RAPPORT DE STAGE

Licence Informatique 3eme Année
2017/2018

Prise en main de la bibliothèque TensorFlow pour l'apprentissage automatique

- ***Stagiaire*** *: M. Layachi .IKNI*

- ***Tuteur Professionnel :*** *M. Vincent PAGÉ*
- ***Enseignant Référent :*** *Mme Suzy GAUCHER-CAZALIS*

Université des Antilles
Laboratoire LAMIA

Table des Matières

1 Introduction

Ce rapport présente le travail que j'ai effectué lors de mon stage au sein du laboratoire de recherche LAMIA, du 15 janvier au 10 février 2018. Ce stage, d'une durée d' un mois, a consisté à permettre aux chercheurs d'utiliser la bibliothèque TensorFlow pour des taches d'apprentissage automatique

Le plan de ce rapport est le suivant : A la suite de cette introduction, la section 2 présentera les généralités du stage : la structure d'accueil, le projet auquel j'ai été intégré, les objectifs de ma mission et l'organisation temporelle de mon travail. La section 3 présentera les différents logiciels utilisés. J'exposerais dans la section 4 les concepts théoriques qui sous-tendent l'utilisation de réseaux de neurones. Mon travail de programmation sera décrit dans la section 5 et les résultats obtenus feront l'objet de la section 6. Enfin, je conclurais sur les différents aspects de mon travail durant ce mois et les apports de ce stage.

Ce document est accompagné d'annexes présentant différents points que nous n'avons pas pu développer dans le corps du rapport. Ce rapport devrait également être accompagné d'un tutoriel au format pdf que nous avons rédigé durant le stage et auquel il sera fréquemment fait allusion dans ce rapport.

2 Présentation

Dans cette section, je présenterais la structure où j'ai effectué mon stage, le projet auquel mon stage a contribué, ainsi que les spécificités de ma mission. Enfin, on trouvera en fin de section une description de l'organisation temporelle de mon stage.

2.1 Le laboratoire d'accueil

Le laboratoire de Mathématiques Informatique et Applications (LAMIA) de l'Université des Antilles (UA) est une unité d'accueil reconnue par le Ministère de l'enseignement supérieur et de la recherche.

Le LAMIA compte une cinquantaine de membres répartis au sein de trois équipes internes. Certains chercheurs sont situés en Guadeloupe, d'autres en Martinique.

Les trois équipes sont :

- Équipe Mathématiques «analyse variationnelle, analyse numérique, EDP, analyse statistique, mathématiques discrètes».
- Équipe « Data analytics and big data gathering with sensors » .
- Équipe « Apprentissages Interactions Données » .

Au delà de ce découpage en équipes, certains chercheurs se regroupent sur des projets.

Durant ce stage, j'ai travaillé en Guadeloupe, sur le campus de Fouillole, essentiellement avec des membres de l'équipe AID, sur le projet "**Spiking Neurons**" qui sera décrit dans la section suivante. Mon encadrant était Monsieur PAGÉ Vincent, maître de conférence en Informatique.

2.2 Le projet de recherche Spiking Neurons

Le groupe de recherche **Spiking Neurons** de l'UA a pour but de comprendre comment faire fonctionner des **réseaux de neurones impulsionnels**. Dans ce groupe, on trouve 1 Professeur des Université, 4 Maitres de conférences et 1 ingénieur de recherche avec lesquels j'ai eu de nombreuses interactions.

Durant ce stage, 3 autres stagiaires travaillaient sur cette thématique en testant différentes configurations pour de la classification sur des bases de données d'exemples bien connues (IRIS, MNIST, FashionMNIST).

2.3 Présentation du stage

Pour ma part, je n'ai pas directement travaillé sur ces **Spiking Neurons**. Il s'agissait pour moi de fournir des classifieurs plus classiques pour pouvoir comparer les performances des Spiking Neurons et les performances de ces classifieurs classiques.

Mon tuteur m'a proposé d'utiliser la librairie TensorFlow pour développer ces classifieurs, pour les raisons suivantes :

- TensorFlow est une librairie OpenSource
- TensorFlow est la librairie utilisée par Google pour ses applications d'Intelligence Artificielle (traduction, recherche d'images, AlphaGo...).
- TensorFlow permet de mettre facilement en œuvre la plupart des techniques classiques et modernes d'apprentissage automatique.
- Les tutoriels présents sur le site officiel utilisent justement les bases de données utilisées par le groupe Spiking Neurons.

Mon stage consistait donc à comprendre les fondamentaux de l'apprentissage automatique, installer TensorFlow, suivre les tutoriels et enfin rédiger un tutoriel plus détaillé sur les points les plus difficiles pour que le groupe Spiking Neurons puisse utiliser cette librairies à l'avenir.

Le groupe avec lequel j'ai travaillé était composé de quatre stagiaires. Chaque personne du groupe s'occupait de différentes recherches concernant l'apprentissage automatique. Nous avons travaillé 35 heures par semaine dans un bureau commun. J'étais encadré par M. PAGÉ qui m'a fortement assisté pour la programmation et pour les différentes rédactions au cours du stage.

2.4 Organisation temporelle

Dès le départ de ce stage, il était prévu le fonctionnement suivant : chaque jour, je devais

- faire des tests de programmation en suivant le tutoriel officiel de TensorFlow (~4h)
- rédiger le tutoriel destiné au LAMIA (~2h)
- rédiger mon rapport de stage (~2h)

Tout ceci avait pour objectif d'atteindre tous les objectifs du stage sans négliger aucun des points importants pour moi et pour l'équipe **spiking neurons**.

Chaque semaine, mes collègues stagiaires et moi devions présenter nos travaux lors d'un séminaire occupant le vendredi après midi.

Dans ce contexte très collaboratif, nous avons également défini dès le départ l'environnement logiciel utilisé permettant de tester et échanger : des codes, des bases de données et des rapports ou des tutoriels. C'est cet environnement logiciel auquel est dédié la section suivante.

3 Environnement de travail

J'ai travaillé sur mon ordinateur portable personnel (Pc portable hp i7 , Windows 7 64bits) dans une salle offrant un accès réseau.

Il faut noter que le LAMIA dispose, par l'intermédiaire du centre de calcul de l'Université des Antilles (le C3I) d'un cluster de calcul. L'utilisation de TensorFlow n'était pas possible sur le cluster de calcul pour des raisons de librairies obsolètes (des essais sont en cours pour changer cela). Tous les tests ont donc été faits sur ma machine et le pc de M. PAGÉ (linux ubuntu récent, 64 bits)

Voyons donc l'ensemble des logiciels qu'il m'a fallu installer.

3.1 Programmation

Le couple TensorFlow / Python m'était imposé par mon tuteur, car il correspondait à la documentation officielle dont nous disposions sur TensorFlow (https://www.tensorflow.org/).

- **TensorFlow :** c'est une librairie de calcul dédiée à l'apprentissage automatique. On peut l'utiliser avec python, java, C,.... Dans notre cas, nous avons utilisé python. Pour l'installation, nous avons suivi les instructions du tutoriel officiel qui se trouve ici, sans difficultés : https://www.tensorflow.org/install/

- **Python :** langage de programmation interprété très utilisé en calcul scientifique pour sa puissance et sa souplesse. Nous avons aussi utilisés quelques modules classiques (numpy, scipy)

Pour programmer, il nous fallait un éditeur de texte quelconque, **Notepad ++** a fait l'affaire (il est gratuit et efficace).

Mon tuteur tenait également à ce que tout mon travail bénéficie d'un suivi de version de façon à pouvoir contrôler l'avancée de mes travaux, archiver les modifications faites et échanger des modifications avec moi. Pour cela, nous avons utilisé Git :

- **Git :** logiciel de gestion de versions décentralisé. Il est conçu pour être efficace tant avec les petits projets, que les plus importants. Git fonctionne de façon décentralisée, c'est-à-dire que le développement ne se fait pas sur un serveur centralisé, mais chaque personne peut développer sur son propre dépôt. Git facilite ensuite la fusion (merge) des différents dépôts.

Le LAMIA dispose, par l'intermédiaire du C3I d'un serveur GIT (http://lic3i.univ-antilles.fr) permettant d'avoir un dépôt central avec lequel nos dépôts locaux (le mien et celui de M. PAGÉ) pouvaient interagir (mettre à jour ou récupérer des mises à jour).

Enfin, si Git est efficace, son utilisation en ligne de commande est assez délicate. Nous avons donc utilisé une un logiciel offrant une interface graphique à Git. Mon tuteur m'a proposé **GitKraken**. (gratuit pour un usage non commercial et convivial).

3.2 Rédaction

Ce stage a comporté une grande partie de rédaction, j'insisterais donc ici sur les outils utilisés. Pour la rédaction de ce rapport, j'ai utilisé Microsoft Word. Pour la préparation des transparents de présentation hebdomadaires, j'ai utilisé Microsoft PowerPoint. Pour dessiner les graphiques présents notamment dans ce rapport, j'ai utilisé Libre Office Draw.

En revanche, pour la rédaction du tutoriel, mon tuteur regrettait que Word ne permette pas de contrôle de version simple. Le tutoriel a donc été rédigé en utilisant **LaTex**. LaTex fonctionnant sur une base de fichiers textes, chaque étape de rédaction pouvait être versionnée et intégrée au projet général avec Git.

Sous Windows, le couple (**MikTex** / **TexMaker**) pour rédiger en Latex est un classique auquel nous avons eu recours.

Cet environnement de stage décrit, nous pouvons nous intéresser aux concepts théoriques qu'il m'a fallu comprendre pour mener ma mission a bien.

4 Concepts théoriques

Mon sujet de stage consistait a utiliser une librairie permettant d'utiliser des réseaux de neurones pour classifier des bases de données bien connues. Une grande partie de mon stage a donc consisté a comprendre une partie des théories qui ces concepts. Les réseaux de neurones faisant partie des techniques de l'apprentissage automatique, il m'a donc fallu comprendre ce que sont :

- une classification.
- l'apprentissage automatique (Machine Learning)
- les réseaux de neurones

Cette section restitue ce que j'ai compris de ces notions.

4.1 La Classification

La **classification** est l'opération qui consiste a affecter une catégorie à un objet. Le **classifieur** est le programme qui va opérer la classification des objets. Sur chaque objet que l'on veut classifier, on dispose de mesures qui sont appelées ses **caractéristiques** (features). Compte tenu des ces caractéristiques, le classifieur choisira à quelle **classe** (quelle catégorie) appartient l'objet.

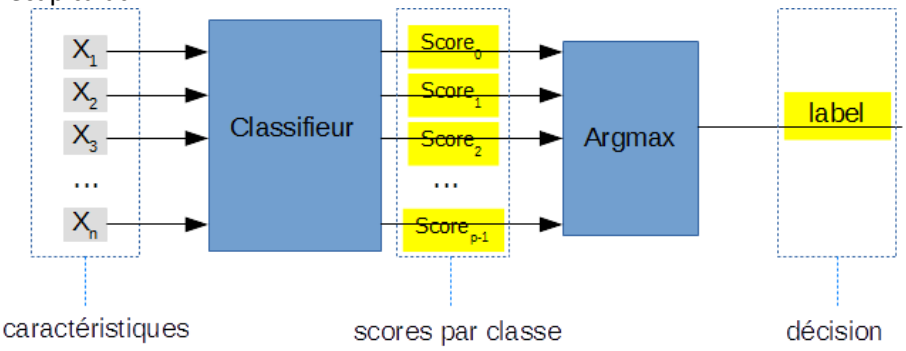
En général, les classes sont désignées par un nombre entier compris entre 0 et le nombre de classe moins un : le **label** de la classe.
A titre d'exemple, on peut essayer de deviner quel est le sexe d'un individu en fonction de son poids et de sa taille.

- Les caractéristiques d'un individu sont son poids et sa taille, par exemple (78, 1.80)
- Il y a deux classes possibles (Homme et Femme). Les sorties de notre classifieur devront donc être 0 ou 1 (avec 0 = Homme et 1 = Femme ou le contraire).

Dans la pratique, les classifieurs ont bien souvent autant de sorties que de classes. Chaque sortie correspond au score de la classe pour l'entrée qui lui est présentée. La classe finale choisie est la classe correspondant a la sortie la plus grande.

Pour bien comprendre ceci, reprenons notre exemple (Homme/Femme) : Un classifieur à qui l'on présente les caractéristiques (78, 1.80) pourrait avoir comme sortie [13.4, -2.7]. Ce s'interpréterait comme le fait que la classe 0 a obtenu un score de 13.4 alors que la classe 1 a obtenu un score de -2.7. La décision finale serait donc "0".

Voici un schéma récapitulatif :



Il nous reste à construire de "bons" classifieurs, ce qui fait l'objet de la section suivante

4.2 Apprentissage automatique (Machine Learning)

Si aux débuts de l'informatique, les chercheurs définissaient eux même comment devaient être calculés les scores, une autre stratégie est apparue vers les années 60 : l'apprentissage automatique. Dans ce cadre, on dispose d'exemples dont la classe est connue (**la base d'apprentissage**). Le classifieur effectue un calcul des scores qui dépend de **paramètres**. Le calcul des scores effectué est appelé **modèle du classifieur**.

Dans la **phase d'apprentissage**, on présente les exemples connus au classifieurs et celui ci modifie ses paramètres pour obtenir de meilleures performances (le bon label pour chaque exemple). Cette recherche des "meilleurs paramètres" du modèle fait appel à des techniques mathématiques d'**optimisation**. On choisit alors une quantité à minimiser et une technique d'optimisation pour trouver les paramètres les plus efficaces.

Au cours de ce stage, nous avons testé les modèles suivant, qui seront décrits dans la section suivante :

- réseau monocouche
- réseau multi couche.

Nous avons également pu tester différents critères à minimiser (taux moyen d'erreur, entropie croisée) et nous avons utilisé différentes techniques de minimisation (la descente du gradient, la méthode Adam), non décrites ici.

A cette phase d'apprentissage succède une **phase d'évaluation** dans laquelle on mesure les performances du classifieur. Pour mesurer ses performances, on peut calculer sa **précision** (son nombre moyen de bonnes réponses). Il est évalué sur la base d'apprentissage, ce qui donne sa **précision en apprentissage**, mais aussi sur des exemples qu'il n'a jamais vu, mais dont la classe est connue (**la base de généralisation**), ce qui donne sa **précision en généralisation**.

Si ses performances sont correctes, le classifieur peut être utilisé en **phase de prédiction** : on lui donne de nouveaux exemples et il donne les classes auxquelles ces exemples correspondent selon lui.

Concentrons nous maintenant sur le concept de **neurone** et de **réseau de neurones**, au cœur de notre modèle.

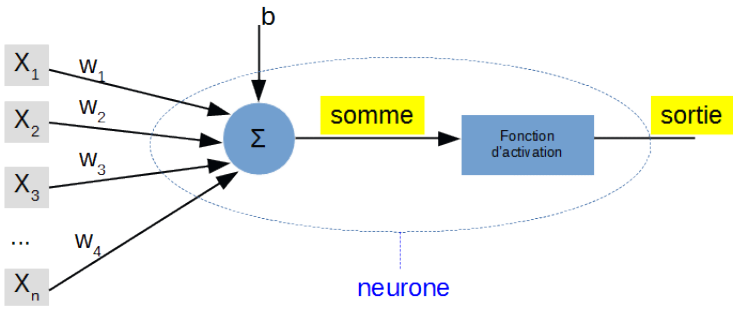
4.3 Modèle d'un neurone

Un **neurone** est l'unité élémentaire de traitement d'un réseau de neurones. Il est connecté à des sources d'information en entrée (X) et renvoie une information en sortie(Y). les paramètres de chaque neurone sont des réels (**les poids**) W et b.

Chaque poids est simplement un coefficient "w_i" lié à l'information " X_i ". La i -ème information qui parviendra au neurone sera donc en fait " $\mathbf{W_i} * \mathbf{X_i}$" . le coefficient de biais(b) est un "poids" supplémentaire.
Un neurone à **N** entrées a donc N+1 paramètres à régler.
Le neurone calcule tout d'abord une somme dont l'équation est la suivante :

$$Somme = \sum_{i=1}^{n} (\mathrm{Wi} * Xi) + \mathrm{b}$$

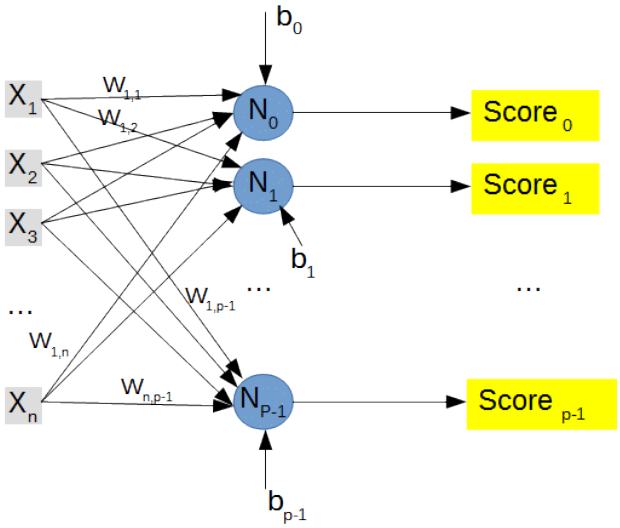
Cette somme passe ensuite à travers une **fonction d'activation** qui donne la sortie du neurone. Cette fonction d'activation pourra être l'identité (neurone linéaire) ou plus complexe (sigmoide) conduisant à des neurones non linéaires.



4.4 Réseau monocouche

Pour faire un classifieur utilisant des neurones, le plus simple est de construire la structure suivante :

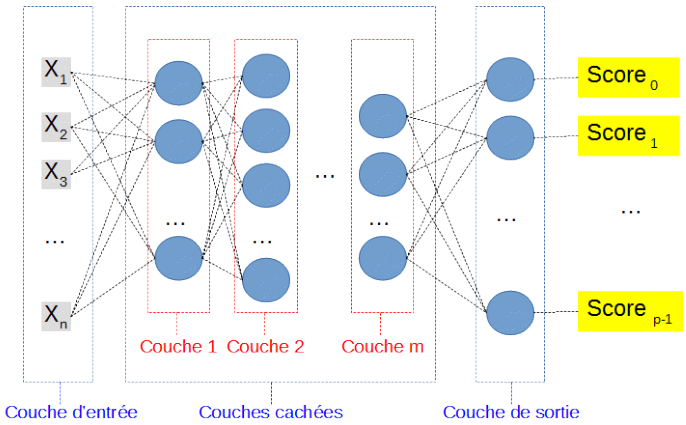
- un neurone par classe, dont la sortie est le score de la classe.
- chaque caractéristique des objets à classifier est une entrée de chaque neurone.



Dans ce qui suit, nous utiliserons la dénomination **réseau monocouche** pour désigner ce modèle, avec dans chaque neurone une fonction d'activation linéaire.

4.5 Réseau multicouche

Pour tirer parti des réseaux de neurones, une stratégie bien connue consiste à empiler des couches entre l'entrée et la sortie pour un modèle dont voici le schéma :



L'optimisation des poids de tels réseaux est assez complexe et est au cœur de l'effort de recherche des dernières années ayant conduit à la technique actuelle en vogue : les **réseaux de neurones profonds** ou DNN, dont le fonctionnement ne sera pas détaillé ici. Néanmoins, lorsque nous utiliserons des réseaux multicouche, ils seront basés sur des réseaux de neurones profonds dont le fonctionnement sera pris en charge par la librairie.

Nous pouvons donc maintenant passer à la mise en œuvre de ces réseaux avec la librairie TensorFlow.

5 Travail de programmation

Tout le code que nous avons produit est détaillé dans le tutoriel que nous avons rédigé avec mon tuteur. Le code que nous avons produit correspond aux tutoriels officiels de TensorFlow (https://www.tensorflow.org/get_started/), modifiés pour correspondre aux besoins du laboratoire. Dans ce rapport, nous nous limiterons à expliquer quelques notions primordiales de TensorFlow.

5.1 Concepts fondamentaux de TensorFlow

Tout d'abord, on peut utiliser TensorFlow de deux façons :

- En utilisant des algorithmes pré-codés (réseaux profonds par exemple). C'est ce que ce rapport appellera le mode **boite noire**
- En maitrisant presque toutes les étapes du calcul. C'est ce que l'on appellera le mode **boite blanche**

En mode **boite blanche**, TensorFlow s'appuie sur des concepts de programmation très différents d'une programmation standard python. Pour bien les comprendre, prenons un exemple : un programme qui prend une valeur réelle (x), calcule une valeur y = W*x+b. W et b sont des valeurs réelles que notre programme sera appelé a modifier plus tard. le code correspondant en python est le suivant:

```
x = 2
W = 0.3
b = -0.3
y = W*x+b
print(y)
```

Néanmoins, dans le contexte de notre programme, W,b, x et y jouent des rôles très différents :

- x est une entrée
- W et b sont des valeurs modifiables
- y est calculé a partir de x, W et b

La programmation en TensorFlow, met en place cette différence.

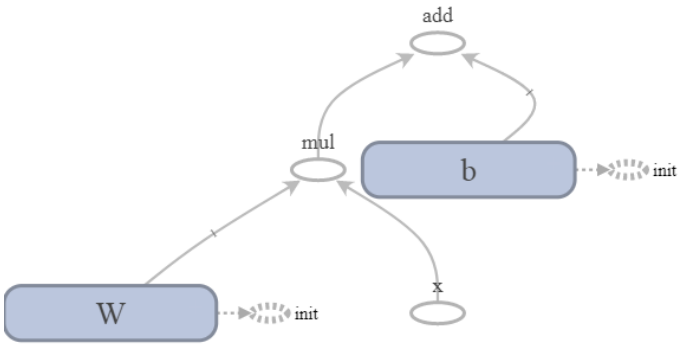
- x sera appelé un **placeholder** , (en deux mots : une variable dont on promet qu'on lui donnera une valeur au moment du run).
- W et b seront définis comme des variables.
- y sera défini implicitement par l'équation de calcul .
- notons qu'il existe aussi la notion de constante, non présentée ici, mais facile a appréhender.

Le code correspondant en TensorFlow est le suivant :

```
import tensorflow as tf
x = tf.placeholder(tf.float32)
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
y = W*x + b
```

De plus, un programme TensorFlow ne manipule pas des variables au sens traditionnel, mais explique les dépendances entres les différents éléments de notre programme (ce sont des **nœuds du graphes de calcul**).

Le graphe correspondant est représenté ci dessous pour information. On retrouve dans ce graphe les deux variables (W et b), le placeholder x, et un noeud add dont la sortie correspond à y.



Le reste d'un programme TensorFlow consiste à :

- construire le graphe de calcul a partir des informations précédentes.
- initialiser les variables W et b.
- lancer le calcul de y avec une valeur choisie pour x...

5.2 L'outil de visualisation TensorBoard

Lors d'une tache d'apprentissage automatique, il peut être utile de visualiser le graphe de calcul, mais aussi de voir l'évolution de l'apprentissage en observant les courbes de différents critères lors de la phase d'apprentissage. Pour cela, TensorFlow est accompagné de l'outil TensorBoard

Tout ceci se fait de la façon suivante:

- On ajoute du code dans le programme TensorFlow de façon a sauvegarder les informations qui nous intéressent dans des fichiers.
- On lance ensuite TensorBoard en lui disant quels fichiers nous intéressent.
- TensorBoard lance un serveur Web local permettant de consulter la page de visualisation qu'il a généré.

Quelques exemples de visualisations sont données dans l'annexe 9.1

5.3 Les programmes réalisés

Nous avons implémenté, sur trois bases de données différentes (IRIS, MNIST, Fashion MNIST) les classifieurs suivants :

- réseau monocouche en boite blanche.
- réseau multicouche en boite noire.

Nos programmes sont également en mesure de sauvegarder un réseau à l'issue de la phase d'apprentissage pour qu'un autre programme charge ce réseau pour des phases d'évaluation ou de prédiction.

Ce travail de programmation nous a pris la majeure partie du stage, car il fallait s'approprier rapidement toutes sortes de spécificités de TensorFlow. Par exemple, il a fallu adapter le code aux formats de fichiers différents des

bases de données (csv ou idx), ou adapter les mécanismes de sauvegarde et de chargement aux cas boites noires / boite blanche.

Le lecteur intéressé par mon travail de programmation pourra se reporter au tutoriel que nous avons rédigé et qui est fourni en pièce jointe de ce rapport. Il pourra également se faire une idée du travail réalisé en observant la visualisation de gitKraken de l'évolution du projet qui figure dans l'annexe 9.2.

La liste des classifieurs pré-codés de TensorFlow figure dans l'annexe : 9.3

6 Bases de données et performances

Un des objectifs de ce stage était de fournir, pour différentes bases de données bien connues en apprentissage automatique, les performances obtenues par des algorithmes de référence. Ces performances serviront d'étalon pour évaluer ce que sont capables de faire les spiking neurons que développe le LAMIA.
Cette section présente tout d'abord les bases de données utilisées puis résumera les performances obtenues.

6.1 Bases de données utilisées

6.1.1 IRIS

Cette base de donnée est téléchargeable ici : http://archive.ics.uci.edu/ml/datasets/Iris. La base comprend 150 échantillons de trois espèces d'iris (*Iris setosa, Iris virginica* et *Iris versicolor*). Quatre caractéristiques ont été mesurées à partir de chaque échantillon : la longueur et la largeur des pétales et la longueur et la largeur des sépales (les supports des pétales). Ces 4 chiffres sont donnés en centimètres.
De notre point de vue, cette base se résume ainsi :

- Entrées : un vecteur de dimension 4
- nombre de classes : 3
- nombre d'exemple en apprentissage 120 échantillons
- nombre d'exemple de généralisation 30 échantillons

6.1.2 MNIST

Cette base de donnée est téléchargeable ici : http://yann.lecun.com/exdb/mnist/. C'est une base de données de chiffres écrits à la main. Chaque échantillon est une image en noir et blanc, normalisée et centrée de taille 28x28 pixels. Voici quelques exemples de ces chiffres.

De notre point de vue, cette base se résume ainsi :

- Entrées : une matrice de dimension 28x28 (ou un vecteur de dimension 784)
- nombre de classes : 10
- nombre d'exemple en apprentissage : 55 000 échantillons
- nombre d'exemple de généralisation : 10 000 échantillons

6.1.3 Fashion MNIST

Cette base de donnée est téléchargeable ici : https://github.com/primaryobjects/fashion. Fashion MNIST est une base conçue pour avoir des caractéristiques identiques à celles de MNIST, dans un domaine totalement différent.
FashionMNIST est une base de données d'images de vêtements. Chaque image est en niveaux de gris, normalisée et centrée de taille 28x28 pixels. Il faut reconnaitre la catégorie de vêtement parmi 10 classes (chemise, pantalon, chaussure, ...).

Voici quelques exemples de ces images.

Tout comme MNIST, cette base se résume ainsi :

- Entrées : une matrice de dimension 28x28 (ou un vecteur de dimension 784)
- nombre de classes : 10
- nombre d'exemple en apprentissage : 55 000 échantillons
- nombre d'exemple de généralisation : 10 000 échantillons

6.2 Performances

Comme annoncé précédemment, cette section résume les résultats obtenus pour les modèles monocouche et multicouche.

Nous n'avons pas cherché à optimiser ces réseaux (en particulier pour le multicouche, on pourrait faire varier le nombre de couches et le nombre de neurones sur chaque couche). Ces points spécifiques d'implémentation varient d'une base à l'autre (IRIS, MNIST) en fonction des sources originales que nous avons adaptées à nos besoins. Il s'agissait surtout de fournir une performance indicative des techniques proposées. Nous avons simplement vérifié la convergence de l'apprentissage avec TensorBoard.

Dans le tableau qui suit, le terme Précision A désigne le taux moyen de bonne reconnaissance sur la base d'apprentissage. Le terme Précision G désigne le taux moyen de bonne reconnaissance sur la base de généralisation. Le temps mesuré correspond au temps cumulé de lecture de la base de données, d'entrainement du réseau et d'évaluation sur la base d'apprentissage puis sur la base de généralisation. Ce temps n'a aucune valeur scientifique, il sert juste a donner un ordre de grandeur de la durée de nos programmes (entre 10s et 1mn)

	Monocouche			Multicouche		
	Précision A	Précision G	temps	Précision A	Précision G	temps
IRIS	0.991667	0.96666664	~ 15 s	0.991667	0.966667	~ 30 s
MNIST	0.91425455	0.9153	~ 12 s	0.997964	0.981500	~ 40 s
Fashion MNIST	0.7976364	0.7792	~ 10 s	0.910455	0.881100	~50 s

En résumé :

- Le multicouche est plus efficace.
- Avec TensorFlow, nous avons été en mesure de créer un classifieur sur chaque base testée avec des précisions en généralisation supérieures à 88%, et de les entrainer en moins d'une minute.

7 Conclusion

Ce stage s'est avéré très intéressant et très enrichissant pour moi sous beaucoup d'aspects.

Tout d'abord, et directement exploitable dans ma vie quotidienne : j'ai amélioré mes connaissances en bureautique en utilisant de nouveaux logiciels comme latex pour rédiger le tutoriel , libre office pour construire mes schémas, GIMP pour modifier les images ou gitKraKen pour le contrôle de version

J'ai également pu acquérir des compétences techniques en programmation python, que je ne connaissais pas. J'ai aussi appris à utiliser des librairies externes (ici TensorFlow), et surtout fouiller dans leur documentation.

Dans ce stage j'ai également acquis des connaissances théoriques sur l'Intelligence Artificielle, et en particulier sur l'apprentissage automatique et les réseaux de neurones.

J'ai également amélioré mes compétences méthodologiques liées à la rédaction de rapports, la préparation de transparents ainsi qu'à la présentation de mes comptes-rendus de travail lors des séminaires qui avaient lieu chaque semaine. J'espère que ce rapport témoigne de ces progrès.

Ce stage m'a permis d'entrevoir en quoi consiste la profession d'enseignant-chercheur. En particulier, j'ai apprécié les séminaires qui m'ont permis de partager mes connaissances et d'en acquérir de nouvelles au travers du travail présenté par mes collègues et par un jeu de questions réponses au sein de l'équipe. Également, pendant quelques jours de la dernière semaine, nous avons adapté tout ce que nous avons fait à une base de donnée proposée par M.J. NAGAU. Il était très satisfaisant de voir fonctionner nos algorithmes dans un autre contexte...

Enfin, travailler dans le domaine de la recherche exige non seulement un savoir-faire technique complexe et varié mais aussi un savoir-être notamment la rigueur, la ponctualité, la concentration et la patience (le chercheur travaille de longues heures sur les programmes et attend les résultats).

Et si j'avais du temps, j'aimerai beaucoup essayer de m'orienter via un prochain stage dans le même sujet.

8 Remerciements

Mes remerciements s'adressent particulièrement à mon tuteur de stage, Monsieur PAGÉ Vincent, pour m'avoir accordé sa confiance dans ce projet. Je le remercie infiniment pour son accompagnement et son suivi, ainsi que tout le soutien et toute l'aide qu'il m'a apporté afin de me permettre de mieux cerner le travail à réaliser tout au long du stage.

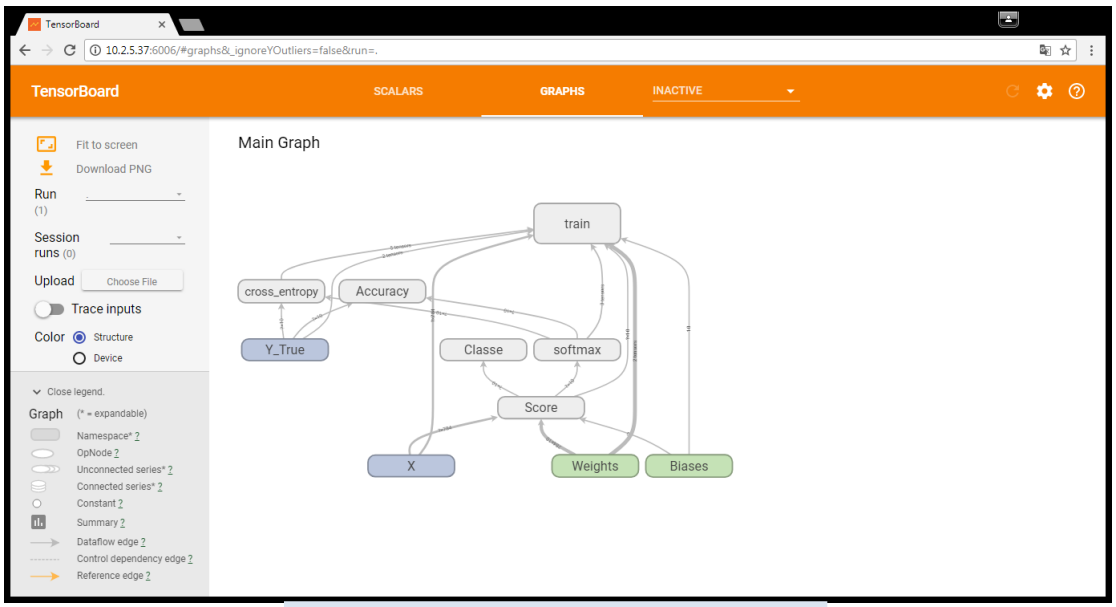
Aussi, je remercie ma responsable à l'UA, Madame GAUCHER-CAZALIS Suzy, de m'avoir proposé ce stage.

Enfin, merci à toute l'équipe LAMIA pour son accueil, sa disponibilité et sa bonne humeur permanente.

9 Annexes

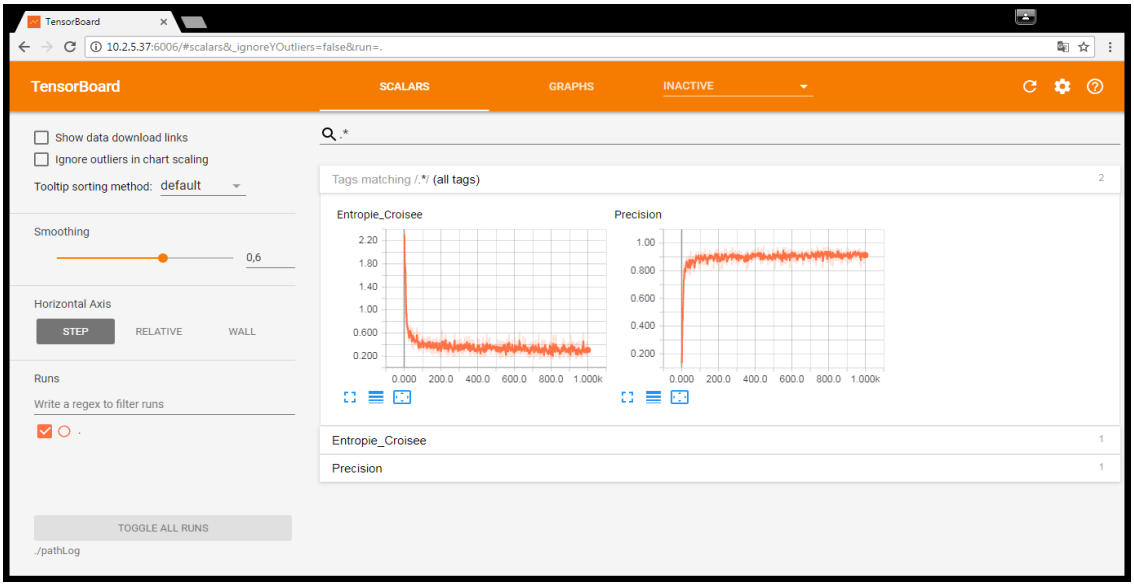
9.1 Exemples de visualisation TensorBoard

Ci dessous, on trouvera deux copies d'écran de la visualisation proposée par TensorBoard pour une tâche de classification avec un réseau monocouche sur la base MNIST. TensorBoard permet en particulier de visualiser le graphe de calcul de notre réseau.



Graphe d'un réseau monocouche

Par ailleurs, TensorBoard permet aussi de suivre l'évolution de certains paramètres lors de l'apprentissage. La figure ci-dessous, présente l'évolution de l'entropie croisée et de la précision lors des 1000 itérations de notre apprentissage.

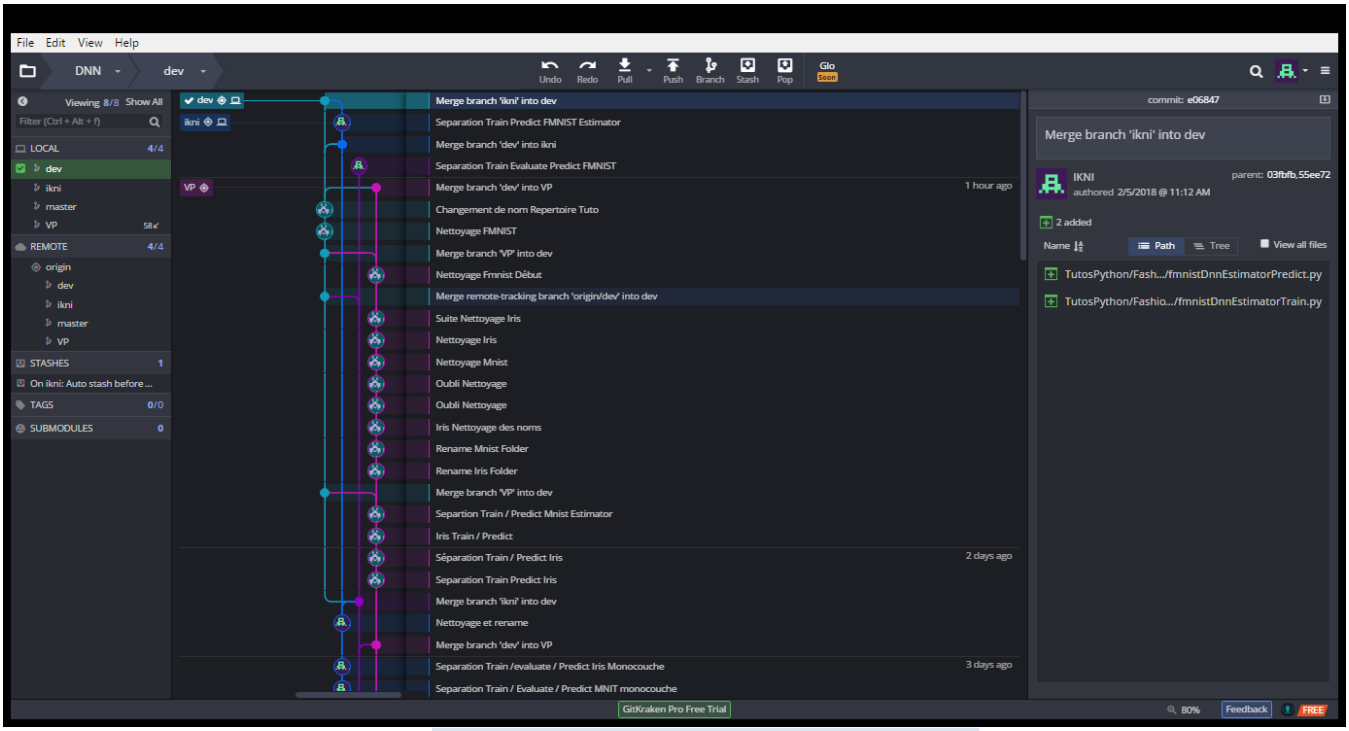


Suivi des performances lors de l'apprentissage

9.2 Exemples de visualisation gitKraKen

GitKraken est une interface graphique pour Git. Il permet une visualisation graphique de l'avancée d'un projet, commit par commit, mais aussi la visualisation des différentes branches de travail. Ci dessous, un exemple de cette visualisation pour notre projet. On retrouvera les branches :

- master : la branche principale qui doit toujours être fonctionnelle
- dev : la branche de travail
- Ikni : la branche dans laquelle j'ai travaillé
- VP : la branche de travail de M. PAGÉ



Travail réalisé dans chaque branche .

9.3 Liste des classifieurs en boite noire proposés par TensorFlow

TensorFlow propose principalement l'utilisation de réseaux de neurones profonds, implémentés par la classe DNNClassifier. On peut néanmoins trouver d'autres classifieurs, implémentés par l'équipe de TensorFlow ou par des contributions externes. Cette liste est difficile a reconstituer, la documentation étant très maigre sur le sujet. Voici la liste des classifieurs que nous avons pu croiser.

- tf.contrib.learn.LinearClassifier (Classifieur Linéaire)
- DNNClassifier (réseau multicouche profond)
- tf.contrib.learn.SVM (Machines à vecteur de support)

Et une variante :

- DNNLinearCombinedClassifier (un classifieur Linéaire joint a un DNN)

9.4 Webographie

- Tutoriels, procédures, explications sur le Site du TensorFlow
  https://www.tensorflow.org