7.4 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.

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 [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.

7.4.2 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).

7.5 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 modulant 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.

7.5.1 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 7.3a, page 37).

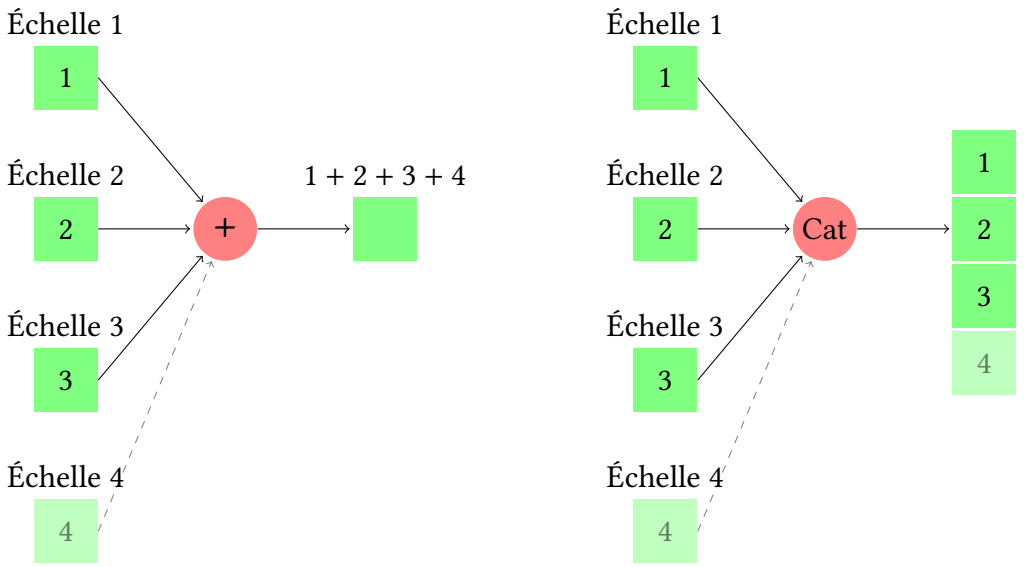
Après discussion, la stratégie d'agrégation à été changée en une concaténation des sorties.

Comme montré dans la figure 7.3b (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 ardue dans ce cas précis, mais nous ne développons pas ici cette difficulté.

En contrepartie, la propriété de croissance à l'infini de l'architecture (décrite sous-section 6.1.3, page 29) a été abandonnée, 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, 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 calculs elle permet d'obtenir de meilleurs résultats. La figure 7.4 (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 119).



(a) Stratégie d'agrégation additive (b) Stratégie d'agrégation par concaténation

FIGURE 7.3 – Stratégies d'agrégation

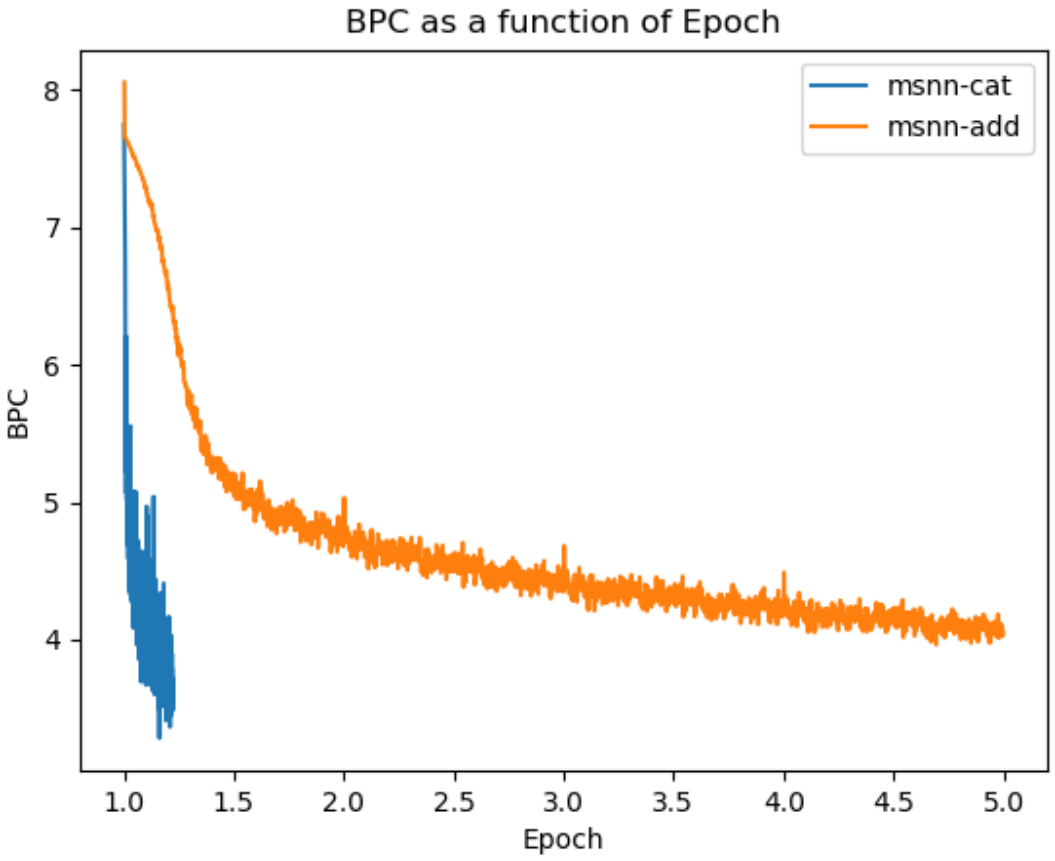


FIGURE 7.4 – Performances comparées des stratégies par concaténation *(msnn-cat)* et additive *(msnn-add)*

7.5.2 Sauvegarde, interruption et reprise de l'entraînement

Une fonctionnalité s'est très vite imposée comme essentielle : la sauvegarde du modèle et l'interruption et 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 permettent de conserver le modèle une fois qu'il est entraîné.

Le système mis en œuvre permet d'effectuer cycliquement des sauvegardes 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 dysfonctionnement initial des outils fournis par PyTorch. La conception d'un système de sauvegarde personnalisé est alors apparue nécessaire, mais malheureusement complexe. Cependant, une mise à jour majeure de la librairie a résolu le problème, et c'est finalement avec les outils de PyTorch que le système de sauvegarde à été mis en œuvre.

Plus de détail sur le système de sauvegarde sont disponibles dans le rapport de l'annexe A.3 (page 95).

7.5.3 Tentatives d'optimisation, fuites de mémoire et lenteur de l'entraînement

Les optimisations testées par la suite ont révélé des fuites de mémoire et mis en lumière une lenteur excessive de l'apprentissage.

Les optimisations en question ont été suspendues le temps de la résolution de ces deux problèmes. Il s'agit de :
- l'utilisation de *batches* simultanés (voir sous-section 7.5.4, page 40) ;
- l'augmentation du nombre de paramètres du modèle (voir sous-section 7.5.5, page 41) ;

Consommation de mémoire et de temps de calcul accrue

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

Cette consommation accrue a causé le plantage [2] de plusieurs tests, révélant la présence de fuites critiques de mémoire. Un ralentissement progressif de l'entraînement a aussi été mis en évidence pendant l'analyse du problème. Le plus surprenant a été la corrélation forte observée 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 7.3.1, page 33) par un RNN basique, moins gourmand. Cela permettait aussi d'éliminer une redondance entre l'architecture LSTM et l'architecture GMSNN.

2. Un plantage en informatique est l'arrêt d'un programme causé par un dysfonctionnement.

Estimation de la consommation normale du modèle

La première étape, qui est détaillée dans l'annexe A.2 (page 90), 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. Ceci a confirmé que l'explosion de la consommation n'était pas due à l'architecture en elle-même et qu'il s'agissait bien d'une anomalie dans le fonctionnement du programme.

Résolution des fuites

L'analyse et la résolution des fuites de mémoire s'est révélée ardue. Si quelques fuites mineures ont été simples à détecter et réparer, la principale fuite était due à une spécificité non documentée 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 le mode de gestion de ce graphe, couplé aux spécificités de l'architecture GMSNN, qui est la cause de la principale fuite mémoire.

Le rapport dans l'annexe A.4 (page 99) présente un extrait de la résolution du problème.

Conclusion

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

On notera 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 de 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

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

Batch

Un *batch* (anglais pour lot), est un paquet 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 meilleures performances. L'algorithme basé sur ce principe est appelé *mini-batch* [23].

L'utilisation de cet algorithme est une optimisation répandue pour l'entraînement de réseaux de neurones [23]. 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 jour le modèle en fonction des différences observées sur l'ensemble des *batches*. Au final, les calculs des résultat sont parallélisés, et le coût de la mise à jour est mis en commun entre les *batches*.

Le temps de calcul est ainsi réduit drastiquement et la qualité de l'entraînement est augmentée, 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 nativement 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 a permis d'établir une méthode pour résoudre ou écarter la majorité des aspect du problème. Après consultation, nous avons décidé d'utiliser l'entraînement par *batches* malgré les problèmes non encore résolus.

Le rapport de l'annexe A.5 (page 101) contient les détails de l'analyse des problèmes théoriques de l'utilisation de *batches* avec l'architecture GMSNN. Les rapports des tests de la solution retenue suite à l'analyse sont contenus dans les annexes A.13 (page 124), A.16 (page 140), A.17 (page 144) et A.18 (page 148).

L'annexe A.2 (page 90) contient les détails de l'impact de la taille des *batches* et du nombre de *batches* sur la consommation de mémoire.

7.5.5 Augmentation du nombre de paramètres

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

Comme décrit dans la sous-section 2.2.4 (page 13), l'augmentation du nombre de ces paramètres augmente la qualité de l'apprentissage et la précision du modèle. Mais le volume du modèle devient alors plus important, et plus de calculs sont nécessaires pour utiliser et entraîner le modèle. En conséquence, l'entraînement est plus lent et une consommation de mémoire est plus élevée.

Cependant, grâce aux optimisations mises en place durant la phase de résolution des fuites de mémoire (voir sous-section 7.5.3, page 38), ces coûts restent raisonnables.

Il existe plusieurs méthodes pour mettre en place l'augmentation du nombre de paramètres :
— augmenter le nombre de neurones par couches ;
— augmenter le nombre de couches dans le RNN qui compose chaque « échelle ».

Ces deux méthodes ont été testées, et aucune n'a apporté d'amélioration de la qualité de l'apprentissage, tout en allongeant la durée d'entraînement. En résumé, ces améliorations apportent des coûts supplémentaires sans aucun bénéfice. Par conséquent, aucune d'entre elles n'a 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. L'algorithme s'apparente aux algorithmes EM *(Expectation Maximization)*.

L'intuition à l'origine de notre algorithme est que pour apprendre les représentations de haut niveau, le modèle a besoin en premier lieu d'apprendre les représentations de bas niveau. Par exemple, sans distinguer les mots, il est difficile de construire des phrases cohérentes.

En suivant ce principe, 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 efficacement.

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

Le fonctionnement détaillé de l'algorithme est décrit dans l'annexe A.7 (page 106).

Performances

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

En outre, il n'y a aucune variation notable de la qualité de l'entraînement.

Les performances de l'algorithme sont disponibles dans le rapport de l'annexe A.19 (page 106).

Remise en cause de l'intérêt de l'architecture

Justement, comme une seule échelle apprend, on pourrait s'attendre à une baisse des performances. En effet, en entraînant une seule couche, on réduit le nombre de paramètres du modèle, avec une influence négative sur la performance (Voir sous-section 7.5.5 pour l'impact du nombre de paramètres, page 41).

Comme le modèle mono-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, 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 dans la sous-section 7.5.6 (page 41) a mis en évidence une faiblesse majeure de l'architecture GMSNN.

On peut aussi noter l'impact de 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.

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'exemples.

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 un aperçu plus concret qu'une mesure de la performance du modèle.

7.6.1 Exemples

Voici quelques exemples produits par le modèle. Parmi les versions du modèle enregistrées, nous avons choisi celle dont la performance mesurée est la meilleure (autrement dit, dont le score BPC est le plus bas).

Cette version a été entraînée pendant 465 époques, et son score BPC est de 1,839. Pour comparaison, le modèle état de l'art atteint 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 difficilement compréhensibles

```
+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]].

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

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

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

7.6.2 Manque d'espaces

Parmi ces exemples, on remarque immédiatement le manque de caractères d'espacement.

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 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 appris une langue moins structurée, donc plus difficile à apprendre, que l'anglais.

7.6.3 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 7.2, page 43, ligne 8) ou [[cy:Alaska]] et [[no:Arni–Anchorages]] de l'exemple 2 (Fragment de code 7.2, 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 structures sont similaire aux annotations présentes dans le code Wikipédia. On peut les voir dans le Fragment de code 5.1, 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 7.1, page 43, ligne 1) ;
- de même pour la suite de mots oldWarInstartedbythePhilosophy : old War In started by the Philosophy (Fragment de code 7.1, page 43, ligne 1) ;
- on trouve aussi des noms propres comme le nom de pays : UnitedStates (Fragment de code 7.2, page 43, ligne 8) et Alaska (Fragment de code 7.3, page 44, ligne 7).

7.7 Analyse des résultats et suspension du projet GMSNN

7.7.1 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 153). Le résultat le plus dérangeant était l'absence d'apprentissage des couche supérieures.

À partir des connaissances de la littérature du 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 faudrait recommencer le développement avec un système de gestion des données maîtrisé et changer le processus de développement du modèle.

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

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

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

7.7.3 Suspension du projet et lancement 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 de Bull (voir chapitre 10, page 55), qui ont été influencés par nos conclusions.

La nécessité de recommencer le projet GMSNN, combinée à l'opportunité de mettre les conclusions et les compétences acquises en pratique dans un projet à grande échelle, nous ont conduit a interrompre le 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.

8 Conclusions sur le projet GMSNN

8.1 Retour sur le travail effectué

Ce projet nous a permis de mettre en œuvre une architecture innovante de réseaux de neurones, à partir du squelette d'un modèle état de l'art. Nous avons pu élaborer un prototype selon 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 d'une durée allant 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'ait pas atteint les performances espérées, le modèle produit est robuste, rapide, et peu volumineux. De plus, l'algorithme présenté sous-section 7.5.6 (page 41) a démontré une faiblesse majeure de l'architecture GMSNN. Enfin, les problèmes rencontrés dans ce projet ont permis 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 permis de cerner les principaux écueils de la réalisation d'un modèle de la langue multi-échelle, et a ainsi rendu possible un meilleur déroulement du projet suivant.

8.2 Apport personnel du projet

La réalisation de ce projet nous a permis d'approfondir substantiellement nos connaissances en apprentissage profond, et de nous habituer aux problématiques de la création et de l'utilisation de réseaux de neurones.

8.3 Discussion et perspectives

8.3.1 Possibilités d'exploration de l'architecture

Si le projet GMSNN devait être relancé, l'expérience acquise durant le stage permettrait de présenter un éventail de pistes de recherche, à évaluer bien-sûr à la lumière de la littérature récente.

Par exemple, nous aurions pu optimiser le taux d'apprentissage, comme dans le projet PAPUD (voir section 13.7, page 65).

De très nombreuses autres possibilités s'offrent, notamment :
— 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].

8.3.2 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 présentée.

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 différents, comme à un travail plus poussé et plus expert sur le sujet.

Deuxième partie

Projet PAPUD

Profiling and Analysis Platform Using Deep Learning

9 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 10 (page 55).

9.1 Équipe de projet

Les personnes avec lesquelles nous avons travaillé 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 de la même équipe nous a rejoint vers la fin du stage, Mme Christine Fay-Varnier.

9.2 EUREKA et ITEA3

« EUREKA est une initiative européenne, intergouvernementale, destinée à renforcer la compétitivité de l'industrie européenne. » [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 à 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 de 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*, littéralement « plateforme de profilage et d'analyse utilisant l'apprentissage profond ») 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 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

D'après le site de ITEA3 [28]

Table 9.1 – Informations générales sur le projet PAPUD

9.4 Bull

9.4.1 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.

9.4.2 Secteurs d'activité

Les activités principales de la filiale Bull sont [29] :
- le matériel informatique et logiciel professionnel de haute sécurité ;
- le matériel informatique et logiciel pour l'Armée et la Défense, y compris le matériel de navigation maritime et terrestre ;
- les serveurs de calcul et de stockage, les centres de calcul (*data-centers* en anglais, 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 grandes entreprises et aux administrations publiques.

Pour en savoir plus

Des informations plus détaillées sur le projet PAPUD sont disponible sur la page du projet sur le site internet d'ITEA3 [28].

10 Réseau de neurones artificiels pour la prédiction de pannes

10.1 Contexte

Nous avons vu que, parmi les secteurs d'activité de Bull, l'utilisation de serveurs et autres systèmes de traitement de gros volumes de données sont très présents.

Ces machines 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 gérer.

Les données disponibles pour remplir cette tâche sont des fichiers de journalisation (*log files* en anglais) de très grande taille (ils sont décrits en détail dans le chapitre 11, page 59). Ils contiennent de nombreuses informations sur les évènements se déroulant dans les machines.

10.2 Solution

Le cas d'utilisation Bull du projet PAPUD est destiné à répondre à cette problématique, en fournissant un système détectant les signes de panne dans les fichiers de journalisation.

Pour cela, il a été décidé de modéliser le comportement normal (sans panne) reflété dans 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 seule précédente.

Ainsi, le plan général des opérations est composé de 2 parties.

1. Initialement, 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 de distance.

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. On y préférera un réseau de neurones artificiels basique.

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 tâche 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 le modèle simple et l'optimiser au mieux, nos objectifs étaient la réalisation d'un prototype du modèle, et la mise en place des 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 (page 61).

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

10.4 Organisation du travail

Contrairement au projet GMSNN, d'autres membres participaient à cette équipe de projet (voir section 9.1, page 51). Étant, avec le maître de stage, les seuls 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 de l'équipe de projet informés de l'avancement et des conclusions de notre travail. Plus exactement, nous avons continué de rédiger des rapports d'avancement et nous avons présenté la progression du travail au cours de réunions hebdomadaires.

Les rapports de ce projet sont également disponibles en annexe (voir l'annexe B, page 161). Le rapport d'une des réunion est disponible pour l'exemple à l'annexe B.6 (page 174).

10.4.1 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 objectifs intermédiaires étaient fixés.

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

Cependant, il a été possible de définir les délivrables suivants, classés par priorité décroissante :

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.

De plus, si du temps supplémentaire était disponible, il serait dédié à l'amélioration des performances du prototype.

Une extension de la durée du stage d'une semaine a été décidée, pour augmenter le temps dédié au projet, en considérant les disponibilités de chaque membre de l'équipe de projet.

La durée totale de travail sur le projet a donc été de 5 semaines.

10.4.2 Déroulement du projet

Tous les délivrables ont été fournis, et deux améliorations notables des performances ont été mises en place.

La répartition du travail du projet est représentée dans la figure 10.1 (page 57). Une case de la figure correspond à un jour de travail.

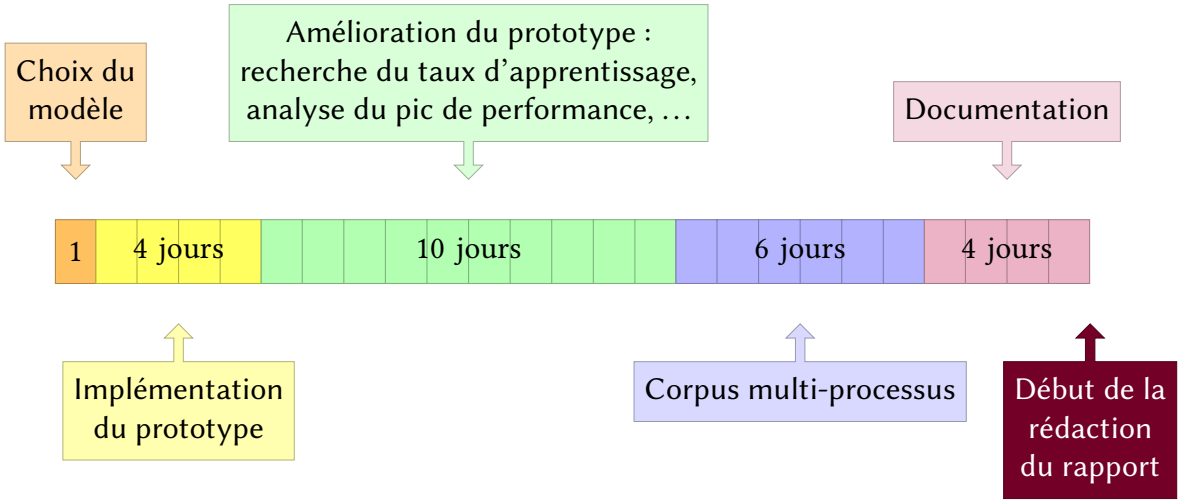


Figure 10.1 – Répartition du travail sur le projet PAPUD

10.5 Outils

10.5.1 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 membres de l'équipe de projet ont accès à l'ensemble de ces données, qui ont été mises à jours tout au long du projet.

10.5.2 Python, PyTorch et Grid5000

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

— la première partie du projet s'est déroulée avec ces outils, nous étions donc habitués à leur utilisation et à leurs spécificités ;
— la base de code accumulée jusque là reposait sur ces outils, et pouvait être réutilisée sans trop d'efforts ;
— il était possible de basculer ultérieurement vers une autre librairie que PyTorch, cette dernière possédant une fonctionnalité de conversion d'un modèle vers les autres principales librairies d'apprentissage profond.

A Rapports d'avancement du projet GMSNN

A.1 Informations sur les rapports contenus dans la présente annexe

Les sections suivantes contiennent les rapports intermédiaires fournis à notre maître de stage au cours du projet GMSNN.

A.1.1 Format d'origine des rapports

Le langage Markdown, plus spécifiquement dans le dialecte nommé Gitlab Flavoured Markdown, fournit une syntaxe facile à lire et à écrire.Il permet la rédaction de documents agrémentés entre autres d'images, de formules, de tableaux et de fragments de codes. Enfin, l'affichage du Gitlab Flavoured Markdown est supporté par Gitlab.

Ces particularités en font un langage de premier choix pour l'écriture de rapports destinés à être lus au format informatique directement sur Gitlab.

A.1.2 Transcription des rapports

L'intégration des rapports intermédiaires dans ce rapport à nécessité l'adaptation du contenu en Gitlab Flavoured Markdown au format papier.

Certains éléments n'ont pas pu être transcrit tels-quels, en particulier les liens, et les tableaux et images de grand taille.

A.1.3 Contenu et langue des rapports

Le contenu des rapports n'a été ni modifié ni corrigé, et est livré en anglais tel qu'écrit à l'origine.

L'anglais à été choisi comme langue de rédaction des rapports pour maintenir la cohérence avec le code, écrit et documenté en anglais lui aussi, et avec la littérature, principalement rédigée en anglais. Ce choix évite aussi d'alourdir le contenu déjà complexe des documents avec des traductions maladroites de termes techniques.

B Rapports d'avancement du projet PAPUD

B.1 Informations sur les documents contenus dans la présente annexe

Les sections suivantes contiennent les rapports intermédiaires fournis à notre maître de stage et au autres membres du projet au cours du projet PAPUD.

B.1.1 Format d'origine, transcription et contenu des rapports

Pour les mêmes raisons que pour l'annexe précédente (décrite dans la section A.1), les documents présentés dans cette annexe ont été rédigés en anglais, au format Gitlab Flavoured Markdown ; les versions présentées ici sont des transcriptions aussi fidèles que possible de ces documents.