DEEP LEARNING

Usages & Solutions

Badr CHENTOUF

Alain ROUEN

SMILE

I.T IS OPEN

SOMMAIRE

CE LIVRE BLANC

Ce livre blanc traite des apports actuels de l'intelligence artificielle, au travers du Deep Learning et des réseaux de neurones. Sur la base principalement de capacités de vision automatisée et de traitement du langage naturel, l'IA et le Deep Learning apportent des plus-values business, avec de nouveaux usages et des performances améliorées dans différents secteurs tels que le médical, l'industrie, le retail ou encore l'énergie.

Ce livre blanc présente enfin un panorama des solutions informatiques disponibles de Deep Learning, avec un focus sur les solutions open source qui dominent le marché. Ces solutions, de plus en plus accessibles, étendent l'usage du Deep Learning grâce à leur forte dynamique et à leur large diffusion.

Auteurs

Ce livre blanc a été rédigé par Badr Chentouf, avec la contribution d'Alain Rouen.

Iconographies: Machine Learning by Noura Mbarki from the Noun Project

Edition 2018

INTELLIGENCE ARTIFICIELLE, C'EST LE MOMENT !

L'Intelligence Artificielle est sur tous les médias, véritable sujet hype du moment, buzzword indispensable des start-ups, mais aussi lame de fond qui change l'informatique et notre relation aux machines.

Les analystes prédisent un marché de 11 milliards de dollars en 2024, avec une croissance de 53% entre 2015 et 2020. Avec ce potentiel, il ne se passe pas une semaine sans voir un nouvel investissement de plusieurs millions dans une startup qui développe une application d'IA.

Ces investissements massifs viennent confirmer que l'Intelligence Artificielle est LA révolution technologique actuelle, qui va changer un grand nombre de domaines, de métiers, voire notre société toute entière.

Censée révolutionner le monde positivement, elle fait aussi peur, avec des capacités hors normes qui lui sont prêtées. Pourtant la réalité est bien loin des fantasmes du robot humanoïde intelligent remplaçant l'homme. L'Intelligence Artificielle permet de résoudre différentes problématiques, mais pas de reproduire l'intelligence humaine. En tout cas, pas encore.

Pour autant, les capacités de l'Intelligence Artificielle, boostées par les récentes capacités du Deep Learning, sont déjà très intéressantes et impactantes à de nombreux égards.

Ce livre blanc traite des potentiels de l'Intelligence Artificielle « faible », c'est à dire centrée sur une fonction unique, boostée par le Deep Learning, et qui fonctionne !

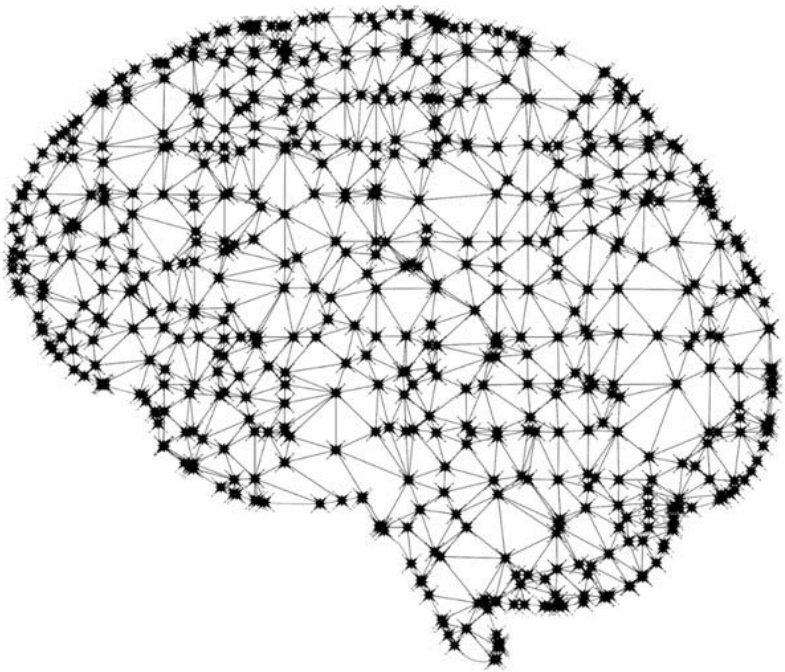
Les exemples opérationnels et d'IA sont nombreux, comme dans le domaine de la finance avec la détection de fraude ou les évaluations de notes de crédit. N'oublions pas le domaine de l'e-commerce, ou du e-marketing, très friand de Machine Learning, pour ses modèles prédictifs et de recommandation de produits.

Les exemples opérationnels et visibles d'IA sont nombreux

Les projets d'IoT bien sûr, et leurs centaines de capteurs déployés, qui génèrent un très grand nombre de données en continu, sont une source importante pour alimenter les machines, établir des modèles prédictifs et résoudre différents problèmes de fonctionnement.

L'IA ne concerne pas que les grands acteurs de l'internet, elle peut apporter dès maintenant un ROI dans de nombreux domaines.

Au sein du Machine Learning émerge depuis quelques mois le Deep Learning, l'apprentissage profond, à base de réseaux de neurones, sujet pas très récent mais longtemps délaissé.



Les nouvelles capacités du Deep Learning sur des informations non structurées - la vidéo, le son, le texte – et sur des modèles plus complexes ouvrent la possibilité de nouveaux usages, de performances améliorées, d'automatisation plus fortes, de nouveaux services jusqu'ici impossibles.

Les nouvelles capacités du Deep Learning ouvrent la possibilité de nouveaux usages

---

Plusieurs domaines sont encore en phase active de recherche, avec de grandes attentes, comme celui des véhicules autonomes, ou du diagnostic médical.

Mais des premiers usages de Deep Learning sont déjà visibles, disponibles pour tous, tels que de la reconnaissance vocale intégrée dans les assistants 'Google Home' ou 'Amazon Echo', la traduction instantanée sur Skype, la reconnaissance faciale 3D sur le dernier iPhone …

La finance, l'industrie, l'agriculture même, sont aussi des domaines d'application pertinents, pour lesquels le Deep Learning peut apporter de meilleures performances et plus d'automatisation.

D'ores et déjà, l'état de l'art du Deep Learning est applicable à de nombreux secteurs, de l'industrie à la finance en passant par l'agriculture

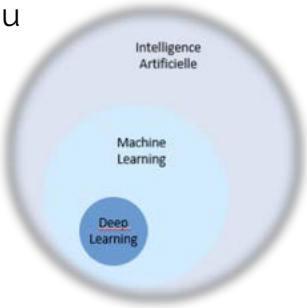
---

Des barrières à l'entrée subsistent, mais grâce à la dynamique open source, les solutions n'ont jamais été aussi performantes et accessibles.

Chaque entreprise doit aujourd'hui se demander comment elle peut utiliser l'IA pour innover, apporter de nouveaux services, améliorer ses performances, réduire ses coûts …

DEEP LEARNING ?

Commençons par expliquer les concepts et méthodes du Deep Learning.



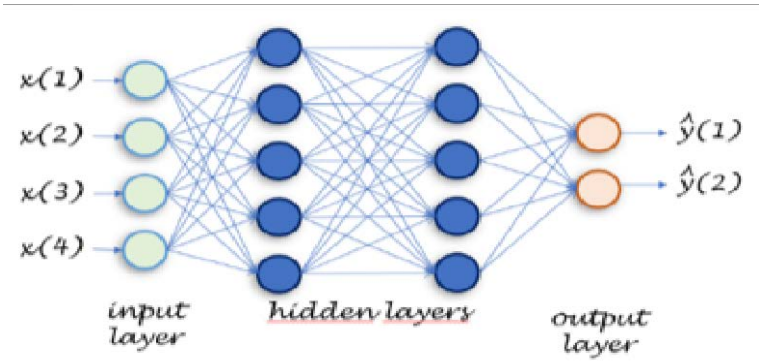
Le Deep Learning est un sous-domaine du Machine Learning. Le Machine Learning, i.e. l'apprentissage machine, est l'apprentissage automatisé par des machines de modèles qui peuvent ensuite être réutilisés pour réaliser des prédictions, classifications, …

Plusieurs méthodes algorithmiques sont disponibles dans le Machine Learning, tels que le 'Random Forest', le 'Gradient Boosting Machines', ou les modèles linéaires. Nous n'irons pas plus loin ici sur ces modèles de Machine Learning, pour nous concentrer sur le Deep Learning qui ouvre un champ d'application nouveau.

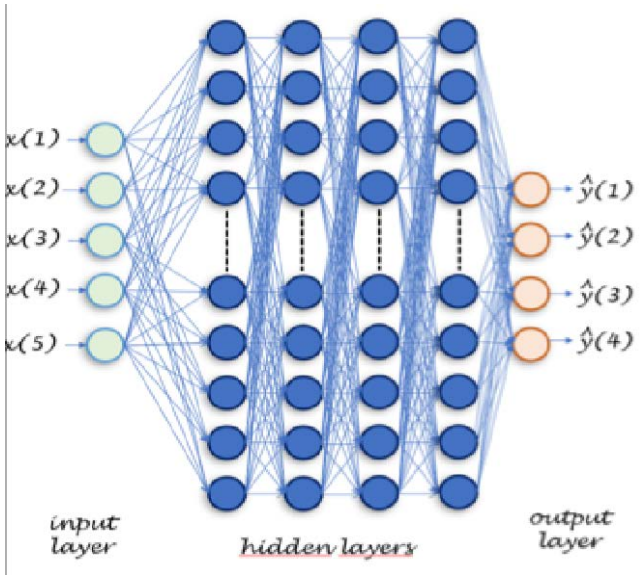
Le Deep Learning est donc une des méthodes possibles du Machine Learning et se base sur les Réseaux de Neurones (« RN » ou « NN » pour Neural Network en anglais), qui permettent de réaliser des modèles bien plus complexes.

Les réseaux de Neurones ne sont pas un domaine récent. Les scientifiques s'y intéressent depuis des années, et les théories mathématiques sous-jacentes sont connues. C'est avec le développement concomitant du big data, avec des méga sources de données disponibles, et de celui de la puissance informatique que le Deep Learning a pu se développer et atteindre les niveaux de performance actuels.

Au lieu de construire des réseaux de neurones à 3 couches de quelques neurones, et de le faire apprendre sur 1000 exemples, on peut maintenant aller à des dizaines de couches, avec chacune des centaines de neurones, et utiliser des dizaines de millions d'exemples pour l'apprentissage.



Réseau de Neurones Simple



Réseau de Neurones Profond

D'un point de vue mathématique, on est dans le monde des matrices à MxN dimensions, dont les éléments correspondent aux valeurs de chaque neurone, transmettant l'information aux neurones suivants tout en appliquant des paramètres de multiplication/addition.

Des améliorations et nouvelles méthodes sont trouvées en continu

Et au-delà de cette disponibilité récente de nouvelles giga sources de données, le Deep Learning fait l'objet d'investissements importants en recherche, et des améliorations sont trouvées régulièrement permettant d'aller encore plus loin dans la profondeur des réseaux, et donc dans la complexité des problèmes à résoudre.

Citons par exemple les nouvelles fonctions d'activations, plus performantes comme ELU (2015) ou SELU (2017), améliorant les résultats de la fonction RELU pour des réseaux à grande profondeur. Ou les nouvelles méthodes de régularisation comme le 'Dropout' (2012), qui peut améliorer la précision des modèles. Ou bien les travaux sur les Capsule Network (2017), une nouvelle architecture de RN qui apporte des informations d'organisation spatiale d'une image.

Pourtant déjà efficient dans de nombreux domaines, on est aussi très certainement encore au début des capacités du Deep Learning.

Ce livre blanc décrit ci-après l'apprentissage supervisé, l'apprentissage par renforcement et l'apprentissage non supervisé. Ils correspondent à des situations de départ différentes, selon les données disponibles, à leur attribution d'un label ou non.

Le plus utilisé est aujourd'hui l'apprentissage supervisé, grâce au big-data et à la nouvelle quantité de données disponible. L'apprentissage par renforcement suscite de nombreux espoirs à court-terme, et l'apprentissage non supervisé probablement à moins court-terme.

APPRENTISSAGE SUPERVISE

Prenons l'objectif de savoir reconnaitre un chat sur une image. Pour cela, il faut constituer un jeu de données avec des millions d'images différentes, avec et sans chat, avec différents angles de vue, et pour chaque image la bonne réponse.

On peut alors « entrainer » le réseau de neurones sur ces images, déterminer les meilleurs paramètres possibles pour réduire le taux d'erreur (et oui, un réseau de neurones n'est pas précis à 100%), puis le réseau de neurones devient capable de déterminer sur une nouvelle image s'il y a un chat ou non.

Pourtant personne n'a expliqué à la machine ce qu'était un chat ni ce que sont ses caractéristiques physiques. Pour autant, le RN est bien capable de déterminer s'il s'agit ou non d'un chat sur une image !

On parle d'apprentissage « supervisé » lorsque l'apprentissage se fait sur un jeu de données étiquetées, dont on connait les résultats, et que le modèle du RN va permettre de prédire. Pour apprendre, on l'aura compris, rien n'est possible sans ces données étiquetées, représentant des millions de cas différents.

Disposer d'une (très grande) quantité de données suffisante, correctement étiquetées est indispensable pour pouvoir entrainer des réseaux de neurones profonds et obtenir des modèles avec des précisions significatives.

Si reconnaitre un chat n'apporte pas beaucoup à l'humanité, le principe du Deep Learning supervisé reste le même pour des choses plus complexes, comme reconnaitre des panneaux stop, des personnes, des produits, des défauts sur une pièce, … avec une caméra, le tout en temps réel :

- Constitution d'un jeu de données avec des milliers/millions d'exemples pour lesquels les résultats sont connus.
- Apprentissage sur un très grand jeu de données, avec un résultat déjà connu pour chaque élément de ce jeu de données.
- Élaboration du réseau de neurones permettant la prédiction la plus précise. Détermination du nombre de couches de neurones nécessaires, des fonctions mathématiques d'activation de chaque étape, optimisation et calcul de tous les paramètres du RN …
- Validation du modèle obtenu sur un jeu de données de test, plus réduit

Pour élaborer ces réseaux de neurones, les data scientists s'appuient sur des outils informatiques qui permettent de les programmer simplement, et donc de passer de la théorie à la pratique le plus vite possible.

Un facteur important du Deep Learning à prendre en compte est le temps de mise en œuvre.

Le temps nécessaire pour collecter les tonnes de données, le temps nécessaire pour l'auto-apprentissage, le temps nécessaire pour déterminer le bon RN, le temps nécessaire pour le tester ….

Avec le volume de données, on peut compter le temps ici en jours, pour chaque étape !

Aussi tout ce qui permettra de gagner du temps dans ces étapes sera crucial pour la réalisation de votre projet de Deep Learning, réduisant les coûts humains, matériels et surtout les délais.

Tout ce qui permettra de gagner du temps à chaque étape sera déterminant pour la réalisation d'un projet de Deep Learning

APPRENTISSAGE PAR RENFORCEMENT

Dans l'apprentissage supervisé, on entraine le RN avec un jeu de données dont on connait le résultat, et pour lequel le RN n'est finalement que le moyen de le modéliser.

Mais comment faire lorsque ce jeu de données n'existe pas ? Lorsqu'il est impossible à constituer ?

L'apprentissage peu ou non supervisé propose différentes approches pour obtenir des modèles, sur la base de quelques données, ou d'un apprentissage itératif.

Un exemple significatif d'un apprentissage par renforcement est celui de « AlphaGo Zero », le programme qui est devenu le meilleur joueur du monde du jeu de Go

---

Dans l'IA, le Go a toujours constitué un challenge extrême – et jusqu'ici impossible – tant le nombre de combinaisons possibles est élevé.
En 2016, la version précédente, AlphaGo, avait battu le champion du monde 4 manches à 1, ce qui était déjà une réussite majeure de l'IA. Mais ses fondamentaux étaient différents. Le AlphaGo de 2016 avait appris entre autres à partir de millions d'exemples de parties jouées par des humains.

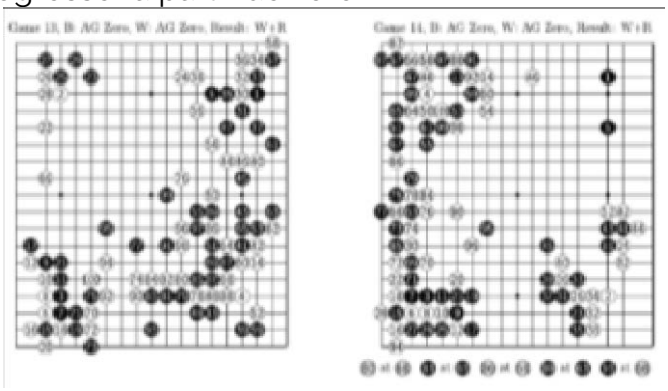
AlphaGo Zero a amélioré son jeu au Go encore et encore, sur des millions de parties, sans aide humaine.

Ce qui est assez spectaculaire avec AlphaGo Zero, c'est qu'il est parvenu à ce résultat en partant de zéro, seulement en s'entraînant à jouer seul, sur des millions de parties, mais sans partir d'exemples de parties déjà jouées comme dans le passé.

Et qu'il ne lui a fallu que quelques jours d'apprentissage pour y arriver !

AlphaGo Zero a été développé par Google DeepMind en utilisant la solution open source TensorFlow que nous verrons plus loin.

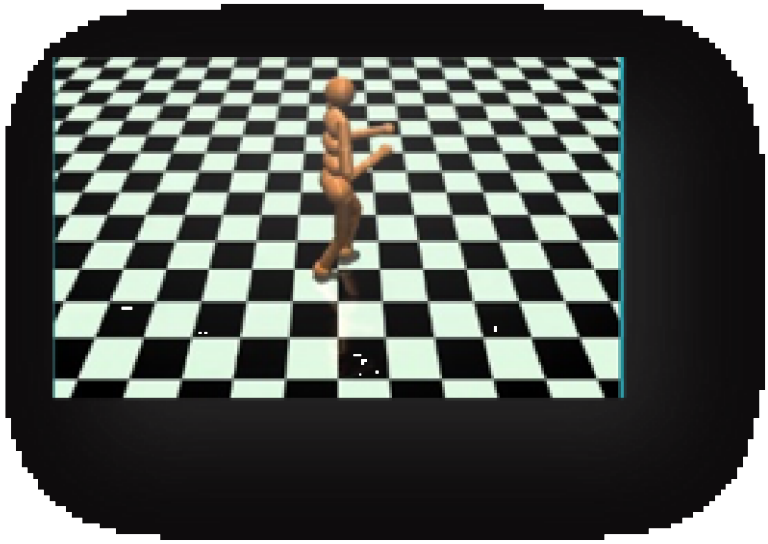
A titre d'exemple, une implémentation pour le jeu « Puissance 4 », - un jeu un peu plus simple que le Go … - est disponible sur GitHub, réalisée par David Foster. Elle est en Python, utilise Keras et TensorFlow. Le programmeur utilise la même architecture de réseaux de neurones que AlphaGo Zéro et comme lui, le programme apprend à jouer à Puissance 4 à partir de zéro. Il est intéressant de voir le peu de lignes de codes nécessaires, et de constater de visu l'apprentissage progresser à partir de zéro.



Ce principe d'apprentissage « par renforcement » a ensuite été appliqué avec succès à d'autres jeux comme les échecs. Pour autant, l'apprentissage par renforcement n'est pas la solution ultime du Deep Learning, car il n'est possible qu'à deux conditions : qu'il y ait une 'récompense' à l'issue d'une action (ce qui est possible dans un jeu, avec une meilleure position sur la partie, ce qui en soi est déjà difficile à mettre en œuvre) et qu'il soit possible de simuler des millions de cas, et d'échouer sur un très grand nombre.

DeepMind défend l'idée que l'apprentissage par renforcement est généralisable à de nombreux domaines, et travaille sur des sujets complètements différents tels que le repliement des protéines.

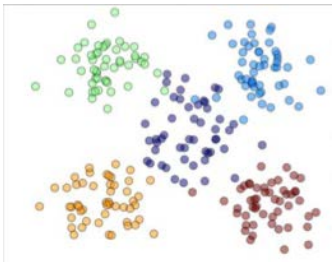
Beaucoup d'espoirs portent aujourd'hui sur ce type d'apprentissage, dans de nombreux domaines, avec l'apport notamment des environnements de simulation comme OpenAI Gym, qui peuvent s'appliquer y compris à des environnements physiques, pour la marche d'un robot humanoïde par exemple.



APPRENTISSAGE NON SUPERVISE

Le troisième type d'apprentissage est l'apprentissage non supervisé. Il correspond au cas où nous n'avons pas de données labellisées pour entrainer le modèle.

On souhaite que le modèle groupe les données pour les organiser en clusters homogènes, et ainsi pouvoir évaluer les distances entre clusters, et par exemple déterminer des données éventuellement anormales.



Un algorithme obtenu par apprentissage non supervisé ne va pas permettre de faire une prédiction d'un résultat, mais d'identifier une donnée anormale.

On comprend que ce type d'apprentissage va pouvoir être utile dans les cas de détection de fraudes pour lesquels il n'y a pas de pattern défini.

APPRENTISSAGE PAR TRANSFERT

Le transfert d'apprentissage est une méthode qui permet de s'appuyer sur des RN déjà modélisés, qui résolvent un problème particulier, et de les appliquer à un contexte proche.
Par exemple, dans un RN de reconnaissance d'images, les premières couches identifient les formes simples, et les couches suivantes vont identifier des formes plus complexes, et la dernière couche va faire la 'prédiction'.

Aussi la plupart des couches d'un modèle pré-entrainé sont utiles pour un nouvel usage, dans la mesure où la plupart des problèmes de 'Computer Vision' ont besoin des mêmes fonctions d'identification des formes simples.

Donc on peut réutiliser la plupart des couches d'un modèle pré-entrainé et juste remplacer la ou les dernières couches, utilisées pour faire la prédiction, en l'entrainant avec un jeu de données réduit, adapté au nouveau contexte d'application.

Dans l'exemple de la reconnaissance d'images, citons le modèle 'ResNet', réalisé par une équipe Microsoft, - qui comprend 152 couches de neurones ! - et qui a eu le meilleur score lors d'une compétition mondiale en 2015.
Ce modèle ResNet est directement disponible dans certaines librairies, librement utilisable pour être modifié par transfert d'apprentissage pour un contexte personnalisable.

Pour bien comprendre, prenons un exemple fourni par Google pour TensorFlow. Dans cet exemple, on part du modèle disponible 'MobileNet', capable de détection/classification dans une image, avec une précision de 84.9% sur le top 5. La précision de Mobilenet est moindre que celle de ResNet, mais avec un besoin en puissance réduit, ce qui le rend adapté avec des matériels de puissance réduite, comme les smartphones ou les caméras embarquées. La question sera de savoir de quelle précision a-t-on besoin …

Puis on ré-entraine le modèle à reconnaitre cinq types de fleurs, à partir d'un jeu d'images complémentaires. Le jeu de données de réapprentissage est ici constitué de 3670 images des 5 types de fleurs (roses, tulipes, tournesol, marguerite et pissenlit), correctement étiquetées.

Sur une VM Linux, ça donne

```
> sudo apt-get install git
> sudo apt-get install python-pip
> pip install tensorflow
> git clone https://github.com/googlecodelabs/tensorflow-for-poets-2
> curl
http://download.tensorflow.org/example_images/flower_photos.tgz |
tar xz -C tf_files
> python -m scripts.retrain \
  --bottleneck_dir=tf_files/bottlenecks \
  --model_dir=tf_files/models/ \
  --summaries_dir=tf_files/training_summaries/mobilenet_0.50_224 \
  --output_graph=tf_files/retrained_graph.pb \
  --output_labels=tf_files/retrained_labels.txt \
  --architecture="mobilenet_0.50_224" \
  --image_dir=tf_files/flower_photos
[…]
INFO:tensorflow:Final test accuracy = 90.9% (N=362)
```

Extrait de https://github.com/googlecodelabs/tensorflow-for-poets-2

Le modèle est ré-entrainé, avec une précision finale de 90.9% sur le jeu de test. L'utilisation du modèle ré-entrainé sur de nouvelles images les classera selon les catégories avec une certaine fiabilité :

```
> wget http://weknowyourdreams.com/images/daisy/daisy-03.jpg
> python -m scripts.label_image \
    --graph=tf_files/retrained_graph.pb  \
    --image=daisy-03.jpg

Evaluation time (1-image): 0.119s
daisy (score = 0.99071)
sunflowers (score = 0.00595)
dandelion (score = 0.00252)
roses (score = 0.00049)
tulips (score = 0.00032)
```

```
> wget https://www.lartefiori.it/39-large_default/les-tournesols-et-
les-roses-blanches.jpg
> python -m scripts.label_image \
    --graph=tf_files/retrained_graph.pb  \
    --image=les-tournesols-et-les-roses-blanches.jpg

Evaluation time (1-image): 0.115s sunflowers 0.6208324
roses 0.34903187
dandelion 0.013828798 tulips 0.009693168
daisy 0.006613833
```

Le transfert d'apprentissage a bien fonctionné, avec un modèle initial déjà efficient et un jeu de réapprentissage restreint, donc un coût réduit pour la mise en œuvre d'une détection d'objets spécifiques à un contexte nouveau.

Avant de vouloir créer un RN à partir de zéro, il faut vérifier s'il n'existe pas déjà un RN efficace pour un contexte général proche, qu'il suffirait de 'transférer' pour la problématique qui nous intéresse.

INTERPRETABILITE

Une fois le modèle validé, un modèle peut être appliqué et prédire des résultats ou faire des classifications.
Pour autant, est-ce que sa prédiction ou sa classification est compréhensible pour un humain ? Est-ce que je peux lui faire confiance ? Respecte-t-il des considérations éthiques ?

La complexité des modèles de Deep Learning est telle qu'il est difficile d'expliquer dans l'algorithme ce qui a déterminé la réponse du modèle.

On est face à une boite noire qui prend des décisions, potentiellement bonnes, mais surtout inexpliquées.

Pour reprendre l'exemple de AlphaGo, certains de ses coups ont été jugés surprenants sur le moment. Les experts ont cru à une erreur du logiciel et préconisaient même un autre coup. Pourtant ces coups surprenants lui ont permis de gagner la partie. Il avait donc raison ! Mais aucune explication ne venait avec la décision de l'algorithme. Juste la décision brute. Est-ce suffisant ? Dans d'autres cas plus critiques, comment laisser l'IA prendre une telle décision sans intervenir ?

On parle alors de XAI « Explainable Artificial Intelligence » ou de MLI, « Machine Learning Interpretation », un domaine encore nouveau et ouvert à la recherche.

Autre exemple, comment être sûr qu'un algorithme d'analyse de risque de crédit, basé sur les données multiples des personnes, ne va pas discriminer les personnes en utilisant des facteurs d'âge, de zone d'habitation défavorisée, … ?

L'Union Européenne s'est d'ailleurs penchée sur le sujet de la décision automatique dans la GDPR (General Data Protection Regulation), en apportant le droit aux individus de ne pas être soumis à une décision d'un algorithme. Certains l'interprètent comme le droit à avoir une explication du résultat d'un algorithme, dès lors qu'il impacte significativement une personne.

Beaucoup de domaines sont impactés, la conduite autonome, l'adtech, l'analyse de risque, le diagnostic médical …

LIME – pour Local Interpretable Model-Agnostic Explanations – est une librairie open source en Python qui donne des moyens d'expliquer les prédictions de n'importe quel modèle de machine learning de classification.

LIME fonctionne y compris pour des modèles de classification d'images ou de textes.

Par exemple, sur l'image ci-dessous sont colorisés les fragments d'images qui ont déterminé la réponse de l'algorithme, pour dire que c'est un chat (en vert) ou non (en rouge). On comprend ici que la réponse est donnée pour une bonne raison.

USAGES

Il est difficile d'être exhaustif sur les usages du Deep Learning tant cela peut transformer de nombreux pans de toutes les industries.

Être capable de remplacer les capacités de vision, d'interprétation d'image, de compréhension de texte, etc. Cela ouvre de nombreuses perspectives !

La technologie est récente, en pleine évolution, les capacités sont réelles et beaucoup d'usages restent à inventer.

Les principaux axes sous-jacents sont 'Computer Vision' - pour tout ce qui a trait à la reconnaissance d'image, de vidéos, à l'interprétation de ses images ou vidéos-, 'Natural Language Processing' - c'est-à-dire le traitement du langage naturel, qu'il soit oral ou écrit.

Les usages de ces techniques sont nombreux : classification d'image, détection d'anomalie, reconnaissance faciale, analyse de vidéos, chatbot, …

Savoir reconnaitre un objet dans une image, quelle que soit sa position, sa couleur, sa taille … Cela paraissait impossible il y a quelques années, et c'est maintenant une fonctionnalité quasiment banale grâce aux capacités du Deep Learning.

Des compétitions mondiales réunissant les meilleurs experts ont permis de faire avancer les modèles de reconnaissance, année après année. Et la bonne nouvelle, c'est que ces modèles de réseaux de neurones sont disponibles et utilisables par tous.

Les principaux modèles issus de la compétition ILSVRC sont listés ci-après. Pour chaque modèle, il existe des variations sur le nombre de couches utilisées, avec une incidence sur le taux d'erreur obtenu.

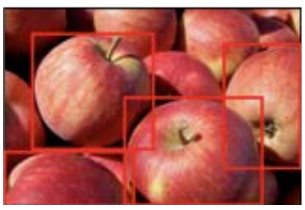
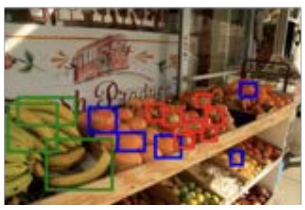
Fruit du travail cumulé de ces experts depuis plusieurs années, ces modèles sont complexes, avec plus d'une centaine de couches pour certains, et des millions de paramètres ! C'est donc un gain très appréciable de les avoir à disposition.

ILSVRC

Les critères des compétitions évoluent chaque année, mais globalement, il s'agit de reconnaitre parfaitement 200 catégories d'objets de premier niveau (accordéon, avion, pomme, banane, abeille, oiseau, …) sur un jeu d'images soigneusement choisies, avec des variations d'échelle, de prise de vue, … Pour bien visualiser la difficulté, ci-après quelques images extraites du jeu d'image du concours.





Et aussi de détecter 1000 catégories d'images plus précises (bateau de sauvetage, serviette de bain, cochon d'inde, …)

Modèle	Nb couches	Tx erreur top 5	
AlexNet	8	15,3 %	ILSVRC 2012
Inception-V1	22	6,7 %	ILSVRC 2014
VGG	16	7,3 %	ILSVRC 2014
ResNet	152	3,6 %	ILSVRC 2015

Les résultats améliorés d'année en année, vont jusqu'à 3.6% d'erreur sur le top 5, i.e. 96.4% de classification juste parmi les 5 premières classifications proposées.

Des modèles performants de détection d'image sont disponibles et utilisables.

```
# Exemple pour le modèle ResNet18 disponible dans le zoo de MXNet
from mxnet.gluon.model_zoo import vision
resnet18 = vision.resnet18_v1(pretrained=True)
```

On peut les utiliser en l'état, ou les adapter (transfert d'apprentissage) à un nouveau contexte de reconnaissance d'objets, pour des objets spécifiques.

Ces modèles fonctionnent relativement bien pour des objets du quotidien, mais dès qu'il faut reconnaitre des objets avec plus de caractéristiques, les choses se corsent, et il faudra un apprentissage spécifique.

Par exemple, au-delà de reconnaitre tel ou tel vêtement sur une image, un e-commerçant voudra que le modèle reconnaisse le type de maille, la coupe du vêtement, la matière, le nombre de boutons, ou encore le fabricant du vêtement … bref ajouter du détail métier à la simple reconnaissance d'un objet. Dans ce cas, il va falloir que le modèle réapprenne sur une base d'images nouvelles, labellisées, et adaptées à ce contexte. Et on peut imaginer que la constitution du jeu de données d'apprentissage ne sera pas une mince affaire.

D'autres cas de reconnaissance spécifique vont nécessiter probablement moins d'apprentissage, par exemple pour un contrôle qualité visuel en fin de chaîne de fabrication, avec des exemples plus faciles à trouver pour des images de produits passant ou non ce contrôle qualité.

Les exemples d'application sont nombreux et n'ont de limite que notre imagination … et la disponibilité d'un jeu d'apprentissage.

Après le traitement des images, le traitement de la vidéo en continu (en fait échantillonné, pour être traité séquence par séquence) ouvre de nombreux cas d'utilisation, notamment pour de la maintenance prédictive.

Des caméras vont pouvoir filmer en continu les matériels à surveiller, et sur la base de modèles de classification sachant détecter des matériels endommagés ou non, vont pouvoir détecter une usure ou un défaut avant que la panne ne se produise.

On voit facilement l'intérêt financier dès lors que le matériel est couteux, pour le réparer avant qu'il ne casse. C'est aussi intéressant lorsque le matériel est critique pour un service, et qu'il faut anticiper ses arrêts.

Notons que toutes les solutions open source citées ici intègrent directement un ou plusieurs modèles pré-entrainés de détection/classification d'image.

Expérimentation : reconnaitre des plateaux repas

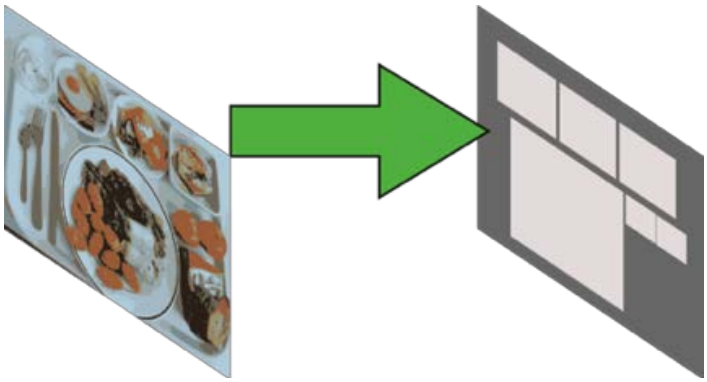
Dans le cadre d'une étude de faisabilité, Smile a utilisé ces technologies pour construire une solution de reconnaissance automatique de plats sur des plateaux repas.

On comprend aisément l'intérêt d'une telle solution, avec la possibilité de caisses automatiques qui détermineraient seules le montant du plateau-repas, sans nécessiter aucun changement important dans le restaurant.

Le principal enjeu de cette étude fut la mise en place d'une méthode de reconnaissance automatique sur un plateau repas de l'intégralité des objets alimentaires (plat, boisson, …) parmi un ensemble d'objets mixtes.

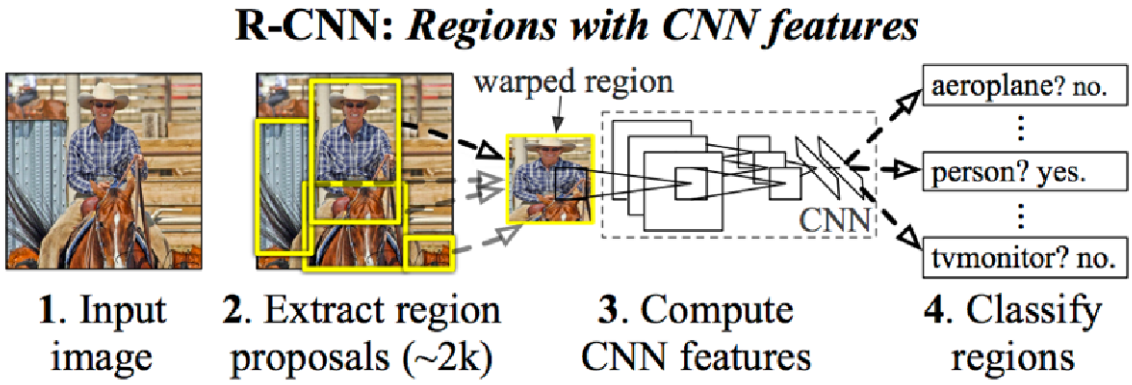
Plusieurs approches ont pu être testées mais schématiquement le processus repose toujours sur ces mêmes étapes :

- Extraire les zones d'intérêts (tout ce qui nous intéresse et rien d'autre)



- Classifier chaque zone d'intérêt

Parmi les différentes techniques testées, la plus adaptée est basée sur un type particulier de réseau neuronal nommé R-CNN (Regions with Convolutional Neural Network Features). Ce type de réseau regroupe les deux étapes (extraction des zones d'intérêts et ensuite leur classification) dans un seul et même réseau.



A noter que ce type de réseau nécessite une base de données d'entrainement adaptée à son fonctionnement. En effet, l'entrée de ce réseau n'étant plus l'association d'une image avec un seul objet mais bien plusieurs objets, il est donc nécessaire d'organiser les données d'entrainement pour représenter ces différentes associations : à chaque image d'entrainement doivent être associées des métadonnées d'annotation qui décrivent chaque zone d'intérêt par ses coordonnées et son nom.

Cette phase peut être énormément consommatrice en temps si la base d'entrainement doit être constituée de zéro. En effet pour fonctionner correctement, la volumétrie de données d'entrainement peut se compter en dizaines de milliers. Si le domaine de reconnaissance peut s'appuyer sur des bases publiques d'images c'est autant de temps gagné.

Dans le cas contraire, une stratégie de constitution est primordiale pour le succès d'un tel projet lors de son déploiement et industrialisation. On peut imaginer par exemple de faire appel à un levier utilisateur final (Crowdsourcing / gamification). Sur le long terme, il est également important d'assurer une amélioration continue de cette base de données avec les corrections sur erreurs.

Cette expérimentation nous a confirmé cette nécessité de collecte massive de données d'entrainement : malgré des résultats satisfaisants avec quelques milliers de photos, la progression des résultats de prédictions s'est toujours réalisée après l'ajout important de données d'entrainement.

Tous les détails de cette expérimentation sont disponibles sur le blog de Smile Innovation.

NATURAL LANGUAGE PROCESSING

Le Natural Language Processing – i.e. le traitement du langage naturel - permet à une machine de comprendre un texte, une parole et de s'exprimer de manière naturelle avec un humain.

Lorsque l'interaction est vocale, elle passe par une étape supplémentaire de transcodage de parole vers texte, ou de texte vers parole. Puis le reste du traitement est identique.

Comprendre une phrase, un texte, savoir en ressortir les idées principales, savoir le synthétiser, sont des compétences qu'on croyait complexes, intelligentes, réservées aux humains.

Pourtant une machine peut maintenant faire de même, résumer un texte en une ligne, ou en plus de lignes, à la demande. Et elle le fera relativement bien. Ce qui en soi, peut être utile dans de nombreux domaines, juridiques, techniques, pour ressortir de corpus documentaires des éléments croisés par exemple. Pour autant, il faut bien comprendre que la machine sait résumer un texte, mais qu'elle n'en comprend en réalité rien du tout. Le programme saura rédiger une synthèse, mais sera incapable d'utiliser les concepts du texte sur un autre sujet.

Par rapport aux autres domaines du Deep Learning – diagnostic médical, reconnaissance d'image, maintenance préventive, … - qui sont identiques pour le monde entier, le NLP est à contextualiser par langue et par pays (pour toutes les expressions idiomatiques). Les jeux de données d'apprentissage sont du coup non généralisable, et la disponibilité de jeux génériques efficaces pour une langue autre que l'anglais reste assez réduite.

Chatbots

Le NLP s'exprime notamment au travers de chatbots, en français des 'agents conversationnels', c'est-à-dire des programmes qui peuvent converser avec des humains.

D'un point de vue business, un des ROI possible est de réduire la charge des centres de relation client en étant capable de répondre aux questions des clients, a minima aux premières questions ou aux plus simples, tout en ajoutant une disponibilité 24/7.

Le chatbot peut prendre la forme d'une fenêtre de discussion sur un site web, ou d'un robot doté de la parole, en magasin ou même chez soi.

Regardons tout d'abord les principes de base.

Pour comprendre les questions qu'on lui pose, le chatbot doit pouvoir interpréter les phrases qu'on lui envoie. A savoir, des librairies existent comme par exemple NLTK, disponible pour plusieurs langues et apportant des capacités d'interprétation.

Ensuite, pour pouvoir répondre aux questions des clients et apporter une réponse pertinente, il faut donner au chatbot les éléments de contexte dans lequel il est.

Il doit savoir donner par exemple les horaires d'ouverture du magasin qu'il représente, l'adresse d'un service précis, donner des informations sur un évènement ponctuel… Il doit aussi savoir décrire les services proposés par le magasin. Tout ça en langage naturel.

Ces éléments de contexte sont les 'intentions' de l'utilisateur, auxquelles on associe des réponses. Par exemple, deux intentions ci-dessous, sur les horaires et sur les locations de scooters possibles.

```
# Extrait et localisation française du tutoriel
https://github.com/ugik/notebooks/blob/master/Tensorflow%20chat-
bot%20model.ipynb

{"intents": [
{"tag": "horaires",
"patterns": ["Vous ouvrez à quelle heure?", "Quels sont vos horaires
d'ouverture ?", "Quand êtes-vous ouvert?" ],
"responses": ["On est ouvert tous les jours de 9h du matin à 19h",
"Nos horaires sont de 9h à 19h tous les jours"]
},
{"tag": "scooter",
"patterns": ["Quels scooter avez-vous ?", "Quels types de scooter il
y a ?", "Qu'est-ce que vous louez ?" ],
"responses": ["On loue des scooters bleus, des verts et des roses",
"On a des scooters bleus, verts et roses"]
}
```

Extrait et localisation française du tutoriel
https://github.com/ugik/notebooks/blob/master/Tensorflow%20chat-bot%20model.ipynb

Ici on utilise un pattern, qui va permettre au chatbot de reconnaitre l'intention de l'utilisateur, avec des variantes d'énoncés pour la même intention. Il répondra « On est ouvert tous les jours… » à la question « Vous ouvrez à quelle heure ? », et aussi à la question « à quelle heure vous ouvrez ? ». C'est l'interprétation du sens de la question qui orientera sur l'intention et donc sur la réponse apportée.

Le chatbot doit donc être entrainé avec ces intentions. Ça correspond à la phase d'apprentissage du réseau de neurones.

On notera qu'il n'y a pas besoin de millions de lignes d'exemples. Juste de quelques dizaines de lignes d'intentions et de variantes. En effet, il s'agit plutôt de transfert d'apprentissage, car le modèle est déjà capable d'extraire les intentions d'un texte, c'est-à-dire les mots signifiants.

En plus des questions/réponses préparées, ce qui fait une discussion, c'est l'enchainement des échanges sur un même sujet, le contexte. Les intentions doivent être créées avec une notion de contexte, qui lie les échanges ensemble, et permettra au modèle de répondre correctement à la 2è question d'un même contexte.

```
client  > je voudrais louer un scooter
chatbot > Souhaitez-vous louer pour aujourd'hui ou pour plus tard ?
client  > pour aujourd'hui
chatbot > Pour les locations du jour, merci d'appeler le 01 01 01 01
01
```

Si le contexte n'est pas pris en compte, la réponse à la 2è phrase du client sera à coup sûr à côté de la plaque…

Sans contexte, aucun échange efficace ne peut avoir lieu avec un chatbot.

Pour cela, chaque tag peut déclencher le début d'un contexte, réutilisé ensuite dans une autre intention.

```
{"tag": "location",
"patterns": ["Peut-on louer un scooter ?", "J'aimerai louer un
scooter", "Comment ça marche ?" ],
"responses": ["Souhaitez-vous louer pour aujourd'hui ou pour plus
tard cette semaine ?"],
"context_set": "jourlocation"
},
{"tag": "aujourdhui",
"patterns": ["aujourd'hui"],
"responses": ["Pour les locations du jour, merci d'appeler le 01 06
07 08 09", "Pour le jour même, il faut appeler le 01 06 07 08 09"],
"context_filter": "jourlocation"
```

La complétude et la qualité de la définition des intentions feront que le chatbot répondra correctement ou non aux questions de leurs interlocuteurs.

On voit dans cet exemple – relativement simple - que la compréhension de la question est liée d'une part à la qualité des intentions déclarées manuellement, et d'autre part aux capacités de compréhension de la librairie utilisée (ici NLTK).

Notons que pour un chatbot qui dialoguerait oralement, c'est exactement le même principe, avec en plus une couche d'encodage/décodage de la parole vers ou à partir du texte. C'est évidemment un peu vite dit, car la compréhension de la parole, indépendamment de l'accent de celui qui parle, du fait que ce soit un enfant, qu'il y ait plus ou moins de bruit environnant, … est un challenge à part entière.

Un enjeu de la réalisation d'un chatbot est donc d'avoir la meilleure compréhension possible, au travers de la librairie NLU – Natural Language Understanding - utilisée.

En plus de la librairie NLTK déjà citée, citons les librairies open source disponibles Apache OpenNLP, AllenNLP, Rasa ou plus récemment Snips NLU, une startup française qui a reversé en open source une partie de son logiciel.

Les plateformes web proposent des assistants pour créer son propre chatbot, à partir d'un modèle déjà opérationnel pour des éléments de conversation, auquel il faut ajouter ses propres intentions. Si on exclue la confidentialité et la propriété des données transmises, cela peut être efficace, mais la problématique pour une entreprise sera de mettre en place un système de chatbot unique pour tous les canaux, pas seulement pour une seule plateforme.

Et ensuite, de le faire évoluer dans le temps, avec de nouvelles 'intentions' et de nouveaux services.

Dans ce cas, interopérabilité et maitrise de son système d'information riment en général avec open source.

Expérimentation autour d'un chatbot

Pour la constitution de son offre chatbot, Smile a pu tester différentes solutions NLU et leur intégration dans des contextes projets chatbot.

Notamment l'une des solutions testées est le moteur NLU de Snips. Ce test avait pour objectif de réaliser un entrainement, en langue française, de plusieurs intentions mixant différents types d'entités. Le modèle entrainé devant ensuite être intégré au sein du moteur chatbot BotPress : BotPress est une solution permettant de construire et gérer un chatbot de la même manière que l'on gérerait un site web par son interface de gestion de contenu (CMS).

L'engine NLU de Snips est disponible pour différentes plateformes d'exécution. Pour cet exercice la librairie Python a été utilisée.

Chaque moteur NLU possède son propre formalisme pour la définition des données d'entrainement.

Snips NLU repose sur une logique de fichiers textes décrivant les intentions et entités désirées. A noter que si certains types d'entités sont disponibles directement (quantité, durée, date, monnaies, …), il est également possible de définir ses propres entités métiers (liste de mots, exhaustifs ou non, pouvant prendre en compte des synonymes). La logique de compositions de ces données est la suivante :

- Un fichier texte par intention, contenant par ligne un exemple de phrase. Le formalisme permettant d'inclure l'annotation de présence d'une entité (son type et un exemple) au sein même de la phrase
- Un fichier texte par entité personnalisée

Ainsi par exemple pour l'intention « connaitre la capitale d'un pays », le fichier d'intention correspondant contiendra ce type de phrase :

```
Pouvez-vous me dire quelle est la capitale de la
[country:country_fr](Grèce) ?

Aux [country:country_fr](Etats-Unis) quelle est la capitale ?
```

Et pour l'intention « connaitre le jour de la semaine pour une date particulière » on aurait :

```
[date:snips/datetime](il y a 50h), on était quel jour ?

Quel jour on était [date:snips/datetime](le 21 décembre 2015) ?
[date:snips/datetime](Dans 3 mois) exactement, on sera quel jour ?
```

L'entité « date » étant gérée directement par Snips, rien de plus ici à faire, mais pour l'entité « country » il faudra constituer un fichier texte contenant l'ensemble des noms des pays en langue française.

Le nombre de phrases par fichier d'intention va déterminer la qualité de prédiction lors de la phase d'analyse d'un texte. Lors de ces tests, on constate une qualité de prédiction minimale à partir d'une trentaine d'exemples. Mais une qualité optimale nécessite généralement plusieurs centaines d'exemples.

Après entrainement, on obtient un modèle qui peut être utilisé pour analyser un texte et en extraire une probabilité d'intention et de ces entités associées trouvées.
Par exemple, pour la phrase « Quelle est la capitale du Burundi ? » on obtiendra du moteur NLU la réponse suivante :

```
{
    "input": "Quelle est la capitale du Burundi ?",
    "intent": {
        "intentName": "searchCountryCapital",
        "probability": 0.851309036737854
    },
    "slots": [
        {
            "entity": "country_fr",
            "range": {
                "end": 31,
                "start": 24
            },
            "rawValue": "Burundi",
            "slotName": "country",
            "value": {
                "kind": "Custom",
                "value": "Burundi"
            }
        }
    ]
}
```

Pour la phrase « Le lendemain du 14 juillet 2018, quel jour on sera ? », on obtiendra la réponse :

```
{
    "input": " Le lendemain du 14 juillet 2018, quel jour on sera ?",
    "intent": {
        "intentName": "sayWhatWeekDayOnDate",
        "probability": 0.8013662432640526
    },
    "slots": [
        {
            "entity": "snips/datetime",
            "range": {
                "end": 31,
                "start": 0
            },
            "rawValue": "Le lendemain du 14 juillet 2018",
            "slotName": "date",
```

```
            "value": {
                "grain": "Day",
                "kind": "InstantTime",
                "precision": "Exact",
                "value": "2018-07-15 00:00:00 +00:00"
            }
        }
    ]
}
```

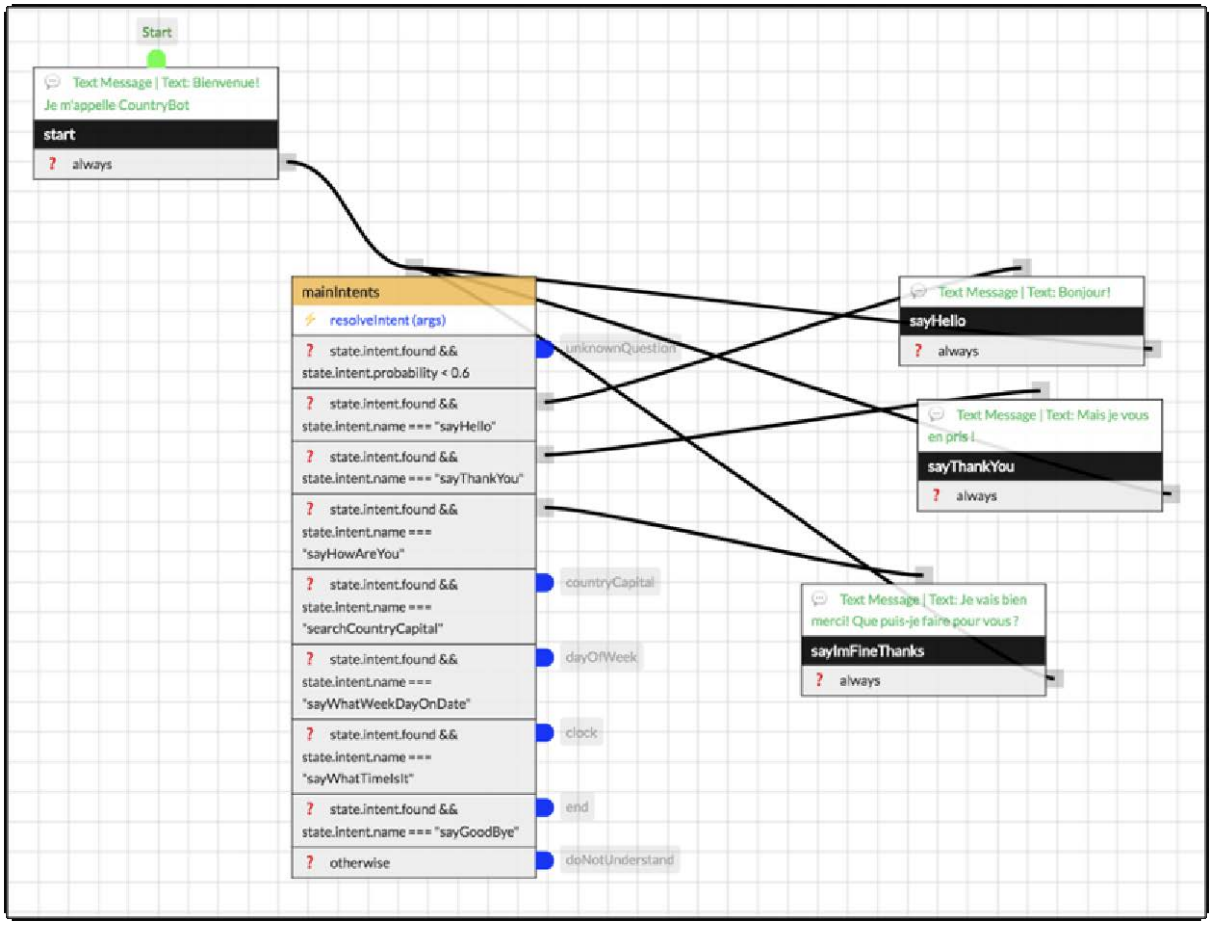
Pour chacune des réponses les éléments importants sont le nom de l'intention trouvée, sa probabilité et les différents slots d'entité trouvés dans la phrase.

Il reviendra donc ensuite à la logique d'intégration d'agir en conséquence en fonction de ces critères.

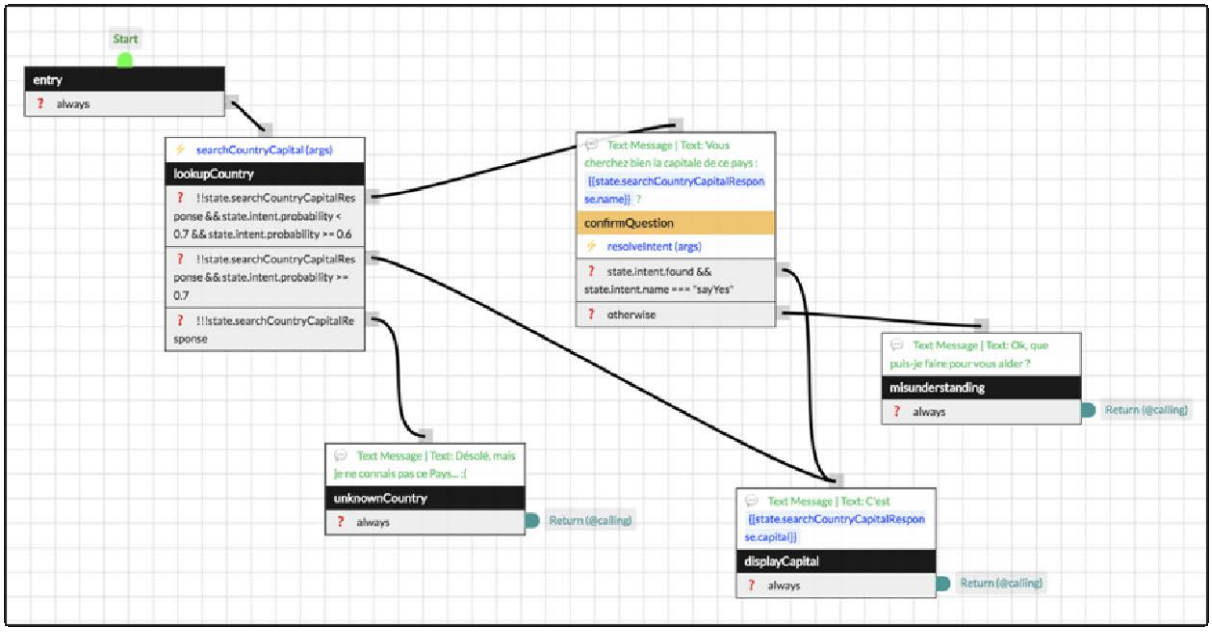
Lors de cette expérimentation du moteur NLU de Snips, on a pu retenir sa facilité d'usage et surtout sa très grande rapidité d'entrainement et d'exécution. En effet, ce moteur étant pensé pour être utilisé dans un environnement mobile ou embarqué (ex : Raspberry), il permet d'aller très vite et sans grand besoin de ressource matériel.

Concernant l'intégration dans la solution chatbot BotPress, l'usage d'un moteur NLU permet de simplifier énormément la logique d'arbre décisionnel en déléguant chaque entrée utilisateur à ce moteur NLU, et en spécialisant chaque nœud de l'arbre à une attente plus ou moins large d'intention.

Ici le nœud principal d'un scénario BotPress, intégrant Snips NLU, peut se résumer au branchement de sous-scénarios pour chacune des intentions gérées par le chatbot.



Un sous-scénario pourra gérer l'intention détectée et/ou demander plus d'information nécessaire à la réponse.



BotPress fonctionnant grâce à un mécanisme de machine à état, créée à chaque discussion il est donc possible de stocker différents états sous-jacents à cette discussion, un historique de choix ou de préférences. Ainsi on améliore d'autant plus l'expérience de l'utilisateur final.

L'OPEN SOURCE AU CŒUR DU DEEP LEARNING

Les avancées de l'IA ces dernières années sont colossales et sont majoritairement dues aux travaux des majors de l'Internet, que sont Google, Facebook, Amazon, Microsoft … Ces entreprises ont développé des usages liés à l'IA, et ont diffusé des solutions informatiques de Machine Learning / Deep Learning dans leurs cloud respectifs.

Influencée par la culture open source du monde de la recherche scientifique qui partage déjà ses travaux très ouvertement, la dynamique autour du Machine Learning est fortement open source. A tel point que ces grands acteurs ont tous diffusé leur solution en open source :

- L'université de Berkeley a publié Caffe en 2014
- IBM a reversé en 2015 ce qui est devenu Apache System ML
- Amazon a reversé DSSTNE en open source en 2015
- Google a reversé TensorFlow fin 2015
- DMLC, un groupe réunissant des développeurs de plusieurs sociétés - dont Amazon, Microsoft et Baidu - et laboratoires, a reversé MXNet en 2016…
- Facebook a reversé Caffe2 en 2017, et en 2018, son laboratoire de recherche en IA, le FAIR, reverse en open source Detectron, un programme de Deep Learning pour la reconnaissance d'objets
- Apple a diffusé Turi, librairie de Deep Learning orientée vers la facilité d'utilisation, et contenant diverses fonctions prêtes à l'emploi de reconnaissance d'objets

---

Les solutions open source de Deep Learning occupent le terrain

---

Bref, les solutions open source de Deep Learning occupent le terrain et sont en compétition entre elles.
Notons aussi qu'il ne s'agit pas de version « n-1 » ou de version « socle » qui sont diffusées en open source, mais bien les dernières avancées disponibles qui sont partagées en direct.
La simple diffusion de sources ouvertes n'est du coup plus suffisante pour l'emporter vis-à-vis des autres acteurs, l'enjeu est d'attirer la plus large communauté de data scientists et de développeurs.
D'une part pour améliorer en continu ces solutions par la communauté. D'autre part pour attirer les meilleurs chercheurs en IA. Et enfin, parce que ces utilisateurs vont construire leurs projets avec ces solutions, et que ces projets auront besoin de puissance de calcul, donc avec en rebond, on l'imagine, une plus grande utilisation à terme du cloud associé, qui lui est payant.

On voit émerger des outils de deep learning de plus en plus faciles d'usage

Au-delà des frameworks, on voit émerger des composants avec des fonctionnalités de Deep Learning plus faciles d'usage et déjà prêtes à l'emploi.
Ces fonctionnalités plus faciles d'accès vont baisser la barrière à l'entrée du Deep Learning, et vont ainsi permettre aux développeurs d'ajouter à leurs programmes de la reconnaissance vocale, visuelle, … sans avoir à connaitre les mathématiques derrière les réseaux de neurones, ni même la constitution pas à pas d'un réseau de neurones, comme c'était le cas il y a à peine quelques mois !

La diffusion open source est un facteur important de la démocratisation de l'IA, à la fois en termes de coût logiciel devenu nul, et en termes de fonctions complexes devenues prêtes à l'emploi.

Devant tant de solutions disponibles, librement accessibles, il est aujourd'hui difficile de choisir une seule solution pour un projet d'IA, d'autant qu'elles sont récentes et fortement évolutives.

SOLUTIONS OPEN SOURCE DE DEEP LEARNING

Ce chapitre détaille les solutions – ou frameworks - open source destinées au Deep Learning.

Les solutions couvrant uniquement le Machine Learning 'classique', sans les réseaux de neurones, ne sont pas listées, sans pour autant déméríter. Citons parmi ces solutions de Machine Learning les composants les plus répandus que sont Scikit-learn, Apache Mahout, … qui servent à résoudre de nombreux problèmes, sans faire appel aux réseaux de neurones.

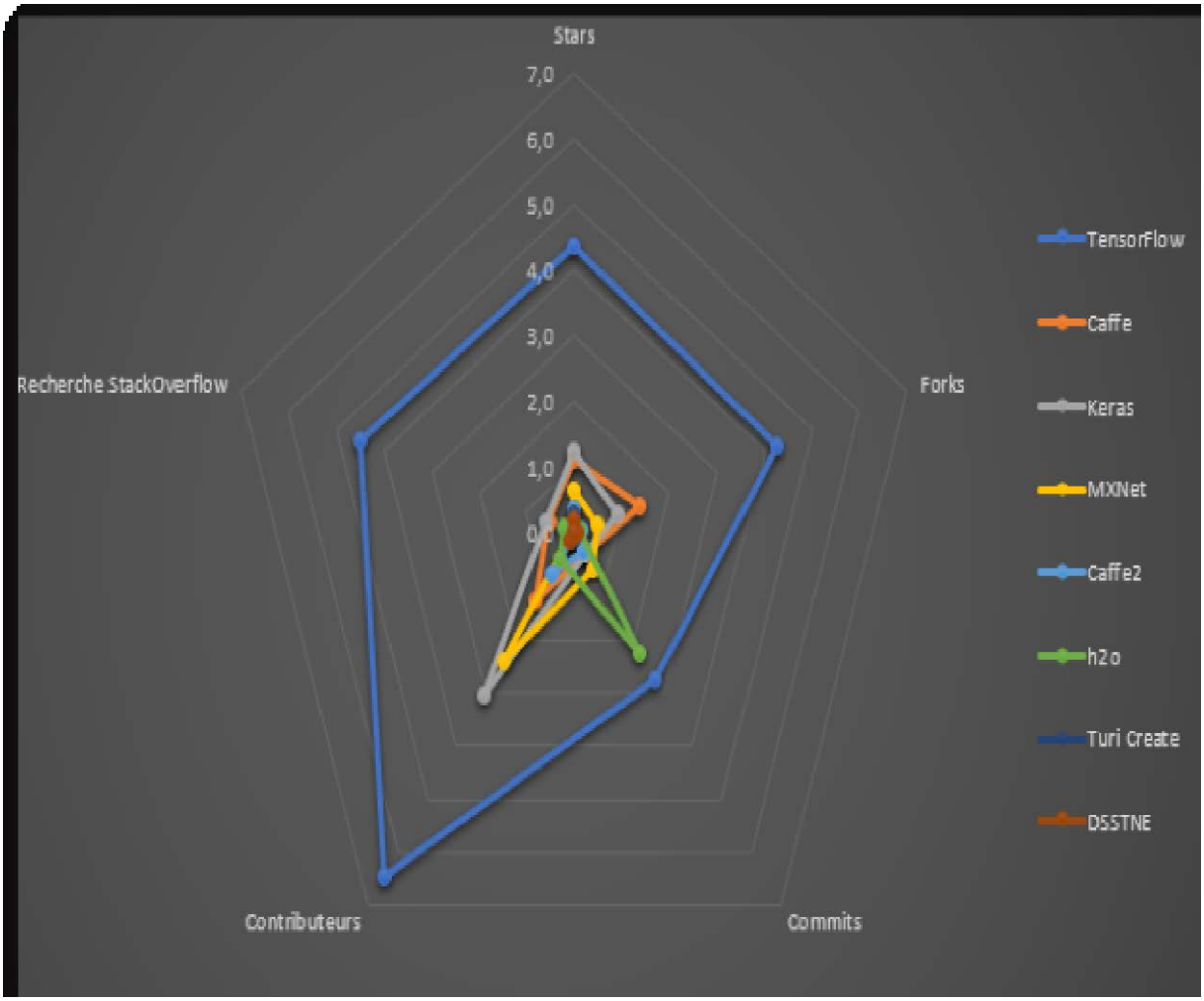
J'ai fait le choix de présenter ici une sélection de solutions open source de Deep Learning, et non l'exhaustivité, pour concentrer les efforts sur TensorFlow, Keras, H2o et MXNet, qui prennent selon moi la tête du classement.

Des solutions comme Theano ou Caffe sont exclues, même si ce sont des précurseurs qui ont beaucoup apporté, mais qui sont en phase descendante en termes d'usages et de dynamique. D'autres solutions comme PyTorch, encore en beta, soutenue par Facebook, et Paddle de Baidu restent à surveiller et entreront probablement dans une prochaine édition de ce livre blanc.

STATISTIQUES

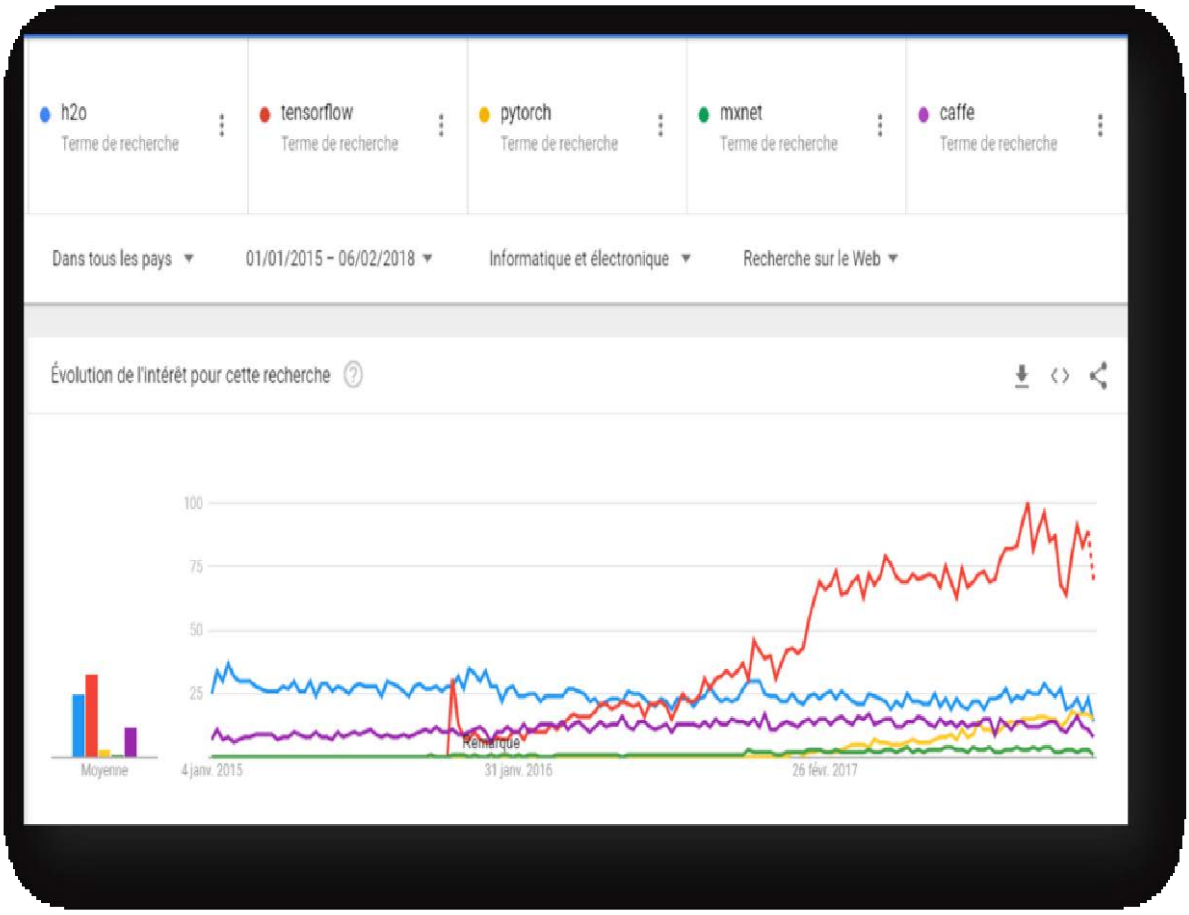
Ci-après quelques métriques prises début 2018 de Github et de StackOverflow pour les solutions de Deep Learning :

Solution	Stars	Forks	Commits	Contributeurs	StackOverflow
TensorFlow	87 062	42 488	27 702	1 286	44 982
h2o	2 800	1 109	22 351	97	2 191
MXNet	12 867	4 733	6 573	477	643
Caffe	22 505	13 776	4 080	256	5 363
Keras	24 552	8 944	4 034	610	5 998
Caffe2	7 088	1 608	3 186	153	82
Turi Create	5 718	487	153	23	23
DSSTNE	4 010	667	281	30	n/a



On voit que TensorFlow a le plus de « stars », le plus de « forks », le plus de commits et le plus de contributeurs …

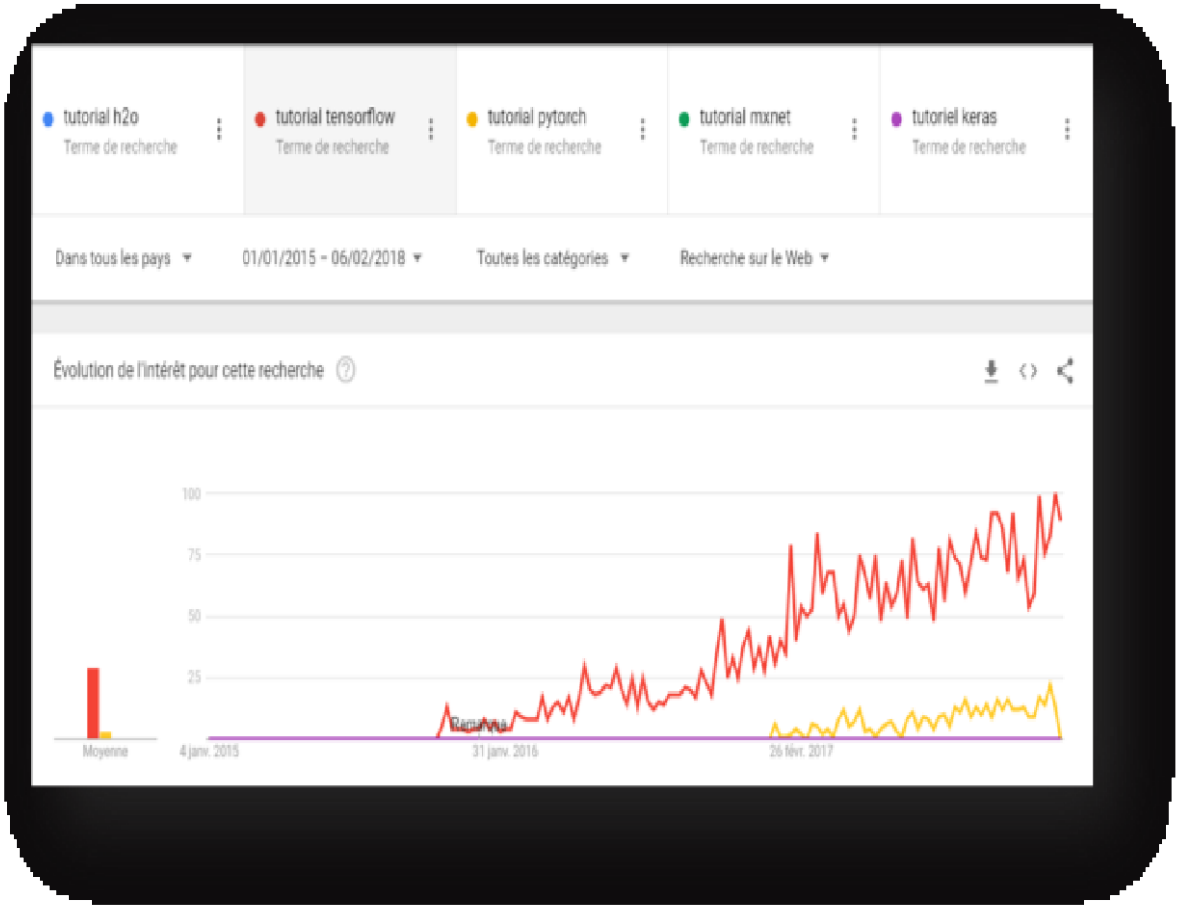
Regardons maintenant les recherches Google sur les 3 solutions back-end citées ici : TensorFlow, Caffe et MXNet, depuis 2015 :



Là encore, la popularité de TensorFlow (en rouge) est sans conteste depuis sa parution fin 2015. Celles de H2O et Caffe sont stables, mais en proportion uniquement, sachant que l'intérêt est grandissant pour ces solutions. Notons que PyTorch voit sa popularité monter en 2017.

Ces éléments donnent des indications claires sur la popularité et sur le leadership qu'a pris TensorFlow sur la communauté IA depuis sa publication en 2015

La popularité d'une solution est en fait déterminante, c'est ce qui fera qu'il y aura plus ou moins de tutoriels disponibles, plus ou moins d'exemples de codes disponibles, plus ou moins de modèles pré-entrainés disponibles, plus ou moins de personnes compétentes sur le sujet …



C'est donc un bon indicateur, mais malgré tout, cela n'est pas suffisant pour déterminer quelle solution est la plus adaptée pour tel contexte, d'autant que la combinaison de solutions est possible.

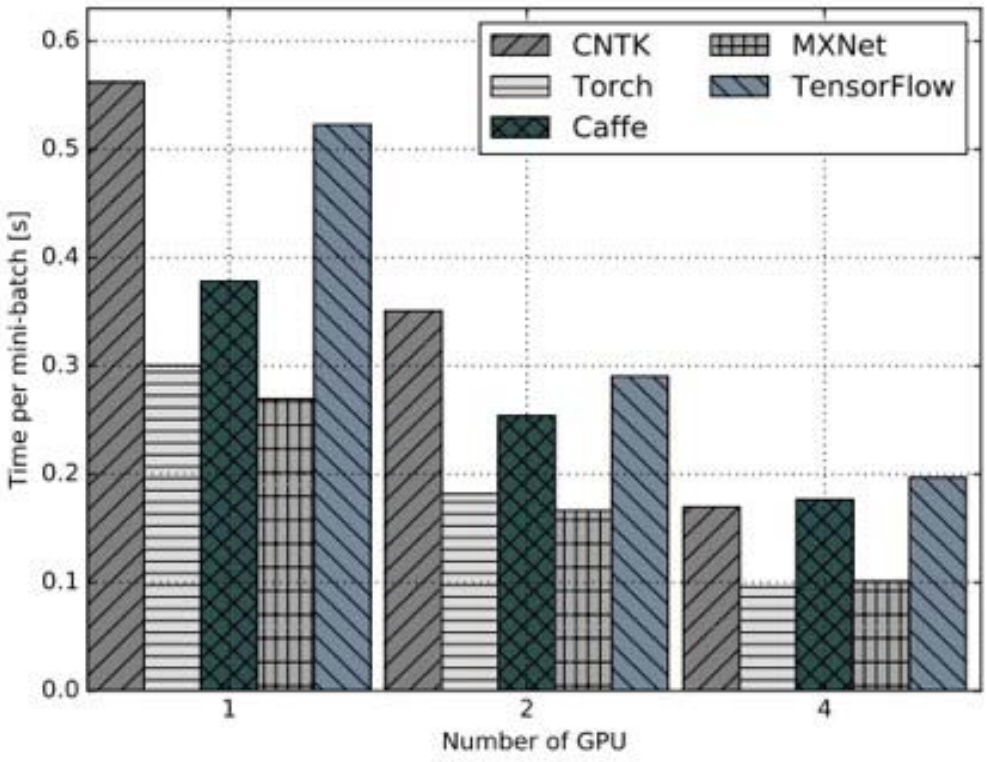
PERFORMANCES

Les phases d'apprentissage sont extrêmement chronophages, et sont de plus itératives et répétitives. La performance dans l'exécution de ces phases est donc importante et peut être un facteur de réduction de coût.

D'après les travaux de benchmarking publiés par des chercheurs de l'université de Hong-Kong, toutes les solutions étudiées sont efficaces dans l'utilisation des GPU. Et aucune solution ne surperforme clairement par rapport aux autres, d'autant qu'il existe diverses stratégies d'optimisation des performances.

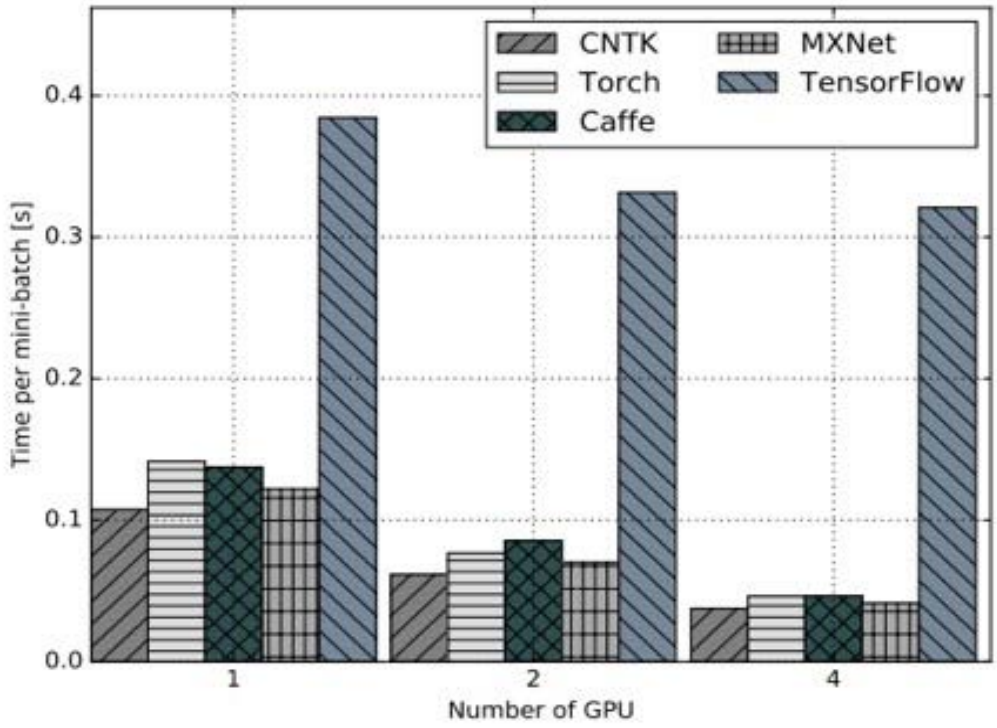
Le moins qu'on puisse dire, c'est que ça n'aide pas au choix …

Dans le détail, les travaux réutilisent des modèles pré-entrainés comme AlexNet ou ResNet et comparent les performances de MXNet, TensorFlow, Caffe, PyTorch et CNTK. Les graphiques suivants montrent le temps nécessaire par mini-batch.



(a) Performance comparison of ResNet-56 on mulit-GPU platform.

Pour ResNet-56, pour 4 GPU, les 4 produits sont dans la même zone, avec une légère meilleure performance pour MXNet et Torch.



(a) Performance comparison of AlexNet-R on mulit-GPU platform.

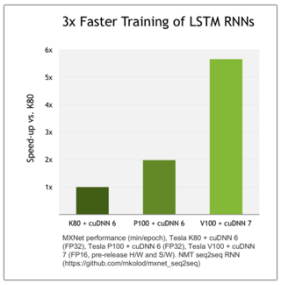
Pour AlexNet-R, à 1, 2 ou 4 GPU, toutes les solutions sont équivalentes, sauf TensorFlow qui est moins performante.

GPU

La puissance de calcul pour l'apprentissage des Réseaux de Neurones est fortement liée à l'utilisation de processeurs de type GPU à la place de CPU, pour une parallélisation des traitements lourds nécessaires.

En bref, les CPU contiennent un nombre réduit de cœurs (2,4,6,8…), optimisés pour le traitement en série, alors que les GPU intègrent des milliers de cœurs plus simples et conçus pour traiter de nombreuses tâches simultanées.

La compatibilité avec une exécution GPU est donc primordiale pour une solution de Deep Learning, tant il peut être long et coûteux d'attendre l'exécution des tâches d'apprentissage.



Les processeurs GPU de NVIDIA sont incontournables, avec la librairie cuDNN spécifiquement optimisée pour le Deep Learning, avec des temps d'exécution divisés par 3 !

Si au début ce n'était pas le cas, les solutions open source supportent tout en maintenant une exécution GPU.

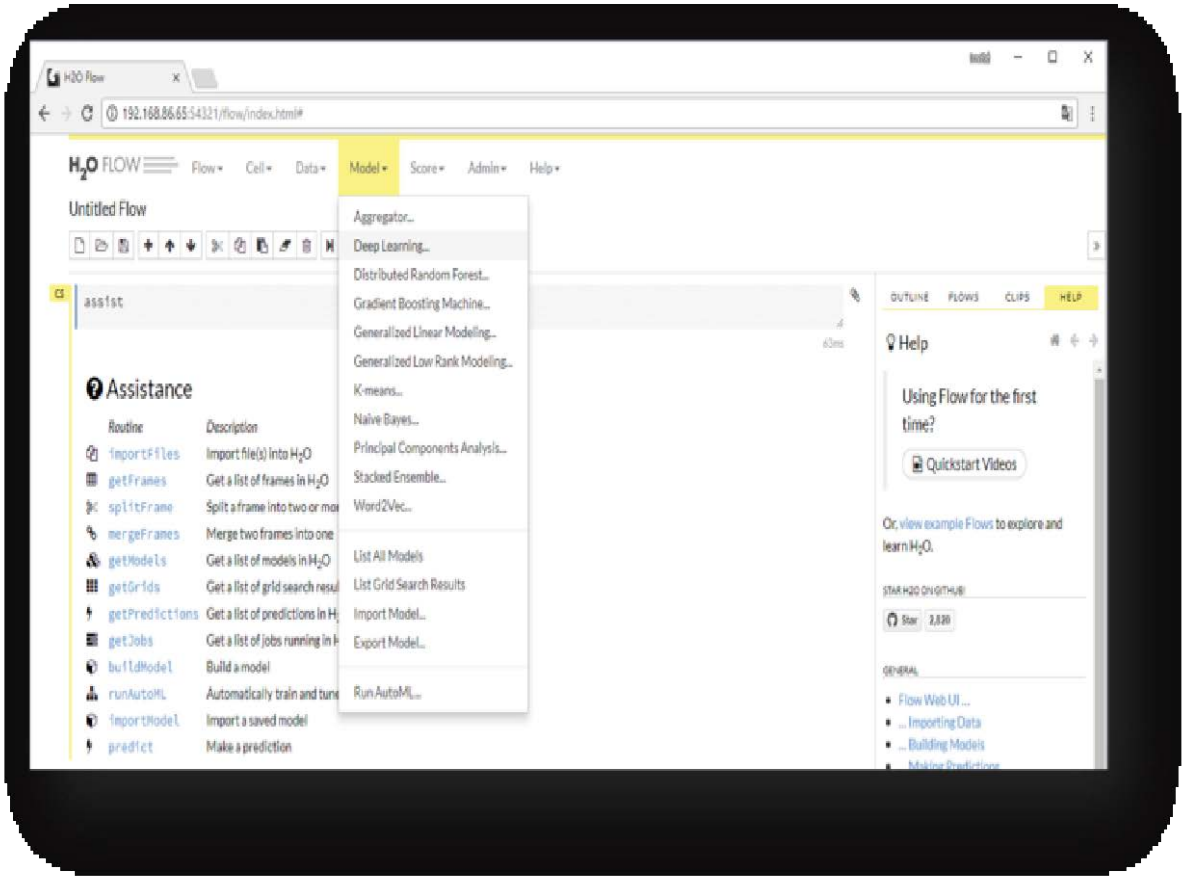
H2O

H2O est édité par la société californienne du même nom (précédemment nommée '0xdata'), créée en 2011, ce qui fait de H2O la plus ancienne des solutions présentées ici.

La solution H2O est très largement utilisée, et l'éditeur communique sur le fait que 222 sociétés du Fortune 500 l'utilisent, avec près de 130 000 utilisateurs.

Notons que H2O couvre tout le spectre du machine learning, pas seulement celui du deep learning. H2O comprend 4 produits : H2O, H2O4GPU, Sparkling Water et Driverless AI.

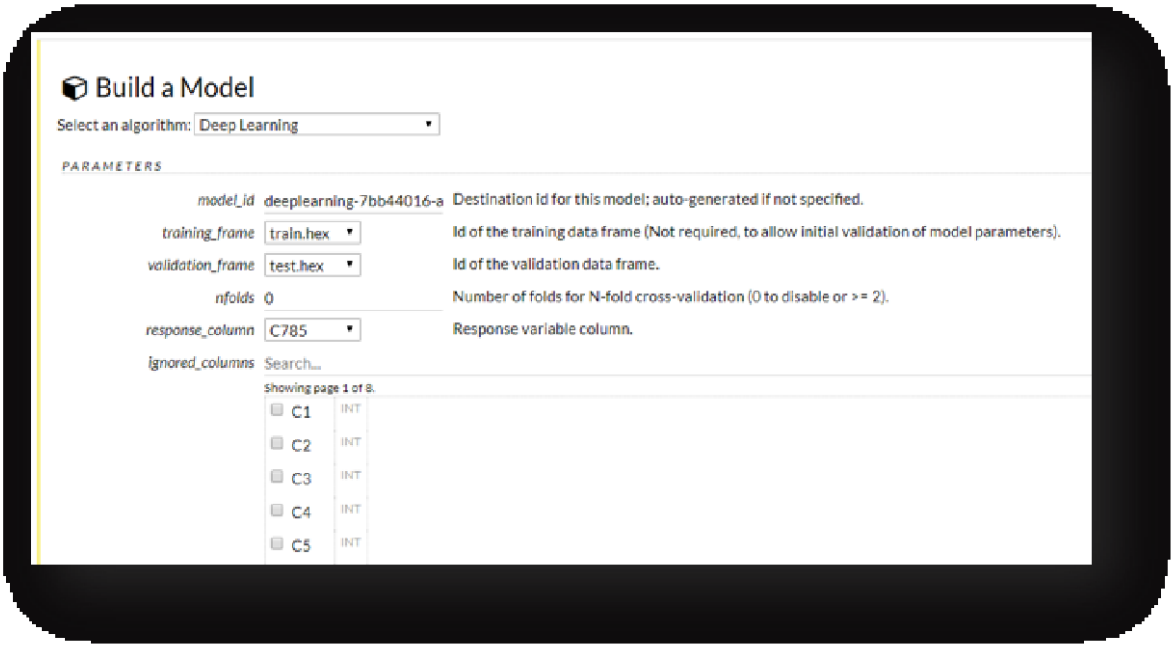
L'interface web de H2O comprend le notebook Jupyter pour l'exécution pas à pas, et un menu permet au data scientist de lancer des actions d'import des fichiers, de choix de modélisation, de scoring …



Dashboard H2O

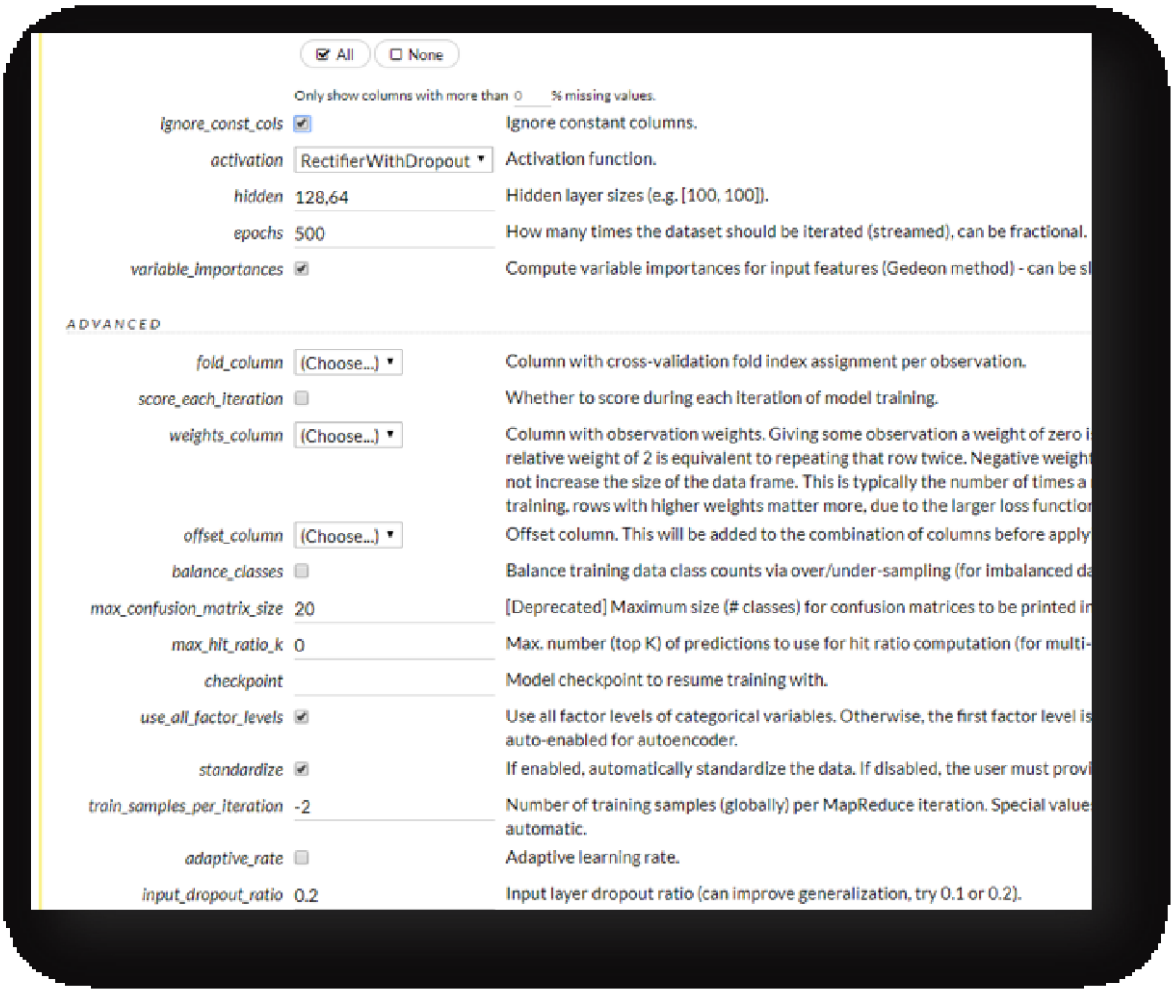
Une des forces de H2O réside dans cette interface web, avec ses formulaires d'aide au paramétrage.

Avec de tels formulaires, listant tous les paramètres à définir pour chaque modèle, le data scientist n'a plus à connaitre le code à écrire pour travailler ses données et ses modèles.



Formulaire de création d'un modèle de type 'Deep Learning', avec les nombreux paramètres à définir.

Les paramètres sont regroupés en 3 types : 'Parameters', 'Advanced' et 'Expert', avec à chaque fois des valeurs par défaut.



Paramètrage d'un RN avec 2 couches cachées de 128 puis 64 éléments, et une fonction d'activation 'RectifierWithDropout'

Les paramètres sont nombreux, d'où une certaine complexité affichée, mais on a l'avantage de les voir tous, et aussi d'avoir des listes déroulantes pour certains choix, ce qui est toujours plus pratique.

Pour créer un modèle rapidement, le premier bloc de paramètres suffit, et les autres blocs viennent dans un second temps, pour permettre d'affiner le modèle.

Au-delà de l'interface web, H2O est appelable via son API en Python (ci-dessous) ou en R.

```
import h2o
h2o.init()

# chargement des données, ici un fichier disponible sur une url
data = h2o.import("http://url-de-mon-fichier")
y = nom_du_champ_à_prédire_dans_le_fichier
# suppression du champ de sortie de la liste des entrées
x = data.names
x.remove(y)
```

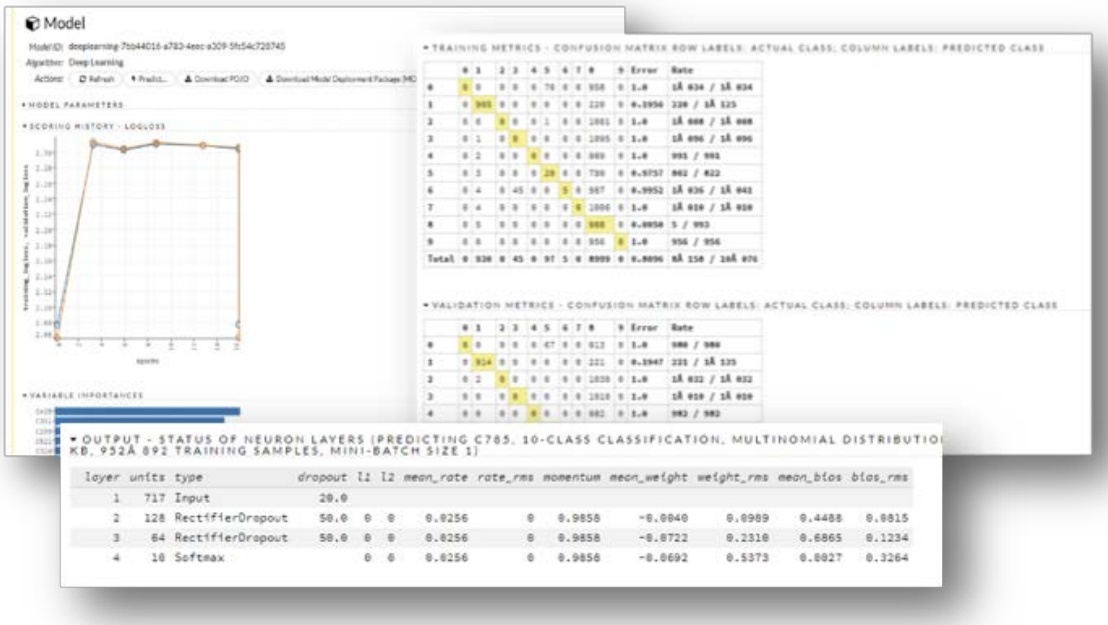
```
# séparation en 2 jeux : ici 90% sur le jeu 'train' et le reste sur
'test'
train, test = data.split_frame([0.90])

# apprentissage sur 'train' et application sur 'test' avec la
fonction DL paramétrée par défaut
m = h2o.estimators.deeplearning.H2ODeepLearningEstimator()
m.train(x,y,train)
p = m.predict(test)
```

Ces mêmes opérations peuvent se faire par l'interface web, avec des enchainements de clics ('import files' ↦ parse' → 'split frames' → 'build model'), donc sans coder une seule ligne.

Graphes

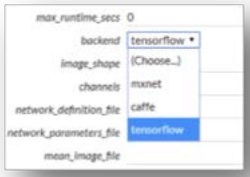
H2O permet de visualiser directement le modèle généré, avec le graphe de l'évolution de la fonction de perte, la matrice de confusion, l'importance des variables d'entrée et les informations sur le réseau de neurone défini



H2O mise donc sur l'assistance à l'utilisation fournie au data scientist, avec tout le spectre du machine learning implémenté, du 'GBM' au réseau de neurones, en passant par l'algorithme de 'forêt aléatoire'.

H2O propose en complément deux versions, intégrées à l'écosystème extérieur du bigdata et du Deep Learning, en fonctionnant nativement avec Spark et TensorFlow/MXNet/Caffe :

- 'Sparkling Water' qui fonctionne directement avec 'Spark' pour le traitement massif et parallélisé des fichiers,
- 'Deep Water' qui fonctionne nativement MXNet et TensorFlow pour la partie DeepLearning



Avec la version DeepWater intégrant les backends TensorFlow et MXNet, c'est peut-être le meilleur des mondes qui est proposé.

Notons que l'installation de DeepWater n'est pas aussi simple que H2O standard, avec la nécessité de recompiler TF ou MXNet selon son choix. La procédure d'installation avec Caffe comme backend n'est quant à elle pas encore publiée.

D'autres outils, comme Keras que l'on verra plus loin, proposent aussi de fonctionner avec plusieurs backends de Deep Learning.

DRIVERLESS AI

Tout dernièrement, H2O a sorti « Driverless AI », (DAI pour faire court) encore en version beta, pour aller encore plus loin dans l'assistance fournie au data scientist dans la recherche de modèles de machine learning.

DAI automatise plusieurs étapes coûteuses telles que « feature engineering », la validation de modèle, le choix de modèle …

L'interface est encore plus simple, et beaucoup de réglages sont automatiques. DAI va chercher lui-même le meilleur modèle à appliquer, parmi les modèles de ML disponibles.

Au-delà du modèle, l'ambition est aussi d'interpréter les résultats d'un modèle, lorsqu'il est appliqué.

En effet, une fois le modèle validé, un modèle peut être appliqué et prédire des résultats ou faire des classifications. Pour autant, est-ce que sa prédiction ou sa classification est compréhensible pour un humain ? Est-ce que je peux lui faire confiance ?

La complexité des modèles de Deep Learning est telle qu'il est difficile d'expliquer via l'algorithme ce qui a déterminé la réponse du modèle.

Pour la partie interprétabilité, DAI fournit plusieurs graphes permettant par exemple de mesurer l'importance de chaque donnée dans le résultat, et de voir la variation du résultat si on fait varier cette donnée en particulier.

C'est un premier pas dans l'interprétation du ML et c'est assez appréciable même si cela exclut par exemple l'interprétation de données non structurées (images / vidéos)

H2O étant assez avancé sur le sujet du MLI, avec plusieurs publications associées, les avancées vont certainement continuer.

APACHE MXNET

Apache MXNet est un framework open source de Deep Learning qui permet de manipuler les réseaux de neurones, et de les déployer dans différents environnements.

MXnet est accessible à partir de plusieurs langages : Python et R, comme les autres solutions, mais aussi Scala, Julia, C++ et Perl.

MXNet est supporté par plusieurs entreprises dont Amazon et Microsoft, et donc disponible dans les clouds respectifs AWS et Azure.

Après une phase d'incubation d'un an, MXNet est devenu un projet officiel de la fondation Apache, et rejoint ainsi des projets comme Hadoop, Hive, Pig ou Spark qui sont la référence dans le domaine du big data.

L'officialisation du projet signifie une activité suffisante, et implique aussi une certaine pérennité, d'autant plus qu'il a le support de plusieurs entreprises, qui vont toutes y contribuer.

Gluon

MXNet intègre le projet Gluon, une API d'accès à MXNet, qui simplifie la manipulation des modèles de réseaux de neurones. Arrivé en 2017, donc après TensorFlow de Google, il s'y compare forcément. L'API est plus simple et la programmation plus classique, de type 'impérative' (à l'inverse de la session de TensorFlow, programmation de type 'symbolique)

L'appréciation de la simplicité est subjective, mais on pourra noter que modéliser un RN avec Gluon s'avère moins verbeux qu'avec TensorFlow. A ce jeu-là, on aura compris que c'est encore moins verbeux avec H2O, où la définition se fait via l'interface web.

```
#extrait de https://github.com/zackchase/mxnet-the-straight-
dope/blob/master/chapter03_deep-neural-networks/plumbing.ipynb

#Définition d'un modèle plus simple, avec 3 couches et RELU en
activation
class MLP(Block):
    def __init__(self, **kwargs):
        super(MLP, self).__init__(**kwargs)
```

```
        with self.name_scope():
            self.dense0 = nn.Dense(128)
            self.dense1 = nn.Dense(64)
            self.dense2 = nn.Dense(10)

    def forward(self, x):
        x = nd.relu(self.dense0(x))
        x = nd.relu(self.dense1(x))
        return self.dense2(x)
```

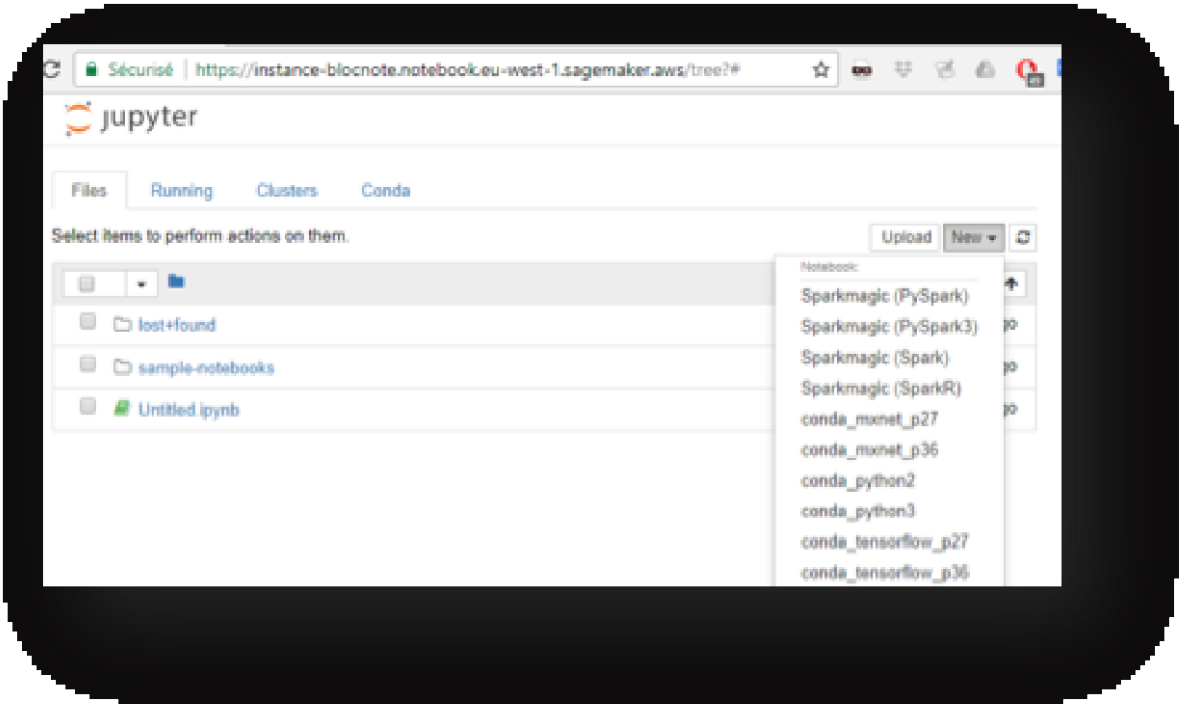
La documentation disponible couvre de nombreux exemples (CNN, RNN, RL, …), et encourage même à réutiliser le code mis à disposition sous forme de notebooks.

Créé pour séduire le plus grand nombre, il est difficile de dire s'il a atteint son objectif ou non tant il est récent et ses concurrents déjà en place et populaires. Le ralliement de plusieurs universités américaines à MXNet va certainement faire grandir la communauté et peut changer la donne.

MXnet a clairement des atouts forts qui peuvent renverser la tendance : documentation, tutoriels, code plus accessible, et surtout la légitimité de la fondation Apache, non dépendante d'un seul acteur.

Déploiement dans AWS

Amazon a fait le choix fort d'appuyer toutes ses fonctionnalités de Deep Learning sur MXNet fin 2017. AWS intègre ou plutôt va intégrer MXNet dans tous ses services IA de haut niveau, sans exclure pour autant les autres solutions en tant que framework proposé.



Par exemple, dans les instances disponibles de notebooks dans AWS, Jupyter intègre différents environnements, dont MXNet (avec Keras, Python3 ou 2, et Cuda9).
MXNet embarqué dans AWS n'a pas pour le moment de spécificité, sauf qu'il peut interagir avec les autres services de AWS.

Par conséquent, il y a peu de différenciation de AWS avec d'autres hébergeurs pour un hébergement de MXNet. A contrario, citons l'offre d'appliance OVH dédiée à l'IA, qui, en plus du support des technologies open source, embarque des GPU de NVIDIA, faits pour les calculs d'IA.

TENSORFLOW

Développé par Google initialement pour l'offre Google Cloud Platform, puis libéré en open source fin 2015, TensorFlow est une librairie complète dédiée au Deep Learning.

Portable sur plusieurs plateformes (écrit en C++), TensorFlow est accessible via une API en Python, cette librairie donne accès à toutes les fonctions de Deep Learning.
Google investit massivement dans l'IA et TensorFlow est un projet très dynamique, avec des releases régulières, une communauté forte qui a rapidement adopté ce produit. Depuis sa parution en open source, c'est tout pour TensorFlow, et les autres ont du mal à exister.

TensorFlow s'intègre dans un écosystème technique, avec Hadoop, Spark, Jupyter, sans pour autant les embarquer. TF fonctionne sur des CPU, GPU, TPU (le processeur dédié créé par Google) et fonctionne en mode distribué.

Du fait de sa popularité, des tutoriels sur TensorFlow sont nombreux sur internet, sur beaucoup de sujets, et permettent d'attaquer des sujets en reprenant les travaux précédents, sans tout réinventer.

Sur les types de réseaux supportés, il est difficile de trouver une différence entre les solutions. Tous supportent les RNN, CNN, et même les tout récents CapsNet.
L'une des forces de TensorFlow est l'extension vers le cloud de Google, et ses fonctions natives de ML. Notons que AWS ET Azure supportent aussi TF, mais sans fonctionnalité supplémentaire.

API

L'API est disponible à plusieurs niveaux : « bas niveau », où l'on peut définir toutes ses variables une par une, avec une verbosité certaine …

Et « haut niveau », avec l'« Estimator API », où l'Estimator regroupe les fonctions de 'training', 'evaluation', 'prediction' et 'export for serving'

```
# extrait de https://www.tensorflow.org/programmers_guide/estimators
def input_fn(dataset):    ...  # manipulate dataset, extracting
feature names and the label    return feature_dict, label

# Define numeric feature columns population =
tf.feature_column.numeric_column('population') crime_rate =
tf.feature_column.numeric_column('crime_rate') median_education =
tf.feature_column.numeric_column('median_education',
                     normalizer_fn='lambda x: x -
global_education_mean')
```

```
# Instantiate an estimator, passing the feature columns. estimator =
tf.estimator.Estimator.LinearClassifier(
     feature_columns=[population, crime_rate, median_education],
     )
# my_training_set is the function created in Step 1
 estimator.train(input_fn=my_training_set, steps=2000)
```

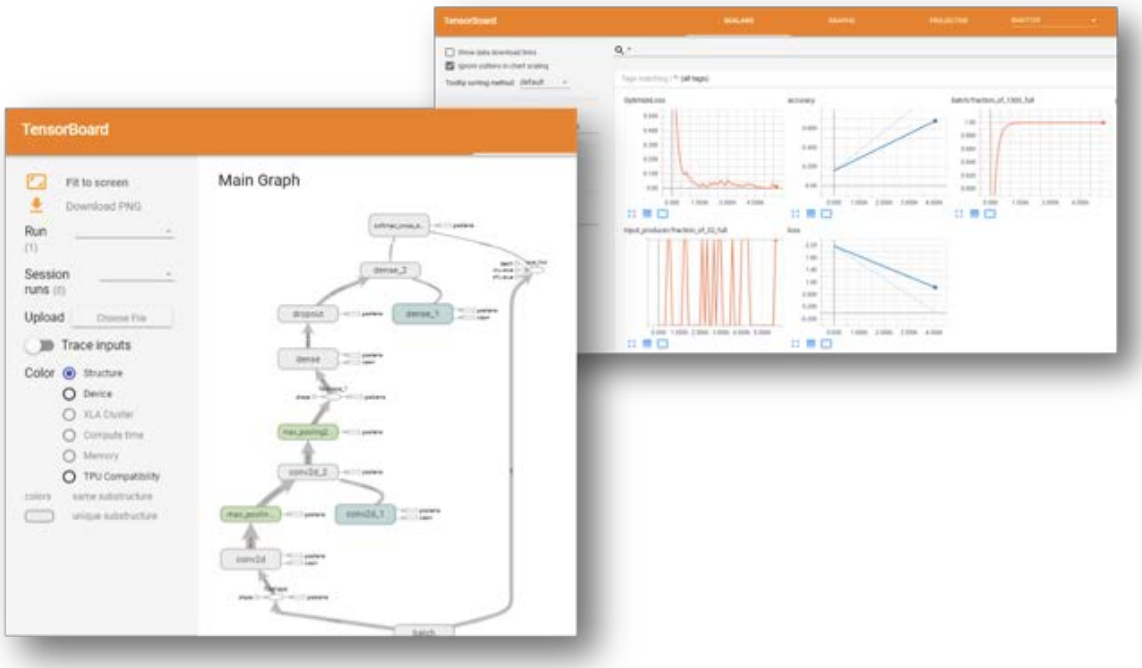
Malgré ces 2 niveaux d'API, plusieurs projets se sont créés ou adaptés pour se positionner au-dessus de TensorFlow et simplifier son usage. Citons Keras et H2O, décrits dans ce livre blanc, mais aussi tflearn.

TensorBoard

TensorBoard apporte une interface graphique à TensorFlow sur plusieurs aspects. TensorBoard est intégré à TensorFlow, l'affichage se fait dans un navigateur, avec un rafraîchissement automatique.

TensorBoard permet de générer des graphiques à partir des données loguées par TensorFlow. Les graphiques générés permettent de suivre la précision, la fonction de perte, … pendant l'entrainement d'un modèle.

En plus des graphiques, TensorBoard peut afficher le graphe du modèle que l'on a défini. L'affichage peut s'avérer incompréhensible par défaut, et il faut ajouter des noms et des regroupements de noms dans le code pour simplifier le graphe global.
TensorBoard permet aussi de comparer visuellement les résultats de différentes versions de ses modèles.



TensorFlow intègre un debugger accessible graphiquement via TensorBoard depuis la version 1.7. L'interface permet de visualiser les valeurs de toutes les variables et paramètres utilisés pendant la phase d'apprentissage. Cela peut permettre par exemple de voir directement qu'une fonction d'activation tend vers 0 et d'aller identifier le code ou les paramètres qui en sont à l'origine. Encore une fois, il faudra pour cela bien nommer tous ses groupes pour avoir une visualisation compréhensible.

Déploiement dans GCP

C'est le même TensorFlow qui est disponible en open source et qui est disponible dans Google Cloud Platform, dans la fonction « ML Engine ».

Aussi il est possible de travailler sur TF localement, puis de déployer TF dans GCP, sur un cluster de serveurs, simplement en ligne de commande. Les clusters peuvent être utiles pour le stockage et la lecture des données source, en utilisant le service Hadoop intégré à GCP. Et aussi pour l'exécution en parallèle de la phase d'entrainement.

Enfin, le modèle peut être déployé en production dans Cloud ML Engine en tant que service, appelable en API par des services tiers.

HyperTune

Dans le Deep Learning, on distingue les paramètres 'appris' par le RN et les hyper-paramètres définissant le RN. Pour le data scientist, c'est autant de leviers à manipuler, et autant de complexité à maîtriser, sans compter le temps passé à essayer les différentes possibilités.

Dans les hyper-paramètres, il y a le nombre de couches, le nombre de neurones de chaque couche, la nature de chaque couche, le learning_rate, …

La fonction 'HyperTune' disponible dans CloudML (et uniquement ici) permet d'évaluer automatiquement différents hyper-paramètres. Pour cela il suffit de lui préciser les différentes valeurs ou plages de valeurs à utiliser, et le logiciel s'en charge. On peut ensuite visualiser les résultats et choisir la meilleure configuration.

```
hyperparameters:
    goal: MINIMIZE
    maxTrials: 30
    maxParallelTrials: 1
    params:
    - parameterName: hidden_units
      type: CATEGORICAL
      categoricalValues: ["128 128 128", "256 128 16", "512 128 64"]
```

Pour les curieux, il s'agit d'une optimisation bayésienne des hyper-paramètres ; des arrêts anticipés automatiques des apprentissages permettent de réduire les temps d'apprentissage, qui se cumuleraient sinon pour toutes les valeurs possibles.

HyperTune/AutoML est plutôt adapté au transfer learning, pour lequel on ne vient modifier les hyper-paramètres que des dernières couches du modèles, et qui ont déjà des résultats comparables pour le modèle réutilisé.

Runtime pour Android

TensorFlow dispose d'un runtime pour Android et iPhone.

Ce qui veut dire que, une fois un modèle développé et validé, il peut être déployé dans une application mobile, et fonctionner sans faire appel à un serveur tiers.

La reconnaissance d'éléments dans une image par exemple, une fois implémentée avec TensorFlow, et déployée dans une app, peut fonctionner avec la caméra et surveiller en continu l'apparition d'un objet. Ou éviter de déclencher l'alarme s'il détecte un mouvement mais qu'il détecte que c'est seulement le chat qui se balade dans la maison. Comme quoi, finalement, savoir reconnaitre un chat peut s'avérer utile !

KERAS

Keras est une API Python de haut niveau fonctionnant au-dessus des librairies de DeepLearning TensorFlow, Theano, CNTK ou MXNet.

Keras se veut plus simple à utiliser que ces librairies, avec un code plus compact – donc moins de code à écrire, moins de syntaxe précise dont il faut se souvenir-, des fonctions déjà packagées, des exemples de jeux de données et un jeu de modèles réutilisables.

La définition des modèles est particulièrement simplifiée.
Un modèle 'Sequentiel' va juste empiler les couches que l'on va définir une à une, en une ligne chacune.

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(units=128, activation='relu', input_dim=100))
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='sigmoid'))
```

Utilisation du backend TensorFlow

Par défaut Keras utilisera TensorFlow comme backend de Deep Learning, qui reste le backend recommandé ici.

Keras permet d'utiliser nativement des modèles à convolution, de les empiler, d'y intégrer une régularisation 'dropout', une compression 'pooling' … pour obtenir des modèles complexes.

Dans l'exemple ci-dessous, le modèle empile plusieurs couches, avec une syntaxe relativement courte. Ce modèle, extrait d'un tutorial sur Keras, est un modèle à convolution servant pour la classification d'images.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(3, 150, 150)))
model.add(Activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3))
model.add(Activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

 model.add(Flatten())
model.add(Dense(512, activation='relu'))
```

```
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

Keras a rapidement été adopté par la communauté et est maintenant intégré à TensorFlow (`tf.keras`) pour proposer directement une API simplifiée, tout en restant disponible indépendamment, compatible avec les autres backends. Notons que Keras a été développé par François Chollet, chercheur en IA chez Google, ce qui peut aussi expliquer les choix logiciels.
Un certain nombre d'exemples sont disponibles pour Keras et permettent de l'utiliser assez rapidement.

Keras est destiné à simplifier les phases de recherche et de prototypage pour les data scientists, et c'est vrai que le code est plus court, plus simple qu'avec TensorFlow.

Keras peut aussi servir en production, même s'il faudra alors coder les notions de gestion d'erreur, de versionning de modèle, de tenue en charge, … qu'ont nativement d'autres produits comme TensorFlow Serving.

CONCLUSION

Le Deep Learning, et à travers lui l'Intelligence Artificielle, crée une rupture technologique majeure. Comme dans les précédentes révolutions industrielles, il y aura ceux qui auront embrassé ces nouvelles technologies, et ceux qui ne l'auront pas fait, ou en retard.

De nombreux usages sont déjà possibles, unitaires pour la plupart, mais opérationnels et impactant à divers endroits la performance des entreprises.

Implémentables avec les solutions open source citées ici – H2O, MXNet, TensorFlow et Keras - les cas d'applications du Deep Learning vont pouvoir être déployés dans des environnements professionnels et apporter du ROI.

Au-delà de l'architecture technique à mettre en place, le choix de la bonne problématique à résoudre et la méthodologie adaptée seront déterminants pour l'efficience et la réussite d'un projet.

De plus, chaque cas nécessitera une dose plus ou moins grande de personnalisation, et forcément d'expérimentation.

Mais chaque cas permettra aussi d'alimenter le savoir collectif, de le partager, pour faire en sorte que les cas suivants puissent s'en inspirer et aller encore plus loin

---

ANNEXE - RESSOURCES COMPLEMENTAIRES

Internet regorge de millions de ressources sur le Machine Learning et le Deep Learning.

Plusieurs ressources – articles et tutoriels – sont citées dans ce livre blanc, mais d'autres mériteraient d'être cités, pour aller plus loin sur le sujet, pouvoir lire l'avis d'experts, suivre des tutoriels pour expérimenter soi-même …

LECTURES

- Machine learning needs machine teaching de Mark Hammond
- Why AlphaGo Zero is a Quantum Leap Forward in Deep Learning de Carlos E. Perez
- Ideas on interpreting machine learning de Patrick Hall, Wen Phan et SriSatish Ambati
- Le mécanisme d'attention en IA de P. Lemberger
- Deep Learning in a Nutshell: Reinforcement Learning de Tim Dettmers
- Industrial AI: 6 Questions to Ask Before Implementing Your Enterprise AI Strategy de Dave Cahill
- Hyperparameter tuning in Cloud Machine Learning Engine using Bayesian Optimization de Puneith Kaul et Greg Kochanski
- Darren's Developer Diary
- Deep Residual Learning, MSRA @ ILSVRC & COCO 2015 competitions de Xiangyu Zhang, Shaoqing Ren, Jifeng Dai et Jian Sun
- Deep Learning for Plant Identification in Natural Environment de Yu Sun, Yuan Liu, Guan Wang, and Haiyan Zhang
- Introduction to Local Interpretable Model-Agnostic Explanations (LIME) de Marco Tulio Ribeiro, Sameer Singh & Carlos Guestrin
- Capturing your dinner, a deep learning story
- A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CN

TUTORIELS

- Building TensorFlow on Android
- How to classify images with TensorFlow using Google Cloud Machine Learning and Cloud Dataflow
- A simple deep learning model for stock price prediction using TensorFlow
- Create a chatbot with TensorFlow
- Image retraining
- Applying Deep Learning to Time Series Forecasting with TensorFlow
- How to build your own AlphaZero AI using Python and Keras
- Building powerful image classification models using very little data
- Keras as a simplified interface to TensorFlow: tutorial
- A Word2Vec Keras tutorial
- Deep matrix factorization using Apache MXNet
- Deep Learning - The Straight Dope
- https://github.com/h2oai/h2o-tutorials/