1 Introduction

1.1 Contexte et enjeux du stage

D'une part, depuis quelques années, le « *Deep Learning* » et les « réseaux de neurones » ont connu une explosion de popularité. Ce qui se cache derrière cet engouement est la combinaison de théories relativement anciennes et d'avancées technologiques permettant la mis en œuvre desdites théories.

Un des domaines exploitant les performances de ces outils est le Traitement Automatique des Langues (TAL, *Natural Language Processing* en anglais). L'application au TAL des réseaux de neurones artificiels ou réseaux de neurones (*Neural Networks* en anglais) qui nous intéressera particulièrement dans ce mémoire est la création de Modèle de la Langue.

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

Ainsi, nous explorerons la question le l'application des méthodes du apprentissage profond du point de vue du TAL sur des ensembles de données de grande taille.

L'axe principal de ce mémoire est l'application de telles méthodes sur des gros 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 pragmatique d'optimisation lié au volume de données.

1.2 Objectifs du stage

Deux objectifs successifs se distinguent dans le stage.

L'objectif initial du stage est d'explorer une idée d'architecture de réseau de neurones artificiels innovante, imaginée par notre maître de stage Mr Cerisara.

À l'issue du deuxième mois du stage, au vu des résultats de l'architecture et de l'évolution du contexte, la mission du stage à aussi évoluée.

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

Les tenants et aboutissants des deux objectifs seront expliqués en détails dans le chapitre 4 et le chapitre 10.

1.3 Plan du rapport

Dans un premier temps, nous avons présenté à la fois le contexte, les enjeux, et les objectif généraux du stage.

Dans un second temps, nous étudierons plus en détail le cadre théorique du rapport, afin de définir les termes et concepts principaux utilisés dans ce rapport.

Dans un troisième temps, nous allons nous attarder plus en détail sur les différentes entités impliquées, en particulier sur l'équipe SYNALP (*SYmbolic and statistical NAtural Language Processing*) et ses entités parentes, ainsi que sur le projet PAPUD.

Dans un quatrième temps, nous allons décrire plus en détail les deux aspects du stage : l'idée d'architecture de réseau de neurones artificiels et l'intérêt d'une telle architecture d'un côté, et le projet PAPUD, ses implications et la portée du stage dans ce projet. Nous verrons aussi en quoi le projet PAPUD est dans la continuité de la première partie du stage.

Dans un cinquième temps, nous exposerons le déroulement pas-à-pas du stage, avec les obstacles rencontrés, la façon de les surmonter, et en quoi chaque résultat entraîne l'étape suivante.

Dans un sixième et dernier temps, nous ferons une rétrospective sur l'avancement des objectifs, la qualité des résultats obtenus, et les apports du stage.

Dans un premier temps, nous avons présenté à la fois le contexte, les enjeux, et les objectif généraux du stage.

Dans un second temps, nous étudierons plus en détail le cadre théorique du rapport, afin de définir les termes et concepts principaux utilisés dans ce rapport.

Dans un troisième et un quatrième temps nous développerons les deux parties du stage, le projet Réseau de Neurones Récurrents Multi-Échelles Croissant (*Growing Multi-Scale Recurrent Neural Network* en anglais, GMSNN) et le projet PAPUD.

Pour cela, nous présenterons le contexte, les enjeux, et les entités impliquées dans le projet. Nous décrirons ensuite les modèles implémentés, avant de dérouler le travail effectué. Enfin, une conclusion résumera les points majeurs du projet.

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

Première partie

Projet GMSNN

3 Présentation du laboratoire et de l'équipe

3.1 Généralités

C'est dans le laboratoire du Laboratoire Lorrain d'Informatique et ses Applications (LORIA) que le stage s'est déroulé, au sein de l'équipe SYNALP dirigée par M. Christophe Cerisara, notre maître de stage.

3.2 Le LORIA

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

Structure administrative du LORIA

Il est dirigé par quatre instances[6] :

- **l'équipe de direction** : composée du directeur, de son adjoint, de la responsable administrative, et de l'assistante de direction ; assiste le directeur dans la prise et la mise en œuvre des décisions ;
- **le conseil scientifique** : composé du directeur du laboratoire, des deux directeurs adjoints et des scientifiques responsables des cinq départements du laboratoire composée de membres élus pour 4 ans et de membres nommés ; assiste le directeur dans la prise et la mise en œuvre des décisions ;
- **le conseil de laboratoire** : composé de membres élus pour 4 ans et de membres nommés ; émet des avis et conseille le directeur sur toutes les questions concernant l'UMR ;
- **l'Assemblée des Responsables des Équipes (AREQ)**.

La recherche au sein du LORIA

Le LORIA est l'établissement qui héberge l'équipe SYNALP, parmi de nombreuses autres équipes.

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

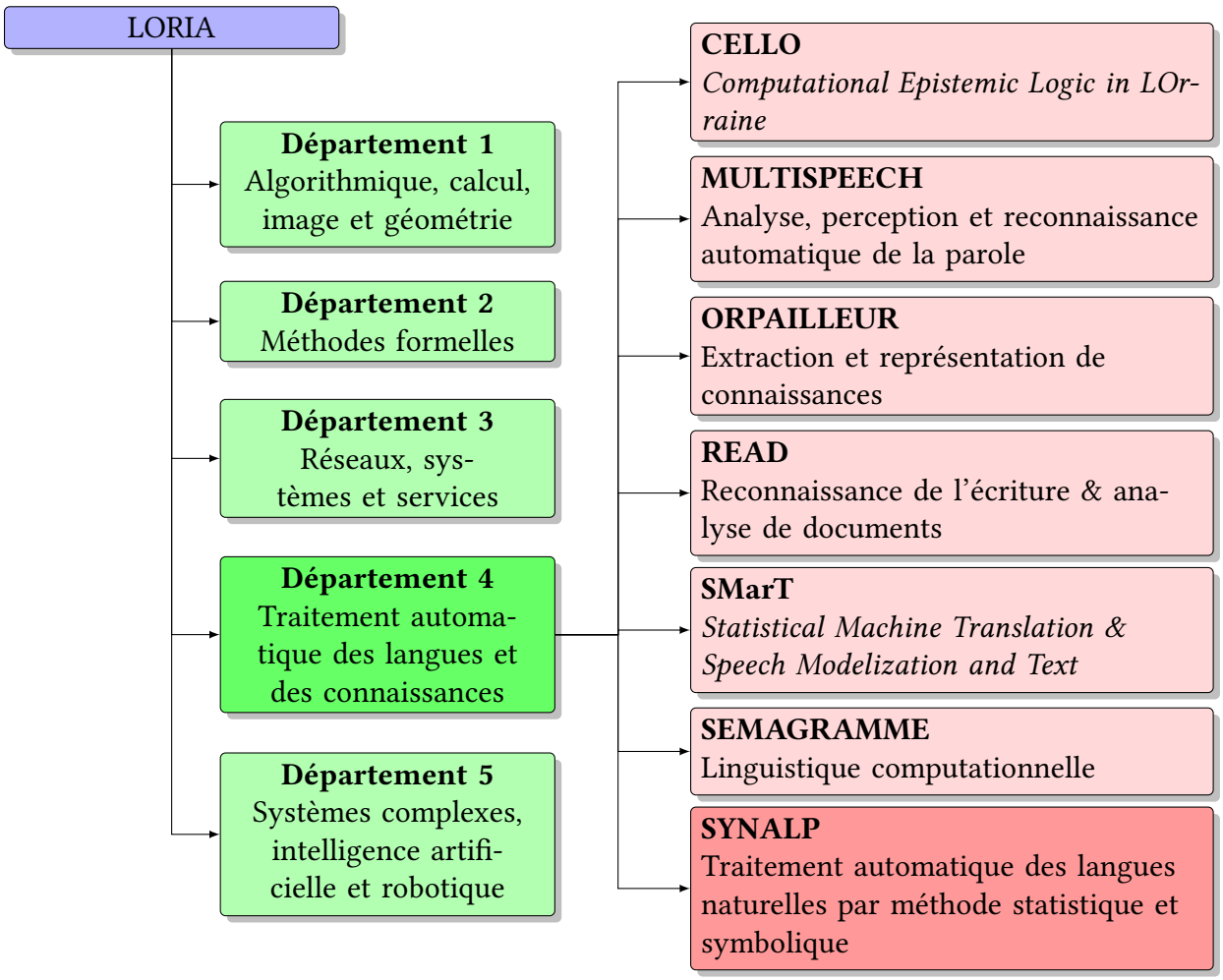


FIGURE 3.1 – Organigramme des départements du LORIA, et des équipes du département 4

La structure générale du LORIA en départements et plus en détail du département 4 est représentée sur l'organigramme de la Figure 3.1. Les thématiques générales de chaque département et des équipes du département 4 y sont présentées brièvement. Un organigramme complet du LORIA est disponible sur le site du laboratoire [7].

3.3 L'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.

Membres

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

Thématiques de recherche

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

Ainsi, les principaux sujets de recherche de l'équipe sont les Modèle de la 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 Modèle de la Langue, et l'élaboration d'outils et ressources utilisés en TAL. Nous verrons en détail pourquoi dans le chapitre **??**.

Pour en savoir plus

Des informations plus détaillées sur le LORIA sont disponible sur le site du laboratoire [8]. Par ailleurs, la liste complète des membres de l'équipe, ainsi que des informations plus détaillées sont disponible sur le site de SYNALP (en anglais) [9].

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

4.1 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 [10, 12, 11].

Les modèles basés sur les caractères se montrent particulièrement flexibles, car ces modèles « apprennent » les mots, à la place de se reposer sur des dictionnaires, très volumineux et qui ont des difficultés à gérer les fautes et les mots nouveaux.

Ces performances sont obtenues entre autres grâce à une gestion du contexte des données, typique des Modèle de la Langue.

4.1.1 Manque d'utilisation des gros volumes de données

Cependant, ces modèles sont souvent développés et entraînes 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 largement augmentés.

4.1.2 Problèmes de mémoire

Une des raison du manque d'augmentation de performance, typique des RNN, est la limite de rappel d'informations en « mémoire » (dans ses états cachés). Ce sont ces informations qui permettent la gestion du contexte.

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 ; un GRU peut se rappeler d'informations vielles d'une 100^{aine} d'entrées ; et un LSTM dépasse difficilement les 200 entrées.

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

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

4.2 Solution proposée

L'architecture proposée par notre maître de stage vise à la fois à tirer partie des grands volumes de données, et à permettre au modèle d'établir dépendances de haut niveau, voir des connaissance contextuelles externes.

L'architecture et ses caractéristiques sont décrits en détail dans le chapitre 6.

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

4.3 Projet GMSNN

La mission qui nous à été confiée est la création d'un apprentissage automatique basé sur l'architecture GMSNN.

L'implémentation devait se réaliser à partir d'une base de code sur laquelle notre maître de stage avait commencé à travailler (plus de détails sont disponibles sous-section 7.2.1).

À cela s'ajoute l'exploration du potentiel de l'architecture, par le biais d'une amélioration du modèle créé à l'aide d'optimisations classiques et de changements de l'architecture.

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

4.4 Organisation du travail

Durant ce projet, nous avons travaillé individuellement.

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

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

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

4.4.1 Organisation initiale du travail

Nous avons prévu l'organisation temporelle du travail dès la prise de connaissance du sujet définitif du stage (la réalisation de l'architecture proposée).

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 4.1 représente cette répartition prévue du travail.

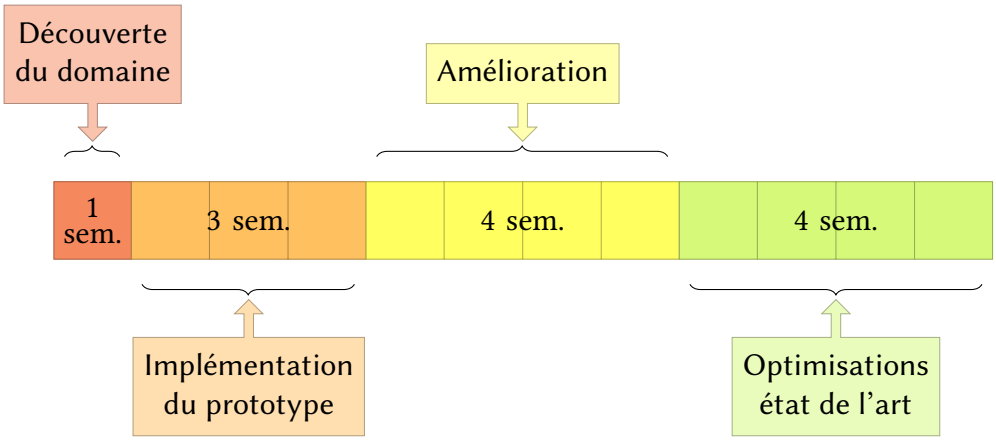


FIGURE 4.1 – Répartition prévue du travail. Une case correspond à 1 semaine de travail.

4.4.2 Déroulement réel du projet

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

Cependant, comme décrit section 7.7, nous avons décidé d'interrompre ce projet pour nous consacrer au projet PAPUD.

La Figure 4.2 représente la répartition réelle du travail.

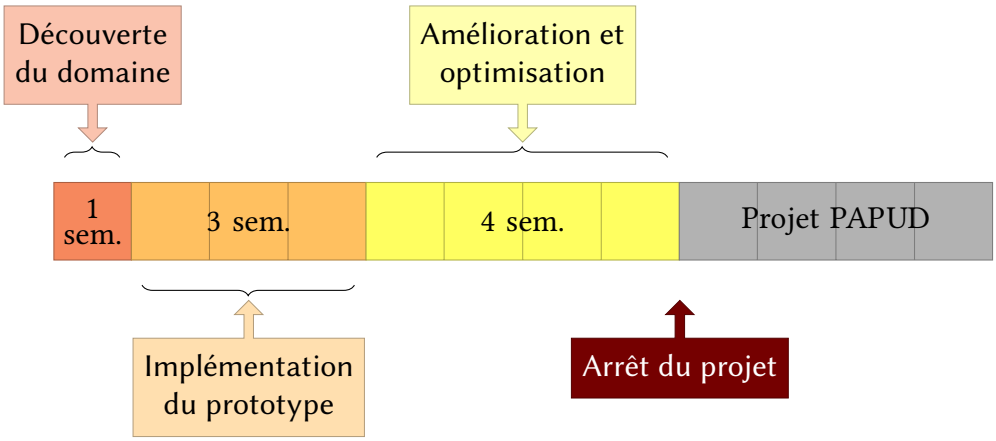


FIGURE 4.2 – Répartition réelle du travail. Une case correspond à 1 semaine de travail.

4.5 Outils

4.5.1 Langage et librairie

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

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

4.5.2 Grid5000 et les machines distantes

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

Ces machines font partie du réseau Grid5000. « Grid'5000 est un banc d'essai à grande échelle et polyvalent pour la recherche expérimentale dans tous les domaines de l'informatique, avec un accent sur l'informatique parallèle et distribuée, y compris Cloud, HPC et Big Data. » D'après le site de Grid5000 [13].

Un des avantage de cet outil est la présence de machines spécialisé dans les calculs GPU [1], qui sont celles que nous avons utilisées.

1. L'utilisation de GPU pour l'entraînement des réseau de neurones artificiels est une pratique fréquente en apprentissage profond, car elle permet d'accélérer les calculs effectués.

5 Données disponibles

Les données d'entraînement utilisées proviennent du Wikipédia anglais. Elles sont tirée du fichier « enwik8 » utilisé pour le prix Hutter [14, 15]. Ce fichier est composé d'environ 100 000 000 caractères.

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

Deux versions alternatives de ce corpus ont été utilisées.

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

5.1 Extrait des données d'entraînent

Voici un extrait des données brutes avant le découpage en caractères.

```
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.

== Origins and predecessors ==

[[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. ''&quot;[http://www.marxists.org/archive/marx/works/1884/origin-family/index.htm Origins of the Family, Private Property, and the State]&quot;'', 1884.&lt;/ref&gt;
[[Image:WilliamGodwin.jpg|thumb|right|150px|William Godwin]]
```

1. XML (*eXtensible Markup Language* en anglais), « est un langage informatique qui sert à enregistrer des données textuelles. [...] Ce langage , grosso-modo similaire à l'HTML de par son système de balisage, permet de faciliter l'échange d'information sur l'internet. » D'après le glossaire sur *infowebmaster* [16]. Il s'agit du format sous lequel sont stockés les articles Wikipédia.

Fragment de code 5.1 – Extrait des premières lignes du fichier enwik8, correspondant à l'article sur l'ansarchisme.

5.2 Prétraitement des données

Le prétraitement des données est composé du découpage du document, et du remplacement des caractères

Comme mentionné dans la sous-section 7.6.2, un défaut dans le prétraitement à mené à la disparition des espaces.

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

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

6 Description de l'architecture proposée

6.1 Propriétés du modèle

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

Elle est caractérisée par trois propriétés majeures :
— la structure récurrente ;
— l'utilisation de plusieurs échelles ;
— la croissance du modèle.
Nous avons nommé cette architecture GMSNN en considérant ses principales caractéristiques.

6.1.1 Récurrence du modèle

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

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

En apprentissage profond, le type de réseau de neurones artificiels considéré le plus adapté à la manipulation de séquences est le RNN (voir sous-section 2.3.5).

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

6.1.2 Passer à l'échelle

Comme décrit sous-section 4.1.2, les RNNs ont un problème inhérent de capacité mémoire, qui limite la distance des dépendances apprises par le modèle.

Afin de compenser ce défaut, l'architecture GMSNN s'appuie sur des couches de plus en plus vastes, appelées « échelles » dans ce rapport. Chacune de ces couches est un RNN, comme montrée dans la **??**.

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

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

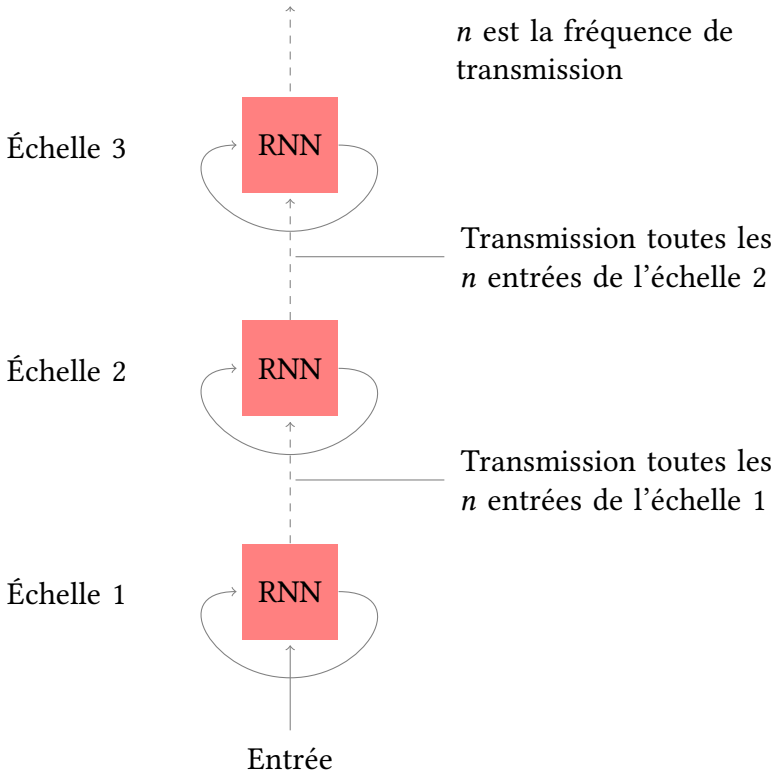


FIGURE 6.1 – Principe de transmission de l'information d'une échelle à la suivante. Ici, la fréquence de transmission est notée n.

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

Plusieurs niveaux d'abstraction de l'information

Une autre caractéristique importante du GMSNN est qu'une échelle prend en entrée les informations **abstraites** par la couche précédente. Ainsi, on peut s'attendre à ce que chaque échelle ajoute un niveau d'abstraction supplémentaire au modèle, comme sur la Figure 6.2.

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

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

En effet, comme décrit dans la sous-section 6.1.2, le nombre d'échelles est dépendant du nombre d'entrées totales.

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

Ainsi, au fur et à mesure de l'entraînement, le modèle croit à la façon d'une pyramide.

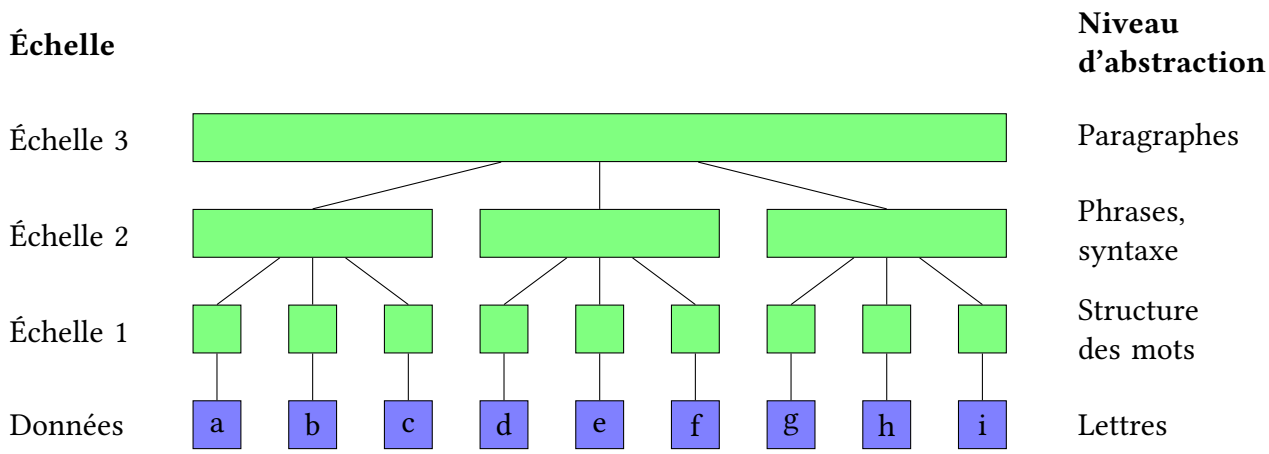


FIGURE 6.2 – Différentes échelles, et les niveaux d'abstraction que l'on pourait attendre de celles-ci. Chaque bloc vert correspond à une entrée pour l'échelle correspondante. Ici la fréquence de transmission vaut 3. Les blocs bleus en bas du graphique correspond aux caractères qui sont fourni en entrée au modèle.

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

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

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

Croissance potentiellement infinie du modèle

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

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

Comme décrit sous-section 7.5.1, cette propriété à rapidement été abandonnée.

7 Réalisation

7.1 Recherche documentaire

La première partie du projet, qui à durée à peu près une semaine, à été la recherche documentaire et la prise en main des outils. Ce travail à été effectué à partir des documents fournis par notre maître de stage, de la documentation de PyTorch[17, 18, 19, 20] et de Grid5000[22, 21], complétés par nos recherches personnelles.

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

7.2.1 Travail effectué

La deuxième partie du projet à été la prise en main de la base de code fournie. Il s'agit d'une implémentation état de l'art d'un Modèle de la Langue au niveau du caractère, sur laquelle notre maître de stage avait commencé à travailler. Le code d'origine provient du dépôt « awd-lstm-lm »[23], qui contenait un modèle état de l'art de Modèle de la Langue au niveau du caractère.

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

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

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

7.2.2 Modèle ré-implémenté simplifiée

Le modèle simplifiée produit est composé d'un module encodant les caractères, d'un RNN particulier (un LSTM, voir sous-section 2.3.5), et d'un module produisant une distribution de probabilité sur les caractères connus. La Figure 7.1 représente cette architecture.

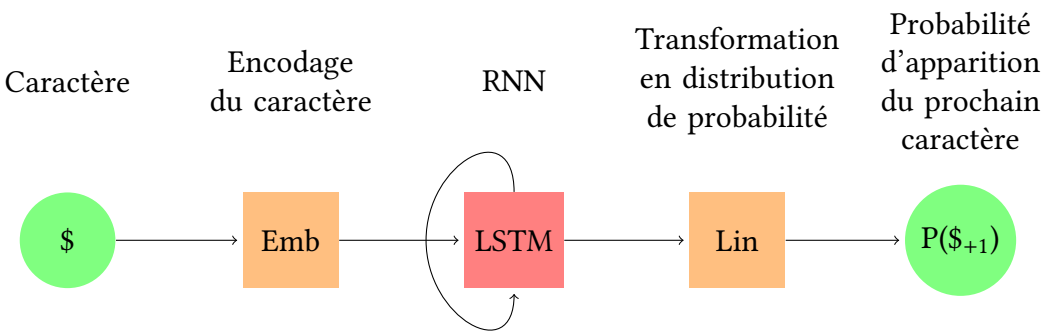


FIGURE 7.1 – Architecture du modèle réimplémenté. Le modèle prend en entrée des caractères, et produit des probabilités sur quel caractère apparaîtra ensuite.

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

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

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

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

7.2.3 Conclusion

Cette étape, outre la préparation du code, à permit la mise en œuvre et une meilleur compréhension des concepts appris durant l'étape précédente. C'est la première pierre de l'édifice qu'est le GMSNN.

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

7.3 Implémentation du nouveau modèle

7.3.1 Travail effectué

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

L'architecture du GMSNN est identique à celle du modèle ré-implémenté, mis à part le RNN qui est remplacé par le module GMSNN (voir Figure 7.2). C'est sur ce nouveau module que le reste du travail au cours du projet GMSNN à été effectué.

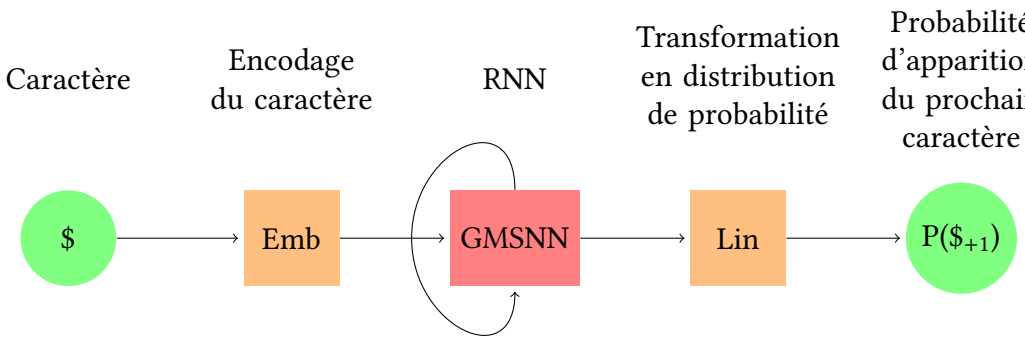


FIGURE 7.2 – Architecture du modèle réimplémenté. Le modèle prend en entrée des caractères, et produit des probabilités sur quel caractère apparaîtra ensuite.

Ce prototype est une implémentation naïve de l'architecture, permettant de mettre en place les mécanismes de base du modèle.

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

Chaque couche de l'architecture (chaque « échelle ») est un LSTM, comme dans le modèle d'origine.

7.3.2 Transmission d'information

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

Par exemple, pour fréquence de transmission de 3 :
- toutes les 3 entrées de la couche $n-1$, la couche n reçoit de l'information de la couche $n-1$;
- toutes les 3 entrées de la couche n (soit toutes les 3^2 entrées de la couche $n-1$), la couche $n+1$ reçoit de l'information de la couche n.

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

Le choix s'est porté sur l'état caché, qui contient les informations abstraite de la séquence de caractères, contrairement à la sortie qui contient les informations sur le caractère suivant uniquement.

Ensuite, il à fallu déterminer comment regrouper les informations avant de les transmettre au module produisant la distribution de probabilité.

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

7.3.3 Conclusion

Cette partie du projet à permit la mise en œuvre de l'architecture proposée. La dernière étape est d'améliorer les performances du modèle, et de l'optimiser.

7.4 Intégration de systèmes de visualisation

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

7.4.1 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 [24, 25].

Malheureusement, aucune de ces librairies ne sont pas en mesure de supporter les architectures les plus complexes (en particulier celles qui impliquent des RNN).

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

7.4.2 Création d'un outil personnalisé

Nous avons donc réalisé un module capable d'enregistrer des données et de réaliser des graphiques. Nous nous sommes basés sur le module « matplotlib »[**matplotlib**] de Python, et sur une variante française du format CSV[**csv**].

Ce module à évolué tout au long du projet pour s'adapter à nos besoins.

L'intégralité des graphiques produits dans les divers rapports du projet (disponibles en annexes) à été produit avec ce module.

7.5 Optimisation et amélioration du nouveau modèle

Une fois le prototype fonctionnel, il à fallu améliorer les performances. Par performances, nous entendons principalement le temps nécessaire pour que la qualité prédictive du modèle dépasse un certain seuil.

Pour améliorer ce temps de calcul, il est possible de travailler sur deux dimensions :
— la **quantité de données** traitées en un laps de temps ; c'est une **stratégie quantitative** ;
— la **qualité** de l'apprentissage pour une quantité fixée de données ; c'est une **stratégie qualitative**.
Ainsi, plusieurs options sont possible :
— optimiser les algorithmes et le modèle pour réduire le temps nécessaire pour traiter les données ;
— améliorer le modèle en réglant les paramètres (comme la fréquence de transmission) ou en implémentant de nouvelles mécaniques ;

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 et d'analyse et d'optimisation, avec peu d'implémentation de nouvelles mécaniques dans le modèle.

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

La première optimisation à été de changer la façon de regrouper les informations de toues 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 » (Figure 7.3a).

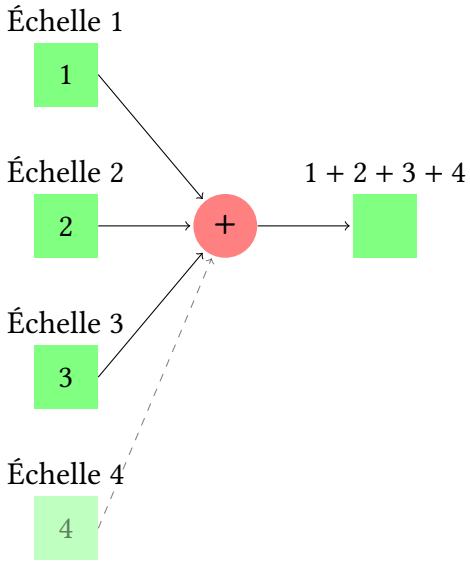
Après discussion avec notre maître de stage, la stratégie d'agrégation à été changé en une concaténation des sorties.

Comme montré dans la Figure 7.3b, la taille du tenseur concaténé change en fonction du nombre d'entrées. La manipulation de tenseurs de taille non fixée est très ardue dans ce cas précis, bien que nous ne développerons pas plus avant les raisons de cette difficulté.

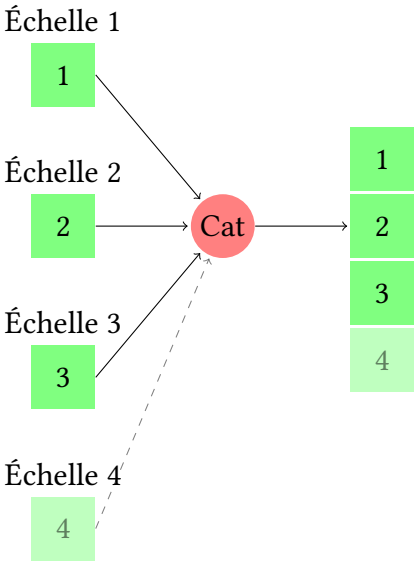
Cela à nécessité l'abandon de la propriété de croissance à l'infini de l'architecture (décrite sous-section 6.1.3), au profit d'un nombre maximal d'échelles défini à l'avance ou déterminée à l'aide d'une formule en fonction des données disponibles (décrite sous-section 6.1.3).

La stratégie par concaténation est plus lente en terme de temps de calcul que la stratégie additive, cependant pour le même temps de calcul elle permet d'obtenir de meilleurs résultats (Figure 7.4).

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



(a) Stratégie d'agrégation additive.



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

FIGURE 7.3 – Stratégies d'agrégation

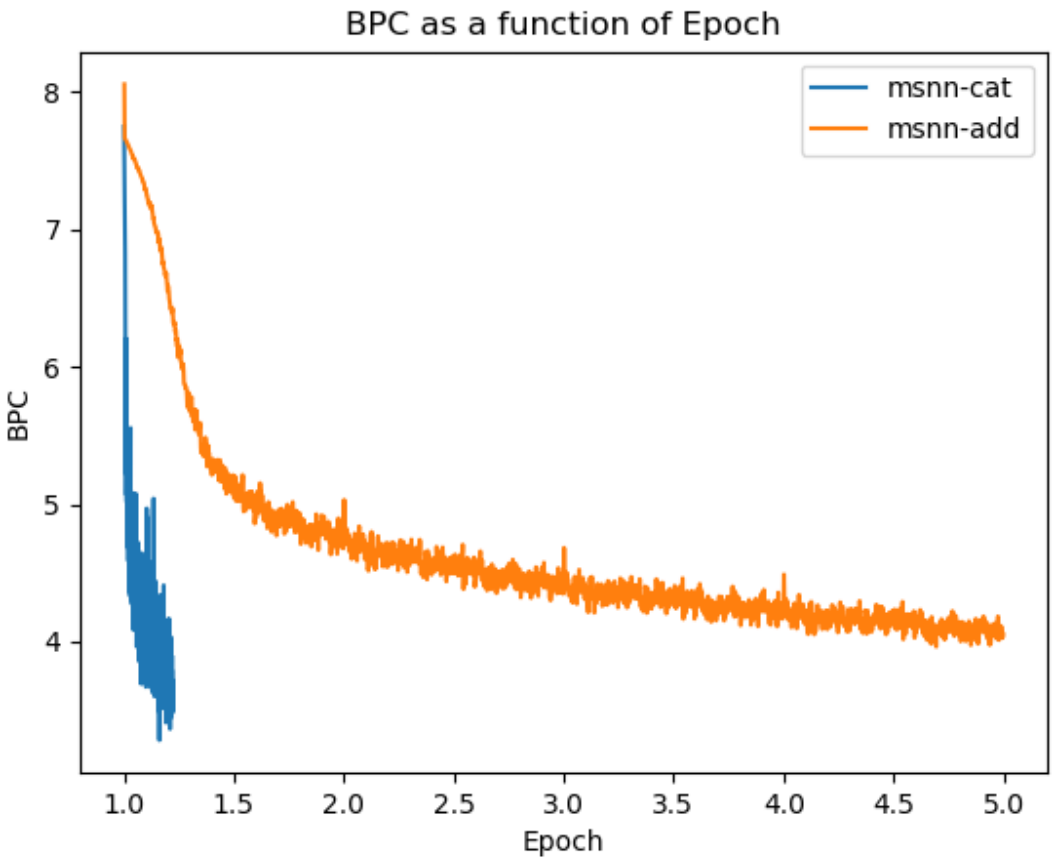


FIGURE 7.4 – Performances comparées des stratégies par concaténation (msnn-cat) et additive (msnn-add). Le temps de calcul alloué est identique. Avec la concaténation on entraîne le modèle sur 1/4 des données, avec l'addition on l'entraîne 5 fois sur l'ensemble des données. Avec la concaténation, on obtient une BPC de 3.5, alors qu'on obtient une BPC de 4 avec l'addition.

7.5.2 Sauvegarde, interruption et reprise de l'entraînement

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

En effet, avec des entraînements très lents et donc longs, il était nécessaire de pouvoir suspendre l'entraînement affin de répartir le temps de calcul sur plusieurs session de plusieurs heures. De plus, les sauvegardes permettent de conserver le modèle une fois entraîné.

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

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

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

7.5.3 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. Ces optimisations sont :
- l'utilisation de *batches* simultanés (voir sous-section 7.5.4) ;
- l'augmentation du nombre de paramètres du modèle (voir sous-section 7.5.5) ;

Consommation de mémoire et de temps de calcul accrue

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

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

Un premier correctif à fourni une amélioration notable mais insuffisante. Il remplaçait le LSTM de chaque couche (voir Figure 7.3.1) par un RNN basique, moins gourmand. Cela permettait aussi d'éliminer une redondance entre l'architecture LSTM et l'architecture GMSNN.

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

Estimation de la consommation normale du modèle

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

Résolution des fuites

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

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

L'annexe A.4 contient une des tentatives de résolution du problème.

Conclusion

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

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

7.5.4 Entraînement par exemples simultanés ; *Batch*

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

batch

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

Le découpage des données en *batches* 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 appelé *mini-batch*[26].

Il s'agit d'une optimisation rependue pour l'entraînement de réseaux de neurones[26]. Elle est souvent couplée à un entraînement simultané sur plusieurs *batches*, décrit dans la partie suivante.

Batches parallèles

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

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

La version de l'algorithme de parallélisation utilisé est similaire à celle décrite dans l'article [27]. Elle est géré de base par PyTorch.

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

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

Une analyse approfondie à permit d'établir une méthode pour contourner et compenser la majorités des aspect du problème.

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

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

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

Conclusion

Les problèmes liés à l'utilisation de *batches* ont été en majorité résolus ou écartés. Après consultation avec notre maître de stage, nous avons décidé d'utiliser l'entraînement par *batches* malgré les problèmes restants.

7.5.5 Augmentation du nombre de paramètres

Les paramètres, ou poidss, du modèle sont des valeurs qui varient au long de l'entraînement du modèle. On peut dire que ce sont ces valeurs qui « apprennent ».

Le fait d'augmenter ces paramètres augmente la qualité de l'apprentissage, et la précision du modèle appris. Mais cela se fait au coût 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 7.5.3), ces coûts ne sont plus dramatiques.

Il existe plusieurs façon de mettre en place cette optimisation :
— augmenter le nombre de neurones par couches ;
— augmenter le nombre de couches dans le RNNs qui compose chaque « échelle ».

Conclusion

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

7.5.6 Entraînement couche par couche

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

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

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

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

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

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

Performances

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

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

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

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 7.5.5 pour l'impact du nombre de paramètres).

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

7.5.7 Conclusion

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

De plus, l'algorithme présenté sous-section 7.5.6 à démontré une faiblesse majeure de l'architecture GMSNN.

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

7.6 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.

7.6.1 Exemples

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

Age du modèle : 465 époques.

Score BPC du modèle : 1.839.

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 7.1 – 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 7.2 – 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]].

```

```
4 :'''maimals,anackreloquedoutwidthofgrawwithluteframedapproyingtoundernverby[[
    hebesination]]of&lt;/smalkan,
    instablishedacondorttodevelopedframesbeforestatedwinkingaroundinrational
    hicarefartoredonaftercanbeagainsthatgroupswouldnear,
    notwhatwasthatisstillastructionCenter,toDagnythat
5
6 On[[ëÃ£teleofAirej]]
7 [[cy:Alaska]]
8 [[no:Arni-Anchorages]]
```

Fragment de code 7.3 – Exemple 3 : des termes balisé comme dans l'exemple 2, et une autre suite de carractères.

7.6.2 Manque d'espaces

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

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

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

7.6.3 Quelque 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 7.2, ligne 8) ou [[cy:Alaska]] et [[no:Arni–Anchorages]] de l'exemple 2 (Fragment de code 7.2, lignes 7 et 8), on a des doubles crochets, qui sont correctement ouverts puis fermés. On remarque aussi la présence de séparateurs (: et |).

Ces structure sont similaire aux annotations présentes dans le code Wikipédia.

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 7.1, ligne 1) ;
- de même pour la suite de mots oldWarInstartedbythePhilosophy : old War In started by the Philosophy (Fragment de code 7.1, ligne 1) ;
- on trouve aussi des noms propres comme le nom de pays : UnitedStates (Fragment de code 7.2, ligne 8) et Alaska (Fragment de code 7.3, ligne 7).

7.7 Analyse des résultats et arrêt du projet

7.7.1 Analyse des résultats

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

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

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

7.7.2 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.

7.7.3 Arrêt du projet et début du projet suivant

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

Celle-ci avait permis de définir les objectifs du projet ITEA3-PAPUD, cas d'utilisation BULL (voir chapitre 10), qui ont été influencés par nos conclusions.

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

8 Conclusions sur le projet GMSNN

8.1 Retour sur le travail effectué

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

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

Bien que le l'architecture GMSNN n'ai pas atteint les performances espérées, le modèle produit est robuste, rapide, et peu volumineux. De plus, l'algorithme présenté sous-section 7.5.6 à démontré une faiblesse majeure de l'architecture GMSNN. Enfin, les problèmes rencontrés dans ce projet ont permit de tirer des conclusions très utiles pour de prochains projets :

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

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

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

8.2 Apport personnel du projet

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

8.3 Discussion et perspectives

8.3.1 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. La pertinence de nos choix est d'autant plus douteuse 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 PA-PUD (voir section 13.7).

Parmi les très nombreuse possibilités envisagées, on trouve aussi :
- l'utilisation d'un RNN pour interpréter les informations des différentes échelles ;
- la poursuite de l'utilisation de l'algorithme couche par couche, en poussant chaque échelle au maximum de ses capacités avant d'intégrer de nouvelles couches ;
- le changement de l'architecture de pyramidale à parallèle, c'est à dire que chaque échelle serait indépendante, similairement à l'article [28].

8.3.2 Travaux similaires

Dans un article datant du 27 Juillet 2018 [29], 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

9 Présentation du projet ITEA3-PAPUD, cas d'utilisation BULL

La seconde partie du stage s'intègre dans le projet PAPUDprojet ITEA (*Information Technology for European Advancement*)-PAPUD (*Profiling and Analysis Platform Using Deep Learning*), en particulier dans le cas d'utilisation BULL. Nous verrons en détail les objectifs du projet dans la **??** (page **??**).

Le projet ITEA3-PAPUD, cas d'utilisation BULL est un projet de l'initiative ITEA3 du réseau EUREKA.

9.1 Collaborateurs

Les personnes avec lesquelles nous avons collaboré durant ce projet sont notre maître de stage M. Christophe Cerisara, 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 à rejoint vers la fin du stage.

9.2 EUREKA et ITEA3

« EUREKA est une initiative européenne, intergouvernementale, destinée à renforcer la compétitivité de l'industrie européenne. » D'après Wikipedia [30].

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.

9.3 Projet PAPUD et cas d'utilisation BULL

C'est lors de la troisième vague d'appels à projets d'ITEA3 que le projet PAPUD à 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 à pour objectif l'analyse des volumes de données devenus stop grand pour être gérés de façon traditionnelle. Ainsi, le projet PAPUD s'inscrit dans la dynamique d'ITEA3.

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 9.1 – Informations générales sur le projet PAPUD, d'après le site de ITEA3 [31]

9.4 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 à été rachetée en 2014 par le groupe ATOS.

Secteurs d'activité

D'après le site d'ATOS[32], 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 [31].

10 Projet PAPUD

10.1 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 panes, afin de pouvoir les éviter.

Les données disponibles pour remplir cette tache sont des fichiers de journaux systèmes (décrits en détail dans le chapitre 11) de très grande taille. Ils contiennent de nombreuses informations sur les évènements se déroulant dans les outils.

10.2 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 anomalies (signes de pannes) dans les journaux systèmes.

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

Ces journaux 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 à 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.

10.3 projet PAPUD

La tache qui nous à 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 12.

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

10.4 Organisation du travail

Contrairement au projet GMSNN, d'autres collaborateurs participaient a ce projet (voir section 9.1). Étant, avec notre 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 à 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. Le rapport d'une des réunion est disponible à l'annexe B.6.

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

10.4.1 Organisation initiale du travail

Ce 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 à 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 à 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 à été décidée, de façon à augmenter le temps dédié au travail sur le projet. Cela c'est fait en considérant les disponibilités de notre maître de stage, des autres collaborateurs ainsi que les nôtres.

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

10.4.2 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 10.1.

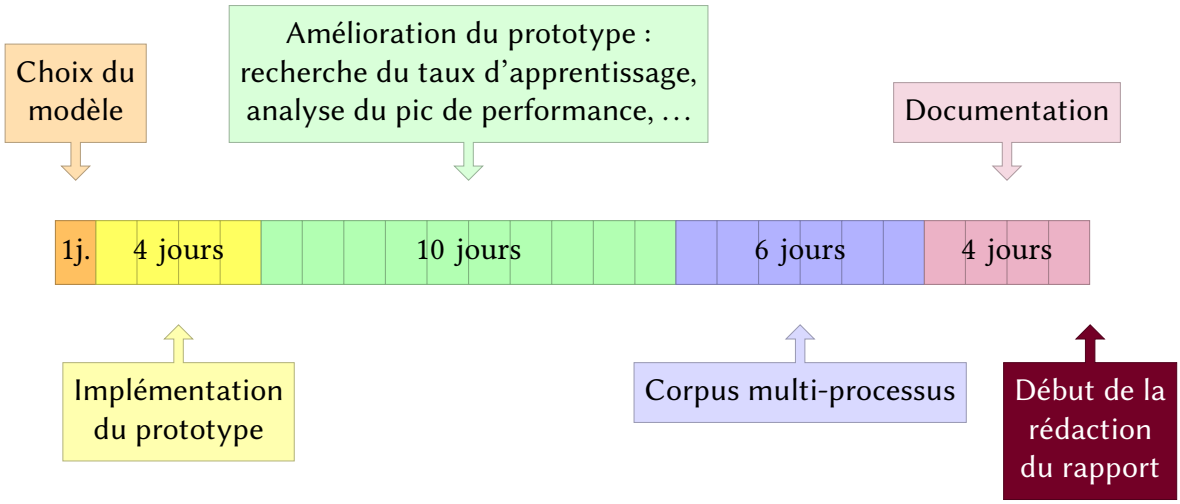


FIGURE 10.1 – Répartition du travail. Une case correspond à 1 jour de travail.

10.5 Outils

Les outils choisis sont Python et PyTorch, 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, PyTorch possédant une fonctionnalité permettant la conversion d'un modèle vers les autres principales librairies disponibles pour Python et quelques autres langages.

11 Données disponibles

Les données disponibles sont des des fichiers de journaux systèmes.

Ce sont des lignes de texte en anglais extrêmement structuré, 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 à 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 systèmes contienne un nombre très grand de message, 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 moins de 10 MiB, soit 40 000 fois moins.

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

11.1 Extrait des données d'entraînent

Les données sont confidentielles utilisées sont confidentielles. Ainsi, l'extrait présenté ici à 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 à été renommé « OOOOOO ».

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

Fragment de code 11.1 – Exemple d'une ligne extraite des journaux systèmes. On à ici dans de gauche à droite : le *timestamp*, la date, l'heure, l'utilisateur (OOOOOO), le processus (authpriv), le type de message (info), et le message.

11.2 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 ».

11.2.1 Traitement des codes hexadécimaux

Par la suite, le remplacement de tout code hexadécimal comme 0x005f par un caractère spécifique du dictionnaire à été ajouté (voir section 13.6). De cette façon, 0x005f, 0 xffff , 0x0007 sont remplacé par le caractère <hexX4>, tandis que 0000005f et 89abcdef sont remplacé par <hex8>. Cela permet de s'abstraire de la valeur du code sans perdre l'information liée à sa présence.

12 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 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 11.2).

12.1 Utilisation d'un réseau de neurones artificiels basique

Comme présenté dans le chapitre 10, un réseau de neurones artificiels basique, intégralement connecté (comme celui présenté dans la Figure 2.3), à été choisi pour le modèle.

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

12.2 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* [1].

Grâce à une opération appelée « *max-pooling* », qui correspond à prendre pou chaque case du tenseur final le

12.3 Représentation de l'architecture

La façon dont s'agencent les différents modules de l'architecture est représentée dans la Figure **??**.

Figure 12.1 – Architecture du modèle du projet PAPUD.

1. Voir Figure 7.2.2, page 26.

12.4 Lien avec l'état de l'art

Il à 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 [33].

13 Réalisation

13.1 Définition du modèle

La toute première étape du projet à été de définir l'architecture à utiliser (décrite chapitre 12). C'est sous la forme de réunions que cette étape s'est déroulée.

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

13.2 Implémentation du modèle et transfert des outils de base

Une fois le modèle défini, l'étape suivante à 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 des données ;
- un algorythme d'entrainement 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 7.4) ;
- un système de sauvegarde et d'intéruption de l'entraînement similaire à celui du projet précédent (voir sous-section 7.5.2)

13.2.1 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.

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

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

Pour cela, un premier prototype à été réalisé. Il utilise des mécanismes optimisés du langage Python pour la lecture de fichier. L'objectif de ce prototype est de maintenir uniquement une fiable 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'algorythme d'entrainement ;
4. on recommence de l'étape 1 avec la portion suivante des données.

L'intégration de cet outil s'est faite en parallèle de la suite du projet.

13.4 Entrainement par batch

La première optimisation du modèle à été l'intégration de l'algorithme d'entraînement par *batches*[1].

Il à été nécessaire de définir le nombre de *batches* optimal en terme de temps de calcul et de performance. Les détails concernant ce choix sont disponibles dans l'annexe B.4.

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

13.5 Changement de métrique pour évaluer la qualité du modèle

Cette partie du projet n'est pas dédie à une optimisation, mais à la rectification d'une métrique 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 à donc été remplacé par une métrique nommée « précision », qui est le taux de caractères correctement prédits (le bon caractère au bon endroit).

13.5.1 Précision de base

Ce changement de métrique à 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 permet de déterminer si le modèle produit apprend réellement de l'information, et dans quelle mesure.

Le détail de ces résultats est disponible dans les rapports des annexes B.2 et B.6.

1. Voir sous-section 7.5.4, page 33.

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

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

Il à donc été décidé, comme décrit sous-section 11.2.1 (page 52), d'intégrer au prétraitement des données la gestion de ces codes. Cette intégration à cependant dù attendre la fin de la réalisation de la nouvelle version du gestionnaire de données, décrite section 13.8 (page 58).

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

13.7 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 des plus classiques[34].

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. La procédure classique en apprentissage profond repose sur l'essai-erreur : on entraîne le modèle avec différentes valeurs pour le taux d'apprentissage, et on choisi celles qui à donné le meilleur résultat pour les entraînements à venir.

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

La nouvelle valeur définie pour le à permit d'augmenter largement la vitesse de convergence du modèle.

Le détail du processus de choix et des résultats est disponible dans les annexes B.7 et B.8.

13.8 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 à été intégralement doté de tests unitaires et testé.

13.8.1 Fichiers multiples

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

13.8.2 Prétraitement modulaire

D'autre part, durant le déroulement du projet nous avons remarqué que l'ajout 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 **??** et **??** à été intégré.

13.8.3 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.

13.8.4 Performances

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

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

14 Conclusions sur le projet PAPUD

14.1 Retour sur le travail effectué

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

On peut compter 4 optimisations principales, outre le prétraitement des codes hexadécimaux et le choix de la taille des*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.
On peut noter l'attention particulière portée à la documentation du code. En effet, le travail effectué n'est que la première pierre 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 à 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.

14.2 Apport personnel du projet

La réalisation de ce projet nous à 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 à été une expérience enrichissante.

Enfin, la documentation du code et des outils, en particulier les derniers jours du projet qui y étaient dédiés, nous à permit de nous perfectionner dans l'art de la documentation en Python. Par là nous entendons surtout les techniques de typage[35, 36] et les pratiques de rédaction de *docstring*.

14.3 Discussion et perspectives

14.3.1 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 ? ?).

15 Rétrospective sur le stage

15.1 Bilan sur le 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 à été interrompu, le principal objectif qui était de sonder le potentiel de l'architecture à été rempli dans une certaine mesure. De plus, le second projet qui est la continuation du premier, à 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.

15.2 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.

Nous avions 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.

La disponibilité limitée de notre maître de stage, qui est aussi le chef de l'équipe SYNALP

L'essentiel du stage s'est déroulé en autonomie, avec la possibilité de consulter notre maître de stage. 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 à permit 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.

15.3 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.

Ce stage nous à permis d'avoir un aperçu du travail de chercheur. D'une part durant les premier mois, pendant lesquels nous avons explorer une problématique jamais explorée. D'autre part en travaillant avec l'équipe du projet PAPUD, qui nous à montré un aspect du travail de recherche plus appliqué.

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

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

En définitive, ce stage à 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 de notre maître de stage, des collaborateurs du projet PAPUD et de nos camarades stagiaires et doctorants.