Apprentissage par renforcement profond

Nathan Barloy
Telecom Nancy
Universite de Lorraine
Email : nathan.barloy@telecomnancy.eu

Aurélien Delage
Telecom Nancy
Universite de Lorraine
Email: aurelien.delage@telecomnancy.eu

Olivier Buffet
and Vincent Thomas
INRIA
Equipe : Larsen
E-mail : olivier.buffet@loria.fr
and vincent.thomas@loria.fr

Résumé—

I. Introduction

Les débuts de l'intelligence artificielle - ou du moins la perception de l'éventualité de créer, un jour, des machines intelligentes qui surpasseraient les humains - peuvent être attribués aux travaux d'Alan Turing. Il proposa en 1936 une théorie formelle de ce qui sera plus tard un ordinateur. Une Machine de Turing est un concept abstrait, qui consiste en un ruban sur lequel se déplace une tête d'écriture, permettant de réaliser des opérations simples (arithmétiques, par exemple) [2].
14 ans plus tard, Alan Turing présente le concept de Test de Turing. Il concerne directement l'intelligence artificielle en proposant un test qui permettrait de mesurer la qualité de l'intelligence de la machine. L'expérience consiste à faire discuter un humain et une machine, l'humain ne sachant pas s'il discute avec un autre humain ou une machine, et simplement à lui demander à la fin de la discution, s'il pense avoir parlé avec un humain ou avec une machine. A ce jour, aucune intelligence artificielle n'a ***réellement*** réussi à passé le Test de Turing [3] [6].

Cela commence à nous amener à notre sujet. Depuis ces définitions formelles d'Alan Turing, la qualité des intelligences artificielle n'a eu de cesse de croître.
En effet, on peut citer la célèbre victoire de l'algorithme *Deep Blue* contre M.Garry Kasparov [4], champion du monde d'échec, en 1996. On peut aussi citer la victoire d'*AlphaGo* battant en 2017 le champion du monde du jeu de Go Lee Sedol. Sa version adaptée au jeu d'échec est aussi particulièrement impressionnante tant dans l'originalité du jeu proposé que dans l'incapacité des maîtres d'échec de comprendre réellement, et d'anticiper les coups de l'algorithme [5].

Et voilà notre sujet: il s'avère que ces deux algorithmes ont été développés par Google^{tm} en utilisant des techniques de ***Deep Reinforcement Learning***, ou autrement dit : ***Apprentissage par Renforcement Profond***.

Dans ce PIDR, nous nous proposons d'appliquer les techniques de base d'Apprentissage par Renforcement Profond à des jeux de type Atari.

A. Machine Learning, Réseaux de neurones

Le Machine Learning (ou Apprentissage Automatique) consiste, entre autre, à faire apprendre à un ordinateur à réagir, à résoudre un problème, en réaction à un certain jeu de données en entrée. Une des technique principale du Machine Learning est l'utilisation de réseaux de neurones, et c'est ce qui est utilisé pour l'Apprentissage par Renforcement Profond. Nous allons donc nous concentrer dessus pour cette partie.

1) Réseaux de Neurones: Qu'est-ce qu'un réseau de neurone?
Commençons par dire - de manière très schématique - que c'est un système, composés de couches et de noeuds, qui permet de répondre quelque chose (une solution) à un jeu de donnée en entrée (un problème). Disons aussi qu'ils réalisent des performances particulièrement impresionnantes sur certains problèmes.

Disons aussi qu'outre les performances sur les jeux, concernant la voiture autonome, ou tout autre application des réseaux de neurones, il est à noter que les algorithmes utilisant des réseaux de neurones tendent à percevoir la ***sémantique des mots et des idées***. Par exemple, l'équation roi-homme+femme est résolue par l'algorithme en donnant la solution: reine. Les mots ne sont plus des chaînes de caractères, ils sont des objets sémantiques en espaces de très grande dimension [7].

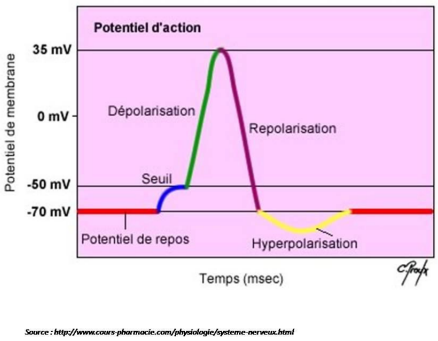
*Nous recommandons **très vivement** au lecteur de regarder la référence citée ci-dessous, ainsi que la série de vidéos de vulgarisation qui l'accompagne, présentant de manière formelle la perception de la sémantique des mots en tant qu'objets d'espaces de grandes dimension, que l'on cherche à applatir en espaces de dimension raisonnable, permettant l'analyse algorithmique,..*

Un réseau de neurone tend doncre à réfléchir, penser comme un humain. Ceci est la production d'un réseau de neurones de $Google^{tm}$ à qui l'on a demander de penser, de créer un tableau artistiques.

De même, la startup *DeepL* a réussi à produire un algorithme traduisant des livres de manière stupéfiante. Leur IA semble avoir appris la sémantique des mots [9]

2) Comment fonctionne un réseau de neurones?:

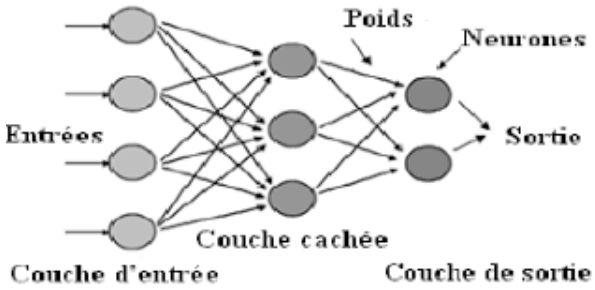
Premièrement, un neurone informatique est construit pour fonctionner de la même façon qu'un neurone humain. Le réseau de neurone fonctionne alors comme le réseau de neurones humain.
Un neurone n'est rien de plus qu'une cellule du cerveau humain recevant des impulsions électrique au cous du temps. Si le potentiel électrique est supérieur à un seuil, ce neurone est alors excité [8]:



Source : http://www.cours-pharmacie.com/physiologie/systeme-nerveux.html

C'est alors la combinaison de ces neurones interconnectés en réseau qui permet la puissance de l'intellect humain.

Un réseau de neurone informatique a pour but d'imiter ce comportement. Un réseau peut être représenté schématiquement de la façon suivante :



On remarque que toutes les valeurs de la couche i-1 entrent en entrée de tous les neurones de la couche i ($i \neq 1$). A chacune de ces valeurs est associé un poids.

Ecrivons les équations, ce sera plus simple.
Notons n_i^j le neurone i de la couche j, $\forall i \in [0,p], \forall j \in [0,n]$, en supposant qu'il y a n couches, et p neurones à la couche i.

Alors on a :

$$\boxed{n_i^{j+1} = \sum_{k=1}^{p}(n_k^j \cdot W_{k,i}^j) + b} \quad (1)$$

où on note $W_{k,i}^j$ les poids, $k \in [0,p]$ associés aux neurones $i \in [0,p]$ de la couche j.

On voit que l'on peut réecrire cette formule sous forme matricielle. Prenons soin de définir correctement les vecteurs et les matrices utilisés.

- $\forall j \in [0,n]$, $n^j \in R^p$
- $\forall j \in [0,n]$, $W^j \in M_{n,p}$
- $\forall j \in [0,n]$, $b \in R^p$

et on a :

$$\boxed{n^{j+1} = W^j \cdot n^j + b} \quad (2)$$

Les poids pouvant être grand, $\forall j \in [0,n]$, $n^j \subset R^p$. On aimerait qu'à l'instar du neurone humain, son état soit activé ou non activé, ie que l'on ait :

$$\forall j \in [0,n],\ n^j \subset [0,1]^p. \quad (3)$$

Ceci se fait en utilisant une fonction d'activation, ie une fontion de $R \mapsto [0,1]$ *ou* $[-1,1]$. Différentes fonctions existent [9], la plus utilisée est la fonction sigmoïde : $s_a(x) = \frac{1}{1+e^{-a}}$.

On remarque que la fonction f pour passer d'une couche à une autre est une fonction affine, de partie linéaire $\mathcal{L}(f) = W$ qui n'est pas nécessairement bijective. f ne l'est alors pas. Ceci montre que l'on peut obtenir des mêmes résultats tout en ayant des poids sur les neurones totalement différent. Cela a une importance quand on test la sensibilité du réseau à des petites pertubations non prévues : si les poids sont trop grands, il y a un risque de propagation de l'erreur et de divergence par rapport à la solution attendue.

On sait maintenant comment réagit un réseau de neurone à un jeu de données en entrée. Mais comment fait-il pour résoudre des problèmes, pour apprendre à donner la bonne solution?

3) Apprentissage du réseau de neurone:

Premièrement, les techniques d'Intelligence Artificielle à base de réseaux de neurones sont des techniques dîtes ***supervisées***, ie des techniques dont une partie des données pour l'apprentissage seront utilisées afin de vérifier la validité de l'apprentissage.

Les neurones humains ont la faculté de renforcer leurs interconnexions. En effet, quand on apprend une nouvelle langue, certaines connexions entre neurones se renforcent, et deviennent de plus en plus facile à emprunter, d'où l'apprentissage de la langue. Les réseaux de neurones utilisent le même procédé.

L'idée est de renforcer certaines connexions en donnant un poids plus important entre neurones, ie un coefficient de la matrice W fort.

Pour apprendre l'Anglais, il faut pratiquer. Pour un réseau de neurones, c'est la même chose! On nourrit le réseau avec un grand nombre de données en entrée et de leurs résultat, plusieurs fois, ce qui lui permet d'apprendre les poids de ses matrices W de chaque couche, afin d'avoir une prédiction la plus fiable possible.

4) Technique de rétro-propagation de l'erreur, du gradient.:

A chaque étape (chaque fois qu'on donne un jeu de donnée au réseau de neurones), le réseau doit mettre à jour ses poids. Une des techniques les plus connues pour faire ceci - et que nous utiliserons dans notre projet - est la ***Rétro-propagation du gradient***.

Pour l'expliquer, il faut d'abord préciser ce que l'on cherche à faire. Globalement, on cherche à minimiser la différence entre les prédictions du réseau de neurone, et les résultats connus.

Le résultat de sortie sera nécessairement un vecteur $y_{pred} \subset R^p$, p pouvant valoir 1.
On définit alors ***la fonction de perte*** L comme étant une mesure de la différence entre y_{connu} et y_{pred}. On peut alors utiliser plein de fontions différentes, comme $||y_{connu} - y_{pred}||_1$, $||y_{connu} - y_{pred}||_2$, $||y_{connu} - y_{pred}||_\infty$,.. Ce sont des applications différentiables. En effet, si on leur associe un produit scalaire, alors $||x|| =< x,x >$, or le produit scalaire étant une application bilinéaire de $R^p \times R^p \mapsto R$, elle est différentiable. En effet, on a, pour toute application bilinéaire f d'un espace vectoriel E dans un espace vectoriel F, la propriété suivante :

$$\boxed{\exists C > 0,\ \forall,\ x \in E^n,\ ||f((x_1,..x_n))||_F \leq C \prod_{i=1}^{n} ||x_i||_E \text{s}} \tag{4}$$

et alors pour $n = 2$ ($n \in N$ marche de la même façon, mais les notations sont plus compliquées),

$$\boxed{\begin{aligned} &||f(x+h) - f(x) - f(x_1+h_1,x_2) - f(x_1,x_2+h_2)|| \\ &= ||f(x_1+h_1,x_2+h_2)|| \leq C \prod_{i=1}^{2} ||h_i||_E \leq C \cdot ||h||_\infty^2 \\ &= o_{h \to 0_E}(||h||) \end{aligned}} \tag{5}$$

La fonction de perte est alors différentiable, sauf en 0, on peut donc ***suivre son gradient*** pour minimiser la fontion de perte, ie $\tilde{x} = x - \alpha \nabla_x(f)$, avec α un coefficient arbitraire. En effet, géométriquement, la différentielle d'une fonction dans un espace vectoriel (ici R^p) correspond à l'hyperplan tangent à la surface.

Dans R^p, $\nabla_x(f)(h) = Jac_x(f) \cdot h$, où $Jac_x(f)$ est la jacobienne de f en x définie par (pour $f : R^p \mapsto R$) :

$$\nabla_x(f) = \left(\frac{\partial f}{\partial x_1}(x),..,\frac{\partial f}{\partial x_n}(x)\right) \tag{6}$$

et on a l'équation suivante de l'hyperplan tangeant à $f(x)$:

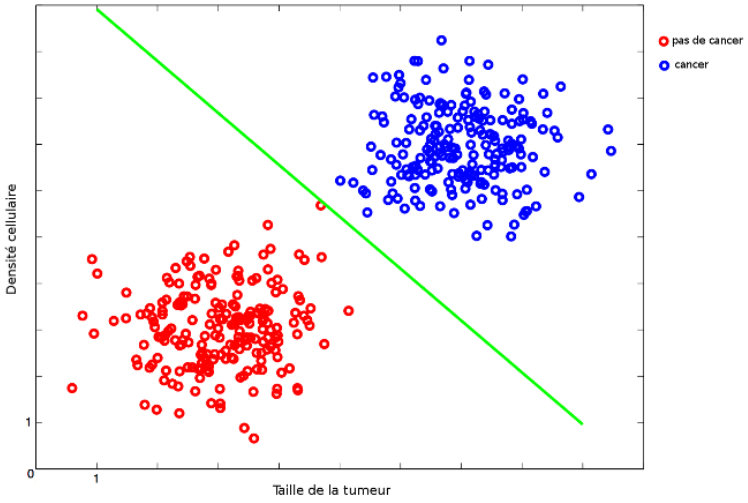
$$\boxed{y(h) = f(x) + \nabla_x(f) \cdot h} \tag{7}$$

Et alors, $-\nabla_x(f)$ donne la direction dans laquelle la fontion est plus faible. En se déplaçant alors selon cette direction, on minimise la fonction f (il faut régulièrement mettre à jour le gradient, si la fonciton est irrégulière).

La mise à jour des poids est un peu complexe et lourde en notation, nous envoyons le lecteur sur la page wikipédia l'expliquant [11].

C'est sur la mise à jour des poids selon cette technique que tout l'apprentissage repose.

5) Resultats, visualisation: On l'a dit tout à l'heure, un neurone se calcule de la façon suivante (1) $n_l^{j+1} = \sum_{k=1}^{p}(n_k^j \cdot W_k^{j,l}) + b$. Il s'agit d'un produit scalaire auquel on ajoute la constante b. Ceci est ***exactement*** l'équation d'un hyperplan de dimension p-1 dans un espace de dimension p. On peut alors interpréter le travail d'un neurone dans un réseau de neurone comme la délimitation d'un hyperplan séparant en deux les données en entrée de la couche. On peut alors visualiser le fonctionnement d'un neurone, par exemple pour la classification de données, dans R^2 [12]:



La compléxité des réseaux de neurones (nombre de neurones), nombre de couches cachées) dépend naturellement de la complexité d'isoler des points dans un espace de dimension n. Les surfaces de séparation peuvent aussi être plus compliquées que des droites..
Ainsi, la reconnaissance de chiffres entre 0 et 9 peut se faire avec le nombre de pixels comme nombre de neurones d'entrée, deux couches de 16 neurones, et 10 neurones de sortie qui correspondent chacun à l'activation pour le chiffre 0, le chiffre 2,.., le chiffre 9 [13].

B. Apprentissage par Renforcement, ou Q-learning

1) Qu'est-ce que l'apprentissage par Renforcement?: Etant donné que nous voulons appliquer ces méthodes aux jeux, définissons-les.
On peut voir une bonne partie des jeux comme un arbre des possibilités - potentiellement infini -, avec des actions faisant office de transition et des récompenses liées à ces actions (un score,..).

Formellement, cela peut s'écrire de la manière suivante
Un jeu est composé des différents éléments suivants:

- S, un ensemble d'états possiblement atteingnable dans le jeu
- A l'ensemble des actions réalisables

et on définit alors la fonction suivante :

$$\begin{array}{rccc} r & : & S \times A & \rightarrow & R \\ & & (s,a) & \mapsto & r(s,a) \end{array}$$

qui est appelée ***fonction de récompense***. Quand je suis dans l'état S et que j'effectue l'action a, alors je récupère une récompense $r(s,a)$, à mon score, par exemple. Cette fonction est au départ de l'apprentissage inconnue. C'est une fonction que l'algorithme construit en se basant sur les résultats qu'il obtient en jouant.

Néanmoins, je n'ai pas forcément intérêt à aller chercher la récompense la plus haute immédiatemment. Dans bon nombre de jeux comme les échecs, ou le jeu de go, j'ai des fois intérêt à faire des sacrifices temporaires pour un gain plus grand plus tard. On aimerait alors disposer de la fonction suivante, appelée ***Stratégie***:

$$\begin{array}{rccc} \Pi & : & S & \rightarrow & A \\ & & s & \mapsto & \Pi(s) \end{array}$$

De plus, dans un jeu, je ne peux pas nécessairement faire ce que je veux, quand je veux. Notamment dans les jeux comportant du hasard, la réalisation d'une action a étant donné un état s est soumise au hasard. Si je la choisis, elle a une probabilité $p(s,a)$ de se réaliser.
On suppose alors disposer aussi de la fonction p:

$$\begin{array}{rccc} p & : & S \times A & \rightarrow & [0,1] \\ & & (s,a) & \mapsto & p(s,a) \end{array}$$

Connaître cette fonction p correspond, en fait, à connaître les règles du jeu. Néanmoins, si cette fonction n'est pas donnée, on peu facilement l'estimer à l'aide d'estimateurs statistiques classiques qui estiment la fonction de répartition de cette probabilité p. Si l'action est binaire : soit je réussis à faire l'action, soit j'échoue, avec une probabilité α inconnue, alors on peut approximer la probabilité p comme suivant une loi Binomiale de paramètre $\tilde{\alpha}$ où $\tilde{\alpha} \in [0,1]$ et est estimé avec un estimateur de la moyenne classique :

$$\tilde{\alpha} = \frac{1}{n} \sum_{i=0}^{n} x_i \tag{8}$$

où les $x_{i \in \{0,..,n\}}$ sont des observations de réalisation ou d'échec de mon action a.

2) Equation de Bellman: En reprennant les notations de la partie précédente, on peut envisager une méthode pour apprendre à jouer correctement. On peut se dire qu'étant donné un état s, mon intérêt à faire l'action a_s dépend de toutes les récompenses qui suivront et des probabilités de réalisation de toutes les actions qui suivront. Prendre le maximum sur tous ces chemins de l'arbre des possibilités du jeu semble particulièrement naturel.

$$\boxed{V(s) = max_{a \in S} \left(\{ r(s,a) \cdot p(s,a) + \sum_{i=0}^{\infty} V_a(s_i) \cdot p(s,a) \right)} \tag{9}$$

Cette équation est appelée ***Equation de Bellman***

Néanmoins, les actions qui viendront dans 10 actions dans le futur doivent naturellement avoir un poids moins important que les actions immédiates. On ajoute alors un poids $\gamma \in [0,1[$ ($\gamma = 0$ revient à choisir la récompense immédiate la plus forte et $\gamma = 1$ empêche la convergence), et on aboutit alors à l'équation suivante :

$$\boxed{V(s) = max_{a \in S} \left(\{ r(s,a) \cdot p(s,a) + \sum_{i=0}^{\infty} \gamma V_a(s_i) \cdot p(s,a) \right)} \tag{10}$$

3) Méthode de Monte Carlo:

II. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

RÉFÉRENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] http://zanotti.univ-tln.fr/turing/

[3] https://culture.univ-lille1.fr/fileadmin/lna/lna66/lna66p04.pdf

[4] $https://fr.wikipedia.org/wiki/Matchs_Deep_Blue_contre_Kasparov$

[5] https://www.youtube.com/watch?v=bo5plUo86BU

[6] https://www.csee.umbc.edu/courses/471/papers/turing.pdf

[7] https://www.youtube.com/watch?v=r4EtRfHSU7E

[8] http://www.cours-pharmacie.com/physiologie/systeme-nerveux.html

[9] https://www.lebigdata.fr/deep-learning-ia-traduction

[10] http://www.statsoft.fr/concepts-statistiques/reseaux-de-neurones-automatises/reseaux-de-neurones-automatises.htm.XGlNLlwzZPZ

[11] https://fr.wikipedia.org/wiki/Rétropropagation_du_gradient

[12] https://markdown.data-ensta.fr/s/machine-learning-introductions

[13] https://www.youtube.com/watch?v=aircAruvnKkt=303s