

Stage de recherche

Réseaux de neurones multi-échelles
pour la modélisation de la langue
à partir de grands volumes de données

Esteban Marquer

Année 2017–2018

Projet réalisé pour l'équipe SYNALP du laboratoire LORIA

Maître de stage : Christophe Cerisara

Parrain universitaire : Jeanine Souquière

Stage de recherche

Réseaux de neurones multi-échelles
pour la modélisation de la langue
à partir de grands volumes de données

Esteban Marquer

Année 2017–2018

Projet réalisé pour l'équipe SYNALP du laboratoire LORIA

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

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

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



Encadrant : Christophe Cerisara

Remerciements

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

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

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

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

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

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

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

Avant-propos

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

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

Par ailleurs, de nombreux termes techniques sont utilisés. Bien qu'habituellement rencontrés sous leur forme anglaise dans le domaine, ces termes ont été traduits en français, à l'exceptions de certains sigles utilisés tels quels dans la littérature française.

Tous les termes techniques seront donc introduits en français, accompagnés d'une explication, de la traduction anglaise et du sigle anglais si nécessaire. De plus, une partie du rapport est dédiée à l'explication des termes et concepts utilisés, et un glossaire est présent en fin d'ouvrage. Enfin, quelques notes de bas de page fournissent des précisions ponctuelles sur certains concepts utilisés. Elles sont notées en exposant.

Les références aux sources sont marquées entre crochets, et numérotées par ordre d'utilisation.

Ce rapport a été réalisé avec L^AT_EX et les diagrammes présentés ont été produits par nos soins avec l'outil TikZ, ce qui explique l'absence de source pour les figures.

Conformément aux consignes données, les rapports intermédiaires ont été inclus dans les annexes du rapport.

Pour la rédaction de ce rapport, nous avons choisi d'utiliser la première personne du pluriel, afin de maintenir autant que possible neutralité et recul par rapport au éléments rapportés. Cependant, les conclusions personnelles du rapport ne nécessitant pas cette neutralité, elles ont été rédigées à la première personne du singulier.

Table des matières

Remerciements	3
Avant-propos	5
Listes des tableaux, des figures et des fragments de code	9
1 Introduction	13
2 Terminologie et concepts fondamentaux	15
I Projet GMSNN	19
3 Présentation du laboratoire et de l'équipe	21
4 Architecture innovante de réseaux de neurones pour un modèle du langage	25
5 Données disponibles	29
6 Description de l'architecture proposée	31
7 Réalisation	35
8 Conclusions sur le projet GMSNN	51
II Projet PAPUD	53
9 Présentation d'ITEA, du projet PAPUD, et de BULL	55
10 Réseau de neurones artificiels pour la prédition de pannes	59
11 Données disponibles	63
12 Description du modèle à réaliser	65
13 Réalisation	67
14 Conclusions sur le projet PAPUD	71
15 Bilan du stage	73
16 Bibliographie / Webographie	75
Glossaire	81
Annexes	83

Listes des tableaux, des figures et des fragments de code

Liste des tableaux

9.1	Informations générales sur le projet PAPUD	56
-----	--	----

Liste des illustrations

3.1	Organigramme des départements du LORIA, et des équipes du Département 4 . . .	22
4.1	Répartition prévue du travail	27
4.2	Répartition réalisée du travail	28
6.1	Principe de transmission de l'information d'une échelle à la suivante	32
6.2	Différentes échelles, et les niveaux d'abstraction attendus	33
7.1	Architecture du modèle ré-implémenté	36
7.2	Architecture du modèle GMSNN	37
7.3	Stratégies d'agrégation	41
7.4	Performances comparées des stratégies additive et par concaténation	41
10.1	Répartition du travail sur le projet PAPUD	61
12.1	Architecture du modèle du projet PAPUD	66

Liste des fragments de code

5.1	Extrait des premières lignes du fichier enwik8	29
7.1	Exemple 1 : une suite de caractères difficilement compréhensibles	47
7.2	Exemple 2 : des termes balisés comme dans le corpus d'origine, les crochets ouverts sont refermés	47
7.3	Exemple 3 : des termes balisés comme dans l'exemple 2, et une autre suite de caractères	48
11.1	Exemple de ligne extraite d'un des fichiers de journalisation	63

Acronymes

GMSNN	réseau de neurones récurrents multi-échelles croissant (<i>Growing Multi-Scale Recurrent Neural Network</i> en anglais)
GPU	processeur graphique (<i>Graphical Processing Unit</i> en anglais, GPU)
LSTM	réseau récurrent à mémoire à court et long terme (<i>Long Short Term Memory</i> en anglais)
RNN	réseau de neurones récurrents (<i>Recurrent Neural Network</i> en anglais)
TAL	traitement automatique des langues

1 Introduction

1.1 Contexte et enjeux du stage

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

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

Un des domaines exploitant les performances de ces nouveaux outils est le traitement automatique des langues (TAL, *Natural Language Processing* en anglais). En particulier, nous nous intéresserons dans ce rapport à la création de modèles de la langue (*Language Models* en anglais).

Ainsi, l'axe principal de ce rapport est l'application des méthodes de l'apprentissage profond sur de grands volumes de données, par la réalisation de modèles de la langue. Cela implique des problématiques relativement classiques en développement de réseau de neurones artificiels : le choix de l'architecture du réseau, de l'algorithme d'entraînement, mais aussi des questions plus pragmatiques d'optimisation liées au volume de données.

1.2 Objectifs du stage

Deux objectifs se sont succédé durant le stage.

L'objectif initial du stage était d'explorer une idée d'architecture innovante de réseau de neurones artificiels, imaginée par Mr. Cerisara, le maître de stage. Il s'agit du projet GMSNN (réseau de neurones récurrents multi-échelles croissant, *Growing Multi-Scale Recurrent Neural Network* en anglais, voir chapitre 4, page 25).

Cependant, à l'issue du deuxième mois du stage, nous avons changé d'objectif.

Le nouvel objectif a été la réalisation d'un réseau de neurones artificiels et des outils nécessaires à son utilisation, en mettant à profit les connaissances acquises durant la première partie du stage. Cette réalisation doit servir de base technique pour une partie du projet PAPUD (*Profiling and Analysis Platform Using Deep Learning*, voir chapitre 10, page 59).

1.3 Plan du rapport

Nous avons présenté à la fois le contexte, les enjeux, et les objectifs généraux du stage.

Tout d'abord, nous définirons les termes et concepts principaux utilisés dans ce rapport.

La complexité du stage est qu'il est composé de deux projets, le projet GMSNN et le projet PAPUD, l'un étant la continuation de l'autre. En effet, les conclusions tirées du premier projet ont servi de base pour le second. C'est pourquoi, pour simplifier la lecture du présent rapport, chaque projet sera traité suivant le même plan.

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

Enfin, nous ferons un bilan de l'ensemble du travail réalisé pour en tirer les apports principaux.

2 Terminologie et concepts fondamentaux

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

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

Les termes présentés ici sont répertoriés dans le glossaire, mais aucun rappel de leur présence dans celui-ci n'est donné au long du texte.

2.1 Apprentissage automatique, modèle, entraînement et données

2.1.1 Apprentissage automatique

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

2.1.2 Modèle

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

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

2.1.3 Entraînement du modèle

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

2.1.4 Données d'entraînement

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

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

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

2.2 Apprentissage profond et réseaux de neurones

2.2.1 Apprentissage profond

L'apprentissage profond représente un ensemble de techniques d'apprentissage automatique consacrées aux réseaux de neurones.

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

2.2.2 Réseau de neurones artificiels

Un réseau de neurones artificiels ou plus simplement réseau de neurones (*Neural Network* en anglais) est un modèle mathématique composé d'éléments interconnectés nommés neurones formels, par analogie lointaine avec le fonctionnement des neurones biologiques.

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

2.2.3 Architecture et modules

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

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

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

2.2.4 Paramètre

Un paramètre pour un module ou un modèle est une valeur qui varie au cours de l’entraînement. Généralement, plus un modèle possède de paramètres, plus son entraînement consomme de ressources mais plus la qualité de l’apprentissage est élevée.

2.2.5 Principales architectures de réseaux de neurones

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

- Le réseau de neurones artificiels intégralement connecté ou module linéaire est un des plus simples. Il est emblématique des réseaux de neurones.
- Le réseau de neurones récurrents (*Recurrent Neural Network* en anglais, RNN) est une architecture de réseaux de neurones particulièrement adaptée au traitement de séquences. Le RNN traite des éléments les uns après les autres, et stocke dans une « mémoire » des informations au fur-et-à-mesure. Il utilise les informations stockées pour améliorer la façon dont il traite les éléments suivants.
- Le réseau récurrent à mémoire à court et long terme (*Long Short Term Memory* en anglais, LSTM) est un RNN particulier. Équipé d’une mémoire supplémentaire, il plus puissant mais plus coûteux à entraîner qu’un RNN classique.

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

2.3.1 Traitement automatique des langues (TAL)

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

2.3.2 Modèle de la langue

Un modèle de la langue (*Language Model* en anglais) est une « distribution de probabilité sur une séquence de mots [ou de caractères] » (d’après Wikipédia [2]) utilisée pour estimer la probabilité d’apparition du prochain mot ou caractère.

Autrement dit, c’est une représentation servant à prédire le mot suivant à partir des mots précédents (ou le caractère suivant à partir des caractères précédents).

2.3.3 Contexte et dépendances

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

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

Parmi les informations contenues dans le contexte d'un mot, on peut trouver aussi bien le sens des mots environnants que la structure syntaxique de la phrase, ou encore des informations plus générales comme le fait que les chats sont des mammifères.

On considère que, plus on a d'informations contextuelles, plus le modèle de la langue est précis. Par exemple, si on nous dit « un chat », il sera plus difficile de prédire la couleur du chat que si on nous dit « un chat de couleur sombre ».

2.4 Performance et mesure

Le dernier concept important présenté dans ce chapitre est celui de performance des modèles produits.

La performance d'un modèle est évaluée par trois composantes :

- la qualité maximale des résultats produits ; dans le cas d'un modèle de la langue, il s'agit de la qualité de la prédiction ;
- le temps d'entraînement nécessaire pour atteindre cette qualité (nombre d'époques, durée, etc.) ;
- la consommation de ressources nécessaire pour atteindre cette qualité (mémoire, puissance de calcul, ...).

Afin d'évaluer la performance, des mesures ont été définies :

- pour mesurer la qualité du résultat, on utilise l'écart entre le résultat produit et le résultat attendu ; une mesure définie dans la littérature et utilisée dans les annexes est le BPC¹ ;
- pour mesurer le temps d'entraînement, on mesure le nombre d'époque et le temps nécessaire par époque, la durée totale en heures, etc. ;
- pour mesurer la consommation de ressources, on évalue l'espace mémoire occupé (en MiB ou GiB), la puissance de calcul utilisée (pourcentage de la puissance disponible), etc.

Un modèle optimal serait un modèle qui atteint d'excellents résultats (l'écart entre ce qui est attendu et ce qui est produit le plus faible possible), le plus rapidement possible, en consommant le moins de ressources possible.

1. Le BPC (*Bits Per Character* en anglais) [3] est originellement une mesure de la qualité de compression de texte, mais plusieurs papiers ont détourné cette mesure en s'en servant d'estimation de la qualité d'un modèle de la langue au niveau du caractère. Dans cet usage, le BPC est assimilable à une mesure de la précision des résultats du modèle de la langue. Plus le BPC est proche de 0 plus la qualité du modèle de la langue est élevée.

Première partie

Projet GMSNN

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

3 Présentation du laboratoire et de l'équipe

3.1 Généralités

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

3.2 Le LORIA

Le LORIA « est une Unité Mixte de Recherche (UMR 7503), commune à plusieurs établissements : le Centre National de la Recherche Scientifique (CNRS), l'Université de Lorraine (UL) et l'Institut National de Recherche en Informatique et en Automatique (INRIA). » [4]. Depuis sa création en 1997, le LORIA se concentre sur les sciences informatiques, que ce soit par la recherche fondamentale ou appliquée.

3.2.1 Structure administrative du LORIA

Le LORIA est dirigé par quatre instances [5] :

- **l'équipe de direction** : elle est composée du directeur du laboratoire, d'un directeur adjoint, de la responsable administrative, et de l'assistante de direction ; elle assiste le directeur dans la prise de décisions et leur mise en œuvre ;
- **le conseil scientifique** : il est composé du directeur, des deux directeurs adjoints et des scientifiques responsables des cinq départements du laboratoire ; il assiste le directeur dans la prise de décisions et leur mise en œuvre ;
- **le conseil de laboratoire** : il est composé de membres élus pour 4 ans et de membres nommés ; il émet des avis et conseillent le directeur sur toutes les questions concernant le LORIA ;
- **l'Assemblée des Responsables des Équipes (AREQ)** : elle est composée des scientifiques responsables des 28 équipes du laboratoire et se réunit tous les deux mois.

3.2.2 Recherche au sein du LORIA

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

La structure générale du LORIA en départements et, plus en détail, du Département 4 est représentée sur l'organigramme de la figure 3.1 (page 22). Les thématiques générales de chaque département, ainsi que les thèmes de recherche des équipes du Département 4, y sont présentés brièvement. Un organigramme complet du LORIA est disponible sur le site internet du laboratoire [6].

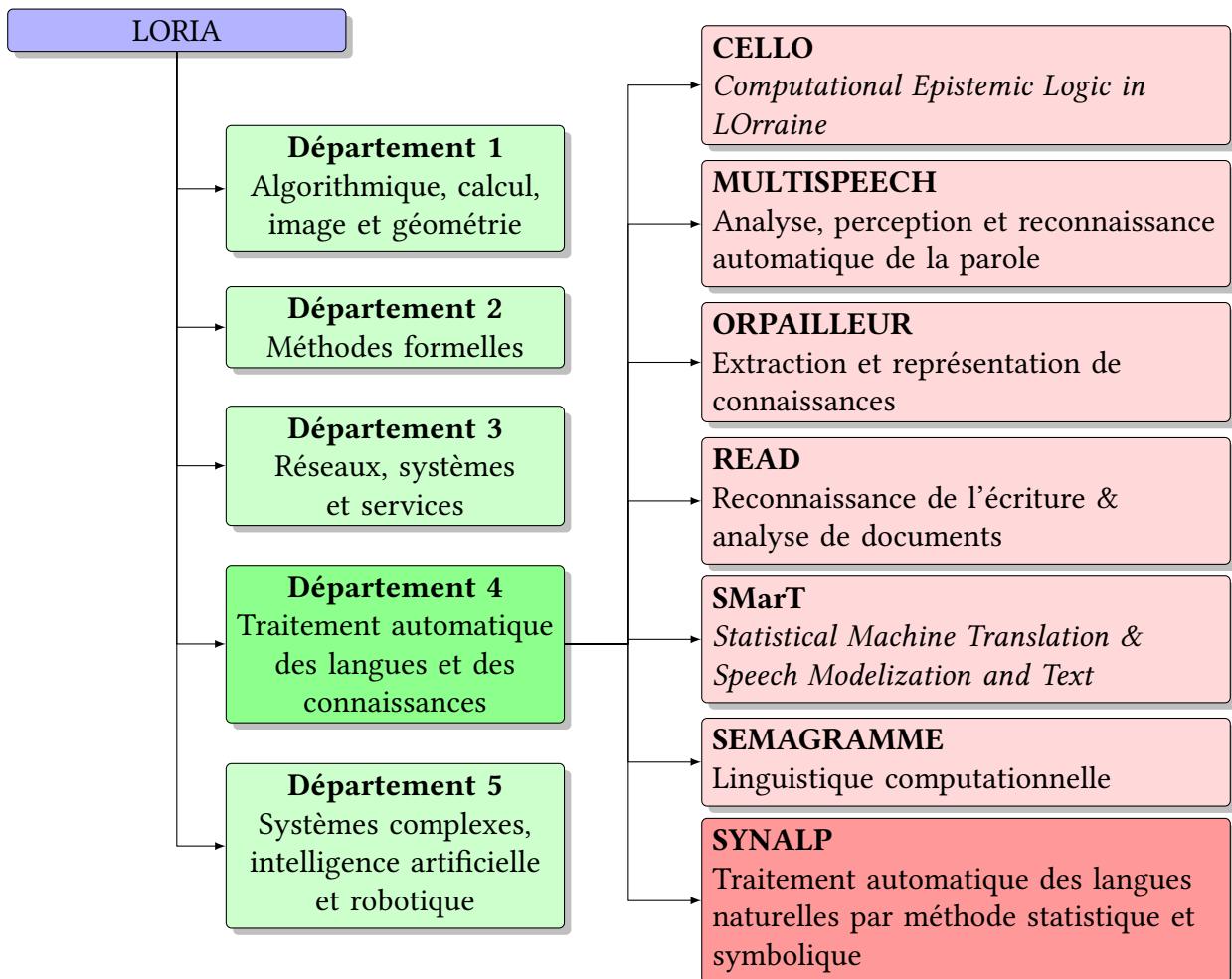


FIGURE 3.1 – Organigramme des départements du LORIA, et des équipes du Département 4

3.3 Équipe SYNALP

L'équipe SYNALP (*SYmbolic and statistical NAtural Language Processing*) est une équipe de recherche affiliée à la fois au CNRS et à l'Université de Lorraine. Elle fait partie, avec 6 autres équipes, du Département 4, dédié au traitement automatique des langues (TAL) et des connaissances.

3.3.1 Membres

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

3.3.2 Thématiques de recherche

La recherche au sein de l'équipe SYNALP se concentre sur les approches hybrides, symboliques et statistiques du TAL, ainsi que sur les applications de ces approches.

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

Ce stage s'inscrit en particulier dans la réalisation de modèles de la langue, et l'élaboration d'outils et ressources utilisés en TAL. Nous verrons en détail pourquoi dans le chapitre 4 (page 25).

3.4 Pour en savoir plus

Des informations plus détaillées sur le LORIA sont disponibles sur le site internet du laboratoire [4]. Par ailleurs, la liste complète des membres de l'équipe, ainsi que des informations plus détaillées sont disponible sur le site internet de l'équipe SYNALP (en anglais) [7].

4 Architecture innovante de réseaux 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 [8, 9, 10].

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

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

4.1.1 Manque d’utilisation des gros volumes de données

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

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.

Pour avoir un ordre d’idée, on peut considérer qu’un RNN basique conserve en mémoire des informations provenant des 20 dernières entrées; d’autres architectures de RNN peuvent se rappeler d’informations vieilles d’une centaine d’entrées; et un LSTM dépasse difficilement les 200 entrées.

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

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

4.2 Solution proposée

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

Cette architecture, nommée GMSNN (voir chapitre 6, page 31), a donné son nom au projet qui l'utilise.

4.3 Projet GMSNN

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

La mise en œuvre devait se réaliser à partir d'une base de code sur laquelle le maître de stage avait commencé à travailler (plus de détails sont disponibles dans la sous-section 7.2.1, page 35).

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

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

4.4 Organisation du travail

Cette tâche a été menée à bien par un travail individuel.

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

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

4.4.1 Organisation initiale du travail

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

La première semaine était dédiée à 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'art¹ contenue dans la base de code devaient être intégrées durant les 4 dernières semaines .

La figure 4.1 (page 27) représente cette répartition prévue du travail. Une case correspond à une semaine de travail.

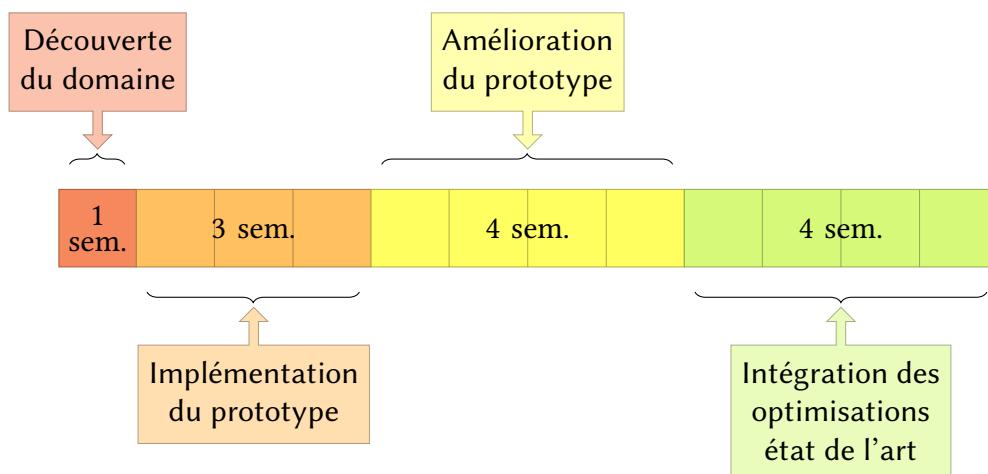


FIGURE 4.1 – Répartition prévue du travail

4.4.2 Déroulement réel du projet

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

Cependant, comme décrit section 7.7 (page 49), nous avons décidé d'interrompre ce projet pour nous consacrer au projet PAPUD.

La figure 4.2 (Figure 4.2) représente la répartition réalisée du travail. Une case de la figure correspond à une semaine de travail.

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

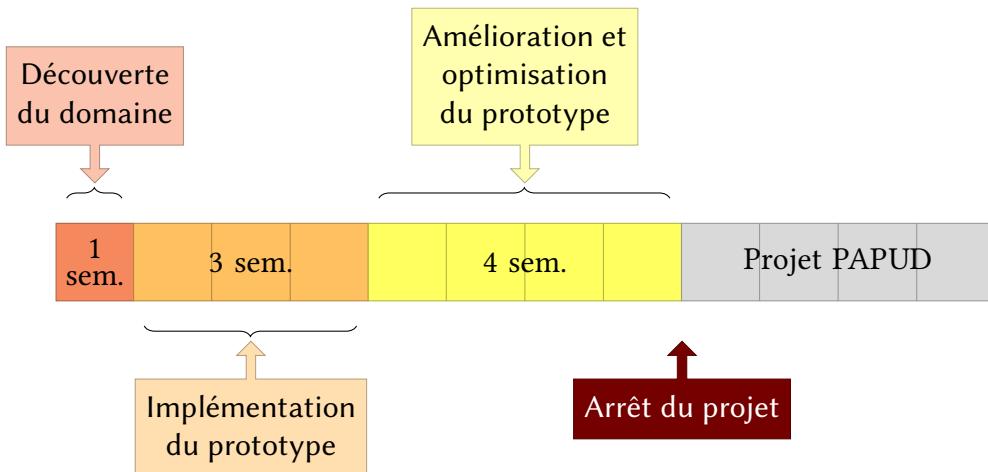


FIGURE 4.2 – Répartition réalisée du travail

4.5 Outils

4.5.1 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.

4.5.2 Langage et librairie

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

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

4.5.3 Grid5000 et les machines distantes

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

Ces machines font partie du réseau Grid5000. « Grid’5000 est un banc d’essai à grande échelle et polyvalent pour la recherche expérimentale dans tous les domaines de l’informatique, avec un accent sur l’informatique parallèle et distribuée, y compris Cloud, HPC et Big Data. » [11]

Un des avantages de cet outil est la présence de machines spécialisées équipées de GPU², qui sont celles que nous avons utilisées.

2. Un processeur graphique (*Graphical Processing Unit* en anglais, GPU) est un composant d’ordinateur spécialisé, qui montre d’excellentes performances dans les calculs impliquant des matrices (ex. : images). Cette propriété s’applique aussi sur les tenseurs. L’utilisation de GPU pour l’entraînement des réseaux de neurones est une pratique fréquente en apprentissage profond, car elle permet d’accélérer les calculs effectués.

5 Données disponibles

Les données d'entraînement utilisées proviennent du Wikipédia anglais. Elles sont tirées du fichier « enwik8 » utilisé pour le prix Hutter [12, 13]. Ce fichier est composé d'environ 100 000 000 caractères.

Ces données sont composées de texte balisé, structuré en paragraphes. Quelques fragments de XML¹ sont aussi présents, mais ils sont minoritaires dans les données. Il est donc peu probable de les retrouver dans les données apprises.

Deux versions alternatives du corpus ont été utilisées :

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

5.1 Extrait des données d'entraînement

Le Fragment de code 5.1 (page 29) est un extrait des données brutes avant le découpage en caractères. Il correspond à l'article Wikipédia sur l'anarchisme.

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

Fragment de code 5.1 – Extrait des premières lignes du fichier enwik8

1. XML (*eXtensible Markup Language* en anglais), « est un langage informatique qui sert à enregistrer des données textuelles. [...] Ce langage , [...] similaire à l'HTML de par son système de balisage, permet de faciliter l'échange d'informations sur l'internet », d'après le glossaire sur *infowebmaster* [14]. Il s'agit du format sous lequel sont stockés les articles Wikipédia.

5.2 Prétraitement des données

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

Comme mentionné dans la sous-section 7.6.2 (page 48), un défaut dans le prétraitement a mené à la disparition des espaces du texte.

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

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

6 Description de l'architecture proposée

6.1 Propriétés du modèle

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

Elle est caractérisée par trois propriétés majeures :

- la structure récurrente ;
- l'utilisation de plusieurs échelles ;
- la croissance du modèle.

Comme annoncé dans la section 4.2 (page 26), nous avons nommé cette architecture réseau de neurones récurrents multi-échelles croissant (*Growing Multi-Scale Recurrent Neural Network* en anglais, GMSNN) en considérant ses principales caractéristiques.

6.1.1 Récurrence du modèle

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

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

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

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

6.1.2 Passer à l'échelle

Comme décrit dans la sous-section 4.1.2 (page 25), les RNN ont un problème inhérent de capacité mémoire, qui limite la distance des dépendances apprises par le modèle.

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

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

De plus, chaque échelle tire ses informations de l'échelle précédente. L'exemple suivant explique ce mécanisme de transfert de l'information, également illustré sur la figure 6.1 (page 32). Sur cette figure, la fréquence de transmission est notée n .

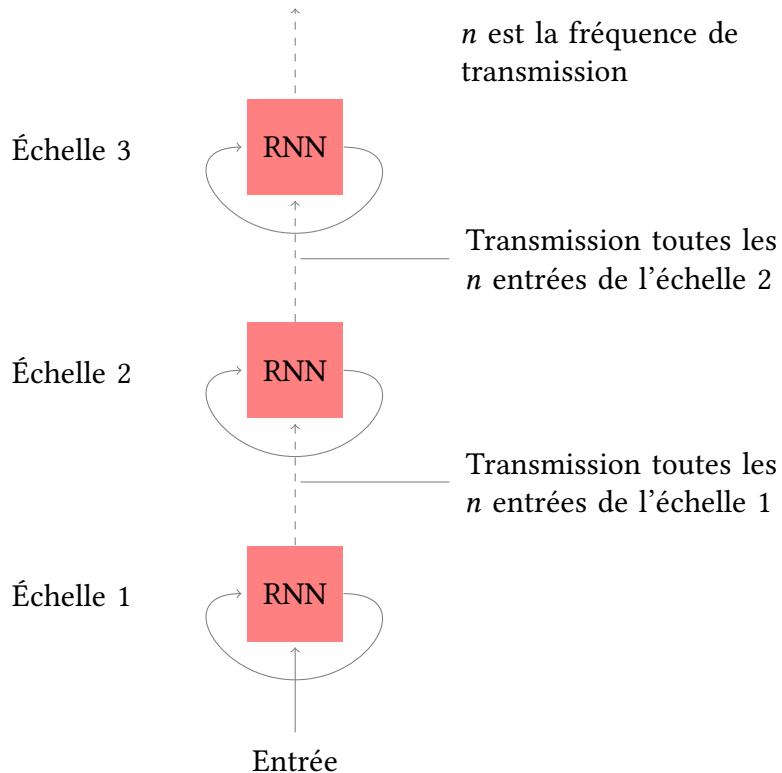


FIGURE 6.1 – Principe de transmission de l'information d'une échelle à la suivante

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

Plusieurs niveaux d'abstraction de l'information

Une autre caractéristique importante du GMSNN est qu'une échelle prend en entrée les informations abstraites par la couche précédente. En effet, l'information stockée dans la mémoire d'un RNN peut s'apparenter à une représentation, donc à une abstraction, des données.

Ainsi, on peut s'attendre à ce que chaque échelle ajoute un niveau d'abstraction supplémentaire au modèle, comme sur la figure 6.2 (page 33). Sur cette figure, chaque bloc vert correspond à une entrée pour l'échelle correspondante ; les blocs rouges en bas du diagramme correspondent aux caractères qui sont fournis en entrée au modèle. Ici la fréquence de transmission vaut 3.

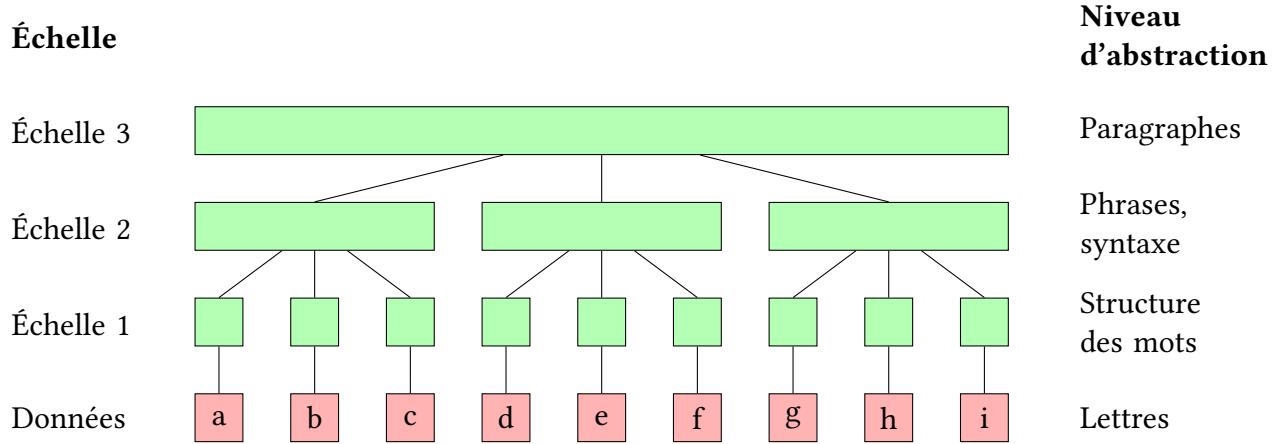


FIGURE 6.2 – Différentes échelles, et les niveaux d’abstraction attendus

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

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

En effet, comme expliqué dans la sous-section 6.1.2 (page 31), le nombre d’échelles est dépendant du nombre d’entrées totales.

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

Ainsi, au fur-et-à-mesure de l’entraînement, le modèle croît.

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

$$l = \lfloor \log_n i \rfloor + 1$$

Croissance potentiellement infinie du modèle

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

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

Comme décrit dans la sous-section 7.5.1 (page 40), l’utilisation de cette propriété a été abandonnée.

7 Réalisation

7.1 Recherche documentaire

La première partie du projet a été la recherche documentaire et la prise en main des outils. Ce travail a été effectué à partir des documents fournis par le maître de stage, de la documentation de PyTorch [15, 16, 17, 18] et de Grid5000 [19, 20], complétés par des 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 a été la prise en main de la base de code fournie. Elle contient une implémentation état de l'art d'un modèle de la langue au niveau du caractère, sur laquelle le maître de stage avait commencé à travailler. Le code d'origine provient du dépôt « awd-lstm-lm »[21], qui contenait plusieurs modèles de la langue états de l'art.

Au début du stage, la base de code contenait :

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

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

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

7.2.2 Modèle ré-implémenté simplifié

Le modèle simplifié prend en entrée des caractères, et produit des probabilités sur le prochain caractère. Le modèle est composé d'un module encodant les caractères, d'un RNN particulier (un LSTM) et d'un module produisant une distribution de probabilités sur les caractères connus. La figure 7.1, (page 36) représente cette architecture.

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

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

Le module produisant la distribution de probabilité est un module linéaire¹. Il transforme les informations produites par le réseau de neurones en probabilité qu'un des caractères soit le prochain caractère de la séquence.

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

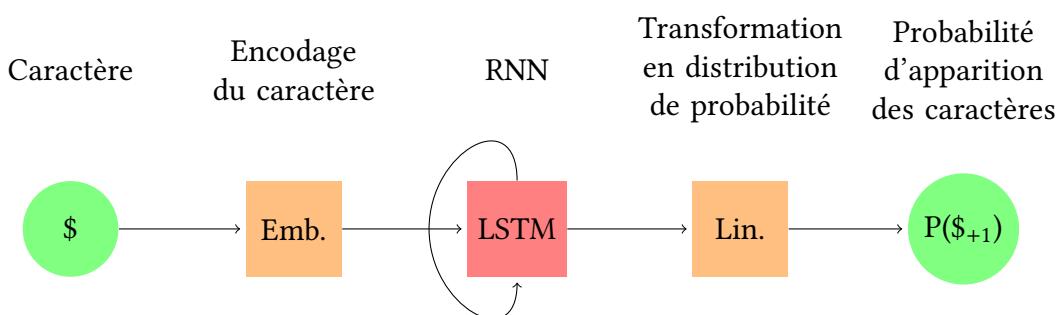


FIGURE 7.1 – Architecture du modèle ré-implémenté

1. « Module linéaire » est le nom donné aux modules composé d'un réseau de neurones artificiels intégralement connecté. Un réseau de neurones intégralement connecté 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 (voir la sous-section 2.2.5, page 17).

7.3 Implémentation du nouveau modèle

7.3.1 Travail effectué

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

L'architecture du GMSNN est celle du modèle ré-implémenté, mais le RNN y est remplacé par le module GMSNN (voir figure 7.2, page 37). C'est sur ce nouveau module GMSNN que le reste du travail au cours du projet GMSNN a été effectué.

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

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

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

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

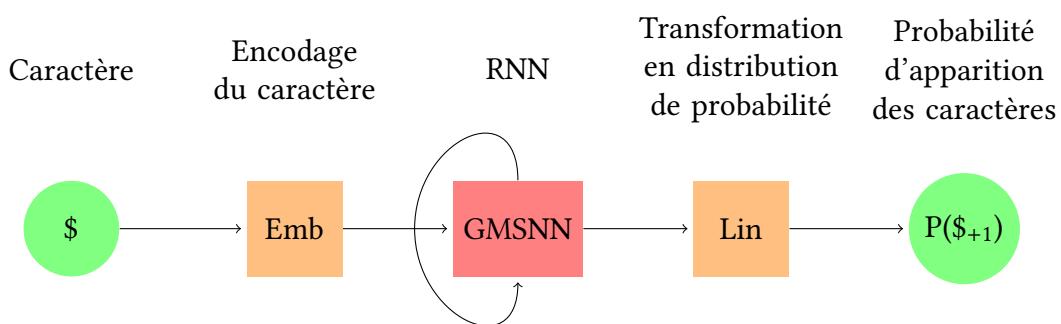


FIGURE 7.2 – Architecture du modèle GMSNN

7.3.2 Transmission de l'information

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

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

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

L'annexe A.11 (page 113) présente le rapport sur le prototype.

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 été testées, en particulier VisualIDL [22, 23].

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 stocker 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 41).

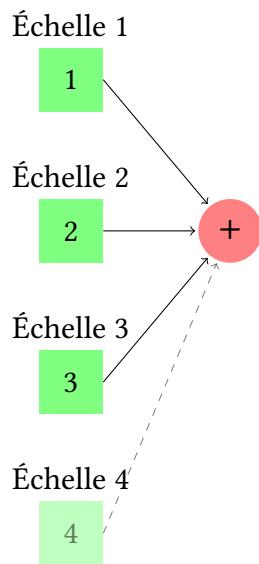
Après discussion, la stratégie d'agrégation a été changée en une concaténation des sorties.

Comme montré dans la figure 7.3b (page 41), 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.

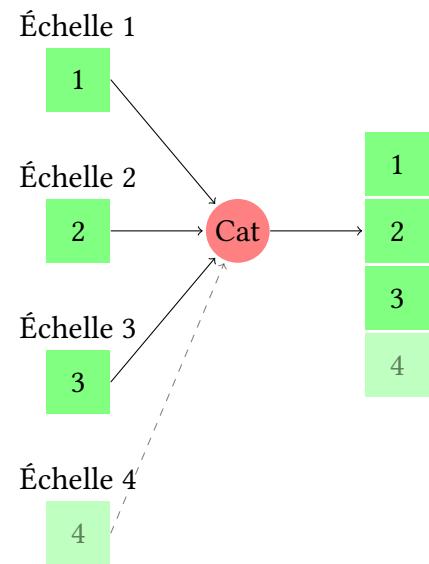
En contrepartie, la propriété de croissance à l'infini de l'architecture (décrise dans la sous-section 6.1.3, page 33) a été abandonnée, au profit d'un nombre maximal d'échelles défini à l'avance ou déterminé à l'aide d'une formule en fonction des données disponibles (décrise dans la sous-section 6.1.3, page 33).

La stratégie par concaténation est plus lente en terme de temps de calcul que la stratégie additive. Cependant, pour le même temps de calcul elle permet d'obtenir de meilleurs résultats. La figure 7.4 (page 41) 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 détails sur le choix de la stratégie d'agrégation sont disponibles dans l'annexe A.12 (page 117).



(a) Stratégie d'agrégation additive



(b) Stratégie d'agrégation par concaténation

FIGURE 7.3 – Stratégies d'agrégation

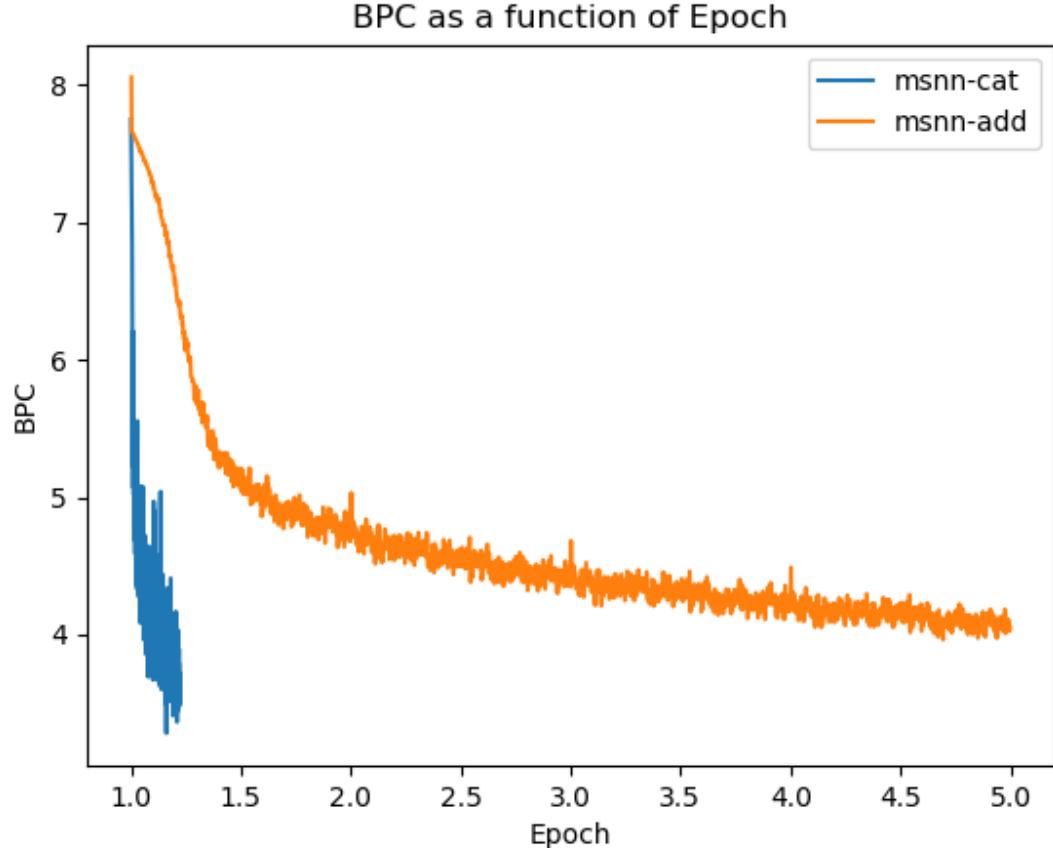


FIGURE 7.4 – Performances comparées des stratégies par concaténation (*msnn-cat*) et additive (*msnn-add*)

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 a été mis en œuvre.

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

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 44);
- l'augmentation du nombre de paramètres du modèle (voir sous-section 7.5.5, page 45);

Consommation accrue de mémoire et de temps de calcul

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

Cette consommation accrue a causé le plantage² 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 37) 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 88), a été d'estimer l'usage normal de la mémoire (sans fuite), et d'isoler les paramètres qui ont le plus d'impact sur la consommation de 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 paramètre 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 97) 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, résoudre les problèmes de fuites de mémoire et de 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 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* [24].

L'utilisation de cet algorithme est une optimisation répandue pour l'entraînement de réseaux de neurones [24]. 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 résultats 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 utilisée est similaire à celle décrite dans l'article [25]. 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 : cette dernière 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 aspects 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 99) 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 122), A.16 (page 138), A.17 (page 142) et A.18 (page 146).

L'annexe A.2 (page 88) 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 17), 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 la 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 42), 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 modifications 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 naïve³ d'un entraînement couche par couche appliquée 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, ce sont d'abord les échelles les plus proches des données doivent apprendre avant que les échelles supérieures puissent le faire à leur tour. Aussi, il semble inutile d'augmenter la charge de l'algorithme d'entraînement en entraînant des couches qui ne peuvent pas apprendre efficacement.

Le fonctionnement 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 104).

Performances

L'algorithme atteint l'objectif d'alléger la charge calculatoire. de fait, on obtient une réduction sensible du temps nécessaire pour l'entraînement.

3. Par « implémentation naïve », nous entendons que l'algorithme est simple, qu'il n'est pas abouti et perfectionné en regard de la littérature.

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 104).

Remise en cause de l'intérêt de l'architecture

L'algorithme développé revient, en tout cas sur le début de l'entraînement, à entraîner un modèle mono-échelle. En effet, en entraînant une seule couche seuls les paramètres correspondants sont mis à jour, ce qui revient à réduire le nombre de paramètres du modèle, et devrait avoir une influence négative sur la performance (Voir sous-section 7.5.5 pour l'impact du nombre de paramètres, page 45).

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 notamment 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 45) 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.

```
1 YeoMMDF| Ph# element at [[ Damous ]]  
that ure often use voir be expounder states and a number of his work for members other than  
novel was me the ce by element or from the last Preenance old War Instated by the  
Philosophy '' the Tayita ( ams methous people aiming shelebel obes in the satie the universalists scientis  
education of [[ Laking forts ]].
```

Fragment de code 7.1 – Exemple 1 : une suite de caractères difficilement compréhensibles

```
1 +EDrFuergCases , are in less such as the sthe alter plains .  
2 * In [[ Stefapes ]]  
3 * [[ AcademyAwards ===  
4  
5 ANASA) as LASCIIR under , and as .  
Matthe busip clears and president have a que fuels if the search from Awarer Lievol  
of any 30020 .  
6  
7 It ' ' [[ Anim ]]  
8 * [[ UnitedStates | raphicsiteDirection ]]  
9  
10 The plant - gain he its returned to a see the war in steast & quot ; One of the [  
e c tly would not by the Integrations capian land ] ( ora ' ' [[ schology ]]  
11 | published :
```

Fragment de code 7.2 – Exemple 2 : des termes balisés comme dans le corpus d'origine, les crochets ouverts sont refermés

```

1 60447 -toNewHarry } }
2 Thenmainst . Rand ' s intereststhe " ; toinpassingtheEarth ' ' s( ' ' [[ par ]].
3
4 : ' ' ' maimals , anackreloquedoutwidthofgrawwithluteframedapproyngtoundernverby [ [
5 hebesination ] ] of< ; / smalkan ,
6 instablishedacondorttodevelopedframesbeforestatedwinkingaroundsinrational
7 hicarefartoredonaftercanbeagainstthatgroupswouldnear ,
8 notwhatwasthatisstillastructionCenter , toDagnythat
9
10 On[[éAft ele ofAirej ]]
11 [[ cy : Alaska ]]
12 [[ no : Arni – Anchorage s ]]

```

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 47, ligne 8) ou [[cy:Alaska]] et [[no:Arni–Anchorage s]] de l'exemple 2 (Fragment de code 7.2, page 47, 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 similaires aux annotations présentes dans le code Wikipédia. On peut les voir dans le Fragment de code 5.1, page 29.

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

- la suite de mots statesandano numberofhisworkformemberso than novelwas methe 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 47, ligne 1);
- de même pour la suite de mots oldWarInstartedbythePhilosophy : old War In started by the Philosophy (Fragment de code 7.1, page 47, ligne 1);
- on trouve aussi des noms propres comme le nom de pays : UnitedStates (Fragment de code 7.2, page 47, ligne 8) et Alaska (Fragment de code 7.3, page 48, 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 151).

Le résultat le plus dérangeant est 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 récurrence.

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 a permis de définir les objectifs du projet ITEA3-PAPUD, cas d'utilisation de Bull⁴ (voir chapitre 10, page 59), qui ont été influencés par nos conclusions.

Devant 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, c'est d'un commun accord avec le maître de stage que nous avons décidé de consacrer la fin du stage au projet PAPUD.

4. L'expression « cas d'utilisation » est utilisé en gestion de projet pour désigner une partie d'un projet. Elle désigne ici la partie du projet PAPUD destinée à répondre aux besoin de l'entreprise Bull.

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 dans la sous-section 7.5.6 (page 45) 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-échelles, et a ainsi rendu possible un meilleur déroulement du projet suivant.

8.2 Apport personnel du projet

La réalisation de ce projet m'a permis d'approfondir substantiellement nos connaissances en apprentissage profond, et de m'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 69).

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 échelles ;
- le changement de l'architecture de pyramidale à parallèle, c'est à dire que chaque échelle serait indépendante, tel que proposé dans l'article [26].

8.3.2 Travaux similaires

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

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

Ceci nous permet de prendre du recul sur les décisions et choix pris pendant le projet, qui n'ont pu bénéficier d'un travail aussi poussé et expert que celui fourni par les chercheurs pour leur article.

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 de 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 le chapitre 10 (page 59).

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. » D'après Wikipédia [28], confirmé par les pages dédiées à EUREKA sur le site internet de la Commission Européenne [29] et celui du gouvernement français [30], ainsi que par le site internet de EUREKA [31].

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 projet 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.

Le cas d'utilisation de Bull du projet PAPUD est la partie du projet destinée à répondre aux besoins de l'entreprise Bull (décrits dans le chapitre 10 (page 59)).

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 [32]

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 [33] :

- 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 disponibles sur la page du projet sur le site internet d'ITEA3 [32].

10 Réseau de neurones artificiels pour la prédition de pannes

10.1 Contexte

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

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

Il serait donc très intéressant de mettre au point un système de prédition 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 63). Ils contiennent de nombreuses informations sur les évènements se déroulant dans les machines.

10.2 Solution

Le cas d'utilisation de 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 65).

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

10.4 Organisation du travail

Contrairement au projet GMSNN, ce projet s'est déroulé en équipe (voir section 9.1, page 55). Cependant, 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 159). Le rapport d'une des réunion est disponible pour l'exemple à l'annexe B.6 (page 172).

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 délivrables¹ 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 61). Une case de la figure correspond à un jour de travail.

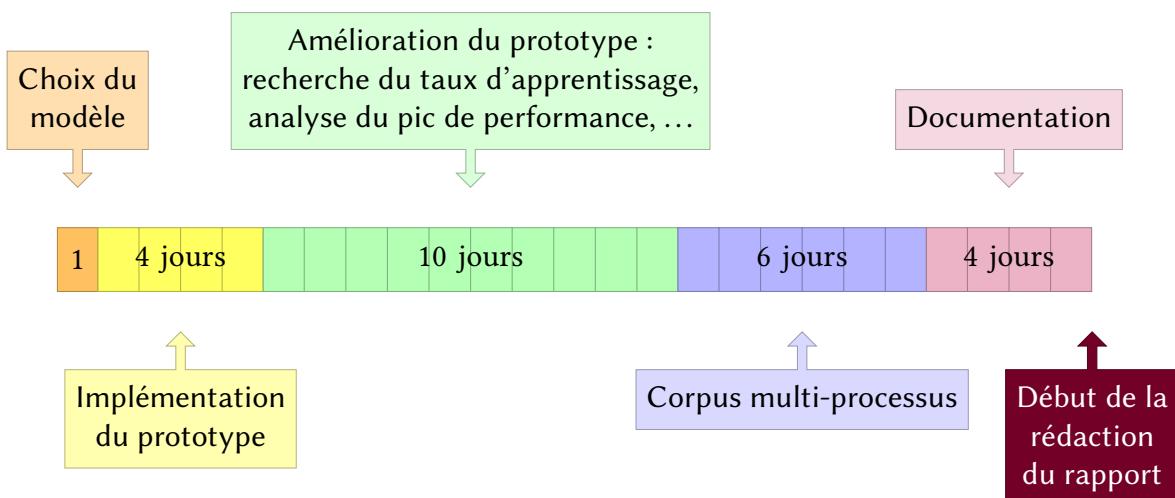


FIGURE 10.1 – Répartition du travail sur le projet PAPUD

1. Le terme « délivrable » est un anglicisme utilisé en gestion de projet pour désigner un « document, [ou une] démonstration, qui doit être délivrée durant un projet », d'après le dictionnaire Cordial [34].

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.

11 Données disponibles

Les données disponibles sont des fichiers de journalisation.

Ces fichiers sont composés de lignes de texte en anglais extrêmement structurées (comme on peut le voir sur le Fragment de code 11.1, page 63), générées par les programmes en cours d'exécution et qui donnent des informations sur les évènements qui surviennent sur la machine. Ce sont par exemple des messages d'information, tels que « le programme A a tenté de faire B », ou « l'utilisateur C a changé son mot de passe ».

Ces messages sont précédés d'informations contextuelles comme l'instant d'écriture du message et le programme d'où il provient.

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

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

11.1 Extrait des données d'entraînement

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

Dans le Fragment de code 11.1 (page 63), on a, de gauche à droite : le *timestamp*, la date, l'heure, l'utilisateur (OOOOOO), le processus (authpriv), le type de message (info), et le message.

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

Fragment de code 11.1 – Exemple de ligne extraite d'un des fichiers de journalisation

11.2 Prétraitemet des données

Le prétraitemet des données est consiste en :

- 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 remplacés 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 a été ajouté (voir section 13.6, page 69). De cette façon, 0x005f et 0x0007 sont remplacés par le caractère <hexX4>, tandis que 0000005f et 89abcdef sont remplacés par <hex8>. Cela permet de s'abstraire de la valeur du code sans perdre l'information liée à sa présence.

12 Description du modèle à réaliser

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

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

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

La sortie est un tableau à 2 dimensions, qui contient, pour chaque caractère prédit, une distribution de probabilité sur les caractères connus (répertoriés dans le dictionnaire, qui est décrit dans la section 11.2, page 64).

12.1 Utilisation d'un réseau de neurones artificiels basique

Comme présenté dans le chapitre 10 (page 59), un réseau de neurones artificiels basique, intégralement connecté, a été choisi pour le modèle.

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

12.2 Réduction de la taille de l'entrée

Comme écrit plus haut, l'entrée du modèle est une ligne de caractères. Chacun caractère, comme dans le modèle GMSNN, est transformé en un tenseur par un module d'*embedding* (voir la sous-section 7.2.2, page 36). Les tenseurs des caractères sont assemblés pour représenter la ligne.

Pour réduire la taille de cette représentation, on utilise une opération appelée « *max-pooling* », qui permet de produire un unique tenseur d'*embedding* par ligne.

Le *max-pooling* consiste à prendre, pour chaque coefficient du tenseur final, la valeur maximale des coefficients correspondants des tenseurs initiaux.

Par exemple :

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

12.3 Représentation de l'architecture

L'agencement des différents modules de l'architecture est représentée dans la Figure 12.1 (page 66).

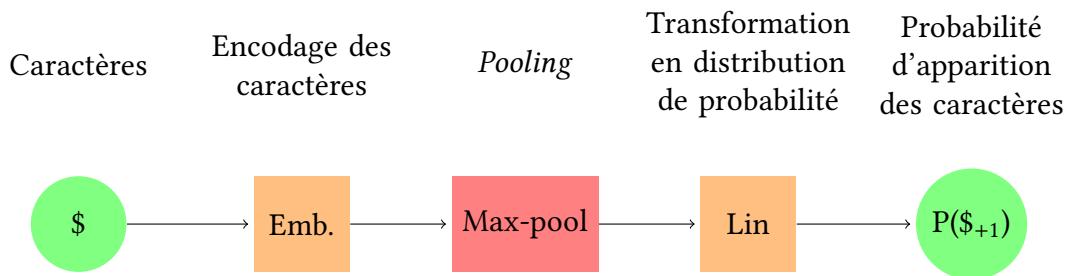


FIGURE 12.1 – Architecture du modèle du projet PAPUD

12.4 Lien avec l'état de l'art

Il a été récemment démontré que les modèles de la langue simples, basés sur des *embeddings* et des opérations de *pooling*, montrent des performances équivalentes voire supérieures à des modèles plus complexes et plus coûteux comme les RNN [35].

13 Réalisation

13.1 Définition du modèle

La toute première étape du projet a été de définir l'architecture à utiliser (décrite chapitre 12, page 65).

Nous avons d'abord élaboré le modèle au cours d'une série de réunions avec le maître de stage. Puis nous l'avons présenté à l'équipe de projet, qui l'a approuvé.

13.2 Implémentation du modèle et transfert des outils de base

L'étape suivante a été l'implémentation d'une version de base du modèle, ainsi que des outils nécessaire pour l'utiliser.

Ces outils sont :

- un outil de prétraitement et de manipulation de corpus ;
- un algorithme d'entraînement sur ces données, élaboré lors 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, page 39) ;
- un système de sauvegarde et d'interruption de l'entraînement, similaire à celui du projet précédent (voir sous-section 7.5.2, page 42)

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

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

Nous avons ensuite adapté le système de gestion de corpus aux grands volumes de données à traiter dans le futur.

Pour cela, un premier prototype a été réalisé. Conçu pour limiter la quantité de données en mémoire, il utilisait des mécanismes optimisés du langage Python pour la lecture de fichier, couplés à un principe de fonctionnement itératif simple.

1. Premièrement, on charge en mémoire une portion des données.
2. Ensuite, on applique le prétraitement sur ces données.
3. On transfère les données traitées à l'algorithme d'entraînement.
4. Enfin, on recommence à partir de l'étape 1 avec la portion suivante des données.

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

13.4 Entrainement par *batches* simultanés

La première optimisation du modèle a été l'intégration de l'algorithme d'entraînement par *batches* (voir sous-section 7.5.4, page 44).

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

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

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

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

Jusqu'à cette étape, les performances 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 performances du modèle.

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

13.5.1 Précision de base

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

Cette valeur représente la précision d'un modèle qui produirait toujours la même ligne, composée du caractère le plus fréquent. Elle sert de base de comparaison pour les performances du modèle. de déterminer si le modèle produit apprend réellement, et dans quelle mesure.

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

13.6 Analyse d'une variation anormale de performance dans l'apprentissage

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

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

Il a été décidé que ce phénomène devait être analysé avant de poursuivre les travaux.

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

Il a donc été décidé, comme décrit dans la sous-section 11.2.1 (page 64), d'intégrer au prétraitement des données la gestion de ces codes. Cette opération a été intégrée durant la réalisation de la nouvelle version du gestionnaire de données, décrite section 13.8 (page 70).

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

13.7 Optimisation du taux d'apprentissage

La troisième optimisation mise en place porte sur le réglage d'un des paramètres de l'algorithme d'apprentissage, est une valeur nommée « taux d'apprentissage ». Il s'agit d'une optimisation courante en apprentissage profond [36].

Ce paramètre permet de déterminer la vitesse d'apprentissage du modèle. Un mauvais réglage peut entraîner des conséquences catastrophiques sur le modèle, comme un apprentissage extrêmement lent, voire la divergence de l'apprentissage¹.

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

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

La nouvelle valeur définie a permis d'augmenter considérablement la vitesse d'amélioration de la qualité du modèle. En effet, il fallait précédemment plus de 400 époques pour atteindre la qualité maximale du modèle, contre moins de 100 époques avec le nouveau taux d'apprentissage.

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

1. On parle de divergence quand la performance du modèle empire au long de l'apprentissage. Ce terme est utilisé par opposition à la convergence souhaitée des performances vers le meilleur score possible.

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

La dernière optimisation est la transformation du système de gestion de corpus en une version plus puissante. Le nouvel outil créé a 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 du maître de stage, nous avons donné au nouvel outil la capacité d'utiliser plusieurs fichiers comme les fragments d'une unique source de données. Cette fonctionnalité a été optimisée quant au temps de calcul et à l'utilisation de mémoire.

13.8.2 Prétraitement modulaire

En outre, durant le déroulement du projet, nous avons remarqué que l'ajout ou la modification d'une étape de prétraitement était complexe avec l'ancien outil. Pour simplifier les modifications futures, 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 la réalisation de ces modules que le prétraitement mentionné section 13.6 (page 69) et sous-section 11.2.1 (page 64) a été intégré.

13.8.3 Processus multiples

Une autre amélioration, tirant profit de l'organisation modulaire du traitement, est l'utilisation de multiples processus en parallèle. Chacun d'entre eux est dédié à une étape du prétraitement. Ceci 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 dans sa version précédente.

13.8.4 Performances

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

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

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 objectifs a été atteinte.

Nous avons réalisé deux principales optimisations, en plus du prétraitement des codes hexadécimaux et du choix du nombre de *batches* :

- un outil d'optimisation du taux d'apprentissage ;
- un outil multi-fichiers multi-processus de gestion du corpus.

De plus, nous avons porté une attention particulière à la documentation du code. En effet, le travail effectué est la première phase de la réalisation du cas d'utilisation Bull du projet PAPUD. Nous avons donc laissé une base de code propre et intégralement documentée pour faciliter la poursuite du projet.

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

14.2 Apport personnel du projet

La réalisation de ce projet m'a permis d'appliquer l'ensemble des connaissances accumulées au cours du projet précédent.

J'ai ainsi utilisé mes compétences dans un projet de grande ampleur.

De plus, le travail avec une équipe de projet m'a permis d'avoir un aperçu du déroulement d'un projet de recherche, ce qui a été une expérience enrichissante.

Enfin, la documentation du code et des outils, en particulier pendant les derniers jours du projet qui y étaient dédiés, m'a permis de me perfectionner dans les techniques de documentation en Python. Par là j'entends surtout les techniques de typage du code [37, 38] et les pratiques de rédaction de *docstring* (le format sous lequel est rédigée la documentation en Python).

14.3 Discussion et perspectives

14.3.1 Poursuite 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 citer :

- 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, c'est à dire adapter le modèle à des dépendances plus longues (décrite section 10.2, page 59).

15 Bilan du stage

15.1 Bilan du travail effectué

Durant ce stage, nous avons travaillé sur deux projets différents, le projet GMSNN et le projet PAPUD. À posteriori, l'un apparaît comme une initiation aux techniques et technologies du apprentissage profond, ainsi qu'une exploration d'une piste de recherche; l'autre serait plutôt une mise en pratique des connaissances et compétences acquises.

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

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

Au final, les résultats et outils fourni pour le projet PAPUD sont le fruit de tout 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 de l'apprentissage profond, ainsi qu'une certaine maîtrise de PyTorch et de Grid5000.

Avant ce stage, nous avions déjà l'habitude du Python et de ses spécificités ; 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 modèle réputé parmi les plus complexes de l'apprentissage profond, le RNN.

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

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

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

15.3 Bilan personnel

Ce stage était une aubaine pour moi, qui cherchais à la fois un stage me 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 l'orientation de mes études.

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

Je tiens à souligner qu'il a été très motivant et agréable de voir mes avis considérés et mes idées bien reçues tout au long des projets, en particulier en considérant que je reste un novice dans le domaine de l'apprentissage profond.

Enfin, ce stage a été l'occasion de laisser libre court à mon penchant pour l'optimisation.

Cependant, je ne peut que déplorer le manque de retour sur la qualité du travail fourni, qui m'aurai permis d'orienter mes efforts au long du stage et de mes prochains travaux.

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

16 Bibliographie / Webographie

- [1] *Apprentissage automatique*. Wikipédia. Juin 2018. URL :
https://fr.wikipedia.org/wiki/Apprentissage_automatique (visité le 10/08/2018) (cf. p. 15).
- [2] *Language model*. Anglais. Wikipédia. Juil. 2018.
URL : https://en.wikipedia.org/wiki/Language_model (visité le 10/08/2018) (cf. p. 17).
- [3] *How to compute bits per character (BPC)*? Anglais. Mai 2017.
URL : <https://stats.stackexchange.com/questions/211858/how-to-compute-bits-per-character-bpc> (visité le 15/05/2018) (cf. p. 18).
- [4] *Le Loria*. LORIA. Juil. 2018. URL : <http://www.loria.fr> (visité le 31/07/2018) (cf. p. 21, 23).
- [5] *Organisation*. LORIA. Août 2018.
URL : <http://www.loria.fr/fr/presentation/organisation/> (visité le 03/08/2018) (cf. p. 21).
- [6] *Organigramme 2017 du Loria*. LORIA. Août 2018.
URL : <http://www.loria.fr/wp-content/uploads/2016/05/organigramme-2017.pdf> (visité le 03/08/2018) (cf. p. 22).
- [7] *About SYNALP*. Anglais. SYNALP. Juil. 2018. URL :
<http://synalp.loria.fr/pages/about-synalp> (visité le 30/07/2018) (cf. p. 23).
- [8] W. XIONG et associés.
« Achieving Human Parity in Conversational Speech Recognition ». Anglais.
Dans : CoRR (oct. 2016). arXiv : [1610.05256 \[cs.CL\]](https://arxiv.org/abs/1610.05256).
URL : <https://arxiv.org/abs/1610.05256> (visité le 17/08/2018) (cf. p. 25).
- [9] Rafal JÓZEFOWICZ et associés. « Exploring the Limits of Language Modeling ». Anglais.
Dans : CoRR (fév. 2016). arXiv : [1602.02410 \[cs.CL\]](https://arxiv.org/abs/1602.02410).
URL : [http://arxiv.org/abs/1602.02410](https://arxiv.org/abs/1602.02410) (visité le 17/08/2018) (cf. p. 25).
- [10] Andrej KARPATHY. *The Unreasonable Effectiveness of Recurrent Neural Networks*. Anglais.
Mai 2015. URL :
<http://karpathy.github.io/2015/05/21/rnn-effectiveness> (visité le 23/04/2018) (cf. p. 25).
- [11] *Grid5000*. Anglais. Grid5000. Avr. 2018. URL :
<https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home> (visité le 25/04/2018) (cf. p. 28).

- [12] Matt MAHONEY. *About the Test Data*. Anglais. Wikipédia. Sept. 2011.
URL : <http://mattmahoney.net/dc/textdata.html> (visité le 17/08/2018) (cf. p. 29).
- [13] Marcus HUTTER. *50'000€ Prize for Compressing Human Knowledge*. Anglais. Fév. 2018.
URL : <http://prize.hutter1.net> (visité le 17/08/2018) (cf. p. 29).
- [14] Tony ARCHAMBEAU. *XML - Définition*. Wikipédia. Août 2018.
URL : <http://glossaire.infowebmaster.fr/xml> (visité le 17/08/2018) (cf. p. 29).
- [15] Soumith CHINTALA. *Deep Learning with PyTorch : A 60 Minute Blitz*. Anglais. PyTorch. Août 2018. URL : http://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html (visité le 12/08/2018) (cf. p. 35).
- [16] Justin JOHNSON. *Learning PyTorch with Examples*. Anglais. PyTorch. Août 2018.
URL : http://pytorch.org/tutorials/beginner/pytorch_with_examples.html (visité le 12/08/2018) (cf. p. 35).
- [17] Sean ROBERTSON. *Classifying Names with a Character-Level RNN*. Anglais. PyTorch. Août 2018. URL : http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html (visité le 12/08/2018) (cf. p. 35).
- [18] *PyTorch 0.4 documentation*. Anglais. PyTorch. Août 2018. URL :
<https://pytorch.org/docs/stable/index.html> (visité le 12/08/2018) (cf. p. 35).
- [19] *User :Ibada/Tuto Deep Learning*. Anglais. Grid5000. Avr. 2018.
URL : https://www.grid5000.fr/mediawiki/index.php/User:Ibada/Tuto_Deep_Learning (visité le 25/04/2018) (cf. p. 35).
- [20] Christophe CERISARA. *How to install pytorch on our GPU cluster*. Anglais.
Intranet DeepLoria. Mar. 2017.
URL : <http://deeploria.gforge.inria.fr/intranet/pytorch> (visité le 25/04/2018) (cf. p. 35).
- [21] Stephen MERITY. *Smerity/awd-lstm-lm*. Anglais. Août 2018.
URL : <https://github.com/Smerity/awd-lstm-lm> (visité le 12/08/2018) (cf. p. 35).
- [22] *PaddlePaddle/VisualDL*. Anglais. Mai 2018.
URL : <https://github.com/PaddlePaddle/VisualDL> (visité le 09/05/2018) (cf. p. 39).
- [23] *VisualDL - Visualize your deep learning training and data flawlessly*. Anglais. Mai 2018.
URL : <http://visualdl.paddlepaddle.org> (visité le 09/05/2018) (cf. p. 39).
- [24] Jason BROWNLEE. « A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size ». Anglais. Dans : *Machine Learning Mastery* (juil. 2017). URL :
<https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size> (visité le 13/04/2018) (cf. p. 44).
- [25] Viacheslav KHOMENKO et associés. *Accelerating recurrent neural network training using sequence bucketing and multi-GPU data parallelization*. 2017. arXiv : [1708.05604](https://arxiv.org/abs/1708.05604). URL : <http://arxiv.org/abs/1708.05604> (cf. p. 44).

- [26] Fen XIAO et associés.
 « MSDNN : Multi-Scale Deep Neural Network for Salient Object Detection ». Anglais.
 Dans : *CoRR* (jan. 2018). arXiv : [1801.04187 \[cs.CV\]](https://arxiv.org/abs/1801.04187).
 URL : <https://arxiv.org/abs/1801.04187> (visité le 17/08/2018) (cf. p. 52).
- [27] P. HUBERetJ. NIEHUES et A. WAIBEL.
 « A Hierarchical Approach to Neural Context-Aware Modeling ». Anglais.
 Dans : *ArXiv e-prints* (juil. 2018). arXiv : [1807.11582 \[cs.CL\]](https://arxiv.org/abs/1807.11582).
 URL : <https://arxiv.org/abs/1807.11582> (visité le 17/08/2018) (cf. p. 52).
- [28] EUREKA. Wikipédia. Août 2018.
 URL : <https://fr.wikipedia.org/wiki/EUREKA> (visité le 01/08/2018) (cf. p. 55).
- [29] EUREKA Network Projects. Anglais. Août 2018.
 URL : <https://ec.europa.eu/growth/tools-databases/eip-raw-materials/en/content/eureka-network-projects> (visité le 21/08/2018) (cf. p. 55).
- [30] Le programme EUREKA. Août 2018.
 URL : <http://www.horizon2020.gouv.fr/cid73270/le-programme-eureka.html> (visité le 21/08/2018) (cf. p. 55).
- [31] EUREKA | Helping your company in growing and bringing an innovation to the market. Août 2018. URL : <http://www.eurekanetwork.org> (visité le 21/08/2018) (cf. p. 55).
- [32] 16037 PAPUD. Anglais. itea3.org. Juil. 2018.
 URL : <https://itea3.org/project/papud.html> (visité le 31/07/2018) (cf. p. 56, 57).
- [33] Produits. Atos. Août 2018.
 URL : <https://atos.net/fr/produits> (visité le 01/08/2018) (cf. p. 57).
- [34] Définition : délivrable. Août 2018. URL : <https://www.cordial.fr/dictionnaire/definition/d%C3%A9livrable.php> (visité le 18/08/2018) (cf. p. 61).
- [35] Dinghan SHEN et associés. « Baseline Needs More Love : On Simple Word-Embedding-Based Models and Associated Pooling Mechanisms ». Anglais.
 Dans : *CoRR* (mai 2018). arXiv : [1805.09843 \[cs.CL\]](https://arxiv.org/abs/1805.09843).
 URL : [http://arxiv.org/abs/1805.09843](https://arxiv.org/abs/1805.09843) (visité le 17/08/2018) (cf. p. 66).
- [36] Pavel SURMENOK.
 « Estimating an Optimal Learning Rate For a Deep Neural Network ». Anglais.
 Dans : *Towards Data Science* (nov. 2017).
 URL : <https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0> (visité le 17/08/2018) (cf. p. 69).
- [37] PEP 483 – Type Hints. Anglais. Août 2018.
 URL : <https://www.python.org/dev/peps/pep-0484/#using-none> (visité le 17/08/2018) (cf. p. 71).
- [38] PEP 484 – Type Hints. Anglais. Août 2018.
 URL : <https://www.python.org/dev/peps/pep-0484/#using-none> (visité le 17/08/2018) (cf. p. 71).

- [39] Stephen MERITY et Nitish Shirish KESKAR et Richard SOCHER.
 « Regularizing and Optimizing LSTM Language Models ». Anglais.
 Dans : *arXiv preprint arXiv:1708.02182* (2017).
- [40] Stephen MERITY et Nitish Shirish KESKAR et Richard SOCHER.
 « An Analysis of Neural Language Modeling at Multiple Scales ». Anglais.
 Dans : *arXiv preprint arXiv:1803.08240* (2018).
- [41] Ilya SUTSKEVER. « Training Recurrent Neural Networks ». Anglais.
 Thèse de doct. Graduate Department of Computer Science, University of Toronto, 2013.
 URL : <https://machinelearningmastery.com/gentle-introduction-backpropagation-time> (visité le 29/05/2018).
- [42] Y. BENGIO et J. S. SENECA. « Adaptive Importance Sampling to Accelerate Training of a Neural Probabilistic Language Model ». Anglais.
 Dans : *IEEE Trans. Neural Networks* 19.4 (avr. 2008), p. 713–722. ISSN : 1045-9227.
 DOI : [10.1109/TNN.2007.912312](https://doi.org/10.1109/TNN.2007.912312).
- [43] *Apprentissage profond*. Wikipédia. Juil. 2018.
 URL : https://fr.wikipedia.org/wiki/Apprentissage_profond (visité le 10/08/2018).
- [44] *Markdown*. Wikipédia. Juil. 2018.
 URL : <https://fr.wikipedia.org/wiki/Markdown> (visité le 10/08/2018).
- [45] *Réseau de neurones récurrents*. Wikipédia. Août 2018.
 URL : https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_r%C3%A9currents (visité le 10/08/2018).
- [46] *Réseau neuronal*. Futura. Août 2018. URL : <https://www.futura-sciences.com/tech/definitions/informatique-reseau-neuronale-601> (visité le 10/08/2018).
- [47] Jason BROWNLEE.
 « What is the Difference Between Test and Validation Datasets ? » Anglais.
 Dans : *Machine Learning Mastery* (juil. 2017).
 URL : <https://machinelearningmastery.com/difference-test-validation-datasets> (visité le 11/08/2018).
- [48] Joe DAVISON. « “Deep Learning est mort. Vive Differentiable Programming !” » Anglais.
 Dans : *techburst* (juin 2018).
 URL : <https://techburst.io/deep-learning-est-mort-vive-differentiable-programming-5060d3c55074> (visité le 11/08/2018).
- [49] Matt MAZUR. *mattm/simple-neural-network*. Anglais. Mar. 2015.
 URL : <https://github.com/mattm/simple-neural-network> (visité le 23/04/2018).
- [50] Andrej KARPATY. *Minimal character-level language model with a Vanilla Recurrent Neural Network, in Python/numpy*. Anglais. Juil. 2015.
 URL : <https://gist.github.com/karpathy/d4dee566867f8291f086> (visité le 23/04/2018).
- [51] Andrej KARPATY. *CS231n Convolutional Neural Networks for Visual Recognition*. Anglais.
 URL : <http://cs231n.github.io/transfer-learning> (visité le 24/04/2018).
- [52] Andrej KARPATY. *karpathy/char-rnn*. Anglais. Avr. 2016.
 URL : <https://github.com/karpathy/char-rnn> (visité le 23/04/2018).

- [53] Edwin CHEN. *Introduction to Conditional Random Fields*. Anglais.
URL : <http://blog.echen.me/2012/01/03/introduction-to-conditional-random-fields> (visité le 24/04/2018).
- [54] Christopher OLAH. *Understanding LSTM Networks*. Août 2015. URL :
<http://colah.github.io/posts/2015-08-Understanding-LSTMs> (visité le 24/04/2018).
- [55] *The Average Sentence Length*. Anglais. Juil. 2008.
URL : <https://strainindex.wordpress.com/2008/07/28/the-average-sentence-length> (visité le 02/05/2018).
- [56] *Optimizing PyTorch training code*. Anglais. Mai 2018.
URL : <https://www.sagivtech.com/2017/09/19/optimizing-pytorch-training-code> (visité le 22/05/2018).
- [57] *Tail Recursion Elimination*. Anglais. Mar. 2018.
URL : <http://neopythonic.blogspot.fr/2009/04/tail-recursion-elimination.html> (visité le 16/05/2018).
- [58] Jason BROWNLEE. « A Gentle Introduction to Backpropagation Through Time ». Anglais.
Dans : *Machine Learning Mastery* (juin 2017).
URL : <https://machinelearningmastery.com/gentle-introduction-backpropagation-time> (visité le 06/07/2018) (cf. p. 102).
- [59] Jason BROWNLEE. *Réseaux de Neurones*. Statistica. Juin 2017. URL :
<http://www.statsoft.fr/concepts-statistiques/reseaux-de-neurones-automatises/reseaux-de-neurones-automatises.htm> (visité le 11/08/2018).
- [60] *Les réseaux de neurones expliqués à ma fille*. OD-Datamining. Août 2018.
URL : <https://od-datamining.com/knibase/les-reseaux-de-neurones-expliques-a-ma-fille> (visité le 11/08/2018).
- [61] *Mémoire, rapport de stage : les règles de présentation*. l'Étudiant. Août 2018.
URL : <https://www.letudiant.fr/jobsstages/nos-conseils/memoire-rapport-de-stage-les-regles-de-presentation.html> (visité le 03/08/2018).
- [62] *Cadre Théorique d'un Mémoire - Contenu et Exemple*. Scribbr. Fév. 2016.
URL : <https://www.scribbr.fr/memoire/cadre-theorique-d-un-memoire> (visité le 02/08/2018).

Glossaire

A

Apprentissage automatique Défini sous-section 2.1.1, page 15. 15, 16

Apprentissage profond Défini sous-section 2.2.1, page 16. 13, 16, 28, 31, 51, 56, 62, 69, 73, 74

Architecture Défini sous-section 2.2.3, page 16. 16, 17

B

Batch Défini sous-section 7.5.4, page 44. 42, 44, 68, 71

Bogue 35

C

Centres de calcul 57

Cloud 57

Corpus Défini sous-section 2.1.4, page 16. 16, 30, 67

D

Différentiation automatique 43

Données Défini sous-section 2.1.4, page 16. 13, 15, 16, 25, 26, 32, 40, 59, 60, 61, 67, 68, 70

E

Embedding Défini sous-section 7.2.2, page 36. 36, 38, 65, 66

Entraînement Défini sous-section 2.1.3, page 15. 15, 16, 67, 68, 69

Époque Défini sous-section 2.1.4, page 16. 16, 18

État de l'art Défini note 1, page 27. 26, 35, 37, 47, 51

Exemple Défini sous-section 2.1.4, page 16. 16, 25, 40

F

Fichier de journalisation Défini chapitre 11, page 63. 59, 63

G

Grammaires formelles 23

L

LSTM Défini sous-section 2.2.5, page 17. 17, 25, 36, 37, 42

M

Matrice 16, 28

Modèle Défini sous-section 2.1.2, page 15. 14, 15, 16, 18, 26, 28, 30, 31, 35, 36, 37, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 52, 67, 68, 69, 73

Modèle de la langue Défini sous-section 2.3.2, page 17. 13, 17, 18, 23, 26, 31, 35, 47, 51, 59, 65, 66

Module Défini sous-section 2.2.3, page 16. 16

P

Paramètre Défini sous-section 2.2.4, page 17. 16, 42, 43, 45, 46

Prétraitement des données Défini sous-section 2.1.4, page 16. 16, 29, 51, 60, 61, 68, 70

Processeur graphique Défini note 2, page 28. 28

R

Réseau de neurones récurrents multi-échelles croissant Défini chapitre 6, page 31. 26, 31, 32, 35, 37, 42, 43, 44, 45, 46, 51, 52, 65

Réseau de neurones Défini sous-section 2.2.2, page 16. 13, 14, 16, 17, 25, 28, 31, 36, 39, 43, 44, 51, 59, 65, 74

RNN Défini sous-section 2.2.5, page 17. 17, 25, 31, 32, 36, 37, 38, 39, 42, 45, 51, 52, 59, 66, 73

S

Sémantique computationnelle 23

T

Tenseur Défini sous-section 2.2.2, page 16. 16, 28, 36, 38, 40, 65

Traitement de la parole 23

Traitement automatique des langues Défini sous-section 2.3.1, page 17. 13, 17, 23, 25, 74

Annexes

Table des annexes

A Rapports d'avancement du projet GMSNN	87
A.1 Informations sur les rapports contenus dans la présente annexe	87
A.2 Étude des problèmes de mémoire	88
A.3 Sauvegarde du modèle	93
A.4 Tentative de solution pour les fuites mémoires	97
A.5 Problèmes théoriques liés à l'utilisation de <i>batchs</i> pour l'entraînement	99
A.6 Tentative de réduction du temps de calcul par utilisation de l'algorithme d'entraînement « <i>Truncated BPTT</i> »	102
A.7 Entraînement couche par couche	104
A.8 Premier test du modèle réimplémenté	106
A.9 Premier entraînement complet du modèle réimplémenté	107
A.10 Premier entraînement à 50 époques du modèle réimplémenté	109
A.11 Premier test du modèle multi-échelles	113
A.12 Comparaison des stratégies de fusion des résultats des différentes couches	117
A.13 Test des effets du changement de taille des paquets (<i>batch</i>)	122
A.14 Entraînement sur le corpus complet avec beaucoup de temps alloué	124
A.15 Changement de stratégie de gestion d'historique	131
A.16 Test de l'implémentation des <i>batchs</i>	138
A.17 Test des performances des <i>batchs</i>	142
A.18 Test des performances des <i>batchs</i> sur 50 époques	146
A.19 Test des performances des différentes améliorations	151

B Rapports d'avancement du projet PAPUD	159
B.1 Informations sur les rapports contenus dans la présente annexe	159
B.2 Informations générales	160
B.3 Résultats de l'implémentation basique	162
B.4 Paquets (<i>batchs</i>) simultanés	164
B.5 Analyse du pic de performance	168
B.6 Rapport de la réunion avec les autres membres du projet	172
B.7 Optimisation du taux d'apprentissage	174
B.8 Effets de l'optimisation du taux d'apprentissage	178
B.9 Taille de <i>batch</i> binaire	180
B.10 Performances du lecteur de corpus multi-fichiers multi-processus	182
C Copie de la convention de stage	187
D Copie de l'avenant à la convention de stage	193

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 au maître de stage tout au long du projet GMSNN.

A.1.1 Format d'origine des rapports

Le langage Markdown, plus spécifiquement dans le dialecte nommé Gitlab Flavoured Markdown (littéralement « Markdown à la saveur de Gitlab »), 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 document a nécessité l'adaptation au format papier du contenu rédigé en Gitlab Flavoured Markdown.

Certains éléments n'ont pas pu être transcrits de façon exacte, en particulier les liens et les tableaux et images de grande taille. Ces éléments ont donc été adaptés.

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 a 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.

A.2 Étude des problèmes de mémoire

Analysis of memory usage

Analysis report

by E. Marquer, 2018/05/23, Synalp and Université de Lorraine

A.2.1 Abstract

Experimental results shows (using nvidia-smi) an increasing memory usage for 4 layers, from an already enormous 3GB to more than 6GB, causing an out-of-memory error.

The objective of the following computations is to estimate the memory consumption of the model, to confirm the hypothesis of a memory leak, and verify that the model should not overflow memory.

A.2.2 Formulas

Tensor and Variable size estimation

Byte size of a tensor is close to 6 times the products of all of its dimensions. Byte size of a variable is similar to that of the corresponding tensor.

```
1 import torch, pickle
2
3 # Object to mesure
4 o = torch.autograd.Variable(torch.ones(100, 100, 100))
5 o = torch.ones(100, 100, 100)
6
7 len(pickle.dumps(o, 0)) / (100 * 100 * 100)
8 #result = 6
```

Computations

$$\begin{aligned}
total &= hidden_states + msnn_weights + emb_weights + out_weights \\
&= detach_interval * (growth_factor + 1) * layers * 6 * (hidden_size * batch_size * sequence_length) \\
&\quad + (((layer - 1) * 8 * hidden_size * (hidden_size + 1)) \\
&\quad + 4 * hidden_size * (hidden_size + emb_size + 2)) * 6 \\
&\quad + (nwords * (emb_size + 1)) * 6 \\
&\quad + (nwords * (layers * hidden_size)) * 6 \\
\\
&= 6 * (detach_interval * (growth_factor + 1) * layers * (hidden_size * batch_size * sequence_length)) \\
&\quad + ((layers - 1) * 8 * hidden_size * (hidden_size + 1)) \\
&\quad + 4 * hidden_size * (hidden_size + emb_size + 2) \\
&\quad + (nwords * (emb_size + 1)) \\
&\quad + (nwords * (layers * hidden_size)))
\end{aligned} \tag{A.1}$$

$$\begin{aligned}
hidden_states &= total_history * 6 * dim \\
&= detach_interval * (growth_factor + 1) * layers * 6 * dim \\
&= detach_interval * (growth_factor + 1) * layers * 6 * \\
&\quad (hidden_size * batch_size * sequence_length)
\end{aligned} \tag{A.2}$$

$$\begin{aligned}
msnn_layer_weights &= weights_ih + weights_hh + bias_ih + bias_hh \\
&= 4 * hidden_size * input_size + 4 * hidden_size * hidden_size \\
&\quad + 4 * hidden_size + 4 * hidden_size \\
&= 4 * hidden_size * (hidden_size + input_size + 2) \\
&= \begin{cases} 8 * hidden_size * (hidden_size + 1) & \text{for all layers except the first one} \\ 4 * hidden_size * (hidden_size + emb_size + 2) & \text{for the first layer} \end{cases}
\end{aligned} \tag{A.3}$$

$$\begin{aligned}
msnn_weights &= msnn_layer_weights * layers \\
&= ((layers - 1) * 8 * hidden_size * (hidden_size + 1)) \\
&\quad + 4 * hidden_size * (hidden_size + emb_size + 2)
\end{aligned} \tag{A.4}$$

$$\begin{aligned}
emb_weights &= (bias + weights) * 6 \\
&= (nwords * emb_size + emb_size) * 6 \\
&= (nwords * (emb_size + 1)) * 6
\end{aligned} \tag{A.5}$$

$$out_weights = (nwords * (layers * hidden_size)) * 6 \tag{A.6}$$

Estimate with basic set of parameters

```
1 detach_interval = 50
2 growth_factor = 5
3 layers = 7
4 hidden_size = 1840 / 4
5 batch_size = 2
6 sequence_length = 100
7 emb_size = 400
8 nwords = 205
9
10 total = 6 * (detach_interval * (growth_factor + 1) * layers * hidden_size *
11     batch_size * sequence_length + ((layers - 1) * 8 * hidden_size * (
12         hidden_size + 1)) + 4 * hidden_size * (hidden_size + emb_size + 2) + (
13         nwords * (emb_size + 1)) + (nwords * (layers * hidden_size)))
14
15 " {}GB {}MB {}kB {}B".format(int(total%(1024**4) / 1024**3), int(total
16     %(1024**3) / 1024**2), int(total%(1024**2) / 1024), int(total%1024))
17 # result: '1GB 741MB 614kB 9B'
```

detach_interval	growth_factor	layers	hidden_size	batch_size	sequence_length	emb_size	nwords	total
50	5	7	1840 / 4	2	200 / batch_size	400	205	1GB 153MB 68kB 6B
100	5	7	1840 / 4	2	200 / batch_size	400	205	2GB 234MB 579kB 262B
200	5	7	1840 / 4	2	200 / batch_size	400	205	4GB 397MB 577kB 774B
50	10	7	1840 / 4	2	200 / batch_size	400	205	2GB 50MB 323kB 390B
50	5	6	1840 / 4	2	200 / batch_size	400	205	1008MB 912kB 414B
50	5	5	1840 / 4	2	200 / batch_size	400	205	840MB 732kB 822B
50	5	4	1840 / 4	2	200 / batch_size	400	205	672MB 553kB 206B
50	5	3	1840 / 4	2	200 / batch_size	400	205	504MB 373kB 614B
50	5	2	1840 / 4	2	200 / batch_size	400	205	336MB 193kB 1022B
50	5	1	1840 / 4	2	200 / batch_size	400	205	168MB 14kB 406B
50	5	7	1840 / 2	2	200 / batch_size	400	205	2GB 431MB 598kB 94B
50	5	7	1840 / 8	2	200 / batch_size	400	205	573MB 28kB 890B
50	5	7	1840 / 4	1	200 / batch_size	400	205	1GB 153MB 68kB 6B
50	5	7	1840 / 4	2	200	400	205	2GB 234MB 579kB 262B
50	5	7	1840 / 4	2	200 / batch_size	200	205	1GB 150MB 743kB 534B
50	5	7	1840 / 4	2	200 / batch_size	800	205	1GB 157MB 764kB 998B
50	5	7	1840 / 4	2	200 / batch_size	400	500	1GB 159MB 182kB 984B

Most impactful factors (memory-wise)

- detach_interval : detach_interval * 2 = memory * 2
- growth_factor : growth_factor * 2 = memory * 2
- hidden_size : hidden_size * 2 = memory * 2
- batch_size * sequence_length (number of examples by sequence) : batch_size*sequence_length * 2 = memory * 2
- layers : layers * 2 = layers * 2

The others factors considered (emb_size and nwords) have almost no impact on memory. It confirms that the most memoryphage element is the MSNN. Also, to keep a stable memory usage, batch_size * sequence_length ratio must be kept constant; increasing batch_size while lowering sequence_length can increase processing speed whithout impacting memory usage.

Impact of layer increase

We can notice that the impacts of layers is most noticeable during the first phases of training, during the creation of the first layer. Later on, the creation frequency of new layers is small, and the change is minimal. For example, from 6 to 7 layer, we need ‘12500‘ sequences to pass, and the increase of memory of about ‘7/6‘; from 7 to 8 layer, we need ‘62500‘ sequences to pass, and the increase of memory of about ‘8/7‘; from 8 to 9 layer, we need ‘312500‘ sequences to pass, and the increase of memory of about ‘9/8‘; and so on.

Partial derivatives :

```
1 d = detach_interval
2 g = growth_factor
3 l = layers
4 h = hidden_size
5 b = batch_size
6 s = sequence_length
7 e = emb_size
8 n = nwords
9
10 complete formula:
11 6 *
12   d * (g + 1) * l * h * b * s +
13   ((l - 1) * 8 * h * (h + 1)) +
14   4 * h * (h + e + 2) +
15   (n * (e + 1)) +
16   (n * (l * h))
17 )
18
19 simplified formula:
20 6*(8*l*h^2 + 1*d*g*h*b*s + 1*d*h*b*s + 8*l*h + l*n*h + 4*e*h + e*n + n - 4*h
21 ^2)
22
23 partial derivate on given variable:
24 d: (6(8*l*h^2 - 4*h^2 + 1*d*g*h*b*s + 1*d*h*b*s + 8*l*h + 4*e*h + l*n*h + e*n
25 + n))
26 g: 6*l*d*h*b*s
27 l: 6*(d*h*b*s*(g+1) + 8*h*(h+1) + nh)
28 h: 6*(l*d*g*b*s + l*d*b*s + l*n + 16*l*h + 8*l + 4*e - 8*h)
29 b: 6*l*d*h*s*(g+1)
30 s: 6*l*d*h*b*(g+1)
31 e: 0
32 n: 6*(l*h+e+1)
```

A.3 Sauvegarde du modèle

Analysis and implementation of job save and restart

Implementation report

by E. Marquer, 2018/05/24
Synalp and Université de Lorraine

A.3.1 Abstract

As the jobs are taking longer than an hour per epoch, it has become necessary to keep an image of the model, the training parameters and the current state of the model to be able to interrupt training when needed.

Multiple choices are available :

- an easy but heavy serialisation (pickle) of the whole system;
- a “full” save of the model and parameters, excluding everything that can be recomputed easily;
- a “partial” save of the model, removing a part of the less important information.

A.3.2 End solution : saving using pytorch utilities

The saving solution currently implemented is the `torch.save(trainer, file)` and `trainer = torch.load(file)` and the alternatie version `trainer = torch.load(file, map_location=lambda storage, loc: storage)` allowing the loading of a CUDA model out of CUDA.

This solution posed a number of problems when it was first tested (internal non-parameter attributes where not correctly saved and loaded), that is why it was not considered at first.

But a more recent try resulted in a perfect result, replacing the need of a custom serialisation system.

A.3.3 Full serialisation

This kind of serialisation is done thanks the pickle package, on the whole Trainer object.

```
1 import pickle
2
3 def load(filename: str) -> object:
4     with open(filename, 'rb') as f:
5         return pickle.load(f)
6
7 def save(filename: str, trainer: object) -> None:
8     with open(filename, 'wb') as f:
9         pickle.dump(trainer, f)
```

A.3.4 Full save

Elements to save

- Corpus file name
- cuda_on
- batch_size
- Trainer
 - Model (see specific points)
 - self.epoch current epoch
 - self.batch and self.i current position in training epoch
 - self.start_time needs a little work : storing the elapsed time, and when loading, remove elapsed time from load time
 - self.log_interval
 - self.save_interval
 - self.bptt
 - self.nwords
 - self.repackage_interval
 - self.repackage_strategy
 - self.reset_growth
 - self.reset_hidden
 - self.epochs
 - self.save_folder
- Model (MSNN)
 - self.training
 - input layer (embeddings) using torch.save(self.emb, f) and torch.load(f)
 - MSNN (see specific points)
 - output layer (linear)
- MSNN
 - Final
 - self.input_size
 - self.hidden_size
 - self.growth_factor
 - self.batch_size
 - self.cuda_on
 - self.layer_id
 - self.max_detach
 - self.repackage_strategy
 - self.max_layers
 - Next MSNN
 - self.detach_count
 - self.rnn
 - self.hidden
 - self.seq_count
 - self.detach_count
 - self.transmitted_output
 - self.transmitted_hidden

Elements to forget and recreate

- Trainer
 - Corpus
 - Corpus file name is needed
 - Optimizer : resign learning rate, weight_decay and model's parameters (create after model is loaded) using `torch.optim.SGD(model.parameters(), lr=args.lr, weight_decay=args.weight_decay)`
 - Criterion : `criterion = torch.nn.CrossEntropyLoss()`
 - self.layers using `self.model.msnn.get_layers()` (create after model is loaded)
 - self.train_data using `batchify(corpus.train, batch_size, cuda_on)`
 - self.val_data using `batchify(corpus.valid, batch_size, cuda_on)`
 - self.plotdata using {"Epoch": None, "Layer": None, "Frac": None} and `self.init_plotter()` (perhaps with a new file, otherwise in append mode without cleaning the file)
 - self.msnn_backup, should be empty if the new backup system (using the with statement) is implemented
- Model
 - None
- MSNN
 - self.tensors

A.3.5 Partial save

The elements to keep are the same as with the Full save, except : - hidden states and transmitted output are to be detached

A.3.6 Other things that need to be added to interrupt and resume job

Methods to interrupt and resume epoch loop and training loop

Interruption strategy Interruption can be done by :

- saving the state at specific timestep ;
- saving automatically when shutting down.

The second option isn't really realistic, as an interruption wouldn't allow a big enough margin to save the model. Even though, it could be added to allow manual interruption at certain timesteps (smaller than the ones implemented in the first option).

As the first option is the only viable one, multiple timesteps are possible, from the shortest to the longest :

- after each operation (example : after forward pass, and after backward pass, and after optimization ...);
- after each sequence ;
- after n sequences ;
- after each epoch ;
- after n epochs ;
- after the whole training.

Option [1.] is not viable, as it would consume a lot of time relative to each computation for the sole writing process. Option [3.], of which option [2.] is a specific case, allows both a fine granulation with only a minimal loss if anything were to occur, and a reduced burden computation-wise. From option [4.] onward, the save time is negligible, and even if the granulation is mediocre, it offers fine milestones for specific usage (usage of an already trained mode for example).

Currently, at creation time, with no history, the model save file is about 770MB.

What will be implemented is the third option, saving the model after n sequences.

Resume strategy It will be necessary to adapt the training loops a little to allow resuming at any sequence in any epoch.

A.3.7 Other methods implemented

Method using **with** keyword for backup system, during evaluation

```
1 class BackupContextManager:
2     def __init__(self, model: DetRNN):
3         self.model = model
4
5     def __enter__(self):
6         # Backing up training linked data
7         self.msnBackup = self.model.msnBackup()
8
9     def __exit__(self, type, value, traceback):
10        # Restoring training linked data
11        self.model.msnRestore(self.msnBackup)
12
13
14 with BackupContextManager(model):
15     """Do computations"""
```

A.4 Tentative de solution pour les fuites mémoires

Analysis of a source of the memory leak problem : the history management system

Analysis report

by E. Marquer, 2018/05/28, Synalp and Université de Lorraine

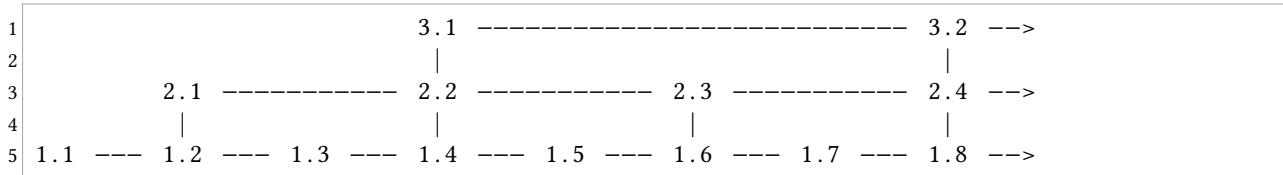
A.4.1 Abstract

A big memory leak is present in the model. One of the identified cause is a malfunction in the history management system.

A.4.2 Problem

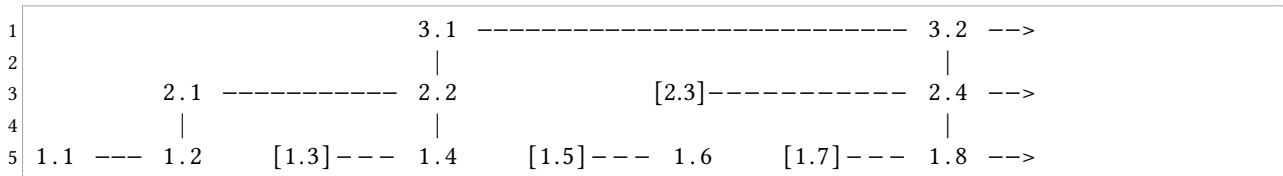
It seems simply detaching a hidden state is not enough :

With a graph :

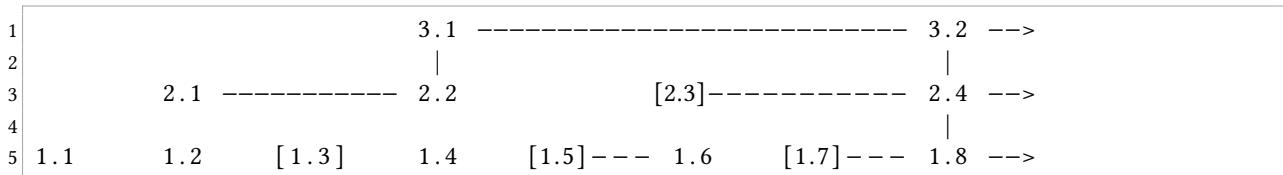


where all nodes are hidden states, and each layer is a line, with a transmission rate of 1 transmission every 2 hidden state.

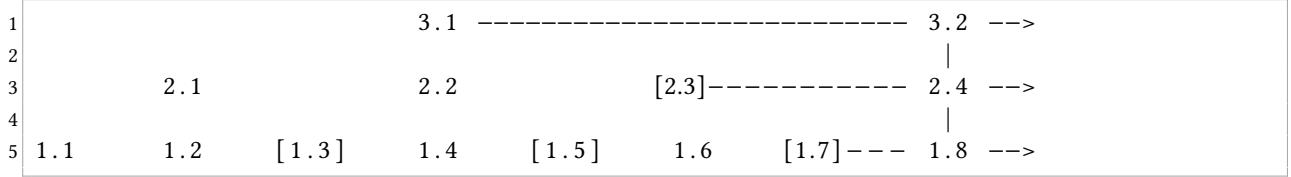
When detaching every 2 hidden, with [i,j] a detached node, the graph becomes :



But it should be :



Or with a more aggressive strategy :



A.4.3 Solution

To solve the problem, keeping track of all history is necessary, and with the way PyTorch works, it won't be a problem to keep references to history before deleting them

A.5 Problèmes théoriques liés à l'utilisation de *batches* pour l'entraînement

Analysis of batch training in msnn

Analysis report

by E. Marquer, 2018/05/29, Synalp and Université de Lorraine

A.5.1 Abstract

A common method to improve training time of a RNN is the batch based training, but MSNN are highly dependent on past history and continuity. This training strategy is based on passing simultaneously multiple inputs to the network. Training with 3 batches is equivalent to a parallel training over 3 corpora composed of respectively every 1st batch, every 2nd batch, and every 3rd batch. It necessitates the splitting of the corpus, and doing so breaks the continuity between the different parts. As such, it would be difficult to use the batch based training for MSNN.

A.5.2 Bacthifying strategies

There are multiple batchifying strategies, here explained with the Alphabet as a corpus.

MSNN is currently trained with the BPTT (and Truncated-BPTT) algorithm. By passing multiple sequences, there are two possible batchifying :

- batchifying across each sequence of the corpus ;
- batchifying across the corpus then sequence of the corpus.

No batchifying

Table 1

Batch\Timestep	1	2	3	4	5	6	7	8	9	10	11	...	24	25	26
1	A	B	C	D	E	F	G	H	I	J	K	...	X	Y	Z

BPTT sequence-wide batchifying

Example with a BPTT sequence length of 3 (first inputs of the sequence are in bold) :

Table 2

Batch\Timestep	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	A	B	C	G	H	I	M	N	O	S	T	U	Y	Z
	D	E	F	J	K	L	P	Q	R	V	W	X		

Corpus-wide batchifying

Batch\Timestep	1	2	3	4	5	6	7	8	9	10	11	12	13
1	A	B	C	D	E	F	G	H	I	J	K	L	M
2	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Other batchifying strategies

Other batchifying strategies exists, mostly by spreading the corpus along the batch dimension and not the timestep dimension.

One such strategy would be :

Batch\Timestep	1	2	3	4	5	6	7	8	9	10	11	12	13
1	A	C	E	G	I	K	M	O	Q	S	U	W	Y
2	B	D	F	H	J	L	N	P	R	T	V	X	Z

A.5.3 Analysis of the different strategies

Spreading the corpus along the batch dimension

The worst possible strategies seem to be A.5.2spreading the corpus along the batch dimension : each input of the corpus is a succession of characters separated by a gap of the length of the batch dimension, so :

- the input is always incomplete, as the different batches do not interact with each other;
- the batches are full of discontinuity;
- the network is not reusable for any number of batches and input.

Sequence batchifying

Then there is the BPTT sequence-wide batchifying. In each sequence, the batches are internally coherent, but between sequences, the batches are discontinued (C|G, F|J, ...). Moreover, if the first layers' input are internally coherent, upper layers are subjected as similar input as presented in A.5.2, and the problems of the corresponding strategy is back.

Corpus batchifying

The last and best strategy is the Corpus-wide batchifying. Even if this one need the pre-processing of the corpus, it offers a lot of advantages :

- the input is not discontinued, even in the last layer;

- the network is usable for any number of batches, as such strategy is equivalent to a parallel training over distinct corpora, each composed of a part of the original corpus;
- the only layers really affected by the remaining discontinuity are the last ones, as over multiple epochs, they would only get the same part of the corpus and would not be able to extract info from the whole corpus :

Table 5

Batch\Epoch	1	2	3	4	...
1	Part 1	Part 1	Part 1	Part 1	...
2	Part 2	Part 2	Part 2	Part 2	...
3	Part 3	Part 3	Part 3	Part 3	...

This last discontinuity can be solved by a rotation of the batches across multiple epochs :

Table 6

Batch\Epoch	1	2	3	4	...
1	Part 1	Part 2	Part 3	Part 1	...
2	Part 2	Part 3	Part 1	Part 2	...
3	Part 3	Part 1	Part 2	Part 3	...

But that solution leaves the problem of the hidden state, which contains important information, and exists in multiple different exemplary, one for each batch. Another consequence is the need of n^* epochs, for n batches, to accumulate equivalent history.

A.5.4 Conclusion

Corpus-wide batchifying seems to be the best of all batchifying strategies, but it still presents multiple downsides :

- the existence of multiple parallel “memory”;
- the need of n^* epochs, for n batches, to accumulate equivalent “memory” for each batch compared to non-batchified training.

A.6 Tentative de réduction du temps de calcul par utilisation de l'algorithme d'entraînement « *Truncated BPTT* »

Analysis and implementation of improved Truncated-BPTT training algorithm

Implementation report

by E. Marquer, 2018/05/29
Synalp and Université de Lorraine

A.6.1 Abstract

Last update (implementation of explicit history to solve memory leaks problem) solved memory problems, leaving the time consumption problem (hundreds of hours for a single epoch).

As the most time-consuming process is the backpropagation, the most evident way to reduce time consumption is to improve the training strategy.

One of the possible optimization is the Truncated Backpropagation Through Time (Truncated-BPTT or TBPTT).

A.6.2 Notations

The following notations are from an introduction article on BPTT [58] :

- **TBPTT(n,n)** : Updates are performed at the end of the sequence across all timesteps in the sequence (e.g. classical BPTT).
- **TBPTT(1,n)** : timesteps are processed one at a time followed by an update that covers all timesteps seen so far (e.g. classical TBPTT by Williams and Peng).
- **TBPTT(k1,1)** : The network likely does not have enough temporal context to learn, relying heavily on internal state and inputs.
- **TBPTT(k1,k2)**, where $k_1 < k_2 < n$: Multiple updates are performed per sequence which can accelerate training.
- **TBPTT(k1,k2)**, where $k_1 = k_2$: A common configuration where a fixed number of timesteps are used for both forward and backward-pass timesteps (e.g. 10s to 100s).

The base implementation of the model, using the (sequence_length, batch_size, values) model for the inputs (and outputs), is already an implementation of the **TBPTT(n,n)** algorithm.

What would improve a lot time efficiency is to implement a **TBPTT(k1,k2)** algorithm.

A.6.3 Algorithm

The algorithm can decompose into 4 steps : 1. Present a sequence of k_1 timesteps of input and output pairs to the network. 2. Compute loss across the k_2 last timesteps. 3. Backpropagate loss 4. Update weights

A.6.4 Pseudo-python code

Old algorithm

```
1 for sequence in sequences:
2     # 1. Present a sequence of *n* timesteps of input to the network.
3     output, hidden = model.forward(sequence.input, hidden)
4
5     # 2. Compute loss across the *n* timesteps.
6     loss = criterion(output, sequence.targets)
7
8     # 3. Backpropagate loss
9     loss.backward()
10
11    # 4. Update weights
12    optimizer.step()
```

New algorithm

```
1 for sequence in sequences:
2     # 1. Present a sequence of *k1* timesteps of input to the network.
3     output, hidden = model.forward(sequence.input, hidden)
4
5     # 2. Compute loss across the *k2* last timesteps.
6     loss = criterion(output[:-k2], sequence.targets[:-k2])
7
8     # 3. Backpropagate loss
9     loss.backward()
10
11    # 4. Update weights
12    optimizer.step()
```

A.7 Entrâinement couche par couche

Layer by layer training

Analysis report

by E. Marquer, 2018/06/25, Synalp and Université de Lorraine

A.7.1 Abstract

Multiple advanced training algorithms use layer “freezing”, meaning that the layer will not be trained.

As it would be interesting to use those algorithms to get the most out of the current architecture, a layer “freezing” will be implemented. To do so, a dummy algorithm will be implemented. This algorithm is a naive layer by layer training.

A.7.2 Layer by layer training

This training is used to see if convergence can be sped up, if performance can be improved, and if the layered architecture is of any use.

Principle

The general principle is to train each layer individually, and to fine-tune them together frequently.

An iterative presentation would be :

1. Create a layer
2. Train the layer alone
3. Train all the layers together
4. Restart from [1.]

Another way to explain this algorithm is that it is a variation of the IM algorithm, were storing the output of the training of a layer is replaced by recomputing those results. It removes the drawback of the increase in memory usage, while speeding up training(time-wise at least, convergence-wise at best).

Example for a 4 layered MSNN

1. We train for n epochs the first layer. It is expected to learn a maximum of things of a low level of abstraction and time dependency.
2. Then, we freeze this layer, and start training a second one over n epochs. It is expected to learn a higher level of abstraction and time dependency from the representations in the first layer's hidden state.
3. We train those two layers together to fine-tune them.
4. We then add a third layer, freeze the first two layers, and train the third layer of information extracted from the second layer.
5. We train the three layers together.
6. We add a new layer and train it alone.
7. We train the four layers together.
8. We train all the layers until the ends of epochs.

Speeding up convergence and improving performance

Training each layer individually requires less computation during backpropagation. Moreover, by pushing each layer to learn the maximum of information it can learn, we can expect each layer to specialize at their scale.

A more detailed way to understand the intuition behind that is that a layer closer to the data has to learn very basic features. Step by step, every layer is constrained to its respective scale (by its memory span due to the architecture). Each layer has a minimal set of knowledge to learn before benefiting to the whole network. Even if a part of this learning is shared over the layers, at least over the first epochs there would be nothing to gain but noise by training all the layers together.

Globally, by skipping the “noisy” part of the training, a reduction of training time (the real computation time, not the number of epochs needed) is to be expected. A bonus effect would be a small acceleration of convergence, as training would be less noisy.

Use of the layered architecture

By training the model layer by layer, we can expect to see if training a single layer with equivalent parameters would have the same effect (if the individual training time is big enough).

A.8 Premier test du modèle réimplémenté

Test run of `detrnn.py`

Test report

by E. Marquer, 2018/04/26, Synalp and Université de Lorraine

A.8.1 Abstract

Test to see time needed, with GPU and without, to run the basic model of DetRNN.

A.8.2 Paradigm

With branch *reimplement*, allocated time 30 min. Test run of *detrnn.py* with full log output, with and without *cuda*. It does not matter if learning ends, as test is only to get time statistics.

Node

grimani-2, with GPU

A.8.3 Results

About 3 to 4 batch-sets are computed in 5 minutes without *cuda*. About 1 batch-set is computed in 1 s with *cuda*.

Details :

- without GPU : [detrnn2018_4_26-16h39.log](#)
- with GPU : [detrnn2018_4_26-16h41.log](#)

A.8.4 Potential ameliorations & next steps

Next step is to test the state-of-the-art version, but probably only with *cuda*.

A.9 Premier entraînement complet du modèle réimplémenté

Test run of `detrnn.py`

Test report

by E. Marquer, 2018/04/27 Synalp and Université de Lorraine

A.9.1 Abstract

Test to do a complete run, with GPU, of the basic model of DetRNN.

A.9.2 Paradigm

With branch *reimplement*, allocated time 24h, not interactive

Test run of *detrnn.py* with info log output, with *cuda*, for 4 epochs.

With curve auto-plotting, and plot data backup in case of interruption.

Node

OAR_JOB_ID=1554682 with GPU

Job start time : 2018-04-27 14 :11 :00

Estimated job stop time : 2018-04-28 14 :11 :00

Command used : bash oarsub -q production -p "GPU <> 'NO'" -l "nodes=1,walltime =24:00:00" ~/awd-lstm-lm/rundet.sh

A.9.3 Results

Total run time for 4 epochs : 5h33

The most rapid progress was during first epoch, with a maximal decrease of loss of 3/epoch, then the decrease of loss became a constant 0.25/epoch.

Plot

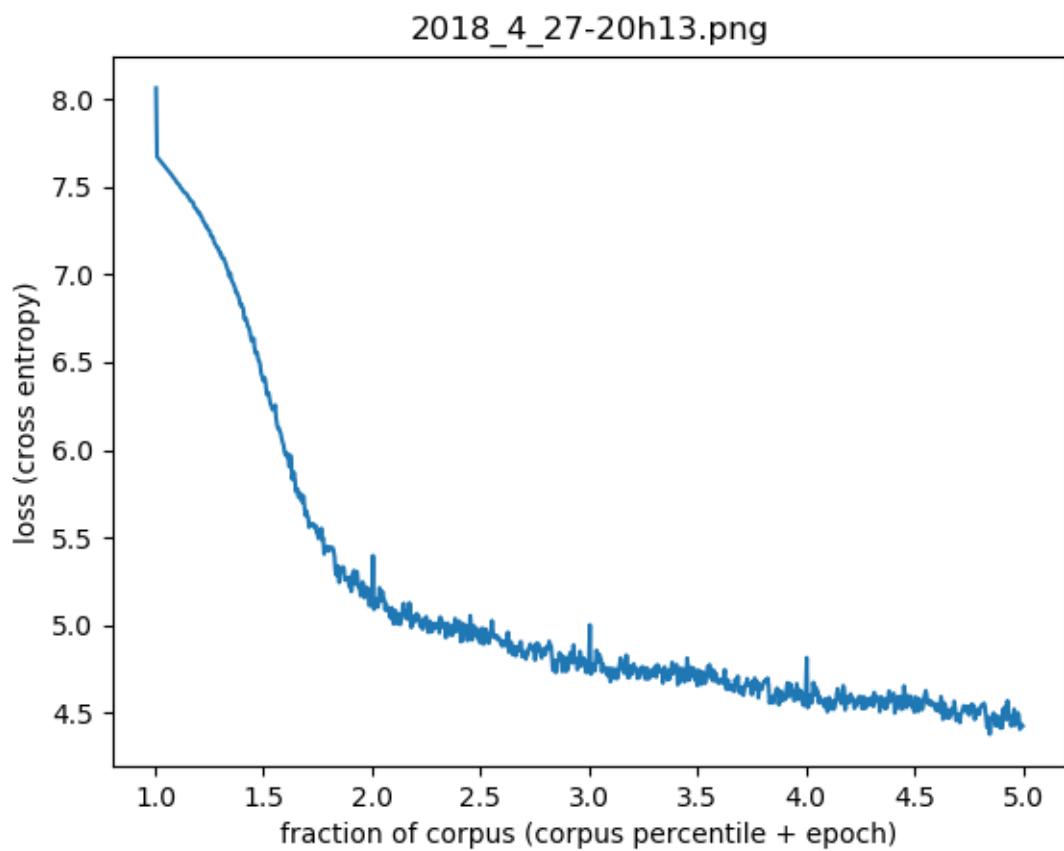


FIGURE A.1 – plot

Details

- log [detrnn2018_4_27-14h11.log](#)
- plot [2018_4_27-16h42.png](#)

A.9.4 Potential ameliorations & next steps

Next step is to test with more epochs, or test the growing model.

A.10 Premier entraînement à 50 époques du modèle réimplémenté

Test run of `detrnn.py`

Test report

by E. Marquer, 2018/04/30, Synalp and Université de Lorraine

A.10.1 Abstract

Test to do a complete run on 50 epoch, with GPU, of the basic model of DetRNN.

A.10.2 Paradigm

This test run of `detrnn.py`, with INFO level log output, loss by percentile and vbpc by epoch, will be executed with `cuda`, for 50 epochs.

The test is done with branch `reimplement`, an allocated time of 76h, not interactive

Run time was estimated for 50 epochs according to the results for 4 epochs (see [2018-04-27_test_run_detrnn.md](#)) :

$$(5\text{h } 33\text{min} / 4 \text{ epoch}) * 50 \text{ epoch} = 4162.5\text{min} = 69\text{h } 22\text{min } 30\text{s}$$

With a security margin of 10h, partially due to reduced batchsize, run time is 80h.

/ Had to reduce batchsize down to 40 because of memory errors */*

Node

OAR_JOB_ID=155659 with GPU

Job start time : 2018-04-30 12 :02 :08

Estimated job stop time : 2018-05-03 16 :02 :08

Command used : bash oarsub -q production -p "GPU <> 'NO'" -l "nodes=1,walltime =80:00:00" ~/awd-lstm-lm/rundet.sh

A.10.3 Results

Total run time for 50 epochs : with real stop time of 2018-05-02 17 :16 :56, the total run time of the training is approximately 53h (2days 5h), corresponding to a little more than an hour per epoch.

BPC-wise, the DetRNN hardly goes under 2.7 even after 50 epoch, with a change of 0.5 BPC in the last 30 epochs.

We can postulate that even after 200 epoch, the DetRNN will not have a BPC under 2.

Plot

BPC/fraction of corpus BPS per fraction of the corpus (an interval of 1 correspond a complete corpus, or an epoch).

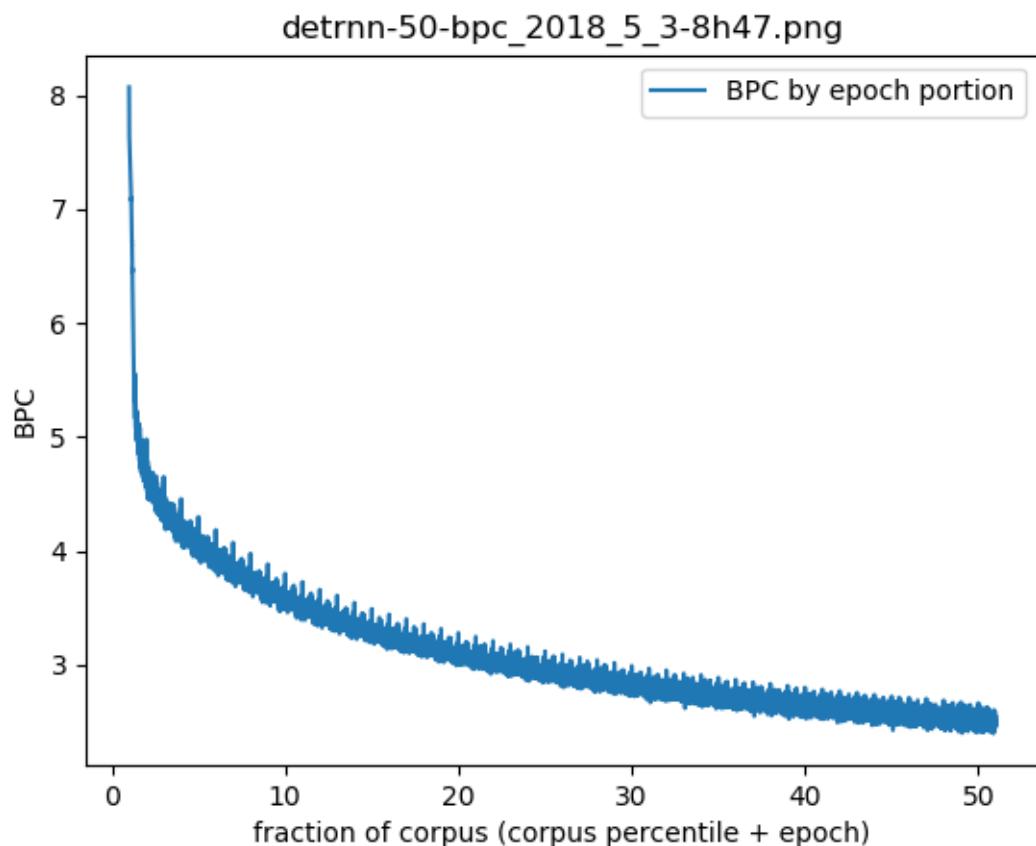


FIGURE A.2 – BPC

ValBPC/epoch Mean BPC over the epoch, at the end of each epoch.

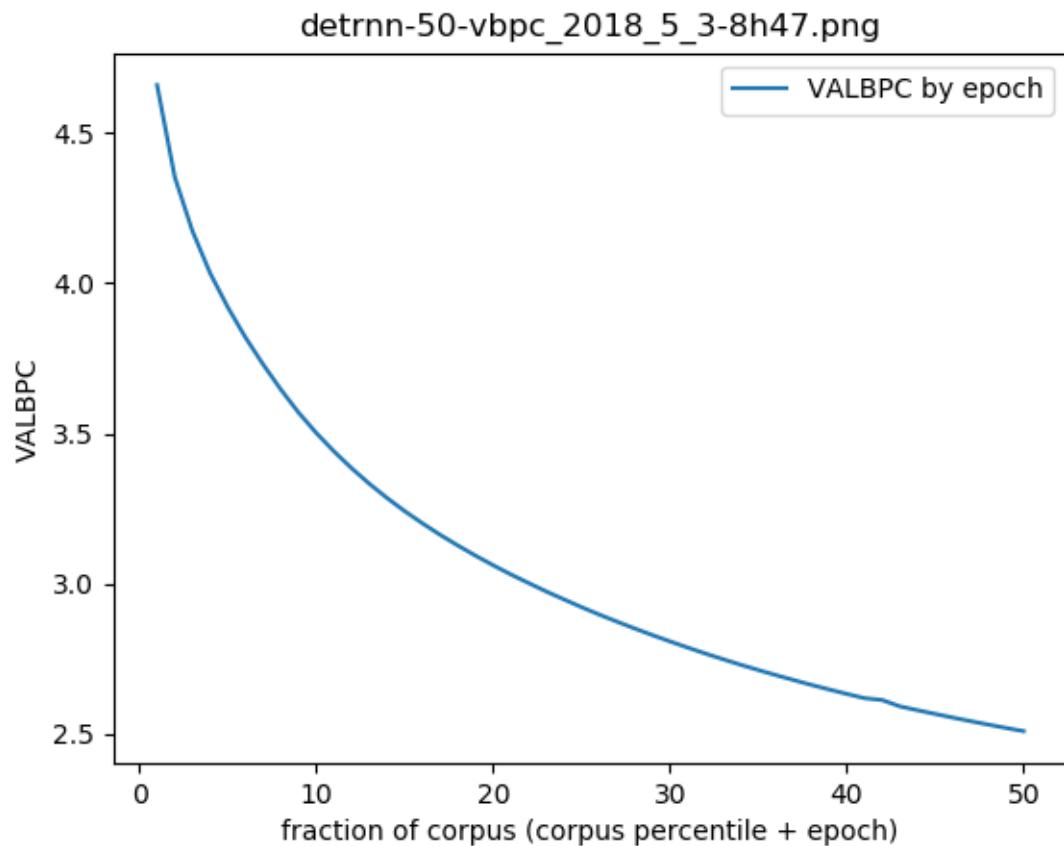


FIGURE A.3 – ValBPC

Loss Loss per fraction of the corpus (an interval of 1 correspond a complete corpus, or an epoch).

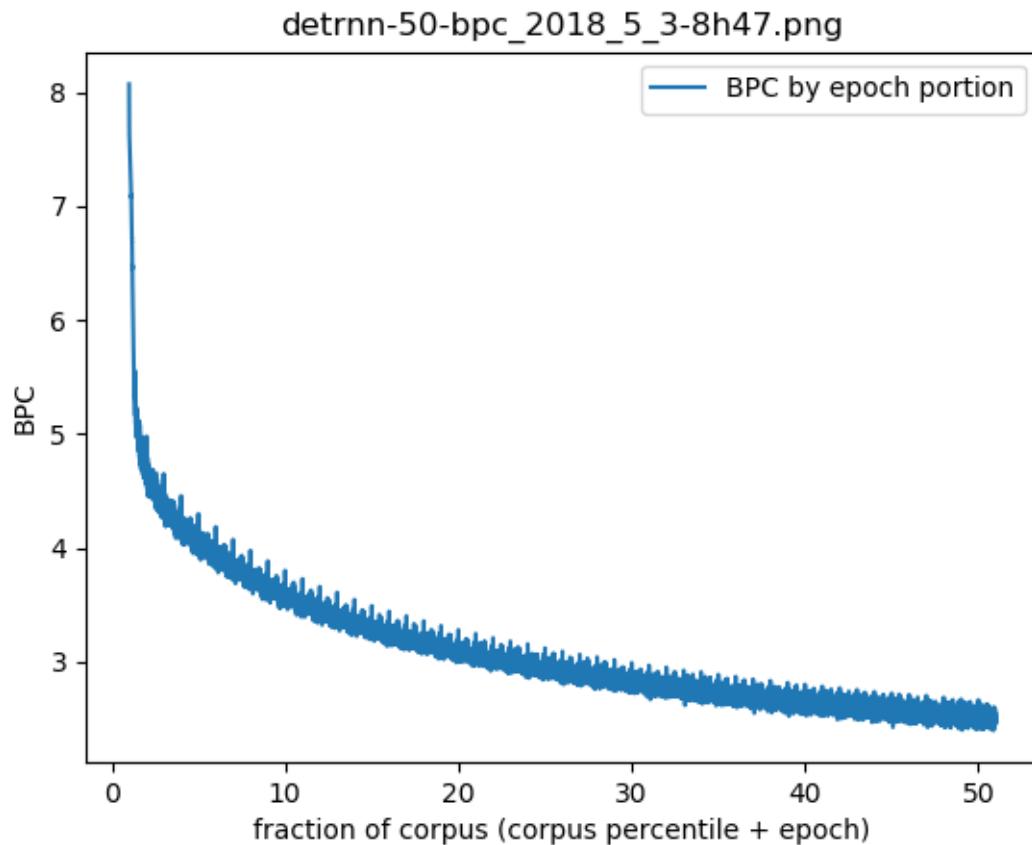


FIGURE A.4 – Loss

Logs

The log is available at https://gitlab.inria.fr/emarquer/awd-lstm-lm/blob/reimplement/logs/detrnn-50_2018_4_30-12h2.log.

A.10.4 Potential ameliorations & next steps

Next step is to test the growing model.

A.11 Premier test du modèle multi-échelles

Test run of `detrnn.py`

Test report

by E. Marquer, 2018/05/03, Synalp and Université de Lorraine

A.11.1 Abstract

First performance test of the Test to do a run on 4 epochs, with GPU, of the basic model of MSNN, with additive output strategy.

A.11.2 Paradigm

This test run of `detrnn.py`, with DEBUG level log output, loss per percentile and vbpc per epoch, that will be executed with `cuda`, for 4 epochs.

The test is done with branch `growing`, an allocated time of 4h, not interactive.

Run time was estimated for 4 epochs according to a debug results for 0.2 epochs :

1 (10 min / 0.2 epoch) * 4 epoch = 200 min = 3h 20min

With a security margin of 40min, run time is 4h.

/\ Had to reduce batchsize down to 40 and halve hidden size because of memoryerrors */*

Hyperparameters

Hyperparameter	Value
nhidden	920
embedsize	400
bptt	200
batch_size	40
eval_batch_size	32
lr	0.001
wdecay	1.2e-06
cuda_on	True
log_interval	20
nepochs	4
max_seqs	15

Node

OAR_JOB_ID=1558426 with GPU

Job start time : 2018-05-04 14:43:40

Estimated job stop time : 2018-05-04 18:43:40

Command used :

```
1 oarsub -q production -p "GPU <> 'NO'" -l "nodes=1,walltime=04:00:00" ~/alt-
repo/awd-lstm-lm/rundet.sh
```

Status verification loop :

```
1 let x=0; while [ "true" ]; do echo "$x" $(oarstat -s -j 1558141); let ++x;
sleep 120; done
```

A.11.3 Results

Total run time for 4 epochs : with real stop time of 2018-05-03 17:57:26, the total run time of the training is approximately 3h 13min, corresponding to the predicted 50 min per epoch.

Comparative analysis

With half the number of hidden parameters, and a yet unknown number of layers, the basic MSNN has very similar results than the classical DetRNN.

However, when analysing closely the learning speed, the MSNN seems to be starting with a slower BPC decrease than the DetRNN, and it also seem to be faster later on. Those variations are probably due to the number of hidden layers.

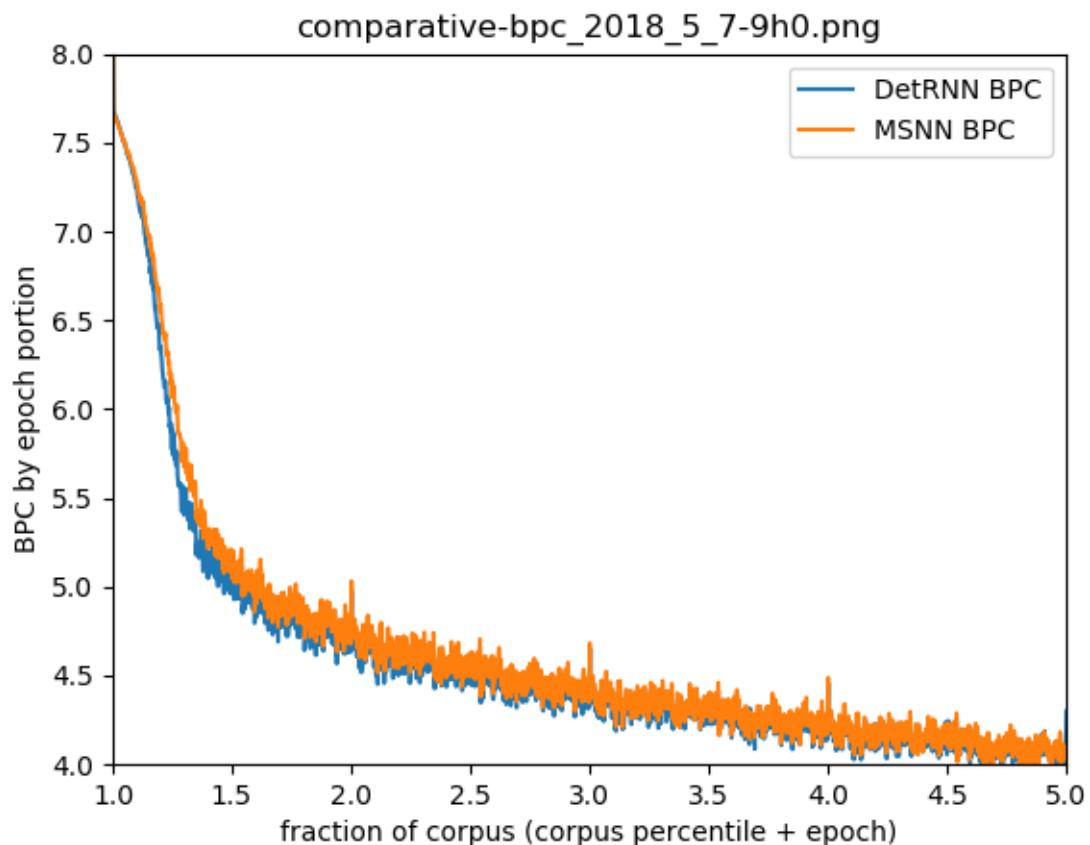


FIGURE A.5 – Comparative BPC

Plot

BPC/fraction of corpus BPS per fraction of the corpus (an interval of 1 correspond a complete corpus, or an epoch).

ValBPC/epoch Mean BPC over the epoch, at the end of each epoch.

Loss Loss per fraction of the corpus (an interval of 1 correspond a complete corpus, or an epoch).

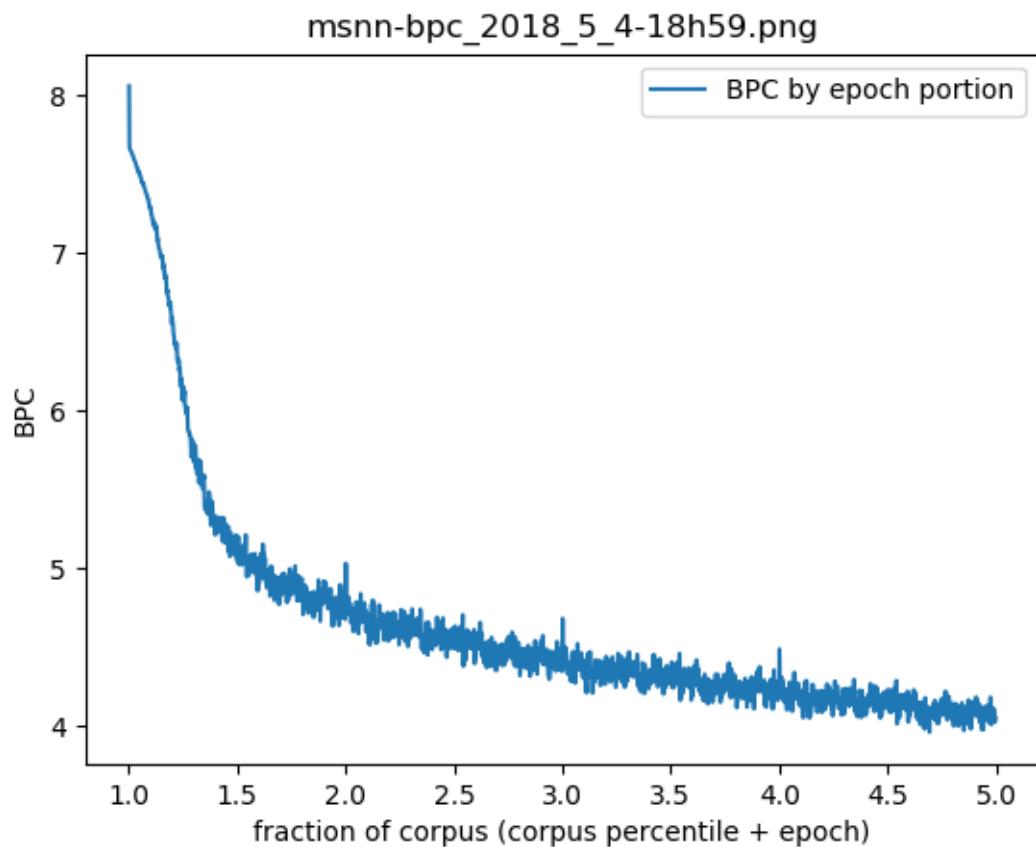


FIGURE A.6 – BPC

Logs

Reduced log is available at [msnn-base/msnn_2018_5_4-14h43.log](#).

A.11.4 Potential ameliorations & next steps

A necessary amelioration is to add a way to track the number of layers.

As of now, the upper hidden layers participates in the output only when updated. It is necessary to make them participate at every step.

The test process is not well defined : what to do when the eval batch is discontinued from the training batch ? what if it is in the same corpus, but not directly adjacent ? A possible yet hazardous solution would be to evaluate a “distance” between the training and evaluation batches, and reset the hidden states depending on that distance (a higher distance would reset a higher number of layers).

Lastly, as the number of values to remember is increasing (bpc, loss, layer number, ...) it would be interesting to improve the .plotdata system.

Next step is to test the recurrently defined growing model.

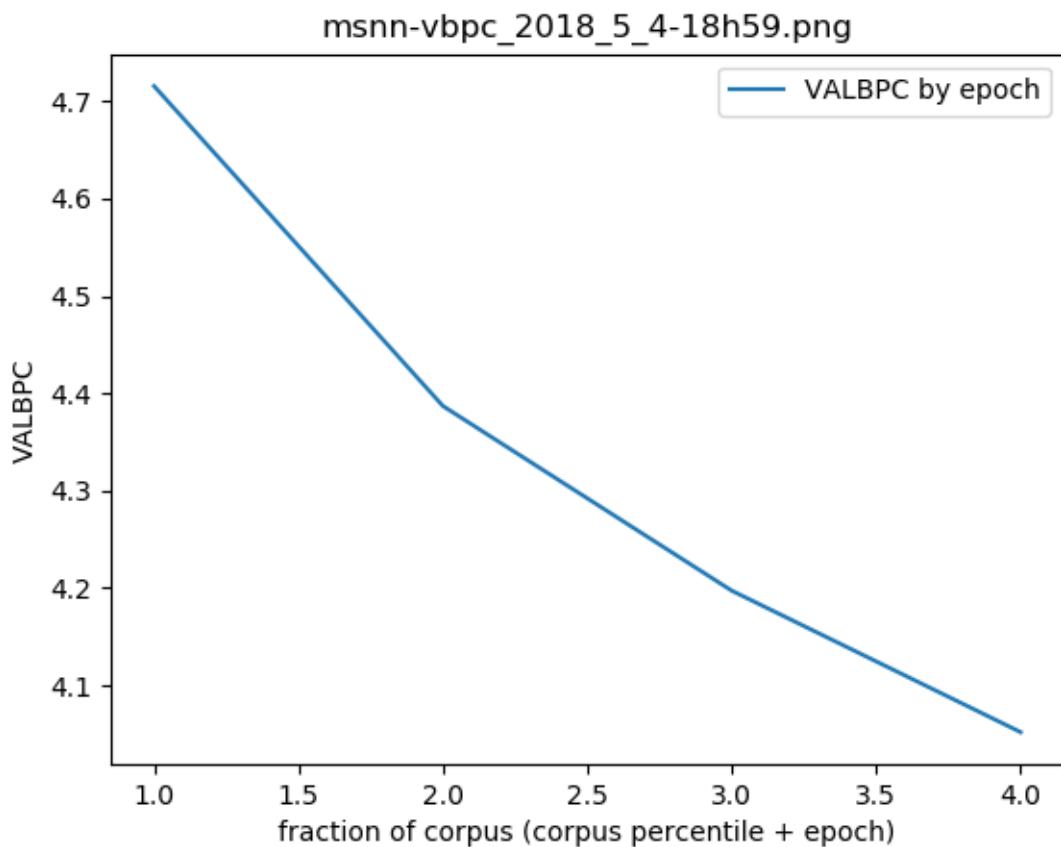


FIGURE A.7 – ValBPC

A.12 Comparaison des stratégies de fusion des résultats des différentes couches

Test run of `detmsnn.py`

Test report

by E. Marquer, 2018/05/16, Synalp and Université de Lorraine

A.12.1 Abstract

Performance test of the ‘cat’ (concatenated) output strategy compared to the ‘add’ strategy. Test to do a run on 4 epochs, with GPU, of the basic model of MSNN, with concatenated output strategy.

A.12.2 Paradigm

This test run of `detmsnn.py`, with INFO level log output, loss per percentile and vbpc per epoch, is executed with *cuda*, for 4 epochs.

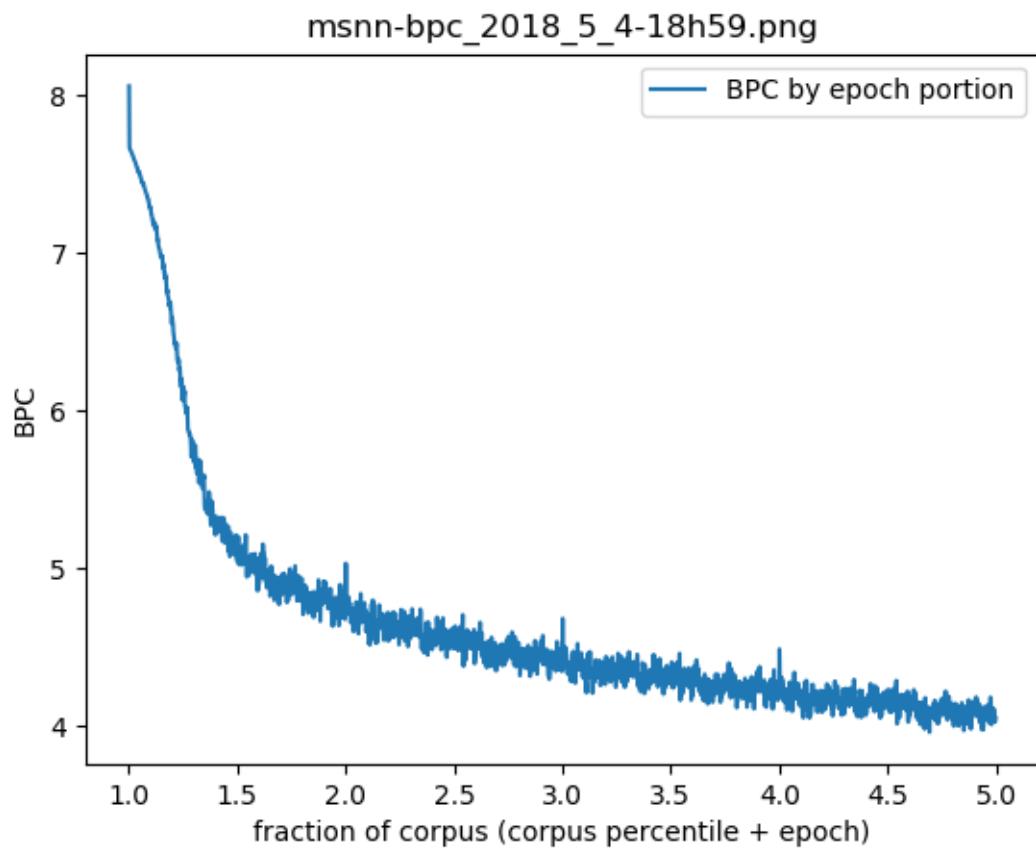


FIGURE A.8 – Loss

The test is done with branch **growing**, an allocated time of 24h, not interactive

/!\ Had to reduce evaluation corpus size down to 1/1000, to reduce computation time while keeping a big enough corpus to compute BPC /!

Hyperparameters

Hyperparameter	Value
nhidden	920
embedsize	400
bptt	200
batch_size	1
eval_batch_size	1
lr	0.001
wdecay	1.2e-06
cuda_on	True
log_interval	100
nepochs	4
max_seqs	5

Node

OAR_JOB_ID=1563805 with GPU grimani-1

Job start time : 2018-05-16 08 :53 :20

Estimated job stop time : 2018-05-17 08 :53 :20

Command used :

```
1 oarsub -q production -p "GPU <> 'NO'" -l "nodes=1,walltime=24:00:00" "bash runmsnn.sh"
```

Status verification loop :

```
1 let x=0; while [ "true" ]; do echo "$x" $(oarstat -s -j 1563805); let ++x; sleep 120; done
```

A.12.3 Results

Total run time for 4 epochs : with real stop time of 08 :53 :28, the total run time of the training is approximately 24h, with only 22% of one epoch done, and a final Validation BPC of 3.57.

Estimated run time for a full epoch : 24h / 22% \approx 109h/epoch. This corresponds to 436h for a 4 epoch run, and this is critical.

The ‘cat’ strategy is way more efficient in corpus consumption, even if it is dramatically slower than the additive strategy.

Comparative analysis

The comparative plot shows that with the ‘cat’ strategy, BPC diminution is way faster than with the additive strategy. Computation-time wise, it is obvious that the ‘cat’ strategy is slower, with ? ?h/epoch, than the ‘add’ strategy, with 50min/epoch. This difference is too large to be due to the device alone.

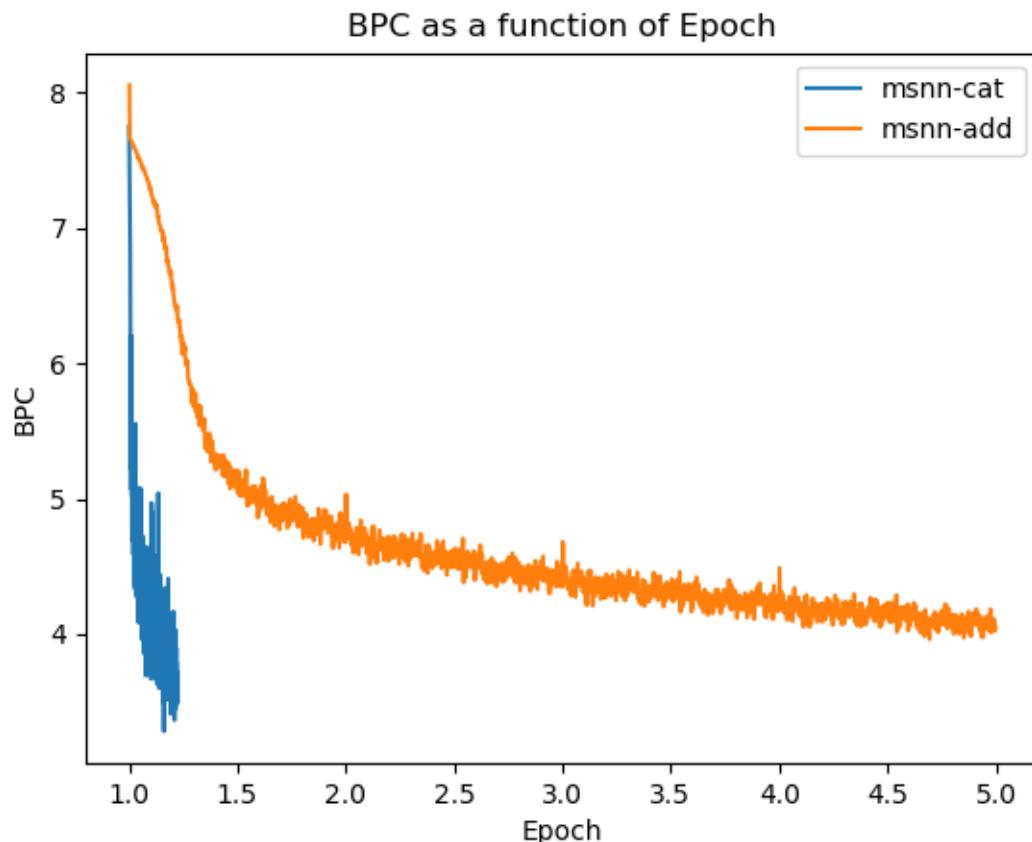


FIGURE A.9 – Comparative BPC

Plot

BPC/fraction of corpus BPC : BPC per fraction of the corpus (an interval of 1 correspond a complete corpus, or an epoch).

Validation BPC : BPC per fraction of the corpus, on the validation corpus.

Layers : Number of layers per fraction of the corpus.

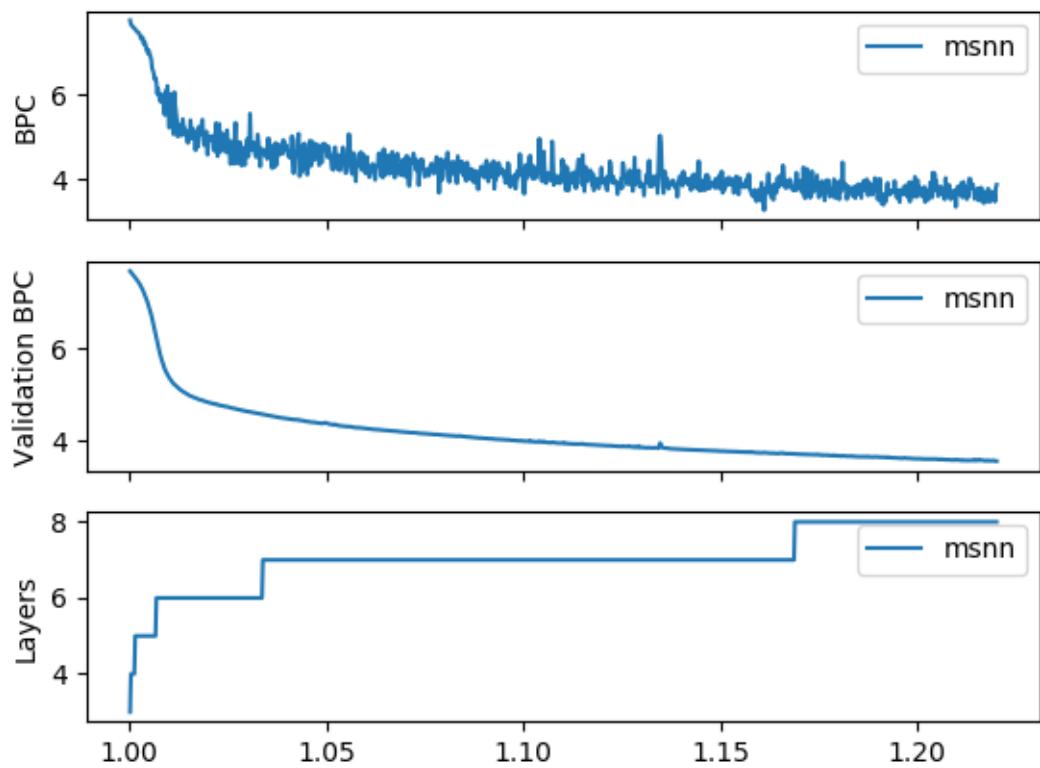


FIGURE A.10 – Comparative BPC

A.12.4 Potential ameliorations & next steps

Next step is to try to reduce run time.

A.13 Test des effets du changement de taille des paquets (batch)

Test run of detmsnn.py

Test report

by E. Marquer, 2018/05/16, Synalp and Université de Lorraine

A.13.1 Abstract

Performance test of batch size variation without deeper . Test to do a run on 4 epochs, with GPU, of the basic model of MSNN, with concatenated output strategy.

A.13.2 Paradigm

This test run of *detmsnn.py*, with INFO level log output, loss per percentile and vbpc per epoch, is executed with *cuda*, for 4 epochs.

The test is done with branch *growing*, an allocated time of 24h, not interactive

**/!\ Had to reduce evaluation corpus size down to 1/1000, to reduce computation time while keeping a big enough corpus to compute BPC /!\
!**

Hyperparameters

Hyperparameter	Value
nhidden	920
embedsize	400
bptt	200
batch_size	16
lr	0.001
wdecay	1.2e-06
cuda_on	True
log_interval	100
save_interval	100
nepochs	4
max_seqs	5

Node

OAR_JOB_ID=1567202 with GPU grimani-1

Planned job start time : 2018-05-19 02:41:08 Job start time : 2018-05-19 02:41:08

Estimated job stop time : 2018-05-22 14:41:08

Command used :

```
1 oarsub -q production -p "GPU <> 'NO' " -l "nodes=1,walltime=84:00:00" "bash  
runmsnn.sh"
```

Status verification loop :

```
1 let x=0; while [ "true" ]; do echo "$x" $(oarstat -s -j 1567202); let ++x;  
sleep 120; done
```

A.13.3 Results

Total run time for 4 epochs : with real stop time of ?, the total run time of the training is approximately ?h, with only, and a final Validation BPC of 3.57.

Comparative analysis

NO PLOT HERE

Plot

BPC/fraction of corpus BPC : BPC per fraction of the corpus (an interval of 1 correspond a complete corpus, or an epoch).

Validation BPC : BPC per fraction of the corpus, on the validation corpus.

Layers : Number of layers per fraction of the corpus.

NO PLOT HERE

A.13.4 Potential ameliorations & next steps

Next step is to continue run time reduction.

A.14 Entraînement sur le corpus complet avec beaucoup de temps alloué

Long-run of RNN-MSNN

Test report

by E. Marquer, 2018/05/29, Synalp and Université de Lorraine

A.14.1 Abstract

The run was done on the reduced enwik8 corpus.

The test is composed of 4 successive runs :

- 1 run of 2h on grimani-4;
- 2 runs of 12h, both on grimani-1;
- 1 run of 50h on grele-11;

End causes are as follow :

- Run 1 : out of time (2h);
- Run 2 : out of time (12h);
- Run 3 : end of epoch crash (7h30);
- Run 4 : end of epoch crash (19h15);

Mean time for an epoch is about 19h 15min (on the reduced version of the corpus). Two epochs were completed.

A.14.2 Results

Each run crashed between epochs, so a bit of patching had to be made on top of fixing the bug.

Memory

Both RAM and video RAM are still subject to a constant leak in memory. But even if it does not show on the plots (scale is too small), logs confirm that there is no leak during validation.

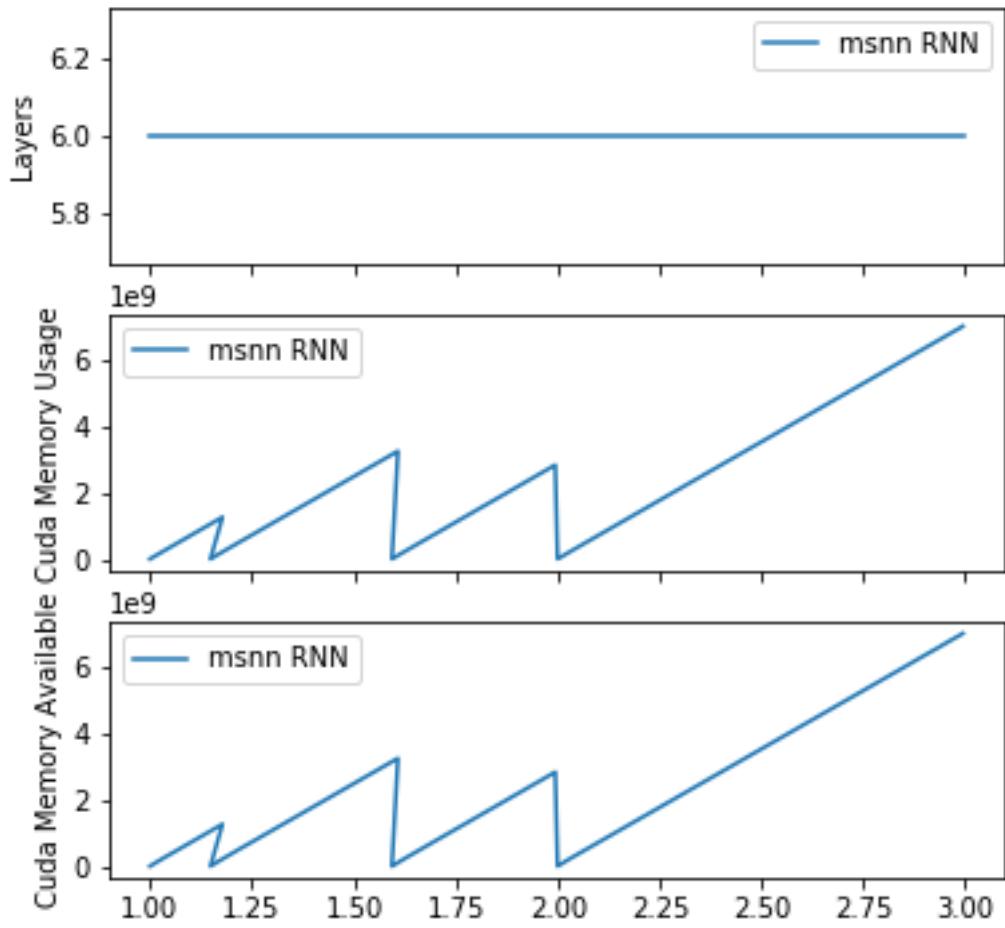


FIGURE A.11 – Memory usage

```
ida3/envs/pytorch/bin/python msnr_starter.py --save-folder logs/long-run_2018-07-06/ --cuda-on --resume-model logs/long-run_2018-07-06/models/.fullmodel
```

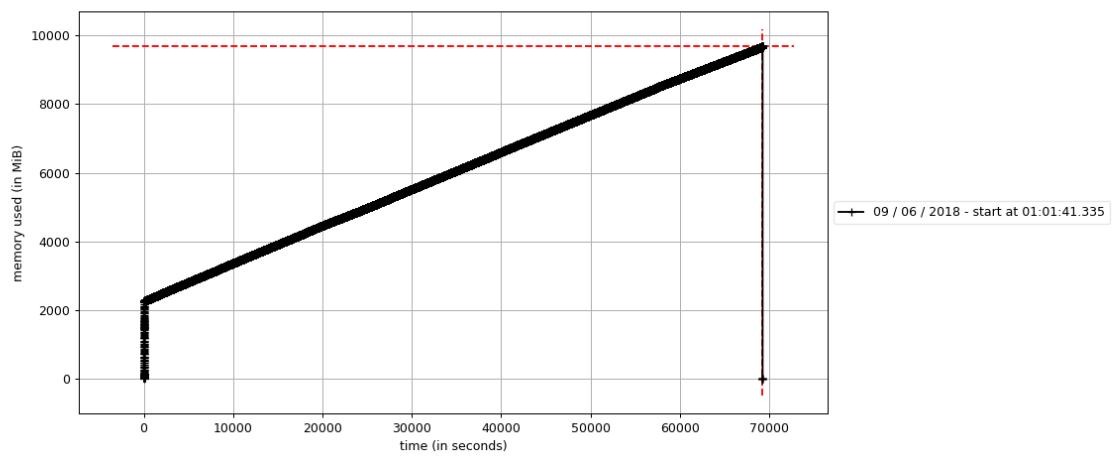


FIGURE A.12 – RAM third run

An other noticeable property is that “Run Time”, corresponding to the time to train over \log_{interval} sequences, is mostly proportional to CUDA memory usage. The source of the cuda memory leak is probably the same as what makes training slower.

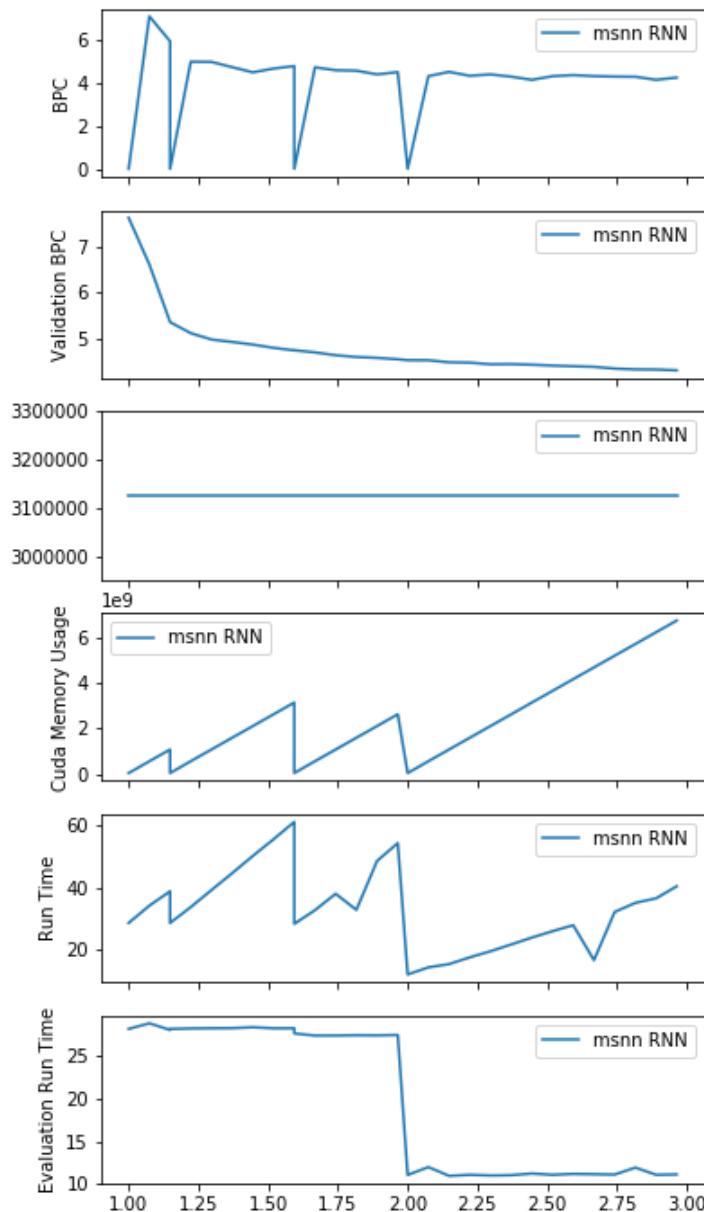


FIGURE A.13 – Memory and computation time

BPC / Validation BPC

BPC and Validation BPC

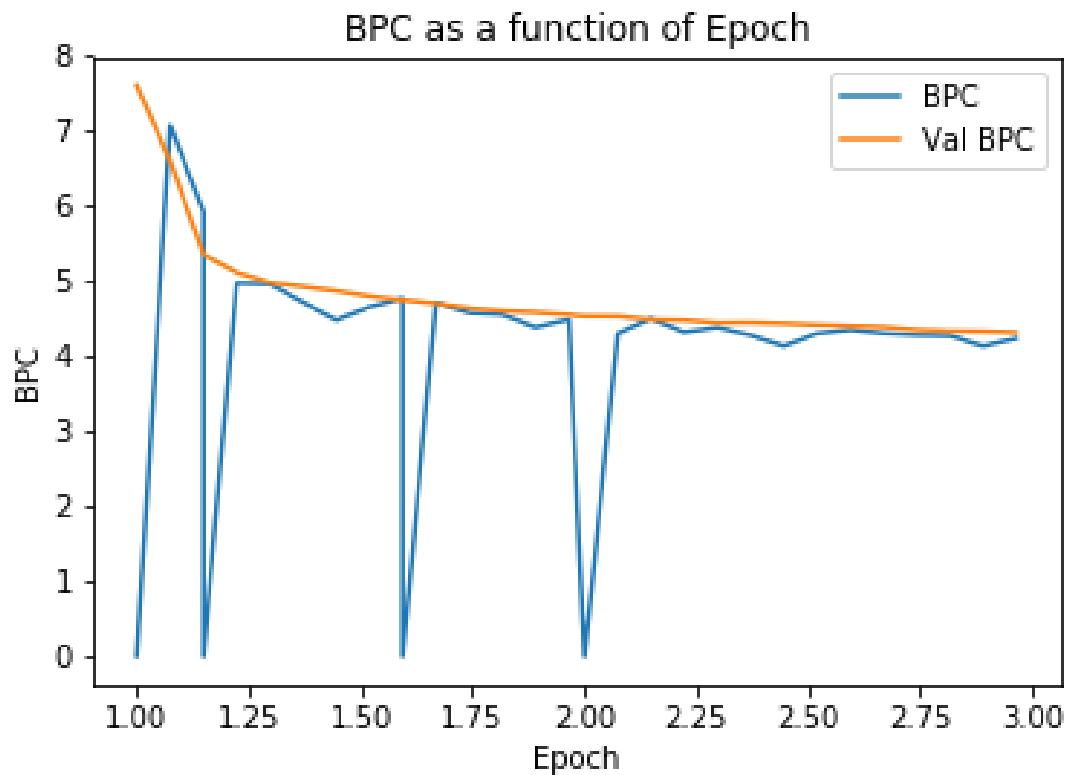


FIGURE A.14 – BPC

Job restart

When resuming a job, CUDA memory is entirely freed. Same thing can be said about RAM.

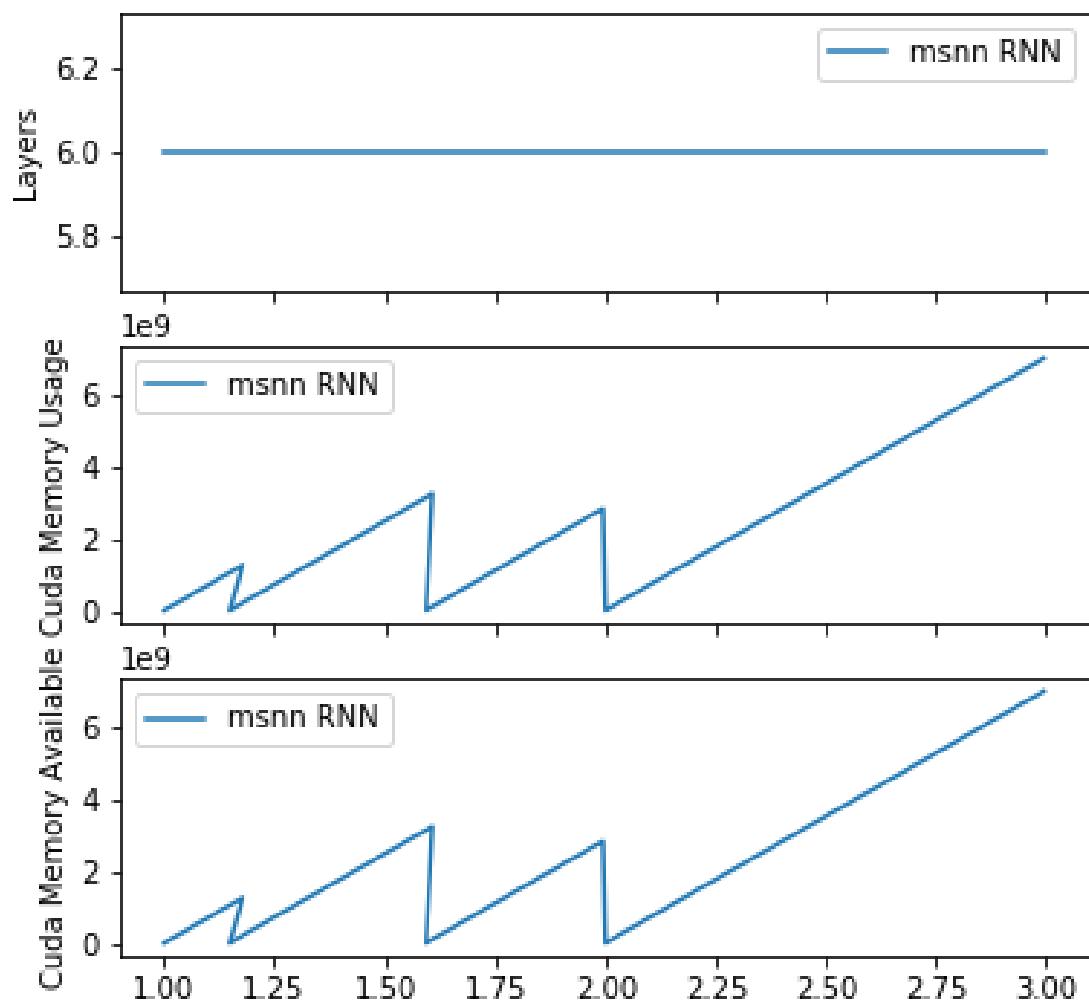


FIGURE A.15 – Memory usage

```
/home/emarquer/miniconda3/envs/pytorch/bin/python msnn_starter.py --save-folder logs/long-run_2018-07-06/ --cuda-on
```

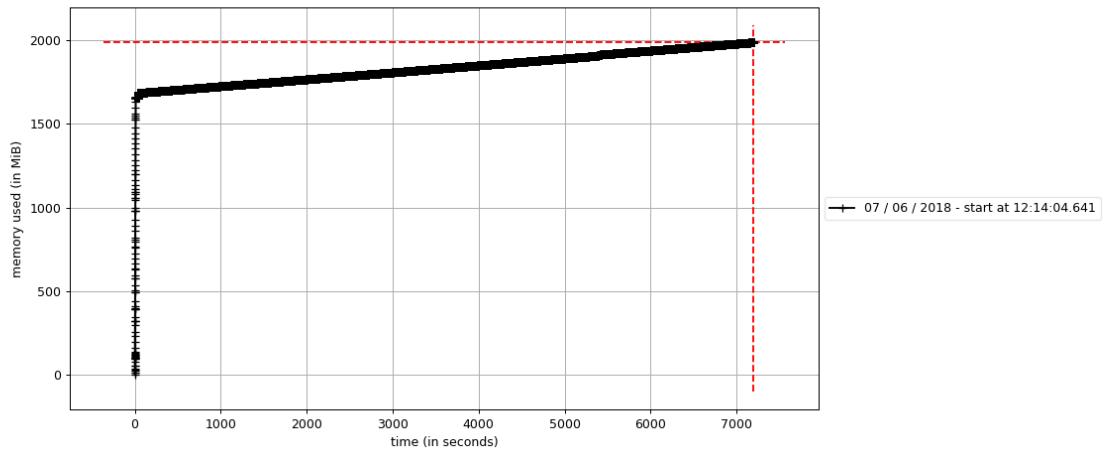


FIGURE A.16 – RAM first run

```
1da3/envs/pytorch/bin/python msnn_starter.py --save-folder logs/long-run_2018-07-06/ --cuda-on --resume-model logs/long-run_2018-07-06/models/.fullmodel
```

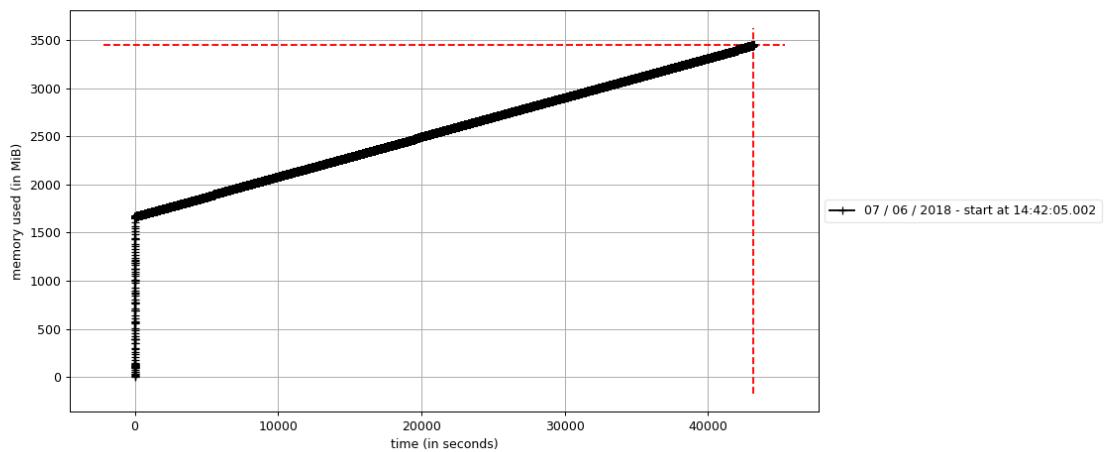


FIGURE A.17 – RAM second run

```
1da3/envs/pytorch/bin/python msnn_starter.py --save-folder logs/long-run_2018-07-06/ --cuda-on --resume-model logs/long-run_2018-07-06/models/.fullmodel
```

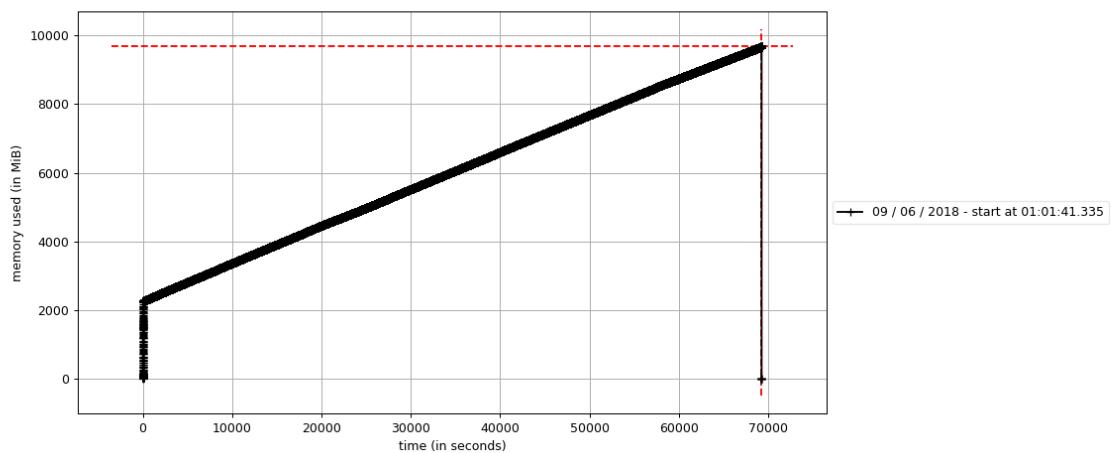


FIGURE A.18 – RAM third run

Memory is freed each time the job is restarted, meaning either a part of the necessary is discarded, or unnecessary data is kept in memory. As in CUDAles tests a memory maximum was reached, CUDA seems to be the source of the leak (data copies not removed, ...).

A.14.3 Next steps

Debug end of epoch bug. Try to patch memory leak. Continue training.

A.15 Changement de stratégie de gestion d'historique

Change history

Analysis report

by E. Marquer, 2018/06/12, Synalp and Université de Lorraine

A.15.1 Abstract

As computation graph has been confirmed to exist/be considered only in the last history, keeping an explicit history is no longer necessary. Worst, it may hinder garbage collection, by keeping references to Tensors that are completely useless.

A new model of history has been made to keep only a reference to the last hidden state. This new model will be referred as “last”, and the old one keeping an explicit history as “classical”.

A.15.2 Tests

Multiple tests were done on the new “enwik8mini” corpus of 100,000 characters :

- “last” a test run on 2 epochs with 1 batches of “last” history
- “last b2” a test run on 10 epochs with 2 batches of “last” history
- “last b2 10epoch” a test run on 10 epochs with 2 batches of “last” history
- “classical” a test run on 2 epochs with 1 batches of “classical” history

Results (1 epoch)

The results are over 1 epoch, with values from the start of the first and the second epoch. As “last b2 10epoch” and “last b2” have the same set of parameters, they coincides.

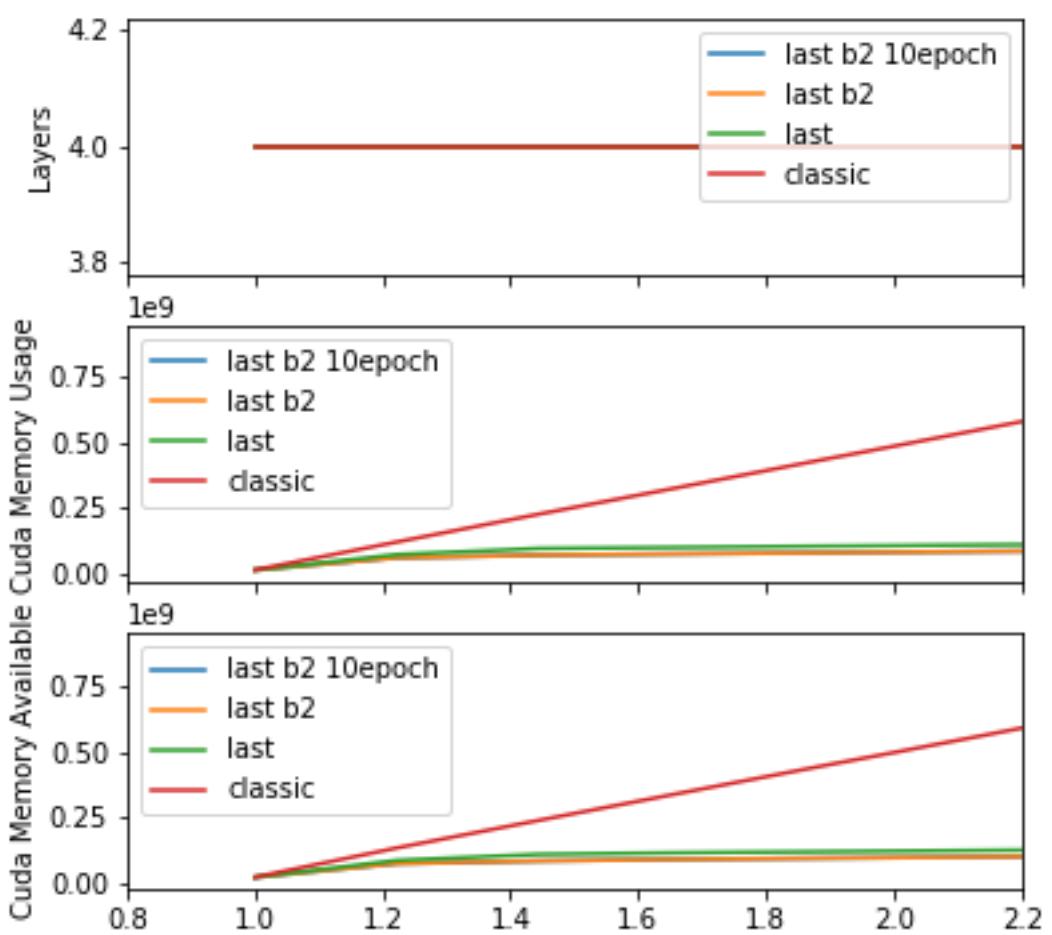


FIGURE A.19 – Memory 1 epoch

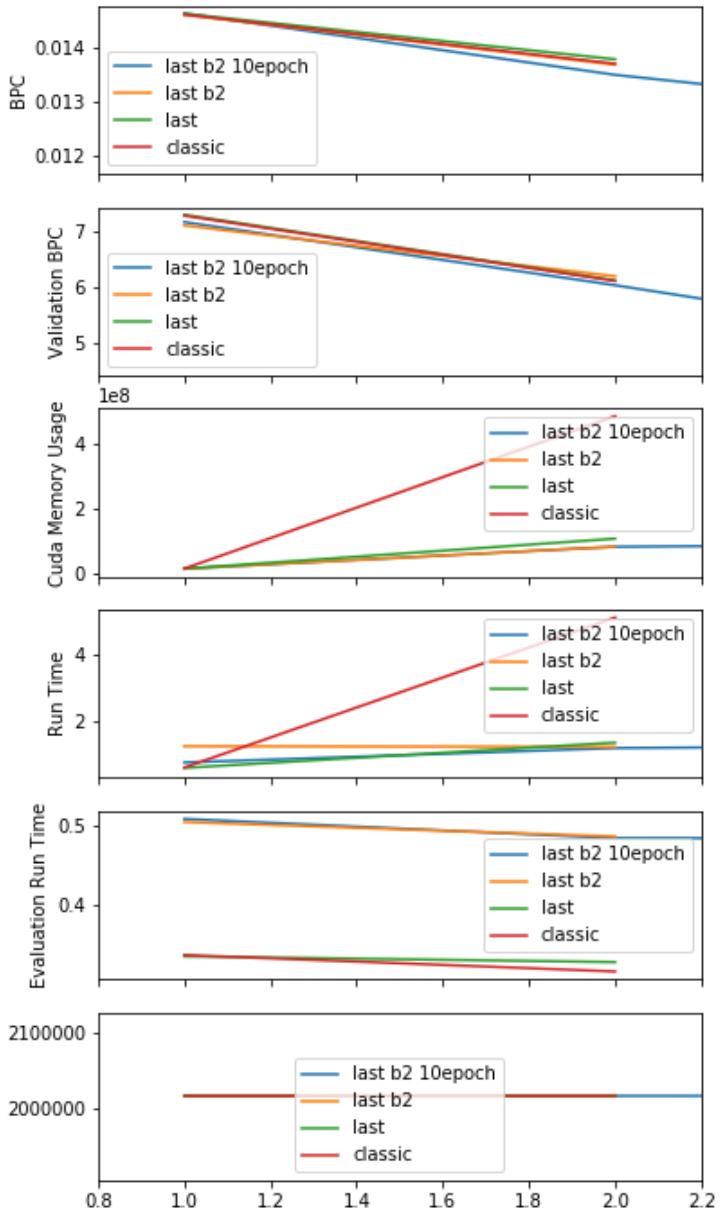


FIGURE A.20 – All info 1 epoch

Results (full)

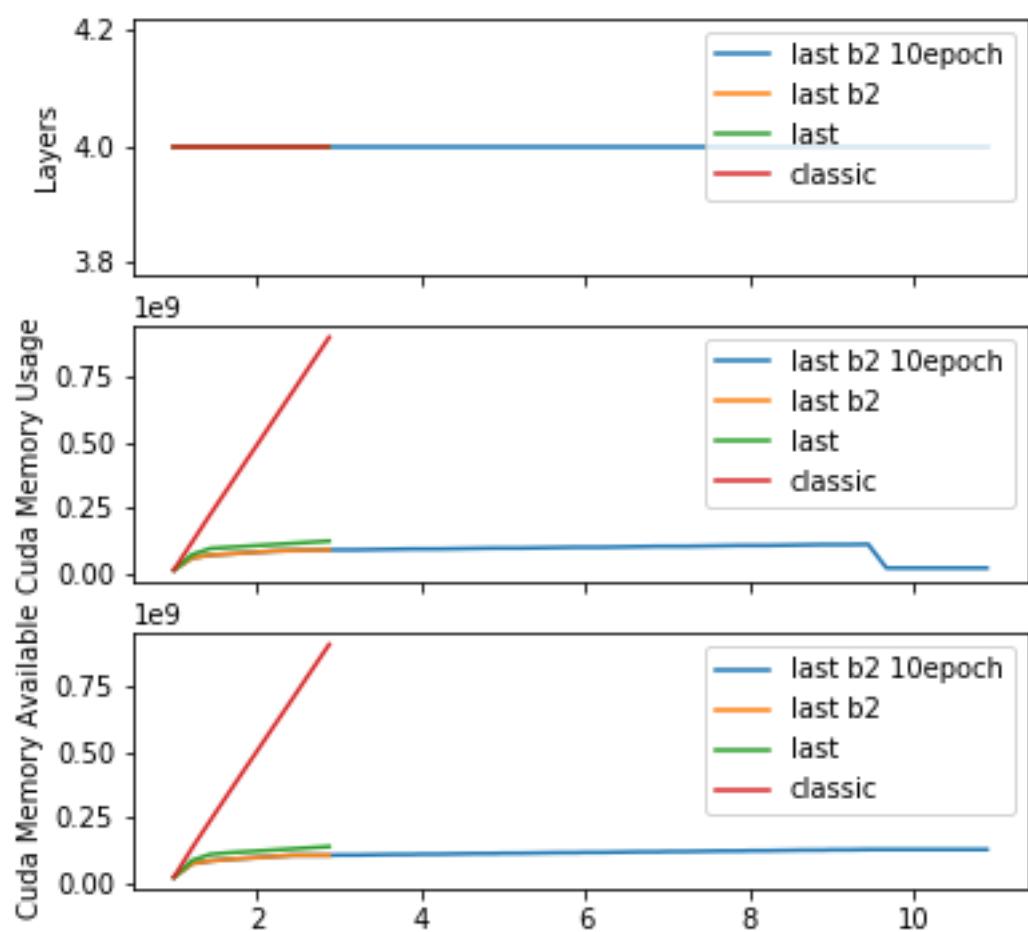


FIGURE A.21 – Memory

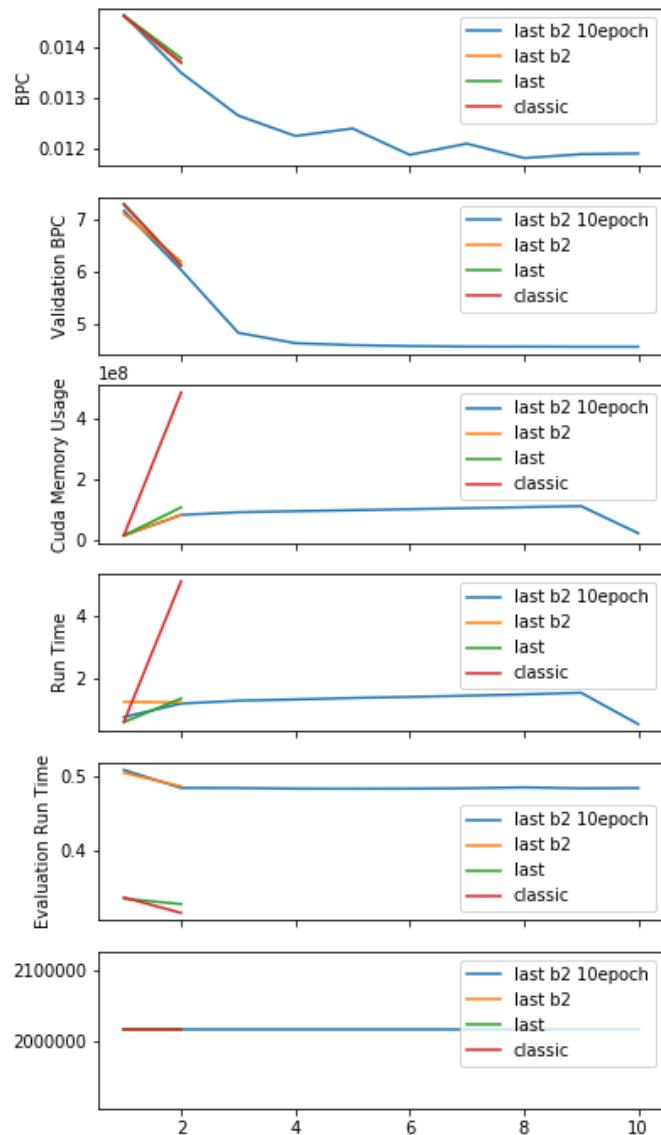


FIGURE A.22 – All info

```
quer/miniconda3/envs/pytorch/bin/python msnn_starter.py --save-folder logs/test4/ --cuda-on --corpus data/enwik8mini --batch-size 2 --epochs 10
```

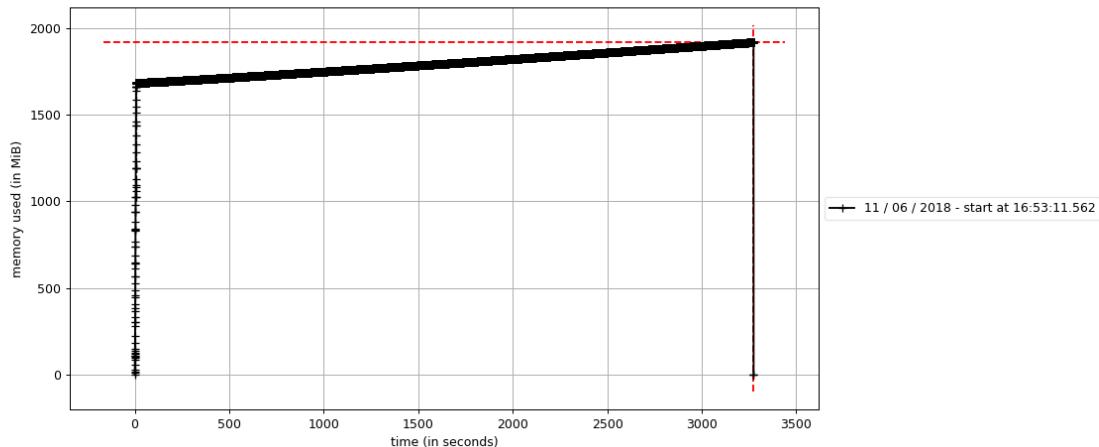


FIGURE A.23 – RAM 10 epoch

Ram consumption for “last b2 10epoch”

A.15.3 Conclusion

Memory leak

The leak in CUDA memory has reduced drastically.

It seems there is still a leak, this leak is not present when CUDA is not used. This leak is present in both RAM and graphical RAM, but the remaining leak in graphical RAM is not a concern anymore, with only a dozen MiB over 1,000,000 characters (10 epochs * 100,000 characters). However, the leak in RAM has not reduced, and even if it is not a problem thanks to the available RAM in the cluster, it would be preferable to identify the source of the leak.

Run (training) time

Additionally, the correlation of CUDA memory usage and training run time is kept, so the small remaining CUDA leak is probably linked to the computation graph. With CUDA memory usage drastically reduced, run time has been reduced too.

Currently, training over 100,000 characters (with 2 batches) only takes 5 minutes.

Performances

No conclusion can be drawn on learning performances, due to the size of the learning corpus. Even though, it is encouraging that even with a small corpus 10 epochs are not enough to have over-fitting (at least with 2 batches).

A.15.4 Next steps

1. long training over the “enwik8reduced” or the “enwik8” corpus;
2. fix (or at least identify) RAM leak

A.16 Test de l'implémentation des *batchs*

Long-run of RNN-MSNN

Test report

by E. Marquer, 2018/06/13 Synalp and Université de Lorraine

A.16.1 Abstract

The test is composed of 2 successive runs :

- id 1582586 : batch-size 1, bptt 200 on grele-4;
- id 1582587 : batch-size 2, bptt 100 on grimani-6.

Run time is about 10h in each case, corresponding to 2h30 for an epoch. In both case corpus batches rotation over epochs was disabled.

Shared parameters

parameter	value
corpus	enwik8reduced
history_strategy	layer-constant-length
max_history	25
bptt	<i>variable</i>
batch_size	<i>variable</i>
epochs	4
lr	1e-3
weight_decay	1.2e-6
epoch	4
valid_len	500,000
log_interval	500
save_interval	500
memory_interval	100
hidden_size	460
embed_size	400
growth_factor	5
rnn_type	RNN
reset_hidden	False

parameter	value
reset_growth	True
cuda_on	True

A.16.2 Results

At the end of each epoch, we see a spike in BPC, due to the first evaluation of the epoch. As corpus rotation is disabled, it is not surprising that with two batches over-fitting appears.

Moreover, we can note that run time is constant and memory usage tend to a constant value (1.6 GiB), with no difference between 1 and 2 batches. Note : the product bptt * batch_size is equal for 1 and 2 batches, so this result is the one predicted by the equations.

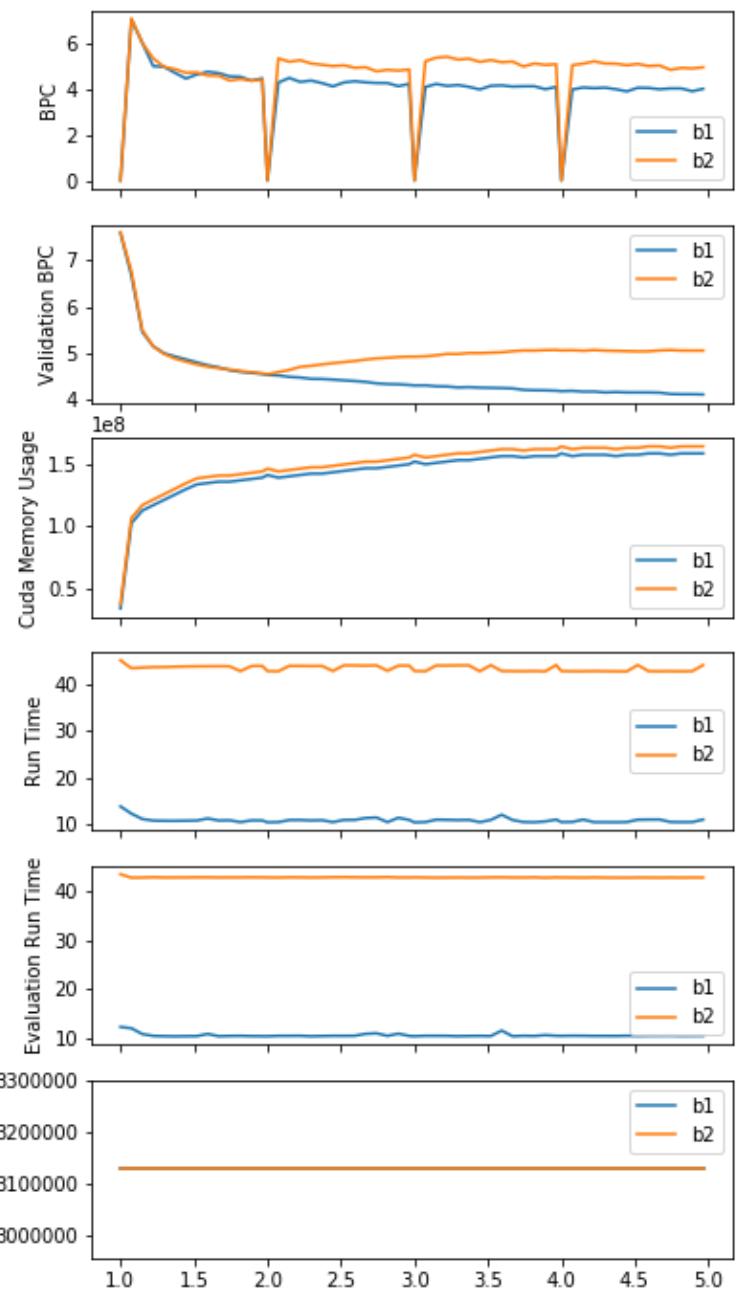


FIGURE A.24 – RAM third run

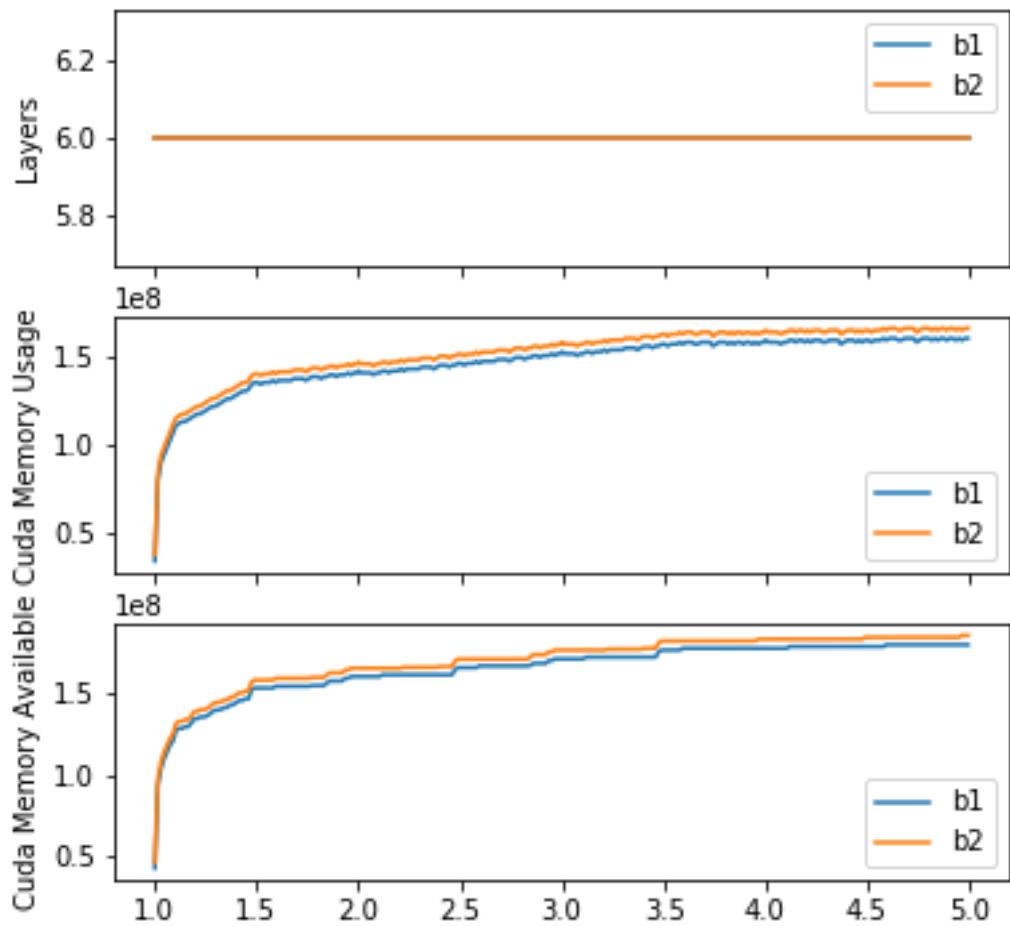


FIGURE A.25 – Memory usage

A.16.3 Next steps

- batches :
 1. Implement corpus rotation
 2. See if corpus rotation solves over-fitting
 3. Compare run time on comparable machines
- long run :
 1. Continue batch 1 for more epochs
 2. See when over-fitting appears

A.17 Test des performances des *batchs*

Run on the same node of RNN-MSNN with different batch size

Test report

by E. Marquer, 2018/06/15, Synalp and Université de Lorraine

A.17.1 Abstract

The test is composed of 2 runs :

- id 1583339 : batch-size 1, bptt 200 on grele-2
- id 1583336 : batch-size 2, bptt 100 on grele-1

Run time per epoch varies from 28 to 20 min for a single batch and from 20 to 11 min for two batches.

Shared parameters

parameter	value
corpus	enwik8reduced
max_history	25
bptt	<i>variable</i>
batch_size	<i>variable</i>
epochs	4
lr	1e-3
weight_decay	1.2e-6
epochs	4
valid_len	500,000
log_interval	500
save_interval	500
memory_interval	100
hidden_size	460
embed_size	400
growth_factor	5
rnn_type	RNN
reset_hidden	False

parameter	value
reset_growth	True
cuda_on	True

A.17.2 Results

At the end of each epoch, we see a spike in BPC, due to the first evaluation of the epoch (BPC is reinitialised to 0, causing a spike). With 2 batches, BPC is also more stable.

As of now, run time is computed after running validation, so training time is `real_training_time = run_time - validation_time`. But validation time increase with the number of batches as the number of characters seen increase (the whole validation corpus is used for each batch, so the number of characters seen during validation is `validation_corpus_len * batches`). That explains that the evaluation time for a batch is lower than for 2 batches.

By removing evaluation time, we can obtain a reasonable and coherent (with regard to epoch run time) estimation of training time over a sequence.

Epoch run time :

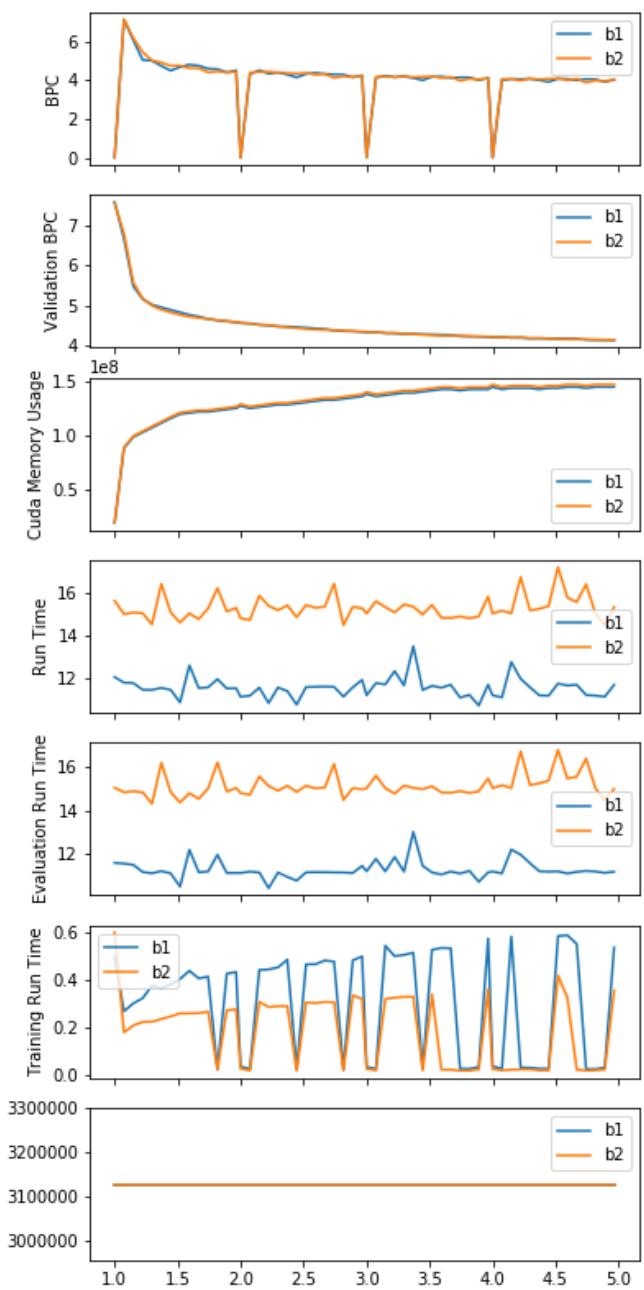
Epoch	Run time b=1	Run time b=2
1	28 min	20 min
2	23 min	17 min
3	22 min	14 min
4	20 min	11 min

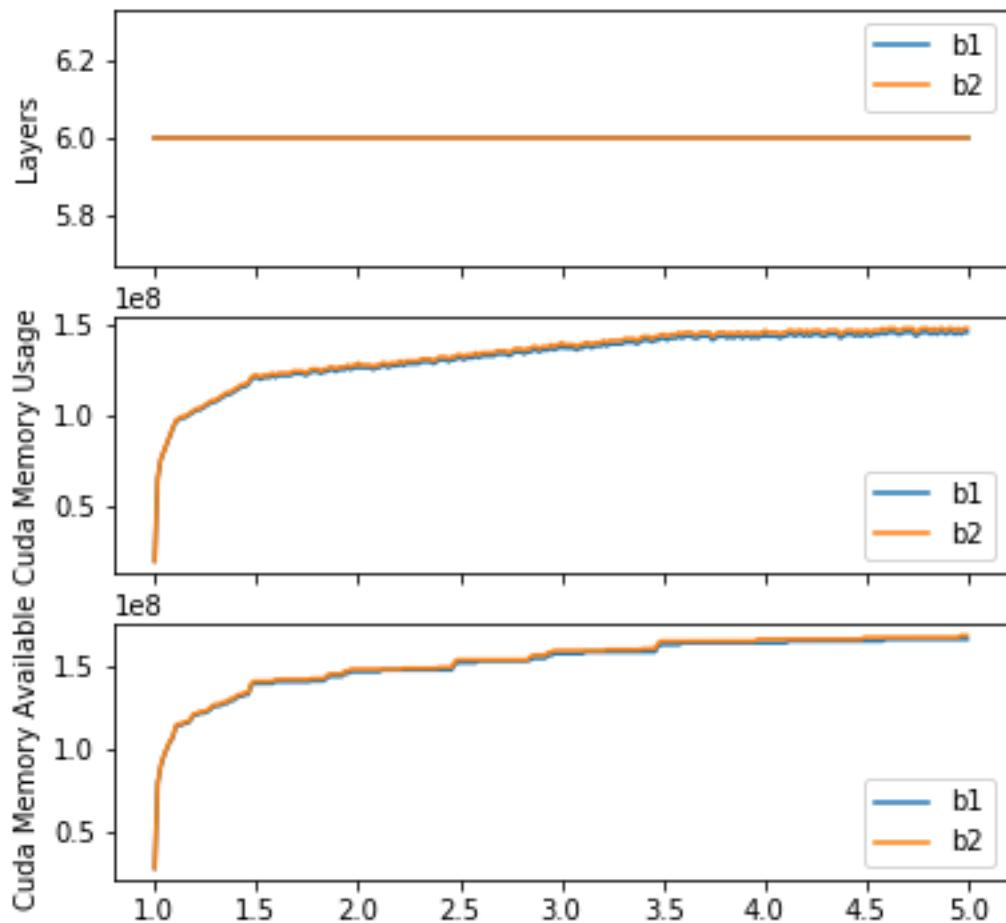
There is a notable decrease in epoch run time (time necessary to run over an epoch), of about 10 min over the 4 epochs, with both runs.

Possible causes for the decrease of run time : - part of the graph is already computed, and this part is skipped; - corpus data is already loaded in cuda memory.

Both of them are highly unlikely to cause such a decrease. A more precise training time storage may provide an explanation.

Plots





A.17.3 Next steps

- Data
 - Implement more precise time data saving (epoch run time and training run time)
 - Runs (objective : 100 epochs)
 - Continue running batches 1 and 2
 - Try with higher number of batches
 - [Optional] Other experimental branch
1. Implement prepared attention module in every layer
 2. Analyse results
 3. Patch probable memory leaks

A.18 Test des performances des *batchs* sur 50 époques

Run over more than 50 epochs with varied batch size

Test report

by E. Marquer, 2018/06/19, Synalp and Université de Lorraine

A.18.1 Abstract

The test is composed of 4 runs on grele, with :

- bptt 200, batch-size 1
- bptt 100, batch-size 2
- bptt 50, batch-size 4
- bptt 25, batch-size 8

Run has been interrupted by an overflow of disk space due to the detailed logs ; next runs will used reduced logs.

Run time per epoch varies from 30 to 7 min.

Shared parameters

parameter	value
corpus	enwik8reduced
history_strategy	layer-constant-length
max_history	25
bptt	<i>variable</i>
batch_size	<i>variable</i>
epochs	4
lr	1e-3
weight_decay	1.2e-6
epoch	4
valid_len	500,000
log_interval	500
save_interval	500
memory_interval	100
hidden_size	460
embed_size	400

parameter	value
growth_factor	5
rnn_type	RNN
reset_hidden	False
reset_growth	True
cuda_on	True

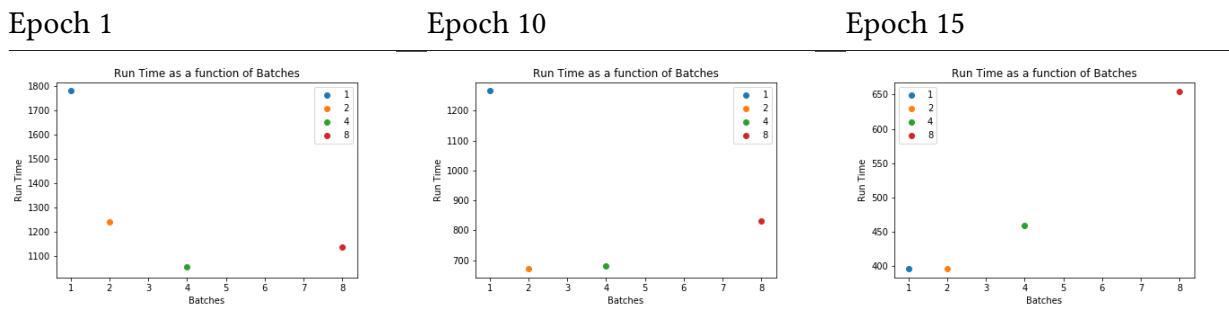
A.18.2 Results

At the end of each epoch, we see a spike in BPC, due to the first evaluation of the epoch (BPC is reinitialised to 0, causing a spike). With any number of batch, while keeping the `bptt * batch_size` ratio, BPC and Validation BPC do not vary.

Even with 200 epochs, with batch size of 1, there is no trace of over-fitting.

Epoch run time :

Epoch	Run time b=1	Run time b=2	Run time b=4	Run time b=8
1	30 min	21 min	18 min	19 min
10	21 min	14 min	14 min	17 min
15	7 min	7 min	8 min	11 min



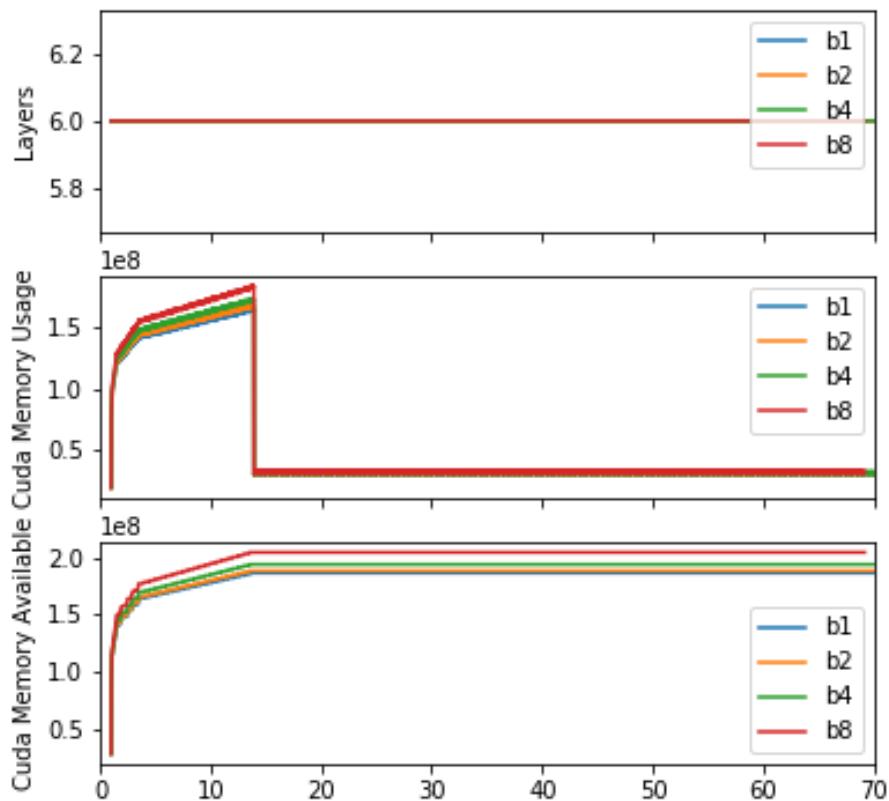
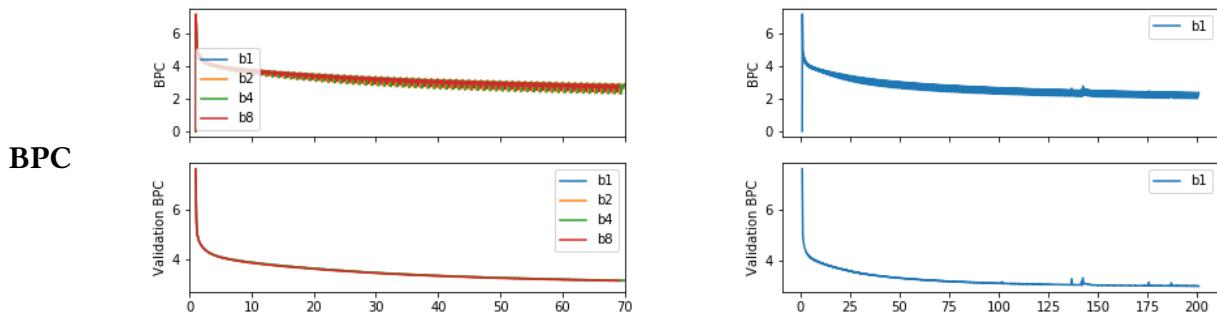
Run time can be split over two set of epochs : before, and after the 15th epoch. Before the 15th epoch, run is faster with more batches, with the exception of batch-size 8, which is slower than batch-size 2 and 4. After the 15th epoch, run is slower with more batches.

To optimize run time, it is necessary to balance run time before 15 epochs. If a high number of batches is planned (50), between 2 and 4 batches are preferable, because the run time after epoch 15 has a lot of impact on global run time; however, if only a few epochs are planned (20), a high number of batches is preferable, as run time before epoch 15 is the most important.

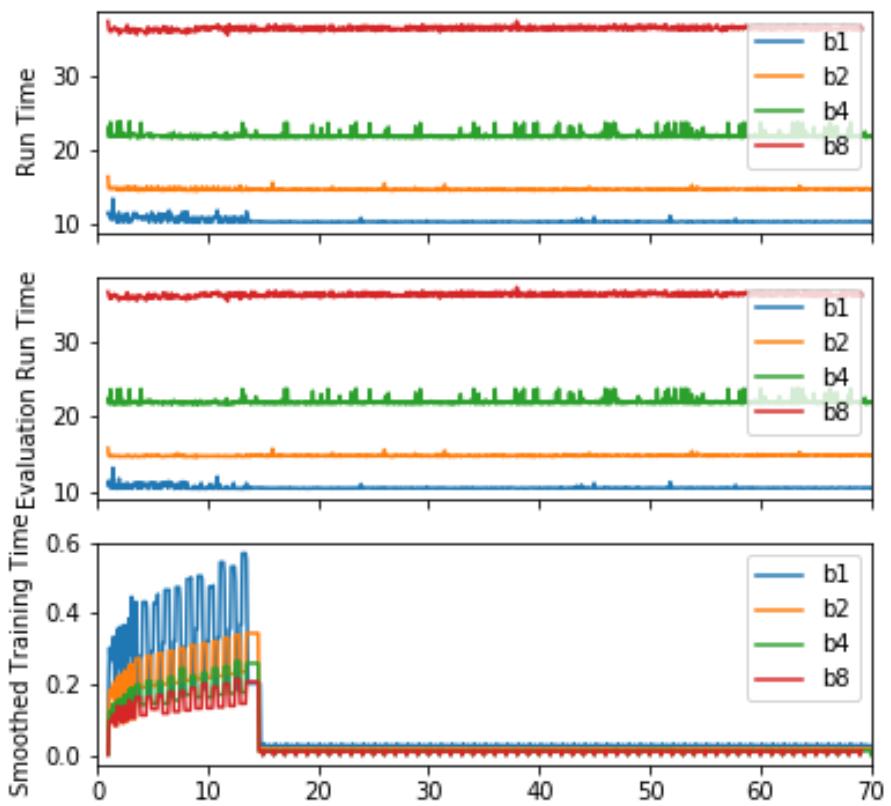
With current corpus, 2, 3 or 4 batches are the most interesting setup.

Decrease in run time is most probably due to the history of the upper layers, that needs multiple epochs to fill.

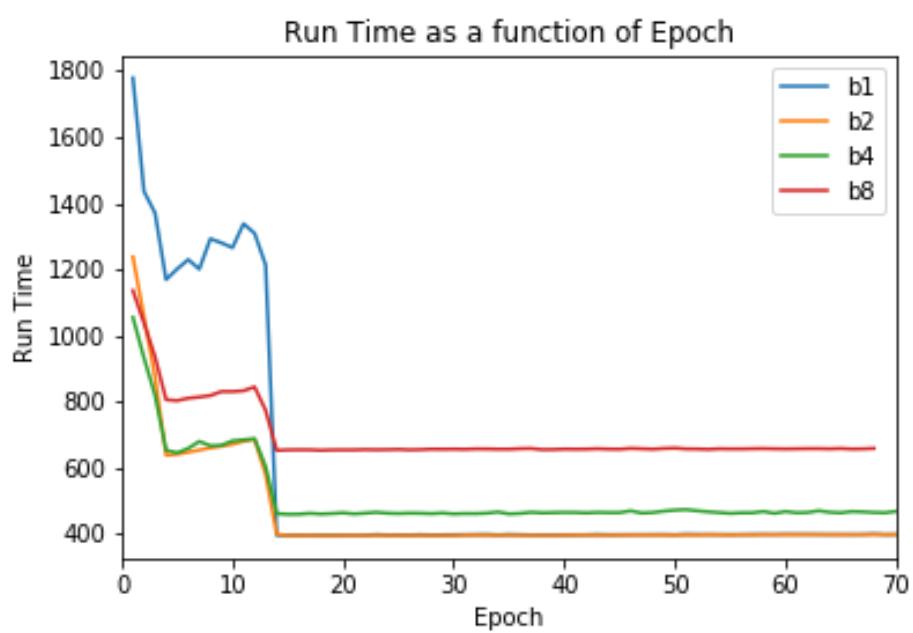
Plots



Memory



Run time (in seconds)



Epoch run time

A.18.3 Next steps

- Independent impact of batch number and sequence length :
 - Run with fixed batch number and varying sequence length
 - Run with fixed sequence length and varying batch number
- Impact of corpus length on optimal number of batch and sequences :
 - Run with varying sequence length and varying batch number over the full length corpus
- Number of parameters :
 - Increase the hidden layer size
- [Future] Transmission rate impact :
 - Compare optimal values of parameters, and BPC reached with varying transmission rate

A.19 Test des performances des différentes améliorations

Run over more than 50 epochs with varied batch size

Test report

by E. Marquer, 2018/06/25, Synalp and Université de Lorraine

A.19.1 Abstract

The test runs are for two parallel experiments :

- test which of 2 and 3 batches are the most interesting
- test the impact of layer by layer training (with an intuitive algorithm developed out of work-time)

The test is composed of 4 runs on grele, with :

- bptt 200/2, batch-size 2
- bptt 200/3, batch-size 3
- bptt 200/2, batch-size 2, layer by layer training,
- bptt 200/2, batch-size 2, layer by layer training, 20 epochs of individual training for each layer, 10 epochs of common fine-tuning for already trained layers (see below the explanation of this algorithm)

Shared parameters

parameter	value
corpus	enwik8reduced
history_strategy	last
max_history	25
lr	1e-3
weight_decay	1.2e-6
epochs	1500
valid_len	500,000
log_interval	500
save_interval	500
memory_interval	100
hidden_size	460
embed_size	400
growth_factor	5
rnn_type	RNN

parameter	value
reset_hidden	False
reset_growth	True
cuda_on	True

Model keys and specificities

When noting is specified, all models have :

- 2 batches, and a sequence length of 200/2
- 1 RNN layer per MSNN layer

Model	Specificity
b2	classical model, comparison basis
b3	3 batches, sequence length of 200/3
s	“scheduled(10,10)” : use layer by layer training; 10 epochs for individual training, 10 epochs for intermediary fine-tuning
s-a20-l10	“scheduled(20,10)” : use layer by layer training; 20 epochs for individual training, 10 epochs for intermediary fine-tuning
l2	2 RNN layer per MSNN layer
l3	3 RNN layer per MSNN layer
s_l3_a	3 RNN layer per MSNN layer, “scheduled(10,10)” (see model “s”), attentive intermediary input

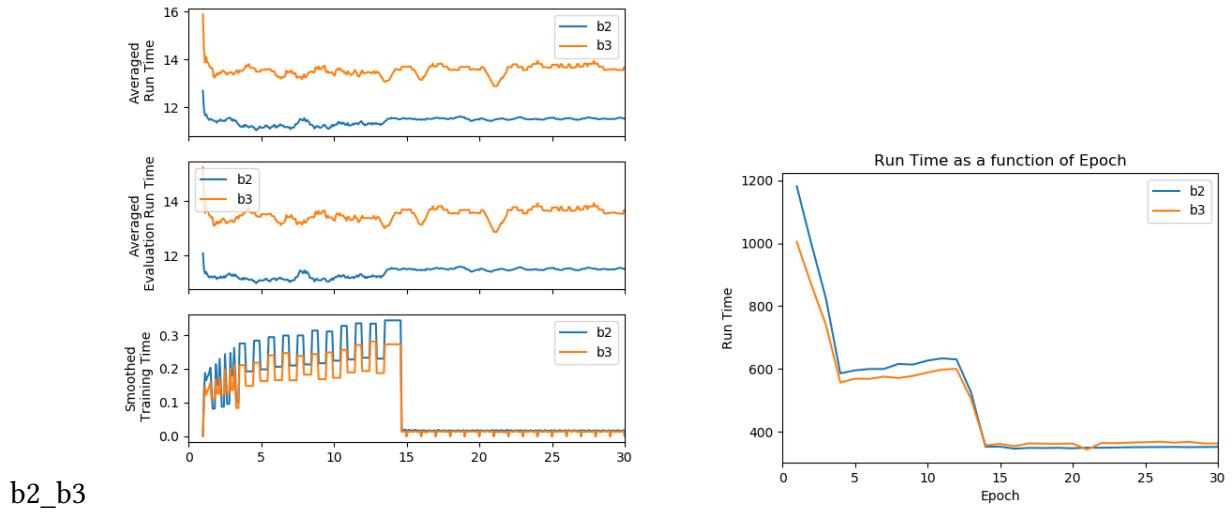
Series

Series	Model	Respective name of models on plots	Objective
b2_b3	b2, b3	“b2”, “b3”	Compare training with 2 and 3 batches
1	b2, l2, l3	“1 RNN layer”, “2 RNN layers”, “3 RNN layers”	See impact of number of RNN layers
lbl	b2, s, s-a20-l10	“Classical”, “Layer by layer (10, 10)”, “Layer by layer (20, 10)”	See impact of layer by layer training
sum	b2, s_l3_a	“Classical”, “3 RNN layers, LbL, attentive”	Check if layer by layer training, attentive model, and multi-RNN-layered architectures are compatibles

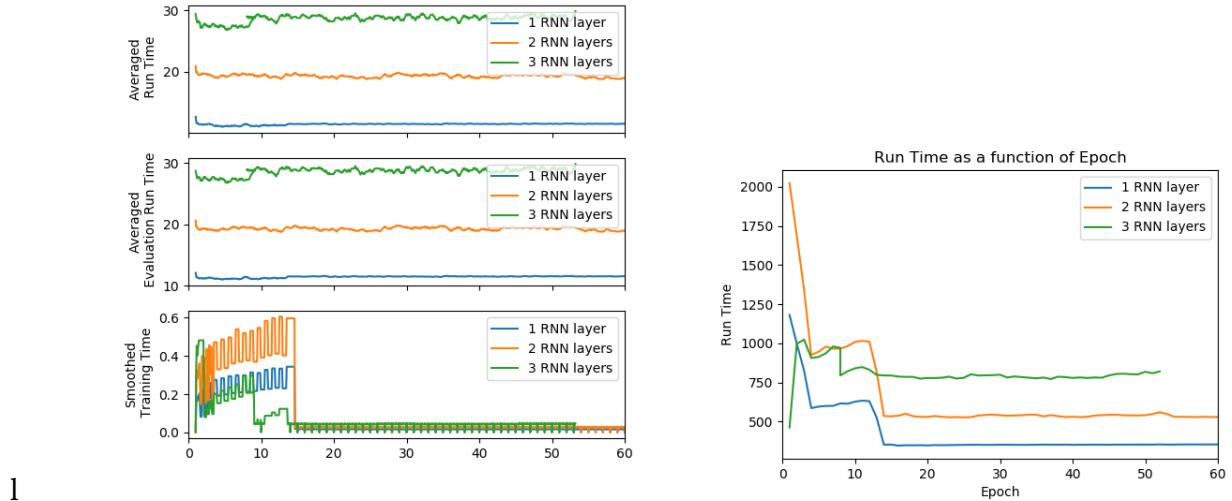
A.19.2 Results

Series Time

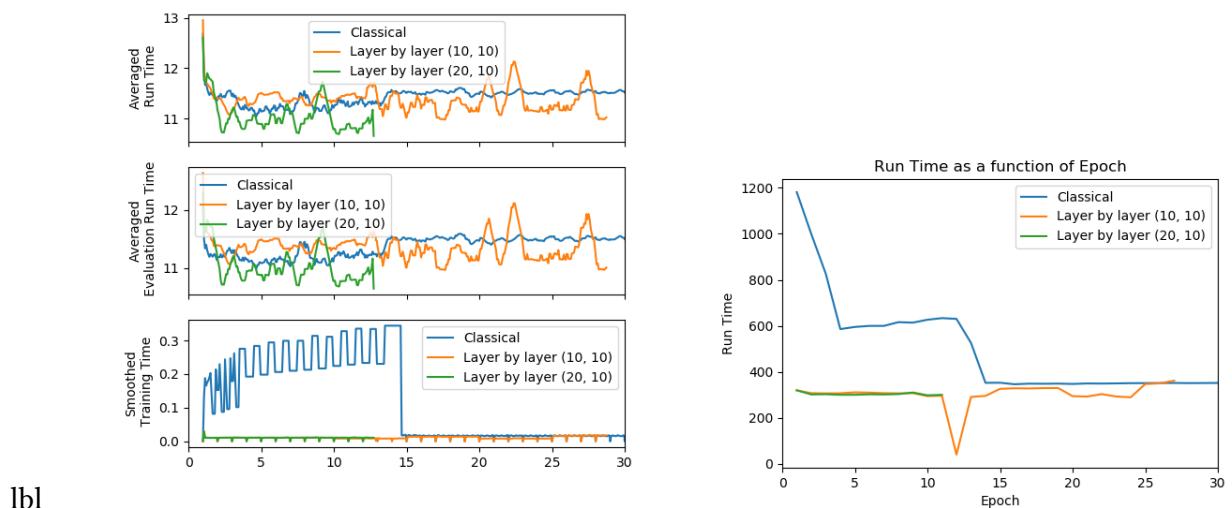
Time to run an epoch BPC



b2_b3



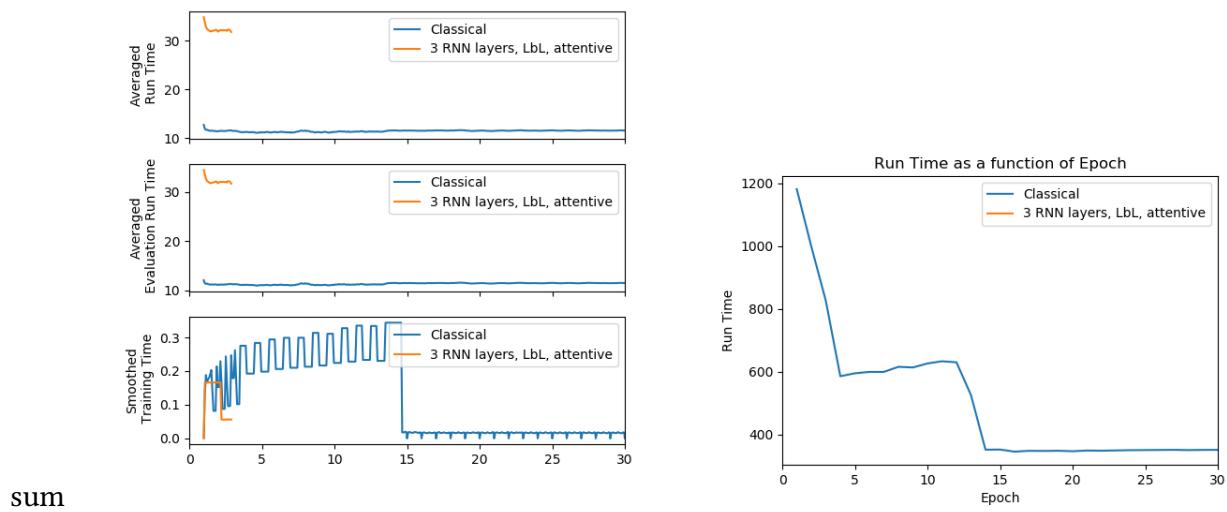
I

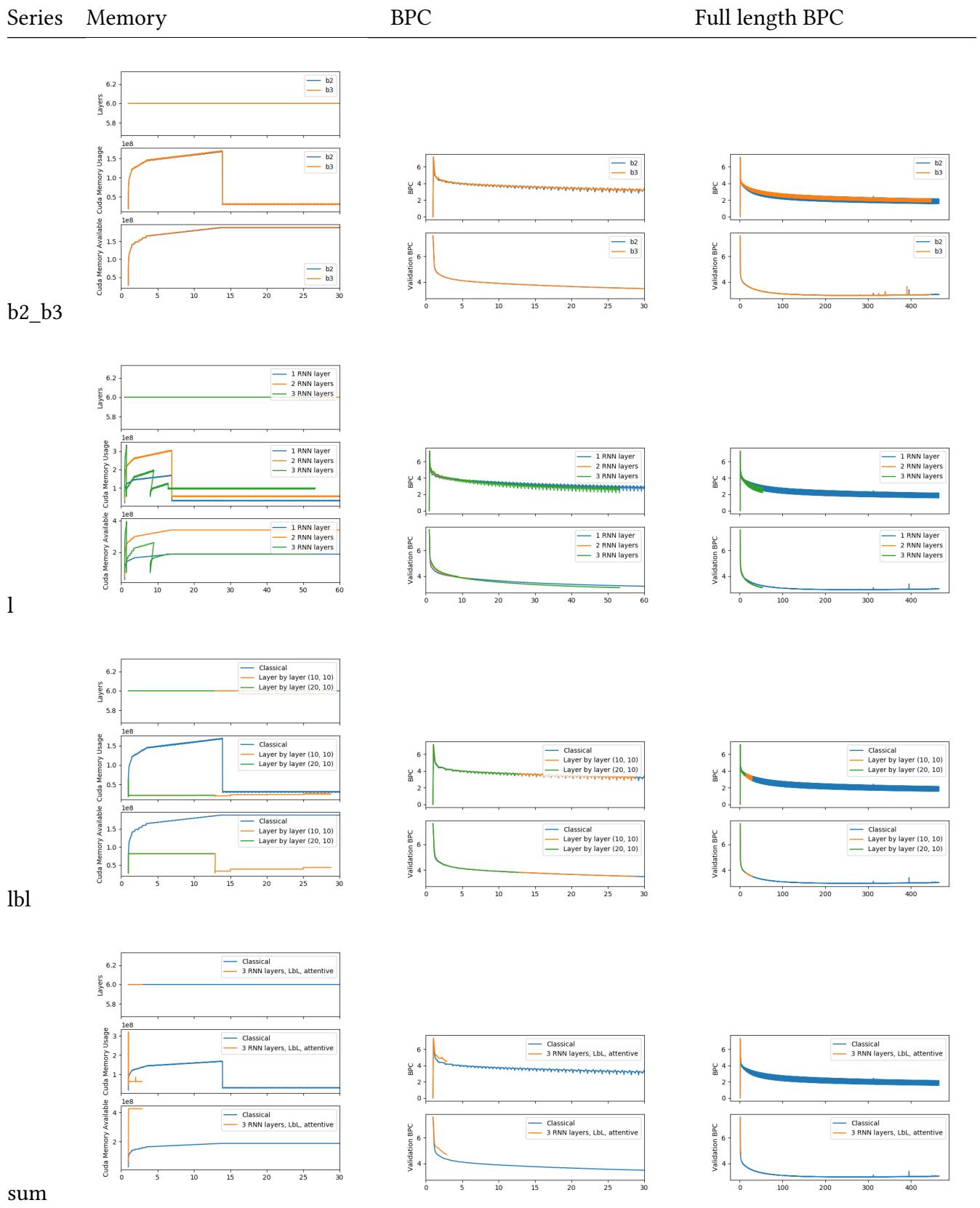


lbl

Series Time

Time to run an epoch BPC





Plots will be referred to with : (Ex : "l :Memory")

Time necessary to reach indicated validation BPC

Model	5 BPC	4 BPC	3 BPC
b2	0H 19M 41S	1H 19M 47S	15H 13M 7S
b3	0H 16M 44S	1H 11M 48S	15H 3M 22S
s	0H 5M 19S	0H 30M 59S	
s-a20-l10	0H 5M 19S	0H 30M 26S	
l2	0H 33M 43S	2H 27M 51S	
l3	0H 7M 41S	2H 13M 2S	
s_l3_a	0H 16M 7S		

Empty fields are where network has not reached BPC yet ~~Crossed out~~ fields are where data has been corrupted (because of an interruption in training)

Epochs necessary to reach indicated validation BPC

Model	5 BPC	4 BPC	3 BPC
b2	0.296	6.000	141.4
b3	0.367	5.954	136.4
s	0.371	6.000	
s-a20-l10	0.371	6.000	
l2	0.593	6.519	
l3	0.816	7.296	
s_l3_a	1.148		

Empty fields are where network has not reached BPC yet

A.19.3 Analysis

Run time and memory on “l” series (l :memory, l :time, and l :time to run an epoch)

Training of “l3” model was interrupted 2 times before the 15th epoch, each time cleaning the memory, inducing training time reduction and epoch time reduction (due to the correlation of memory usage and training time).

Run time and memory on “lbl” series (lbl :memory, lbl :time, and lbl :time to run an epoch)

The specificity of layer by layer training are a lead to the cause of the collapse of memory usage and run time during the 15th epoch.

During layer by layer training, memory usage is kept constant over the first 15 epochs, whereas with classical training, memory usage increases during this period.

Layer by layer training reduces simultaneous graph accumulation over the different epochs by training a layer at a time. It possibly reduces SGD inertia too.

Performance of layer by layer training

No notable variation on BPC. For run time and memory, see above.

Performance of multi-RNN-layered architecture

A slight improvement of BPC can be seen with 3 layers (sadly, BPC data for 2 layers is corrupted). Computation-time is proportional to the number of layers (time data for 3 layers is unusable).

Performance of 3 batches compared to 2 batches

2 and 3 batches have an almost identical performance (time-wise, BPC-wise and memory-wise), except 3 batches seem to get less dispersed BPC.

Results of long training with 2 and 3 batches (b2_b3 :full length bpc)

The plot “b2_b3 :full length bpc” shows that after 200 epochs, validation BPC stagnates. After 300 epochs, validation BPC begins to increase very slowly, those are the first signs of over-fitting. These observations were cross-checked with the raw data.

Note on attention module

A model with the attention module was tested for 5 hours, and did not reach a full epoch. It can be deemed that without a training algorithm like layer by layer training, using attention is not viable.

B Rapports d'avancement du projet PAPUD

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

Les sections suivantes contiennent les rapports intermédiaires fournis aux membres de l'équipe du projet PAPUD.

Pour les mêmes raisons que pour l'annexe précédente (décrise dans la section A.1, page 87), 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.

B.2 Informations générales

General information

B.2.1 Corpus

firsttest

Baseline accuracy True baseline accuracy (for char -) : training 0.443 ; valid 0.414 ; test 0.423

char	training	valid	test
-	0.443	0.414	0.423
	0.064	0.067	0.063
a	0.021	0.022	0.023
o	0.015	0.013	0.011
<unk>	0.0	0.0	0.0

B.2.2 Grid-5000

To develop and train the model, we are using Grid-5000 computers clusters. For specific information on Grid-5000, see <https://www.grid5000.fr/>.

GPU-equipped nodes

Due to the properties of neural networks models, it is really efficient to use GPUs to train them.

Here are the main GPU-equipped nodes of Grid-5000 :

Nodes	GPU	Graphical memory	RAM	Production	CUDA
graphique-1	2 x Nvidia Titan Black	2 x 6GB	64GB	X	2880
graphique-2/6	2 x Nvidia GTX 980	2 x 4GB	64GB	X	2048
graphite-1/4	Intel Xeon Phi 7120P	16GB	256GB		?
grele-1/14	2 x Nvidia GTX 1080 Ti	2 x 11GB	128GB	X	3584
grimani-1/6	2 x Nvidia Tesla K40M	2 x 12GB	64GB	X	2880

[NOT TESTED YET] Model conversion

See <https://github.com/ysh329/deep-learning-model-convertor>

B.3 Résultats de l'implémentation basique

Results of the basic implementation of the model

2018/07/09 - SYNALP - Esteban MARQUER

B.3.1 Paradigm

The test is run on a minimal number of epoch (10), with a minimal model.

The training algorithm used is an example by example training.

Model architecture

The model is a line by line predictive model, composed of : - a character embedding layer; - a pooling layer; - a linear layer; - an output layer.

The output of the model is a probability distribution over known characters for every character of the predicted line.

B.3.2 Results

GPU memory usage

As expected from the model architecture, GPU memory usage is constant.

Computation time

Loss and accuracy

The loss used is cross-entropy loss, a character per character negative-log-likelihood loss over the soft-maxed distribution.

Overall, the loss gives a score to the prediction of the model, by comparing the target character and a distribution of probabilities for each character. If the probability for the target character is high and other character low, the model does a good prediction of the character, and the score given is low. The closer the score is to 0, the better it is. The scores of each characters is averaged, producing a global loss over the line.

Accuracy is a percentage. The closer to 100 % the better. As the loss is bound by 0 and +Infinity, and the closer to 0 the better, a correct transformation to accuracy could be : $\exp(-\text{loss})$ for an accuracy between 0 and 1.

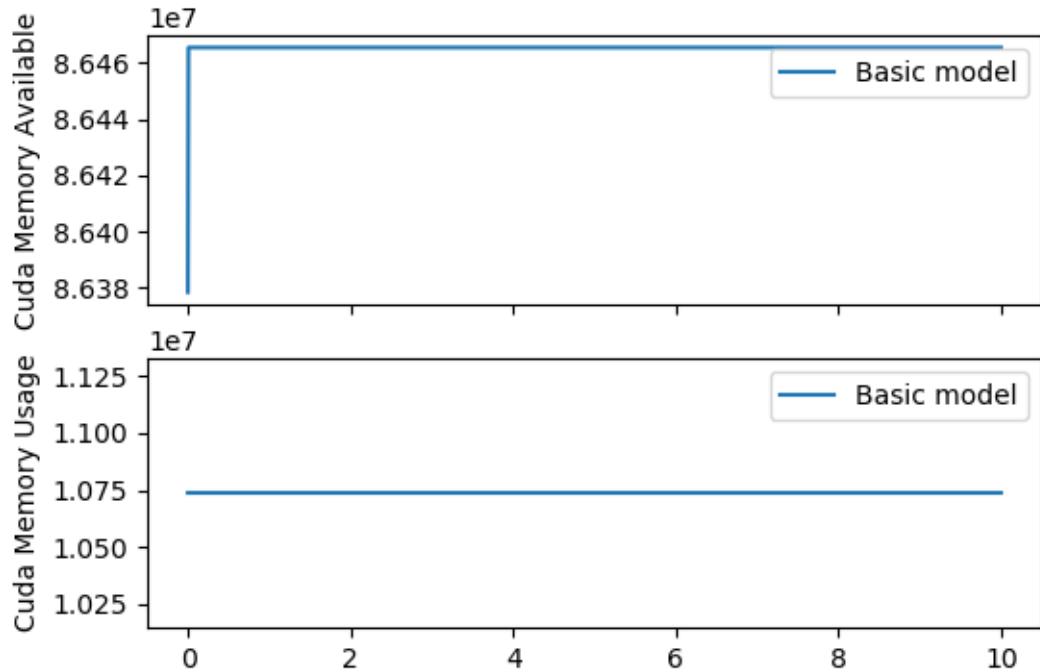


FIGURE B.1 – memory usage

The small spike recurrently appearing in the loss and accuracy is most likely due to a noisy part of the corpus (around the middle of the corpus) causing the model to learn wrongly on those specific examples.

The best precision obtained at the end of 10 epochs is 50%, corresponding to a loss of about 0.7.

B.3.3 Improvements and next steps

Mini-batch

Currently, the models learn one example at a time, meaning it computes the result for a line of input, compares it to the target, and updates weights. A common algorithm is the mini-batch algorithm, computing simultaneously a set of examples, their loss compared to the target, and updates the weights of the model all at once for the whole set of examples.

This algorithm speeds-up training while making the most of the GPU.

Dynamic corpus

While with the current corpus there is no real problem in storing the whole corpus in the memory, the future corpus will be over 400GB of text. It is necessary to replace the current method by a dynamic loading and transformation of the parts of the corpus currently used by the model. An ideal solution would be to read the target data directly from the archive containing the corpus.

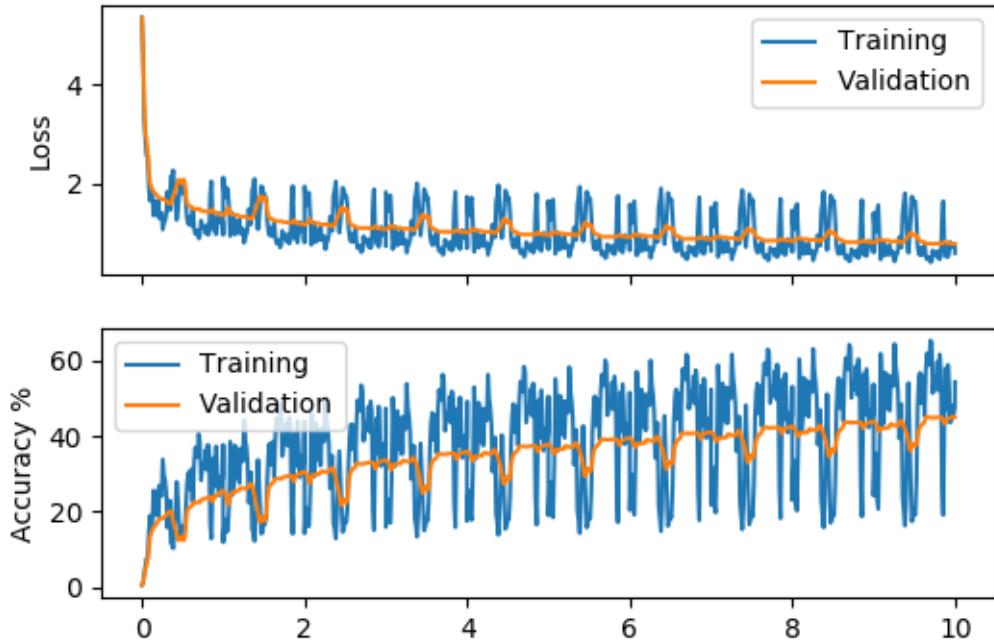


FIGURE B.2 – loss

B.4 Paquets (*batchs*) simultanés

Increasing the number of simultaneous examples

2018/07/10 - SYNALP - Esteban MARQUER

B.4.1 Paradigm

The test is run on a small number of epoch (20), with a minimal model and a new training algorithm.

The training algorithm used is a **mini-batch training**, meaning we compute the output for multiple examples all at once, we compute an averaged loss over those examples, and we update the model.

The potential effects of this algorithm are :

- an increase of GPU memory usage, as computations are done on larger data ;
- a decrease of computation time, with the number of computations reduced ;
- a smother training loss, because it is averaged over multiple examples ;
- avoidance of some local minima.

A second test with random batch-size between 1 and 1000 was done on 50 epoch, to evaluate the effect of the batch size and find an optimum.

B.4.2 Results

GPU memory usage

As more examples are fed to the model, there is a very slight increase in GPU memory usage : $0.013e7$ B, corresponding to 127kiB (this amount is negligible with more than 10GiB available and a current usage of about 10MiB).

Conclusion : increasing the number of simultaneous examples has no substantial downsides memory-wise.

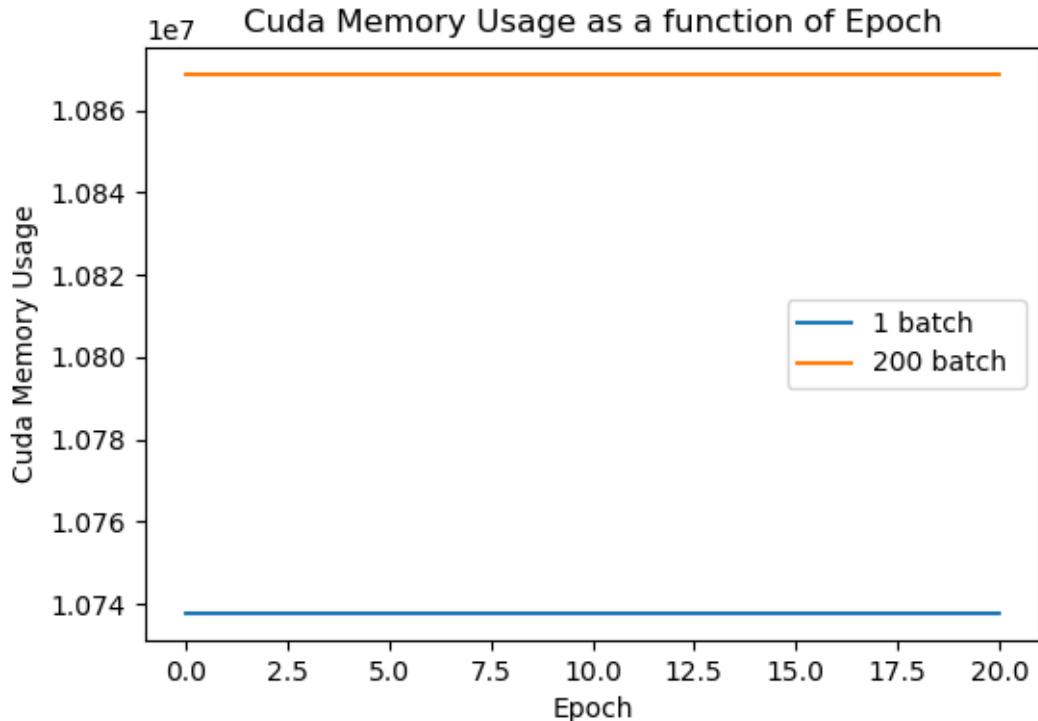
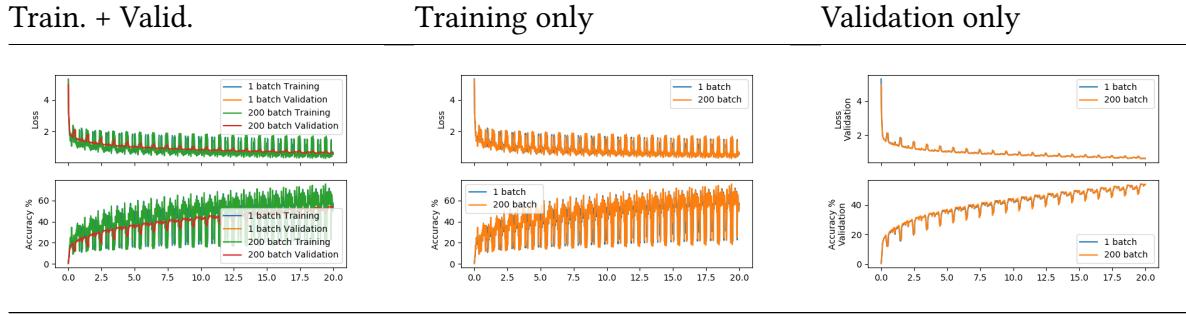


FIGURE B.3 – memory usage

Loss and accuracy

As loss is averaged on multiple examples, it should be smoother. But, probably because the number of simultaneous examples is too small, there is no noticeable change of loss, with the curves superposed.

Conclusion : increasing the number of simultaneous examples has no substantial effect loss-wise.



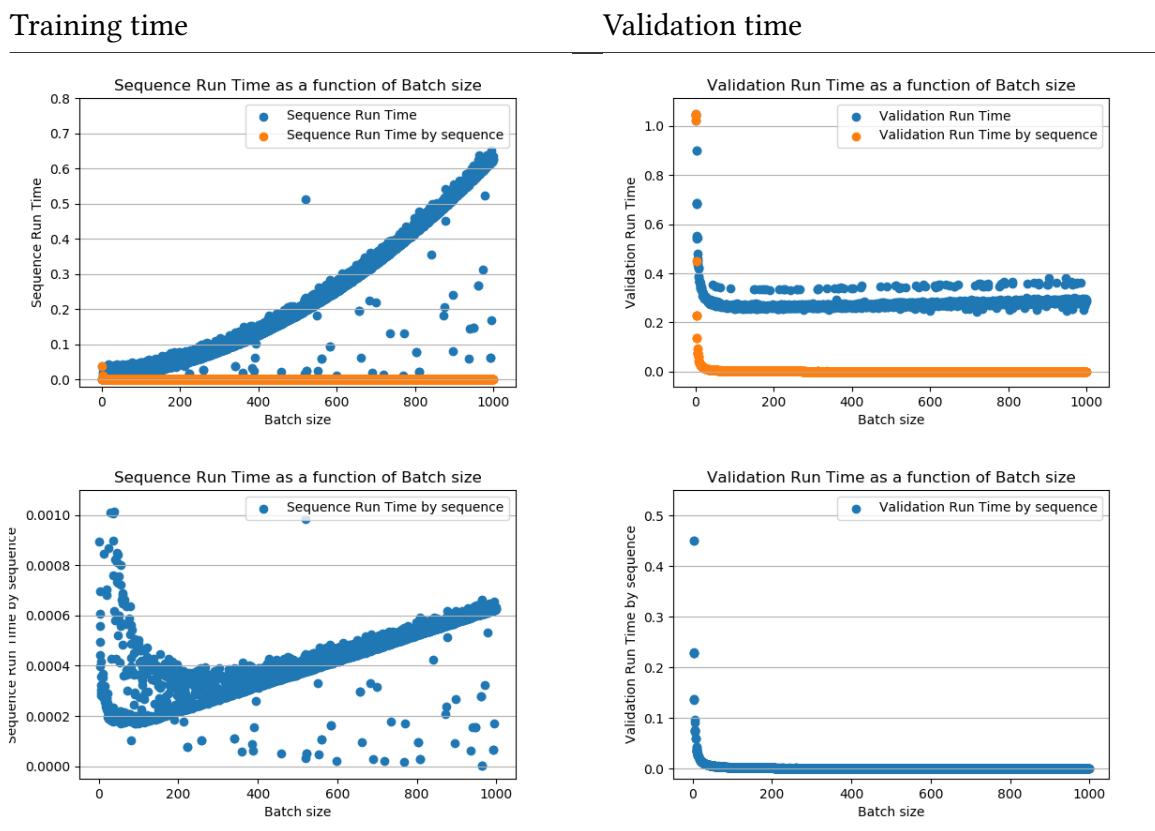
Computation time

The computations done on GPU benefit from grouping similar operations. By computing multiple examples together, we can use this property to speed up training. Moreover, retro-propagation and model updates are less frequent, reducing computational load and training time.

The best-case time allow an improvement from 10ms to less than 2ms per **training** sequence with a batch-size of 50, and the worst-case one allow 4ms per training sequence with a batch-size of 200.

A small gain can be achieved on **validation** time by increasing batch size over 50, but increasing it more has no effect.

Conclusion : increasing the number of simultaneous examples leads to a notable improvement of computation time.



B.4.3 Conclusion

Even if there is no improvement of loss or memory, the gain in computation time is enough to accept this algorithm.

The ideal batch-size (with the current node “grele”) is between 50 and 200. In future works, a batch-size of 200 will be used, as it present the best worst- and best-case time performances.

B.4.4 Improvements and next steps

Dynamic corpus

The dynamic corpus implementation is ready (except small details) and working, only integration is left.

Buffer size The dynamic corpus can use a buffer, and the size of this buffer must be at least the size of the batch. It will be necessary to test which size is optimal. An optimal buffer has the minimal size to make computation time over the buffer size only slightly higher than pre-loading time. It allows training to continue without interruption, while maintaining a low memory usage.

B.5 Analyse du pic de performance

Performance spike analysis

2018/07/18 - SYNALP - Esteban MARQUER

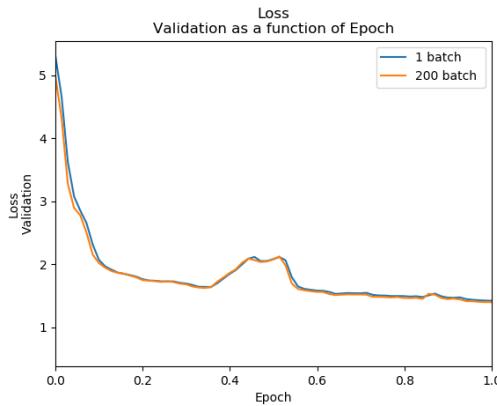
B.5.1 Problem

During training, a big performance spike appeared periodically. It is necessary to know why this spike appeared.

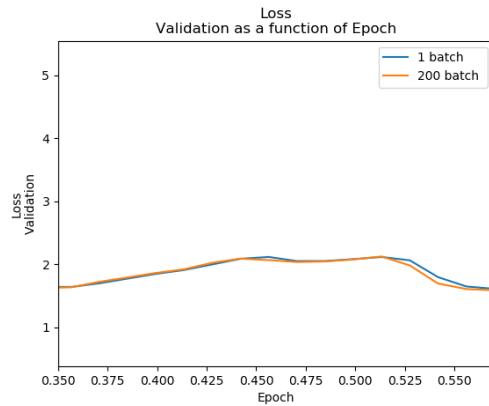
The different parts of the spike are :

- from line 24545 to line 31558 (35% to 45% of the corpus) : the increase of the BPC ;
- from line 31558 to line 36468 (45% to 52% of the corpus) : a stable part at a high value ;
- from line 36468 to line 38572 (52% to 55% of the corpus) : the decrease of the BPC.

Loss of 1 epoch (first epoch)



Zoom on spike



B.5.2 Analysis

By extracting the parts of the corpus corresponding to the parts of the spike, and scrolling through them, some recurrent elements appear : - lines beginning by kern, more specifically kern info and kern debug ; - lines containing a memory address, like 0 x91ffff , 0x0093 and 00000000fed18000, or an error code like 0x0100, - lines beginning by daemon, more specifically daemon err ;

The most interesting part of the spike is the increase of the BPC, were the performance deteriorate.

Given the repartition and percentages (see the next two sub-sections), the most likely causes for the spikes are :

- the memory address and hexadecimal codes ;
- the kernel messages (very repetitive, and containing memory address and hexadecimal codes).

Examples of Kernel messages Ces données ont été modifiées pour des raisons de confidentialité.

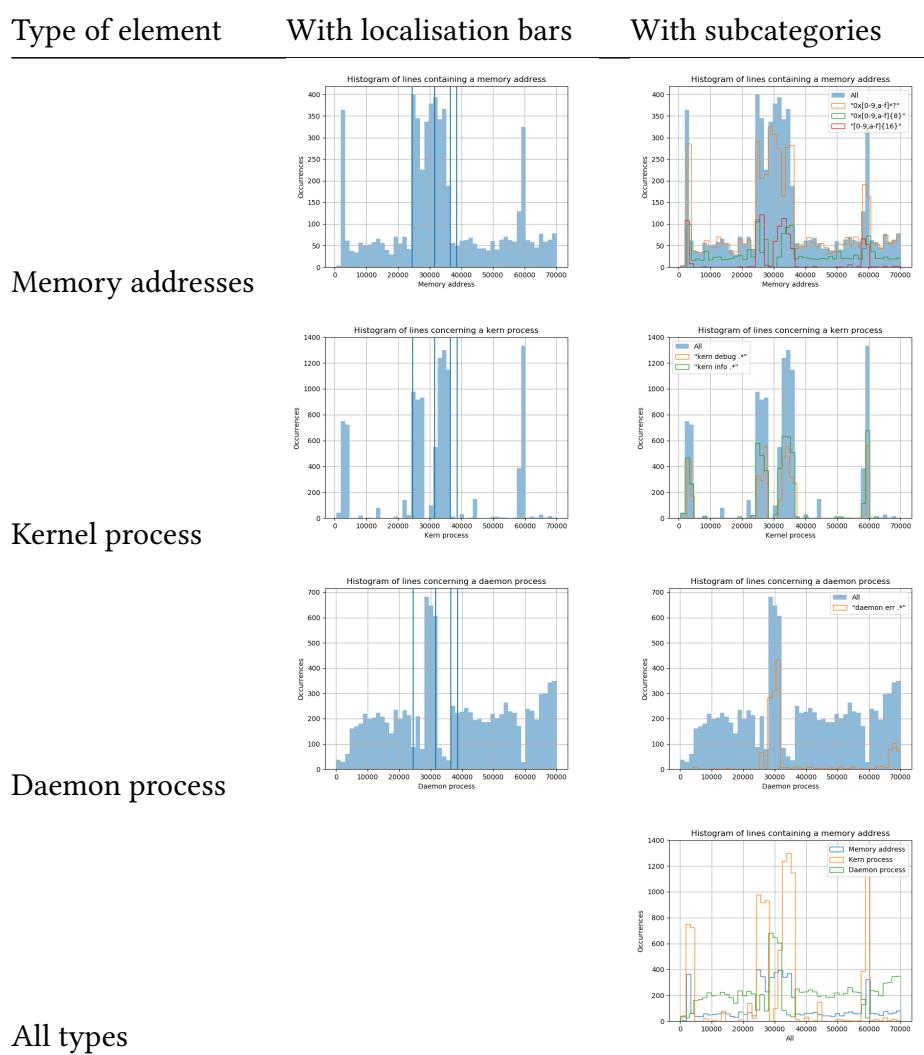
```

1 kern info kernel ABCD: FHJLN (abcd_id[0x00] fhjln_id[0x00] enabled)
2 kern info kernel ABCD: FHJLN (abcd_id[0x02] fhjln_id[0x02] enabled)
3 kern info kernel ABCD: FHJLN (abcd_id[0x04] fhjln_id[0x04] enabled)
4 kern info kernel ABCD: FHJLN (abcd_id[0x06] fhjln_id[0x06] enabled)
5 kern info kernel ABCD: FHJLN (abcd_id[0x08] fhjln_id[0x08] enabled)

```

Repartition of match in the corpus

The “localisation bars” (vertical blue lines) delimit to the different parts of the spike.



Percentages of match in each part of the corpus

Percentages of match are percentages on the total line number of the part of the corpus analysed.

```

1 --- spike up slope (lines 24545 to 31558, 35% to 45%) ---
2 Total lines: 7013
3 Matching "kern .*": 2888 (41%)
4 Matching "kern info .*": 1458 (20%)
5 Matching "kern debug .*": 1172 (16%)

```

```

6 Matching "daemon .*": 2048 (29%)
7 Matching "daemon err .*": 935 (13%)
8 Matching any memory pattern: 1728 (24%)
9 Matching "0x[0-9,a-f]{8}": : 196 (2%)
10 Matching "[0-9,a-f]{16}": : 250 (3%)
11 Matching "0x[0-9,a-f]*?": : 1478 (21%)
12
13 --- spike flat (lines 31558 to 36468, 45% to 52%) ---
14 Total lines: 4910
15 Matching "kern .*": 4218 (85%)
16 Matching "kern info .*": 2154 (43%)
17 Matching "kern debug .*": 1760 (35%)
18 Matching "daemon .*": 346 (7%)
19 Matching "daemon err .*": 174 (3%)
20 Matching any memory pattern: 1149 (23%)
21 Matching "0x[0-9,a-f]{8}": : 294 (5%)
22 Matching "[0-9,a-f]{16}": : 296 (6%)
23 Matching "0x[0-9,a-f]*?": : 853 (17%)
24
25 --- spike down slope (lines 36468 to 38572, 52% to 55%) ---
26 Total lines: 2104
27 Matching "kern .*": 27 (1%)
28 Matching "kern info .*": 15 (0%)
29 Matching "kern debug .*": 0 (0%)
30 Matching "daemon .*": 383 (18%)
31 Matching "daemon err .*": 15 (0%)
32 Matching any memory pattern: 90 (4%)
33 Matching "0x[0-9,a-f]{8}": : 34 (1%)
34 Matching "[0-9,a-f]{16}": : 5 (0%)
35 Matching "0x[0-9,a-f]*?": : 85 (4%)
36
37 --- whole spike (lines 24545 to 38572, 35% to 55%) ---
38 Total lines: 14027
39 Matching "kern .*": 7133 (50%)
40 Matching "kern info .*": 3627 (25%)
41 Matching "kern debug .*": 2932 (20%)
42 Matching "daemon .*": 2777 (19%)
43 Matching "daemon err .*": 1124 (8%)
44 Matching any memory pattern: 2967 (21%)
45 Matching "0x[0-9,a-f]{8}": : 524 (3%)
46 Matching "[0-9,a-f]{16}": : 551 (3%)
47 Matching "0x[0-9,a-f]*?": : 2416 (17%)
48
49 --- full corpus ---
50 Total lines: 70131
51 Matching "kern .*": 10972 (15%)
52 Matching "kern info .*": 5298 (7%)
53 Matching "kern debug .*": 4134 (5%)
54 Matching "daemon .*": 10828 (15%)
55 Matching "daemon err .*": 1504 (2%)
56 Matching any memory pattern: 5859 (8%)
57 Matching "0x[0-9,a-f]{8}": : 1586 (2%)
58 Matching "[0-9,a-f]{16}": : 893 (1%)
59 Matching "0x[0-9,a-f]*?": : 4987 (7%)
60
61 Matching "kern .*" outside of spike: 3839 (5%)

```

B.5.3 Conclusion(s)

There are two possible conclusions :

- the kernel messages are the cause of the spike ;
- or the memory addresses and hexadecimal codes are the cause of the spike.

Kernel messages

If the kernel messages are the cause of the spike, the most likely explanation is that this part of the corpus represent a crash of the server or a major error. In that case, we must remove that part of the corpus from the training set, as it is not the “normal” evolution of the log.

Memory addresses and hexadecimal codes

If the memory addresses and hexadecimal codes are the cause of the spike, it should be because a succession of number is a very specific thing to learn. In that case, either we let the model learn the brute codes, or we replace every code by a “<hex>” character to ease the learning process. It is also possible to replace the different kind of code by a different character.

B.5.4 Improvements and next steps

To check whether the memory addresses and hexadecimal codes, or the kernel messages are the cause of the spike, trying to train the model while replacing every code by a “<hex>” character. If there is no improvement, then the codes are not the cause of the performance spike.

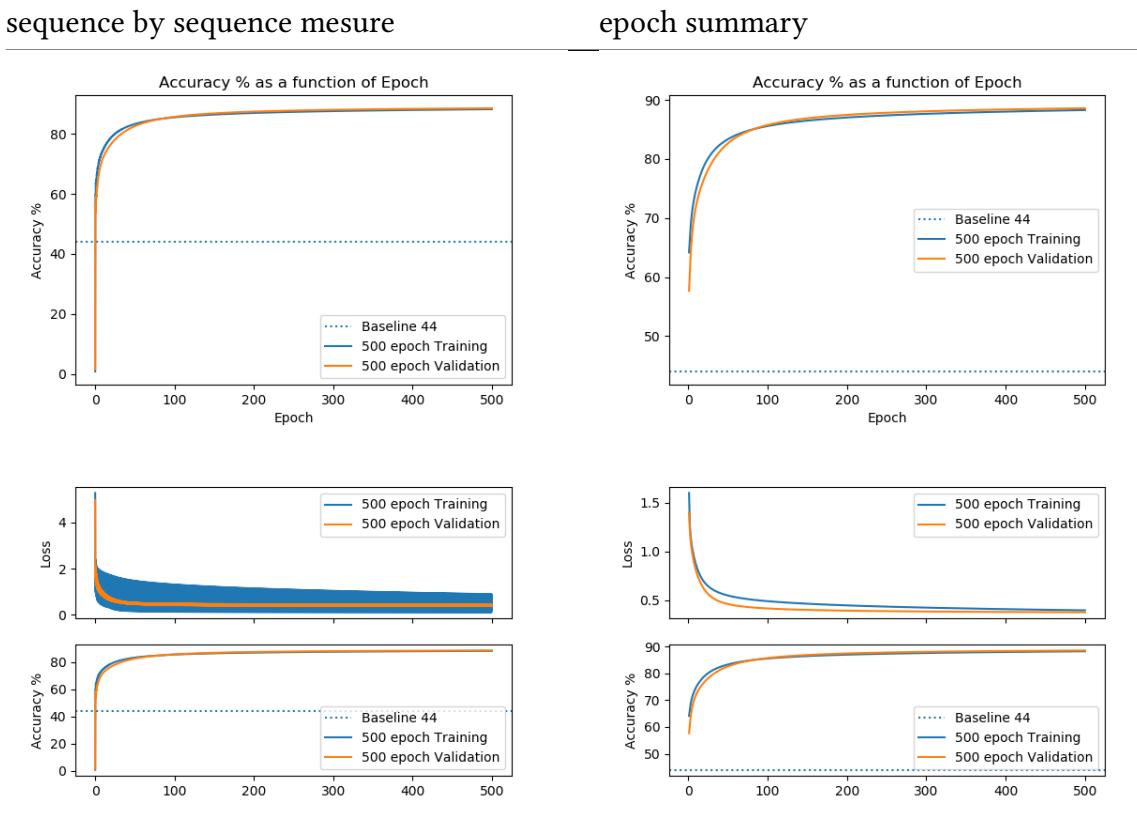
B.6 Rapport de la réunion avec les autres membres du projet

Team meeting report

2018/07/20 - SYNALP - Esteban MARQUER

B.6.1 Main points

- Performance spike issue : see report “2018_07_18-Performance_spike_analysis”
- solution chosen : replace hexadecimal codes with a special label per kind of code
- Long training : no over-fitting, 90% accuracy in 500 epochs



- Baseline accuracy : see first section of “General_information.md” for values.
 - Baseline accuracy is high with padding (about 50%), perhaps lines are too long (a lot of padding is needed)
 - Compared with performance, performance is still good
- GPU usage with completely loaded corpus : 30% to 60%
 - This is bad news, with such a model training should go at 99% all the time
 - It is necessary to locate the element slowing the process, try removing all unnecessary processes (logging, storage in memory, accuracy, ...)
- New corpus implementation (with buffer and iterators) : about 180s/epoch, 65s/epoch with old implementation (everything loaded in memory)
 - a good implementation of the data loading is critical

- perhaps pre-loading is too slow because of computations, a pre-processed version could help
- training the model multiple time on a loaded segment could bridge the gap between the two processes used
- using more processes could do the trick (one for loading only, one for processing, and one for training)
- using “binary”-sized batches (like 8, 64 or 1024) is said to achieve faster results, maybe a bit of speed can be gained there
- Development of a learning rate optimisation script : good, better if offline (good if both online and offline)
 - online stands for optimisation before, or/and during every training
 - offline stands for an analysis done a single time, aside from any training

B.6.2 Improvements and next steps

- Finish the development of learning rate optimisation.
- Try to make the corpus implementation clean and fast enough (with compared run times).
- Integrate the modifications of the corpus processing (memory address management)
- Use “binary”-sized batches ; 128 seems perfect, as it is between 50 and 200 (the bounds found when optimising batches).

B.7 Optimisation du taux d'apprentissage

Learning rate optimisation

2018/07/23 - SYNALP - Esteban MARQUER

B.7.1 General information

Learning rate is an hyperparameter in the training algorithm which changes the speed of the training and the performance of the model. Specifically, it is a coefficient of the gradient used to update the parameters of the model.

There are three main learning rates for a model :

- a learning rate that is too small (closer to 0) : the training is slow, and can get blocked in some local minima ;
- a learning rate that is too high (closer to +infinity, usually closer to 1) : the learning is faster and avoid local minima, but could diverge from the solution ;
- a balanced learning rate : what we want to find, the traing is fast yet does not diverge.

B.7.2 Optimisation process

Usually, learning rate optimisation is done with a logarithmic scale of the learning rate. The shape of the produced curves confirm the use of such a scale.

The optimisation process is driven by three parameters and a single metric.

The metric is the accuracy of the model on the validation corpus at the end of the training.

The parameters are : the two bounds of the learning rate, and the learning rate variation factor : the “learning rate multiplier”.

The learning rate varies as follow in the psedo-python algorythm :

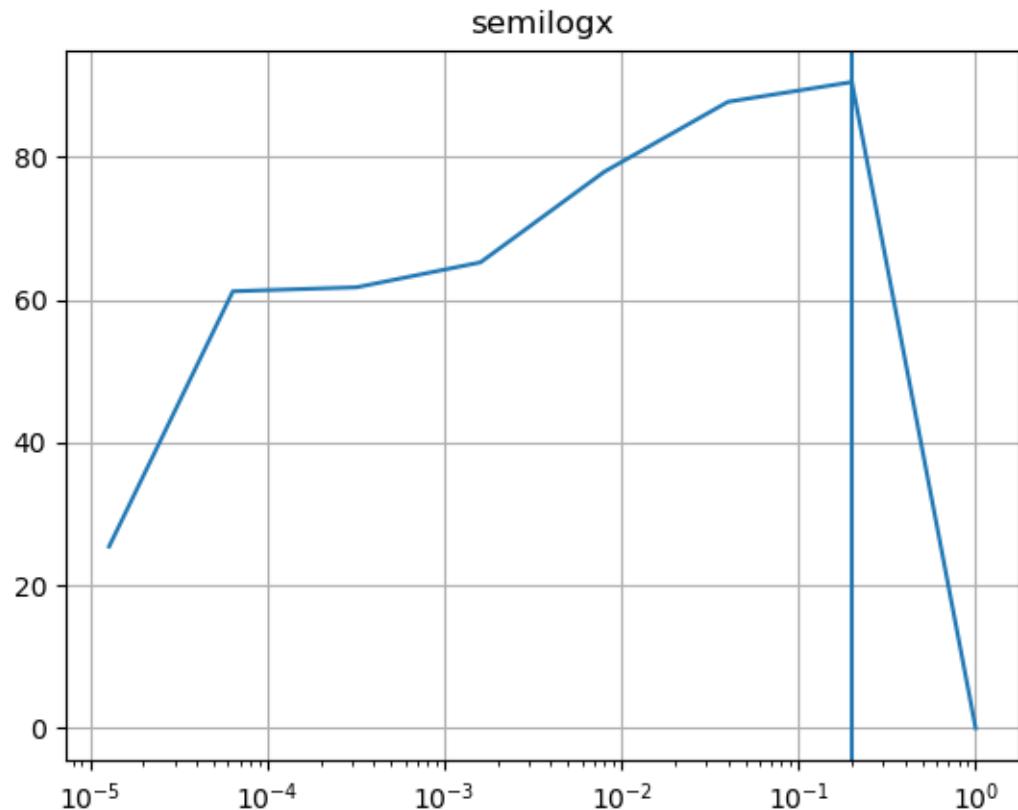
```
1 # the learning rate takes the highest value as start value, as it will
  # decrease over time
2 learning_rate = start_value
3
4 # until the learning rate as reach the stop value (the lowest of the two
  # bounds), we make it vary logarithmically
5 while learning_rate > stop_value:
6     performance = train_model(learning_rate)
7     save_model_performance(performance, learning_rate)
8
9     # the learning rate is updated
10    # example with a learning rate of 1 and a learning rate multiplier of 0.1:
11    # at first the learning rate is 1, then 0.1, then 0.01 ...
12    learning_rate = learning_rate * learning_rate_multiplier
13
14 # we compare the performance of the model with the different learning rates
15 compare_model_performance()
```

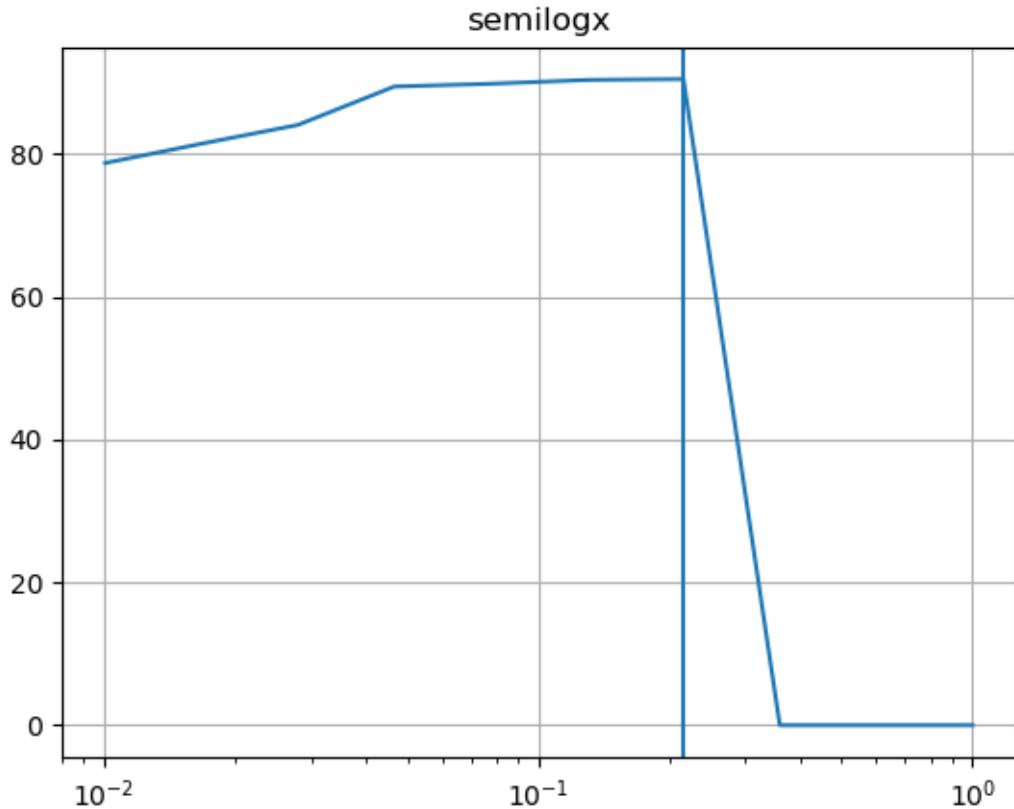
The closer to 0 the learning rate multiplier is, the faster the variation will be, and the closer to 1 it is, the slower the variation will be. To have a decent resolution in the curves, a multiplier close to 1 is crucial.

The training is done each time on a full epoch.

B.7.3 Results

The first plot is done with a learning rate between 1 and 10^{-5} , with a multiplier of 0.2. The second one is done with a learning rate between 1 and 10^{-2} , with a multiplier of 0.6 (the total of dots is 10).





There is a strange accuracy at the end of the first plot : the theoretically unreachable 0%. It is confirmed by the second plot, with multiple points having an accuracy of 0%. With a baseline accuracy of 44%, it is clear that the result diverge from what is expected. It is a case of divergence due to a learning rate that is too high.

The best learning rate found is 0.216 ($0.6^{3/2}$), giving the fastest learning (over 90% of accuracy in 1 epoch) without diverging.

B.7.4 Additional information

The current implementation consist of a script building a model, and finding the ideal learning rate for this model. It is an offline implementation of the model.

The way it is implemented is ideal for an online use too, as the only operation needed are the removal of the plotting part, and adding the reload of the model with an updated learning rate.

B.7.5 Improvements and next steps

Use the new learning rate in the training. As it is quite close to diverge, it would be advised to use a slightly lower learning rate like 0.2.

If inline optimisation is used, three possibilities seem viable : - choosing the learning rate every time we start a training, to adapt to the current hyperparameters ; - updating the learning rate every set number of epochs, to adapt the learning to the current state of the model ; - doing each epoch with a set of learning rates, and every time choosing the best result ; even if costly (every epoch is done multiple time), it should allow a really fast learning with lowered chances of divergence or over-fitting.

Personally, my preferred option is the second one.

B.8 Effets de l'optimisation du taux d'apprentissage

Learning rate optimisation effect on learning curve

2018/07/24 - SYNALP - Esteban MARQUER

B.8.1 Context

The previous results of learning rate optimisation lead to a potentially optimal learning rate of 0.2 (instead of 0.001).

A run with the new learning rate and every other thing identical was done to compare performance to previous 500 epoch run. That specific run had a learning rate of 0.001.

B.8.2 Results

The new learning rate has two effects : 1. convergence is achieved in less than 100 epoch, compared to previous learning achieving convergence in more than 400 epochs; 2. a small but constant gap between training performance and validation performance, but it is not over-fitting (if both training and validation are constant, we can not conclude that the model over-fits).

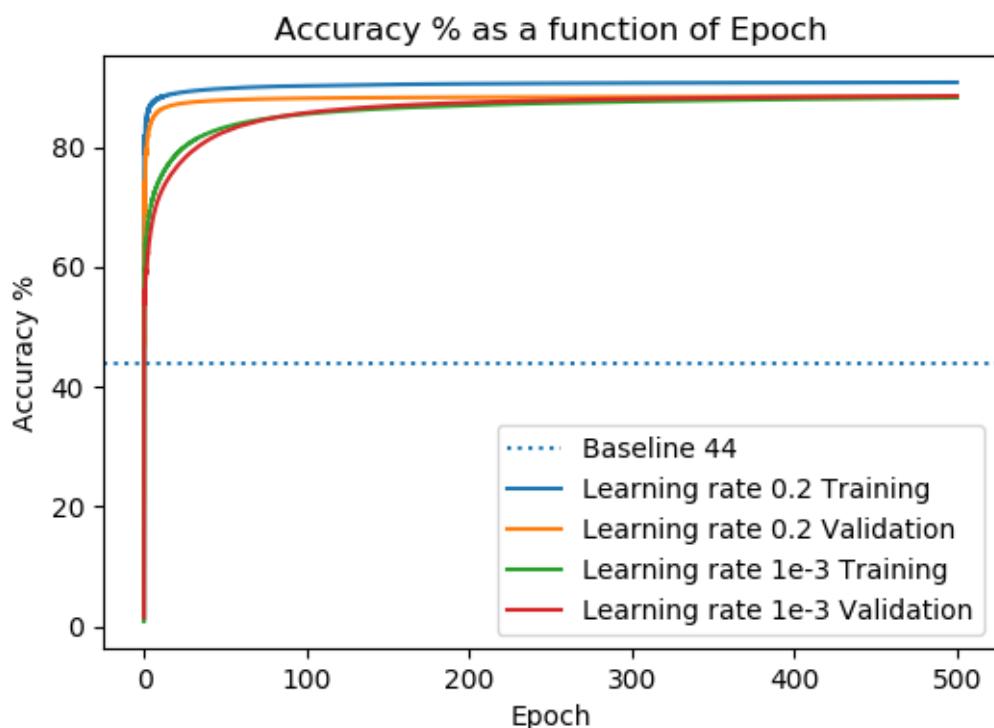


FIGURE B.4 – accuracy

B.8.3 Conclusion

The new learning rate is better than the previous one, it will be kept.

B.8.4 Improvements and next steps

[OPTIONAL] Use inline optimisation to update the learning rate every set number of epochs (once convergence is reached).

B.9 Taille de *batch* binaire

Binary batch size effect on run speed

2018/07/26 - SYNALP - Esteban MARQUER

B.9.1 Context

It has been said that batches using binary sizes (64, 256, 1024, ...) perform faster than non-binary sizes.

B.9.2 Paradigm

To verify this phenomenon and perhaps improve the training speed, a comparative experiment has been done with a batch size of 128 and a batch size of 200.

B.9.3 Results

There is no notable effect, except the effect predicted by the batch size comparison done previously, and stating that batches with a size of 200 are faster than with a size of 128.

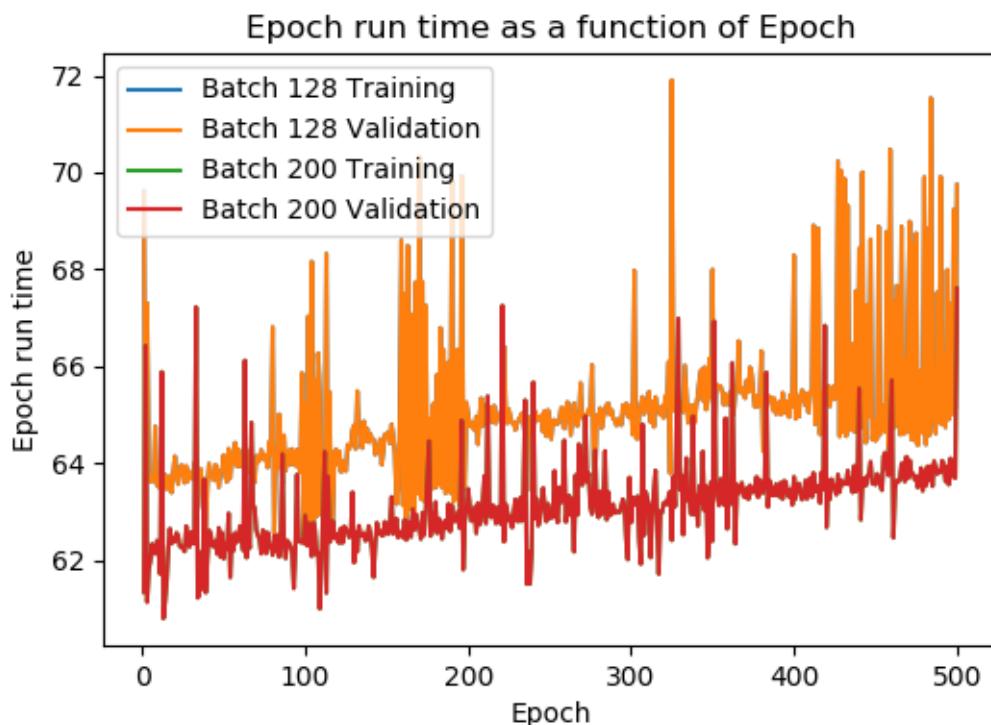


FIGURE B.5 – time per epoch

B.9.4 Conclusion

The most straightforward conclusion is that either the effect of binary sizes is negligible if not non-existent, or there is a hidden parameter changing the true size of the data (for example a set of bytes used to store metadata together with the regular data).

As the previous batch size (200) performs better, we will keep it.

B.9.5 Next steps and improvements

Investigating a potentially non-existent effect (at least with the current model) does not seem efficient considering the potential gain in computation time (compared to improving the naive model).

No further work will be done on that now (at least for now).

B.10 Performances du lecteur de corpus multi-fichiers multi-processus

Learning rate optimisation effect on

2018/07/26 - SYNALP - Esteban MARQUER

B.10.1 Context

Given the large amount of data to process, and the way it is structured in many small archives, a way to load and pre-process them efficiently had to be implemented.

B.10.2 Multi-file-multi-process corpus concept

Multi-process system

To avoid the need to completely pre-processing the data and storing it, and to shift the computational weight of the loading process, the different tasks are splitted between multiple processes.

At first, the idea was to use three processes, with one for the loading, one for the pre-processing, and the last one for the transformation into tensor. All those process fetch data from a multiprocess-safe queue and store the result in an output queue, used by the next process as an input.

A representation of the initial concept :

```
1 HDD --> Loader --> [ raw data queue ]
2      --> Processing --> [ processed data queue ]
3      --> Tensorising --> [ tensors queue ]
4      --> model
```

Given the performance of that system, the slowest part was the processing. Moreover, the processing could be splitted in multiple modules : removing the end-of-line characters, replacing patterns by tags and splitting into characters, transforming the data via a dictionary, and padding/cropping the sequence.

A representation of the modular processing concept :

```
1 HDD --> Loader --> [ raw Q. ]
2      --> Process 1 --> [ partially processed Q. 1 ]
3      --> Process 2 --> [ partially processed Q. 2 ]
4      ...
5      --> Process n --> [ processed Q. n ]
6      --> Tensorising --> [ tensors queue ]
7      --> model
```

This architecture allows to change the order of the pre-processing modules, and to add or remove some of them.

By splitting those different tasks on multiple processes, an efficient processing is achieved. Moreover, it is way easier to add pre-processing steps.

Multi-file system

By considering a set of files as a single sequence of line, and by loading only the one file containing the current data, combined by efficient line-by-line sequential reading, a light and fast loading is achieved.

B.10.3 Tests

Paradigm

While setting up the unitary tests for the corpus implementations, three main situations have produced useful insight on the performances of the new implementations.

Everything was run on my laptop, with other processes running that may have hindered the performance (for example, a heavy IDE).

Process-specific tests were run after every element was proven to do their expected job. Those tests added the processes and the data queues and information on queue filling and process state.

Results

Multiple test were done with very light, light treatment of the data, and real-situation training (the training is a way to process the data).

Printing only the status of the process at each batch Total time per epoch 12 s

```
1 { 'example': '28032/67923',
2   'batch': '218/527',
3   'iterator status':
4     'Process MultiFile status: process: alive; output queue: 1023/1024
5     Process EndLine status: process: alive; output queue: 1023/1024
6     Process Regex status: process: alive; output queue: 1024/1024
7     Process Dictionary status: process: alive; output queue: 2/1024
8     Process CropPad status: process: alive; output queue: 10/1024
9     Process Batch status: process: alive; output queue: 0/32'}
```

Printing the status of the process at each batch and printing the data Total time per epoch 47 s

```
1 { 'example': '30208/67923',
2   'batch': '235/527',
3   'iterator status':
4     'Process MultiFile status: process: alive; output queue: 1024/1024
5       Process EndLine status: process: alive; output queue: 1024/1024
6       Process Regex status: process: alive; output queue: 1023/1024
7       Process Dictionary status: process: alive; output queue: 916/1024
8       Process CropPad status: process: alive; output queue: 1024/1024
9       Process Batch status: process: alive; output queue: 32/32'}
```

Printing the status of the process at each batch and training the model Total time per epoch about 300 s, the old implementation needed about 200 s with the corpus full loaded in GPU RAM. CPU usage (usage mainly by the program sub-processes) : from 60% to 100%

```
1 { 'example': '26800/67923',
2   'batch': '134/338',
3   'iterator status':
4     'Process MultiFile status: process: alive; output queue: 1024/1024
5       Process EndLine status: process: alive; output queue: 1022/1024
6       Process Regex status: process: alive; output queue: 1017/1024
7       Process Dictionary status: process: alive; output queue: 905/1024
8       Process CropPad status: process: alive; output queue: 975/1024
9       Process Batch status: process: alive; output queue: 64/64'}
```

B.10.4 Conclusions

The first test shows the without processing the data, we can find where the data is “blocked”, and so find the slowest process in the bunch. Here, the slowest process is the transformation into ids.

As it is a simple dicitonary reading, which is already a very fast process in Python, if it is the slowest process of the chain, we can conclude that the basic performance of this implementation is very good.

When we do even a tny bit of processing (like printing the data), the data is blocked at the end of the chain, meaning the printing process is slower than the loading/pre-processing/tensorising process.

In a true training situation, we can confirm that processing the data is slower than pre-processing it. The only process slowing the whole training is the transfer to GPU RAM, which is not in a separate process yet (due to specificities of pytorch tensor mangement).

The implementation produce equivalent results with multipple files.

Globally, this new implementation should be enough to load and preprocess the data for the model, with both small and large datasets.

B.10.5 Next steps and improvements

- It should be possible to delegate the transfer to GPU RAM to a specific process (given the explanations on pytorch manual). This should reduce the gap between the speed achieved with pre-loaded data andthe speed of the new corpus system.
- The direct next step is to test the new corpus implementation on the computer cluster.
- Then, changing the current small corpus by a larger multi-file corpus will be possible.

C Copie de la convention de stage



CONVENTION DE STAGE

ENTRE

L'établissement d'enseignement supérieur :

Université de Lorraine, établissement public à caractère scientifique, culturel et professionnel, sis 34 Cours Léopold – CS 25233 – 54 052 NANCY Cedex, siren n° 130 015 506 00012, représenté par son Président, Monsieur Pierre Mutzenhardt,

Représenté par : (nom du (de la) signataire de la convention) : Antoine TABBONE

Qualité du représentant : Directeur de l'UFR Mathématiques et Informatique

Composante /UFR/ : UFR Mathématiques et Informatique

Adresse : 56 bis, boulevard de Scarpone, 54000 Nancy

Tel : 03 72 74 16 18

L'organisme d'accueil :

Nom : LORIA UMR 7503

Adresse : Campus Scientifique BP 239, 54506 Vandoeuvre-les-Nancy

Tél : 03 83 59 20 00

Fax : _____

Mél : dirrection@loria.fr

Représenté par : (nom du signataire de la convention) : Jean-Yves MARION

Qualité du représentant : Directeur du LORIA

Nom du service dans lequel le stage sera effectué : Équipe SYNALP

Lieu du stage : (si différent de l'adresse de l'entreprise) _____

Et l'étudiant stagiaire :

Nom : MARQUER Prénom : Esteban

Sexe : F M né(e) le : 08 / 06 / 1997

Adresse : 6, Rue Cyfflé, 54000, Nancy

Tél : 06 78 09 35 84

Mél : marquer.esteban7@etu.univ-lorraine.fr

Intitulé complet de la formation ou du cursus suivi dans l'établissement d'enseignement supérieur et son volume horaire :

Licence L3 MIASHS: MIAGE / Sciences cognitives Volume horaire : 600 h de présence / an

SUJET DE STAGE : Interpréter les couches cachées des réseaux récurrents en TAL

DATES DE STAGE : Du 23 / 04 / 2018 Au 27 / 07 / 2018

DUREE TOTALE DU STAGE * : 3 Heures ou Semaines ou Mois (rayer la mention inutile) soit en JOURS : 66

*7 heures = 1 jour et 22 jours = 1 mois

Encadrement du stagiaire assuré par :

L'établissement d'enseignement supérieur en la personne de :

Nom : THOMANN

Prénom : Laurent

Fonction : Professeur. Responsable des stages

Tél : 03 72 74 16 24

Mél : Laurent.thomann@univ-lorraine.fr

Caisse primaire d'assurances maladie à contacter en cas d'accident (lieu de domicile de l'étudiant sauf exception) :

¹ Article L612-9 du code de l'éducation : La durée du ou des stages effectués par un même stagiaire dans une même entreprise ne peut excéder six mois par année d'enseignement.

Article 1 : Objet de la convention

La présente convention règle les rapports de l'organisme d'accueil (entreprise, organisme public, association...) avec l'établissement d'enseignement supérieur et le stagiaire.

Article 2 : Objectif du stage

Le stage correspond à une période temporaire de mise en situation en milieu professionnel au cours de laquelle l'étudiant(e) acquiert des compétences professionnelles et met en œuvre les acquis de sa formation en vue de l'obtention d'un diplôme ou d'une certification et de favoriser son insertion professionnelle. Le/la stagiaire se voit confier une ou des missions conformes au projet pédagogique défini par son établissement d'enseignement et approuvées par l'organisme d'accueil.

Le programme est établi par l'établissement d'enseignement et l'organisme d'accueil en fonction du programme général de la formation dispensée.

Activités confiées : état de l'art sur les réseaux récurrents (RNN)
et leurs méthodes d'analyse ; récupération de programmes RNN
en pytorch ; analyse et interprétation des informations apprises
dans les couches cachées

Compétences à acquérir ou à développer :

maîtrise du deep learning ; programmation python

Article 3 : Modalité du stage

La durée hebdomadaire maximale de présence du (de la) stagiaire dans l'entreprise sera de 35 heures.

Le stage est à :
 Temps complet Temps partiel

Si temps partiel, préciser la quotité : _____

Si le (la) stagiaire doit être présent(e) dans l'organisme d'accueil la nuit, le dimanche ou un jour férié, l'organisme doit indiquer ci-après les cas particuliers : _____

Article 4 : Statut du stagiaire – Accueil et encadrement

L'étudiant(e), pendant la durée de son stage dans l'organisme d'accueil, conserve son statut antérieur; il (elle) est suivi(e) régulièrement par l'enseignant référent désigné dans la présente convention ainsi que par l'établissement. L'organisme d'accueil nomme un tuteur organisme d'accueil chargé d'assurer le suivi et d'optimiser les conditions de réalisation du stage. L'étudiant(e) pourra revenir à l'établissement pendant la durée du stage, pour y suivre certains cours demandés explicitement par le programme, participer à des réunions, les dates étant portées à la connaissance de l'organisme d'accueil par l'établissement et être autorisé, le cas échéant, à se déplacer.

Toute difficulté survenue dans la réalisation et le déroulement du stage ou, qu'elle soit constatée par le/la stagiaire ou par le tuteur de stage, doit être portée à la connaissance de l'enseignant référent et de l'établissement d'enseignement afin d'être résolue au plus vite.

Modalités d'encadrement : _____

Article 5 : Gratification – Avantages en nature Remboursement de frais

Lorsque la durée du stage est supérieure à deux mois consécutifs ou non, celui-ci fait obligatoirement l'objet d'une gratification sauf en cas de règles particulières applicables dans certaines collectivités d'outre-mer françaises et pour les stages relevant de l'article L4381-1 du code de la santé publique.

Le montant horaire de la gratification est fixé à 15 % du plafond horaire de la sécurité sociale défini en application de l'article L.241-3 du code de la sécurité sociale. Une convention de branche ou un accord professionnel peut définir un montant supérieur à ce taux.

La gratification est due à compter du premier jour du premier mois de la période de stage.

La gratification ne peut être cumulée avec une rémunération versée par l'administration ou l'établissement public d'accueil au cours de la période concernée.

La gratification est due au stagiaire sans préjudice du remboursement des frais engagés par le/la stagiaire pour effectuer son stage et des avantages offerts, le cas échéant, pour la restauration, l'hébergement et le transport.

L'organisme peut décider de verser une gratification pour les stages dont la durée est inférieure ou égale à deux mois.

En cas de suspension ou de résiliation de la présente convention, le montant de la gratification due au/à la stagiaire est proratisé en fonction de la durée du stage effectué.

La durée donnant droit à gratification s'apprécie compte tenu de la présente convention et de ses avenants éventuels, ainsi que du nombre de jours de présence effective du/de la stagiaire dans l'organisme.

Montant de la gratification (si différent du montant légal)

Modalités de versement de la gratification : _____

Le/la stagiaire bénéficie des protections et droits mentionnés aux articles L.1121-1, L.1152-1 et L.1153-1 du code du travail, dans les mêmes conditions que les salariés.

Le/la stagiaire a accès au restaurant d'entreprise ou aux titres-restaurants prévus à l'article L.3262-1 du code du travail, dans les mêmes conditions que les salariés de l'organisme d'accueil. Il/elle bénéficie également de la prise en charge des frais de transport prévue à l'article L.3261-2 du même code.

Les stagiaires accèdent aux activités sociales et culturelles mentionnées à l'article L.2323-83 du code du travail dans les mêmes conditions que les salariés.

Liste des avantages offerts : _____

Les trajets effectués par les stagiaires d'un organisme de droit public entre leur domicile et leur lieu de stage peuvent être pris en charge dans les conditions fixées par le décret n°2010-676 du 21 juin 2010 instituant une prise en charge partielle du prix des titres d'abonnement correspondant aux déplacements effectués par les agents publics entre leur résidence habituelle et leur lieu de travail.

la stagiaire accueilli(e) dans un organisme de droit public et qui effectue une mission dans ce cadre bénéficie des dispositions du décret n°2006-781 du 3 juillet 2006 fixant les conditions et les modalités de règlement des frais occasionnés par déplacements temporaires des personnels civils de l'Etat.

Est considéré comme sa résidence administrative le lieu du stage indiqué dans la présente convention.

Autres avantages :

Autre avantage : _____

Article 6 : Protection sociale

Pendant la durée du stage, l'étudiant(e) reste affilié(e) à son système de sécurité sociale antérieur : il(elle) conserve son statut étudiant. Les stages effectués à l'étranger doivent avoir été signalés préalablement au départ de l'étudiant(e) et avoir reçu l'agrément de la Sécurité Sociale. Les dispositions suivantes sont applicables sous réserve de conformité avec la législation du pays d'accueil et de celle régissant le type d'organisme d'accueil :

6.1 Gratification inférieure ou égale au produit de 15 % du plafond horaire de la sécurité sociale par le nombre d'heures de stage effectuées au cours du mois considéré :

Dans ce cas, conformément à la législation en vigueur, la gratification de stage n'est pas soumise à cotisation sociale. L'étudiant(e) continue à bénéficier de la législation sur les accidents de travail au titre de l'article L 412-8-2 du code de la Sécurité Sociale, régime étudiant. En cas d'accident survenant à l'étudiant(e), soit au cours des travaux dans l'organisme, soit au cours du trajet, soit sur les lieux rendus utiles pour les besoins de son stage et pour les étudiant(e)s en médecine, en chirurgie dentaire ou en pharmacie qui n'ont pas un statut hospitalier, du stage hospitalier effectué dans les conditions prévues au b du 2o de l'article L. 412-8, l'organisme d'accueil envoie la déclaration à la Caisse Primaire d'Assurance Maladie (voir adresse en première page) en mentionnant l'établissement comme employeur, avec copie à l'établissement.

6.2 Gratification supérieure au produit de 15 % du plafond horaire de la sécurité sociale par le nombre d'heures de stage effectuées au cours du mois considéré :

Les cotisations sociales sont calculées sur le différentiel entre le montant de la gratification et 15 % du plafond horaire de la Sécurité Sociale pour une durée légale de travail hebdomadaire de 35 heures. L'étudiant(e) bénéficie de la couverture légale en application des dispositions des articles L 411-1 et suivants du code de la Sécurité Sociale. En cas d'accident survenant à l'étudiant(e), soit au cours des travaux dans l'organisme, soit au cours du trajet, soit sur des lieux rendus utiles pour les besoins de son stage, l'organisme d'accueil effectue toutes les démarches nécessaires auprès de la Caisse Primaire d'Assurance Maladie et informe l'établissement dans les meilleurs délais.

6.3 Protection Maladie du stagiaire à l'étranger :

1) Protection issue du régime étudiant(e) français :

- Pour les stages au sein de l'Espace Economique Européen (EEE) effectués par les étudiant(e)s de nationalité d'un pays membre de l'Union Européenne, l'étudiant doit demander la Carte Européenne d'Assurance Maladie (CEAM).

- Pour les stages effectués au Québec par les étudiant(e)s de nationalité française, l'étudiant doit demander le formulaire SE401Q (104 pour les stages en entreprise, 106 pour les stages en université).

- Dans tous les autres cas de figure :

Les étudiant(e)s qui engagent des frais de santé à l'étranger peuvent être remboursé(e)s auprès de la mutuelle qui leur tient lieu de Caisse de Sécurité Sociale étudiante, au retour, et sur présentation des justificatifs : le remboursement s'effectue alors sur la base des tarifs de soins français, des écarts importants peuvent exister. Il est donc fortement recommandé à l'étudiant(e) de souscrire une assurance Maladie complémentaire spécifique, valable pour le pays et la durée du stage, auprès de l'organisme d'accueil de son choix (mutuelle étudiante, mutuelle des parents, compagnie privée ad hoc...).

Exception : si l'organisme d'accueil fournit à l'étudiant(e) une couverture Maladie en vertu des dispositions du droit local (voir 2 ci-dessous), alors l'étudiant(e) peut choisir de bénéficier de cette protection Maladie locale. Avant d'effectuer un tel choix, il vérifiera l'étendue des garanties proposées.

2) Protection issue de l'organisme d'accueil :

En cochant la case appropriée, l'organisme d'accueil indique ci-après s'il fournit une protection Maladie au stagiaire, en vertu du droit local :

OUI (celle-ci s'ajoute au maintien, à l'étranger, des droits issus du régime français étudiant)

NON (la protection découle alors exclusivement du maintien, à l'étranger, des droits issus du régime français étudiant)

Si aucune case n'est cochée, le 6.3 s'applique.

6.4 Protection Accident du Travail du stagiaire à l'étranger :

1) Pour pouvoir bénéficier de la législation française sur la couverture accident de travail, le présent stage doit :

- Etre d'une durée au plus égale à 6 mois, prolongations incluses.
- Ne donner lieu à aucune rémunération susceptible d'ouvrir des droits à une protection accident de travail dans le pays étranger (une indemnité ou gratification est admise à hauteur de 15 % du plafond horaire de la sécurité sociale pour une durée légale hebdomadaire de 35 heures sous réserve de l'accord de la Caisse Primaire d'Assurance Maladie).
- Se dérouler exclusivement dans l'entreprise partie à la présente convention.
- Se dérouler exclusivement dans le pays étranger cité.

Lorsque les conditions ne sont pas remplies, l'organisme d'accueil s'engage à cotiser pour la protection du stagiaire et à faire les déclarations nécessaires en cas d'accident de travail.

2) La déclaration des accidents de travail incombe à l'établissement qui doit être informé par l'organisme d'accueil par écrit dans un délai de 48 heures.

3) La couverture concerne les accidents survenus :

- Dans l'enceinte du lieu du stage et aux heures de stage.
- Sur le trajet aller-retour habituel entre la résidence du stagiaire sur le territoire étranger et le lieu du stage.
- Sur le trajet aller-retour (début et fin de stage) du domicile du stagiaire situé sur le territoire français et le lieu de résidence à l'étranger.
- Dans le cadre d'une mission confiée par l'organisme d'accueil et obligatoirement par ordre de mission.

4) Pour le cas où l'une seule des conditions prévues au point 6.4 1/ n'est pas remplie, l'organisme d'accueil s'engage par la présente convention à couvrir le stagiaire contre le risque d'accident de travail, de trajet et les maladies professionnelles et à en assurer toutes les déclarations nécessaires.

5) dans tous les cas,

- Si l'étudiant(e) est victime d'un accident du travail durant le stage, l'organisme d'accueil doit impérativement signaler immédiatement cet accident à l'établissement.
- Si l'étudiant(e) remplit des missions limitées en-dehors de l'organisme d'accueil ou en en-dehors du pays du stage, l'organisme d'accueil doit prendre toutes les dispositions nécessaires pour lui fournir les assurances appropriées.

Article 7 : Responsabilité civile et assurances

L'organisme d'accueil et l'étudiant(e) déclarent être garantis au titre de la responsabilité civile. Quelle que soit la nature du stage et le pays de destination, le(la) stagiaire s'engage à se couvrir par un contrat d'assistance (rapatriement sanitaire, assistance juridique etc.) et par un contrat d'assurance individuel accident. Lorsque l'organisme d'accueil met un véhicule à la disposition du(de la) stagiaire, il lui incombe de vérifier préalablement que la police d'assurance du véhicule couvre son utilisation par un étudiant. Lorsque dans le cadre de son stage, l'étudiant(e) utilise son propre véhicule ou un véhicule, prêté par un tiers, il(elle) déclare expressément à l'assureur dudit véhicule cette utilisation qu'il(elle) est amené à faire et le cas échéant s'acquitte de la prime y afférente.

Article 8 : Discipline

Durant son stage, l'étudiant(e) est soumis à la discipline et au règlement intérieur (qui doit être porté à la connaissance de l'étudiant(e)) de l'organisme, notamment en ce qui concerne les horaires, et les règles d'hygiène et de sécurité en vigueur dans l'organisme d'accueil. Toute sanction disciplinaire ne peut être décidée que par l'établissement. Dans ce cas, l'organisme d'accueil informe l'établissement des manquements et lui fournit éventuellement les éléments constitutifs. En cas de manquement particulièrement grave à la discipline, l'organisme d'accueil se réserve le droit de mettre fin au stage de l'étudiant(e) tout en respectant les dispositions fixées à l'article 9 de la présente convention.

Article 9 : Absence et Interruption du stage

Toute difficulté survenue dans le déroulement du stage devra être portée à la connaissance de tous les intéressés afin d'être résolue au plus vite.

En France (sauf en cas de règles particulières applicables dans certaines collectivités d'outre-mer françaises), en organisme de droit privé, en cas de grossesse, de paternité ou d'adoption, le/la stagiaire bénéficie de congés et d'autorisations d'absence d'une durée équivalente à celle prévues pour les salariés dans les organismes de droit privé aux articles L.1225-16 à L.1225-28, L.1225-35, L.1225-46 du code du travail.

Pour les stages dont la durée est supérieure à deux mois et dans la limite de la durée maximale de 6 mois, des congés ou autorisations d'absence sont possibles.

NOMBRE DE JOURS DE CONGES AUTORISES / ou modalités des congés et autorisations d'absence durant le stage :

Pour toute autre interruption temporaire du stage (maladie, absence injustifiée...) l'organisme d'accueil avertit l'établissement d'enseignement par courrier.

Toute interruption du stage, est signalée aux autres parties à la convention et à l'enseignant référent. Une modalité de validation est mise en place le cas échéant par l'établissement d'enseignement supérieur. En cas d'accord des parties à la convention, un report de la fin du stage est possible afin de

permettre la réalisation de la durée totale du stage prévue initialement. Ce report fera l'objet d'un avenant à la convention de stage.

Un avenant à la convention pourra éventuellement être établi en cas de prolongation du stage sur demande conjointe de l'organisme d'accueil et du(de la) stagiaire, dans le respect de la durée maximale du stage fixée par la loi (6 mois).

Interruption définitive :

En cas de volonté d'une des trois parties (organisme d'accueil, établissement, étudiant(e)) d'interrompre définitivement le stage, celle-ci devra immédiatement en informer les deux autres parties par écrit. Les raisons invoquées seront examinées en étroite concertation. La décision définitive d'interruption du stage ne sera prise qu'à l'issue de cette phase de concertation.

Article 10 : Devoir de réserve et confidentialité

Le devoir de réserve est de rigueur absolue. Les étudiant(e)s stagiaires prennent donc l'engagement de n'utiliser en aucun cas les informations recueillies ou obtenues par eux pour en faire l'objet de publication, communication à des tiers sans accord préalable de l'organisme d'accueil, y compris le rapport de stage. Cet engagement vaudra non seulement pour la durée du stage mais également après son expiration. L'étudiant(e) s'engage à ne conserver, emporter, ou prendre copie d'aucun document ou logiciel, de quelque nature que ce soit, appartenant à l'organisme d'accueil, sauf accord de ce dernier.

Nota : Dans le cadre de la confidentialité des informations contenues dans le rapport, l'organisme d'accueil peut demander une restriction de la diffusion du rapport, voire le retrait de certains éléments très confidentiels.

Les personnes amenées à en connaître sont contraintes par le secret professionnel à n'utiliser ni ne divulguer les informations du rapport.

Article 11 : Propriété intellectuelle

Conformément au code de la propriété intellectuelle, si le travail du stagiaire donne lieu à la création d'une œuvre protégée par le droit d'auteur ou la propriété industrielle (y compris un logiciel), si l'organisme d'accueil souhaite l'utiliser et que le stagiaire est d'accord, un contrat devra être signé entre le stagiaire (auteur) et l'organisme d'accueil. Devront notamment être précisés l'étendue des droits cédés, l'éventuelle exclusivité, la destination, les supports utilisés et la durée de la cession, ainsi que, le cas échéant, le montant de la rémunération due à l'étudiant au titre de la cession. Cette clause s'applique également dans le cas des stages dans les Organismes publics.

Article 12 : Recrutement

S'il advenait qu'un contrat de travail prenant effet avant la date de fin du stage soit signé avec l'organisme d'accueil, la présente convention deviendrait caduque ; l'« étudiant(e) » ne relèverait plus de la responsabilité de l'établissement d'enseignement. Ce dernier devrait impérativement en être averti avant la signature du contrat.

Article 13 : Fin de stage – Rapport – Evaluation

A l'issue du stage, l'organisme d'accueil délivre au stagiaire une attestation de stage et remplit une fiche d'évaluation de l'activité du stagiaire mentionnant au minimum la durée effective du stage et, le cas échéant le montant de la gratification perçue qu'il retourne à l'établissement d'enseignement supérieur.

Le(la) stagiaire devra produire cette attestation à l'appui de sa demande éventuelle d'ouverture de droits au régime général d'assurance vieillesse prévue à l'art. L.351-17 du code de la sécurité sociale ;

A l'issue du stage, les parties à la présente convention sont invitées à formuler une appréciation sur la qualité du stage.

Le(la) stagiaire transmet au service compétent de l'établissement d'enseignement un document dans lequel il(elle) évalue la qualité de l'accueil dont il(elle) a bénéficié au sein de l'organisme d'accueil. Ce document n'est pas pris en compte dans son évaluation ou dans l'obtention du diplôme ou de la certification

A l'issue de son stage l'étudiant devra : (préciser la nature du travail à fournir éventuellement en joignant une annexe)

Préciser le cas échéant les modalités de validation du stage :

Nombre de crédits ECTS : 6

Le tuteur organisme d'accueil ou tout autre membre de l'organisme d'accueil appelé à se rendre à l'établissement dans le cadre de la préparation, du déroulement et de la validation du stage ne peut prétendre à une quelconque prise en charge ou indemnisation de la part de l'établissement.

Un avenant à la convention pourra éventuellement être établi en cas de prolongation de stage faite à la demande de l'organisme et de l'étudiant(e). En aucun cas la date de fin de stage ne pourra être postérieure au 30/09 de l'année en cours.

L'accueil successif de stagiaires, au titre de conventions de stage différentes, pour effectuer des stages dans un même poste n'est possible qu'à l'expiration d'un délai de carence égal au tiers de la durée du stage précédent. Cette disposition n'est pas applicable lorsque ce stage précédent a été interrompu avant son terme à l'initiative du stagiaire.

Article 14 : Droit applicable – Tribunaux compétents

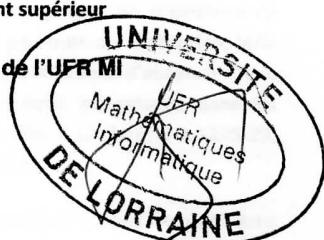
La présente convention est régie exclusivement par le droit français. Tout litige non résolu par voie amiable sera soumis à la compétence de la juridiction française compétente.

A Nancy le 12/04/18

Pour l'établissement d'enseignement supérieur

(nom et signature du représentant)

Antoine TABBONE, directeur de l'UFR MI



Pour l'organisme d'accueil

(nom et signature du représentant)

Le directeur du LORIA

Jean-Yves MARION



Pour l'étudiant

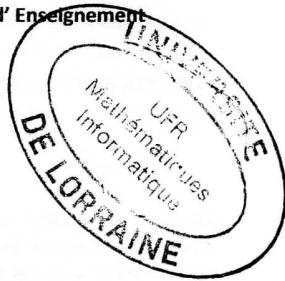
(nom et signature)

MARQUE Escobar

VISAS :

L'enseignant référent de l'Etablissement d'Enseignement Supérieur
(nom et signature)

Laurent THOMANN



Tuteur Organisme d'accueil

(nom et signature)

CERISARA Christophe Cenier

Annexe 1 : Charte des stages / Annexe 2 : Fiches d'évaluation / Annexe 3 à fournir par l'étudiant(e) : Attestation de responsabilité civile



D Copie de l'avenant à la convention de stage



UNIVERSITÉ
DE LORRAINE

AVENANT A LA CONVENTION DE STAGE

Signée le 12.10.18

ENTRE

L'établissement d'enseignement supérieur :

Université de Lorraine, établissement public à caractère scientifique, culturel et professionnel, sis 34 Cours Léopold – CS 25233 –

54 052 NANCY Cedex, siret n° 130 015 506 00012, représenté par son Président, Monsieur Pierre Mutzenhardt,

Représenté par : (nom du (de la) signataire de la convention) : Antoine TABBONE

Qualité du représentant : Directeur de l'UFR Mathématiques et Informatique

Composante /UFR/ : VER Mathématiques et Informatique

Adresse d'envoi de la convention à compléter obligatoirement par la composante :

56 bis, boulevard de Scarpone, 54 000 Nancy

L'organisme d'accueil :

Nom :

LORIA UMR 7503

Adresse : Campus Scientifique BP239, 54 506 Vandoeuvre-les-Nancy

Tél : 03 83 59 20 00 Fax : _____ Mél : direction@loria.fr

Représenté par : (nom du signataire de la convention) : Jean-Yves MARION

Qualité du représentant : Directeur du LORIA

Nom du service dans lequel le stage sera effectué : Equipe SYNALP

Lieu du stage : (si différent de l'adresse de l'entreprise)

Et l'étudiant stagiaire :

Nom : MARQUER Prénom : Estrhan

Sexe : F M né(e) le : 08/06/1997

Adresse : 6, rue CYFFÉ, 54 000 Nancy

Tél : _____ Mél : _____

Intitulé complet de la formation ou du cursus suivi dans l'établissement d'enseignement supérieur et son volume horaire :

Licence L3 MIASHS : MIAGE / Sciences Cognitives Volume horaire : 600h de présence/an

SUJET DE STAGE :

Interpréter les couches cachées des réseaux récurrents en TAL

DATES DE STAGE : Du 27/04/2018 Au 03/08/2018

DUREE DU STAGE * : 1 Heures ou Semaines ou Mois (rayer la mention inutile) soit en JOURS : 5

*7 heures = 1 jour et 22 jours = 1 mois

Encadrement du stagiaire assuré par :

L'établissement d'enseignement supérieur en la personne de :

Nom : THOMANN

Prénom : Laurent

Fonction : Professeur, Responsable des Stages

Tél : 03 72 74 16 24

Mél : laurent.thomann@univ-lorraine.fr

L'organisme d'accueil en la personne de :

Nom : CERISARA

Prénom : Christophe

Fonction : Responsable équipe SYNALP

Tél : 03 54 95 86 25

Mél : cerisara@loria.fr

PREAMBULE

Vu la loi n°2014-788 du 10 juillet 2014 tendant au développement, à l'encadrement des stages et à l'amélioration du statut des stagiaires,

Vu le décret n°2014-1420 du 27 novembre 2014 relatif à l'encadrement des périodes de formation en milieu professionnel et des stages,

Les parties ont signé une convention de stage en date du 12/04/2015. Elles souhaitent aujourd'hui préciser les modifications suivantes.

Article 1 – Modification de l'article 5 :

L'alinéa 2 de article est modifié comme suit :

Article 5 – Gratification - Avantages

(...)

Le montant horaire de la gratification est fixé à 15 % du plafond horaire de la sécurité sociale défini en application de l'article L.241-3 du code de la sécurité sociale. Une convention de branche ou un accord professionnel peut définir un montant supérieur à ce taux.

(...)

Article 2 – Modification de l'article 6 :

La présente convention règle les rapports de l'organisme d'accueil avec l'établissement d'enseignement et le/la stagiaire.

L'article est modifié comme suit :

Article 6 – Régime de protection sociale

(...)

6.1 Gratification inférieure ou égale à 15 % du plafond horaire de la sécurité sociale :

(...)

6.2 – Gratification supérieure à 15 % du plafond horaire de la sécurité sociale :

Les cotisations sociales sont calculées sur le différentiel entre le montant de la gratification et 15 % du plafond horaire de la Sécurité Sociale.

(...)

6.4 Protection Accident du Travail du stagiaire à l'étranger

I) Pour pouvoir bénéficier de la législation française sur la couverture accident de travail, le présent stage doit :

(...)

- ne donner lieu à aucune rémunération susceptible d'ouvrir des droits à une protection accident de travail dans le pays d'accueil ; une indemnité ou gratification est admise dans la limite de 15 % du plafond horaire de la sécurité sociale (cf point 5), et sous réserve de l'accord de la Caisse Primaire d'Assurance Maladie ;

Article 3 – Modification de l'article 9 :

La présente convention règle les rapports de l'organisme d'accueil avec l'établissement d'enseignement et le/la stagiaire.

L'article est modifié comme suit :

Article 9 – Congés – Interruption du stage

(...)

Pour toute autre interruption temporaire du stage (maladie, absence injustifiée...) l'organisme d'accueil avertit l'établissement d'enseignement par courrier.

Toute interruption du stage, est signalée aux autres parties à la convention et à l'enseignant référent. Une modalité de validation est mise en place le cas échéant par l'établissement d'enseignement supérieur. En cas d'accord des parties à la convention, un report de la fin du stage est possible afin de permettre la réalisation de la durée totale du stage prévue initialement. Ce report fera l'objet d'un avenant à la convention de stage.

(...)

Article 4 –Dispositions finales :

Les autres dispositions de la convention de stage initiales restent inchangées.

Fait à Nancy Le _____

POUR L'ÉTABLISSEMENT D'ENSEIGNEMENT

Nom et signature du représentant de l'établissement

STAGIAIRE (OU SON REPRESENTANT LEGAL LE CAS ÉCHÉANT)

Nom et signature

MARION YVES STEPHAN

POUR L'ORGANISME D'ACCUEIL

Nom et signature du représentant de l'organisme d'accueil

Le directeur du LORIA
Jean-Yves MARION

L'enseignant référent du stagiaire

Nom et signature

CERISARA Christophe
Perison

Le tuteur de stage de l'organisme d'accueil

Nom et signature