Rapport du travail de bachelor

Version 1.0

Sommer Nicolas

22 July 2017

Table des matières

CHAPITRE 1

Introduction

Ce rapport présente un projet développé dans le cadre du Travail de Bachelor, au sein de la HES de Neuchâtel et en collaboration avec le Human Brain Project et le Laboratoire de Recherche en Neuro-imagerie (LREN) du CHUV de Lausanne. Il a été développé par Monsieur Nicolas Sommer sous la supervision de Monsieur Fabrizio Albertetti. Ce travail étant un mandat de la Medical Informatics Platform du Human Brain Project, la personne de contact et mandant était Arnaud Jutzeler.

Ce travail était d'une durée de 450h et c'est déroulé sur 10 semaines durant le semestre de printemps 2017 à raison de 4h par semaine puis sur une durée de 8 semaines a raison de 8h par jour.

Les premières semaines furent donc consacré à l'analyse préliminaire du projet, aux choix et à l'apprentissage des technologies qui seront employés. Le seconde partie fut consacré à l'implémentation et aux expériences.

Le reste de ce rapport présente les contraintes et les analyses préliminaires effectuées pour préparer la suite du projet, la conception et l'implémentation du programme lié au projet ainsi qu'une expérience qui a été réalisé à l'aide de ce dernier.

La suite de ce chapitre expose le contexte du travail, les buts de ce projet et les objectifs qui fixés pour celui-ci et une note sur la confidentialité liée à ce projet.

1.1 Contexte du travail

Le Human Brain Project (HBP) a pour but d'enrichir les connaissances humaines en matière de neuro-science en cherchant à mieux comprendre les mécanismes du cerveau humain. Ce projet s'inscrit dans le cadre du programme européen pour la recherche et l'innonvation Horizon 2020 et vise a accélerer les domaines des neuro-sciences, de l'informatique et de la médecine liée au cerveau. La première étape du Human Brain Project veut mettre à disposition des chercheurs un portail web. Ce portail web sera constitué d'un total de 6 plateformes de recherche. Celles-ci porteront sur la neuro-informatique, la simulation du cerveau, le calcul à haute performance, l'informatique médicale, l'informatique neuromorphique et la neuro-robotique.

Le département des Neuro-sciences Clinique du CHUV est chargé de la plateforme d'Informatique Médicale. Celle-ci est une plateforme open-source permettant aux hôpitaux et aux centre de recherche de partager des données médicales. Elle permettra aux utilisateurs d'avoir accès à des informations précises et pertinentes sur les maladies liées au cerveau en préservant la confidentialité des patients. Cette plateforme servira donc de pont entre la recherche en neuro-sciences, la recherche clinique et les soins aux patients. Elle pourrait également permettre la découverte de mécanisme à différentes échelles qui expliquerait l'apparition et le développement de maladies cérébrales.

C'est dans ce cadre que l'équipe du Laboratoire de Recherche En Neuro-imagerie cherche à développer un ensemble d' outils pour l'acquisition, le traitement et l'analyse des données. Elle cherche, entre autre chose, à pouvoir automatiser aussi efficacement que possible les diagnostiques de maladie pouvant atteindre le cerveau, tel que les maladies d'Alzheimer ou de Parkinson. Dans l'état actuel cette analyse peut se faire avec des méthodes de machine learning. Toutefois, il n'existe pas encore de méthode d'apprentissage profond disponible sur la plateforme et le LREN aimerait pouvoir proposé cette option aux utilisateurs de la plateforme.

C'est dans ce but que la Haute-Ecole Arc de Neuchâtel a été contacté et ce projet proposé comme travail de diplôme à un étudiant de troisième année.

1.2 Problème à résoudre et but du projet

Actuellement, la plateforme rassemble un certain nombre d'images d'IRM. Celles-ci sont stockés sous la forme de DICOM ou de fichier au format NIFTI. Le DICOM est une norme standard pour la gestion informatique des données issues de l'imagerie médicale. N'étant pas employé dans le reste du projet, il ne sera pas plus détaillé ici. Le format NIFTI est un format d' image IRM mis en place par quelques uns des acteurs les plus influents de la neuro-imagerie. Etant le format principalement employé dans ce projet, il fera l'objet d'une description détaillé dans la partie consacré aux analyses préliminaires.

(Corriger ce passage à propos de SPM) Afin d'être employé par les outils d'automatisation de diagnostique mis en place par la plateforme d'informatique médicale, les images ont besoin d'être pré-traité. En effet, les outils de machine learning utilisé pour le diagnostique fonctionne en se basant sur un certain nombre de caractéristiques du cerveau. Ces caractéristiques peuvent être le volume de matière grise ou de matière blanche d'une zone spécifique du cerveau, la quantité de liquide cérébro-spinal, le volume du cerveau, etc. Il faut donc extraire ces informations des images à disposition. Ceci se fait à l'aide du framework SPM. Ce framework permet grâce à la segmentation de récupérer des images d'une de ces caractéristique. La segmentation permet, par exemple, de récupérer une image IRM de la matière grise du cerveau au format NIFTI. SPM utilise la segmentation afin de créer un atlas des caractéristiques. Ainsi l'atlas fait correspondre, sous la forme de tableau, un certain nombre de quantité caractéristique à chaque région du cerveau.

(Insérer image du dataflow)

Une fois ce pré-traitement effectué, les données sont prêtes pour être utilisé par l'"'algorithm factory". Cette dernière correspond à l'ensemble des outils de diagnostique de la plateforme.

Le problème principal de cette manière de faire est qu'il existe une quantité non-négligeable d'information qui sont perdu au cours du pré-traitement. L'idée du LREN est donc de trouver une solution pour traiter directement les images entières avec des outils de diagnostique automatique.

Pour se faire, ils proposent de mettre en place une extension de l'"'algorithm factory". Cette extension permettra d'appliquer des algorithmes pour l'apprentissage de modèles et de faire valider ces derniers directement sur les images d'IRM.

Ce projet vise donc à explorer la possibilité de mettre en place cette extension. Il mettra en place un workflow alternatif à celui existant dans l'"'algorithm factory". Cette alternative a pour contrainte de permettre de lancer de nouveaux algorithmes travaillant directement sur les images en utilisant le framework de calcul distribué Apache-Spark. Cette nouvelle fonctionnalitée sera illustrée par l'intégration d'une bibliothèque de deep-learning et fera l'objet d'une expérience avec de véritable image d'IRM.

1.3 Rappel des objectifs du projet

Les premières semaines du projet ont été utilisé afin de fixer les objectifs principaux et secondaires de ce projet. Ainsi, les objectifs principaux de ce travail sont : 1. L'installation et la prise en main d'Apache-Spark 2. L'intégration d'Apache-Spark à l'"'algorithm factory" 3. L'interfaçage des bases de données d'image de la plateforme à Apache-Spark. Ces bases de données sont a créer et a améliorer si besoin durant le projet. 4. Faire un état de l'art technique sur les différentes bibliothèques de deeplearning compatible avec Apache-Spark pour obtenir suffisamment d'information pour permettre le choix de l'une d'entre elles à intégrer au-dessus de Spark. 5. Traduire la partie prédictive de l'algorithme au format PFA (Portable Format for Analytics). 6. Tester les nouvelles fonctionnalités avec une expérience concrète fournit par le LREN. Cette expérience utilisera des images d'IRM utilisé par le laboratoire. Elle consistera en une classification de ces images.

En plus de ces objectifs principaux, s'ajoute un objectif optionnel. Celui-ci consiste a étendre le portail web de la plateforme pour permettre l'utilisation des nouvelles fonctionnalités de l'"'algorithm factory".

Il faut tout même signalé que ces objectifs ont évolué au cours du projet. Ces changements seront expliqué durant les chapitres concernant la conception et l'implémentation du projet.

1.4 Note sur la confidentialité au cours du projet

Comme déjà rappelé dans le cahier des charges de ce travail, l'aspect de l'utilisation d'image extraite d'IRM est un aspect sensible du point de vue de la confidentialité.

Pour pallier tous soucis de confidentialité, les images employées durant la phase de développement seront des images totalement ouvertes même si ces dernières ne sont pas des images issues d'IRM.

Si des images autres que des données de recherche devaient être utilisées, elles seront anonymisée et ne quitteront jamais le réseau sécurisé des hôpitaux dont elles sont originaires.

Une attention particulière devra également être portée sur la réutilisation de l'existant afin de respecter les directives de plagiat et le droit d'auteur (cf. directives générales en matière de plagiat de la HE-ARC).

CHAPITRE 2

Analyses préliminaires

2.1 Le format NIFTI

Ce travail est un projet de neuro-imagerie, il est donc naturel de devoir travailler avec des images IRM du cerveau. Le format utilisé par le CHUV pour les images est le format NIFTI (Neuroimaging Informatics Technology Initiative), un format d'image très spécialisé mais également très répandu dans ce domaine.

Ce chapitre présente donc ce format afin de mieux le comprendre. Pour faire celà, nous allons voir l'origine du format, une vue d'ensemble des principales caractéristiques du format et quelques outils qui ont été utiles à la réalisation de ce travail.

2.1.1 Origine du format NIFTI

NIFTI est un format de fichier pour sauvegarder des données d'IRM. Il fonctionne sur le principe des voxels et est multidimensionnel.

Ce format a été imaginé il y a une dizaine d'année pour remplacer le format ANALYZE 7.5. ANALYZE 7.5 était très utilisé mais était également très problématique. Le soucis principal de ce format étant le manque d'information sur l'orientation dans l'espace de l'élément scanné. Les données enregistrées ne pouvaient donc pas être lu et interprêté sans ambiguité. A cause de ce manque d'information, il existait principalement une confusion entre le côté droit et le côté gauche du cerveau.

Deux conférences furent alors mises en place par quelques-uns des concepteurs des plus grands logiciels de neuroimagerie. Ces deux conférences, le Data Format Working Group (DFWG), se sont réunit au "National Institue of Health" (NIH) pour trouver un format de remplacement. De ces réunions naquit le format NIFTI. Celui-ci intégre de nouvelles informations et est devenu le nouveau standard de neuroimagerie.

2.1.2 Vue d'ensemble du format NIFTI

Le format ANALYZE 7.5 avait besoin de deux fichiers pour fonctionner. Un fichier *.hdr contenant le header pour stocker les méta-données et un fichier *.img contenant les données de l'image. Le format NIFTI a conservé l'idée d'avoir un header et des données afin de préserver la compatibilité avec les systèmes déjà en place. Toutefois, des améliorations ont été apportés et pour évité de faire l'erreur d'oublier l'un des deux fichiers du format, il a été décidé de permettre le stockage de ces informations dans un seul fichier avec l'extension *.nii. Ces images contenant de grandes zones d'image noires, elles sont donc parfaites pour être compressées avec gzip. Il n'est donc absolument pas rare de trouver des fichiers NIFTI au format *.nii.gz. Pour ce travail nous avons utilisé les formats *.nii et *.nii.gz.

Le format NIFTI est un format de fichier que l'on peut représenter par une matrice multidimensionnel. Au total, elle peut compter jusqu'à 7 dimensions. Dans tous les cas, les 3 premières dimensions sont des dimensions spatiales (x, y, z) et la quatrième est une dimension temporelle. Les dimensions suivantes (5-7) sont des dimensions réservées à d'autres usages et sont plus ou moins libre. Dans le cadre de ce projet, les images utilisées ne possèdent que 3 dimensions (les 3 dimensions spatiales). On peut donc voir les images comme étant un instantané du cerveau en 3 dimensions et chaque case de la matrice de données représente un voxel de cette image.

Les dimensions et d'autres informations importantes sur le fichier sont stocké dans un fichier header. Ce dernier est d'une taille de 348 octets. (Il y a un tableau de toutes les valeurs sur brainder.org il doit venir etre collé ici.)

Le champs principalement utilisé lors de ce projet est le champs short dim[8]. Ce champs est un tableau contenant les données sur les dimensions du fichier. Ce tableau contient pour : - Dim[0] : Le nombre de dimensions - Dim[1 -7] : Un nombre positif contenant la longueur de la dimension en question.

Pour ce travail deux types de NIFTI ont été employé. Le premier type de NIFTI a avoir été utilisé sont des images générés très simple. Ces images correspondent à des sphères et des cubes. La dimension de ces images générées peut être choisi. Au début du projet, de manière a facilité les tests, la taille de ces images étaient de 100x100x100. Puis lorsque le projet eut une forme plus concrète la taille fut changer pour correspondre à la taille standard utilisé par le CHUV (190x190x160). Le second type de données correspond aux images fournient par le LREN. A savoir des images de la matière grise du cerveau avec une taille standard de 190x190x160.

2.1.3 Outils pratique

Le format NIFTI est un format très spécifique au domaine de la neuro-imagerie. Il est donc nécessaire afin de se familiariser avec ce format d'utiliser un certain nombre d'outils de visualisation. Une gamme d'outils en ligne de commande existe. Cette dernière s'appelle Fslutils. Il fournit un set complet de ligne de commande utiles pour convertir et manipuler les fichiers NIFTI. Une liste complète des outils fournit par Fslutils est fournit sur leur pages Internet.

De toute cette gamme d'outils, deux ont principalement été employé : * fslinfo : affiche les informations du header du fichier NIFTI ou ANALYZE lu * fslsplit : découpe une image IRM 4D en un lot de fichier 3D

Une fois le format appréhendé, il faut un outil permettant de les lire dans un programme. La plateforme du CHUV et notre projet devant pouvoir tourner sur une JVM, une librairie JAVA a été trouvé. Cet outil est le projet niftijio. Cet outil permet de lire le header et les données d'un fichier NIFTI et de les récupérer sous la forme d'un tableau dans un programme en Java ou Scala.

2.2 Le calcul distribué

Le nombre d'image et la taille de ces dernières font qu'il y a un nombre très important de données et de calcul a effectué. Pour le confort de l'utilisateur, le temps de traitement de ces données doit être le plus court possible. La plateforme actuellement en place au CHUV tourne donc sur un cluster de machine afin de permettre à l'utilisateur d'obtenir le plus rapidement possible les résultats des analyses qu'il demande.

Ce projet doit donc pouvoir se porter sur l'infrastructure en place. De plus, le Laboratoire de Recherche En Neuro-imagerie désire intégrer la technologie Spark pour effectuer leur calcul. Ces deux contraintes ont donc fait l'objet d'une analyse et sont exposé dans ce chapitre.

2.2.1 Qu'est ce que le calcul distribué ?

Ces dernières années la quantité de données disponibles a explosé. Rapidement, les technologies ont du s'adapter à cette quantité d'information toujours plus importante à traiter. L'une des solutions trouvé pour résoudre se problème consiste à répartir les tâches de traitement (de calcul) sur plusieurs unité de travail. Ainsi, on répartit le besoin en puissance de calcul, pour un projet, en petite entités sur autant d'ordinateurs disponible qu'il y en a dans notre réseau distribué.

Celà permet d'exploiter les ressources de chaques machines au profit d'un projet commun. Ce projet dispose alors d'une puissance de calcul correspondant à la somme de tous les ordinateurs individuels.

Le calcul distribué s'effectue donc au sein d'un cluster de machine. C'est à dire, au sein d'un groupe de machines indépendantes fonctionnant comme une seule et même entité. Chacune de ces entités correspond à un noeud. Si une machine est ajouté au cluster, la puissance de calcul est directement augmenté contrairement à une machine seule, où si l'on veut augmenter la puissance de calcul, il faut augmenter la puissance des processeurs.

Pour le calcul distribué, les noeuds sur lesquels les calculs sont exécuté sont donc distant, autonome et ne partage pas de ressources. Il faut donc que chaques noeuds communiquent avec les autres au travers de message qu'il s'envoie au travers du cluster.

Pour pouvoir distribuer son projet, il faut donc diviser le problème initial en sous-problème et assigner à chaque noeud l'un de ces sous-problèmes. Chaque noeud effectue la tâche qui lui est assigné. On récupère alors le résultat de chacun des sous-problèmes et on les combine pour obtenir le résultat finale du projet initial.

Afin de gérer tout celà il est possible d'utiliser des framework de calcul distribué. Ces framework fournissent un ensemble d'outils pour faciliter la création d'application distribuées. Le CHUV à choisi pour ce projet d'utiliser le framework Apache-Spark. La suite de ce chapitre présentera donc ce framework et son fonctionnement.

2.2.2 Spark

Spark est un framework open-source de calcul distribué écrit en Scala. Il a été conçu en 2009 par Matei Zaharia lors de son doctorat au sein de l'université de Californie à Berkley. L'objectif de Matei Zaharia lors de la conception de Spark était de trouver une solution pour accélerer le traitement des systèmes Hadoop. Spark est transmis a Apache en 2013 et devient l'un des projets les plus actifs de la firme. Le framework a le vent en poupe (à l'instar de Docker que nous verrons plus loin dans ce rapport) et est en train de remplacer Hadoop. En effet, il a été démontré que Spark permet des temps d'exécution jusqu'à 100 fois plus courts qu'Hadoop pour les mêmes tâches. La dernière version de Spark est Spark 2.2.0 et est disponible depuis le 11 juillet 2017. Spark fournit une API haut-niveau en Java, Scala, Python et R.

Afin de fonctionner aussi rapidement Spark fonctionne directement en mémoire et cherche a avoir un traitement proche du temps-réel. Lorsque Spark execute des tâches, il cherche à maintenir les résultats intermédiaires en mémoire plutôt que sur le disque. Cette manière de faire permet de facilement pouvoir travailler à plusieurs reprises sur le même jeu de données. Toutefois Spark n'est pas restreint au travail en mémoire. Il peut aussi bien travailler sur le disque. Les opérateurs réalisent des opérations externes lorsque les données ne tiennent pas en mémoire. Par défaut, Spark essaie de stocker le plus d'info en mémoire avant de basculer sur le disque. Cependant, ce comportement est configurable. Il est possible de demander a Spark de ne travailler que sur le disque ou uniquement en mémoire mais également avec une partie des données en mémoire et l'autre partie sur le disque.

Spark possède un écosystème contenant des bibliothèques additionnelles qui permettent de travailler dans les dommaines du "big data" et du machine learning. Dans cet écosystème, on trouve notamment :

- Spark Streaming : Permet le traitement temps-réel des données de flux.
- Spark SQL : Permet d'exécuter des requêtes SQL pour charger et transformer les données et ce quel que soit le format d'origine de celles-ci.
- Spark GraphX : Permet le traitement et la parallélisation de graphes.
- Spark MLlib : Est une bibliothèque d'apprentissage automatique qui contient tous les algorithmes et utilitaires d'apprentissage classiques, tel que la classification,

la régression, le clustering, le filtrage collaboratif et la réduction de dimension, en plus des primitives d'optimisation nécessaires à ces tâches.

L'architecture de Spark comprend les trois composants principaux suivants : * Un composant de stockage des données qui utilise le système de fichier HDFS pour le stockage. * Une API haut-niveau * Un composant de gestion des ressources. Ce composant permet a Spark d'être déployé comme un serveur autonome ou sur un framework de traitements distribués comme Apache-Mesos ou Apache-YARN.

L'élément de base principal au coeur de Spark est le "Resilient Distributed Dataset" ou RDD. Un RDD est une abstraction de collection sur laquelle les opérations sont effectué de manière distribué et en étant tolérante aux pannes matérielles. On peut donc les voir comme une table dans une base de données. Un RDD peut contenir n'importe quel type de donné et est stocké par Spark sur différentes partitions. Ainsi, le traitement que l'on écrit pour un RDD semble s'exécuter sur une JVM mais il sera en fait découpé pour s'exécuter sur plusieurs noueds. Si le cluster de machine perd un noeud, le sous-traitement sera automatiquement relancé sur un autre noeud par le framework. Ceci est possible car un RDD sait recréer et recalculer son ensemble de données. Les RDD supportent deux types d'opérations : * Les transformations(map, filter, flatMap, groupByKey, reducebyKey, etc...) : Celles-ci retourne un nouvel RDD. * Les actions(reduce, collect, count, first, take, foreach, etc...) : Celles-ci évaluent et retournent une nouvelle valeur.

L'exécution de Spark peut se faire de plusieurs manière différente. Pour celà il suffit de donner le bon paramètre de connexion au moteur de Spark (Master, chef d'orchestre). Ainsi, la connexion au moteur peut se faire de manière

local (sur un ou K “worker”), en se connectant à un cluster Spark, Mesos ou Yarn.

(Add tableau)

Spark fournit également une interface web. Pour joindre cette interface, il suffit, une fois Spark en cours d'exécution, de se connecter sur le port 4040 du localhost. Cette interface permet de surveiller le stockage, l'environnement, les exécuteurs et les étapes effectué par Spark.

Spark possèdent encore bien des caractéristiques qui font de lui l'un des leaders du domaine. Toutefois, nous avons vu ici ces principales caractéristiques et les principaux outils utilisé durant l'élaboration de ce travail de Bachelor. L'utilisation de Spark dans le projet sera détaillé plus loin dans la rédaction de ce rapport.

2.3 Le deeplearning et choix d'une bibliotheque

La plateforme d'informatique médicale tenue par le LREN aimerait pouvoir donner à ces utilisateurs la possibilité de lancer des expériences de deeplearning. Ce projet a donc pour objectif d'ouvrir la voie à ce procédé.

Il est donc important de faire le point sur cette technologie. Cette partie va donc permettre de voir ce que sont les réseaux de neurones et le deeplearning. Puis dans un second temps, les réseaux de convolution seront abordé. Dans une troisième partie, ce rapport abordera les différentes manières de mélanger calcul distribué et deeplearning. Ces trois premières parties permettront de se faire une idée de ce concept et d'aborder plus sereinement l'état de l'art des bibliothèques de deeplearning et le choix de l'une d'entre elle pour ce travail.

2.3.1 Considération générale

Machine learning et bases

Le deeplearning est un ensemble de méthodes de machine learning. Le machine learning est l'un des champs d'étude de l'intelligence artificielle et cherche à permettre à une machine de modéliser des phénomènes dans le but de prendre des décisions et de résoudre des problèmes concrets. Cette capacité à prendre des déscisions se fait sans être explicitement programmé par le développeur.

Un problème concret peut, par exemple, être d'identifier des fraudes, d'aider aux diagnostiques médicaux, de recommander un article personnalisé à un client, de prédire le prix d'un produit, etc. L'idée derrière le machine learning est alors de permettre à la machine de se construire une représentation interne du problème sans que le développeur n'ait besoin de la modéliser pour elle.

A l'aide de cette modélisation, la machine pourra alors effectuer la tâche qui lui est demandé. La tâche demandé au cours de ce projet est une tâche de classification. La classification sert à pouvoir ranger une donnée (une image par exemple) dans une classe spécifique. Pouvoir dire d'une image qu'elle représente un chat ou un chien par exemple. Etant la tâche sur laquelle ce travail se base la classification sera utilisé comme exemple dans la suite de ce rapport.

Pour que l'algorithme de machine learning puisse se construire une représentation du problème, il faut lui fournir un jeu de données d'exemple. Grâce à ce jeu de données, l'algorithme va pouvoir s'entraîner et s'améliorer dans la tâche qui lui a été confié. Nous pourrons par la suite lui fournir des données réels et obtenir un résultat aux problèmes posés.

Il existe différent algorithme de machine learning. Parmis eux nous pouvons noté : * La régression linéaire * La classification naïve de Bayes * Machine à vecteurs de support (SVM : support vector machine) * K-nn * Random Forest (Forêt d'arbres décisionnnels) * Réseau de neurones

Les neurones formels

Le deeplearning est une technique qui fonctionne sur la base des réseaux de neurones. Les réseaux de neurones sont construit à partir d'un paradigme biologique. Ce paradigme est celui du neurone formel. Un neurone formel est une représentation mathématique et informatique d'un neurone biologique. Le neurone formel possède généralement plusieurs entrées et une sortie. Les entrées correspondent ainsi aux dendrites d'un neurone, tandis que la sortie correspond à l'axone de ce dernier. Pour fonctionner, un neurone biologique reçoit des signaux excitateurs et

inhibiteurs grâce aux synapses (lien entre deux neurones). Ces signaux sont simulés dans un réseau de neurones informatiques par des coefficients numériques associés aux entrées des neurones. Ces coefficients sont appelés les biais. Les valeurs numériques de ces coefficients sont ajustées durant la phase d'apprentissage dans un réseau. Le neurone formel fait alors des calculs avec les poids pondérés des entrées reçues, puis applique au résultat de ce calcul une fonction d'activation. La valeurs finale obtenue se retrouve alors sur la sortie du neurone. Ces neurones formels peuvent ensuite être assemblé entre eux pour former des réseaux et réaliser des tâches plus complexe. Ainsi le neurone formel est l'unité élémentaire des réseaux de neurones artificiels.

L'un des éléments capitaux d'un neurones formels est la valeur de ces biais. Un biais est la pondération d'une des entrées. La plupart des neurones informatiques effectue une somme pondéré de leur entrée. Ainsi, plus une entrée à une valeur de biais importante, plus la "force" de la connexions est grande. Cela simule le comportement d'un neurone biologique, dans lequel plus une connexion est importante, plus celle-ci est sensible aux stimulis chimiques. Dans un reseau de neurones, la valeur des biais est mis à jour durant la phase d'apprentissage que nous avons évoqué plus haut. Durant cette phase, on va chercher à faire converger les valeurs des biais pour s'assurer une classification aussi proche que possible de l'optimal.

L'autre éléments important d'un neurones formel est sa fonction d'activation. Dans un premier temps, le neurone va récupérer ces entrées et les agrégers avec une fonction d'agrégation. La fonction d'agrégation la plus simple consiste en une somme pondéré par les biais des valeurs en entrée. Toutefois, cette fonction peut être plus complexe. Le but de cette fonction est d'obtenir une valeur agrégé des entrées du neurones. Ce dernier utilise cet agrégat comme paramètre d'une fonction d'activation. Cette fonction a pour rôle de décider si le neurone est actif ou non. Elle permet également de donner la valeur a fournir en sortie du neurone. Il est donc important de bien choisir cette dernière. Il existe en effet plusieurs fonctions d'activations typiques : * La fonction sigmoïde * La fonction de Heaviside * La fonction tangente hyperbolique * La fonction de base radiale * La fonction sigma-pi * La fonction RELU * La fonction SOFTMAX * etc

L'objectif de la fonction d'activation est d'introduire de la non-linéarité dans le fonctionnement du neurone. Les fonctions d'activation présente, en règle général, trois intervalles : * Sous le seuil d'activation, le neurone est inactif * Au alentour du seuil, le neurone est dans une phase de transition * Au dessus du seuil, le neurone est actif

Le neurone formel est la brique de base des réseaux de neurones que nous allons voir dans la partie suivante.

Les réseaux de neurones

Un réseau de neurones est simplement un ensemble de neurones liés entre eux, la sortie d'un premier neurone devenant l'entrée d'un second neurone.

Il existe plusieurs types de réseaux de neurones qui vont chacun dépendre de la manière d'organiser les neurones. Le modèle de réseau le plus simple est le perceptron simple. Un perceptron est un réseau de neurones monocouche et permet une classification linéaire. Nous allons prendre ce modèle pour expliquer quelques notions importante. Un perceptron possède n informations sur ces entrées et p neurones formels formant une couche. Chacun des p neurones est connecté aux n informations d'entrée et a sa propre sortie.

Un perceptron peut avoir une représentation matricielle. Ainsi on peut considérer le n information d'entrée comme un vecteur à n composantes. Et la sortie du perceptron est un vecteur de p composantes. Et finalement le perceptron est une matrice de n lignes et p colonnes. Cette matrice est rempli avec les différents poids de chacunes des connexions avec le vecteur d'entrée. Elle est régulièrement appelé matrice de poids. En étendant ce principe nous pouvons en déduire que les réseaux de neurones même plus complexe, sont en réalité de longues chaines de calcul matriciel.

Un perceptron tel que décrit au dessus est également un réseau feed-forward. Il existe, en effet, manière de "nourrir" un réseau de neurones. Un réseau de neurones peut donc être "feed-forward" ou récurrent. Un modèle récurrent peut alimenter ses entrées avec ses sorties. Tandis que les réseaux "feed-forward" ne le peuvent pas.

Nous pouvons pouvons étendre le concept du perceptron en lui ajoutant des couches. Le perceptron de l'exemple précédent devient alors un perceptron multi-couches. Une couche est un ensemble de neurones connectés aux mêmes entrées mais pas relié entre eux. Dans le modèle multi-couches, les réseaux récurrents peuvent utiliser leurs sorties pour alimenter des entrées des couches précédentes.

Les modèles de deeplearning sont bâtis sur le même principe que les perceptrons multi-couche. Dans ce genre de modèle, le nombre de couche du perceptron sont plus nombreuses. Chacune des couches intermédiaires étant chargé de résoudre ou de découper un sous-problème. Ainsi si l'on imagine en entrée du réseau une image de

voiture, les couches pourraient hierarchisé cette image de cette façon : 1. Couche 1 : Cette image représente un véhicule 2. Couche 2 : Ce véhicule est motorisé 3. Couche 3 : Ce véhicule motorisé a 4 roues 4. Couche 4 : Ce véhicules motorisé à 4 roues est une voiture Ainsi chaques couches ajoute un contexte de plus en plus précis aux données passées en entrée.

Note sur l'apprentissage d'un réseau de neurones

Comme déjà souligné, un réseau de neurones ne peut effectué la tâche pour laquelle il est conçu qu'après avoir subi une phase d'apprentissage. Cette phase d'apprentissage consiste à mettre à jour les biais de chaques neurones pour les faire converger vers une meilleure précision de l'algorithme.

Il existe différents type d'apprentissage. Les apprentissage supervisé et ceux qui ne le sont pas (non-supervisé). Un apprentissage supervisé se fait en donnant des labels aux données d'entrée, en les étiquetant. Le modèle peut alors apprendre en se basant sur ces labels. Dans l'apprentissage non-supervisée les données ne sont pas étiquetté. Pour le projet qui nous intéresse la méthode d'apprentissage utilisé est une méthode supervisée.

De manière générale pour s'entraîner le réseau va lire des données d'entrainement, tenté de les classifier, calculer l'erreur qu'il fait à chaque itération et modifier les poids de manière à diminuer l'erreur calculé.

Cette manière de faire est une méthode que l'on appelle algorithme de rétro-propagation. On peut ensuite utilisé la technique de la descente de gradient qui consiste a calculer la direction, dans l'espaces des poids, dans laquelle la décroissance de l'erreur est maximal et mettre à jour nos poids.

Il est a noté que le temps nécessaire à l'apprentissage augmente très rapidement avec le nombre de couche du réseau et que les réseaux de neurones peuvent subir un surapprentissage.

Le surapprentissage (overfitting) est un problème qui empêche le réseau de généraliser les caractéristiques des données. Le réseau perd alors sa capacité à prédire. Il existe des manières simple d'éviter le surapprentissage : * Cross-validation : consiste a créer deux sous-ensembles de données. On créer un sous-ensemble d'entraînement et un sous-ensemble de validation. On entraine ensuite le réseau avec le sous-ensemble d'apprentissage et on test son pouvoir prédictif avec l'ensemble de validation. Ainsi si l'erreur lors de l'apprentissage diminue alors qu'elle augmente sur les données de validation le réseau est sur-entrainé. On essaie donc d'arrêter l'entraînement lorsque l'on constate cette divergence. * Régularisation : consiste a pénaliser les valeurs extrêmes des paramètres.

Au terme de l'entraînement, il est possible d'élaguer notre réseau. Cette technique consiste a supprimer les connexions ayant peu d'influence sur le reste du réseau. Cela permet de faire gagner du temps pour la tâche a effectuer ensuite.

2.3.2 Réseaux de convolution

Notions générales

Ce projet vise a traiter des images d'IRM. L'un des type de réseau de neurones dont la spécialité est de traiter des images est le réseau de neurones convolution (CNN). Cette partie du rapport est consacré à ce type de réseau.

Les réseaux de convolution sont un type de réseau de neurones acyclique ("feed-forward") dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. On peut donc lui passer en entrée une image et lui demander de la classifier.

Ce genre de réseau de neurones est, en général, conçu en deux parties distinctes. La première partie est la partie convolutive du réseau. Cette dernière agit comme un extracteur de caractéristiques. On passe l'image à travers un certain nombre de filtre (noyau de convolution) pour créer de nouvelles images (carte de convolution). Les couches de convolution sont en principe suivi d'une couche de correction utilisant la fonction d'activation RELU. On peut placer, entre les opérations de convolution, des filtres chargé de réduire la résolution de l'image par une opération de maximum local (Pooling). Au terme de cette étape, les cartes de convolutions sont concaténé en un vecteur de caractéristiques. Ce vecteur est souvent appelé le code CNN.

Ce code est alors utilisé en entrée de la seconde partie du réseau convolutif. Cette partie est en règle général constitué de couches entièrement connectées entre elle. Le but de cette partie est de combiner les caractéristiques du code CNN pour classifier l'image.

La sortie de ce genre de réseau est en principe une dernière couche contenant autant de neurones qu'il n'y a de classe possible. La sortie est en principe un vecteur dont le nombre de composant est le nombre de classe disponible. La valeur de ces composantes est compris entre 0 et 1. Et la somme des composantes vaut 1. En lisant ce vecteur on obtient la distribution de probabilité que l'image appartiennent a une catégorie ou une autre.

Pour concevoire des couches de convolution, il existe trois paramêtres particulièrement important a géré. Ces paramêtres sont : * La profondeur : le nombre de filtre que l'on va utiliser * Le pas : Le pas contrôle le chevauchement des noyaux de convolution * La marge : La marge contrôle la dimension spatiale du volume de sortie. Elle ajoute des 0 à la frontières de l'image en entrée.

Après la couche de convolution, on peut trouver une couche de correction. Cette couche semble permettre d'accélerer la vitesse de traitement du réseau sans affecté la précision. La fonction d'activation de cette couche peut être : * Correction RELU : permet d'augmenter les propriétés non-linéaire du réseau * Correction par tangente hyperbolique * Correction par la fonction sigmoide * etc La correction la plus utilisé est la correction RELU.

Une fois la correction effectué, on peut trouver une couche de pooling. Elle permet de sous-échantillonner les données d'entrée (l'image). Elle permet de réduire le risque de sur-entraînement du réseau. Toutefois, il faut faire attention à utiliser de petit filtre afin de ne pas perdre trop d'information. Pour cette couche on va en effet passer sur l'image des filtres chargés d'extraire une valeur de l'image. L'une des méthodes les plus utilisé est d'employer un filtre de taille 2x2 chargé d'extraire la valeur maximum des pixels contenu dans le filtre. Le pooling permet d'augmenter l'efficacité du traitement. Toutefois, il casse le lien entre une image et son contenu (par exemple entre le nez et l'image d'un visage). Cette relation peut être intéressante a conservé. En faisant débordé les filtres du pooling les unes sur les autres, il est possible de définir une position pour un élément. Il devient alors possible de dire que le nez est au milieu du visage par exemple. Cependant, il faut noter que faire ceci empechera tout autres formes d'extrapolation (changement d'angle de vue ou d'échelle), contrairement à ce que le cerveau humain peut faire. Pour améliorer ce problème, on peut passer des données d'entraînement plus variées à notre réseau. Ces images peuvent être plus variées en terme de luminosité, angle de vue ou taille.

La toute dernière couche d'un réseau de convolution est une couche de LOSS. Cette couche est chargé de définir la classe dans laquelle l'image se situe. Elle peut être activé par différentes fonction d'activation : * Fonction SOFTMAX : Une fonction idéal lorsque l'on doit ranger les images en 2 classes. C'est une fonction mutuellement exclusive. * Fonction d'entropie sigmoide croisé : prédis des valeurs de probabilité indépendante dans [0, 1] * Fonction euclidienne : Regression vers des valeurs réelles (contenu entre moins l'infini et plus l'infini)

Il existe plusieurs modèles de réseau convolutif devenus des standards. Ces architectures sont les suivantes : * INPUT + CONVOLUTION + RELU + FULLY CONNECTED + LOSS * INPUT + (CONVOLUTION + RELU +POOLING)*2 + FULLY CONNECTED + RELU + FULLY CONNECTED + LOSS * Input + (CONVOLUTION + RELU + CONVOLUTION + RELU + POOLING)*3 + (FULLY CONNECTED + RELU)* 2 + FULLY CONNECTED + LOSS

2.3.3 Deeplearning et calcul distribué

Ce projet devra pouvoir fonctionner avec du calcul distribué. Comme vu plus tôt dans ce rapport, le calcul distribué permet de répartir des tâches entre plusieurs machines connecté à un même cluster de machine. Il existe différentes techniques pour distribuer des tâches dans un réseau de deeplearning. Il existe deux modèles principaux : * La parallélisation des données * La parallélisation du modèle

Dans la parallélisation du modèle, les différentes machines sur le réseau distribué sont en charge d'une partie du réseau. Par exemple, chaque machine peut se voir assigné la gestion d'une couche du réseau de neurones.

Dans la parallélisation des données, les différentes machines sur le réseau distribué ont une copie complète du modèle de réseau. Chaque machine reçoit alors une partie des données et entraîne son modèle. Au terme de l'entraînement, les résultats sont combinées entre eux.

Les approches d'entraînement en utilisant la parallélisation des données nécessitent toutes une méthode de combinaison des résultats et de synchronisation des paramètres du modèle entre chaque machine.

L'implémentation actuel de la bibliothèque de deeplearning choisi pour ce projet permet de faire de la parallélisation des données. Pour faire celà, la bibliothèque utilise les techniques de moyennes des paramêtres synchrone.

L'approche de la moyenne des paramètres est l'approche la plus simple de la parallélisation des données. En utilisant cette technique, l'apprentissage fonctionne ainsi : 1. Les paramètres du réseau sont inialisé de manière

aléatoire en fonction de la configuration du modèle 2. Une copie des paramètres actuel est distribué sur chaque machine 3. Chaque machine entraîne son modèle avec les données en sa possession 4. De nouveaux paramètres globaux sont calculé en fonction de la moyenne des paramètres de chaque machine 5. Tant qu'il y a des données à traiter, on retourne a l'étape 2

2.3.4 Bibliothèque disponible et choix

Cette partie du chapitre va faire un état des biblitothèques de deeplearning actuellement disponible. Puis en ce basant sur les contraintes fourni par le CHUV et la plateforme d'Informatique Médical du Human Brain Project, elle défendra le choix de la bibliothèque choisi pour le reste du projet. Il faut rappeler que ces contraintes sont : * L'utilisation du calcul distribué avec Spark * Une plateforme qui fonctionne en Scala

Liste de bibliothèques disponible

Ce rapport va ici faire une liste des bibliothèques vu pour ce projet. Chacune d'entre elle sera accompagné d'une brève description et de ces caractéristiques principales.

TensorFlow

TensorFlow est une bibliothèque de programme open-source développé par Google. Cette bibliothèque est utilisé dans de nombreux produit Google. Ces principales caractéristiques sont : * Qu'elle est utilisable en Python * Qu'elle possède une API en C++ * Qu'elle possède une grosse documentation et est très utilisé * Que c'est un projet très solide de Google Toutefois, pour pouvoir être utilisé en scala il est nécessaire d'utiliser un outil comme ScalaPy. Cette biblitothèques a donc été rejeté car on ne peut pas se passer de ScalaPy.

TensorFrames

TensorFrames est un portage expérimentale en Scala de TensorFlow. Ce portage est fait par Databricks. Ces principales caractéristiques sont : * Que c'est un portage expérimentale ne fonctionnant que sur des plateformes Linux 64 bits * Qu'elle est utilisable directement en Scala et en Python Cette bibliothèque étant expérimentale, cette bibliothèque a été écarté.

BigDL

BigDL est une bibliothèque conçu pour Spark et pouvant fonctionner sur les clusters Spark ou Hadoop existant. Elle a été crée par Intel. Ces principales caractéristiques sont : * Qu'elle fonctionne nativement en Java * Qu'elle est directement intégrable a Spark * Qu'elle a été conçu pour supporter le calcul distribué * Qu'elle ne fonctionne que sur les chips Intel Le fait que cette bibliothèque ne fonctionne que sur les chips Intel a écarté cette bibliothèque.

Keras

Keras est une API de haut-niveau écrit en python et capable de fonctionner sur TensorFlow, CNTK ou Theano. Ces principales caractéristiques sont : * Qu'elle fonctionne en Python et nécessite donc d'être binder à du Java/Scala * Qu'elle supporte le CPU et le GPU * Qu'elle est conçu pour faire du prototyping rapidement Le fait qu'elle fonctionne en python a permis son élimination.

Caffe on Spark

Caffe on Spark est une bibliothèque mêlant le framework Caffe et Spark ou Hadoop. Elle est géré par Yahoo. Ces principales caractéristiques sont : * Qu'elle fonctionne sur GPU et CPU * Qu'elle fonctionne sur les systèmes de fichier HDFS d'Hadoop * Qu'elle permet de gérer le réseau de neurones depuis Spark ou Hadoop * Qu'elle

fonctionne en Java * Qu'elle a besoin d'être installé sur chaque noeud du cluster Le fait que cette bibliothèque ait besoin d'être installé sur chaque noeud l'a écarté. En effet, c'est une chose dont le LREN aimerait se passer.

SparkNet

SparkNet est une bibliothèque de deeplearning conçu en Scala dont les pincipales caractéristiques sont : * Qu'elle fonctionne sur Spark * Qu'elle est nativement conçu en Scala * Qu'elle est supportée que sur Ubuntu (CPU/GPU) et sur CentOS Le nombre de plateforme sur laquelle elle est employable a éliminer cette bilbiothèque.

Deeplearning4j

Deeplearning4j est une bibliothèque conçu pour la JVM et capable de fonctionner sur Spark. Elle est conçu pour tourner tant sur CPU que sur GPU. Ces concepteurs la vende comme un outils pour le deeplearning à échelle industrielle. Ces principales caractéristiques sont : * Qu'elle est conçu pour fonctionner avec la JVM, codé en Java * Qu'elle fonctionne sur GPU et sur CPU * Qu'il est possible de la faire fonctionner avec des modèles issu de Keras * Qu'elle fournit des outils pour tourner sur Spark. * Que sont intégration à un projet se veut simple en utilisant Maven, Graddle ou encore SBT * Qu'elle possède une API Scala (ScalNet) * Qu'elle a un support actif Pour tous les avantages qu'elle donne cette librairie a été choisi en concertation avec le LREN.

2.4 Docker

Le LREN utilise pour sa plateforme un système de container Docker. Ce travail devra donc pouvoir être contenu dans un environnement Docker. Cette technologie étant relativement nouvelle, ce chapitre va brièvement exposer ce qu'est Docker.

Docker est un logiciel open-source qui automatise le déploiement d'application dans des conteneurs logiciels. Le développement avec Docker permet d'éliminer le problème de la collaboration lors de l'écriture d'un logiciel en fournissant à chaques collaborateurs un environnement de travail semblable. Docker permet d'exécuter et de gérer des applications fonctionnant côte à côte dans des conteneurs isolés. Il fournit également les outils nécessaires pour créer des pipelines de livraison et de partage de logiciel de manière sûre et rapide.

Un conteneur Docker contient tout ce qui est nécessaire pour faire exécuter un logiciel. Au contraire des machines virtuelles, les conteneurs ne regroupe pas un système d'exploitation complet. Il ne contient, en effet, que les bibliothèques et les paramètres requis pour que le logiciel fonctionne. Cela permet d'avoir des systèmes autonomes, légers et garantit que les logiciels fonctionnent de la même manière quel que soit l'endroit où ils sont déployé. Les conteneurs isolent le logiciel de son environnement.

Les conteneurs et les machines virtuelles ont des avantages similaires en matière d'isolation et d'allocation des ressources. Toutefois, leurs fonctionnements sont très différents. En effet, les conteneurs préfèrent virtualiser le système d'exploitation plutôt que le matériel. Les conteneurs se veulent donc plus portable et efficaces. Toutefois, les conteneurs et les machines virtuelles peuvent être utilisé ensemble.

Docker automatise les tâches répétitives de configuration des environnements de développement. Lorsqu'une application est encapsuler dans un conteneur, la difficulté de configurer et installer un système est également encapsulé dans le conteneur.

CHAPITRE 3

Conception

3.1 Schémas conceptuels

3.2 Description des classes

3.2.1 Package “Core”

La classe “Main”

La classe “DataReader”

3.2.2 Package “Config”

La classe “Configuration”

3.2.3 Package “Generator”

La classe “DataTestGenerator”

3.2.4 Package “Wrapper”

La classe “WrapperDl4j”

La classe “LocalWrapperDl4j”

La classe “SparkWrapperDl4j”

3.3 Choix de la topologie du/des reseaux de neurones

3.4 Description du workflow

CHAPITRE 4

Implémentation

CHAPITRE 5

Expérience réalisée avec le CHUV

5.1 Donnée de l'expérience

5.2 Préparation et exécution de l'expérience

5.3 Résultats

CHAPITRE 6

Analyses des résultats du projet

CHAPITRE 7

Gestion de projet

7.1 Diagramme de Gantt

7.2 Journal de travail

7.3 Analyse de la gestion de projet

CHAPITRE 8

Conclusion

8.1 Améliorations futures

8.2 Ressenti personnel

CHAPITRE 9

Sources

CHAPITRE 10

Annexes

10.1 Cahier des charges

10.2 Journal de travail

10.3 Plannification

10.4 Manuel utilisateur

10.5 Bibliographie