REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE SAAD DAHLAB DE BLIDA 1
FACULTE DES SCIENCES
DEPARTEMENT INFORMATIQUE

Projet de Fin d'Etudes
Pour l'obtention du Diplôme de Master en Informatique
Option : Systèmes Informatiques et Réseaux

Thème

Application du Machine Learning à l'analyse et à la prédiction des défaillances dans les systèmes Calcul à Hautes Performances (HPC)

Travail réalisé par :

M. GUEDDOUD Seddik
M. ALLACHE Youcef

Soutenu le : 04/10/2021

Devant le Jury composé de :

Président:	**Dr. CHIKHI N.**	**MCA**	**USDB1**
Examinateur:	**Dr. YAKHLEF H.**	**MCB**	**USDB1**
Promoteur:	**Dr. ZEKRIFA D.M.S.**	**MCB**	**ESSAIA**

Année universitaire 2020-2021

Remerciements

A notre promoteur

Nous adressons tout d'abord nos sincères remerciements à notre chère promoteur ***Dr. ZEKRIFA D.M.S.****, Maître de conférences à l'ESSAIA qui nous a témoigné son soutien et nous a prodigué une orientation judicieuse et rigoureuse durant toutes les phases de ce mémoire. Qu'elle trouve ici l'expression de nos sincères gratitudes.*

Aux membres du jury

Nos vifs remerciements vont à ***Mr CHIKHI N.****, Maître de conférences à l'USDB pour nous avoir fait l'honneur de présider le jury. Veuillez trouver, cher Professeur, le témoignage de notre grande reconnaissance et de notre profond respect.*

Nos vifs remerciements vont à ***Mme YAKHLEF H.*** *Maître de conférences à l'USDB1, pour nous avoir fait l'honneur de siéger dans ce jury en tant qu'examinateur. Veuillez croire à l'assurance de notre respect et de notre reconnaissance.*

Nous remercions le Professeur ***OULD KHAOUA M.*** *une encyclopédie qui enveloppe : savoir sagesse et persévérance pour donner le meilleur, ainsi que tous les enseignants du département informatique de l'université de Blida qui nous ont aidés de près ou de loin durant notre cursus d'études supérieurs*

Enfin, nous adressons nos plus sincères remerciements à nos familles : Nos parents,ma femme et nos sœurs, nos frères, et tous nos proches et amis, qui nous ont accompagnés, aidé, soutenu et encouragé tout le long de la réalisation de ce mémoire.

Résumé

Actuellement avec l'évolution des performances de calcul des systèmes HPC et lors de la récupération des composants défaillants une importante capacité et puissance de calcul sont perdues. Pour rendre ces systèmes tolérants aux pannes des applications sont proposées et exploitées, ainsi que les approches de récupération telles que les points de contrôle, le redémarrage et une meilleure compréhension des journaux du système sont proposées et exploitées. Une solution alternative et désormais nécessaire est la prédiction de la défaillance avec un délai défini et l'identification du nœud sur lequel elle va se produire.

L'objectif de ce travail est d'abord d'étudier/optimiser les journaux du supercalculateur MIRA, puis de développer une méthode générale de gestion des journaux.

Pour le faire, les journaux sont d'abord analysés et visualisés à l'aide de python3 et de pandas, puis la mémoire LSTM est utilisée pour prédire les chaînes de défaillance.Une analyse des journaux non étiquetés, qui peuvent ou non conduire à une chaîne de défaillance, est effectuée. Il existe une approche d'apprentissage profond en trois phases, une formation est d'abord effectuée pour prédire les phrases suivantes, puis un réapprentissage est effectué uniquement pour les séquences de phrases conduisant à des chaînes d'échec augmentées par les délais d'exécution prévus et, enfin, le délai d'exécution pendant le test est prédit pour prévoir quel nœud particulier échoue et en combien de minutes.

Mots Clé: Chaines de defaillances, délai d'attente, Intelligence Artificiel , Apprentissage profond, Traitement du Langage Naturel, Skip-Gram, LSTM.

Abstract

Currently with the evolution of the computational performance of HPC systems and during the recovery of failed components a significant computing capacity and power are lost. To make these systems tolerant to application failures are proposed and exploited, as well as recovery approaches such as control points, reboot and better understanding of system logs are proposed and exploited. An alternative and now necessary solution is the prediction of the failure with a defined delay and the identification of the node on which it will occur.

The objective of this thesis is first to study/optimize the logs of the MIRA supercomputer, and then to develop a general method for log management.

To do this, logs are first analyzed and visualized using python3 and pandas, and then LSTM memory is used to predict fault chains. An analysis of unlabelled logs, which may or may not lead to a fault chain, is performed. There is a deep learning approach in three phases, a training is first carried out to predict the following sentences, then a relearning is performed only for sentence sequences leading to failure chains increased by the expected execution times and, finally, the execution time during the test is predicted to predict which particular node fails and in how many minutes.

Key words: Failure Chains, Lead Time, Artificial Intelligence, Deep Learning, Natural Language Processing, Skip-Gram, LSTM.

ملخص

وفي الوقت الراهن ، ومع تطور الأداء الحسابي لنظم CPH وخلال استعادة المكونات الفاشلة ، فقدت قدرة وقدرة حاسوبية كبيرة. ومن المقترح والاستغلال لجعل هذه النظم متسامحة مع إخفاقات التطبيقات ، فضلا عن نهج الاسترداد مثل نقاط المراقبة وإعادة التشغيل وتحسين فهم سجلات النظم. والحل البديل والضروري الآن هو التنبؤ بالفشل مع تأخير محدد وتحديد العقدة التي سيحدث عليها.

الهدف من هذه الأطروحة هو أولًا دراسة / تحسين سجلات الكمبيوتر العملاق MIRA ، ثمتطوير طريقة عامة لإدارة السجل للقيام بذلك ، يتم تحليل وتصور السجلات أولا باستخدام البيثون3 والباندا ، ومن ثم يتم استخدام ذاكرة LSTM للتنبؤ بسلاسل . الخطأ. ويجري تحليل الجذوع غير المقيدة ، التي قد تؤدي أو لا تؤدي إلى سلسلة أخطاء. وهناك نهج عميق للتعلم على ثلاث مراحل ، ويجري أولا تدريب للتنبؤ بالجمل التالية: وبعد ذلك ، لا يُجرى النقل إلا لتسلسل الأحكام مما يؤدي إلى زيادة سلاسل الفشل حسب أوقات التنفيذ المتوقعة و ، وأخيرا ، يتوقع أن يتنبأ وقت التنفيذ أثناء الاختبار بأي عقدة معينة تفشل وكم دقيقة تفشل.

الكلمات المفتاحية:سلاسل الفشل,الذكاء الاصطناعي,التعلم العميق
.Skip-Gram,Natural Language Processing,Lead Time,LSTM

Table des matières

Contenu

Liste des figures

Liste des tableaux

Liste des abréviations

ALCF Argonne Leadership Computing Facility, 2

ANN Artificial neural network, 7

CBOW Continuous Bag of Words, 18

CDC Control Data Corporation, 1

CNN Convolutional neural network, 7

EQM Erreur quadratique moyenne, 68

FLOPS Floating-point operations per second, 1

FT Fine-tuning, 39

HPC Hight performance computing, 69

IA Intelligence artificielle, 4

IBM International Business Machines, 2

LSTM Long short term memory, 25

NLP Natural language processing, 8

RAS Reliability, Availability, Serviceability, 69

RMSprop Propagation de la moyenne des carrés, 56

RNN Recurrent neural network, 7

SGD Stochasticl gradient descent, 57

STL Sequential transfer learning, 37

SVM Support vector machine, 78

Chapitre 1 : Introduction Générale

1.1 Les Supercalculateurs (Supercomputers)

Un supercalculateur est un ordinateur doté d'un niveau de performance élevé par rapport à un ordinateur à usage conventionnelles (personal computer). Les performances d'un superordinateur sont généralement mesurées en opérations en virgule flottante par seconde (FLOPS) plutôt qu'en millions d'instructions par seconde (MIPS).

Les superordinateurs ont été introduits dans les années 1960 et pendant plusieurs décennies, les plus rapides ont été fabriqués par Seymour Cray chez Control Data Corporation (CDC). Depuis 2017, il existe des superordinateurs qui sont capables de réaliser plus de cent quadrillions de FLOPS (petaFLOPS). Depuis novembre 2017, tous les superordinateurs les plus rapides utilisent des systèmes d'exploitation basés sur Linux.

Pour construire des superordinateurs exascale plus rapides, plus puissants et technologiquement supérieurs, des recherches sont menées en Chine, aux États-Unis, dans l'Union européenne, en Australie, Taïwan et au Japon.

Les supercalculateurs jouent un rôle important dans le domaine de la science informatique et sont utilisés pour un large éventail de tâches à forte intensité de calcul dans différents domaines, notamment le calcul quantique. L'ingénierie, les prévisions météorologiques, la recherche sur le climat, l'exploration pétrolière et gazière, la modélisation moléculaire et les simulations physiques (telles que les simulations des premiers instants de l'univers, l'aérodynamique des avions et des vaisseaux spatiaux). Ils sont également essentiels dans le domaine de la cryptanalyse.

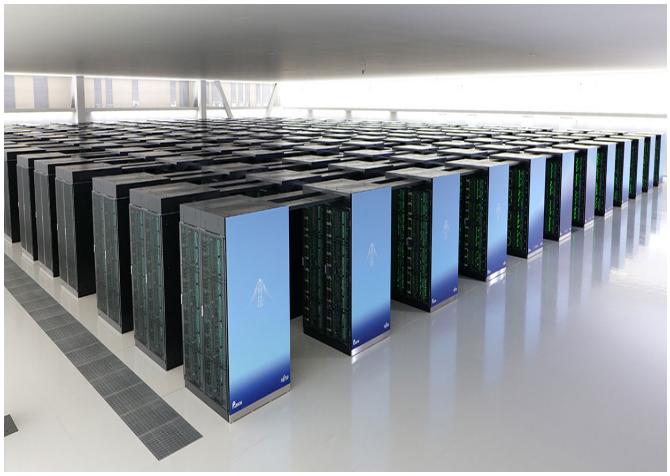


Figure 1.1 : Fugaku -Le supercalculateur le plus puissant du monde actuellement 2021

.

En particulier, au cours de la dernière décennie, des efforts considérables ont été déployés pour assurer la résilience de ces systèmes de calcul à haute performance. Parmi ces efforts, citons les approches de récupération réactive telles que le checkpointing et le redémarrage, qui permettent de mieux comprendre les journaux du système et de mieux appréhender les compromis entre performance, résilience et puissance. Ainsi, alors que les systèmes informatiques ont évolué en termes d'efficacité et d'architectures sophistiquées, la propagation des fautes est devenue plus complexe. Les infrastructures existantes de calcul à haute performance, avec le nombre toujours croissant de composants requis pour les nœuds exascale 10^{18}, rendent difficile la prédiction précise des défaillances. Les travaux abandonnés en raison de la défaillance de nœuds entraînent des coûts énergétiques importants. Plus de 20 % de la capacité informatique est gaspillée en pannes et en récupération (Elnozahy et al., 2008).

1.2 MIRA

Dans ce travail, les données de Mira ont été utilisées. La machine Mira est un superordinateur Blue Gene/Q pétascale. Ce superordinateur a été construit par IBM pour Argonne National Laboratory avec le soutien du département *de l'énergie des États-Unis* et est partiellement financé par la *National Science Foundation*. *L'Argonne National Laboratory* est un laboratoire national de recherche en sciences et en ingénierie exploité par *l'Université de Chicago* Argonne LLC pour le compte du *ministère de l'Énergie des États-Unis*, situé *à Lemont*, dans *l'Illinois*, à l'extérieur de *Chicago*. Il s'agit du plus grand laboratoire national par sa taille et sa portée dans le Midwest. Ce laboratoire mène des recherches dans un large éventail de disciplines, exploite de nombreuses installations de grande taille et coûteux pour les universités ou les entreprises et donne accès à des ressources qui sont généralement très diversifiées. L'une de ces installations est l'Argonne Leadership Computing Facility (ALCF), qui a hébergé le supercalculateur Mira jusqu'à la fin de 2019, date à laquelle il a été mis hors service pour être remplacé par un supercalculateur plus puissant, le Theta.

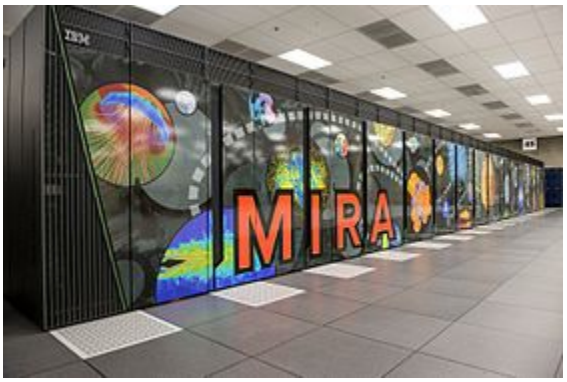


Figure 1.2: Blue Gene/Q supercomputer (”Mira”)

Mira est un système IBM Blue Gene de 10 pétaflops, avec 48 racks, 49 152 nœuds de calcul, équipés de 786 432 cœurs et de 768 téraoctets de mémoire. Il est composé de 16 cœurs et chacun des 512 nœuds forme un tore (grille) 5D dans une topologie 4x4x4x4x2 formant un plan médian. Chaque rack est composé de deux midplanes et le système compte 48 racks comme mentionné ci-dessus. Ces 48 racks de calcul sont désignés par R00-R0F, R10-R1F et R20-R2F et les deux plans médians de chaque rack sont désignés par M0 et M1. En novembre 2017, il figure au TOP500 en tant que 9ème superordinateur le plus rapide au monde, après avoir débuté en juin 2012 à la 3ème place. Il dispose d'une interconnexion à torus (5D torus), qui réduit le nombre moyen de sauts et la latence entre les nœuds de calcul, ce qui permet d'atteindre une grande efficacité de calcul et d'économiser de l'énergie pour le transfert de données sur de longues distances. La plus petite partition de Mira est de 512 nœuds. Les travaux dont la taille est inférieure à la partition minimale s'exécutent exclusivement sur une partition. La machine enregistre les événements des travaux par lots programmés, y compris les demandes et les ressources utilisées (par ex.) ainsi que les horodatages d'événements planifiés importants tels que l'heure de soumission, l'heure du début, l'heure de la fin, sans oublier l'enregistrement d'informations sur l'environnement d'exécution d'un travail telles que les partitions de la machine.

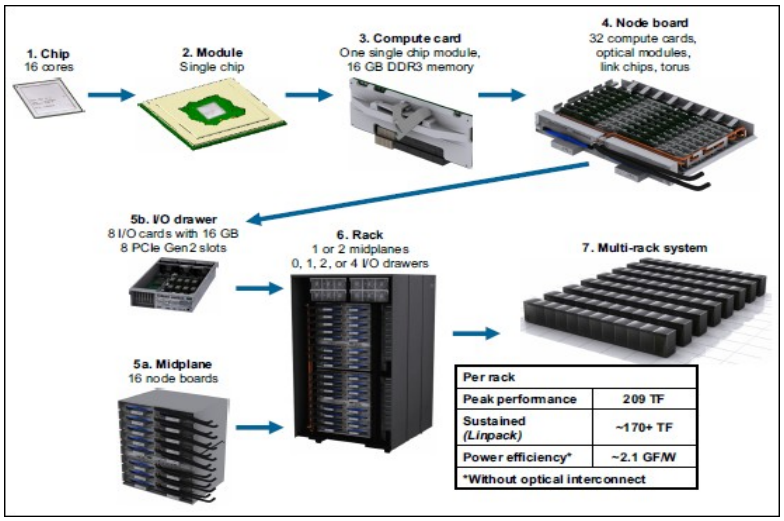


Figure 1.3: Blue Gene/Q Hardware Overview (Lakner et al., 2013).

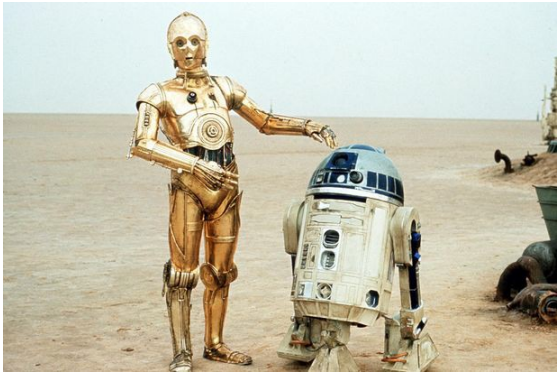
Parmi les développements de recherche les plus notables rendus possibles par Mira, citons :

1. Modélisation des supernovæ avec un niveau de détail sans précédent.
2. Simulation de collisions en grappe pour le grand collisionneur de hadrons.
3. Technique de microscopie avancée pour la détection précoce du cancer.
4. Accélérer la conception de moteurs plus efficaces.

1.3 Intelligence artificielle

Depuis l'Antiquité, des êtres artificiels capables de penser sont apparus dans des contes de fées et des histoires, mais l'intelligence artificielle (IA) moderne remonte aux années 1940. L'accès des chercheurs aux ordinateurs ainsi que l'introduction de nouvelles idées telles que la théorie des jeux de John von Neumann en 1944 et les réseaux neuronaux artificiels (McCulloch-Pitts, 1943) ont amené les chercheurs de divers domaines à envisager la possibilité de créer un cerveau artificiel ou une machine à penser. Après la Seconde Guerre mondiale, le mathématicien britannique Alan Turing a discuté, dans son article Computing Machinery and Intelligence, des conditions permettant de considérer une machine comme intelligente. Il a affirmé que si la machine pourrait réussir à se faire passer pour un être humain auprès d'un observateur expérimenté, il serait alors certainement considéré comme intelligent - le fameux test de Turing (Harasim, 2012). En 1956, le domaine de la recherche en IA est né lors d'une conférence au Darthmouth College, aux États-Unis d'Amérique. Au cours de la conférence, John McCarthy a défini l'IA comme la science et l'ingénierie de la création de machines intelligentes et en particulier de programmes intelligents (McCarthy et al., 2006). La plupart des participants à la conférence sont devenus les premiers chercheurs de cette nouvelle science et ont créé des laboratoires dans les grandes universités des États-Unis. À partir de ce moment et jusqu'au milieu des années 1970, la recherche dans le domaine de l'IA a été florissante. Les résultats ont été frappants : l'électronique, les ordinateurs apprenaient à résoudre des problèmes d'algèbre, à comprendre et à exécuter des commandes simples en langage naturel. Les financements était généreux et sans restriction, et les attentes des chercheurs étaient élevées : en 1967, Marvin Minsky déclarait qu'en l'espace d'une génération, le problème de la création de l'intelligence artificielle serait résolu (Minsky, 1967).

Figure1.4 : C-3PO et R2-D2 de Star Wars (imdb).



Cependant, la recherche a trébuché en cours de route, se heurtant à des problèmes difficiles à résoudre, et les premiers temps des années 70 sont arrivées sans les résultats escomptés. Le manque de résultats a progressivement conduit à l'arrêt du financement et le premier "hiver" de l'IA a commencé.

Au début des années 1980, l'intérêt pour l'IA a commencé à renaître avec la création d'un nouveau type de programmes d'IA, les "systèmes experts" (McCorduck, 2004). Les systèmes experts sont des programmes qui répondent à des questions ou tirent des conclusions utiles sur un domaine très spécifique, en utilisant un ensemble de règles et une base de données contenant des connaissances pertinentes pour le domaine. Les règles et les connaissances sont introduites dans le système par des experts ayant une expérience dans le domaine spécifique. Les systèmes expérimentés ont commencé à être utilisés avec succès par les entreprises du monde entier, ce qui a donné lieu à un marché de plusieurs milliards de dollars provenant de la vente des machines spécialisées qui faisaient fonctionner les systèmes. Les chercheurs de cette décennie se sont concentrés sur les systèmes basés sur la connaissance et sa représentation.

Cet intérêt n'a cependant pas duré longtemps, car les ordinateurs devenaient de plus en plus rapides, dépassant les machines des systèmes expérimentés en termes de puissance de calcul.

Du début des années 1990 jusqu'à aujourd'hui, le domaine de l'IA s'est progressivement redressé et a atteint un bon nombre des objectifs initiaux fixés par les premiers chercheurs. Les résultats des recherches de ces années sont de plus en plus utilisés dans des applications pratiques. L'IA est utilisée dans l'extraction de données, les diagnostics médicaux et d'autres domaines. Ce succès est dû à l'augmentation de la puissance de calcul, à la résolution de problèmes spécifiques, aux nouveaux liens entre l'IA et d'autres domaines (tels que les statistiques, l'économie et les mathématiques) et l'engagement des chercheurs envers les méthodes mathématiques et les normes scientifiques.

Selon Jack Clark de Bloomberg, 2015 a été une année marquante pour l'IA, le nombre de logiciels utilisant l'IA chez Google augmentant rapidement. Nous disposons désormais d'une énorme quantité de données, car l'activité de l'humanité sur l'internet a laissé une énorme empreinte numérique, que nous pouvons exploiter pour former nos algorithmes d'IA. De nombreux chercheurs et scientifiques expriment bien sûr une version plus dystopique de l'IA.

Selon eux, l'IA risque de provoquer de graves troubles sociaux et de constituer une menace directe pour la démocratie et la paix dans le monde. Cependant, quelles que soient les inquiétudes, elles

ne peuvent pas arrêter la marée du progrès technologique : "C'est comme si nous tournions le dos à une vague qui vient tout balayer". Le plus important est de réfléchir à la façon dont nous pouvons orienter cette force de transformation de manière à changer positivement la vie des gens (Askalakis, 2018).

1.4 Apprentissage automatique

L'apprentissage automatique est un sous-domaine de l'informatique qui est né de l'intersection de l'informatique et des statistiques (Mitchell, 2006). En 1959, Arthur Samuel a fait une découverte très importante : au lieu d'apprendre aux ordinateurs ce qu'ils doivent savoir sur le monde et comment effectuer des tâches, il est possible de leur apprendre à se connaître eux-mêmes. Samuel définit l'apprentissage automatique comme "un domaine d'étude qui donne aux ordinateurs la capacité d'apprendre sans être explicitement programmés". Au cours de la dernière décennie, l'apprentissage automatique nous a permis d'obtenir des voitures à conduite autonome, une reconnaissance vocale pratique, une recherche efficace sur le web et une bien meilleure compréhension du génome humain. L'apprentissage automatique est si répandu aujourd'hui qu'il est probablement utilisé par chaque être humain des dizaines de fois par jour sans être immédiatement remarqué. En substance, l'apprentissage automatique est une classe d'algorithmes capables de faire des prédictions et de s'adapter lorsque de nouvelles données sont présentées. Un algorithme d'apprentissage automatique est généralement imprécis au début, mais s'améliore par prédiction sur de nouvelles données et est adapté sur la base d'un processus d'essais et d'erreurs. Cette méthode est capable de résoudre des problèmes que la programmation classique ne peut pas résoudre, mais à un coût plus élevé en termes de ressources informatiques et de temps. En outre, à mesure que les données dont dispose un tel algorithme augmentent, il est en mesure de résoudre des problèmes plus difficiles tels que la reconnaissance des images, des sons et du langage naturel avec une plus grande précision.

Il existe trois grandes catégories d'algorithmes d'apprentissage automatique, en fonction de selon la manière dont ils traitent les données d'entrée et le retour d'information disponible dans le système d'apprentissage (J Russell et Norvig, 1995) :

1. **Apprentissage supervisé** : les algorithmes prennent en entrée des données accompagnées de leurs étiquettes et sont invités à apprendre une règle générale afin de trouver la correspondance (association) entre les données et les étiquettes.
2. **Apprentissage non supervisé** : les algorithmes prennent en entrée des données non étiquetées (c'est-à-dire sans qu'aucune expérience ne soit fournie à l'algorithme d'apprentissage) et on leur demande de trouver leur structure. L'apprentissage non supervisé permet de découvrir des modèles cachés dans les données ou de trouver des caractéristiques, qui sont ensuite utilisées dans le processus d'apprentissage.
3. **Apprentissage par renforcement** : algorithmes qui interagissent avec un environnement dynamique pour atteindre un objectif spécifique. Dans ce cas, aucune information ne permet de savoir si la formation s'est rapprochée ou non de l'objectif. Un exemple de cet apprentissage pourrait être la conduite autonome d'un véhicule.

Dans cet ouvrage, nous allons d'abord appliquer l'apprentissage non supervisé, puis l'apprentissage supervisé.

1.5 Apprentissage profond (deep learning)

Le terme Deep Learning a été introduit dans la communauté de l'apprentissage automatique par Rina Dechter en 1986. Apprentissage profond - L'apprentissage profond est un sous-domaine de l'apprentissage automatique. Dans le cas de l'apprentissage profond, contrairement aux réseaux neuronaux artificiels (ANN) simples, il y a plus de couches empilées les unes sur les autres, ce qui permet d'extraire davantage de caractéristiques de haut niveau.

Les architectures d'apprentissage profond, telles que les réseaux de croyance profonds, les CNN et les RNN, ont été appliquées à des domaines tels que la vision par ordinateur, la reconnaissance vocale, le traitement du langage naturel, le filtrage des réseaux sociaux, la bioinformatique et la conception de médicaments, et ont produit des résultats comparables à ceux des humains, voire meilleurs dans certains cas. Les techniques d'apprentissage profond ont donc amélioré la capacité à classer, reconnaître, détecter et décrire - en un mot, à comprendre. Les systèmes Siri et Cortana d'Apple et de Microsoft, respectivement, sont des exemples de l'utilisation de l'apprentissage profond. De nombreux développements favorisent aujourd'hui l'apprentissage profond, comme l'amélioration des architectures d'apprentissage et des algorithmes. De nouvelles approches d'apprentissage automatique ont également permis d'améliorer la précision des données.

Nous disposons de beaucoup plus de données pour la construction de réseaux neuronaux, avec de nombreuses couches profondes. Un point très important à mentionner est que les gens disposent aujourd'hui d'une puissance de calcul incroyable, qui est essentielle pour construire des algorithmes profonds. L'utilisation des GPU accélère les algorithmes de formation par ordre de grandeur, réduisant les temps d'exécution de plusieurs semaines à quelques jours. Le Deep Learning modifie donc la façon de penser la représentation des problèmes, en passant de la suggestion à l'ordinateur de la manière de résoudre le problème à la formation de l'ordinateur pour former la machine à résoudre elle-même le problème. La promesse de l'apprentissage profond est qu'il peut conduire à des systèmes de prédiction qui généralisent bien, s'adaptent bien, améliorent leurs performances avec davantage de données et sont plus dynamiques que les systèmes de prédiction basés sur des règles strictes. Il n'est plus nécessaire d'installer un modèle, la tâche est entraînée.

1.6 Traitement du langage naturel

Le traitement du langage naturel (NLP) est un domaine de l'intelligence artificielle qui permet aux machines de lire, de comprendre et de déduire le sens des langues humaines. C'est un principe qui se concentre sur l'interaction entre la science des données et le langage humain et qui est mis à l'échelle dans de nombreuses industries.

Aujourd'hui, le traitement du langage naturel évolue grâce à de vastes améliorations de l'accès aux données et à une puissance de calcul accrue, qui permettent aux chercheurs d'obtenir des résultats significatifs dans des domaines tels que les soins de santé, les médias, la finance et les ressources humaines, entre autres.

Le traitement du langage naturel est considéré comme l'un des problèmes les plus difficiles de l'informatique. C'est la nature du langage humain qui rend la NLP difficile. Les règles qui dictent la transmission d'informations à l'aide de langues naturelles ne sont pas faciles à comprendre pour les ordinateurs. Certaines de ces règles peuvent être de haut niveau et abstraites, par exemple lorsqu'on utilise une remarque sarcastique pour transmettre une information.

D'autre part, certaines de ces règles peuvent être de bas niveau, comme, par exemple, l'utilisation du caractère "s" pour déclarer la majorité des objets. Pour comprendre pleinement le langage humain, il faut comprendre à la fois les mots et la manière dont les concepts sont reliés pour délivrer le message souhaité. Ainsi, alors que les humains peuvent facilement maîtriser une langue, c'est l'imprécision et les caractéristiques floues des langues naturelles qui font du NLP un problème difficile à mettre en œuvre pour les machines. En termes simples, donc, le traitement du langage

naturel représente la manipulation automatique du langage humain naturel tel que la parole, le texte et bien que le concept en lui-même soit fascinant, la véritable valeur de cette technologie provient de ses cas d'utilisation.
Le traitement du langage naturel nous aide dans de nombreuses tâches et ses domaines d'application semblent s'étendre chaque jour. En voici quelques exemples :

1. Le **NLP** permet de reconnaître et de prédire les maladies à partir des dossiers médicaux électroniques et des conversations des patients. Cette possibilité est actuellement étudiée dans le cadre d'affections telles que les maladies cardiovasculaires, à la dépression et même à la schizophrénie. Par exemple, Amazon Comprehend Medical est un service qui utilise la NLP pour extraire les pathologies, les médicaments et les traitements des notes des patients, des essais cliniques et d'autres dossiers médicaux électroniques.
2. Les organisations peuvent déterminer ce que les clients disent d'un service ou d'un produit en identifiant et en extrayant des informations de sources telles que les médias sociaux. Cette analyse des sentiments peut fournir de nombreuses informations sur les choix des clients et leurs facteurs de décision.
3. Des entreprises telles que Yahoo, Google, filtrent et trient les courriels à l'aide de la NLP en analysant le texte des courriels qui transitent par leurs serveurs et arrêtent les messages indésirables avant qu'ils n'entrent dans la boîte de réception.
4. Pour identifier les fake news, l'équipe NLP du MIT a mis au point un nouveau système permettant de déterminer si une source est exacte ou politiquement biaisée, en identifiant si une source d'information est digne de confiance ou non.
5. Alexa, d'Amazon, et Siri, d'Apple, sont des exemples d'interfaces vocales intelligentes qui utilisent la NLP pour répondre aux instructions vocales et effectuer toutes sortes de tâches : trouver un magasin comparable, donner les prévisions météorologiques, suggérer le meilleur itinéraire pour se rendre au bureau ou allumer les lumières de la maison.
6. Prédiction de la défaillance d'un nœud à partir de journaux non structurés.

Dans ce travail, nous nous concentrerons sur la prédiction de l'échec des tâches par l'analyse des journaux générés par les ordinateurs. Cependant, le langage naturel des journaux non structurés produits par les systèmes informatiques met en évidence deux problèmes. Premièrement, comme les données n'ont pas de structure ou d'étiquettes, les techniques classiques d'apprentissage

automatique souffrent de limitations en termes de prétraitement de ces données, par exemple la formation de vecteurs de caractéristiques ou de classificateurs n'est pas simple. Deuxièmement, il n'est pas pratique de déduire rapidement des modèles complexes à partir de données à haute dimension, à moins que les données ne soient traitées et alimentées par une représentation d'entrée appropriée.

L'apprentissage profond a récemment fait d'énormes progrès dans ces aspects, notamment en ce qui concerne la compréhension du langage naturel. Cela encourage le besoin d'explorer des techniques d'apprentissage non supervisé évolutives dans le contexte de la prédiction de la défaillance des nœuds. Les chercheurs s'accordent à dire que la prédiction des défaillances est utile même si elle est imparfaite et d'une précision limitée (DOD et al., 2012). Si nous supposons que 50% des défaillances de nœuds sont correctement prédites et que le reste l'est de manière incorrecte (faux positifs) alors nous pouvons éviter la moitié des coûteux points de contrôle et redémarrages qui nécessitent une coordination totale avec des falsifications de processus beaucoup moins coûteuses.

1.7 Objectif de la these

L'objectif de la thèse est d'étudier les travaux soumis au superordinateur mentionné dans la section 1.2 et est divisé en deux points principaux. Tout d'abord, un outil développé par *l'Argonne National Laboratory* en collaboration avec *l'Université de l'Illinois* est utilisé pour trouver les corrélations possibles des logs provenant du HPC. Cet outil s'appelle LogAider. De plus, une visualisation et une analyse des données dont nous disposons concernant les logs se référant aux jobs est effectuée et ensuite les délais sont prédits pour prévenir les échecs des jobs. Les données utilisées dans cette thèse sont générées comme mentionné précédemment par MIRA et sont des données ouvertes et publiques (Argonne). Elle sera analysée plus en détail dans le chapitre suivant. Dans ces systèmes, comme mentionné ci-dessus, les dysfonctionnements du matériel, des logiciels ou des applications provoquent des erreurs. Les erreurs se propagent sous la forme de défauts qui conduisent à des défaillances.

Manifestations de défaillances dans ces systèmes et propose une technique efficace pour prédire les défaillances du système, lorsqu'un nœud cesse de répondre, et donc qu'un travail cesse de répondre. À cette fin, les journaux générés par le superordinateur MIRA sont analysés et des techniques d'apprentissage automatique sont développées pour faciliter la prédiction de la

résilience du HPC. Dans le document, nous utilisons une certaine terminologie que nous ne traduirons surtout pas. Il s'agit de ce qui suit :

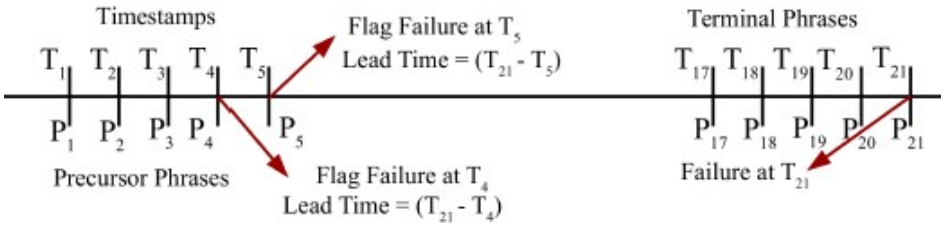


Figure 1.5 : délai d'attente avant une défaillance (Das et al., 2018).

- **Phrase :** Une phrase fait référence à un événement ou à un message de journal dans un fichier journal reçu d'un système informatique. Habituellement, les messages du journal sont affichés avec l'horodatage, mais ce n'est pas nécessairement le cas.
- **Lead Time :** les messages ou les journaux sont enregistrés à un moment donné (horodatage). Le lead time est la différence de temps entre le moment où un composant est devenu insensible et le moment où une défaillance imminente est notée de manière proactive dans une phrase précédente. Cette phrase "précédente" peut ou non indiquer une erreur et peut ou non précéder la terminaison La figure 1.5 ci-dessus illustre la définition du délai d'exécution. Supposons qu'un message terminal indiquant une panne confirmée d'un nœud apparaît dans le message de journal $P21$ au moment $T21$. Si notre structure prédictive de défaillance signale cette défaillance après vérification de la phrase $P5$, le délai calculé est : $\Delta T = (T21 - T5)$. S'il est signalé plus tôt après la vérification de la phrase $P4$, alors le délai est porté à $\Delta T = (T21 - T\ 4)$,l'augmentation est de $(T5\ T4)$.
  Plus le délai est long, plus il reste de temps pour prendre les mesures de rétablissement appropriées. Toutefois, le fait de détecter des défaillances potentielles dans des événements "précurseurs" beaucoup plus précoces peut également entraîner de fausses prédictions (faux négatifs et faux positifs), ce qui réduit la précision. Cela rend bien sûr nécessaire l'étude de la sensibilité aux délais.

- **Défaut** : un défaut est une condition qui fait que le système ne remplit pas la fonction requise.
- **Erreur** : Désigne la différence entre le résultat réel et le résultat attendu.

- **Défaillance** : c'est l'incapacité d'un système ou d'un composant à remplir la fonction requise conformément à ses spécifications. Les défaillances sont la capacité d'un composant à répondre à ses spécifications peuvent être dans le matériel, le logiciel ou l'application. Ces défaillances sont différentes de l'arrêt massif intentionnel des nœuds ou du redémarrage périodique des services associés à la maintenance.
- **Chaîne de défaillance :** c'est une séquence en chaîne de messages qui conduit à une défaillance du système. Une chaîne de défaillance est définie par l'existence d'un message indicateur de défaillance.

  Ce document n'examine pas la cause exacte de l'anomalie qui conduit à la défaillance, mais insiste sur l'analyse des phrases pour identifier et prédire les défaillances imminentes avant qu'elles ne se produisent.

  En particulier, l'objectif du travail est atteint en prédisant les chaînes de défaillance potentielles à l'aide de LSTM. Nous effectuons une analyse des phrases des journaux non étiquetés, qui peuvent ou non appartenir à des chaînes de défaillance. Ces chaînes de défaillance sont identifiables par un message de journal de terminal, qui est vérifié en consultation avec les administrateurs du système. Une approche d'apprentissage machine profond en trois phases est suivie pour la réussite efficace de la tâche (Das et al., 2018).

Conclusion

Dans un premier temps, nous nous entraînons à identifier les chaînes d'événements du journal qui conduisent à des défaillances, puis nous réentraînons les chaînes d'identification d'événements, cette fois avec les informations sur les délais prévus jusqu'à la défaillance. Enfin, nous prédisons les délais pendant le développement du testing/inference où nous prédisons la fin imprévisible d'un travail et sur quel nœud spécifique cette défaillance a eu lieu. La méthodologie est présentée en détail dans la section 3.

Chapitre 2 : Théorie

2.1 Les Données

Comme mentionné dans la section sur l'objectif de la thèse, 1.7, on utilise les logs du supercalculateur Mira, qu'on analyse afin que ces données soient ouvertes et publiques (Argonne) et nous permettent donc de les analyser pour comprendre le comportement des utilisateurs, les performances de la machine, l'apparition d'erreurs et donc d'appliquer diverses techniques qui permettent de mieux utiliser la puissance de calcul des superordinateurs. En particulier, pour la première partie de la visualisation, nous utilisons les journaux de travail. Ensuite, pour développer la méthodologie de prédiction des pannes, nous combinons deux types de journaux système, les journaux RAS et JOB, afin d'exploiter les caractéristiques des travaux en fonction de leurs états d'exécution.

Les journaux RAS (reliability, availability and serviceability) sont parmi les journaux les plus importants du système car ils indiquent sa fiabilité. Chacun des éléments des journaux RAS est représenté comme un événement spécifique avec l'un des trois niveaux de gravité (INFO, WARN, FATAL). Cependant, dans ce travail, ces niveaux de gravité ne sont pas pris en compte. En effet, ils ne constituent pas une catégorisation efficace des messages dans les données à long terme. Certains messages apparemment "sûrs" peuvent conduire à des échecs. Sur les 17 champs, seuls quelques-uns sont critiques pour notre étude, tels que MESSAGE ID, EVENT TIME, BLOCK, LOCATION, MESSAGE. Les journaux RAS sont les plus importants.

Les journaux importants liés à la fiabilité du système, tels que la défaillance d'un nœud et les pannes de courant.

L'infrastructure de fiabilité, de disponibilité et d'aptitude au service (RAS) fournit donc les moyens de signaler les événements matériels et logiciels pour le système Blue Gene/Q. Un ensemble prédéfini de messages est intégré, mais la conception permet également aux utilisateurs de définir leurs propres journaux RAS qui sont enregistrés dans la base de données. Ces journaux sont la principale source d'informations qu'un administrateur système peut utiliser pour comprendre les défaillances.

Dans MIRA, les utilisateurs qui souhaitent exécuter une application ou une simulation avec une grande efficacité de calcul doivent soumettre une tâche au planificateur de tâches du système. Les journaux JOB contiennent des informations sur les travaux qui ont été soumis. Parmi les 57 champs des journaux qui décrivent l'état de chaque tâche, tels que l'état de la file d'attente, l'état de l'exécution, le nombre de nœuds ou de cœurs utilisés, le temps nécessaire pour terminer les tâches et l'état de sortie, nous en utilisons quelques-uns (START TIMESTAMP, END TIMESTAMP, QUEUE NAME, WALLTIME SECONDS, etc.)

Pourquoi est-il important d'anticiper les défaillances potentielles dans les systèmes à haute performance ?

Des systèmes plus fiables permettent des calculs scientifiques plus productifs et donc des progrès scientifiques plus rapides. Cependant, l'amélioration et le maintien d'un niveau élevé de fiabilité des systèmes sont difficiles pour de nombreuses raisons. Tout d'abord, le nombre de composants augmente rapidement dans les systèmes HPC à grande échelle afin de répondre aux exigences des scientifiques en matière de puissance de calcul, et la probabilité de défaillance augmente donc. Deuxièmement, en raison du rétrécissement des taille des processeurs, il devient plus facile de faire des erreurs liées au changement de traitement et aux défauts de fabrication (Cappello et al., 2014).

Troisièmement, la complexité des systèmes augmente, ce qui rend plus difficile la gestion de la fiabilité des systèmes (Lucas et al., 2014).

Avec l'étude réalisée dans ce document, nous examinons comment les événements fatals du système affectent l'exécution des tâches du côté de la planification des tâches du système. Contrairement à des travaux tels que (Di et al., 2018), qui se concentrent uniquement sur les corrélations entre les événements du système (par exemple, RAS), dans ce travail, nous suivons une méthodologie qui corrèle d'abord les événements RAS du système avec les exécutions de tâches. Cette corrélation est très importante pour les administrateurs système, les utilisateurs d'applications et les chercheurs car elle indique la fiabilité du système du point de vue des emplois.

Bien sûr, l'étude des journaux n'est pas facile car ils sont trop nombreux et la corrélation entre les journaux RAS et JOB seulement prend du temps et nécessite une grande consommation de

ressources du travail ont également été exigeantes, les messages du journal devant tout d'abord être mis dans un format adapté à l'analyse des données. Et ensuite les deux phases de la formation.

L'ensemble de ce processus a été assez exigeant en termes de ressources.

Enfin, la compréhension des échecs de travail sur ce superordinateur est d'une importance plus large car de nombreux superordinateurs tels que Sequoia (USA), Vulcan (USA) et Blue Joule (UK) adoptent également la même architecture que le système Blue Gene/Q et sont toujours en fonctionnement.

2.2 Word-Embedding: (integration de mots)

Travailler avec des données textuelles non structurées est assez difficile, surtout lorsque l'objectif est de créer un système intelligent qui interprète et comprend le langage naturel libre comme le font les humains. Il serait nécessaire de traiter et de transformer les données textuelles non structurées en données structurées et vectorielles pouvant être comprises par tout algorithme d'apprentissage automatique. Chaque algorithme d'apprentissage automatique est basé sur des principes statistiques, mathématiques et d'optimisation. Par conséquent, ils ne sont pas assez intelligents pour commencer à traiter le texte dans sa forme originale et naturelle.

Les emboîtements de mots sont l'une des représentations du vocabulaire textuel les plus connues dans le traitement du langage naturel, où les mots ou les phrases du vocabulaire sont représentés dans des vecteurs de nombres réels. D'un point de vue conceptuel, il s'agit d'une intégration mathématique d'un espace à plusieurs dimensions par mot à un espace vectoriel continu de dimension bien inférieure. Ils sont capables de capturer le contenu d'un mot dans un document, la similarité sémantique et syntaxique, la relation avec d'autres mots, etc. Il a été démontré que les incorporations de mots et de phrases, lorsqu'elles sont utilisées comme représentation d'entrée sous-jacente, améliorent les performances dans les tâches de traitement du langage naturel telles que l'analyse syntaxique et l'analyse des sentiments.

Pourquoi en avons-nous besoin ? Supposons que nous ayons les propositions similaires suivantes : **Have a good day** et **Have a great day**, n'ont guère de sens différent. Si nous construisons un vocabulaire exhaustif, disons V, alors ce serait V = Have, a, good, great, day. Nous créons ensuite un vecteur codé à un coup pour chacun de ces mots dans V. La longueur de notre vecteur codé à un coup sera égale à la taille de V (=5). Nous aurons un vecteur b de zéros, sauf pour l'élément de

l'index représentant le mot correspondant dans le vocabulaire. Cet élément particulier sera 1. Les codages suivants expliquent mieux cela.

$$Have = [1,0,0,0,0]‘;\ a = [0,1,0,0,0]‘;\ good = [0,0,1,0,0]‘;\ great = [0,0,0,1,0]‘;$$

$$day = [0,0,0,0,1]‘\ (‘\ represents\ transpose).$$

Si nous essayons de visualiser ces encodages, nous pouvons penser à un espace à 5 dimensions, où chaque mot occupe une des dimensions et n'a aucune relation avec les autres (aucune projection dans les autres dimensions). Cela signifie que le bon et le grand sont aussi différents que le jour et la nuit, ce qui n'est pas vrai. Notre objectif est de posséder des mots avec un contenu similaire dans des emplacements spatiaux proches.

2.3 Word2Vec Model

Le modèle Word2Vec a été créé en 2013 par Google (Mikolov et al., 2013b) et est un modèle prédictif basé sur l'apprentissage profond pour calculer et générer des représentations vectorielles denses de haute qualité, distribuées et continues des mots, qui capturent la similarité contextuelle et sémantique. Il s'agit essentiellement de modèles non supervisés, qui peuvent prendre en charge d'énormes textes, créer un vocabulaire de mots possibles et générer des encastrements de mots denses pour chaque mot dans l'espace vectoriel représentant ce vocabulaire. Habituellement, la taille des vecteurs d'intégration des mots est spécifiée et le nombre total de vecteurs correspond essentiellement à la taille du vocabulaire. La dimensionnalité de cet espace vectoriel dense est donc beaucoup plus faible que celle de l'espace vectoriel clairsemé de haute dimension que l'on retrouve dans l'espace vectoriel dense. Construits à l'aide de modèles traditionnels de type "sac de mots". Deux architectures de modèles différentes peuvent être exploitées par Word2Vec pour créer ces représentations d'incorporations de mots. Il s'agit de (Mikolov et al., 2013a) :

- **Continuous Bag of Words (CBOW) Model**

L'architecture du modèle CBOW tente de prédire le mot cible actuel (le mot central) sur la base des mots du contexte source (mots environnants). Si l'on considère une phrase simple, "le renard brun est rapide saute par-dessus le chien paresseux", cela peut être des paires de (fenêtre_contexte, mot_cible) ou si l'on considère une fenêtre contextuelle de taille 2, nous avons

des exemples comme ([renard, rapide], brun), ([le brun], rapide), ([le chien], paresseux) et ainsi de suite. Ainsi, le modèle tente de prédire le mot cible sur la base des mots de la fenêtre contextuelle.

- **Skip-gram Model**

Dans cet ouvrage, nous comptons utiliser le modèle Skip-gram qui est présenté dans la section 3.1.

2.4 Recurrent Neural Networks

Les gens ne commencent pas à penser à partir de zéro à chaque seconde. En lisant un livre, par exemple, on peut comprendre chaque mot en fonction de la compréhension des mots précédents. Nos pensées sont persistantes.

Cependant, les réseaux neuronaux traditionnels ne peuvent pas retenir les informations, ce qui constitue un inconvénient très important. Par exemple, imaginez que vous souhaitiez classer le type d'événement qui se produit à chaque moment d'un film. On ne voit pas comment un réseau neuronal traditionnel pourrait utiliser son raisonnement sur les événements antérieurs du film pour informer les plus récents.

Les réseaux neuronaux récurrents (RNN) permettent de résoudre ce problème. Il s'agit de réseaux comportant des boucles, ce qui permet de retenir l'information.

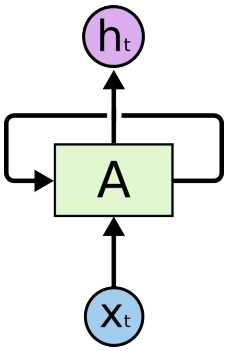


Figure 2.1: Recurrent Neural Networks have loops. (Olah, 2015).

Dans le schéma ci-dessus, un segment de réseau neuronal, A, examine une entrée xt et en extrait une valeur ht. Une boucle permet de transmettre des informations d'une étape du réseau à la suivante. Ces boucles font que les réseaux neuronaux récurrents semblent assez mystérieux, mais

si l'on y réfléchit un instant, on s'aperçoit qu'ils ne sont pas très différents d'un réseau neuronal normal. Un réseau neuronal récurrent peut être considéré comme de multiples copies du réseau lui-même, chacune transmettant un message à son successeur. L'image ci-dessous montre le dépliage d'une boucle :

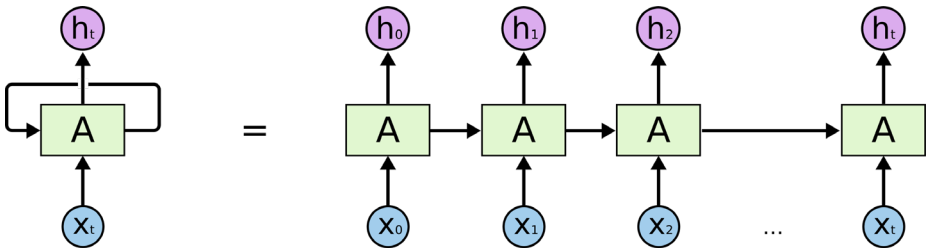


Figure 2.2: An unrolled recurrent neural network. (Olah, 2015).

Cette forme en chaîne révèle que les réseaux neuronaux récurrents sont étroitement liés aux séquences et aux listes. C'est l'architecture physique du réseau neuronal qui utilisera ces données.

Ces dernières années, l'application des réseaux neuronaux récurrents à divers problèmes a connu un succès incroyable : reconnaissance vocale, modélisation du langage, traduction, sous-titrage d'images... la liste est longue.

Ces succès sont dus à l'utilisation de LSTM, un type très particulier de réseau neuronal récurrent qui fonctionne, pour de nombreuses tâches, bien mieux que la version ordinaire. Presque tous les résultats passionnants basés sur les réseaux neuronaux récurrents sont obtenus avec les LSTM.

The Problem of Long-Term Dependencies

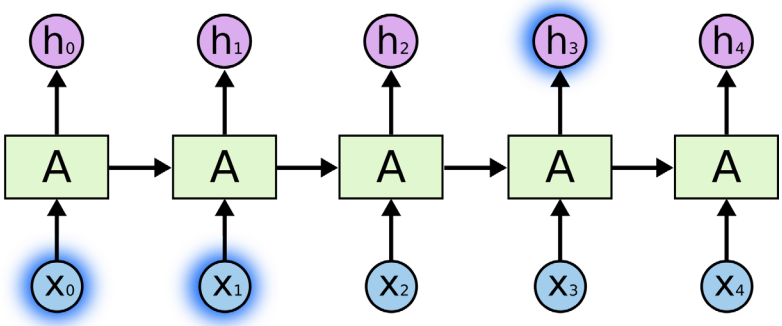
L'une des approches des RNN est l'idée qu'ils peuvent relier des informations passées à la tâche présente, tout comme l'utilisation d'images vidéo passées peut aider à comprendre des images vidéo présentes. Donc si les RNNs pouvaient faire cela, ce serait extrêmement utile. Mais le peuvent-ils ? Cela dépend.

Parfois, il suffit d'examiner les dernières informations pour faire le travail. Par exemple, supposons qu'un modèle de langage essaie de prédire le mot suivant sur la base des mots précédents. Si nous

essayons de prédire le dernier mot de "les nuages sont dans le ciel", nous n'avons pas besoin d'autre contenu - il est assez évident que le prochain mot sera "ciel".

Dans ce cas, lorsque l'écart entre les informations pertinentes et l'espace nécessaire est faible, les RNN peuvent apprendre à utiliser des informations antérieures.

Figure 2.3 : l'écart entre les informations (Olah, 2015).



Mais il y a aussi des cas où nous avons besoin de plus de contexte. Supposons que nous voulions prédire le dernier mot du texte “I grew up in France. . . I speak fluent French”. Des informations récentes suggèrent que le prochain mot est probablement le nom d'une langue, mais si nous voulons préciser de quelle langue il s'agit, nous avons besoin du contexte du français, qui est loin derrière. Il est donc très probable que l'écart entre les informations pertinentes et les endroits où elles sont nécessaires soit très important.

Malheureusement, au fur et à mesure que ce fossé se creuse, il devient de plus en plus difficile pour que les RNN apprennent à connecter les informations.

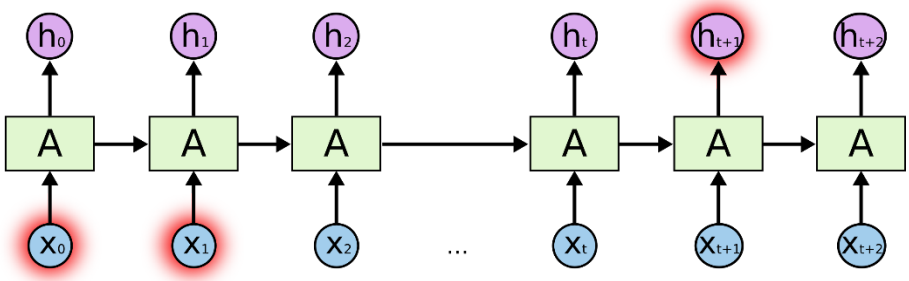


Figure 2.4 : Probleme des dépendances à long terme (Olah, 2015).

En théorie, les RNN sont parfaitement capables de gérer ces "dépendances à long terme". Un humain pourrait choisir soigneusement leurs paramètres pour résoudre des problèmes de cette

nature. Malheureusement, en pratique, les RNN ne semblent pas être capables de les apprendre. Le problème a été étudié en profondeur par Hochreiter (Hochreiter, 1991) et Bengio, et al. (Bengio et al., 1994), qui ont trouvé quelques raisons fondamentales pour lesquelles cela peut être difficile.

Heureusement, les LSTM n'ont pas ce problème !

2.5 LSTM Networks

Les réseaux à mémoire à long terme – généralement LSTM – sont un type particulier de RNN, capable d'apprendre des dépendances à long terme. Ils ont été introduits par Hochreiter & Schmidhuber (1997) (Hochreiter et Schmidhuber, 1997) et ont été améliorés et utilisés par de nombreuses personnes dans des travaux ultérieurs.

Les LSTMs sont spécifiquement conçus pour éviter le problème de la dépendance à long terme. Se souvenir d'une information pendant de longues périodes est pratiquement leur comportement par défaut, et non quelque chose qu'ils ont du mal à apprendre !

Tous les réseaux neuronaux récurrents se présentent sous la forme d'une chaîne de modules de réseaux neuronaux récurrents. Dans les RNN standard, ce module récurrent aura une structure très simple, telle qu'une couche tanh uniquement.

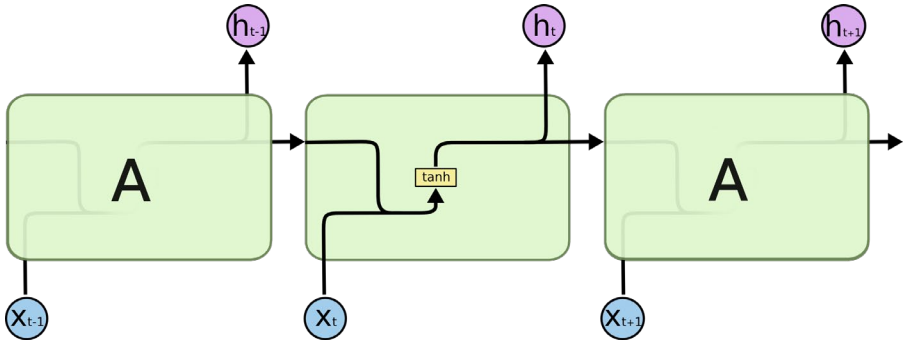


Figure 2.5: Le module de répétition d'un RNN standard contient une seule couche. (Olah, 2015).

Les LSTM ont également cette structure en chaîne, mais le module répétiteur a une structure différente. Au lieu d'avoir une seule couche de réseau neuronal, il y en a quatre, qui interagissent d'une manière très particulière.

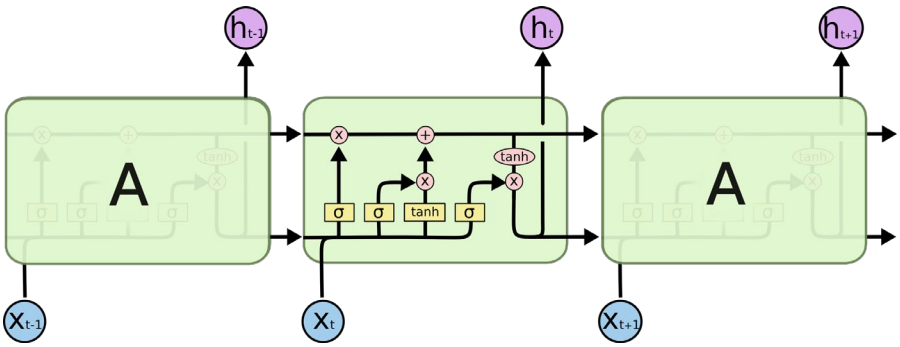


Figure 2.6: The repeating module in a LSTM contains four interacting layers. (Olah, 2015).

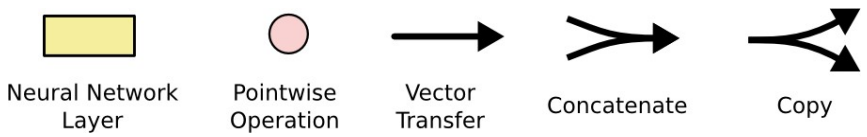


Figure 2.7 : Legend(Olah, 2015).

Dans le diagramme ci-dessus, chaque ligne porte un vecteur entier, de la sortie d'un nœud aux entrées des autres nœuds. Les cercles roses représentent des opérations ponctuelles, comme l'addition de vecteurs, tandis que les cases jaunes sont des couches de réseau neuronal formées. Les lignes de fusion indiquent une fusion, tandis qu'une ligne de branchement indique que son contenu est copié et que ses copies sont transférées à différents endroits.

2.5.1 L'idée de base des LSTMs

La clé des LSTM est l'état de la cellule, la ligne horizontale qui traverse le haut du diagramme. L'état des cellules est un peu comme un tapis roulant. Il traverse toute la chaîne, avec seulement quelques petites interactions linéaires. Il est très facile de transférer des informations sur une longueur inchangée.

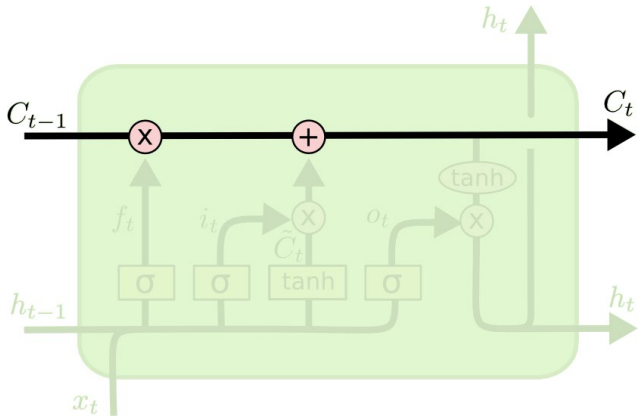


Figure 2.8 : l'état de la cellule(Olah, 2015).

Le LSTM a la capacité de supprimer ou d'ajouter des informations à l'état de la cellule, soigneusement configurées par des structures appelées portes.

Les portails sont un moyen d'éliminer l'information à travers eux. Ils se composent d'une couche de réseau neuronal sigmoïde et d'une fonction de multiplication de points.

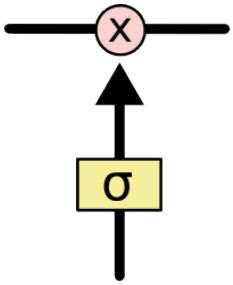


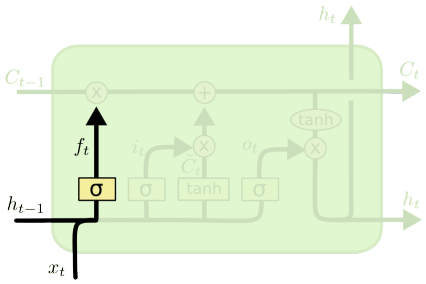
Figure 2.9 : contrôle des états de cellules(Olah, 2015).

Les sorties de la couche sigmoïde produisent des nombres entre 0 et 1, décrivant la quantité de chaque composant à faire passer. Une valeur de 0 signifie « ne rien laisser passer », tandis qu'une valeur de 1 signifie « tout laisser passer ». Un LSTM possède trois de ces portes pour protéger et contrôler les états des cellules

2.5.2 LSTMs pas à pas

La première étape dans les LSTM consiste à décider quelle information rejeter des états des cellules. Cette décision sera prise par une couche sigmoïde appelée « couche de porte d'oubli ». Il examine $h_t - 1$ et xt et obtient en sortie un nombre compris entre 0 et 1 pour chaque nombre dans l'état de cellule $C_t - 1$ .1 représente « garder cette information », tandis que 0 représente « jeter cette information ».

Prenons donc l'exemple du modèle de langage qui tente de prédire le mot suivant sur la base de tous les mots précédents. Dans un tel problème, l'état de la cellule peut inclure le sexe du sujet actuel, de sorte que les pronoms corrects peuvent être utilisés. Lorsque nous voyons un nouveau sujet, nous voulons oublier le genre du sujet précédent.

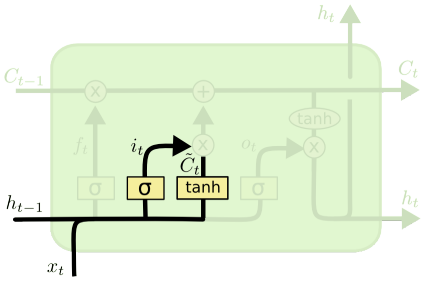


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.10 : couche de porte d'oubli(Olah, 2015).

L'étape suivante consiste à décider des nouvelles informations à stocker dans l'état de la cellule, ce qui comporte deux parties. Tout d'abord, une couche sigmoïde appelée « couche porte d'entrée » décide des valeurs à mettre à jour. Sur alors une couche de tanh crée un vecteur de valeurs possibles, $\widetilde{C_t}$, qui sera dans l'étape suivante, nous allons combiner les deux pour créer une mise à jour de l'état.

Ainsi, pour l'exemple du modèle de langue, nous aimerions ajouter le genre du nouveau sujet à l'état de la cellule, pour remplacer l'ancien que nous avons oublié.

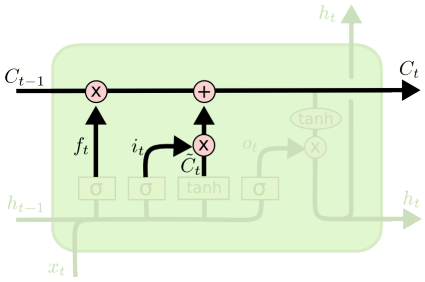


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 2.11 : couche porte d'entrée(Olah, 2015).

Il est maintenant temps de mettre à jour l'ancien état de la cellule, $C_t - 1$, vers le nouvel état de la cellule c_t. Les étapes précédentes ont déjà décidé de ce qu'il faut faire, il ne reste plus qu'à le faire. Nous multiplions l'ancien état par ft, en laissant les choses que nous avons décidé de laisser plus tôt. Puis on l'ajoute au $\tilde{c}_t$. Il s'agit des nouvelles valeurs des candidats, mises à l'échelle en fonction du nombre de mises à jour que nous avons décidé de faire pour chaque état.

Dans l'exemple du modèle linguistique, c'est ici que nous avons laissé les informations sur le sexe de l'ancien sujet et ajouté de nouvelles informations, comme nous l'avons décidé dans les étapes précédentes.

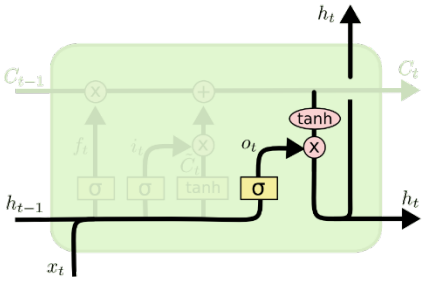


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.12 :mise a jour de l'etat de cellule (Olah, 2015).

En fin de compte, nous devons décider de ce que nous avons comme sortie. Cette sortie sera basée sur l'état de la cellule, mais il s'agira d'une version filtrée. Tout d'abord, nous avons une couche sigmoïde qui décide des parties de l'état de la cellule à sortir. Ensuite, nous faisons passer l'état de la cellule par tanh (pour pousser les valeurs entre -1 et 1) et le multiplions par la sortie de la porte sigmoïde pour extraire les parties que nous avons décidées.

Donc pour l'exemple du modèle de langage, étant donné qu'il a vu un sujet, il est possible de vouloir produire des informations sur un verbe, au cas où il viendrait ensuite. Par exemple, il peut vouloir extraire si un sujet est singulier ou pluriel, afin de savoir quelle forme le verbe doit prendre s'il le suit.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.13 : couche porte de sortie(Olah, 2015).

Les LSTMs sont un grand pas en avant dans ce que nous pouvons réaliser avec les RNNs.

2.5.3 Couches supplémentaires et fonctions d'activation

Dans cette section, nous allons d'abord mentionner les couches utilisées pour l'entraînement avec LSTM, puis les fonctions d'activation que nous avons utilisées dans ce travail.

Comme mentionné dans la section précédente, les modèles LSTM sont un type de réseaux neuronaux récurrents qui ont la capacité d'apprendre des séquences d'observations. Cela en fait un réseau adapté à la prédiction des séries temporelles. L'un des problèmes qui se posent avec les LSTM est qu'ils peuvent facilement s'adapter aux données d'apprentissage, ce qui réduit leur capacité de prédiction.

Le ***Dropout*** est une méthode de régularisation qui permet de réduire de manière significative l'**overfitting**[1] en ignorant des neurones sélectionnés de manière aléatoire pendant la formation et réduit ainsi la « sensibilité » aux poids spécifiques des neurones individuels.

Dans ce cas, au lieu d'utiliser des couches d'exclusion individuelles, nous utilisons l'exclusion au sein des cellules LSTM. Si des couches d'exclusion distinctes avaient été utilisées, seule la sortie finale du LSTM aurait été affectée, alors que dans notre cas, nous appliquons l'exclusion à 4 fonctions différentes du réseau neuronal secondaire au sein du LSTM.

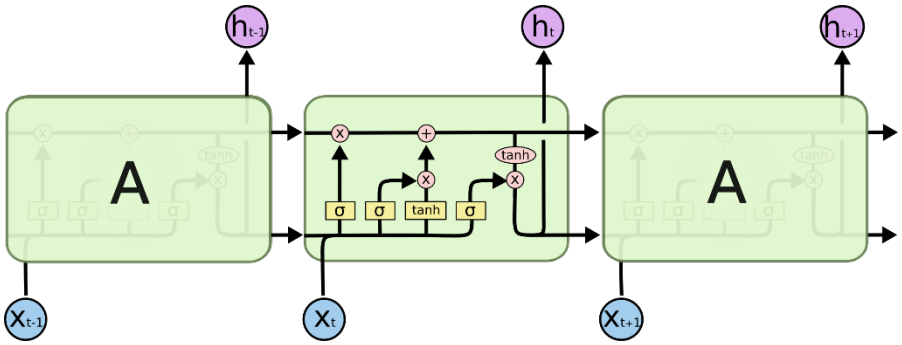


Figure 2.14 :Le module répétitif d'un LSTM contient quatre couches en interaction (Olah 2015)

[1] La suradaptation (**overfitting**) désigne un modèle qui donne de très bons résultats sur les données d'apprentissage mais qui ne se généralise pas bien. La suradaptation se produit lorsqu'un modèle apprend à modéliser les détails et le bruit des données d'apprentissage au point de nuire aux performances du modèle sur les nouvelles données. Cela signifie que le bruit ou les variations aléatoires des données de formation sont collectés et formés comme des concepts par le modèle. Le problème est que ces concepts ne sont pas valables pour les nouvelles données et affectent négativement la capacité des modèles à généraliser. La suradaptation est plus probable avec les modèles non paramétriques et non linéaires qui ont plus de flexibilité et de liberté pour construire le modèle sur la base de l'ensemble de données et peuvent donc effectivement construire des modèles irréalistes. Par conséquent, de nombreux algorithmes d'ingénierie non paramétriques
L'apprentissage comprend également des paramètres ou des techniques permettant de limiter la quantité de détails que le modèle apprend.

Dans l'image ci-dessus, les cases jaunes montrent les 4 fonctions de réseau entièrement connectées (chacune avec son propre poids) qui se produisent sous le LSTM.

En particulier, dans les LSTM, nous avons deux types de paramètres de dropout, le dropout qui s'applique aux n-inputs (qui sont convertis à une dimension compatible) et le dropout récurrent, qui s'applique aux entrées récurrentes (sorties et/ou états précédents). Ainsi, les entrées et les connexions récurrentes aux unités LSTM sont vraisemblablement exclues des activations et des mises à jour des poids pendant la formation du réseau. Ceci, comme mentionné ci-dessus, a pour effet d'améliorer la formation.

Une fois que les couches LSTM ont effectué tout le travail de transformation de l'entrée pour permettre la prédiction de la sortie souhaitée, nous devons réduire (ou dans de rares cas étendre) le schéma pour qu'il corresponde à la sortie souhaitée. Dans notre cas, puisque l'analyse des logs est un problème multi-classes, nous aurons autant de sorties que le nombre total de vocabulaire qui composent les logs. Ce processus est effectué dans la couche dense. Selon la description de Keras, cette couche est l'application de l'équation output = activation(dot(input,kernel)+bias). Cela signifie que nous prenons le produit entre le tenseur d'entrée et ce que la matrice du noyau de poids affiche dans la couche dense. Nous pouvons alors ajouter le vecteur de biais (si nous le souhaitons) et obtenir une activation par éléments des valeurs de sortie.

Le paramètre le plus important pris par cette couche est le paramètre des unités, un nombre positif indiquant la taille de sa sortie. Ce paramètre détermine essentiellement la taille de la matrice de poids et du vecteur de polarisation (le vecteur de polarisation aura la même taille), mais la matrice de poids sera calculée en fonction de la taille des données d'entrée, de sorte que le produit produira des données avec une taille de sortie des unités.

Le prochain paramètre que nous allons mentionner est celui des activations. Il peut être utilisé soit par une couche d'activation, soit par un paramètre d'activation. Dans notre cas, il serait inclus dans la couche dense. Si une fonction d'activation n'est pas appliquée, le signal de sortie ne serait qu'une simple fonction linéaire, c'est-à-dire un polynôme d'un degré. Une équation linéaire est facile à résoudre, mais elle est limitée dans sa complexité et a moins de pouvoir pour apprendre des mappings fonctionnels complexes à partir des données. Une couche de réseau neuronal sans fonction d'activation serait simplement un modèle de régression linéaire, dont la puissance est

limitée et qui ne donne pas de bons résultats la plupart du temps. De plus, ce qui est nécessaire, c'est le réseau neuronal et non pas le non seulement pour apprendre et calculer une fonction linéaire, mais quelque chose de plus complexe que ça.

Une équation linéaire est facile à résoudre, mais elle est limitée dans sa complexité et a moins de pouvoir pour apprendre les mappings fonctionnels complexes à partir des données. Une couche de réseau neuronal sans fonction d'activation serait simplement un modèle de régression linéaire (Linear modèle of régression), qui a une puissance limitée et n'est pas performant la plupart du temps.

De plus, ce qui est nécessaire, c'est le réseau neuronal pas seulement pour apprendre et calculer une fonction linéaire, mais quelque chose de plus complexe que ça. De même, sans la fonction d'activation, notre réseau neuronal ne sera pas en mesure d'apprendre et de modéliser d'autres types de données complexes, tels que les images, la vidéo, l'audio, la parole, etc. C'est pourquoi les techniques de réseaux neuronaux artificiels telles que l'apprentissage profond sont utilisées pour comprendre des ensembles de données plus complexes, de haute dimension et non linéaires, où le modèle comporte de nombreuses couches cachées entre elles et possède une architecture très complexe qui aide à comprendre et à extraire des connaissances de ces grands ensembles de données complexes.

Alors pourquoi les µ.h-grams sont-ils nécessaires ? Le non-linéaire

Les fonctions non linéaires sont celles qui ont un degré supérieur à un et qui présentent une courbure lorsqu'une fonction non linéaire est tracée. Les réseaux neuronaux sont considérés comme des approximateurs universels de fonctions (Universal Function Approximators). Cela signifie qu'ils peuvent calculer et apprendre n'importe quelle fonction. Presque tous les processus auxquels on peut penser peut-être représenter comme un calcul fonctionnel dans les réseaux neuronaux. Pour cette raison, une fonction d'activation f(x) doit être appliquée pour rendre le réseau plus puissant et ajouter la capacité d'apprendre quelque chose de complexe à partir des données et de représenter des mappings fonctionnels arbitraires complexes non linéaires entre les entrées et les sorties.

Par conséquent, en utilisant l'activation non linéaire, il est possible de produire des mappings non linéaires des entrées aux sorties.

La fonction d'activation à utiliser dépend du problème à résoudre. Pour l'apprentissage du modèle Skipgram, la fonction d'activation sigmoid est utilisée, tandis que pour l'analyse des logs dans la première phase, qui comme mentionné ci-dessus est un problème multi-classes, la fonction d'activation softmax est utilisée. Toujours pour la deuxième phase, qui est l'entraînement sur des séries temporelles multivariées, la fonction d'activation ELU est utilisée. Ces fonctions seront décrites dans la prochaine sous-section 2.5.4.

Chaque échantillon peut appartenir à l'une des classes C. Le LSTM aura c neurones à sa sortie. H (ground-truth) sera un vecteur à un coup avec une classe positive et c 1 classes négatives.

2.5.4 Softmax-Sigmoid-ELU

- **Softmax**

La dernière couche du réseau neuronal, sans la fonction d'activation, est ce que nous appelons " logits layer ". Il fournit simplement les sorties finales du réseau neuronal. Dans le cas d'une classification multi-classes, avec n classes, il y aura n-neurones et donc n sorties. Bien entendu, ces nombres de "logits" ne nous fournissent aucune information utile.

D'une certaine manière, la prédiction de la classe cible (Target class) à laquelle une entrée appartient est liée à une distribution de probabilité. Pour cela, si nous connaissons les probabilités qu'une valeur qui appartient à l'une des n issues possibles, nous pourrions simplement prendre l'argmax de ces probabilités discrètes et trouver l'issue de la classe. Il suffit donc de convertir les

logits ci-dessus en une distribution de probabilité. La distribution de probabilité appropriée est la distribution de probabilité discrète (discrete probability distribution), qui peut prendre un nombre mesurable de valeurs, c'est-à-dire une probabilité pour chaque valeur.

La fonction Softmax nous permet d'exprimer nos entrées comme une distribution de probabilité discrète. Mathématiquement, cela se définit comme suit :

$$softmax(xi) = \frac{\exp\,(xi)}{\sum_{j=1} \exp\,(xj)}$$

D'un point de vue esthétique, cela peut se traduire comme suit :

Pour chaque valeur (par exemple, l'entrée) dans notre vecteur d'entrée, la valeur Softmax est l'exposant de l'entrée modulée divisé par une somme des exposants de toutes les entrées. Cela garantit que de nombreuses choses se produisent :

- Les entrées négatives seront converties en valeurs non négatives, grâce à la fonction exponentielle.

- Chaque entrée sera dans l'intervalle (0,1).

- Comme le dénominateur de chaque calcul Softmax est le même, les valeurs deviennent proportionnelles les unes aux autres, ce qui garantit que leur somme s'élève à ensemble à 1.

En résumé, la fonction Softmax convertit les logits en probabilités dont la somme est égale à 1. Les fonctions Softmax produisent un vecteur représentant les distributions de probabilité d'une liste de résultats possibles.

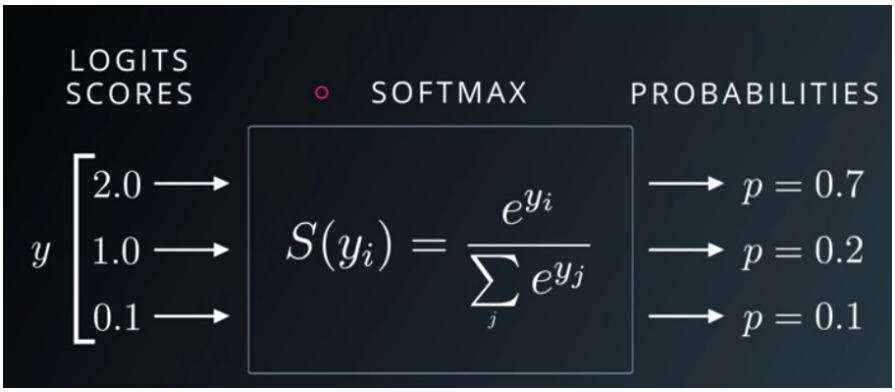


Figure 2.15: Diapositive d'Udacity Deep Learning sur Softmax.

La figure ci-dessus nous montre que la fonction Softmax convertit les logits [2.0, 1.0, 0.1] en probabilités [0.7, 0.2, 0.1] et que la somme des probabilités donne 1.

- **Sigmoïde**

La fonction d'activation Sigmoïde a l'équation mathématique suivante est représentée ci-dessous.

$$A = \frac{1}{1 + e^{-x}}$$

Figure 2.16 : Fonction sigmoïde (Sharma, 2017).

La principale raison pour laquelle nous utilisons la fonction sigmoïde est qu'elle existe entre (0 à 1). Par conséquent, il est utilisé spécifiquement pour les modèles où nous devons prédire la probabilité en tant que sortie. Puisque la probabilité d'une chose n'existe qu'entre 0 et 1, la sigmoïde est le bon choix.

Il y a un (smooth gradient) qui empêche les sauts dans les valeurs de sortie. Les valeurs de sortie sont verrouillées comme mentionné entre 0 et 1, normalisant ainsi la sortie de chaque neurone. Les prédictions avec cette fonction sont donc claires - pour un X supérieur à 2 par exemple, ou inférieur à -2, elle tend à amener la valeur Y (la prédiction) au bord de la courbe, très proche de 1 ou de 0.

Bien sûr, un inconvénient majeur est l'apparition du vanishing gradient

- pour des valeurs très élevées ou très faibles de X, il n'y a pratiquement aucun changement dans la prédiction, ce qui entraîne un problème de vanishing gradient. Cela peut conduire le réseau à refuser d'apprendre davantage ou à être trop lent pour réaliser une prédiction précise. Il s'agit également d'une option coûteuse en termes de calcul.

Exponential Linear Unit –E L U

Cette fonction d'activation corrige certains des problèmes de ReLU et préserve certains des aspects positifs. Pour cette fonction d'activation, une valeur alpha a est prise, une valeur courante se situant entre 0,1 et 0,3.

$$ELU\ (x) = \begin{cases} x & if\ x > 0 \\ \alpha(ex\ -\ 1) & if\ x < 0 \end{cases}$$

Si nous entrons une valeur de x supérieure à 0, c'est la même chose que ReLU.

- le résultat sera une valeur y égale à la valeur x. Mais avec l'ELU, si la valeur d'entrée x est inférieure à 0, nous obtenons une valeur légèrement inférieure à zéro.

La valeur y obtenue dépend à la fois de la valeur d'entrée x et du paramètre alpha a, qui peut être ajusté selon les besoins.

À chaque fois. En outre, nous introduisons une fonction exponentielle Ex, ce qui signifie que l'ELU est plus précise sur le plan informatique que la ReLU.

L'ELU est représentée ci-dessous avec a=0,2.

Le nombre de poids ou de biais reçoit essentiellement de très petites mises à jour. Les nœuds du réseau sont alors éloignés de leur valeur optimale, ce qui empêche finalement le réseau neuronal d'apprendre. Il a été observé que le problème s'aggrave encore s'il y a différentes couches qui apprennent à des vitesses différentes. Les couches apprendront à des vitesses différentes et la première sera toujours la plus mauvaise en termes de taux d'apprentissage.

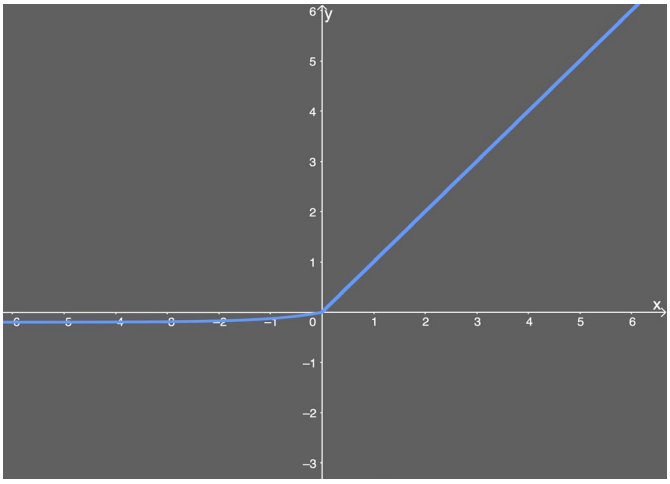


Figure 2.17 : Le diagramme de la fonction d'activation ELU (HANSEN).

Avant de parler de la dérivée, permettez-nous de noter qu'elle est très importante car, au fur et à mesure que la courbe est mise à jour, savoir dans quelle direction et de combien il faut changer ou mettre à jour la courbe dépend de la pente. C'est pourquoi il est si important. Son équation est la suivante :

$$\boldsymbol{ELU(x)} = \begin{cases} 1 & if\ x > 0 \\ ELU(x) + \alpha & if\ x \leq 0 \end{cases}$$

La valeur y est égale à 1 si x est supérieur à 0. Si à nouveau la valeur de x est inférieure à zéro, alors la sortie est la fonction ELU, plus la valeur a. Le graphique correspondant est le suivant :

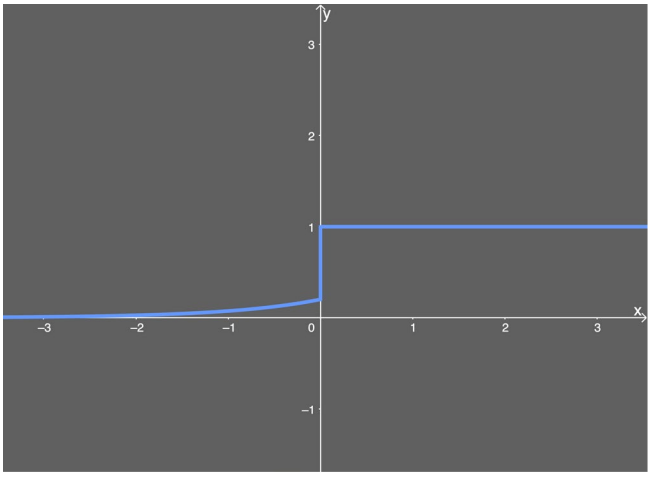


Figure 2.18 : Le diagramme de la dérivée de la fonction d'activation ELU (HANSEN).

Comme vous l'avez peut-être remarqué, nous évitons le problème de la " dead ReLU " tout en conservant une partie de la vitesse de calcul obtenue grâce à la fonction d'activation ReLU.

- **Avantages**

- Evite le problème du Dead ReLU.

- Produit des sorties négatives, qui aident le réseau à pousser les poids. (weights) et (biases) dans les bonnes directions.

- Lors du calcul du gradient, il génère des déclencheurs au lieu de les laisser à zéro.

- **Inconvénients**

- Introduit un temps de calcul plus long en raison de la fonction exponentielle incluse.

- N'évite pas le problème de exploding gradient descent[2].

- Le réseau neuronal n'apprend pas la valeur de a.

2.6 Classes de déséquilibre (Imbalance classes)

La plupart des classifications du monde réel présentent un certain niveau de déséquilibre entre les classes (Imbalance class), ce qui signifie que chaque classe n'est pas représentée par une part égale de l'ensemble de données. Il est toutefois important que les mesures et les méthodes soient correctement adaptées aux objectifs de chaque enquête. Par exemple, supposons qu'il y ait 2 classes A et B. La classe A représente 90 % de l'ensemble de données et la classe B les 10 % restants, mais le plus grand intérêt est d'identifier les cas de classe B. Il est facile d'atteindre une précision de 90 % en prédisant simplement la classe A à chaque fois, mais cela ne contribue en rien à l'objectif initial de la recherche. Au contraire, une méthode correctement calibrée peut atteindre une précision moindre, mais aura un taux de vrais positifs (ou de rappel) beaucoup plus élevé. Dans ce document, le problème du déséquilibre des classes est également abordé. En particulier, le plus grand nombre de phrases rencontrées dans les journaux sont des phrases sûres ou inconnues. Pour résoudre ce problème, des poids de classe ont été utilisés pour introduire des pondérations différentes dans les différentes classes. Plus précisément, cela a été fait avec la méthode

[2] Les poids ici "explosent", c'est-à-dire que leurs valeurs augmentent rapidement. Essentiellement, pour avoir un gradient qui disparaît, nous aurions $0 < w < 1$ et un gradient qui explose lorsque $w > 1$.

sklearn.utils.class weight.compute class weight('balanced', unique classes2, y ints2) (Pedregosa et al., 2011). La section 2.9.2 traite des mesures qui sont plus objectives lorsqu'il y a un déséquilibre entre les classes.

2.7 Apprentissage par transfert

Le traitement du langage naturel est un outil puissant, mais dans le monde réel, nous sommes souvent confrontés à des tâches qui présentent un manque de données et une mauvaise généralisation des modèles. En particulier, les textes sont si divers, bruyants et non structurés. L'apprentissage par transfert résout ce problème en nous permettant de prendre un modèle pré-entraîné d'une tâche et de l'utiliser dans d'autres. Il s'agit d'une méthode très populaire, notamment dans le domaine de la vision par ordinateur, qui permet de construire des modèles précis en « timesaving way" (Rawat et Wang, 2017). Pour le traitement automatique des langues (NLP) en particulier, ce concept représente un passage de l'utilisation de l'intégration des mots, qui a été utilisée dans de nombreuses tâches de NLP ces dernières années, à l'utilisation de représentations plus sophistiquées et abstraites. Avec cette méthode, au lieu de commencer le processus de formation à partir de zéro, on part d'une connaissance antérieure qui a été formée lors de la résolution d'un problème différent, ce qui permet d'utiliser les connaissances antérieures et de ne pas apprendre quelque chose de nouveau en partant de zéro. Une définition plus formelle de l'apprentissage par transfert est la suivante :

Supposons que nous ayons Ds un domaine source avec sa tâche source correspondante Ts, et également un domaine cible Dt avec sa tâche cible Tt. L'objectif de l'apprentissage par transfert est de nous permettre d'apprendre la distribution de probabilité conditionnelle cible P (Yt Xt) dans Dt avec les informations acquises dans Ds où Ds = Dt ou Ts = Tt. Cette méthode peut être divisée selon les formules suivantes :

- Adaptation du domaine
- L'apprentissage inter linguistique
- Apprentissage multi-tâches
- Apprentissage par transfert séquentiel

Dans ce travail, le dernier type d'apprentissage par transfert Séquentiel.est suivi.

2.7.1 Apprentissage par transfert séquentiel

Comme son nom l'indique, l'apprentissage par transfert séquentiel (STL) implique le transfert de connaissances en une séquence d'étapes, où les tâches source et cible ne sont pas nécessairement similaires. La STL se compose de deux étapes. Dans la première phase de pré-entraînement(pre-trained), le modèle est entraîné sur les données sources et dans la seconde phase d'adaptation, le modèle source est entraîné pour la tâche cible.

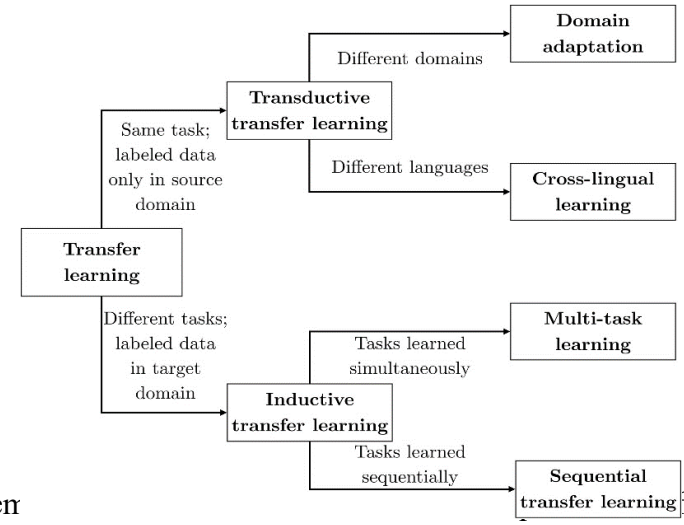


Figure 2.19 : Types d'apprentissage par transfert (Ruder, 2019).

La tâche de préapprentissage est généralement plus rapide car elle agit comme une étape de réglage fin. Le STL est généralement utile dans ces trois cas de figure :

- Les données de la tâche source et de la tâche cible ne sont pas disponibles en même temps.
- La tâche source a plus de données que la tâche cible.
- Un ajustement est nécessaire sur plusieurs tâches cibles.

En ce qui concerne la préformation, en général, pour obtenir le maximum de bénéfices, nous voulons une formation source qui profitera à de nombreuses tâches cibles. Il est difficile de trouver une telle tâche dans la pratique, mais c'est toujours mieux que de partir de zéro. En ce qui concerne la formation à la source, nous pouvons la réaliser de trois manières :

- Supervision à distance
- Supervision traditionnelle
- Aucune supervision

Nous suivons la troisième voie, celle qui est sans supervision. Cette méthode, également connue sous le nom de modélisation du langage, nécessite l'accès à un grand nombre de données non étiquetées pour nous, ces données sont les journaux. Par rapport à l'apprentissage supervisé, il s'agit d'une approche beaucoup plus évolutive, car le texte pour n'importe quel domaine est facilement disponible. Cette approche couvre des connaissances beaucoup plus générales sur la langue, par rapport à l'approche suivante l'apprentissage supervisé qui ne saisit que les caractéristiques nécessaires à la tâche.

En ce qui concerne la deuxième étape de la STL, le réglage, nous avons deux approches pour utiliser un modèle pré-entraîné pour la tâche cible - l'extraction de caractéristiques et le réglage fin. La première utilise les représentations d'un modèle pré-entraîné et les introduit dans un autre modèle, tandis que le réglage fin implique l'entraînement du modèle pré-entraîné sur la tâche cible.

- **Feature extraction (EX)**

Les poids du modèle sont gelés et la sortie de celui-ci est envoyée directement à un autre modèle. Les caractéristiques peuvent être envoyés à un modèle entièrement connecté ou nous pouvons simplement former un modèle classique tel qu'une machine à vecteurs de support ou une forêt aléatoire sur eux. L'avantage d'utiliser cette méthode est que le modèle spécifique à la tâche peut être utilisé pour des données similaires. En outre, si les mêmes données sont utilisées à plusieurs reprises, l'extraction des caractéristiques une seule fois peut permettre d'économiser beaucoup de ressources informatiques.

- **Fine-tuning (FT)**

Les poids sont maintenus entraînables et ajustés finement pour la tâche cible. Ainsi, le modèle pré-entraîné agit comme un point de départ pour le modèle, ce qui conduit à une convergence plus rapide par rapport à une initialisation aléatoire.

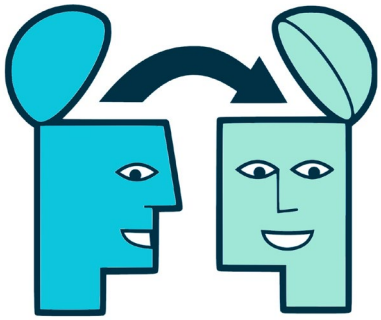
Pour résumer ce qui précède, dans EX, nous avons l'avantage de créer des fonctionnalités une fois et de tester différents modèles avec eux, ce qui permet d'économiser de précieuses ressources informatiques à des fins de recyclage et d'expérimentation. Par ailleurs, le CE est idéal à la fois pour affiner le modèle à utiliser pour de nombreuses tâches différentes, et pour faciliter notre travail, puisque nous n'avons pas besoin d'expérimenter une quelconque variation de notre modèle.

Peters et al.2019 (Peters et al., 2019), ont effectué une analyse sur l'effet de l'EX et du FT sur sept tâches différentes. Ils ont constaté que les deux approches atteignent les mêmes performances la plupart du temps, mais que l'ajustement fin est plus performant lorsque les tâches source et cible sont similaires, avec...

L'inconvénient, bien sûr, est que seuls les mots qui apparaissent dans la formation auront des embeddings mis à jour, tandis que les embeddings des mots non visibles seront oubliés[3]. L'extraction de caractéristiques donne de meilleurs résultats lorsque les tâches source et cible sont différentes.

En général, si le contenu de notre jeu de données n'est pas radicalement différent de celui du jeu de données sur lequel le modèle pré-entraîné a été formé, nous devons procéder à un ajustement fin.

Figure 2.20: transfert learning



.

Pour notre jeu de données, qui est constitué des journaux HPC, nous suivrons l'approche du réglage fin. Pour la méthode de réglages fine-tuning, il existe quelques techniques spécifiques que nous pouvons utiliser, comme :

1. La pratique courante consiste à tronquer le dernier (tronquez la dernière couche) (couche softmax) de la couche pré-trained et le remplacer par la nouvelle couche softmax associée à notre problème. Par exemple, les données pré-entraînées le modèle d'ImageNet comprend une couche softmax avec 1000 catégories. Si notre tâche était une classification de 10 catégories, notre nouvelle couche softmax du réseau serait de 10 catégories au lieu de 1000. Nous effectuons ensuite une rétropropagation du réseau pour affiner les poids pré-entraînés. Ici, la validation croisée est nécessaire pour que le réseau puisse bien se généraliser.

[3] Cela peut affecter les performances lorsque l'ensemble à former est trop petit ou que l'ensemble de test contient beaucoup d'éléments hors vocabulaire (OOV).

2. Utiliser un taux d'apprentissage plus faible (smaller learning rate) pour former notre réseau. Comme nous nous attendons à ce que les poids pré-entraînés soient déjà assez bons par rapport aux poids initiaux aléatoires, nous ne voulons pas les surentraîner trop rapidement et trop fortement.

3. Il est très courant de figer les poids des premières couches (freeze the weights of the first few layers) du modèle pré-entraîné. En effet, les premières couches enregistrent les caractéristiques globales, par exemple, dans un problème d'images, elles enregistrent les courbes, et les bords, qui sont liés au nouveau problème d'image. Nous voulons garder ces poids intacts. Au lieu de cela, nous ferons en sorte que notre réseau se concentre sur l'apprentissage de caractéristiques spécifiques de notre ensemble de données dans les couches suivantes.

Dans cet article, nous avons utilisé la deuxième et la troisième des techniques pour réglage fin présenté ci-dessus.

2.8 Performances des Reseaux neuronaux

2.8.1 Erreurs

Dans l'apprentissage automatique, notre objectif principal est de minimiser l'erreur, définie par la fonction de perte. Dans la plupart des réseaux neuronaux, l'erreur est calculée comme la différence entre la valeur réelle et la valeur prédite.

$$J(w) = p - \bar{p}$$

La fonction utilisée pour calculer cette erreur est connue sous le nom de fonction de perte, J. Des fonctions d'erreur différentes donneront des erreurs différentes pour la même prédiction et auront donc un effet significatif sur les performances du modèle. La fonction de perte est l'un des deux paramètres nécessaires à la construction d'un modèle (KERAS). Ainsi, si nous optimisons la mauvaise fonction de perte, nous arriverons à une mauvaise réponse, nous devons donc choisir la fonction appropriée. Dans cette thèse, nous utilisons (Categorical Crossetnropy) pendant la phase 1 et l'erreur quadratique moyenne (Mean Squared Error) pendant l'apprentissage du skipgramme et la phase 2.

- **Erreur quadratique moyenne (EQM) ou Mean Squared Error-MSE**

Calculé comme la moyenne des différences au carré entre les valeurs prédites et réelles. L'EQM ou MSE est une fonction de risque correspondant à la valeur attendue de la perte de l'erreur au carré.

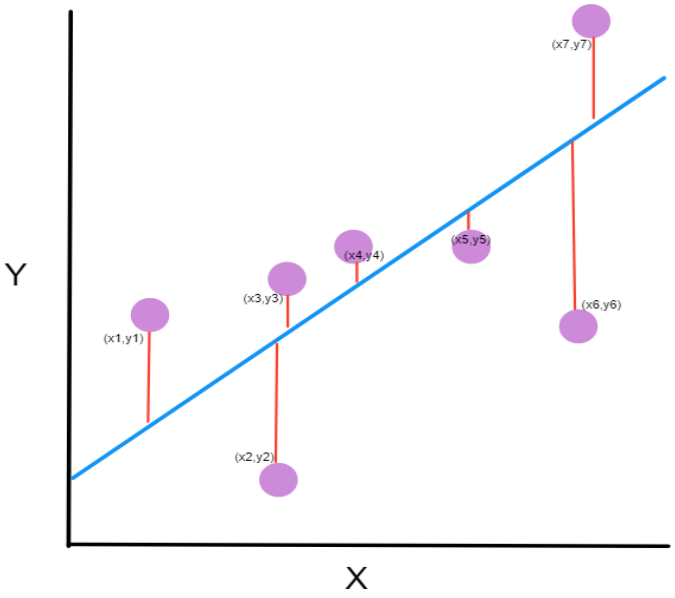


Figure 2.21 : Points sur un graphique simple.

- Les points violets sont les points du graphique. Chaque point a x et y coordonnées.

- La ligne bleue est notre ligne de prédiction. Cette ligne passe en revue tous les points et les fait correspond de la meilleure façon possible. Cette ligne comprend les points prédits.

- La ligne rouge entre chaque point violet et la ligne de prédiction représente les erreurs. Chaque erreur est la distance entre le point et le point prédit.

L'erreur quadratique moyenne indique à quel point une ligne de régression est proche d'un ensemble de points. On l'obtient en prenant les distances des points par rapport à la ligne de régression et en les élevant au carré. Cette mise au carré est nécessaire pour éliminer les éventuelles valeurs négatives – le résultat doit toujours être positif, quel que soit le signe des valeurs prédites et réelles. Elle donne également plus de poids aux plus grandes différences. Il est appelé ainsi car il identifie la moyenne d'un ensemble d'erreurs. Plus l'EQM est petite, plus nous sommes proches de trouver la ligne avec le meilleur ajustement.

- ## Categorical Cross-entropy

Qu'est-ce que le Categorical Cross-entropy ?

Cross-Entropy est une mesure de la différence entre deux distributions de probabilité pour une variable aléatoire ou un ensemble d'événements donnés. L'information, comme on l'appelle, quantifie le nombre de bits nécessaires pour coder et transmettre un événement. Comme on le sait, les messages à faible probabilité contiennent plus d'informations, tandis que les messages à forte probabilité en contiennent moins.

L'entropie ou (Entropy) – est une fonction de l'état du système. Ainsi, la variation de l'entropie d'un système est déterminée par les états initial et final. Idéalement, si un processus est réversible, l'entropie ne change pas, alors que les processus irréversibles augmentent toujours l'entropie totale. Parce qu'elle est déterminée par le nombre de micro-états aléatoires, l'entropie se rapporte à la quantité d'informations supplémentaires nécessaires pour déterminer l'état physique exact d'un système compte tenu de ses spécifications macroscopiques. C'est pourquoi on dit souvent que l'entropie est l'expression du désordre, du caractère aléatoire d'un système ou du manque d'informations à son sujet. Le concept d'entropie joue un rôle central dans la théorie de l'information. En particulier, l'entropie est un logarithme mesure du nombre d'états ayant une probabilité significative d'être « occupé ».

Une distribution asymétrique[4] a une entropie plus faible, tandis qu'une distribution où les événements du message ont la même probabilité a une entropie plus grande. En théorie de l'information, nous voulons décrire la « surprise » d'un événement.

- **Probability Distribution** (sans surprise) : faible entropie.
- **Balanced Probability Distribution** (surprenante) : entropie élevée.

L'entropie croisée (EC) Categorical Cross-Entropy est donc basée sur l'idée d'entropie de la théorie de l'information et calcule le nombre de bits nécessaires pour représenter ou transmettre un événement à partir d'une distribution comparée à un autre. La perte de CE peut être définie comme suit :

[4] On dit d'une distribution qu'elle est asymétrique lorsque les points de données sont plus concentrés d'un côté de l'échelle que de l'autre, créant une courbe qui n'est pas symétrique. En d'autres termes, les côtés droit et gauche de la distribution ont une forme différente l'un de l'autre.

$$CE = -\sum_{i}^{C} t_i \, log\,(s_i)$$

Où ti et si sont le ground truth et le score pour chaque classe i dans C.

L'entropie croisée catégorielle est utilisée pour les problèmes multi-classes. Les catégories de classe sont codées à un coup, ce qui signifie qu'il y a une représentation binaire pour chaque classe, seule la classe positive Cp conserve son terme dans la perte. Il n'y a qu'un seul élément du vecteur cible t qui n'est pas nul ti = tp. De plus, les prédictions doivent être des probabilités prédites pour chacune des classes. CE peut être écrit comme suit :

$$CE = -Log\left(\frac{e^{s_p}}{\sum_{j}^{C} e^{s_j}}\right)$$

Où **Sp** est le score de la classe positive.

2.8.2 Métriques

Plusieurs métriques sont utilisées dans cet ouvrage. Une métrique est une fonction utilisée pour juger les performances d'un modèle. Étant donné les valeurs prédites et la (ground truth)[5] cette convolution donne une mesure graduée de l'adéquation du modèle, avec les données existantes. Une fonction métrique est similaire à une fonction d'erreur, sauf que les résultats de l'évaluation d'une métrique ne sont pas utilisés dans la formation du modèle.

- la Matrice de confusion (Confusion Matrix)

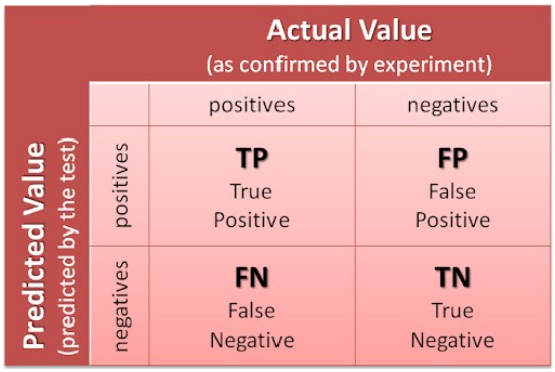
Dans le domaine de l'apprentissage automatique, une matrice de confusion, également appelée matrice d'erreurs, (Stehman, 1997) est une disposition matricielle spécifique qui permet de visualiser les performances d'un algorithme d'apprentissage supervisé (dans le cas d'un apprentissage non supervisé, on parle généralement de matrice de correspondance (Matching

[5] La ground truth est ce qui est mesuré pour la variable cible pour les exemples d'apprentissage et de test. Habituellement, les valeurs de vérité terrain sont traitées de la même manière que les étiquettes. Il y a bien sûr des cas où il ne correspond pas exactement à l'étiquette.

Matrix)). Chaque ligne de la matrice représente les instances dans une classe prédite tandis que chaque colonne représente les instances d'une classe réelle (ou vice versa) (Powers, 2011). Ce nom vient du fait qu'il permet de déterminer facilement si le système confond deux classes (par exemple, qui se confondent souvent).

Il s'agit d'un type particulier de tableau de contingence, avec deux dimensions ("réel" et "prédit") et des ensembles identiques de "classes" dans les deux dimensions (chaque combinaison de dimension et de classe est une variable dans le tableau de contingence).

Figure 2.22 : Table de confusion.



Toutes les prédictions correctes sont situées sur la diagonale de la matrice, il est donc facile d'inspecter visuellement la matrice pour détecter les erreurs de prédiction, car elles seront représentées par des valeurs situées en dehors de la diagonale.

True Positive-TP : Un vrai positif est une prévision dont on a estimé qu'elle appartenait à une classe particulière et qui est vraie.

False Positive-FP : Le faux positif est le nom donné à une prévision dont on a estimé qu'elle appartenait à une classe particulière et qui s'avère fausse.

False Negative-FN : Le faux négatif est le nom donné à une prévision dont on a estimé qu'elle n'appartenait pas à une classe particulière et qui s'avère fausse.

True Negative-TN : Un vrai négatif est le nom donné à une prévision dont on estime qu'elle n'appartient pas à une classe et qui est vraie.

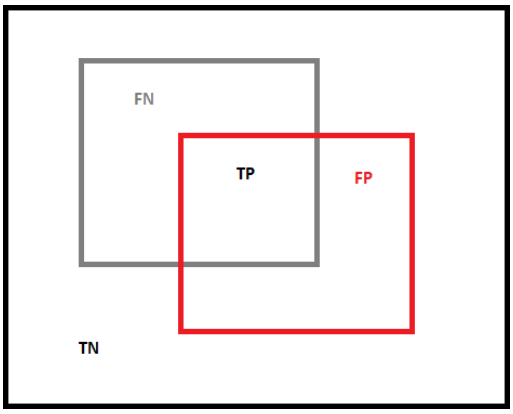


Figure 2.23 : Exemple sur les classes de predictions.

FP-FN-TP-TN, la classe est la boîte grise et TP est à l'intersection des boîtes grise et rouge, les prédictions estimées appartenir à la classe et être valides. Dans la case grise FN sont les prédictions estimées ne pas appartenir à la classe et ce n'est pas vrai. Dans la boîte rouge se trouvent les prédictions qui ont été estimées comme appartenant à la classe mais qui ne s'appliquent pas. Enfin, à l'extérieur de toutes les boîtes intérieures et à l'intérieur de la boîte noire se trouve TN qui est la prédiction estimée ne pas appartenir à la classe et qui est valide.

Les métriques suivantes sont utilisées dans la thèse : exactitude catégorielle et coefficient de Jaccard pour la phase 1, exactitude pour la phase 2 et rappel, précision et score f1 pour la phase 3.

- **Précision(accuracy) [6]**

La précision est une mesure qui évalue, comme son nom l'indique, l'exactitude de la prédiction du modèle par rapport aux données réelles, c'est-à-dire le rapport entre les résultats réels (vrais et négatifs) et le nombre total de cas testés.

$$F = \frac{TP + TN}{TP + FP + FN + TN}$$

- **Rappel[7]**

Le rappel (ou la sensibilité) est une mesure d'évaluation traditionnelle qui, comme toutes les mesures, suppose une notion de vérité fondamentale. Il s'agit de la fraction des prédictions réussies d'une classe par rapport au nombre total d'exemples de cette classe. Cette mesure peut être interprétée comme la quantité d'échantillons positifs qui ont été effectivement classés comme positifs. Moins un classificateur donne de faux négatifs (FN), plus son rappel est élevé.

Le rappel peut donc être considéré comme une mesure de l'exhaustivité ou de la quantité. Il est également utilisé dans la partie évaluation.

$$\frac{TP}{TP + FN}$$

- **Précision[8]**

Elle indique la proportion de prédictions réussies par rapport au total des prédictions d'une classe à partir d'un modèle. Intuitivement, la précision peut être considérée comme la capacité du classificateur à prédire que seuls les échantillons positifs vrais sont positifs. Ainsi, s'il n'y a pas de faux positifs (FP), c'est-à-dire s'il ne classe que des vrais positifs, il aura une précision de 1. Ainsi, en substance, moins un classificateur donne de FP, plus sa précision est élevée.

Ce paramètre peut être considéré comme une mesure de la qualité.

[6] https://developers.google.com/machine-learning/crash-course/classification/accuracy
[7] https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html
[8] https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html

$$\frac{TP}{TP + FP}$$

Ainsi, plus la précision et le rappel sont élevés, meilleures sont les performances du classificateur car il détecte la plupart des échantillons positifs (rappel élevé) et ne détecte pas beaucoup d'échantillons qui ne devraient pas être détectés (précision élevée).

Pour quantifier cela, nous utilisons une autre métrique appelée F1 Score.

- **F1 SCORE**

Cette métrique est utilisée dans la partie évaluation et combine la précision et le rappel. En particulier, il s'agit de la moyenne pondérée de la précision et du rappel. Le score F1 atteint la meilleure valeur de 1 (précision et rappel parfaits) et la pire valeur de 0.

$$\frac{2*(Precision * Recall)}{Precision + Recall}$$

Les mesures ci-dessus sont utilisées non seulement pour évaluer les modèles formés mais aussi pour évaluer la méthodologie globale utilisée pour prédire la défaillance ultime des nœuds. Le tableau ci-dessous montre à la fois la manière et la logique par lesquelles ces métriques sont calculées dans le contexte de la prédiction de la défaillance des nœuds.

Tableau 2.1 : Mesures d'évaluation

Metric	**Formula & Implication**
Recall	TP/(TP+FN) # Nodes failures correctly predicted
Precision	TP/(TP+FP) # Total node failures predicted
FP Rate	FP/(FP+TN) # False Positive Rate
True Positive (TP)	# Actual node failures, successfully predicted
True Negative (TN)	# Nodes actually didn't fail, are not be predicted as failures
False Positive (FP)	# Nodes actually didn't fail, but are predicted as failure
False Negative (FN)	# Actual node failures, but failed to be predicted

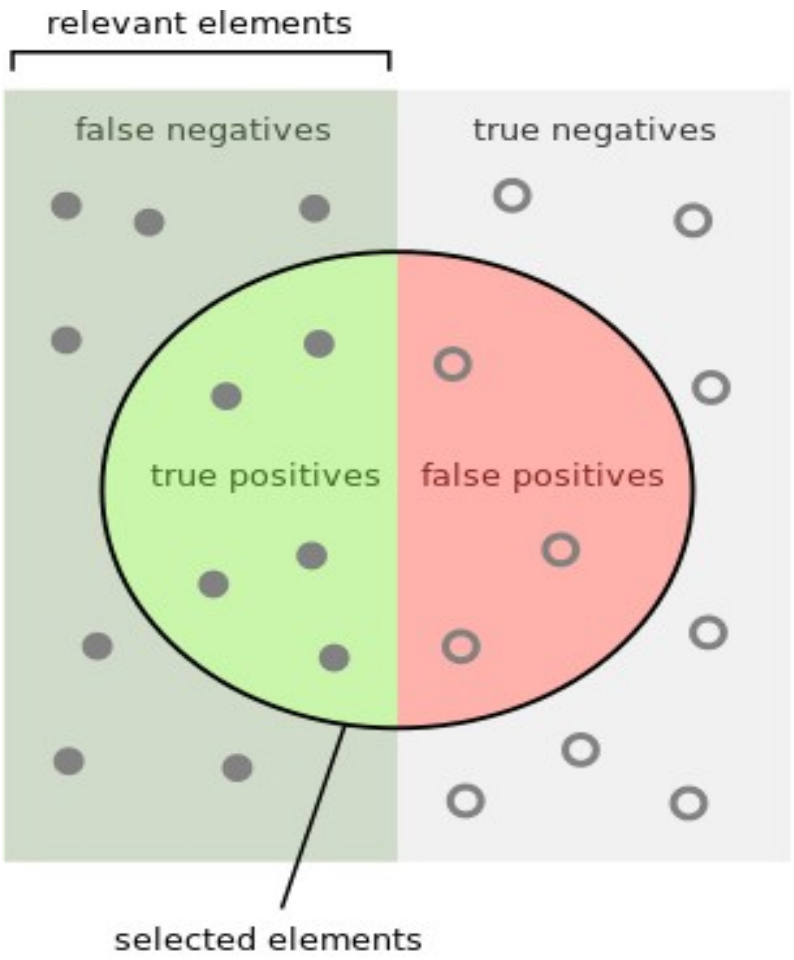


Figure 2.24 : Précision et Rappel.

2.9 Optimisation

Les algorithmes d'optimisation minimisent une fonction d'erreur J(x), qui est simplement une fonction mathématique qui dépend des paramètres internes du modèle utilisé pour calculer les valeurs cibles (Y) à partir de l'ensemble des valeurs prédictives (X) utilisées dans l'algorithme d'optimisation modèle. Les paramètres internes d'un modèle jouent un rôle très important dans

l'apprentissage efficace et efficient du modèle et produisent des résultats précis. C'est pourquoi nous utilisons diverses stratégies et algorithmes d'optimisation pour mettre à jour et calculer les valeurs appropriées et optimales de ces paramètres de modèle qui affectent le processus d'apprentissage de notre modèle ainsi que sa sortie. En d'autres termes, il calcule comment modifier les poids de notre réseau neuronal afin que l'erreur devienne plus faible à chaque itération. Le but de tous les optimiseurs est donc d'atteindre les minima globaux où la fonction de coût atteint la valeur minimale possible. Si nous essayons de visualiser la fonction de coût en 3 dimensions, nous obtiendrons quelque chose comme la figure ci-dessous :

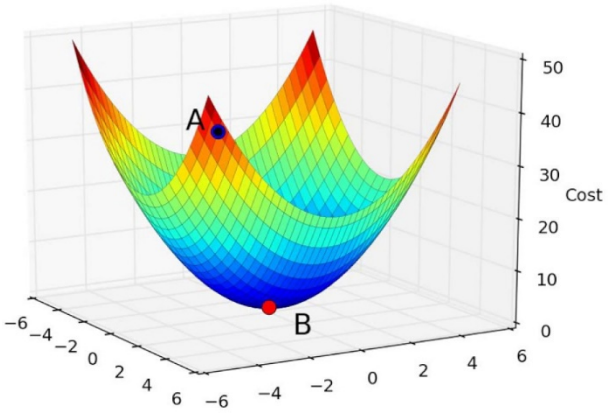


Figure 2.25 : fonction de coût convexe.

Bien entendu, la fonction de coût n'est pas toujours aussi lisse que dans la figure 2.25. La plupart du temps, les fonctions de coût seront non convexes.

Le problème avec la fonction non convexe est qu'il est possible que nous restions bloqués à un minimum local et que la perte ne converge jamais vers un maximum global. Comme dans la figure ci-dessous :

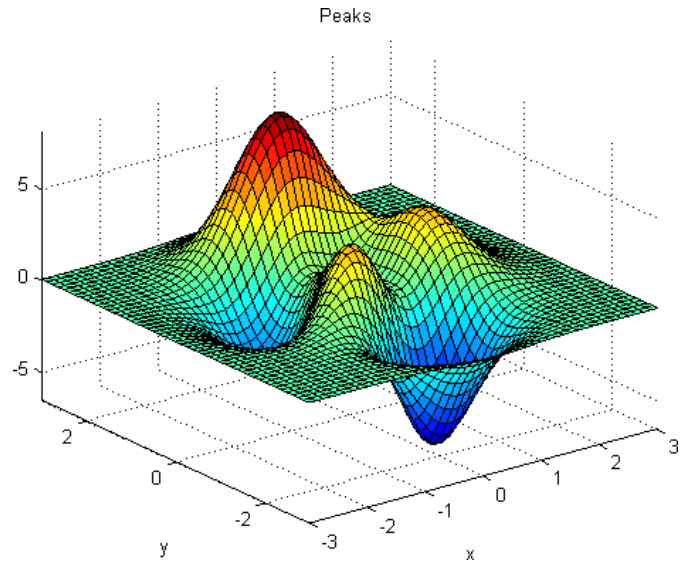


Figure 2.26 : fonction de coût non convexe

- **Taux d'apprentissage (Learning rate)**

Le taux d'apprentissage est peut-être l'aspect le plus important de la descente de gradient et de nombreux autres optimiseurs. Le taux d'apprentissage est un hyperparamètre qui contrôle le degré d'ajustement des poids du réseau par rapport au gradient de perte. Il détermine la taille des pas effectués pour s'approcher d'un minimum (local). En d'autres termes, la direction de la pente descendante de la surface créé par la fonction objective est suivie jusqu'à ce que la vallée soit atteinte. En d'autres termes, supposons que la fonction de coût soit une fosse, nous commencerons par le sommet de la fosse dans le but d'atteindre le fond de la vallée de la fosse. Ainsi, comme nous l'avons dit plus haut, nous pouvons considérer le taux d'apprentissage comme le pas que nous faisons pour atteindre le fond du puits (global minima).

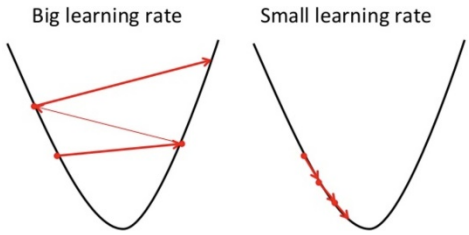


Figure 2.27 : Différents taux d'apprentissage (Gandhi).

Le choix d'un taux d'apprentissage approprié peut être difficile. Si nous choisissons un taux d'apprentissage élevé, nous effectuerons de grands changements dans les poids, les valeurs de biais, etc. – donc des sauts énormes pour arriver au fond. Mais on court le risque de dépasser les minima globaux (vallée) et de se retrouver de l'autre côté de la fosse. Ainsi, avec un taux d'apprentissage élevé, nous ne serons jamais en mesure de converger vers les minima globaux et nous errerons toujours autour d'eux. Si, en revanche, nous choisissons un taux d'apprentissage faible, nous n'avons pas le risque de dépasser le minimum global, mais notre algorithme mettra plus de temps à converger ; nous faisons des pas plus courts, mais nous devons en faire plus. Par conséquent, nous devrons nous entraîner pendant une période plus longue. (Gandhi) De plus, si la fonction de coût n'est pas convexe, l'algorithme peut facilement être piégé dans un minimum local et ne pourra pas converger vers le minimum pan-universel. Il n'y a pas de valeur correcte générale pour le taux d'apprentissage. Les programmes de taux d'apprentissage (Robbins et Monro, 1985) tentent d'ajuster le taux d'apprentissage pendant la formation, par ex. par recuit, c'est-à-dire en réduisant

le taux d'apprentissage selon un programme prédéterminé ou lorsque la variation inter-période de la cible tombe en dessous d'un seuil. Cependant, les programmes et les seuils doivent être définis à l'avance et ne sont donc pas capables de s'adapter aux caractéristiques d'un ensemble de données (Darken et al, 1992). En outre, le même taux d'apprentissage s'applique à toutes les mises à jour des paramètres. Si les données sont éparses et que les caractéristiques ont des fréquences très différentes, il peut ne pas être souhaitable de les mettre toutes à jour au même degré, mais d'effectuer une mise à jour plus longue pour les caractéristiques rares.

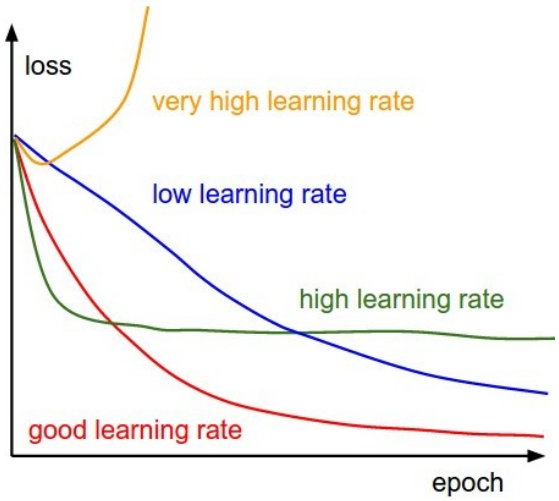


Figure 2.28 : L'effet de différents taux d'apprentissage sur la convergence.

Dans ce travail, les algorithmes d'optimisation utilisés sont SGD dans la phase 1 et RMSprop dans la phase 2 et skipgram.

2.9.1 Descente par gradient

La descente en gradient est la technique la plus importante et le fondement de la méthode dans lequel nous formons et optimisons les systèmes intelligents. Il s'agit de l'algorithme le plus populaire pour l'optimisation d'un réseau neuronal. Il est utilisé dans une large mesure pour effectuer des mises à jour de poids dans un modèle de réseau neuronal, c'est-à-dire pour mettre à jour et régler les paramètres de l'algorithme de réseau neuronal. Dans une direction afin de minimiser la fonction d'erreur.

Parallèlement, chaque bibliothèque moderne de Deep Learning comprend des implémentations de divers algorithmes d'optimisation par descente de gradient (par exemple, caffe et keras). Toutefois,

ces algorithmes sont souvent utilisés comme des optimiseurs de type boîte noire, car il est difficile de fournir des explications pratiques sur leurs capacités et leurs faiblesses.

La descente de gradient est un algorithme d'optimisation itératif, donc utilisé en apprentissage automatique pour trouver le meilleur résultat (le minimum d'une courbe).

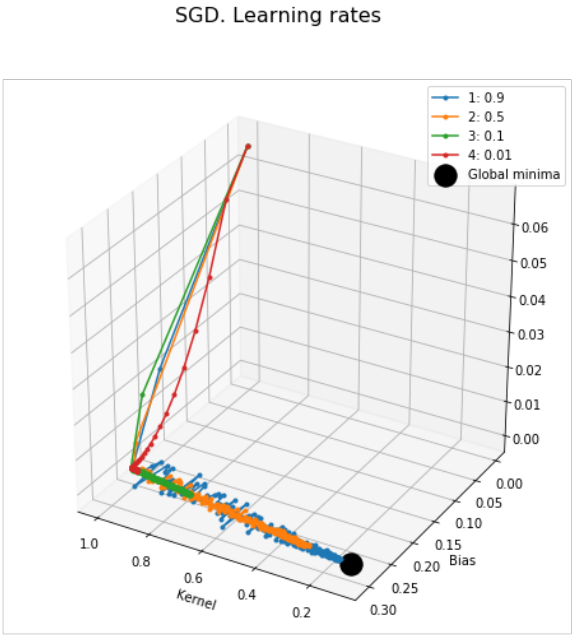


Figure 2.29 : Visualisation du processus d'apprentissage pour l'optimiseur SGD avec différents taux d'apprentissage.

Le gradient désigne le taux d'augmentation ou de diminution d'un gradient. La descente est le cas de la descente. L'algorithme est itératif car nous devons prendre les résultats plusieurs fois pour obtenir le résultat optimal. L'état itératif de la descente de gradient aide un graphique mal ajusté, par exemple, à s'ajuster de manière optimale aux données.

Dans la figure ci-dessus, nous voyons que dans la plupart des cas, les optimiseurs n'ont pas été en mesure d'approcher les minima dans le temps imparti. Ici, parmi de nombreuses autres raisons, la taille des pas était trop petite pour pouvoir avancer assez vite, et ils se sont donc arrêtés bien avant le minimum. L'optimiseur avec le taux d'apprentissage relativement plus élevé, d'autre part, a été en mesure d'atteindre le minimum et, avec un temps d'apprentissage suffisant, il finira par atteindre le point souhaité.

2.9.2 Descente de gradient stochastique-SGD

Descente de gradient stochastique – SGD effectue une mise à jour des paramètres pour chaque exemple d'apprentissage. Il s'agit généralement d'une technique beaucoup plus rapide. Il effectue une mise à jour à la fois pour chaque exemple d'apprentissage $x^{(i)}$ et chaque étiquette $y^{(i)}$:

$$\theta \leftarrow \theta - \eta \cdot \qquad (\theta ; x\ (x\ ) ; y\ (x\ ))$$

La descente de gradient par lots effectue des calculs inutiles pour les grands ensembles de données car elle redéfinit les gradients pour les exemples similaires avant chaque mise à jour des paramètres. La SGD supprime cette redondance en effectuant une mise à jour à la fois. Il est donc généralement beaucoup plus rapide et peut également être utilisé pour apprendre en ligne. SGD effectue des mises à jour fréquentes avec une grande variance, ce qui entraîne des mises à jour de paramètres avec une grande variance, ce qui provoque un changement important de la fonction objectif, comme dans la figure 2.30 :

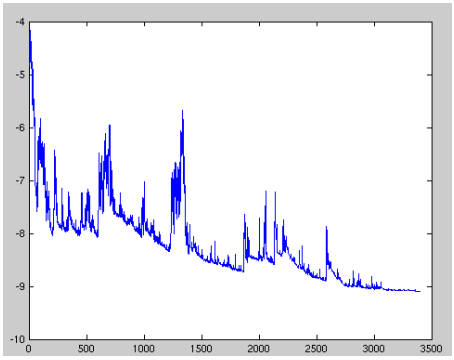


Figure 2.30 : Fluctuations du rapport : SGD (Ruder, 2016).

Les variations de SGD font varier la fonction d'erreur à des intensités différentes. C'est en fait une bonne chose car cela permet de découvrir de nouveaux minima locaux, peut-être meilleurs (Walia, 2017). D'autre part, cela finit par compliquer la convergence vers le minimum exact car le SGD continuera à dépasser. Cependant, il a été démontré que lorsque le taux d'apprentissage est lentement réduit, le SGD converge presque certainement vers un minimum local ou global pour l'optimisation non convexe et convexe respectivement (Ruder, 2016).

- **Taille du lot (Batch Size)**

Définit le nombre d'échantillons que le modèle voit avant de mettre à jour les poids. Par exemple, supposons que nous ayons 1050 échantillons d'entraînement et une taille de lot égale à 100. L'algorithme prend les 100 premiers échantillons (du 1er au 100e) de l'ensemble de données d'entraînement et entraîne le réseau. Il prend ensuite les 100 autres échantillons (du 101e au 200e) et entraîne à nouveau le réseau. Ce processus peut être poursuivi jusqu'à ce que tous les échantillons aient été propagés dans les réseaux. Le problème survient généralement avec la dernière série d'échantillons. Dans l'exemple particulier donné, il y a 1050 échantillons, mais 1050 n'est pas partagé avec 100 sans aucun reste. La solution la plus simple consiste à transmettre simplement les 50 derniers échantillons à l'algorithme et à entraîner le réseau. À ce stade, l'algorithme a effectué une période de temps.

De cette façon, moins de mémoire est nécessaire, car le réseau est entraîné en utilisant un plus petit nombre d'échantillons. Elle est particulièrement importante dans le cas où il n'y a pas de possibilité d'adapter l'ensemble des données à l'environnement de l'entreprise. La mémoire. Les réseaux sont généralement formés plus rapidement avec des mini-batchs. Ceci est dû au fait que les poids sont mis à jour après chaque propagation. Dans cet exemple, 11 lots ont été propagés (10 d'entre eux avaient 100 échantillons et 1 avait 50 échantillons) et après chacun d'eux les paramètres du réseau ont été mis à jour. Si tous les échantillons étaient utilisés dans la diffusion, une seule mise à jour du paramètre du réseau serait effectuée.

Bien entendu, plus le lot est petit, moins l'estimation de la pente est précise.

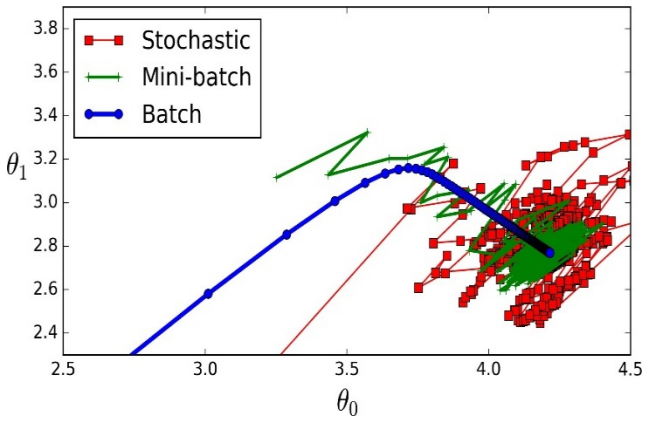


Figure 2.31 : Graphe de comparaison entre sgd et transfert par lot.

Sur cette figure, il est facile d'observer que la direction de la pente du mini-lot (couleur verte) varie par rapport au lot complet (couleur bleue).

- **La descente de gradient stochastique – SGD**

Il n'est qu'un mini-batch avec une taille de lot égale à 1. Le gradient change de direction encore plus fréquemment qu'un mini-batch. Lors de la formation ou de l'évaluation du réseau neuronal, les données sont diffusées par lots, comme indiqué ci-dessus. D'après ce qui précède, il est évident que le réseau neuronal sera alimenté par tous les lots. L'utilisation de tous les lots une seule fois correspond à 1 saison. Ainsi, s'il y a 10 époques, cela signifie que toutes les données (séparées en lots) seront utilisées 10 fois.

- **Momentum**

Le SGD avec momentum converge presque toujours plus rapidement que le SGD standard. Dans un SGD standard, nous ferions des pas plus longs dans une direction et des pas plus courts dans l'autre, ce qui ralentit l'algorithme.

Dans l'image ci-dessous, à gauche, nous voyons que le sgd standard fait de plus grands pas dans la direction y et de plus petits pas dans la direction x. Si notre algorithme est capable de réduire les pas effectués dans l'axe des y et de concentrer la direction des pas dans l'axe des x, alors l'algorithme convergera plus rapidement.

C'est ce que fait le momentum, il limite l'oscillation dans une direction afin que l'algorithme puisse converger plus rapidement. De plus, étant donné que le nombre de pas effectués dans la direction y est limité, nous pouvons fixer un taux d'apprentissage plus élevé (Gandhi).

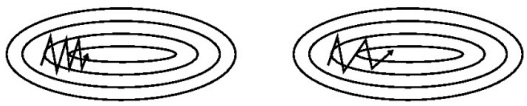


Figure 2.32 : Le sgd sans momentum à gauche et le sgd avec momentum à droite.

Le momentum (Qian, 1999) est une méthode qui permet d'accélérer le sgd dans la direction relative et de réduire les oscillations comme le montre la figure 2.32 à droite. Ceci est réalisé en ajoutant une fraction γ du vecteur de mise à jour du pas de temps précédent au vecteur de mise à jour actuel :

$$vt = \gamma vt - 1 + \eta \nabla \theta J(\theta)$$

$$\theta = \theta - vt$$

Note : Certaines applications échangent les points dans les équations. Le terme de momentum γ est généralement fixé à 0,9 ou à une valeur similaire.

Essentiellement, lorsqu'on utilise le momentum, c'est comme si on poussait une balle en bas d'une colline. La sphère accumule de l'élan en roulant vers le bas, de plus en plus vite en cours de route (jusqu'à atteindre sa vitesse finale s'il y a une résistance de l'air, c'est-à-dire $\gamma < 1$). Il en va de même pour les mises à jour des paramètres. Le terme d'élan augmente pour les dimensions dont les pentes pointent dans les mêmes directions et diminue les mises à jour pour les dimensions dont les pentes changent de direction. Par conséquent, on obtient une convergence plus rapide et une oscillation réduite.

2.9.3 RMSprop (Propagation de la moyenne des carrés)

L'optimiseur RMSprop est apparu pour la première fois dans les cours d'une série de cours en ligne sur les réseaux neuronaux dispensés par le professeur Geoffrey Hinton de l'université de Toronto. Hinton n'a pas publié RMSprop dans un article universitaire formel, mais il est largement utilisé dans l'apprentissage profond. Il l'a développé pour résoudre le problème qui était commun lors de l'utilisation de rprop avec des mini-batchs, qui est que les poids seraient ajustés en fonction de la taille du gradient pour chaque mini-batch, résultant en de très grandes augmentations ou diminutions de poids si les mini-batchs successifs n'ont pas de gradients similaires. Cela va bien sûr à l'encontre des résultats souhaités de la SGD, qui consiste à procéder à de petits ajustements des poids et des biais afin qu'un réseau neuronal soit calibré pour être de plus en plus performant sur une tâche convergente à chaque itération de l'algorithme d'optimisation.

Le problème qui peut se poser dans RMSprop si les mini-batchs successifs varient fortement est atténué par l'utilisation d'une moyenne mobile du gradient au carré pour chaque poids. Cela signifie

que le gradient de chaque mini-lot est divisé par la racine carrée de MeanSquare, où MeanSquare est calculé comme suit :

$$MeanSquare\ (w,t) = 0.9\ MeanSquare\ (w,t-1) + 0.1\left(\partial E/\partial w^{(t)}\right)^2$$

Cette procédure permet de calculer la moyenne des gradients dans les périodes successives des mini-lots afin que les poids puissent être calibrés avec précision.

De manière plus pratique, l'optimiseur RMSprop est similaire à l'algorithme SGD avec momentum. RMSprop limite essentiellement les oscillations dans la direction verticale.

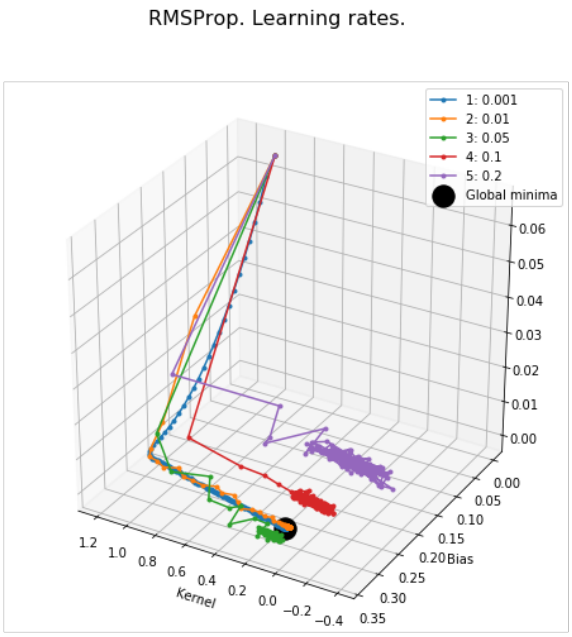


Figure 2.33 : Visualisation du processus de training pour l'optimiseur RMSprop avec différents taux d'apprentissage.

Ici, tous les optimiseurs ont pu s'approcher de la région entourant le minimum, mais ceux dont le taux d'apprentissage est le plus élevé (0,2, 0,1 et 0,05) en sont un peu plus éloignés. Leurs premiers pas étaient plutôt plus loin, ils ont donc manqué de peu le chemin dans la bonne direction, ils n'ont pas pu le retrouver.

Chapitre 3 : Méthodologie et modèles spécifiques

3.1 Skip-Gram

Dans ce travail, une partie de la méthodologie de prédiction des chaînes de défaillance est l'entraînement du modèle Skip-gram pour obtenir des encastrements de mots. Le modèle Skip-gram tente généralement d'atteindre l'exact opposé de ce que le modèle CBOW tente d'atteindre. Il tente de prédire les mots du contenu source (mots autour du mot cible) à partir d'un mot cible (mot central).

Supposons que nous ayons la phrase suivante **the quick brown fox jumps over the lazy dog**. Si nous utilisons le modèle CBOW, nous obtenons des paires de (**context window, target word**), ou si nous considérons une taille de fenêtre de 2, nous obtenons des exemples comme ([rapide, renard], brun), ([le, brun], rapide), ([le, chien], paresseux) et ainsi de suite. En supposant maintenant que l'objectif du modèle Skip-gram est de prédire le contexte à partir du mot cible, le modèle inverse généralement le contenu tente de prédire chaque mot du contexte à partir de son mot cible.

La tâche se transforme donc en une prédiction du contexte-contexte, ([rapide, renard]) étant donné le mot cible-brun, ou ([le brun]) étant donné le mot rapide, et ainsi de suite. Ainsi, le modèle tente de prédire le (mot contextuel) sur la base du (mot cible).

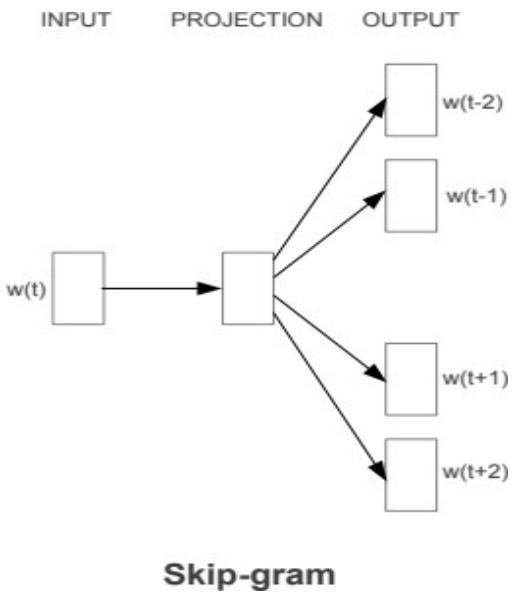


Figure 3.1 : L'architecture du modèle Skip-gram (Mikolov et al., 2013).

Nous modélisons l'architecture Skip-gram comme un modèle de classification par apprentissage profond, de sorte que nous prenons (mot cible) comme entrée et essayons de prédire (mots de contexte). Cela devient plus complexe s'il y a beaucoup de mots dans notre contenu (contexte). Nous simplifions encore le processus en divisant chaque paire (cible, mots du contexte) en paires (cible, contexte) de sorte que chaque contexte consiste en un seul mot. Par conséquent, notre ensemble de données original est transformé en paires telles que (brun, rapide), (brun, renard), (rapide, le), (rapide, brun) et ainsi de suite. Mais comment former ou superviser le modèle pour qu'il sache ce qui est contextuel et ce qui ne l'est pas ?

Pour cette raison, nous alimentons le modèle Skip-gram avec des paires (X, Y) où X est notre entrée et Y est notre étiquette. Pour ce faire, nous utilisons des paires [(cible, contexte), 1] comme échantillons d'entrée positifs, où (cible est notre mot d'intérêt et contexte est le mot de contenu qui se produit près du mot (cible et l'étiquette positive 1 indique qu'il s'agit d'une paire contextuelle de contextes. Nous lui fournissons également des paires [(cible, aléatoire), 0] comme échantillons d'entrée négatifs (la cible est à nouveau le mot d'intérêt).

Mais aléatoire est un mot choisi au hasard dans notre vocabulaire et qui n'a aucun contenu ou relation avec notre mot cible). L'étiquette négative 0 indique donc qu'il s'agit d'une paire non pertinente en termes de contexte. Nous faisons cela afin que notre modèle puisse apprendre quelles paires de mots sont contextuellement pertinentes et lesquelles ne le sont pas, ce qui permet d'obtenir des incorporations similaires pour des mots sémantiquement similaires.

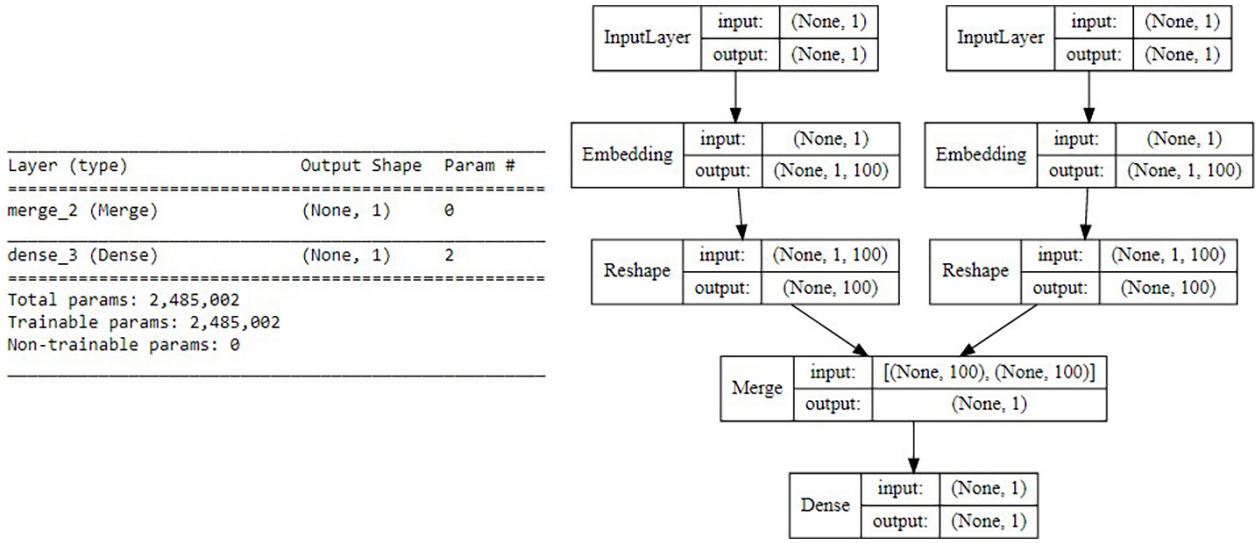
Pour obtenir la paire de mots avec leur contexte, nous utilisons une fonction utilitaire utile de Keras (KERAS), les skipgrams.

Cette fonction convertit une séquence de pointeurs de mots (liste d'entiers) en tuples de mots de la forme :

- (word,word in the same window) étiqueté 1 (échantillons positifs).
- (word,random word from the vocabulary) étiqueté 0 (échantillons négatifs).

Pour l'architecture d'apprentissage profond du modèle de saut de programme, nous aurons comme entrées le mot cible et le contexte ou des paires de mots aléatoires. Chacun d'entre eux est transféré à sa propre couche d'incorporation (initialisée avec des poids aléatoires) avec une taille (vocab sizexembed size) qui nous donnera des incorporations de mots denses pour chacun de ces deux mots (1xembed sizeforeachword). Après avoir obtenu les incorporations de mots pour le mot cible

et le mot contextuel respectivement, nous les passons à une couche de fusion dans laquelle nous calculons le produit de ces deux vecteurs. Nous transférons ensuite ce produit à une couche dense avec une fonction d'activation sigmoïde qui prédit soit 1 soit 0 selon que la paire de mots est liée en termes de contenu ou qu'il s'agit simplement de mots aléatoires (Y'). Nous comparons ce résultat à l'étiquette de pertinence réelle (Y), nous calculons la perte en exploitant la perte de l'erreur quadratique moyenne et nous effectuons une rétropropagation sur les erreurs pour ajuster les poids dans la couche d'intégration et nous répétons le processus pour toutes les paires (target, context) pendant plusieurs époques.



```
____________________________________________________
Layer (type)                 Output Shape  Param #
====================================================
merge_2 (Merge)              (None, 1)     0
____________________________________________________
dense_3 (Dense)              (None, 1)     2
====================================================
Total params: 2,485,002
Trainable params: 2,485,002
Non-trainable params: 0
____________________________________________________
```


Figure 3.2: skip gram model summary. (Sarkar).

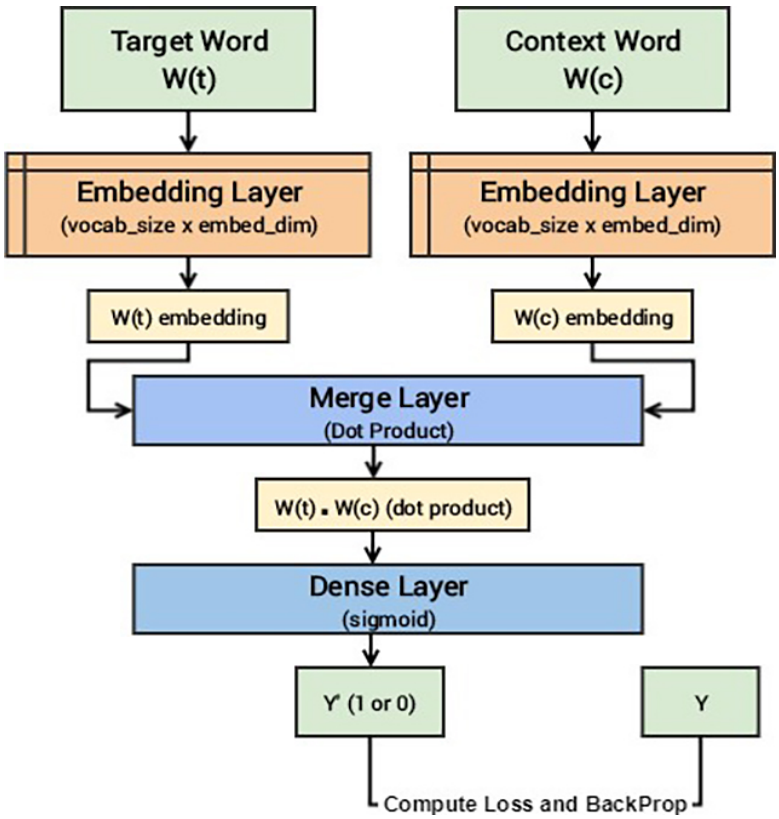


Figure 3.3: visual depiction of the skip-gram deep learning model(Sarkar).

3.2 Méthodologie

L'objectif de cette thèse est de prédire les défaillances du système afin de prévenir les défaillances de travail et de calculer les délais. Nous y parvenons en prévoyant les chaînes de défaillance possibles.

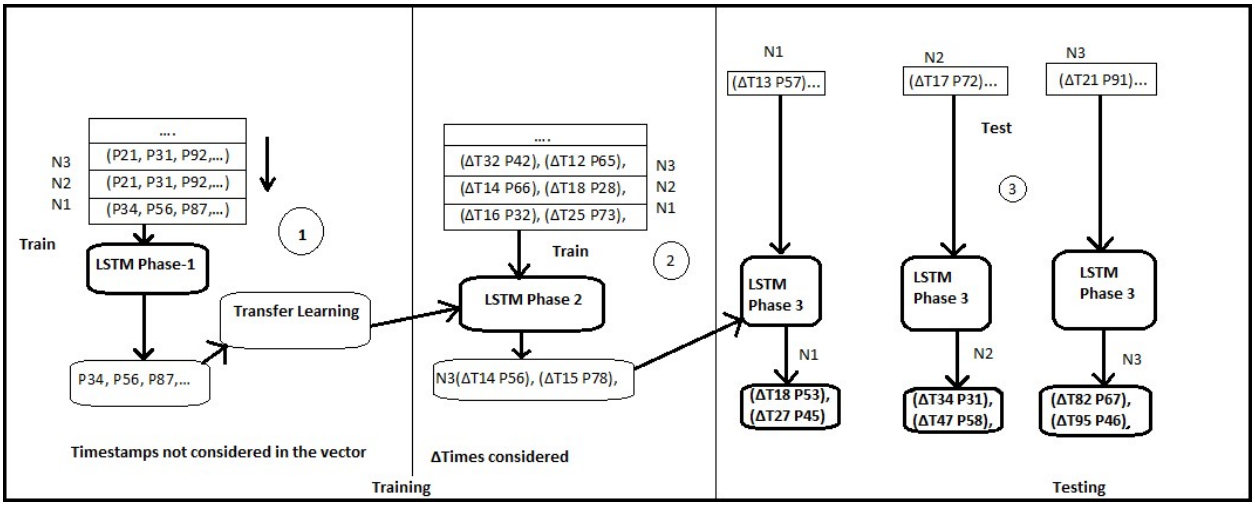


Figure 3.4 : phases du LSTM.

- **Phase 1**

La première phase consiste à former des chaînes de défaillance à partir de données brutes. Les journaux bruts de la machine BlueGene d'IBM contiennent des phrases parsemées d'anomalies, ainsi qu'un nombre important de bruits et d'événements sans danger. Un message de journal a un horodatage T1, une phrase d'événement P1 et un noeud N1. De même, de nombreuses phrases horodatées correspondront à des nœuds spécifiques, c'est-à-dire des phrases horodatées.

Le suivi des identifiants de nœuds permet de conserver l'emplacement spécifique où se produit l'erreur.

on entraîne le jeu de données basé sur les noeuds de la phase 1. En d'autres termes, les journaux de chaque nœud sont "épissés" et introduits dans le même LSTM, ce qui présente deux avantages. Tout d'abord, il n'y a pas de surcharge pour stocker l'identifiant du nœud et le traiter dans le vecteur, ce qui permet d'économiser de la mémoire et des coûts de calcul. Deuxièmement, afin de permettre que notre modèle apprend les modèles à partir des chaînes de défaillances observées, l'identifiant du nœud n'a pas d'importance ; ce qui est nécessaire, ce sont les phrases qui se produisent dans une séquence menant à diverses défaillances de nœuds. Dans cette phase, aucune chronologie n'est fournie, mais nous trions les messages du journal par leur horodatage pour cette raison. L'ordre des phrases est important ici, pas leur décalage horaire.

Dans la phase 1, les identifiants des phrases sont utilisés comme vecteurs. Chaque phrase d'événement est ensuite séparée en contenu statique et dynamique, de sorte que la partie fixe de la phrase est identifiée de la partie variable de la phrase.

La phrase P1 est donc divisée en une partie statique et une partie variable. La partie dynamique est rejetée. Les exemples suivants illustrent la séparation des phrases en parties statiques et dynamiques :

Tableau 3.1 : Exemples de parties de phrases statiques et dynamiques

static	dynamic
DDR Correctable Error Summary	Count=8713 MCFIR error status :[POWERBUS_WRITE_BUFFER_CE] this bit is set when a PBUS ECC CE is detected on a PBus write buffer read op;
L1P Correctable Error Summary	Count=27355 cores=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 L1P_ESR :[ERR_RELOAD_ECC_X2] correctable reload data ECC error;
DDR Correctable Error Summary	Count=236 MCFIR error status : [MEMORY_CE] this bit is set when a memory CE is detected on a non-maintenance memory read op;
L1P Correctable Error Summary	Count=2 cores 3,13 L1P_ESR : [ERR_RELOAD_ECC_X2] correctable reload data ECC error;

Lorsque toutes les parties fixes sont extraites, elles sont encodées dans un numéro identifiable de manière unique. Ces phrases sont des entités conjuguées de plusieurs mots (par exemple ddrcorrectableerrorsummary). Nous aurions donc une séquence de phrases codées, c'est-à-dire par exemple 45, 67, 89, 40, pour le nœud N1. Mais les LSTM ne peuvent pas comprendre leur relation sémantique ou syntaxique sous cette forme discrète.

C'est là qu'interviennent les modèles d'espace vectoriel à représentation distribuée, qui contribuent à la création de la corrélation sémantique. Ces phrases codées reçoivent ensuite une forme vectorielle en utilisant des encastrements de mots. Comme nous l'avons mentionné dans la section 2.2, les incorporations sont des cadres définis qui contrôlent ce qui apparaît avant et après une phrase d'événement cible dans une série d'événements. Dans cette étude, on utilise le skip-gram pour extraire les incorporations de mots, qu'on décrit dans la section 3.1. Ainsi, pour chaque phrase codée dans les données, nous disposons d'un ensemble d'incorporations de phrases qui font référence à leur contenu. L'espace vectoriel est donc créé en utilisant les encastrements et ce qui apparaît avant et après une cible.

Donc, en substance, pour une séquence 45, 67, 89, 40, 89, 102, nous avons la probabilité que 40 soit observé après la séquence 45, 67, 89 et avant la séquence 89, 102. Ces vecteurs sont ensuite introduits dans un LSTM empilé utilisant deux couches cachées pour effectuer une prédiction en 3 étapes (prédire les 3 phrases suivantes).

Les tailles des fenêtres utilisées sont de 15 et 3, pour tenir compte, respectivement, du nombre de phrases à gauche et à droite d'une phrase cible donnée. Notre modèle dans la phase 1 utilise l'optimiseur SGDavec une perte d'entropie croisée catégorielle comme mentionné dans les sections

précédentes. La phase 1 est semi-supervisée et produit des séquences entraînées des vecteurs de phrases.

Étiquetage des phrases. On filtre ensuite les logs après avoir conservé leur partie fixe en suivant la procédure ci-dessus et on les classe en trois catégories : sûr, erreur et inconnu. Safe représente les phrases "sûres", qui ne sont en aucun cas associées à une quelconque anomalie du système. L'expression "erreur" désigne les phrases qui indiquent clairement une anomalie. L'étiquette "Inconnu" est attribuée aux phrases qui peuvent ou non être une indication d'une anomalie.

Il est important de noter que le fait de marquer une phrase comme Erreur ne signifie pas qu'elle fera toujours partie des défaillances du nœud, mais qu'elle peut ou non faire partie de la défaillance du nœud. Cependant, ces phrases seront soit des messages de terminal, soit des messages d'un dysfonctionnement matériel ou logiciel significatif détecté dans les journaux de Linux. De plus, comme mentionné dans la section 2.1, nous ne prenons pas en compte les niveaux de sévérité des journaux. En effet, la classification directe des journaux de messages sur la base de balises apparaissant occasionnellement n'est pas efficace pour les données sensibles à long terme, car de nombreux messages non critiques pourraient être un meilleur indicateur des défaillances dans le temps. De plus, comme nous l'avons déjà mentionné, certains événements apparemment "sûrs" dans un contexte peuvent entraîner des événements mortels dans un autre contexte. Par conséquent, nous considérons que les phrases sont indépendantes des drapeaux et des niveaux de gravité.

Exemples de phrases classées dans les catégories "sûr", "erreur" et "inconnu" sont les suivants :

Tableau 3.2 : Exemples de phrases basées sur leur balisage

SAFE	UNKNOWN	ERROR
ddrcorrectableerrorsummary	memorycontrollerinitializationwarning	badphywasdetected
I1pcorrectableerrorsummary	Warningmc1rank1	unabletoupdatethelcddisplay
serviceaction6585restartednodedcar13	encounteredanexception	Kernelunexceptedoperation (Indicator-panic)
serviceaction6585startedtoservicer13	Warninggmc0rank1	Cfamlivelockbusterfailure (Indicator-Hardware(H/W))

Les phrases sont représentées comme des entités conjuguées de plusieurs mots. La troisième colonne présente des phrases d'erreur, dont deux sont également des messages terminaux signalant des défaillances. L'une appartient à la classe Hardware, car elle fait référence à de telles erreurs,

tandis que l'autre appartient à la classe Panic, car elle fait référence à des messages de panique du noyau.

Après la classification, nous avons des phrases avec des étiquettes Safe(S), Error(E) ou Unknown(U). Les phrases sûres sont éliminées car notre intérêt principal est l'erreur et les phrases inconnues.

L'étiquetage des phrases n'est délibérément pas effectué avant la représentation vectorielle, car l'apprentissage devient plus robuste avec le bruit. De plus, les indicateurs d'événements dangereux ou non ne sont pas a priori pour l'apprentissage non supervisé. Le marquage est effectué pour optimiser le coût de calcul dans la phase 2. Par conséquent, une séquence d'événements menant à la défaillance d'un nœud est formée à l'aide de phrases étiquetées Unknown (U) et Error (E), pour autant que les messages terminaux, indiquant qu'un nœud est en panne, soient également connus.

- **Phase 2**

Dans la phase 1, la présence de bruit a rendu le calcul de T peu pratique, puisque nous devons considérer des phrases avec des anomalies menant à des défaillances de nœuds sans phrases sûres dispersées. Ainsi, dans cette phase, la phase 2, nous séparons les phrases qui forment des chaînes de défaillance du reste (qui ne font pas partie d'une chaîne de défaillance) et nous calculons les différences de temps entre ces phrases de chaîne pour prédire le délai d'exécution.

Comment calcule-t-on ΔT ?

Notre objectif est de prédire le délai menant à la défaillance d'un nœud. Dans une séquence de messages menant à une défaillance, il y a des messages "anormaux" avant les messages terminaux. L'"échec" qui se produit est indiqué par l'ordre supérieur de la série chronologique. Par conséquent, nous trions les données dans l'ordre décroissant des horodateurs et nous calculons le T, qui est la différence de temps cumulée entre la phrase actuelle et la dernière phrase dans le temps (ordre supérieur) de la séquence. La phrase la plus horodatée de la séquence se voit attribuer T=0, puisqu'il ne reste aucune autre phrase de la séquence pour calculer la différence de temps.

Tableau 3.3 : Exemple de chaîne de défaillance (Das et al., 2018).

#	Timestamp	Label	Phrase Vector
P1	03:59:58.466 (T1)	U	ΔT1=07.822, P1
P2	03:59:59.543 (T2)	U	ΔT2=06.745, P2
P3	04:00:00.477 (T3)	U	ΔT3=05.811, P3
P4	04:00:01.706 (T4)	E	ΔT4=04.582, P4
P5	04:00:01.731 (T5)	E	ΔT5=04.557, P5
P6	04:00:06.288 (T6)	E	ΔT6=00:000, P6

Dans le tableau 3.3, par exemple, la valeur 0 est attribuée à T6. Nous calculons ensuite les T en soustrayant respectivement les horodatages de chaque phrase de l'horodatage de T6 dans la chaîne de défaillance. Nous avons donc T6-T5, T6-T4, T6-T3, T6-T2, T6-T1 et de cette façon nous obtenons les différences de temps en secondes (7.822, 6.745, 5.811, 4.582, 4.557, 0).

Nous entraînons ensuite un nouveau modèle consistant à nouveau en un lstm empilé de deux couches cachées, cette fois avec des chaînes d'échec pour apprendre les différents T avec les identifiants des phrases et enfin prédire les heures à attendre dans le futur. L'entrée du LSTM dans la phase 2 sera un vecteur à 2 états contenant le T et l'id de la phrase correspondante, comme le montre la dernière colonne du tableau 3.3. Cette formation est un problème de séries temporelles multivariées, qui attend que les valeurs prédites soient suffisamment proches des valeurs réelles présentes dans l'ensemble de formation. Dans la phase 2, le LSTM reçoit des vecteurs avec une taille d'historique de 10 et effectue une prédiction en une étape pour chaque série avec deux couches cachées. En outre, on utilise l'apprentissage par transfert de la phase 1 à la phase 2, de sorte que le modèle formé de la phase 1 est un point de départ pour la formation du modèle de la phase 2, ce qui entraîne une convergence plus rapide par rapport à l'initialisation aléatoire.

La méthodologie suivie pour les phases 1 et 2 est l'entraînement des vecteurs en coalescence, c'est-à-dire que les vecteurs sont affectés à un nœud après l'autre. Ainsi, les deux modèles apprennent les séquences de défaillance des différents nœuds de manière séquentielle. La différence réside dans le contenu du vecteur d'entrée pour chaque message de journal associé à un nœud. Alors que la phase 1 ne contient que les ids des phrases, la phase 2 contient les différences de temps entre les phrases, ainsi que leurs ids. C'est-à-dire que dans la phase 2, étant donné les identifiants des phrases

et les différences de temps entre elles pour l'heure précédente, nous prédisons les identifiants des phrases dans les heures suivantes.

Dans la phase 3, par contre, les défaillances basées sur la formation précédente sont identifiées et appliquées aux nouvelles données de test (différentes des données de formation). Cependant, les vecteurs introduits dans le LSTM dans la phase 3 proviennent d'un nœud spécifique (non coalescé). Ainsi, nous apprenons d'abord de tous les nœuds et utilisons cet apprentissage global pour détecter les défaillances par nœud individuel pendant les tests et l'inférence dans la phase 3.

- **Phase 3**

Dans cette phase, le LSTM est utilisé dans les données de test pour valider les chaînes de défaillance entraînées de la phase 2. Les données du test sont traitées pour former des vecteurs codés avec des décalages horaires et des phrases similaires à ce qui a été dit en phase 2. Bien sûr, nous avons ici toutes sortes de phrases, sûr, erreur, inconnu. Dans la phase 3, nous devons garder la trace de l'identifiant du nœud, afin de savoir quel nœud est susceptible de tomber en panne, et combien de temps il reste avant une éventuelle panne. Donc, au-delà de cette phase, les vecteurs ne sont pas liés entre les nœuds comme dans les phases 1 et 2. Au lieu de cela, les journaux de chaque nœud sont transmis à un LSTM entraîné lui-même. En fait, des séquences de vecteurs contenant des phrases T et id relatives aux nœuds sont transmises à chaque LSTM entraîné - ce sont les données de test.

Supposons que nous ayons 100 nœuds différents dans l'ensemble de test. Nous aurons alors 100 "paquets" d'entrée pour chaque LSTM (par exemple, s'il s'agit de de taille M, alors nous aurons M vecteurs à 2 états dans chaque "paquet" d'entrée). Cela représente les identifiants des nœuds d'une manière qui permet de réduire les coûts et de se conformer au format du vecteur d'entrée. En substance, l'évaluation de la phase 2 est effectuée sur les nouvelles données de test. Le LSTM prédit l'échantillon suivant pour ces données de test et calcule le MSE. Au fur et à mesure que les phrases d'une séquence sont validées, la prédiction de l'apparition d'une défaillance de nœud est déterminée en essayant de trouver une correspondance proche dans la chaîne de défaillance cible réelle.

Nous utilisons un seuil pour déduire les défaillances de nœuds. En d'autres termes, lorsque le LSTM acquiert MSE<= "seuil", nous pouvons considérer ces résultats comme suspects pour vérifier

l'échec. Le seuil que nous utilisons est 0.0145 et nous le trouvons après une procédure expérimentale. En particulier, une EQM supérieure à 0,0145 sur les données de test produit des chaînes très différentes de celles trouvées dans les chaînes de défaillance entraînées. C'est-à-dire que si nous avons la prédiction [P1, P5, P6] et que la chaîne de défaillance est [P1, P5, P7], elle est considérée comme une défaillance et nous la vérifions avec les données réelles. Nous pouvons ensuite extraire les T du vecteur et calculer le délai d'exécution.

Conclusion

Dans notre travail, la methodologie prise pour la prediction des chaines de defaillances est l'entrainement du modèle skipgram pour obtenir des incorporations de mots. Elle est repartie compose de trois (03) phases.

Chapitre 4 : Expérimentation, processus et résultats

4 Les résultats

4.1 Outils

Pour la mise en œuvre de ce travail, nous avons utilisé Keras, qui fonctionne au-dessus de TensorFlow et d'autres bibliothèques. Keras contient de nombreuses implémentations des blocs de construction couramment utilisés dans les réseaux neuronaux, tels que les couches, les pertes, les fonctions d'activation, les optimiseurs et les métriques. Les représentations graphiques ont été réalisées avec matplotlib. Tout le code du travail a été écrit en python3.

4.2 LogAider

Dans un premier temps, un outil développé par l'Argonne National Laboratory en collaboration avec l'Université de l'Illinois est utilisé pour trouver des corrélations potentielles de logs provenant de HPC (Di et al., 2017). Il est basé sur des logs tels que les logs RAS et JOB rapportés dans la section 2.1. Cet outil, qui a été créé et utilisé pour les logs Mira, peut révéler trois types de corrélation potentielle : **across-field, spatial et temporal**.

La corrélation **across-field** vise à comprendre l'importance des champs et de leurs combinaisons dans la détermination de leurs caractéristiques. Les messages, tels que la gravité du message et sa catégorie. Cette corrélation est liée au calcul de la distribution de probabilité antérieure d'un événement particulier, tel qu'une défaillance du système. **LogAider** peut également révéler des corrélations statistiques (telles que la probabilité de la limite de la distribution de probabilité postérieure) entre différents ensembles de champs ou de métriques combinés. Ainsi, par exemple, **LogAider** peut répondre à la question quelle est la probabilité que le système tombe dans le domaine de la défaillance logicielle, pour un composant particulier (comme le noyau du nœud de contrôle par exemple) et pour un niveau de gravité particulier (comme Fatal). Pour ce type de corrélation, on utilise donc un modèle de probabilité bayésien.

Dans la **spatial correlation**, LogAider peut extraire des corrélations spatiales avec un algorithme de regroupement optimisé sur une topologie de réseau Torus. Pour ce faire, il utilise un algorithme de clustering K-means avec un nombre amélioré d'ensembles de clustering. En particulier, il optimise le nombre de clusters en fusionnant leurs centres proches de manière itérative afin d'obtenir un résultat de clustering optimal.

Quant à la **temporal corrélation**, elle est conçue pour extraire une éventuelle corrélation des événements dans la dimension temporelle. En particulier, l'objectif est de rechercher un ensemble d'événements qui présentent des similitudes très importantes et qui se produisent également très près dans le temps (ou dans une période tolérable spécifiée par l'utilisateur). Cette corrélation est assez proche du comportement humain, car elle explore les corrélations temporelles non seulement sur la base des horodatages, mais aussi sur la similarité de deux événements à plusieurs niveaux, tels que l'ID du message, l'emplacement, la catégorie. La fonction de similarité pour chaque champ est également personnalisable en fonction des besoins. Ainsi, par exemple, la similarité pour le champ messageID, qui comporte 8 chiffres dans la machine Mira (par exemple 00080012), sera dérivée comme suit :

- 0 si les 4 premiers chiffres sont différents.
- 0.5, si les 4 premiers chiffres sont identiques et les 4 derniers différents.
- 1, si les 8 chiffres sont identiques.

Cet outil est écrit en Java et, comme mentionné plus haut, il a été mis en œuvre au-dessus de Mira. Bien que LogAider propose des fichiers destinés à être modifiés par l'utilisateur en fonction du "schéma" de chaque système, il reste un outil très lié à la topologie Mira (topologie torus 5D), ce qui le rend plus difficile à utiliser pour d'autres machines.

Avec une topologie différente, comme par exemple le système Shasta de Cray, qui fonctionne avec la dernière génération d'interconnexion, nom de code "Slingshot" (cray).

De plus, l'exécution de ce programme, même pour des années différentes à partir des journaux de la machine elle-même, nécessite de nombreux changements dans le code. En effet, chaque année semble avoir des champs différents. Tout ceci en fait un bon outil d'un côté, mais avec plusieurs limitations quant à sa facilité d'utilisation, notamment dans des topologies très différentes de celles des systèmes Blue Gene.

4.3 Analyse – Imagerie

Comme mentionné dans la section 1.7, une visualisation et une analyse des données des journaux se rapportant aux travaux Mira ont d'abord été effectuées. Il y a 125 projets, 298 utilisateurs qui ont soumis des travaux, 67000 travaux ont été soumis et environ 71% des travaux ont quitté le système normalement durant la période qu’on a analysé dans ce document.

Tableau 4.1 : Informations de base des journaux de tâches

Basic Information of the job log
Period April 9, 2019 - December 31, 2019
Total number of jobs 67000
Total number of projects 125
Total number of users 298
Percentage of jobs exit system normally 71.004477611940%

Les informations provenant des journaux sont utiles pour l'analyse des données et pour une nouvelle évaluation du programme. Cette analyse aide les administrateurs à connaître l'état des machines et les problèmes d'un système.

Les schémas typiques de cette analyse sont les suivants :

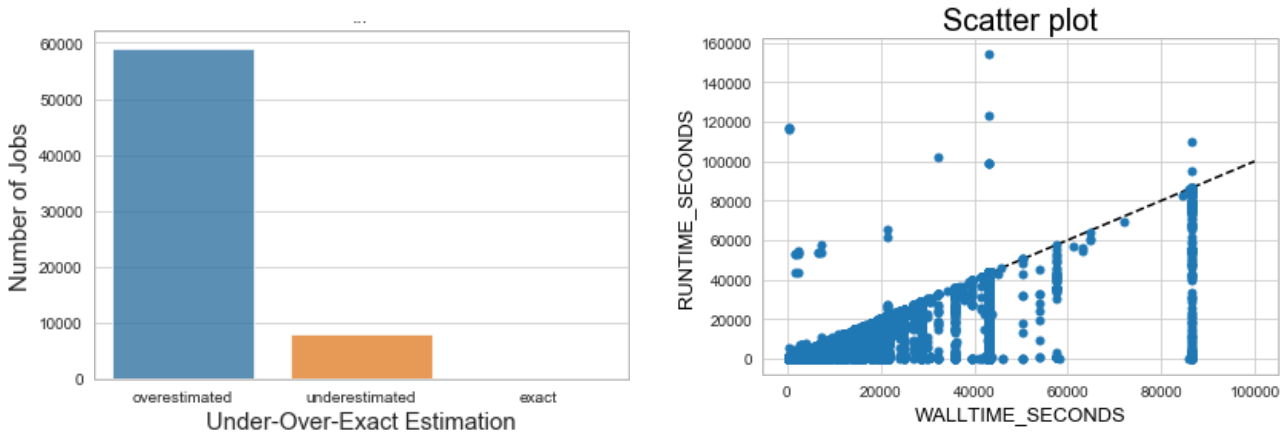


Figure 4.2 : Analyse et evaluation des fichiers journaux

A gauche, nous voyons que la plupart des jobs sont surestimés. Des informations similaires peuvent être obtenues à partir du graphique de droite.

Nous concluons des graphiques ci-dessus que la plupart des jobs ont été surestimés. Dans le diagramme de dispersion, nous avons les secondes de walltime[9] sur l'axe horizontal et les secondes de runtime[10] sur l'axe vertical. On peut remarquer que la plupart des tâches se situent en dessous de la ligne y = x, ce qui signifie qu'elles prennent moins de temps que prévu, tandis que les tâches

[9] Le temps demandé pour le travail peut être différent du temps d'exécution.
[10] L'heure à laquelle le travail a été exécuté.

situées au-dessus de cette ligne prennent plus de temps que prévu. Des Informations complémentaires sont données par le graphique suivant qui représente le nombre de jobs en fonction de statut de sorties regroupés en prenant en considération s'ils ont été surestimés ou non.

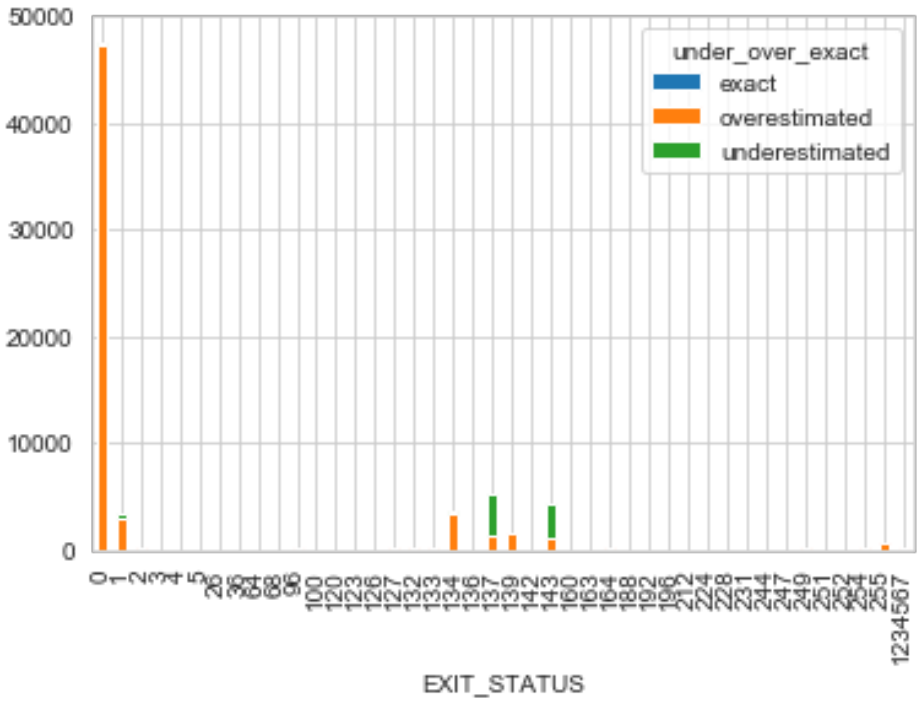


Figure 4.3 : Informations complémentaires du nombre de jobs en fonction de statut de sortie.

La plupart des emplois dont l'état de sortie est égal à 0 - c'est-à-dire ceux qui se sont terminés avec succès - ont été surestimés.

Les informations que nous recevons des files d'attente les plus courantes de la machine[11] (backfill, prod-short, prod-long, prod-capability) sur le nombre de travaux exécutés avec succès (code de sortie 0) sont également intéressantes.

[11] En général, tous les travaux qui ne font pas partie d'une réservation effectuée par un utilisateur sont soumis aux files d'attente. Ces travaux sont ensuite exécutés par un routeur qui applique une politique d'ordonnancement pour les placer dans des files d'attente en fonction de leur taille et du temps demandé.

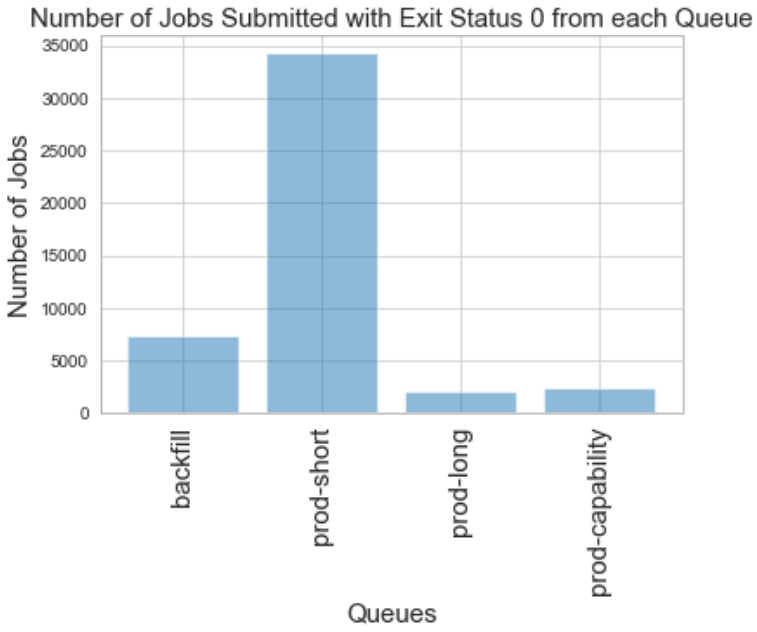


Figure 4.4 : Etats des traveaux soumis en fonction des files d'attente prod-short et de Backfill.

Ce graphique montre le nombre de travaux qui ont été soumis et dont l'état de sortie est 0, en fonction des files d'attente les plus courantes. Comme on peut le voir, la plupart des travaux qui réussissent sont dans la file d'attente prod-short et ensuite dans la file d'attente de Backfill.

Les files d'attente que nous allons mentionner dans le schéma ci-dessus ont l'interprétation suivante :

- **Prod-capability** : toute tâche dont la demande est égale ou supérieure à la capacité de production de l'entreprise. 20% des nœuds, est acheminé vers cette file d'attente, jusqu'à une durée maximale de 24 heures. Cette file d'attente a accès à tous les nœuds de la machine. Idéalement, chaque emploi devrait répondre à ces critères, puisque ces emplois de "capability" constituent la tâche principale d'ALCF.
- **Prod-short** : Tout travail qui requiert moins de 20% des nœuds et qui a demandé un walltime inférieur ou égal à 6 heures est exécuté dans cette file. Cette file d'attente a également accès à tous les nœuds de la machine. Ce ne sont pas des travaux de capacité, mais ils sont relativement courts, de sorte qu'ils bloqueront les travaux les plus importants pendant une période plus courte.

- **Prod-long** : Tout travail qui requiert moins de 20% des nœuds et dont la durée de vie demandée est supérieure à 6 heures est exécuté dans cette file. Cette file d'attente n'a accès qu'à 1/3 des nœuds de l'ensemble du réseau de la machine. Ces travaux sont de petite taille et de longue durée, de sorte qu'ils peuvent bloquer les travaux les plus importants pendant un laps de temps significatif et sont donc très perturbateurs par rapport à notre objectif d'exécuter des travaux de plus grande envergure. Nous garantissons que nous avons au moins une partition 32Ko et toutes les sous-partitions disponibles dans la file d'attente prod-capability (Allcock et al., 2017).
- **Backfill** : d'autres travaux sont autorisés à utiliser les créneaux réservés, pour autant que ces autres travaux ne retardent pas le démarrage d'un autre travail. En fait, l'ordonnancement de remplissage permet d'utiliser les créneaux réservés à de petits travaux qui peuvent être exécutés et terminés avant le début du gros travail. Cela augmente également l'utilisation des ressources. Le backfill fonctionne plus efficacement lorsque tous les travaux d'un cluster ont une limite d'exécution. Les travaux dont la limite d'exécution est petite sont plus susceptibles d'être programmés en tant que travaux de remplissage. Ainsi, comme indiqué ci-dessus, seuls les travaux dont la limite d'exécution expire avant l'heure de début du gros travail sont autorisés à utiliser l'emplacement réservé (IBM).

Le graphe suivant est aussi important puisqu'il représente le pourcentage de travaux en fonction des statuts d'erreur correspondants.

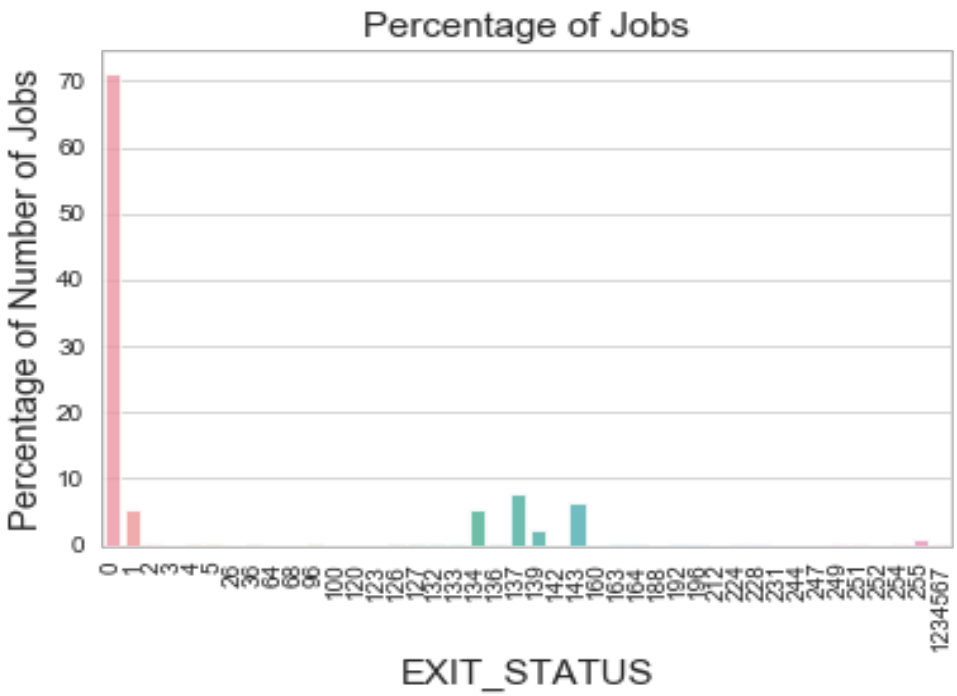


Figure 4.5 : Graphe Etats des traveaux soumis.

Ce graphique montre que 70% des travaux soumis ont un code de sortie de 0, ce qui signifie qu'ils ont été exécutés avec succès. Les 30% restants n'aboutissent pas, un chiffre toutefois très significatif si l'on considère qu'un éventuel redémarrage par exemple est pénible pour les exigences du poste et le fonctionnement de la machine dans son ensemble.

Des conclusions intéressantes peuvent être tirées des deux graphiques suivants.

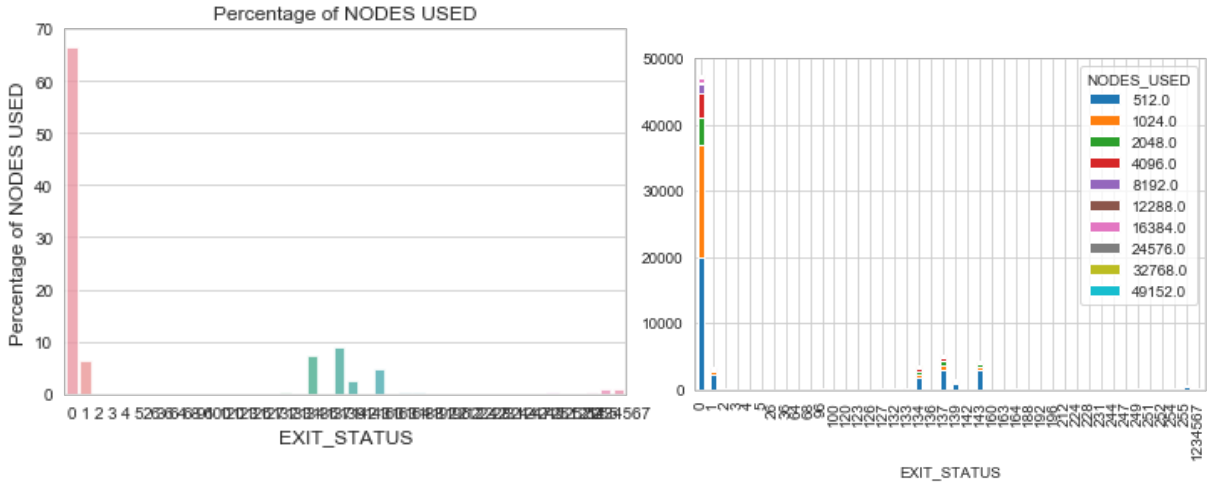


Figure 4.6 : Graphe représente le pourcentage du total des nœuds utilisés d'état de sortie et graphe du rapport entre le nombre de nœuds utilisés dans les tâches et leurs états de sortie.

Le graphique de gauche représente le pourcentage du total des nœuds utilisés en termes d'état de sortie. Le graphique de droite représente le rapport entre le nombre de nœuds utilisés dans les tâches et leurs états de sortie respectifs

Comme on peut le constater, moins de 70% du total des nœuds utilisés correspondent à des travaux dont le code de sortie est 0, tandis que plus de 30% des nœuds utilisés correspondent à des travaux qui se sont terminés sans succès. En outre, le rapport entre les paquets de nœuds utilisés et leurs états de sortie correspondants est également indiqué. Dans tous ces cas, il apparaît que le nœud 512 "packet" est le plus utilisé.

En outre, les informations obtenues sur les codes de sortie des travaux dans les files d'attente mentionnées précédemment sont également importantes, comme le montre la figure 4.6.

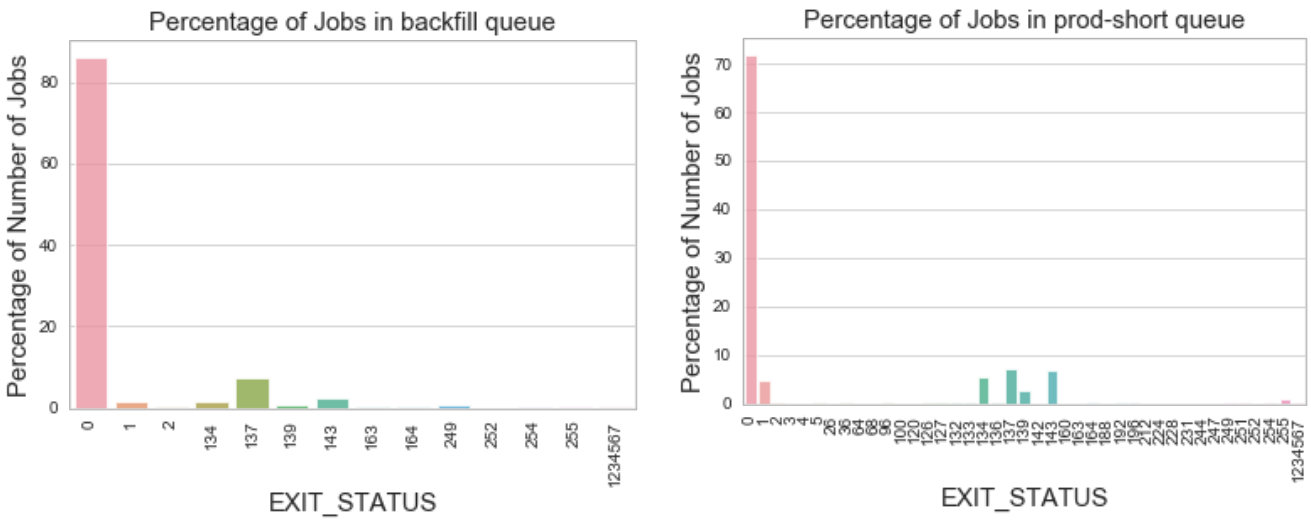


Figure 4.7 : Graphe de pourcentage de travaux avec l'état de sortie correspondant à backfill et prod-short.

A gauche, le pourcentage de travaux avec l'état de sortie correspondant dans la file d'attente de backfill, et à droite pour la file prod-short.

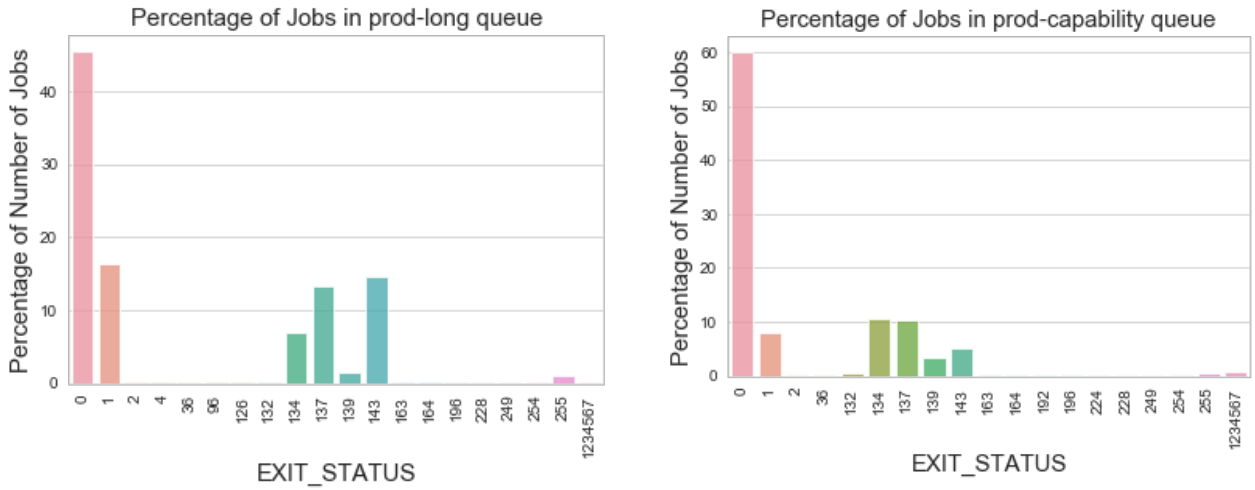


Figure 4.8 : Graphe de pourcentage de travaux avec l'état de sortie correspondant à prod-long et prod-capacity.

A gauche, le pourcentage de travaux avec l'état de sortie correspondant dans la file d'attente prod-long, et à droite pour la file d'attente prod-capability.

Comme le montrent les figures 4.7 et 4.8, si la majorité des travaux s'exécutent avec succès, avec l'état de sortie 0, un pourcentage significatif d'entre eux ne s'exécutent pas avec succès. On observe également une différence dans les pourcentages selon le type de file d'attente, par exemple dans la file prod-long, plus de 10 % des travaux ont l'état de sortie 143[12]. En outre, plus de 10 % des travaux dans la même file d'attente ont un statut de sortie de 137[13]. Dans la même file d'attente.

L'état de sortie 134[14] occupe également un pourcentage significatif. Enfin, dans la file d'attente prod-long, nous voyons que l'état de sortie 1 occupe également un pourcentage significatif, ce qui indique des erreurs de type général causées par les utilisateurs. On peut donc conclure que dans cette file d'attente, plus de 50% des travaux soumis ne se terminent pas avec succès.

Des conclusions similaires sont tirées des trois autres queues, avec des taux d'échec plus élevés dans la file d'attente de la capacité de production (environ 40%) et des taux plus faibles dans les files d'attente de la capacité de production (moins de 30%) et du remplissage (moins de 20%).

D'après les diagrammes ci-dessus, on peut donc voir que les tâches dans la file d'attente de production et de remplissage ont une probabilité plus élevée de se terminer de manière harmonieuse. Seulement 70 % des travaux soumis à la machine Mira au cours de la période étudiés (9 avril-31 décembre 2019) se terminent avec succès, les 30% restants n'y arrive pas. Cela crée plusieurs problèmes, tant dans la gestion des ressources fournies aux utilisateurs que dans

[12] Ce statut de sortie envoie le signal Sigterm qui est utilisé pour demander à un processus de se terminer. Avec ce signal, la fin de l'opération est plus facile à réaliser. Lorsque le processus reçoit le signal, il peut soit s'arrêter immédiatement, soit s'arrêter après un court délai après que les ressources ont été libérées, ou il peut fonctionner indéfiniment.

[13] Avec le code de sortie 137, le signal SIGKILL est envoyé. Le signal SIGKILL est envoyé à un processus pour qu'il se termine immédiatement. Ce signal ne peut être ignoré et le processus récepteur ne peut effectuer aucun nettoyage à la réception de ce signal, en fait le processus n'est même pas conscient du signal SIGKILL, puisqu'il va directement au noyau. n Cependant, le noyau peut ne pas être en mesure de détruire avec succès le processus dans certains cas. Par exemple, si le processus attend des disques réseau ou des disques d'E/S, le noyau peut ne pas être en mesure de l'arrêter. De même, les processus zombies et ceux qui sont dans un sommeil ininterrompu ne peuvent pas être arrêtés. Un redémarrage est nécessaire pour effacer ces processus du système.

[14] Au 134, nous recevons le signal SIGABRT, qui indique le signal de fin du processus de core dump. Ce message est un signal de terminaison sévère (dû à une assertion d'abandon ou à une double absence de mémoire). Ces erreurs appartiennent à la catégorie des bogues.

l'identification et la résolution rapide des problèmes que ces défaillances créent. Bien que certaines défaillances soient évidentes, la plupart sont difficiles à identifier.

Avec la complexité croissante des systèmes à haute performance, les défaillances des systèmes HPC à grande échelle augmentent encore plus, avec pour conséquence que le MTBF (Mean Time Between Failures) diminue également, ce qui entraîne un gaspillage important de puissance informatique.

Tout cela nous amène à rechercher les moyens de prédire une anomalie du système, à quel endroit du système elle se produira (à quel nœud) et dans combien de temps (lead time). Pour atteindre ces objectifs, les techniques d'exploration de données les plus courantes jusqu'à présent, telles que les machines à vecteurs de support (SVM) (Fulp et al., 2008) ou l'analyse en composantes principales (PCA) (Lan et al., 2009), ne sont pas suffisamment évolutives pour des données de cette taille et en temps réel.

Le traitement du langage à partir de journaux non structurés générés par des systèmes informatiques met en évidence deux problèmes, comme mentionné dans la section 1.6. Premièrement, que les journaux ne sont pas étiquetés et deuxièmement, que pour déduire des modèles complexes, les données doivent être traitées et alimentées par une représentation d'entrée appropriée. C'est là que le développement rapide de l'apprentissage profond dans la compréhension du langage naturel nous aide.

Sur la base de ce qui précède, nous sommes amenés à développer une méthodologie pour prédire les chaînes de messages menant à l'échec et en même temps extraire le délai d'exécution avec la capacité d'identifier le nœud où l'échec du travail s'est produit. En fait, les données que nous utilisons sont réelles (Argonne) et publiques, sans ajout supplémentaire "factice" de défaillance pour les anomalies du système.

4.4 Prévision des défaillances

Pour mettre en œuvre la prédiction des défaillances de nœuds, nous avons utilisé les journaux système réels du superordinateur IBM Mira pour la période allant d'avril 2019 au 31 décembre 2019. Plus précisément, nous avons utilisé les journaux RAS et JOB, comme indiqué dans la section 2.1 Nous avons fusionné ces journaux avec une clé commune des informations partagées des blocs (qui se présente sous la forme par exemple de MIR-40000-737F1-4096) et

conservé les informations des événements temporellement proches. En outre, nous avons conservé les événements qui ont lieu dans le même bloc et dont l'horodatage de début est inférieur à l'heure de l'événement et, à son tour, l'heure de l'événement est inférieure à l'horodatage de fin.[15]

StartTimestamp < EventTimestamp < EndTimestamp

Cette fusion a pris beaucoup de temps car les logs étaient trop nombreux, en particulier le log RAS qui était de l'ordre de 5 Go et le log JOB de 33 Mo.

Avec la méthode que nous avons suivie, nous avons un délai d'environ 10 secondes à 1 heure, avec un taux de rappel de pas moins de 81%. Nous avons divisé notre ensemble de données en deux parties : formation et test, où 30% des données sont utilisées pour la formation et le reste pour le test. Notre évaluation expérimentale quantifie l'efficacité de notre méthodologie en termes de mesures de performance standard.

Initialement, pour la formation de la phase 1, les encastrements de mots ont été exportés. Avec l'aide du modèle de skip-gram.

	0	1	2	3	4	...	95	96	97	98	99
ndcorrectableerror	-0.005538	2.680929	24.234669	-8.306913	-24.042770	...	31.209141	-36.268063	-1.395723	1.290433	-7.899679
warningmc0rank1	-0.179043	0.370571	-7.521682	-7.401166	-11.498628	...	11.767866	-7.519443	1.518415	-0.117205	1.809196
serviceaction6585restartednodedcar13	0.016986	0.009768	-16.279041	-9.973520	-11.286119	...	11.680753	-5.424914	2.927208	-0.769646	4.300936
kernelinternalassertionfailure	-6.977578	-0.200877	17.313423	20.057041	-18.864708	...	7.534953	-6.133167	-2.827714	3.030792	3.924452
abqlbiterrorthresholdwasexceeded	0.219328	-3.853128	-11.354437	-1.463878	5.039874	...	-6.411012	8.927105	3.251094	-1.470589	3.760504
alinkchipdidnotalignalongtheaportonswitch2	-0.045018	-0.053850	-4.676926	-2.973093	-3.529989	...	3.492841	-1.598395	0.854124	-0.205243	1.386150
cfamrecoverableerror	0.125706	-1.365464	-3.360045	0.067303	2.761106	...	-3.305687	4.010753	0.965712	-0.484375	1.109738
ndsenderretransmissioncorrectableerror	-0.109295	-0.202403	-15.544802	-7.710242	-7.962912	...	7.159219	-1.393445	2.459007	-0.632582	4.615002
checkforsuccessofporsequencefailed	-0.046419	0.726082	9.227051	-0.840424	-5.163361	...	7.074319	-9.179214	-0.826632	0.494978	-2.958648
alinkchipdidnotbitalignalongthereceivercport	-0.079056	-0.415448	-10.627165	-4.518475	-3.853862	...	3.220978	0.370925	1.666339	-0.529350	3.263497

[10 rows x 100 columns]

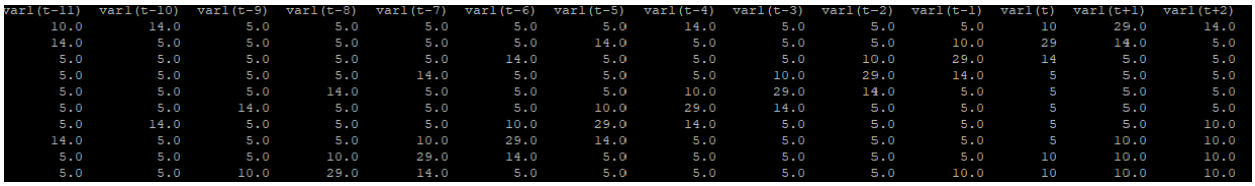
Figure 4.9 : La figure ci-dessus montre certains des mots incorporés de 10 phrases de notre jeu de données.

Ensuite, pour l'entraînement à l'aide de LSTM, nous avons converti notre ensemble de données codées dans un format adapté à l'entraînement de nos séries temporelles.

Techniquement, dans la terminologie des prévisions de séries chronologiques, le moment actuel (t) et les moments futurs (t+1, t+n) sont des moments de prévision et les observations précédentes (t-

[15] L'horodatage du début correspond à l'heure du début d'une tâche, tandis que l'horodatage de fin correspond à l'heure de la fin de la tâche. L'horodatage de l'événement est l'heure à laquelle un événement est enregistré dans les journaux RAS pour ce travail.

1, t-n) sont utilisées pour faire les prévisions. Cela permet non seulement la prédiction classique X=>y, mais aussi X=>Y, où l'entrée et la sortie peuvent être des séquences. En outre, cela fonctionne également dans les problèmes de séries temporelles multivariées (phase 2), où, au lieu d'avoir un seul ensemble d'observations pour une série temporelle, nous en avons plusieurs (par exemple, ids de phrase et T6).



var1(t-11)	var1(t-10)	var1(t-9)	var1(t-8)	var1(t-7)	var1(t-6)	var1(t-5)	var1(t-4)	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)	var1(t+1)	var1(t+2)
10.0	14.0	5.0	5.0	5.0	5.0	5.0	14.0	5.0	5.0	5.0	10	29.0	14.0
14.0	5.0	5.0	5.0	5.0	5.0	14.0	5.0	5.0	5.0	10.0	29	14.0	5.0
5.0	5.0	5.0	5.0	5.0	14.0	5.0	5.0	5.0	10.0	29.0	14	5.0	5.0
5.0	5.0	5.0	5.0	14.0	5.0	5.0	5.0	10.0	29.0	14.0	5	5.0	5.0
5.0	5.0	5.0	14.0	5.0	5.0	5.0	10.0	29.0	14.0	5.0	5	5.0	5.0
5.0	5.0	14.0	5.0	5.0	5.0	10.0	29.0	14.0	5.0	5.0	5	5.0	5.0
5.0	14.0	5.0	5.0	5.0	10.0	29.0	14.0	5.0	5.0	5.0	5	5.0	10.0
14.0	5.0	5.0	5.0	10.0	29.0	14.0	5.0	5.0	5.0	5.0	5	10.0	10.0
5.0	5.0	5.0	10.0	29.0	14.0	5.0	5.0	5.0	5.0	5.0	10	10.0	10.0
5.0	5.0	10.0	29.0	14.0	5.0	5.0	5.0	5.0	5.0	10.0	10	10.0	10.0

Figure 4.10 : On voit les colonnes des observations précédentes (t-1,...,t-n) et les colonnes des observations prédites (t,...,t+n) pour un ensemble de données constitué de séries temporelles, dans une forme d'apprentissage supervisé.

Cette fonction nous permet de créer de nouvelles trames de problèmes de séries temporelles étant donné la longueur souhaitée des séquences d'entrée et de sortie. Il s'agit d'un outil important car il nous permet d'explorer différentes trames d'un problème de série temporelle avec des algorithmes d'apprentissage automatique pour voir lesquelles pourraient conduire à des modèles plus performants.

En suivant ce qui précède, nous entraînons ensuite les modèles avec LSTM. Tout d'abord, dans la phase 1, nous pré-entraînons, comme nous l'avons mentionné dans les sections précédentes, les données sources, qui contiennent tout le vocabulaire de l'ensemble d'entraînement, avant l'étiquetage des phrases. L'entraînement durant cette phase prend environ 110 heures et nous obtenons une précision d'environ 76%.

Elle est suivie de la phase 2, qui est essentiellement la deuxième partie de l'apprentissage par transfert séquentiel, comme décrit dans la section 2.8.1. Au-delà, le modèle pré-entraîné de la phase 1 est utilisé comme point de départ pour notre modèle cible, afin d'obtenir une convergence plus rapide et optimale. L'entraînement dans cette phase prend environ 23,72 minutes et nous obtenons une précision d'environ 83%.

Le temps que prend chaque formation montre à quel point le Transfer learning est bénéfique pour nos résultats. En particulier, il y a une énorme différence entre la phase 1, qui dure environ 4,5 jours, et la phase 2, qui ne dure que quelques minutes.

Les courbes de perte et de précision de la méthodologie que nous avons validée pour la phase 1 et la phase 2 sont présentées ci-dessous.

- **Phase 1**

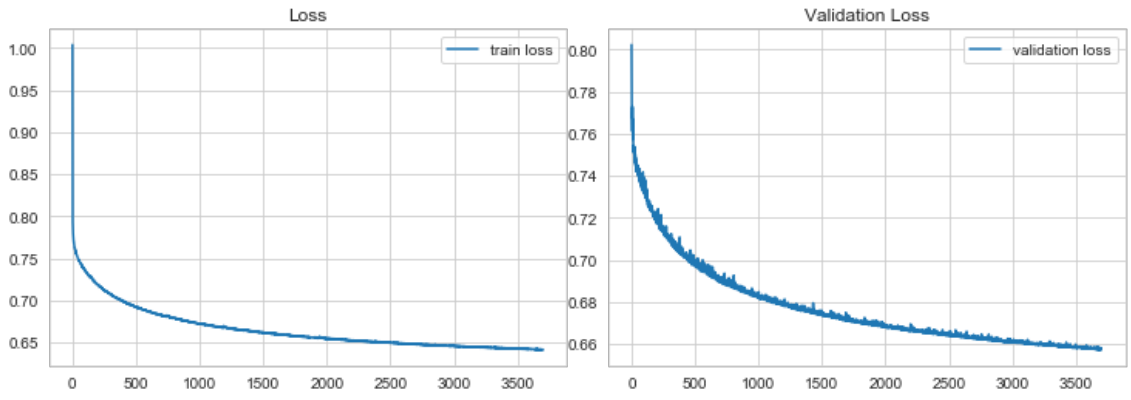


Figure 4.11 : A gauche, le graphique des pertes, à droite, le graphique des *validation loss*.

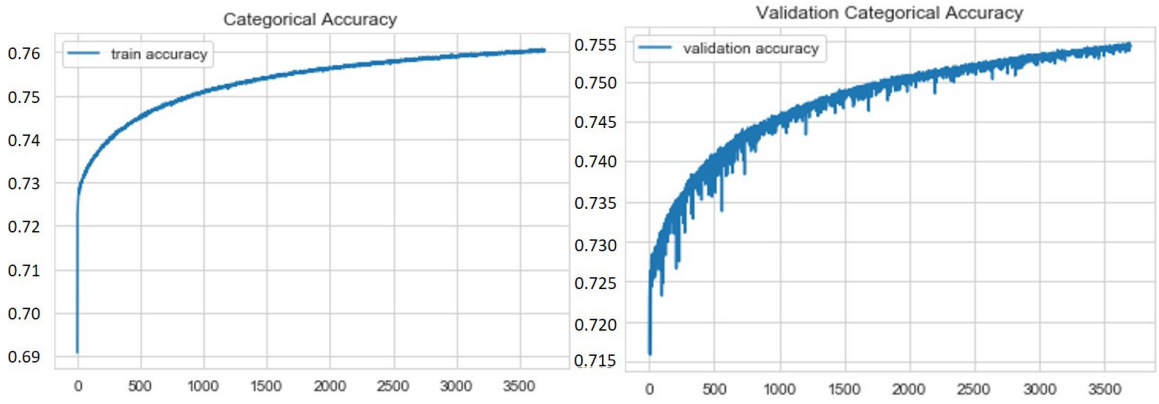


Figure 4.12 *:* A gauche, le graphique de précision, à droite, le graphique de validation accuracy.

- **Phase 2**

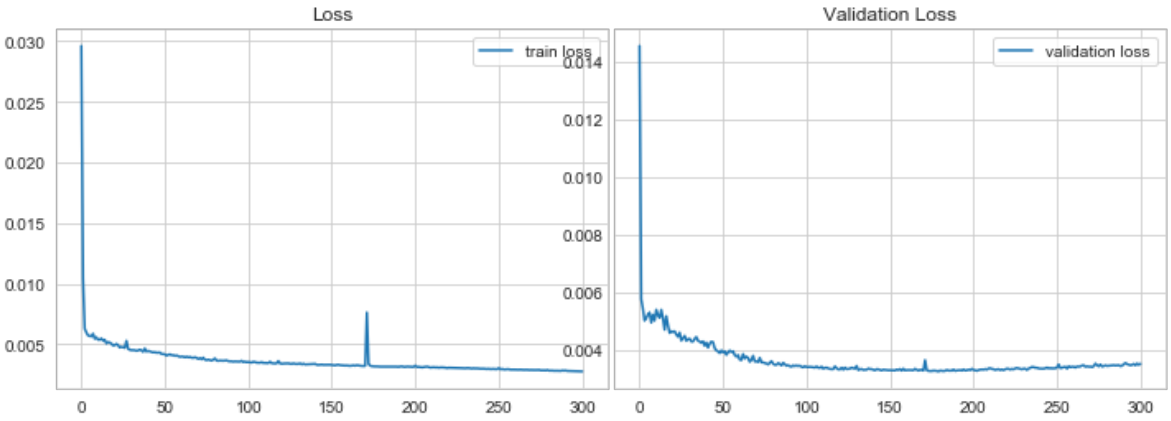


Figure 4.13 : A gauche, le graphique des pertes, à droite, le graphique des *validation loss*.

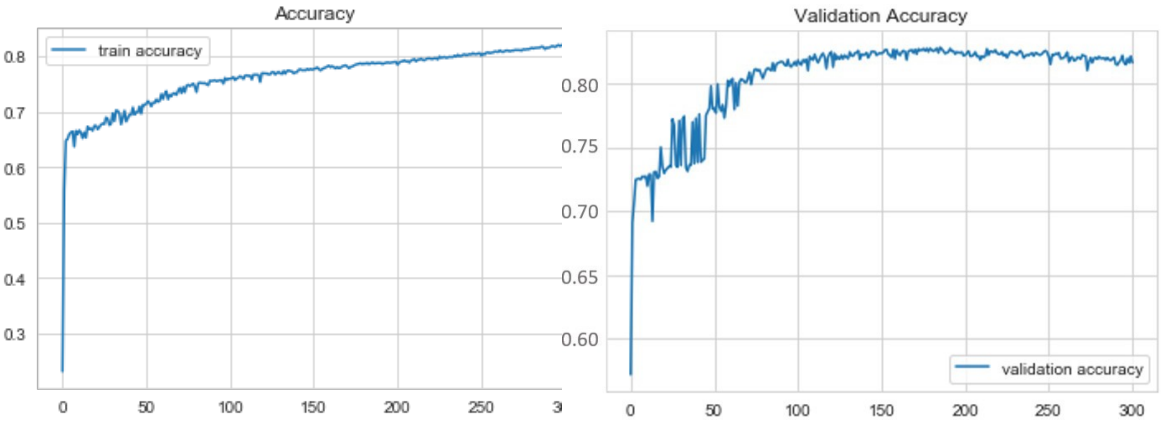


Figure 4.14 : A gauche, le graphique de précision, à droite, le graphique de *validation accuracy*.

La performance de notre méthodologie est due en grande partie à la taille de la fenêtre historique 8 que nous avons choisie. En particulier, les résultats ci-dessus sont obtenus avec une taille d'historique de 15 et 10 pour les phases 1 et 2 respectivement. Une taille d'historique plus importante améliore la précision, mais consomme beaucoup plus de temps. En réduisant la taille

de l'historique[16] à pour la phase 1 et à 5 pour la phase 2, les résultats ne sont pas aussi bons qu'avec les tailles d'historique précédentes la précision est de 10 %.

En outre, avec la précision de l'efficacité de notre méthodologie, nous calculons des métriques appropriées qui, comme mentionné dans la section 2.9.2, sont très importantes pour l'évaluation de notre méthodologie globale. Les métriques prédictives sont : *recalls, weighted accuracy, precisions, F1* score ainsi que le *taux de FP*, le *taux de FN*. Ces métriques sont calculées dans la phase 3 et expriment le pouvoir prédictif de la méthode pour les chaînes de défaillance possibles.

Pour des historiques de taille 15 et 10 dans les phases 1 et 2, nous avons un total pour notre méthodologie de 81% de rappel, 77% d'exactitude pondérée, 78% de précision, 79% de score F1. Nous pouvons donc voir que nous prédisons le plus grand pourcentage de chaînes de défaillance dans les données de test avec un rappel très significatif et nous avons un pourcentage de précision assez élevé.

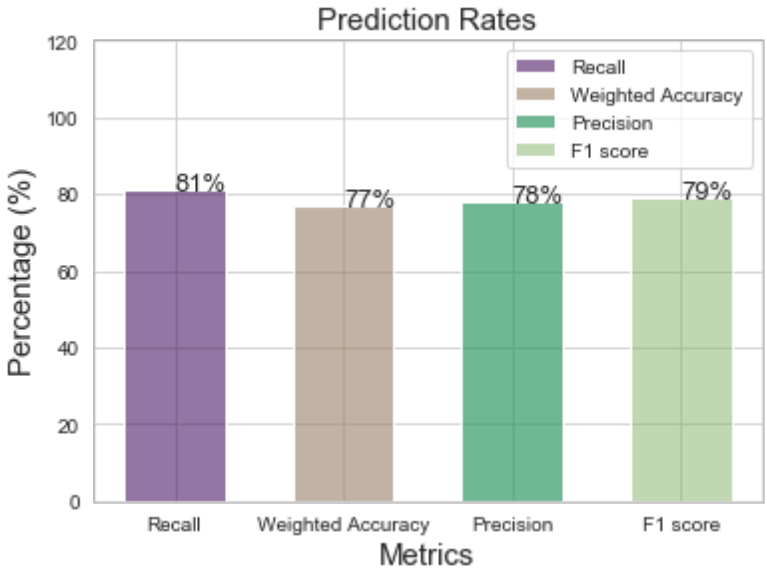


Figure 4.15 : La figure ci-dessus montre les taux de prédiction obtenus par notre méthodologie.

[16] La taille de l'historique est la taille de la fenêtre des échantillons fournis pendant la formation sur laquelle les prédictions sont faites.

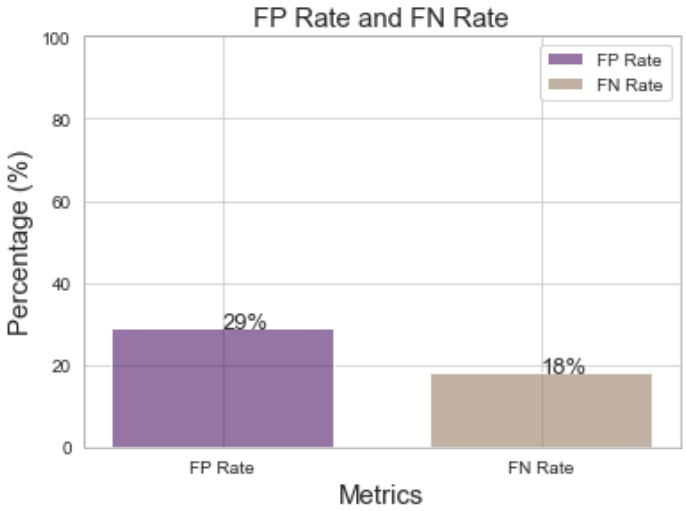


Figure 4.16 : Cette figure montre les taux de FP et FN

Analyse des données de test : 71,43% des phrases dans toutes les données de test sont des phrases liées à FC. Il est important de noter que ce pourcentage n'indique pas à lui seul le rappel ou la précision attendus, puisque les phrases liées à la FC qui font partie des chaînes de défaillance et qui se trouvent dans des séquences très différentes de celles-ci (par exemple, ne se terminent pas par un message terminal indiquant une défaillance) ne sont pas des défaillances réelles, mais présentent certaines similitudes. De nombreux nœuds reçoivent des messages similaires au cours de la journée, mais ils ne tombent pas tous en panne. Cette estimation est simplement un indicateur de chevauchement des phrases courantes qui conduisent à l'échec par rapport à celles qui ne conduisent pas à l'échec.

En outre, comme mentionné dans la section 3.2, un seuil de MSE est utilisé dans la phase 3, en dessous duquel nous pouvons rechercher des chaînes de défaillance possibles (MSE<=0.0145), tandis qu'au-dessus duquel il est moins probable de trouver une chaîne de défaillance. Le seuil est calculé expérimentalement.

Conclusion

Enfin, ce chapitre d'expérimentation a été la partie la plus intéressante de notre étude de recherche, car nous avons présenté les différents outils utilisés, et les résultats obtenus à partir de notre modèle. En outre, nous sommes même allés jusqu'à mesurer la performance de notre modèle. Alors maintenant, nous passons à la conclusion générale

Conclusion Générale

Tout d'abord, en ce qui concerne la visualisation des journaux des travaux que nous avons effectués, nous remarquons que :

- La plupart des travaux soumis au cours de la période où nous étudions les logs de la machine ont été surestimés.
- La plupart des travaux dont l'état de sortie est 0, c'est-à-dire ceux qui ont été interrompus, ne sont pas pris en compte. Avec succès, ont été surestimés.
- La plupart des travaux qui se sont terminés avec succès (code de sortie 0) se trouvent dans la file d'attente prod-short, puis dans la file d'attente backfill.
- Dans les files d'attente prod-long et prod-capability, la majorité des travaux ne se terminent pas avec succès (respectivement 50% et 40% des travaux non réussis).
- Plus de 30 % des offres d'emploi soumises n'aboutissent pas. Ils ont un pourcentage assez élevé, cependant, pour la difficulté du bon fonctionnement de la machine dans son ensemble, ainsi que pour le temps nécessaire à la réalisation des travaux. Il est donc très important de prévoir les défaillances du système, ce qui permet de prévoir les défaillances du travail.

Une approche d'apprentissage profond en trois phases est utilisée pour s'entraîner d'abord à prédire la phrase suivante, puis pour se réentraîner afin d'identifier les chaînes de journaux menant à la défaillance, augmentées des délais d'exécution prédits, et enfin pour prédire les chaînes de défaillance et donc les délais d'exécution pendant les tests. Il est ainsi possible de prévoir où la défaillance se produira et en combien de minutes/secondes.

D'après les résultats de la section précédente 4, nous voyons que :

- Nous avons un rappel assez élevé - 81%, mais aussi une précision - 78%. Ainsi, sur l'ensemble des chaînes de défaillance qui existent réellement dans les données de test, on détecte un pourcentage assez élevé d'entre elles (rappel). De plus, sur le total des chaînes d'échec qu'on a prédit, le plus grand pourcentage d'entre elles sont effectivement des chaînes d'échec (précision).

- Le score élevé de F1 - 79%, qui évalue la précision de notre méthodologie pour prédire l'échec, en le considérant comme la moyenne pondérée du rappel et de la précision, est un autre indicateur de la qualité de cette méthodologie.
- Ce qui précède implique que les informations obtenues à partir des chaînes de défaillance entraînées (phase 2) ont permis de faire des prédictions précises pour l'ensemble des données d'essai.
- Il apparaît que l'apprentissage par transfert utilisé dans la phase 2 est assez efficace à la fois en termes de performance de la méthode ainsi que de temps, et donc en termes d'utilisation des ressources.
- L'utilisation d'une taille d'historique plus petite diminue la précision, tandis qu'une taille plus grande l'augmente, mais au prix d'un temps d'apprentissage plus long. Dans ce travail, les expériences ont montré que les tailles d'historique appropriées sont de 15 pour la phase 1 et de 10 pour la phase 2.
- Avec la méthode que nous avons suivie, nous avons un délai d'environ 10 secondes à 1 heure.

Les chaînes de défaillance sont donc suffisamment fiables pour maintenir le pouvoir prédictif de notre modèle pour les événements rencontrés lors des tests.

En outre, après avoir étudié de nombreuses chaînes différentes menant à des nœuds défaillants, les phrases dominantes menant à l'arrêt des nœuds non défaillants sont déterminées. Ces phrases appartiennent à certaines classes qui sont énumérées ci-dessous :

- Les MCE font référence à des exceptions de contrôle matériel de la machine qui sont très courantes.
- FileSystem (FS) (access alert).
- Hardware (H/W) qui fait référence aux hardware errors.
- Panique concernant les messages de panique du noyau.

Les délais varient en fonction de la classe de la défaillance. Les MCE et les Panic failures ont des délais comparativement beaucoup plus élevés que les autres, environ 1 heure. Enfin, il est également important de mentionner que le plus grand pourcentage de phrases conduisant à l'échec appartient à la classe MCE et Hardware.

Travaux futurs

Dans le contexte de la prédiction des défaillances, une prochaine étape que nous aimerions franchir est d'appliquer la méthodologie à d'autres années de fonctionnement de la machine Mira. Dans ce travail, comme mentionné ci-dessus, les journaux au cours de la période du 9 avril 2019 au 31 décembre 2019 ont été utilisés. Les résultats obtenus au cours des autres années d'exploitation seraient donc également importants.

Il serait également très intéressant de l'appliquer aux journaux d'autres HPC car la méthode est générique. Cela signifie qu'il peut certainement être adapté à d'autres systèmes de production à grande échelle avec une journalisation différente par une certaine personnalisation. Cette méthode est essentiellement non supervisée.

Dans ce travail, des techniques d'apprentissage profond sont utilisées pour permettre la détection des anomalies et la prédiction des pannes dans Mira. Des efforts similaires ont été faits par d'autres, comme (Fu et Xu, 2007), (Berrocal et al., 2014) et (Du et al., 2017), qui utilisent des techniques telles que le regroupement, le SVM, l'ACP, les réseaux neuronaux, etc.

Cependant, ces efforts qui révèlent la "richesse" des informations issues des travaux basés sur l'apprentissage automatique ne porteront vraiment leurs fruits que lorsque nous pourrions construire de véritables cadres pour la prédiction des échecs en ligne. La méthode que nous suivons pourrait être applicable aux approches en temps réel qui nécessitent une détection rapide des anomalies en ligne.

En conclusion, il est clair que ce travail a plusieurs possibilités de développement dans le futur. Les résultats ainsi que les étapes de la méthode motivent son extension à la détection et au débogage des défaillances dans différents systèmes de supercalculateurs, et favorise le développement d'un outil en ligne pour la détection des anomalies.

Bibliographie

W. Allcock, P. Rich, Y. Fan, and Z. Lan. Experience and practice of batch scheduling on leadership supercomputers at argonne. In Workshop on Job Scheduling Strategies for Parallel Processing, pages 1–24. Springer, 2017.

Argonne. This data was generated from resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. https://reports.alcf.anl.gov/data/ANL-ALCF-AUTOPERF-MIRA_20180101_20181231.html.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2):157–166, 1994.

E. Berrocal, L. Yu, S. Wallace, M. E. Papka, and Z. Lan. Exploring void search for fault detection on extreme scale systems. In 2014 IEEE International Conference on Cluster Computing (CLUSTER), pages 1–9. IEEE, 2014.

F. Cappello, G. Al, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward exascale resilience: 2014 update. Supercomputing Frontiers and Innovations: an International Journal, 1(1):5–28, 2014.

cray. ARCHITECTURAL INNOVATION FOR THE EXASCALE ERA. https://www.cray.com/resources/Architectural-innovation-for-exascale-era.

C. Darken, J. Chang, and J. Moody. Learning rate schedules for faster stochastic gradient search. In Neural Networks for Signal Processing [1992] II., Proceedings of the 1992 IEEE-SP Workshop, pages 3–12. IEEE, 1992.

A. Das, F. Mueller, C. Siegel, and A. Vishnu. Desh: deep learning for system health

prediction of lead times to failure in hpc. In Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, pages 40–51,

2018.

S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello. Logaider: A tool for mining potential correlations of hpc log events. In 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pages 442–451. IEEE, 2017.

S. Di, H. Guo, R. Gupta, E. R. Pershey, M. Snir, and F. Cappello. Exploring properties and correlations of fatal events in a large-scale hpc system. IEEE Transactions on Parallel and Distributed Systems, 30(2):361–374, 2018.

J. D. DOD, B. Harrod, T. Hoang, L. Nowell, B. Adolf, S. Borkar, N. DeBardeleben, M. Heroux, D. Rogers, V. Sarkar, et al. Inter-agency workshop on hpc resilience at extreme scale. 2012.

K. Du, F. Li, G. Zheng, and V. Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 1285–1298, 2017.

E. Elnozahy, R. Bianchini, T. El-Ghazawi, A. Fox, F. Godfrey, A. Hoisie, K. McKinley, R. Melhem, J. Plank, P. Ranganathan, et al. System resilience at extreme scale. Defense Advanced Research Project Agency (DARPA), Tech. Rep, 2008.

F. Fu and C.-Z. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In Proceedings of the 2007 ACM/IEEE conference on Supercomputing, pages 1–12, 2007.

E. W. Fulp, G. A. Fink, and J. N. Haack. Predicting computer system failures using support vector machines. WASL, 8:5–5, 2008.

A. Gainaru, F. Cappello, M. Snir, and W. Kramer. Fault prediction under the microscope: A closer look into hpc systems. In SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pages 1–11. IEEE, 2012.

R. Gandhi. A Look at Gradient Descent and RMSprop Optimizers. https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b.

S. Gupta, D. Tiwari, C. Jantzi, J. Rogers, and D. Maxwell. Understanding and exploiting spatial properties of system failures on extreme-scale hpc systems. In 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pages 37–44. IEEE, 2015.

S. Gupta, T. Patel, C. Engelmann, and D. Tiwari. Failures in large scale systems: long-term measurement, analysis, and implications. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–12, 2017.

C. HANSEN. Activation Functions Explained - GELU, SELU, ELU, ReLU and more. https://mlfromscratch.com/activation-functions-explained/#elu.

L. Harasim. Learning theory and online technologies. Routledge, 2012.

S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. Diploma, Technische Universität München, 91(1), 1991.

S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

IBM. Backfill scheduling. https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_admin/backfill.html.

imdb. Star Wars. https://www.imdb.com/title/tt0076759/?ref_=ttmi_tt.

S. J Russell and P. Norvig. Artificial Intelligence: A Modern Approach, chapter Intelligent agents, pages 31–52. Prentice Hall, 01 1995. ISBN 0137903952.

KERAS. Keras Documention. https://keras.io/.

G. Lakner, B. Knudson, et al. IBM system Blue Gene solution: Blue Gene/Q system administration. IBM Redbooks, 2013.

Z. Lan, Z. Zheng, and Y. Li. Toward automated anomaly identification in large-scale systems. IEEE Transactions on Parallel and Distributed Systems, 21(2):174–187, 2009.

R. Lucas, J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, J. Dongarra, et al. Doe advanced scientific computing advisory subcommittee (ascac) report: top ten exascale research challenges. Technical report, USDOE Office of Science (SC)(United States), 2014.

S. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. AI magazine, 27(4):12, 2006.

P. McCorduck. Machines Who Think (2nd Ed.). A. K. Peters, 2004.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013a.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 3111–3119, 2013b.

M. L. Minsky. Computation: finite and infinite machines. Prentice-Hall, Inc., 1967.

”Mira”. Mira (supercomputer). https://en.wikipedia.org/wiki/Mira_(supercomputer).

T. M. Mitchell. The discipline of machine learning, volume 9. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

C. Olah. Understanding lstm networks. 2015.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. the Journal of machine Learning research, 12:2825–2830, 2011.

G. Peters, S. Ruder, and N. A. Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. arXiv preprint arXiv:1903.05987, 2019.

D. M. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

N. Qian. On the momentum term in gradient descent learning algorithms. Neural networks, 12(1):145–151, 1999.

W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. Neural computation, 29(9):2352–2449, 2017.

H. Robbins and S. Monro. A stochastic approximation method. In Herbert Robbins Selected Papers, pages 102–109. Springer, 1985.

S. Ruder. An overview of gradient descent optimization algorithms. http://ruder.io/optimizing-gradient-descent/index.html#stochasticgradientdescent, 2016.

S. Ruder. Neural transfer learning for natural language processing. PhD thesis, NUI Galway, 2019.

D. D. Sarkar. A hands-on intuitive approach to Deep Learning Methods for Text Data — Word2Vec, GloVe and FastText. https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c44370bbfa

A. Sharma. Understanding Activation Functions in Neural Networks. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0, 2017.

S. V. Stehman. Selecting and interpreting measures of thematic classification accuracy. Remote sensing of Environment, 62(1):77–89, 1997.

Top500. Top500. https://www.top500.org/.

A. S. Walia. Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent. https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f, 2017.