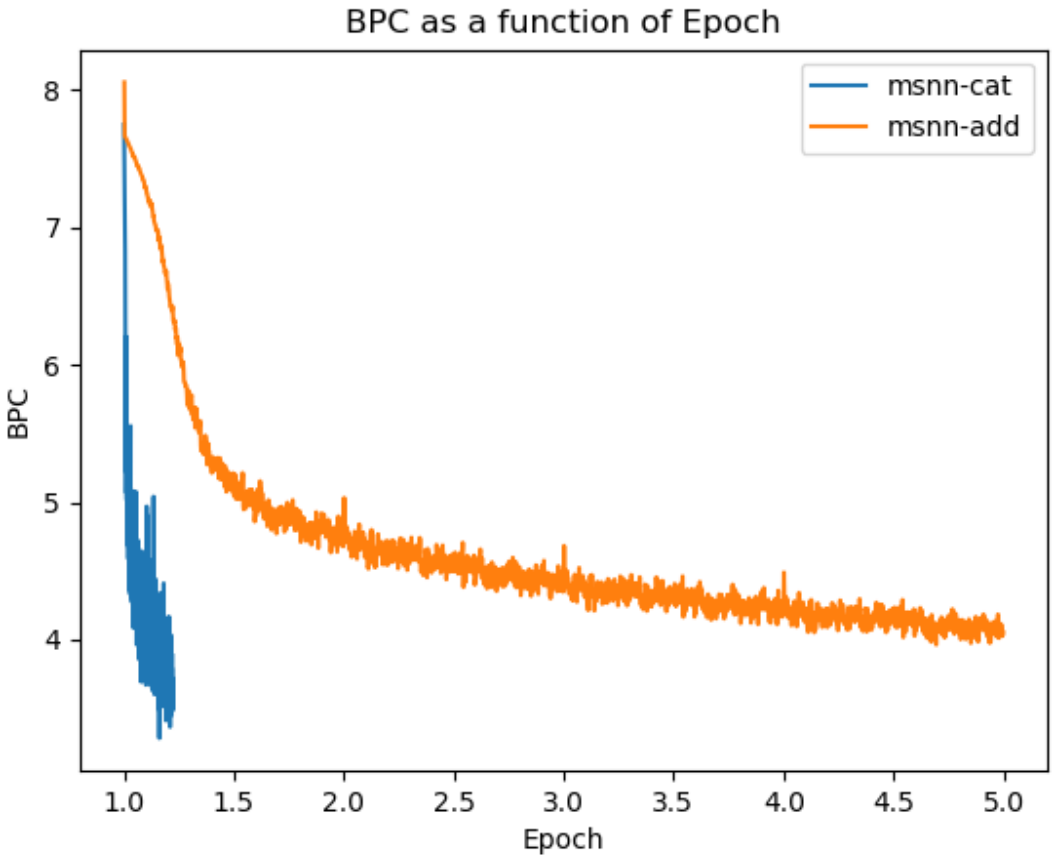


FIGURE 6.4 – Performances comparées des stratégies par concaténation *(msnn-cat)* et additive *(msnn-add)*

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 a été le malfonctionnement initial des outils fournis par PyTorch. Cela a 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.

Plus de détail sur le système de sauvegarde sont disponibles dans le rapport de l'annexe A.3 (page 93).

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 39) ;
- l'augmentation du nombre de paramètres du modèle (voir sous-section 6.5.5, page 40) ;

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 a causé le plantage [3] 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 a fourni une amélioration notable mais insuffisante. Il remplaçait le LSTM de chaque couche (voir sous-section 6.3.1, page 33) par un RNN basique, moins gourmand. Cela permettait aussi d'éliminer une redondance entre l'architecture LSTM et l'architecture GMSNN.

3. 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 (page 88), a 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 a 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 a 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 (page 97) contient le début de la résolution du problème.

Conclusion

Le problème de la fuite mémoire a 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 a pu reprendre.

On peut noter l'ampleur de l'optimisation par rapport à la version initiale :
- le temps de d'entraînement a 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 jour 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 a permit d'établir une méthode pour résoudre ou écarter la majorités des aspect du problème. Après consultation avec le maître de stage, nous avons décidé d'utiliser l'entraînement par *batchs* malgré les problèmes restants.

Le rapport de l'annexe A.5 (page 99) contient les détails de l'analyse des problèmes théoriques de l'utilisation de *batchs* avec l'architecture GMSNN.

L'annexe A.2 (page 88) contient les détails sur l'impact de la taille des *batchs* et du nombre de *batchs* sur la consommation mémoire.

Les annexes A.13 (page 122), A.16 (page 138), A.17 (page 142) et A.18 (page 146) contiennent 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, page 38), 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 RNN 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 IM *(Information-Maximizing)*.

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 (page 104).

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 a aucune variation notable de la qualité de l'entraînement.

Les performances de l'algorithme sont disponible dans le rapport dans l'annexe A.19 (page 104).

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, page 40).

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, a 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 (page 41) a 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 a 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 a appris une langue dans laquelle l'espace n'existe pas.

C'est un problème majeur, qui a 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 a 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, page 43, ligne 8) ou [[cy:Alaska]] et [[no:Arni–Anchorages]] de l'exemple 2 (Fragment de code 6.2, page 43, 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 25.

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, page 43, ligne 1) ;
- de même pour la suite de mots oldWarInstartedbythePhilosophy : old War In started by the Philosophy (Fragment de code 6.1, page 43, ligne 1) ;
- on trouve aussi des noms propres comme le nom de pays : UnitedStates (Fragment de code 6.2, page 43, ligne 8) et Alaska (Fragment de code 6.3, page 44, ligne 7).

6.7 Analyse des résultats et arrêt du projet

6.7.1 Analyse des résultats

Avec le maître de stage, nous avons étudié attentivement les résultats des dernières optimisations sur les performances du modèle (voir l'annexe A.19, page 151). Le résultat le plus dérangeant était l'absence d'apprentissage des couche supérieures.

À partir des connaissances de la littérature possédées par le 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, le 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, page 53), 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 a interrompre projet GMSNN pour consacrer la fin du stage au projet PAPUD.

C'est d'un commun accord avec le 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 a 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 (page 41) a 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 prétraitement 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 a abouti sur le rejet de l'architecture proposée. Néanmoins, ce résultat a permit de cerner les principaux écueils de la réalisation d'un modèle de la langue multi-échelle, et a ainsi permit un meilleur déroulement du projet suivant.

7.2 Apport personnel du projet

La réalisation de ce projet nous a 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, page 63).

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.

Deuxième partie

Projet PAPUD

Profiling and Analysis Platform Using Deep Learning

8 Présentation d'ITEA, du projet PAPUD, et de BULL

La seconde partie du stage s'intègre dans le projet PAPUD, en particulier dans le cas d'utilisation BULL. Ce projet fait partie de l'initiative ITEA (*Information Technology for European Advancement*) du réseau EUREKA. Nous verrons en détail les objectifs du projet dans la chapitre 9 (page 53).

8.1 Collaborateurs

Les personnes avec lesquelles nous avons collaboré durant ce projet sont M. Christophe Cerisara le maître de stage, ainsi que deux autres chercheurs de l'équipe SYNALP, Mme. Nadia Bellalem et M. Samuel Cruz-Lara. Une quatrième chercheuse, Mme. Christine Fay-Varnier, nous a rejoint vers la fin du stage.

8.2 EUREKA et ITEA3

« EUREKA est une initiative européenne, intergouvernementale, destinée à renforcer la compétitivité de l'industrie européenne. » D'après Wikipédia [27].

ITEA3 est la troisième itération d'un programme du réseau EUREKA nommé ITEA (*Information Technology for European Advancement*). ITEA est un programme de recherche, développement et innovation basé sur un partenariat public-privé, et fonctionnant par appels de projet. Ces appels à projets se concentrent sur des problématiques des technologies de l'information et de la communication, et ce dans une perspective industrielle.

ITEA3 implique plus de 40 pays, ainsi que de nombreuses entreprises.

8.3 Projet PAPUD et cas d'utilisation BULL

C'est lors de la troisième vague d'appels à projets d'ITEA3 que le projet PAPUD a été accepté.

L'objectif du projet PAPUD *(Profiling and Analysis Platform Using Deep Learning)* est l'élaboration d'une série d'outils basés sur les techniques de l'apprentissage profond. La plateforme ainsi produite a pour objectif l'analyse des volumes de données devenus trop grands pour être gérés de façon traditionnelle. Ainsi, le projet PAPUD *(Profiling and Analysis Platform Using Deep Learning)* s'inscrit dans la dynamique d'ITEA3, tout comme dans la thématique de notre stage.

Nom complet du projet	16037 PAPUD
Période de réalisation	Janvier 2018 - Décembre 2020 (3 ans)
Appel à projet	ITEA 3 Call 3
Partenaires	16
Coûts estimés	10 927 000 €
Volume de travail estimé (en personne.année)	151,88
Pays participants	Belgique, Espagne, France, Roumanie, Turquie

Table 8.1 – Informations générales sur le projet PAPUD, d'après le site de ITEA3 [28]

8.4 BULL

Présentation de l'entreprise

BULL est une entreprise française spécialisée dans la sécurité informatique et la gestion des gros volumes de données informatiques.

L'entreprise a été rachetée en 2014 par le groupe ATOS.

Secteurs d'activité

D'après le site d'ATOS [29], les activités principales de la filiale BULL sont :
- le matériel informatique et logiciel professionnel de haute sécurité ;
- le matériel informatique et logiciel pour l'Armée et la Défense, y compris du matériel de navigation maritime et terrestre ;
- les serveurs de calcul et de stockage, les *data-centers* (infrastructures spécialisées regroupant de nombreux serveurs) et les solutions nuagiques (*cloud*) ;
- les solutions de calcul haute performance (les « supercalculateurs ») ;
- les systèmes intégrés, à savoir du matériel informatique spécifique intégré à un produit, comme par exemple l'ordinateur de bord intégré dans une voiture.

Globalement, BULL concentre ses activités sur le matériel informatique et les logiciels de pointe en matière de sécurité et de fiabilité. Les gammes de produits BULL s'adressent principalement à des grosses entreprises et aux états.

Pour en savoir plus

Des informations plus détaillées sur le projet PAPUD sont disponible sur la page web du projet [28].

9 Réseau de neurones artificiels pour la prédiction de pannes

9.1 Contexte

Nous avons vu que parmi les secteurs d'activité de BULL, les serveurs et autres systèmes de traitent de gros volumes de données sont très présents.

Ces outils tombent rarement en panne, mais quand ils le font cela occasionne des pertes très importantes pour l'entreprise.

Il serait donc très intéressant de mettre au point un système de prédiction des pannes, afin de pouvoir les éviter.

Les données disponibles pour remplir cette tache sont des fichiers de journaux systèmes (décrits en détail dans le chapitre 10, page 57) de très grande taille. Ils contiennent de nombreuses informations sur les évènements se déroulant dans les outils.

9.2 Solution

D'après les documents de travail officiels (en particulier le fichier README.md du dépôt de code officiel du projet).

Le cas d'utilisation BULL du projet PAPUD est dédié à répondre à cette problématique, en fournissant un système détectant les signes de pannes dans les journaux systèmes.

Pour cela, il a été décidé de modéliser le comportement normal (sans panne) de ces journaux.

Ils sont composés de lignes de textes en anglais. Il est donc possible de produire un modèle de la langue capable de prédire la prochaine ligne. Par la suite, le modèle sera augmenté d'un système prenant en compte le contexte de la ligne à prédire pour améliorer sa précision. Par contexte on entend ici des dépendances avec des lignes plus anciennes que la ligne précédente.

Ainsi, le plan général des opérations est divisé en 2 parties :

1. on suppose que la structure en dépendances entre les lignes est simplissime : une ligne dépend uniquement de la ligne précédente ; on cherche donc à établir un modèle de la langue capable de modéliser au mieux cette dépendance ;
2. une fois le modèle simple établi, on abandonne le postulat précédent, et on cherche à établir à partir du modèle créé un modèle capable de modéliser des dépendances à la fois plus complexes et sur plus d'une ligne au par avant.

Pour ce qui est du modèle simple, il a été décidé de ne pas utiliser de RNN, bien trop lent pour la quantité de données à traiter. À la place, un réseau de neurones artificiels basique sera utilisé.

On peut noter que les conclusions du projet GMSNN ont été appliquées, autant pour le déroulement du projet que pour le type de modèle à utiliser.

9.3 Projet PAPUD

La tache qui nous a été confiée est la réalisation du modèle simple.

Plus exactement, étant donné qu'il était évident que la durée restante du stage serait insuffisante pour réaliser et pousser au maximum le modèle simple, nos objectifs étaient la réalisation d'un prototype du modèle, et de mettre en place les outils nécessaires à l'entraînement. Ceux-ci sont principalement les outils de gestion et de prétraitement des données, les outils d'évaluation des performances du modèle, et l'algorithme d'entraînement du modèle.

La description des caractéristiques du modèle est disponible dans le chapitre 11 (page 59).

Par la suite, nous désignerons ce projet par « projet PAPUD ».

9.4 Organisation du travail

Contrairement au projet GMSNN, d'autres collaborateurs participaient a ce projet (voir section 8.1, page 50). Étant, avec le maître de stage, les seuls parmi les collaborateurs habitués à manipuler des réseaux de neurones, nous avons travaillé individuellement durant ce projet.

Le projet GMSNN s'étant bien déroulé, une organisation similaire a été mise en place afin de tenir ces autres membres du projet informés de l'avancement et des conclusions de notre travail. C'est-à-dire que des rapports fréquents et des réunions hebdomadaires durant lesquelles nous pressentions notre progression ont été mis en place.

Les rapports de ce projet sont également disponibles en annexe (voir l'annexe B, page 159). Le rapport d'une des réunion est disponible à l'annexe B.6 (page 172).

9.4.1 Organisation initiale du travail

Le projet PAPUD s'est déroulé sur la base de cycles de développement. C'est durant les réunions hebdomadaires que les prochains objectifs étaient décidés.

En effet, contrairement au projet GMSNN pour lequel il a été simple de définir des périodes réservées aux grandes étapes du projet, ce projet PAPUD s'est déroulé dans un temps très restreint.

Cependant, il a été possible de définir les priorités suivantes :

1. la réalisation d'un prototype fonctionnel ;
2. la mise en place d'un algorithme d'entraînement basique ;
3. la mise en place de moyens d'évaluer le modèle et l'obtention de premières performances ;
4. le prétraitement des données et la préparation de la gestion des très gros volumes de données à venir ;
5. le temps restant est dédié à l'amélioration des performances.

Une extension de la durée du stage de 1 semaine a été décidée, de façon à augmenter le temps dédié au travail sur le projet. Cela c'est fait en considérant les disponibilités du maître de stage, des autres collaborateurs ainsi que les nôtres.

La durée totale du projet a donc été de 5 semaines.

9.4.2 Déroulement réel du projet

Tous les objectifs nécessaires ont été remplis, et deux améliorations notables des performances ont été mises en place.

La répartition réelle du travail du projet est représentée dans la figure 9.1 (page 55). Une case de la figure correspond à un jour de travail.

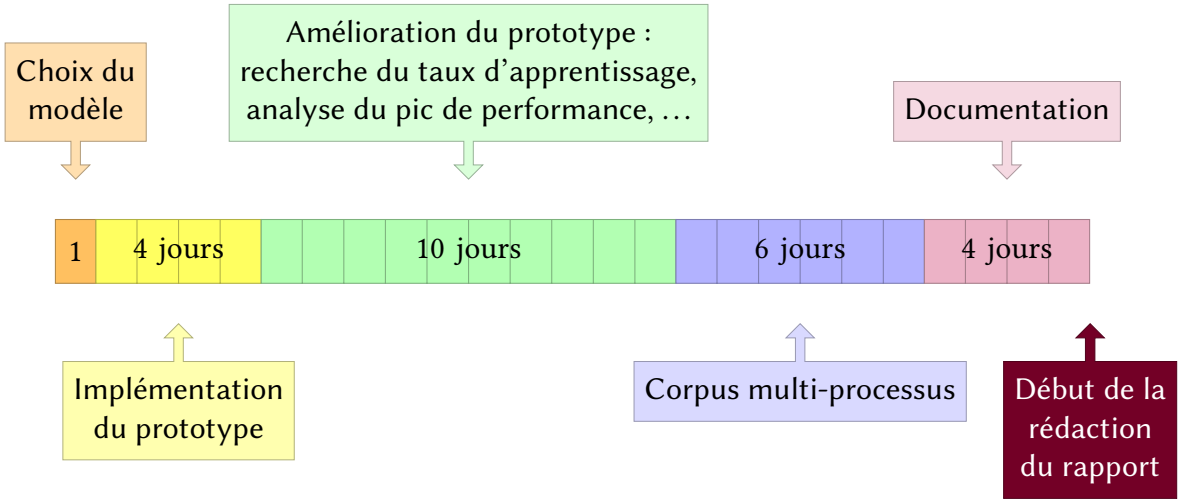


FIGURE 9.1 – Répartition du travail

9.5 Outils

9.5.1 Gestion du code

Le code et les rapports ont été gérés avec le système de gestion de version Git, et ont été stockés sur les serveurs Gitlab de l'INRIA. Les collaborateurs du projet ont accès à l'ensemble de ces données, qui ont été mises à jours tout au long du projet.

9.5.2 Python, PyTorch et Grid5000

Nous avons décidé de continuer à travailler avec Python, PyTorch et Grid5000, pour trois principales raisons :

- la première partie du projet s'est déroulée avec ces outils, nous étions donc habitués à leur utilisation et à leurs subtilités ;
- la base de code accumulée jusqu'à présent reposait sur ces outils, et pouvait être réutilisée sans trop d'efforts ;
- il était possible de basculer plus tard vers une autre librairie que PyTorch, cette dernière possédant une fonctionnalité permettant la conversion d'un modèle vers les autres principales librairies disponibles pour Python et quelques autres langages.

10 Données disponibles

Les données disponibles sont des des fichiers de journaux système.

Ce sont des lignes de texte en anglais extrêmement structuré (comme on peut le voir sur le Fragment de code 10.1, page 57), qui donnent des informations sur les programmes en cours d'exécution sur l'outil, et les évènements qui de déroulent. Par exemple, des messages d'information, comme « le programme A a tenté de faire B », ou « l'utilisateur C changé son mot de passe ».

Ces messages sont précédées d'informations comme le moment d'écriture du message et le programme d'ou il provient.

De plus, ces journaux système contiennent un nombre très grand de messages, et la quantité de données disponible pour le projet dépasse les 400 GiB de texte brut. En comparaison, les données utilisées pour le projet précédent pesaient au maximum 100 MiB, soit 4 000 fois moins.

Un échantillon de 9 MiB des données disponibles a été utilisé pendant le projet. Cela correspond à 70 131 lignes de journaux système.

10.1 Extrait des données d'entraînent

Les données utilisées sont confidentielles. Ainsi, l'extrait présenté ici a été largement modifié. Entre autre, la date et le *timestamp* (nombre correspondant à la date) ont été remplacées par le date de début du stage et le *timestamp* correspondant, et l'utilisateur a été renommé « OOOOOO ».

Dans le Fragment de code 10.1 (page 57), on a de gauche à droite : le *timestamp*, la date, l'heure, l'utilisateur (OOOOOO), le processus (authpriv), le type de message (info), et le message.

```
1    1524463200 2018 Apr 23 08:00:00 OOOOOO authpriv info access granted for
    user root (uid=0)
```

Fragment de code 10.1 – Exemple d'une ligne extraite des journaux systèmes

10.2 Prétraitement des données

Le prétraitement des données est composé de :
— la suppression du *timestamp*, de la date, de l'heure, et de l'utilisateur ;
— le découpage du texte restant à une certaine longueur ;
— le remplissage des ligne n'atteignant pas cette longueur par un caractère spécifique nommé caractère de remplissage ;
— la transformation de tous les caractères en nombres, à l'aide d'un dictionnaire ; les caractères apparaissant peu fréquemment ou n'apparaissant pas dans le dictionnaire sont tous remplacé par un caractère spécial nommé « caractère inconnu ».

10.2.1 Traitement des codes hexadécimaux

Par la suite, le remplacement de tout code hexadécimal comme 0x005f par un caractère spécifique du dictionnaire a été ajouté (voir section 12.6, page 63). De cette façon, 0x005f et 0x0007 sont remplacés par le caractère <hexX4>, tandis que 0000005f et 89abcdef sont remplacés par <hex8>. Cela permet de s'abstraire de la valeur du code sans perdre l'information liée à sa présence.

11 Modèle à réaliser

Le modèle à réaliser est conçu pour être le plus rapide possible.

Le modèle est un modèle de la langue au niveau du caractère, qui prend en entrée une ligne et qui prédit la ligne suivante.

Ainsi, l'entrée du modèle est une ligne de taille fixe de nombres correspondant à des caractères.

La sortie est un tableau à 2 dimensions, qui contient pour chaque caractère prédit une distribution de probailité sur les caractères connus (répertoriés dans le dictionnaire décrit section 10.2, page 58).

11.1 Utilisation d'un réseau de neurones artificiels basique

Comme présenté dans le chapitre 9 (page 53), un réseau de neurones artificiels basique, intégralement connecté, a été choisi pour le modèle.

Étant le réseau de neurones artificiels le plus simple, il est extrêmement rapide à entraîner.

11.2 Réduction de la taille de l'entrée

Comme écrit plus haut, l'entrée du modèle est une ligne de caractères. Pour chacun d'entre eux, comme dans le modèle GMSNN, est transformé en un tenseur par un module d'*embedding* (voir la sous-section 6.2.2, page 32).

Grâce à une opération appelée « *max-pooling* », qui correspond à prendre pour chaque case du tenseur final la valeur maximale des cases correspondantes des tenseurs initiaux.

Par exemple :

$$\text{max-pool}\left(\begin{bmatrix} 0 & 30 & 6 \\ -5 & 6 & 12 \end{bmatrix}, \begin{bmatrix} 2 & 42 & 3 \\ 5 & -60 & -2 \end{bmatrix}\right) = \begin{bmatrix} 2 & 42 & 6 \\ 5 & 6 & 12 \end{bmatrix}$$

Car $0 < 2$, $30 < 42$, $6 > 3$, $-5 < 5$, $6 > -60$ et $12 > -2$.

11.3 Représentation de l'architecture

La façon dont s'agencent les différents modules de l'architecture est représentée dans la Figure 11.1 (page 60).

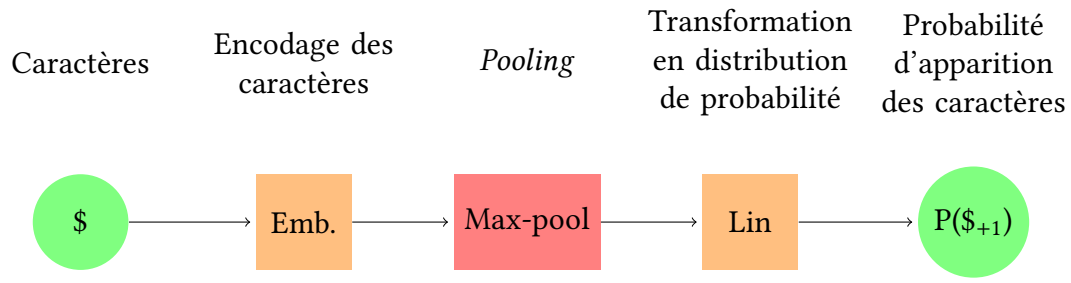


FIGURE 11.1 – Architecture du modèle du projet PAPUD

11.4 Lien avec l'état de l'art

Il a été récemment montré que les modèle de la langue simples basés sur des *embeddings* et des opération de *pooling* montre des performances équivalentes voir supérieures à des modèles plus complexes et plus coûteux comme les RNN [30].

12 Réalisation

12.1 Définition du modèle

La toute première étape du projet a été de définir l'architecture à utiliser (décrite chapitre 11, page 59). C'est sous la forme de réunions que cette étape s'est déroulée.

Tout d'abord, nous avons élaboré le modèle avec le maître de stage. Par la suite, nous avons présenté nos conclusions aux autres membres du projet, qui les ont approuvées.

12.2 Implémentation du modèle et transfert des outils de base

Une fois le modèle défini, l'étape suivante a été l'implémentation d'une version de base, ainsi que des outils nécessaire pour les utiliser.

Ces outils sont :
- un premier outil de prétraitement de manipulation du corpus ;
- un algorithme d'entraînement sur ces données, récupéré du projet précédent ;
- un outil d'analyse et de traçage des performances du modèle, qui est une version améliorée, plus fluide et plus puissante, des outils de visualisation du projet précédent (voir section 6.4, page 35) ;
- un système de sauvegarde et d'intéruption de l'entraînement similaire à celui du projet précédent (voir sous-section 6.5.2, page 38)

12.2.1 Premiers résultats

Les premiers résultats du modèle sont encourageants, autant sur la rapidité du modèle que sur ses performance. Ils sont disponibles dans l'annexe B.3 (page 162).

12.3 Adaptation du système de gestion de corpus au volume de données

L'étape suivant l'implémentation du modèle a été l'adaptation du système de gestion de corpus aux grands volumes de données à traiter dans le futur.

Pour cela, un premier prototype a été réalisé. Il utilisait des mécanismes optimisés du langage Python pour la lecture de fichier. L'objectif de ce prototype est de maintenir uniquement une faible quantité de données en mémoire.

Le principe de fonctionnement est simple :

1. on charge en mémoire une portion des données ;
2. on applique le prétraitement sur ces données ;
3. on transfert les données traitées à l'algorithme d'entraînement ;
4. on recommence de l'étape 1 avec la portion suivante des données.

L'intégration de cet outil à l'algorithme d'entraînement s'est faite en parallèle de la suite du projet.

12.4 Entraînement par *batchs* simultanés

La première optimisation du modèle a été l'intégration de l'algorithme d'entraînement par *batchs* (voir sous-section 6.5.4, page 39).

Il a été nécessaire de définir la nombre de *batchs* optimal en terme de temps de calcul et de performance. Les détails concernant ce choix sont disponibles dans l'annexe B.4 (page 164).

Cette optimisation a permis d'accélérer considérablement l'entraînement du modèle.

12.5 Changement d'unité de mesure pour évaluer la qualité du modèle

Cette partie du projet n'est pas dédie à une optimisation, mais à la rectification d'une unité de mesure inadaptée.

Jusqu'à cette étape, les performance étaient évaluées à partir du score utilisé pour mettre à jour le modèle. Ce score est difficile à utiliser pour se faire une idée réelle des performance du modèle.

Il a donc été remplacé par une mesure nommée « précision », qui correspond au taux de caractères correctement prédits (le bon caractère au bon endroit).

12.5.1 Précision de base

Ce changement d'unité de mesure a été l'occasion de déterminer ce que l'on appelle la précision de base (*baseline accuracy* en anglais).

Cette valeur représente la précision d'un modèle qui répondrait toujours la même chose. Elle sert de base de comparaison pour les performances du modèle. Cela permet de déterminer si le modèle produit apprend réellement modèle l'information, et dans quelle mesure.

Le détail de ces résultats est disponible dans les rapports des annexes B.2 (page 160) et B.6 (page 172).

12.6 Analyse d'une étrange variation de performance dans l'apprentissage

La seconde optimisation apportée au modèle était dédiée à réduire les sources d'erreur dans les données.

Dans les courbes d'apprentissage du modèle basique, d'étranges baisses de performances sont visible, toujours sur la même portion des données.

Il a été décidé de consacrer du temps à l'analyse de ce phénomène.

L'étude du phénomène a révélé une forte présence de codes hexadécimaux (comme `0x005f` ou `4A6D005F`) dans le fragment des données mis en cause, codes qui semblaient être la cause de la baisse de performance.

Il a donc été décidé, comme décrit sous-section 10.2.1 (page 58), d'intégrer au prétraitement des données la gestion de ces codes. Cette intégration a cependant dù attendre la fin de la réalisation de la nouvelle version du gestionnaire de données, décrite section 12.8 (page 64).

Le déroulement de l'analyse et les conclusions détaillées sont disponibles dans l'annexe B.5 (page 168).

12.7 Optimisation du taux d'apprentissage

La troisième optimisation mise en place pour le modèle et le réglage d'un des paramètres de l'algorithme d'apprentissage, est une valeur nommée « taux d'apprentissage ». Il s'agit d'une optimisation courante en apprentissage profond [31].

Ce paramètre permet de déterminer la vitesse d'apprentissage du modèle. Cependant, un mauvais réglage entraîne des conséquences catastrophiques sur le modèle, comme un apprentissage extrêmement lent, voir une divergence de l'apprentissage.

Il est donc nécessaire de correctement choisir ce paramètre. Les algorithmes que nous avons étudiés reposent sur des séries de brefs entraînements : on entraîne le modèle avec différentes valeurs pour le taux d'apprentissage, et on conserve celle qui a donné le meilleur résultat pour les entraînements à venir.

Un module permettant la détermination du taux d'apprentissage idéal a donc été implémenté, et utilisé.

La nouvelle valeur définie a permis d'augmenter largement la vitesse d'amélioration de la qualité du modèle. En effet, il fallait plus de 400 époques pour atteindre la qualité maximale du modèle, et la nouvelle valeur a permis de l'atteindre en moins de 100 époques.

Le détail du processus de choix et des résultats est disponible dans les annexes B.7 (page 174) et B.8 (page 178).

12.8 Mise en place d'un système de gestion de corpus plus puissant

La dernière optimisation mise en place est la transformation du système de corpus en une version plus puissante. Le nouvel outil créé a été intégralement doté de tests unitaires et testé.

12.8.1 Fichiers multiples

L'étude des données disponibles a révélé une structure en nombreux fichiers successifs. Ainsi, à la demande du maître de stage, nous avons donné au nouvel outil la capacité à utiliser plusieurs fichiers comme source de données continue. Cette fonctionnalité a été implémentée de façon optimisée en temps de calcul et en utilisation mémoire.

12.8.2 Prétraitement modulaire

D'autre part, durant le déroulement du projet nous avons remarqué que l'ajout ou la modification d'une étape de prétraitement était ardue avec l'ancien outil. En gardant cela à l'esprit, nous avons développé un système de prétraitement modulaire, dans lequel chaque étape du traitement est séparée et interchangeable. Cette architecture permet d'ajouter ou de retirer à volonté des étapes au traitement. C'est d'ailleurs lors de l'implémentation de ces modules que le prétraitement mentionnée section 12.6 (page 63) et sous-section 10.2.1 (page 58) a été intégré.

12.8.3 Processus multiples

Une autre amélioration, tirant profit de l'aspect modulaire du traitement, est l'utilisation de multiples processus en parallèle. Chacun d'entre eux est dédié à une étape du prétraitement. Cela permet de répartir la charge de travail sur les différents processus, à la façon d'un travail à la chaîne. Ainsi, l'outil est beaucoup plus rapide que la version précédente.

12.8.4 Performances

Le nouvel outil est largement plus rapide et efficace que son prédécesseur. De plus, il augmente la facilité d'utilisation et d'entretien. Il reste cependant plus lent que la version adaptée aux petits fichiers.

La description des performances et le détail du fonctionnement de l'outil sont disponibles dans le rapport de l'annexe B.10 (page 182).

13 Conclusions sur le projet PAPUD

13.1 Retour sur le travail effectué

Le projet PAPUD s'est déroulé sans accroc, et l'intégralité des objectifs ont été remplis.

Nous avons réalisé deux principales optimisations, outre le prétraitement des codes hexadécimaux et le choix du nombre de *batchs* :
- la réalisation d'un outil d'optimisation du taux d'apprentissage ;
- la réalisation d'un outil multi-fichiers multi-processus de gestion du corpus.

De plus, nous avons porté une attention particulière à la documentation du code. En effet, le travail effectué était le début de la réalisation du cas d'utilisation BULL du projet PAPUD. Il est donc normal que nous ayons laissé une base de code propre et intégralement documentée pour notre successeur sur le projet.

En conclusion, les objectifs du projet ont été remplis. Le prototype réalisé a permis de fournir des premiers résultats encourageants pour la suite du projet PAPUD. De plus de nombreux outils ont été réalisées, qui seront réutilisables pour la suite du projet.

13.2 Apport personnel du projet

La réalisation de ce projet nous a permit de mettre en application l'ensemble des connaissances accumulées durant le projet précédent.

Nous avons ainsi mettre nos compétences à bon usage dans un projet de grande ampleur.

De plus, le travail avec une équipe de projet nous a permit de découvrir certains aspects de la réalisation d'un projet de recherche, cela a été une expérience enrichissante.

Enfin, la documentation du code et des outils, en particulier les derniers jours du projet qui y étaient dédiés, nous a permit de nous perfectionner dans l'art de la documentation en Python. Par là nous entendons surtout les techniques de typage du code [32, 33] et les pratiques de rédaction de *docstring* (le format sous lequel est rédigé la documentation en Python).

13.3 Discussion et perspectives

13.3.1 Continuation du travail

Le projet s'est très bien déroulé, et de notre point de vue les résultats ont été satisfaisant.

Mais ce n'est que le début du projet, et il serait très intéressant de poursuivre le travail et de continuer à construire sur les bases que nous avons posé, forts de notre connaissance du projet et de notre expérience.

La réalisation d'un stage l'an prochain en serait l'opportunité idéale.

Parmi les pistes pour continuer le projet, on peut compter :
— l'amélioration du système de gestion de corpus ;
— l'étude de l'utilisation encore faible des ressources et l'optimisation de cet usage ;
— l'entraînement et le perfectionnement du modèle sur l'ensemble des données disponibles, jusqu'à atteindre les limites du modèle ;
— le début du travail sur la deuxième grande étape du projet (décrites section 9.2, page 53).

14 Bilan du stage

14.1 Bilan du travail effectué

Durant ce stage, nous avons travaillés sur deux projets différents, le projet GMSNN et le projet PAPUD. L'un était une initiation aux techniques et technologies du apprentissage profond, ainsi qu'une exploration du potentiel d'une idée ; l'autre était une mise en pratique des connaissances et compétences acquises.

Même si le premier projet a été interrompu, le principal objectif qui était de sonder le potentiel de l'architecture a été rempli dans une certaine mesure. De plus, le second projet qui est la continuation du premier, a été mené à bien.

Ainsi, les éléments développés durant la première partie du stage ont été essentiels au succès de la dernière.

Au final, les résultats et outils fourni pour le projet PAPUD sont les fruits de tous le travail effectué durant ce stage.

14.2 Bilan professionnel

Ces deux projets nécessitaient pour leur réalisation de bonnes compétences en programmation Python, une connaissance théorique des mécanismes et théories principales du apprentissage profond, ainsi qu'une certaine maîtrise de PyTorch et de Grid5000.

Avant ce stage, nous avions déjà l'habitude du Python et de ses subtilités ; nous avions aussi une vague connaissance des théories et des mécanismes utilisés en apprentissage profond, de par notre formation et la lecture de quelques articles.

Conscients de nos lacunes, nous avons dédié une période au début du stage à l'acquisition des connaissances nécessaires, que nous avons complétées tout au long du stage.

Nous avons aussi appris à maîtriser PyTorch et Grid5000, à gérer des entraînements de modèles d'apprentissage profond, ainsi qu'à manipuler un des modèles réputé parmi les plus complexes de l'apprentissage profond, le RNN.

L'essentiel du stage s'est déroulé en autonomie, avec la possibilité de consulter le maître de stage (malgré sa disponibilité limitée, car il est aussi le responsable de l'équipe SYNALP). Nous avons ainsi eu une grande liberté de mouvement dans notre travail, dont il nous semble nous avons su tirer parti.

Nous avons été menés à rédiger beaucoup de comptes rendus, ainsi que présenter à des profanes les résultats de nos travaux. Cela nous a permis de développer nos capacités d'analyse, de synthèse et de vulgarisation.

En conclusion, durant ce stage, nous avons acquis un panel de connaissances et de savoirs-faire liées à la conception et à l'utilisation de réseau de neurones artificiels, ainsi qu'une méthodologie expérimentale typique de l'apprentissage profond.

14.3 Bilan personnel

Ce stage était une aubaine pour nous, qui cherchions à la fois un stage permettant de découvrir le travail de recherche, et une occasion de découvrir les domaines de l'intelligence artificielle ou du TAL, afin de décider de la poursuite de nos études.

En effet, ce stage nous a permis d'avoir un aperçu du travail de chercheur. D'une part durant les premier mois, pendant lesquels nous avons exploré une problématique état de l'art. D'autre part en travaillant avec l'équipe du projet PAPUD, qui nous a montré un aspect plus appliqué du travail de recherche.

Nous tenons à souligner qu'il a été très agréable de voir nos avis considérés et nos idées bien reçues tout au long projet.

Enfin, ce stage a été l'occasion de laisser libre court à un de nos vices cachés : l'optimisation compulsive du code produit.

En définitive, ce stage a été une expérience extrêmement enrichissante, et ce fut un réel plaisir de manipuler le code, d'apprendre et de relever les différent défis qui se sont présentés, en compagnie du maître de stage, des collaborateurs du projet PAPUD et de nos camarades stagiaires et doctorants.

15 Bibliographie / Webographie

[1] *Apprentissage automatique*. Wikipédia. Juin 2018. URL : https://fr.wikipedia.org/wiki/Apprentissage_automatique (visité le 10/08/2018) (cf. p. 10).

[2] *Language model*. Anglais. Wikipédia. Juil. 2018. URL : https://en.wikipedia.org/wiki/Language_model (visité le 10/08/2018) (cf. p. 12).

[3] *Le Loria*. LORIA. Juil. 2018. URL : http://www.loria.fr (visité le 31/07/2018) (cf. p. 17, 19).

[4] *Organisation*. LORIA. Août 2018. URL : http://www.loria.fr/fr/presentation/organisation/ (visité le 03/08/2018) (cf. p. 17).

[5] *Organigramme 2017 du Loria*. LORIA. Août 2018. URL : http://www.loria.fr/wp-content/uploads/2016/05/organigramme-2017.pdf (visité le 03/08/2018) (cf. p. 18).

[6] *About SYNALP*. Anglais. SYNALP. Juil. 2018. URL : http://synalp.loria.fr/pages/about-synalp (visité le 30/07/2018) (cf. p. 19).

[7] W. Xiong et associés. « Achieving Human Parity in Conversational Speech Recognition ». Anglais. Dans : *CoRR* (oct. 2016). arXiv : 1610.05256 [cs.CL]. URL : https://arxiv.org/abs/1610.05256 (visité le 17/08/2018) (cf. p. 21).

[8] Rafal Józefowicz et associés. « Exploring the Limits of Language Modeling ». Anglais. Dans : *CoRR* (fév. 2016). arXiv : 1602.02410 [cs.CL]. URL : http://arxiv.org/abs/1602.02410 (visité le 17/08/2018) (cf. p. 21).

[9] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*. Anglais. Mai 2015. URL : http://karpathy.github.io/2015/05/21/rnn-effectiveness (visité le 23/04/2018) (cf. p. 21).

[10] *Grid5000*. Anglais. Grid5000. Avr. 2018. URL : https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home (visité le 25/04/2018) (cf. p. 24).

[11] Matt Mahoney. *About the Test Data*. Anglais. Wikipédia. Sept. 2011. URL : http://mattmahoney.net/dc/textdata.html (visité le 17/08/2018) (cf. p. 25).

[12] Marcus Hutter. *50'000€ Prize for Compressing Human Knowledge*. Anglais. Fév. 2018. URL : http://prize.hutter1.net (visité le 17/08/2018) (cf. p. 25).

[13] Tony Archambeau. *XML - Définition*. Wikipédia. Août 2018. URL : http://glossaire.infowebmaster.fr/xml (visité le 17/08/2018) (cf. p. 25).

[14] Soumith Chintala. *Deep Learning with PyTorch : A 60 Minute Blitz*. Anglais. PyTorch. Août 2018. URL : http://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html (visité le 12/08/2018) (cf. p. 31).

[15] Justin Johnson. *Learning PyTorch with Examples*. Anglais. PyTorch. Août 2018. URL : http://pytorch.org/tutorials/beginner/pytorch_with_examples.html (visité le 12/08/2018) (cf. p. 31).

[16] Sean Robertson. *Classifying Names with a Character-Level RNN*. Anglais. PyTorch. Août 2018. URL : http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html (visité le 12/08/2018) (cf. p. 31).

[17] *PyTorch 0.4 documentation*. Anglais. PyTorch. Août 2018. URL : https://pytorch.org/docs/stable/index.html (visité le 12/08/2018) (cf. p. 31).

[18] *User :Ibada/Tuto Deep Learning*. Anglais. Grid5000. Avr. 2018. URL : https://www.grid5000.fr/mediawiki/index.php/User:Ibada/Tuto_Deep_Learning (visité le 25/04/2018) (cf. p. 31).

[19] Christophe Cerisara. *How to install pytorch on our GPU cluster*. Anglais. Intranet DeepLoria. Mar. 2017. URL : http://deeploria.gforge.inria.fr/intranet/pytorch (visité le 25/04/2018) (cf. p. 31).

[20] Stephen Merity. *Smerity/awd-lstm-lm*. Anglais. Août 2018. URL : https://github.com/Smerity/awd-lstm-lm (visité le 12/08/2018) (cf. p. 31).

[21] *PaddlePaddle/VisualDL*. Anglais. Mai 2018. URL : https://github.com/PaddlePaddle/VisualDL (visité le 09/05/2018) (cf. p. 35).

[22] *VisualDL - Visualize your deep learning training and data flawlessly*. Anglais. Mai 2018. URL : http://visualdl.paddlepaddle.org (visité le 09/05/2018) (cf. p. 35).

[23] Jason Brownlee. « A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size ». Anglais. Dans : *Machine Learning Mastery* (juil. 2017). URL : https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size (visité le 13/04/2018) (cf. p. 39, 40).

[24] Viacheslav Khomenko et associés. *Accelerating recurrent neural network training using sequence bucketing and multi-GPU data parallelization*. 2017. arXiv : 1708.05604. URL : http://arxiv.org/abs/1708.05604 (cf. p. 40).

[25] Fen Xiao et associés. « MSDNN : Multi-Scale Deep Neural Network for Salient Object Detection ». Anglais. Dans : *CoRR* (jan. 2018). arXiv : 1801.04187 [cs.CV]. URL : https://arxiv.org/abs/1801.04187 (visité le 17/08/2018) (cf. p. 48).

[26] P. HuberetJ. Niehues et A. Waibel. « A Hierarchical Approach to Neural Context-Aware Modeling ». Anglais. Dans : *ArXiv e-prints* (juil. 2018). arXiv : 1807.11582 [cs.CL]. URL : https://arxiv.org/abs/1807.11582 (visité le 17/08/2018) (cf. p. 48).

[27] *EUREKA*. Wikipédia. Août 2018.
URL : https://fr.wikipedia.org/wiki/EUREKA (visité le 01/08/2018) (cf. p. 50, 81).

[28] *16037 PAPUD*. Anglais. itea3.org. Juil. 2018.
URL : https://itea3.org/project/papud.html (visité le 31/07/2018) (cf. p. 51, 52).

[29] *Produits*. Atos. Août 2018.
URL : https://atos.net/fr/produits (visité le 01/08/2018) (cf. p. 51).

[30] Dinghan SHEN et associés. « Baseline Needs More Love : On Simple Word-Embedding-Based Models and Associated Pooling Mechanisms ». Anglais.
Dans : *CoRR* (mai 2018). arXiv : 1805.09843 [cs.CL].
URL : http://arxiv.org/abs/1805.09843 (visité le 17/08/2018) (cf. p. 60).

[31] Pavel SURMENOK.
« Estimating an Optimal Learning Rate For a Deep Neural Network ». Anglais.
Dans : *Towards Data Science* (nov. 2017).
URL : https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0 (visité le 17/08/2018) (cf. p. 63).

[32] *PEP 483 – Type Hints*. Anglais. Août 2018.
URL : https://www.python.org/dev/peps/pep-0484/#using-none (visité le 17/08/2018) (cf. p. 65).

[33] *PEP 484 – Type Hints*. Anglais. Août 2018.
URL : https://www.python.org/dev/peps/pep-0484/#using-none (visité le 17/08/2018) (cf. p. 65).

[34] Stephen MERITYETNitish Shirish KESKAR et Richard SOCHER.
« Regularizing and Optimizing LSTM Language Models ». Anglais.
Dans : *arXiv preprint arXiv :1708.02182* (2017).

[35] Stephen MERITYETNitish Shirish KESKAR et Richard SOCHER.
« An Analysis of Neural Language Modeling at Multiple Scales ». Anglais.
Dans : *arXiv preprint arXiv :1803.08240* (2018).

[36] Ilya SUTSKEVER. « Training Recurrent Neural Networks ». Anglais.
Thèse de doct. Graduate Department of Computer Science, University of Toronto, 2013.
URL : https://machinelearningmastery.com/gentle-introduction-backpropagation-time (visité le 29/05/2018).

[37] Y. BENGIO et J. S. SENECAL. « Adaptive Importance Sampling to Accelerate Training of a Neural Probabilistic Language Model ». Anglais.
Dans : *IEEE Trans. Neural Networks* 19.4 (avr. 2008), p. 713–722. ISSN : 1045-9227.
DOI : 10.1109/TNN.2007.912312.

[38] *Apprentissage profond*. Wikipédia. Juil. 2018.
URL : https://fr.wikipedia.org/wiki/Apprentissage_profond (visité le 10/08/2018).

[39] *Markdown*. Wikipédia. Juil. 2018.
URL : https://fr.wikipedia.org/wiki/Markdown (visité le 10/08/2018) (cf. p. 78).

[40] *Réseau de neurones récurrents*. Wikipédia. Août 2018.
URL : https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_r%C3%A9currents (visité le 10/08/2018).

[41] *Réseau neuronal*. Futura. Août 2018. URL : https://www.futura-sciences.com/tech/definitions/informatique-reseau-neuronal-601 (visité le 10/08/2018).

[42] Jason Brownlee.
« What is the Difference Between Test and Validation Datasets ? » Anglais.
Dans : *Machine Learning Mastery* (juil. 2017).
URL : https://machinelearningmastery.com/difference-test-validation-datasets (visité le 11/08/2018).

[43] Joe Davison. « "Deep Learning est mort. Vive Differentiable Programming !" » Anglais.
Dans : *techburst* (juin 2018).
URL : https://techburst.io/deep-learning-est-mort-vive-differentiable-programming-5060d3c55074 (visité le 11/08/2018).

[44] Matt Mazur. *mattm/simple-neural-network*. Anglais. Mar. 2015.
URL : https://github.com/mattm/simple-neural-network (visité le 23/04/2018).

[45] Andrej Karpathy. *Minimal character-level language model with a Vanilla Recurrent Neural Network, in Python/numpy*. Anglais. Juil. 2015.
URL : https://gist.github.com/karpathy/d4dee566867f8291f086 (visité le 23/04/2018).

[46] Andrej Karpathy. *CS231n Convolutional Neural Networks for Visual Recognition*. Anglais.
URL : http://cs231n.github.io/transfer-learning (visité le 24/04/2018).

[47] Andrej Karpathy. *karpathy/char-rnn*. Anglais. Avr. 2016.
URL : https://github.com/karpathy/char-rnn (visité le 23/04/2018).

[48] Edwin Chen. *Introduction to Conditional Random Fields*. Anglais.
URL : http://blog.echen.me/2012/01/03/introduction-to-conditional-random-fields (visité le 24/04/2018).

[49] Christopher Olah. *Understanding LSTM Networks*. Août 2015. URL :
http://colah.github.io/posts/2015-08-Understanding-LSTMs (visité le 24/04/2018).

[50] *How to compute bits per character (BPC) ?* Anglais. Mai 2017.
URL : https://stats.stackexchange.com/questions/211858/how-to-compute-bits-per-character-bpc (visité le 15/05/2018) (cf. p. 13).

[51] *The Average Sentence Length*. Anglais. Juil. 2008.
URL : https://strainindex.wordpress.com/2008/07/28/the-average-sentence-length (visité le 02/05/2018).

[52] *Optimizing PyTorch training code*. Anglais. Mai 2018.
URL : https://www.sagivtech.com/2017/09/19/optimizing-pytorch-training-code (visité le 22/05/2018).

[53] *Tail Recursion Elimination*. Anglais. Mar. 2018.
URL : http://neopythonic.blogspot.fr/2009/04/tail-recursion-elimination.html (visité le 16/05/2018).

[54] Jason Brownlee. « A Gentle Introduction to Backpropagation Through Time ». Anglais.
Dans : *Machine Learning Mastery* (juin 2017).
URL : https://machinelearningmastery.com/gentle-introduction-backpropagation-time (visité le 06/07/2018) (cf. p. 102).

[55] Jason Brownlee. *Réseaux de Neurones*. Statistica. Juin 2017. URL : http://www.statsoft.fr/concepts-statistiques/reseaux-de-neurones-automatises/reseaux-de-neurones-automatises.htm (visité le 11/08/2018).

[56] *Les réseaux de neurones expliqués à ma fille*. OD-Datamining. Août 2018. URL : https://od-datamining.com/knwbase/les-reseaux-de-neurones-expliques-a-ma-fille (visité le 11/08/2018).

[57] *Mémoire, rapport de stage : les règles de présentation*. l'Étudiant. Août 2018. URL : https://www.letudiant.fr/jobsstages/nos-conseils/memoire-rapport-de-stage-les-regles-de-presentation.html (visité le 03/08/2018).

[58] *Cadre Théorique d'un Mémoire - Contenu et Exemple*. Scribbr. Fév. 2016. URL : https://www.scribbr.fr/memoire/cadre-theorique-dun-memoire (visité le 02/08/2018).

16 Listes des tables, des figures et des fragments de code

Liste des tableaux

Liste des illustrations

Liste des fragments de code