*Université Clermont-Auvergne*



**Rapport Informatique et Technologies**

1ère Année de Master Informatique

Par

Alexis PLESSIER

Le 22/03/2021

I – TensorFlow

TensorFlow est un outil open source d’apprentissage automatique développé par Google et publié sous licence Apache en 2015. Il est écrit en C++ et Python et disponible sur toutes les plateformes, ce qui en fait l’un des outils les plus utilisés dans le domaine de l’IA. TensorFlow est capable d’exécuter de profonds réseaux de neurones, de la classification de caractères manuscrits, de reconnaissance d’image (notamment utilisé par le Pentagone), de modèle de séquences ou encore du traitement de langage. Il est également capable de faire de la prédiction de résultat sur différents modèles.

TensorFlow permet aux développeurs de créer des graphes de flux de données - des structures qui décrivent comment les données se déplacent dans un graphe, ou une série de nœuds de traitement. Chaque nœud du graphe représente une opération mathématique, et chaque connexion ou bord entre les nœuds est un tableau de données multidimensionnel, ou tenseur. Tout ceci est possible notamment grâce à son implémentation en Python de par ses objets mais en étant transformé sous forme binaire en C++ pour être plus performant.

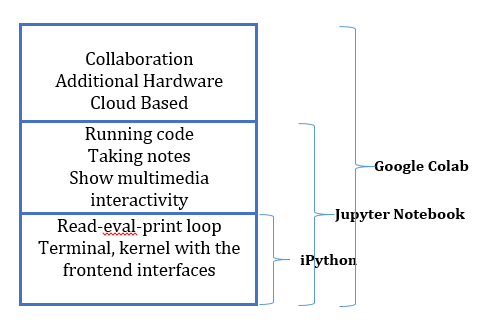
Son plus grand atout est également sa simplicité, l’utilisateur peut se concentrer sur son modèle et TensorFlow s’occupera des détails et d’éventuelles autres paramètres. En concurrence directe, il existe d’autres framework de Machine Learning qui ont tous leurs avantages et inconvénients comme PyTorch, CNTK ou encore ApacheMXNet.

II – Google Colab

Google Colab est l’équivalent d’un jupyter NoteBook (.ipynb) qui s’exécute dans le cloud Google totalement gratuit. Il ne nécessite aucune installation quelconque et autorise le partage et la modification simultanée. Il est surtout connu pour supporter différentes bibliothèques de Machine Learning qui sont plutôt faciles à charger dans Google Colab.

Il permet notamment d’écrire/charger et de sauvegarder des NoteBooks ou simplement du code python dans le Drive google mais possède également un lien direct avec GitHub où l’on peut importer/exporter des projets. Il permet aussi d’importer ses propres datasets et une intégration de PyTorch, TensorFlow, Keras ou encore OpenCSV, tous les outils pratique pour du Machine Learning. L’un des grands avantages est qu’il nous permet d’utiliser le GPU dédié de Google Colab et son accélérateur qui peut faire la différence dans certain projet demandant plus de calcul.

Voici la différence majeure entre Google Colab et ses concurrents :

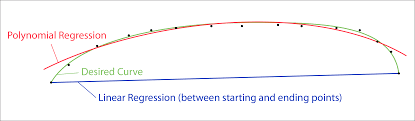


III – Curve Fitting

Le Curve Fitting désigne un type d’optimisation qui met en place un ensemble de paramètres optimaux pour une fonction définies pour correspondre au mieux à ses observations. A l’inverse de l’apprentissage supervisé, le Curve Fitting nécessite une fonction qui possède des entrées et des sorties connues. La fonction peut prendre plusieurs formes comme celle d’une ligne (régression linéaire), celle d’une ligne courbée (régression polynomiale) ou autres.

Imaginons que nous avons un graphique avec un nuage de points. Notre volonté serait de trouver une fonction y qui approcherai au mieux tous les points présents. Pour cela, nous allons essayer de réduire l’erreur en modifiant nos paramètres. L’erreur est calculée en faisant la différence entre le résultat à la sortie de notre fonction et la sortie observée sur le graphique pour une même entrée.

Pour s’approcher de la fonction la plus probable , il faut déceler une tendance comme une ligne droite, courbée, sinusoïdale, … . Les paramètres sont ensuite déterminés selon la fonction de la courbe probable (Exemple : y=a1x1+a2x2+a3x3+b pour 3 points et une courbe linéaire, on cherche ici les a et b).



(ici, la régression linéaire est inutile car trop éloignée du modèle

et des points que l’on a, donc régression polynomiale)

IV – TensorFlow Callbacks

Le TensorFlow Callback est une fonction ou un groupe de code qui s’exécute à un moment précis lors de l’entrainement de notre modèle sous TensorFlow.

Avec des modèles de plus en plus complexe et nécessitant plus de ressources, le temps d’entrainement de ces derniers a largement augmenté ces dernières années, et c’est normal qu’il puisse prendre plusieurs heures. Après avoir fixé les paramètres d’apprentissage qui nous intéressent (taux d’apprentissage, optimiseurs, loss, …), le modèle s’entraine. A partir de ce moment, il n’y a aucun moyen d’arrêter le processus ou même de modifier les paramètres tant que le modèle n’a pas fini. Il peut arriver qu’après des heures, nous voulons modifier des paramètres selon l’évolution du modèle. Les Callbacks servent à ça, c’est-à-dire qu’a un moment bien défini dans l’apprentissage, nous avons le choix de modifier des paramètres ou autre ou de le laisser continuer. Cela peut être très utile pour corriger une erreur de notre part ou encore implémenter une nouvelle directive.

V – Image Classification

La classification d’image est quelque chose de simple pour un Homme mais s’avère vite compliqué pour une machine et est vite devenu une discipline importante comme la vision par ordinateur. Il y a potentiellement autant de classes que d’images dans un dataset d’entrainement et les classer à la main serai une tâche fastidieuse c’est pourquoi on a décider de les automatiser. Quelques exemples seraient la détection de cancer sur X-ray ou encore déterminer le nombre qui est écrit.

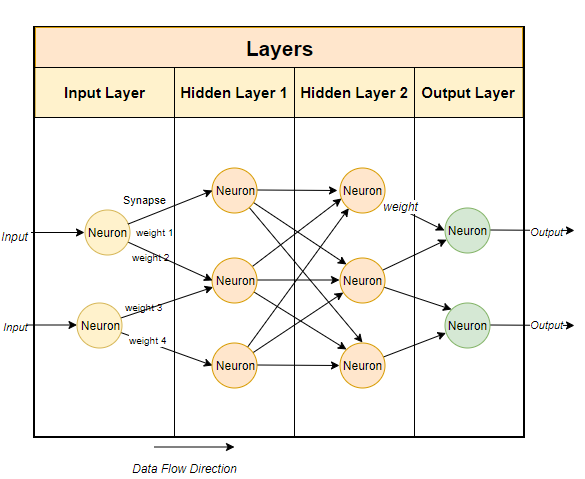
Une classification se caractérise en 4 étapes majeures :

1. Préparation de l’image : L’image doit souvent être retravaillé car elle présente des défauts de forme ou de couleurs et risque de tromper le modèle que l’on entraine. Ceci est également utilise pour avoir des images avec des paramètres similaires et donc faire ressortir de possibles différences.
2. Détection d’un objet : Détecte un objet sur l’image et détermine sa position pour pouvoir l’identifier.
3. Extraction des paramètres et entrainement : On cherche les paramètres les plus précis et unique qui puissent se référer à une classe d’image qui existe et correspondent à l’image pour aider le modèle à différencier les classes à l’avenir.
4. Classification de l’objet : Une fois le modèle entrainé, il va être appliqué sur l’image et va appliquer les paramètres de chaque classe pour déterminer à laquelle l’image est la plus susceptible d’appartenir.

VI – Weight Comparaison

Les poids et biais sont des variables du modèle qui sont mises à jour pour améliorer la précision du réseau. Un poids est appliqué à l’entrée de chacun des neurones pour calculer une donnée de sortie.

Les réseaux de neurones mettent à jour ces poids de manière continue. Il existe donc une boucle de rétro-action mise en œuvre dans la plupart des réseaux de neurones.



VII – Transfer Learning

Le Transfer Learning désigne l’ensemble des méthodes qui permettent de transférer les connaissances acquises à partir de la résolution de problèmes donnés pour traiter un autre problème.

Bien souvent, les modèles utilisés dans ce domaine nécessitent des temps de calcul élevés et des ressources importantes. Or, en utilisant des modèles pré-entraînés comme point de départ, le Transfer Learning permet de développer rapidement des modèles performants et résoudre efficacement des problèmes complexes en Computer Vision ou Natural Language Processing, NLP.

Le Transfer Learning repose sur une idée simple, celle de réexploiter les connaissances acquises dans d’autres configurations (sources) pour la résolution d’un problème particulier (cible). Dans ce contexte, on peut distinguer plusieurs approches selon ce que l’on souhaite transférer, quand et comment réaliser le transfert.

L’architecture des modèles de Deep Learning se présente très souvent sous la forme d’un empilement de couches de neurones. Ces couches apprennent différentes caractéristiques en fonction du niveau où elles se situent. La dernière couche (généralement une couche entièrement connectée, dans le cas de l’apprentissage supervisé) est utilisée pour obtenir la sortie finale. L'idée est ici de rajouter une sortie par exemple. Au lieu de faire réapprendre tout le modèle avec une sortie en plus, on va utiliser le modèle déjà entrainé et rajouter notre sortie.

VIII - Time series forecasting with LSTM

Le LSTM (Long Short-Term Memory) est une architecture basée sur les réseaux neuronaux récurrents (RNN) qui est largement utilisée dans le traitement du langage naturel et les prévisions de séries temporelles.

Le LSTM corrige un énorme problème dont souffrent les réseaux neuronaux récurrents : la mémoire courte. À l'aide d'une série de "portes", chacune dotée de son propre RNN, le LSTM parvient à conserver, oublier ou ignorer des points de données sur la base d'un modèle probabiliste.

Les LSTM aident également à résoudre les problèmes de gradient explosif et de gradient évanescent. En termes simples, ces problèmes sont le résultat d'ajustements répétés des poids lors de la formation d'un réseau neuronal. Avec des époques répétées, les gradients deviennent plus grands ou plus petits, et avec chaque ajustement, il devient plus facile pour les gradients du réseau de se composer dans les deux sens. Cette composition rend les gradients soit beaucoup trop grands, soit beaucoup trop petits. Si l'explosion et la disparition des gradients sont des inconvénients majeurs de l'utilisation des RNN traditionnels, l'architecture LSTM atténue fortement ces problèmes.

Une fois la prédiction effectuée, elle est réinjectée dans le modèle pour prédire la valeur suivante de la séquence. Avec chaque prédiction, une erreur est introduite dans le modèle. Pour éviter l'explosion des gradients, les valeurs sont "écrasées" par des fonctions d'activation (généralement) sigmoïde et tanh avant l'entrée et la sortie de la porte.