Résolution de Formules Booléennes Quantifiées à l'aide de Réseaux de Neurones

Université Paris-Diderot

Mohammed Younes Megrini
Nicolas Nalpon

6 juin 2016

Table des matières

1 Introduction

Il y a 20 ans, Deep Blue, un programme développé par IBM, remportait une victoire historique aux échecs contre Garry Kasparov, le champion mondiale d'échecs à l'époque. En mars 2016, AlphaGo, développé par Google Deepmind, vient de remporter une victoire au jeu de Go contre Lee Sedol, l'un des meilleurs joueurs de Go au monde. Le jeu de Go est considéré comme un défi majeure de l'intelligence artificielle et cette victoire témoigne d'avancées très prometteuses.

Au cœur d'AlphaGo, l'apprentissage profond est la technologie qui est en phase de révolutionner le domaine de l'intelligence artificielle. Faisant partie de la catégorie des algorithmes évolutifs et, plus particulièrement, des algorithmes d'apprentissage automatique, l'apprentissage profond consiste à optimiser progressivement une succession de transformations non-linéaires des données afin de résoudre des problèmes d'analyse et de classification. Grâce à cette approche, on arrive à développer des algorithmes de classification (reconnaissance faciale et vocale, vision par ordinateur, reconnaissance d'écriture manuscrite...) ou de décision (AlphaGo pour le jeu de Go, Deep Q-Learner pour les jeux Atari...).

Notre projet consiste à utiliser l'apprentissage profond pour développer une méthode heuristique de traiter le problème de la satisfaisabilité des formules booléennes quantifiées (dit QBF-SAT) qui est une généralisation du problème de satisfaisabilité booléenne (dit SAT). Nous nous limiterons aux formules normales conjonctives prénexes suivant le format standard QDimacs.

Comme il est possible de voir une formule booléenne quantifiée comme un jeu à deux adversaires et que l'existence d'une stratégie gagnante pour l'un des deux joueurs détermine la valeur de vérité de la formule, notre algorithme consistera à faire s'affronter les deux stratégies opposées, en les améliorant progressivement, de sorte à en déduire la valeur de vérité de la formule.

Dans ce rapport, nous allons premièrement décrire le fonctionnement des réseaux de neurones et du processus d'apprentissage et les illustrer par un exemple théorique simple. Ensuite, on décrira en détail le fonctionnement de l'algorithme présenté plus haut.

2 Les Réseaux de Neurones

Le premier modèle mathématique et informatique du neurone biologique a été donné par Warren McCulloh et Walter Pitts en 1943. Ce modèle inspiré de la neurobiologie, fait intervenir n entrées auxquelles des fonctions sont appliquées afin d'obtenir une sortie. Voici une définition plus formelle de ce modèle qu'on nomme: Modèle de McCulloh-Pitts

2.1 Définition: Modèle de McCulloh-Pitts

Soit la fonction

$$\begin{array}{rccc} \varphi & : & \mathbb{R}^n & \to & \{0,1\} \\ & & (x_1,....,x_n) & \mapsto & \varphi(x_1,....,x_n) \end{array}$$

telle que

$$\varphi(x_1,...,x_n) = g(f(x_1,....,x_n)).$$

1. f est définie comme étant la fonction somme

$$\begin{array}{rccc} f & : & \mathbb{R}^n & \to & \mathbb{R} \\ & & (x_1,....,x_n) & \mapsto & \sum_{i=1}^{n} x_i \end{array}$$

2. On appelle g la fonction d'activation à seuil. Celle-ci est définie de la manière suivante :

$$g(x) = \begin{cases} 0 & \text{Si } x < \theta \\ 1 & \text{Si } x \geq \theta \end{cases}$$

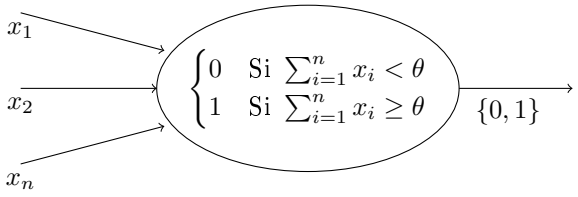

figure 1 - Modèle de McCulloh-Pitts

Remarque : La valeur θ qu'on utilise dans la fonction g est appelée le seuil.

Le modèle de McCulloh-Pitts a été décliné de plusieurs manières au cours du temps. Les deux déclinaisons auxquelles nous allons nous intéresser sont appelées le modèle Perceptron et le modèle sigmoïde. Ces deux modèles introduisent la notion de poids qui est au cœur du processus d'apprentissage d'un réseau de neurone.

2.2 Définition: Modèle Perceptron

Le premier modèle du Perceptron a été introduit en 1958 par Frank Rosenblatt. Celui-ci a ensuite été repris et perfectionné dans les années soixante par Minsky et Papert. Ce modèle reprend la définition précédente sauf que pour chaque entrées x_i ($i \in [\![1\ ;\ n]\!]$) on associe une valeur (les fameux poids) $w_i \in \mathbb{R}$. On définit alors la fonction f de la façon suivante :

$$\begin{array}{rccc} f & : & \mathbb{R}^n & \to & \mathbb{R} \\ & & (x_1, ...x_n) & \mapsto & \sum_{i=1}^{n} w_i x_i \end{array} .$$

Remarque : On note parfois

$$b = -\theta$$

et on appel cette valeur l'inclinaison (*bias* en anglais). La fonction g est alors définie de la manière suivante :

$$g(x) = \begin{cases} 0 & \text{Si } x + b < 0 \\ 1 & \text{Si } x + b \geq 0 \end{cases}$$

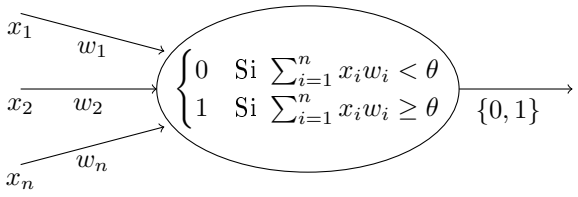

figure 2 - Perceptron

Cette notion de poids va intervenir dans le processus d'apprentissage d'un réseau de neurone. Le but de ce processus sera d'optimiser les poids associés aux neurones afin que le réseau de neurone produise la meilleure sortie possible. L'un des algorithmes d'optimisation le plus utilisé est l'algorithme de la descente de gradient. Nous détaillerons plus tard cet algorithme mais le principe de celui-ci est d'ajuster progressivement les poids afin d'affiner la sortie du réseau de neurone vers la valeur voulue. Cet affinement implique que la sortie $s \in [0, 1]$ et que la fonction g ne soit plus une fonction en escalier mais une fonction continue. Le neurone Perceptron ne répondant pas à ces critères, le neurone Sigmoïde a été introduit.

2.3 Définition: Modèle sigmoïde

Le modèle sigmoïde est défini de la même manière que le modèle Perceptron mais avec une fonction g différente. La fonction g aura pour particularité d'être continue. La fonction choisie est la fonction sigmoïde.

$$\begin{array}{rccc} g & : & \mathbb{R} & \to & [0,1] \\ & & x & \mapsto & \frac{1}{1+e^{-x}} \end{array}$$

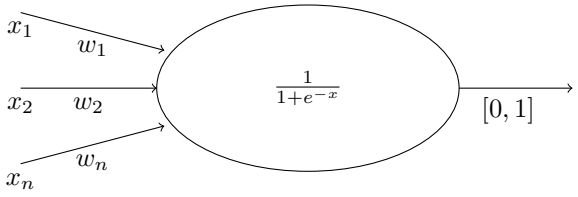

figure 3 - Sigmoïde

2.4 Définition: Réseau de Neurone

Un réseau de neurone est un graphe orienté connexe dont les nœuds sont les neurones.

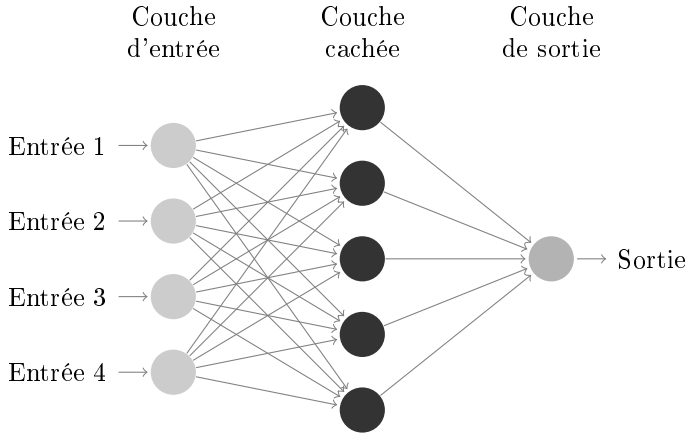

figure 4 - Réseau de neurones

Ce réseau est composé de plusieurs couches :

1. la couche d'entrée : Cette couche est composée de neurones dont leurs entrées n'est pas la sortie d'autre neurone.

2. la couche cachée : Cette couche peut être composée de plusieurs couches de neurones. Ils prennent en entrée les sorties des neurones de la couche précédente.

3. la couche de sortie : Cette couche nous donne le résultat du réseau. Elle peut être formée d'un ou de plusieurs neurones.

Nous avons défini dans cette partie qu'es ce qu'était un réseau de neurone et comment etait-il formé. Nous allons maintenant voir comment nous pouvons utiliser celui-ci dans divers problèmes.

3 Le processus d'apprentissage

Un réseau de neurones, avec une configuration donnée, peut être vu comme une classe de fonctions paramétrée par l'ensemble des poids et des inclinaisons. L'apprentissage consiste donc à rechercher la fonction optimale, au sein de cette classe, selon un critère donné, d'où l'utilisation d'une fonction de coût permettant d'évaluer l'optimalité d'une instance particulière du réseau.

3.1 Définition: Fonction d'erreur quadratique moyenne

Soit C_x une fonction telle que :

$$\begin{array}{rccc} C_x & : & \mathbb{R}^{n+1} & \to & \mathbb{R} \\ & & (w_1, ..., w_{n+1}) & \mapsto & \frac{1}{2}(y(x) - \Phi_x(w_1, ..., w_{n+1}))^2 \end{array} .$$

avec $(w_1, ..., w_n)$ un vecteur poids, w_{n+1} une valeur de seuil choisie pour le réseau de neurone et x correspondant à une entrée choisit dont $y(x)$ est la valeur attendue après l'application du réseau. Cette fonction est appelée la fonction d'erreur quadratique moyenne.

Remarque :

1. On définit E comme étant un ensemble d'entrées pour un réseau de neurone particulier dont le résultat voulu est connu.

2. La fonction C_x définie ne concerne qu'une valeur de l'ensemble particulière $x \in E$. Afin que le réseau de neurone soit le plus performant possible, nous optimiserons la fonction suivante $\gamma(w_1, ..., w_{n+1}) = \sum_{x \in E} C_x(w_1, ..., w_{n+1})$.

3. L'optimisation de la fonction γ dépendra de l'ensemble E fourni.

L'optimisation de γ nous permettra de trouver la famille $(w_1, ..., w_{n+1})$ qui permettra d'approximer γ vers zéro. Nous allons effectuer cette optimisation à l'aide de l'algorithme de descente de gradient.

3.2 Définition: Gradient

Soit $f : \mathbb{R}^n \to \mathbb{R}$ une fonction à plusieurs variables à valeurs réelles. On appelle gradient de f, noté ∇f

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

3.3 Algorithme de descente du Gradient

Cet algorithme va permettre de générer une suite $x_1, ..., x_k \in \mathbb{R}^{n+1}$ jusqu'à que la condition d'arrêt soit satisfaite. On note $x_1, ..., x_k$ les différents vecteurs poids que l'algorithme calculera.

Algorithm 1 La Descente de Gradient

Require: $x_0 \in \mathbb{R}^{n+1}$ vecteur poids qu'on choisit initialement
  c nombre d'étape maximum qu'on autorise
  $\varepsilon \geq 0$ seuil qu'on choisit que le gradient ne doit pas franchir
  $k = 0$
  while $|| \nabla f(x_k) || \geq \varepsilon$ and $k \neq c$ **do**
    Calcul de $\nabla f(x_k)$
    Calcul de $\alpha_k > 0$
    $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$
    k++;
  end while

Remarque : α_k qu'on appelle le pas d'apprentissage doit être choisi avec précaution. En effet, si celui-ci est trops grand, l'algorithme peut amener à une oscillation autour du minimum et si il est trop petit la convergence vers le minimum se ferra très lentement. On calcul le pas d'apprentissage à l'aide de l'algorithme suivant:

Algorithm 2 Calcul du pas d'apprentissage

Require: x_0 vecteur poids qu'on choisit initialement
  $k = 0$
  Calculer $\nabla f(x_K)$
  Choisir α_k afin de minimiser la fonction $h(\alpha) = f(x_k - \alpha \nabla f(x_k))$;

Remarque :

1. α_k sera bien-sûr la valeur pour laquelle on aura $h'(\alpha_k) = 0$.
2. Dans les deux algorithmes précèdent le vecteur x_0 est choisi aléatoirement.

3.4 Exemple

Le but de cet exemple est de montrer comment un réseau de neurone N peut apprendre à déterminer la position des points d'un plan par rapport à une droite.

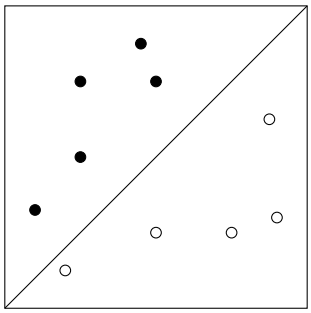

figure 5 - exemple 1

Pour cela nous allons introduire un réseau de neurone composé d'une unité de type Sigmoid. Ce neurone aura pour entrées les coordonnées du point choisi et pour sortie la valeur de la fonction d'activation qui est définie comme suit:

$$f(u,v) = \frac{1}{1+e^{-w_1 u - w_2 v + \theta}}.$$

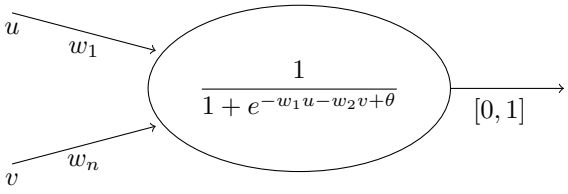

figure 6 - exemple 2

Soit la fonction $\Phi_x(w_1, w_2, \theta) = \dfrac{1}{1 + e^{-w_1 u - w_2 v + \theta}}$ avec $x = (u, v)$.

Soit E un ensemble de triplets de la forme : $x = (u, v, p)$ avec u et v l'abscisse et l'ordonnée du point x et p sa position par rapport à la droite d.

$$\begin{cases} 0 & \text{Si p est en dessous de d} \\ 0.5 & \text{Si p est sur d} \\ 1 & \text{Si p est au dessus de d} \end{cases}$$

Soit $C_x(w_1, w_2, \theta) = \frac{1}{2}(y(x) - \Phi_x(w_1, w_2, \theta))^2$ la fonction de coût avec $y(x) = p$. Il suffit ensuite de minimiser le gradient de la fonction γ, décrite précédemment, afin d'obtenir les poids et inclinaison pour lesquelles le réseau N nous donne des valeurs de position par rapport à la droite d, par interpolation sur les triplets de E.

3.5 Rétropropagation

Nous avons vu dans la partie précédente, comment nous pouvons minimiser la fonction γ à condition d'avoir son gradient. Le calcul du gradient devient très vite compliqué lorsque le réseau de neurone est formé de plusieurs couches de plusieurs neurones. Ce calcul est possible, néanmoins, grâce à la méthode de rétropropagation du gradient.

Avant de parler de la rétropropagation, nous devons introduire une nouvelle structure qui est le B-Diagramme (*Backpropagation diagram*). Ce diagramme représente les neurones utilisés dans notre réseau mais dans lesquelles nous stockons dans leurs partie gauche la valeur de la dérivée de leurs fonction et dans leurs partie droite la valeur de leurs fonction pour une entrée donnée.

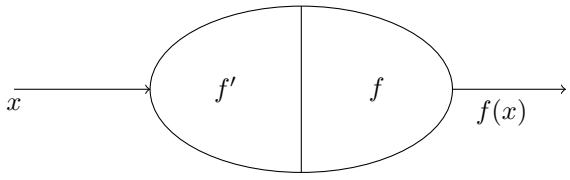

figure 7 - B-Diagramme

La rétropropagation se divise en deux étapes :

1. L'étape de *feed-forward* : Cette étape consiste à parcourir notre réseau dans le sens direct afin de calculer pour chaque neurone, la valeur de sa

fonction et de sa dérivée pour une valeur donnée. Une fois calculés, ces valeurs seront stockées dans chacun des neurones. Le résultat stocké à droite sera ensuite transmit au nœud suivant et le processus sera réitéré.

2. L'étape de rétropropagation : Dans cette étape le réseau sera parcouru dans le sens indirect et la valeur stockée à gauche sera utilisée, selon les cas suivants :

   (a) La composition : Le réseau prend en entrée la constante 1. Celle-ci est multipliée par la valeur stockée dans la partie gauche du neurone. Le résultat sera ensuite transmis à l'unité suivante dans laquelle la constante sera multipliée par sa valeur de gauche. Le résultat final sera alors égale à $f'(g(x)).g'(x)$.

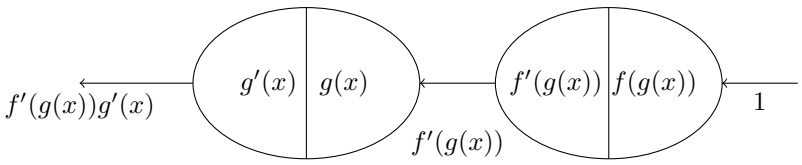

figure 8 - B-Diagramme

   (b) L'addition : Le réseau prend en entrée la constante 1, qui sera ensuite distribuée dans chaque unité. Lorsque deux unités se rencontre on additionne le résultat

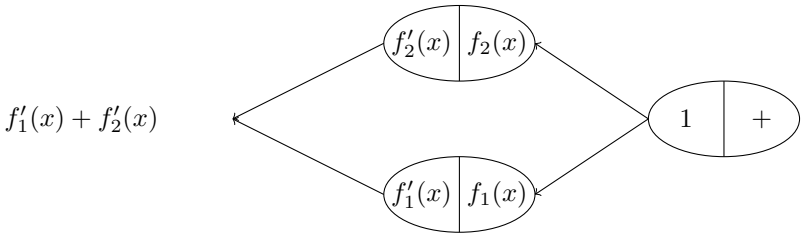

figure 9 - B-Diagramme

(c) Les entrées avec poids :

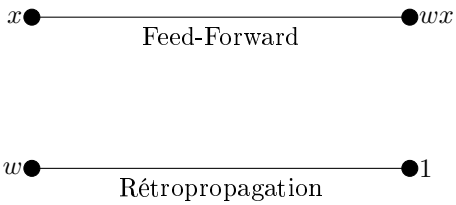

figure 10 - B-Diagramme

4 Résolution de QBF-SAT

Dans la partie précédente nous avons vu comment le processus d'apprentissage fonctionne. Dans cette partie, nous allons voir comment nous pouvons entrainer un réseau de neurone afin que celui-ci puisse résoudre des formules booléennes quantifiées. Afin qu'un réseau de neurone puisse résoudre une formule QBF, il faut que celle-ci soit mise sous forme prénexe.

4.1 Définition : Forme prénexe

On dit que une formule et sous forme prénexe si tout ses quantificateurs existentiels et universels sont placés au début de la formule.

$$G = Q_1 x_1 Q_2 x_2 .... Q_n x_2 G'$$

avec G' une formule booléenne sans quantificateur.

4.2 Définition : Forme Normale Conjonctive (CNF)

Soit F une formule booléenne et $(C_i)_{1 \leq i \leq n}$ une famille de clauses telles que

$$F = C_1 \wedge C_2 \wedge ... \wedge C_n$$

On dit alors que F est sous forme normale conjonctive.

4.3 Format QDimacs

Le format standard QDimacs permet de décrire des formules booléennes quantifiées qui sont prénexes et sous forme normale conjonctive. Il est compatible avec le format standard Dimacs des SAT-*solver*.

La première ligne d'un fichier de type QDimacs est toujours de la forme "p cnf v c" où v est le nombre de variables et c le nombre de clauses que contient la formule. Les variables étant numérotées de 1 à v.

Le fichier contient ensuite un nombre potentiellement nul de lignes, se terminant par 0, et décrivant l'ordre et la nature des quantificateurs. Une telle ligne définit des variables quantifiées existentiellement si elle commence par le caractère e, et universellement si elle commence par a.
Toute variable qui n'apparait dans aucune de ces ligne est supposée quantifiée existentiellement au tout début de la formule.

Les lignes suivantes du fichier décrivent les clauses de la formule une à une. Chaque ligne se termine par 0 et liste toutes les variables de la clause, les variables sont négatives si elles apparaissent niées dans la clause, et positives sinon.

Exemple de fichier en format QDimacs :

```
p cnf 5 2
a 1 4 0
e 3 0
a 2 0
1 -2 4 0
-3 4 5 0
```

Ce fichier décrit la formule suivante :

$$\exists x_5 \forall x_1 \forall x_4 \exists x_3 \forall x_2 \quad (x_1 \wedge \neg x_2 \wedge x_4) \vee (\neg x_3 \wedge x_4 \wedge x_5)$$

4.4 Choix des réseaux de neurones

Pour résoudre une formule on utilisera deux réseaux qui s'affronteront dans un jeu. Un réseau représentera les quantificateurs universels et l'autre représentera les quantificateurs existentiels. Celui qui représentera les quantificateurs universels aura pour but de choisir des valuations qui permettront de rendre la formule fausse. Le réseau qui représentera les quantificateur existentiels aura pour but de choisir des valuations qui permettront de rendre vraie la formule. Au fur et mesure de l'exécution de l'algorithme, le pourcentage de victoires du réseau existentiel tendra vers 0% si la formule est fausse et vers 100% si elle est vraie. Les deux réseaux de neurone auront la même structure.
La taille de l'entrée du réseau de neurone est le nombre de variables de la formule donnée.

Leur structure est donnée par le schéma suivant pour une formule :

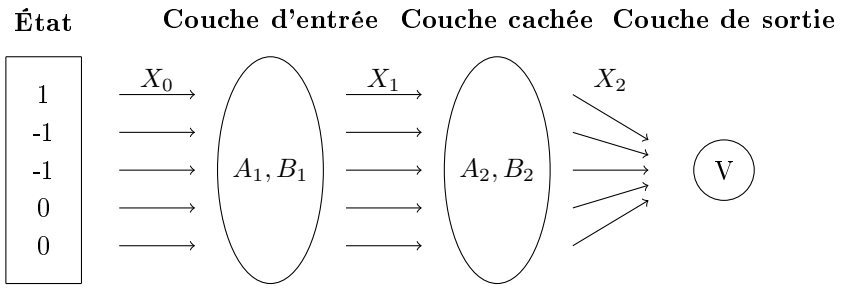

figure 11 - Structure des réseaux de neurones (taille 5)

Chaque réseau de neurone prend, en entrée, un état particulier du jeu (représentant une valuation partielle) et renvoie sa valeur $V \in [0, 1]$.
Chaque neurone d'une couche de intermédiaire calcule l'image par la fonction sigmoïde d'une somme linéaire pondérée avec inclinaison de ses entrées. La transformation effectuée par la couche $i + 1$ se traduit donc par :

$$X_{i+1} = S(A_{i+1}X_i + B_{i+1})$$

où :

- X_i : vecteur reçu en entrée de couche
- X_{i+1} : vecteur obtenue en sortie de couche
- S : calcule l'image coordonnée par coordonnée du vecteur $A_{i+1}X_i + B_{i+1}$ par la fonction sigmoïde
- A_{i+1} : matrice des poids
- B_{i+1} : vecteur d'inclinaison

La matrice des poids de la couche d'entrée est une matrice carrée, celle de la couche cachée est une matrice ligne.

4.5 Description formelle de l'algorithme

L'algorithme consiste, pendant un nombre d'itérations donné k, à faire s'affronter les deux réseaux de neurones dans le cadre du jeu existentiel/universel décrit précédemment. À chaque itération, on recueille de nouvelles données d'expérience pour chacun des réseaux. On les ajuste selon le résultat de la partie puis on les rajoute aux données d'entrainement qu'on utilise pour le processus d'apprentissage. Les différentes étapes de chaque itération sont les suivantes :

Algorithm 3 Deep QBF-SAT solver

Require: k
  while $cmp < k$ **do**
    s, r = result(session())
    us, uv, es, ev = build(s, r);
    store(us, uv, es, ev);
    train(uni, uniMset, uniMval);
    train(exi, exiMset, exiMval);
    percentage(r)
    $cmp++$;
  end while

1. **session:** calcule une session du jeu existentiel/universel et renvoie une matrice contenant la valuation choisie et les valeurs renvoyées par les réseaux de neurones pour chacun des choix
2. **result:** prend en entrée une session et renvoie un couple (s, v) où s correspond à ladite session et r correspond à la valeur de vérité de la formule pour la valuation contenue dans s et leau session
3. **build:** prend en entrée le couple (s, v, r) et renvoie les données d'expérience us, uv pour le réseau *UNI* et es, ev pour le réseau *EXI*, us et es contiennent les états rencontrés par les deux réseaux, uv et ev contiennent les nouvelles valeurs cibles pour la phase d'apprentissage
4. **store:** prend en entrée les données d'expérience us, uv et es, ev et les enregistre respectivement dans les données d'apprentissage $uniMset, uniMval$ et $exiMset, exiMval$
5. **train:** effectue la phase d'apprentissage pour chacun des réseaux en utilisant les données d'apprentissage $uniMset, uniMval$ et $exiMset, exiMval$ en utilisant comme critère la fonction d'erreur quadratique moyenne et comme algorithme d'optimisation la descente de gradient
6. **percentage:** calcule le pourcentage de victoire du réseau *UNI* et l'affiche

Ainsi donc, la convergence vers 0% ou 100% du pourcentage affiché par le programme permet d'obtenir une indication concernant la valeur de vérité de la formule donnée.

5 Implémentation effective

Le programme que nous avons décrit plus haut a été implémenté à l'aide d'un langage de script s'appelant Lua et en utilisant la librairie Torch, conçue pour le domaine d'apprentissage automatique. Il est fournit en complément de ce rapport.

5.1 Lua

Lua est un langage de script, développé par une équipe de chercheurs brésiliens de l'Université pontificale Catholique de Rio de Janeiro, qui a le mérite d'être rapide et portable (car implémenté en C) en plus d'être facilement utilisable avec d'autres langages tel que le Java, C#, Fortan ou bien d'autre langage de script tel que le Perl ou le Ruby. Il est celui utilisé par la librairie Torch.

5.2 Torch

Torch comme est une librairie d'algorithme scientifique du langage Lua conçu pour l'apprentissage automatique. Cette librairie, permet d'avoir accès aux outils nécessaires à l'utilisation de l'apprentissage profond. Elle offre une implémentation efficace et simple des réseaux de neuronnes ainsi que de plusieurs algorithmes d'optimisation.

6 Conclusion

Ce projet s'inscrit dans une démarche expérimentale. De ce fait, l'algorithme proposé est tout à fait original.
La résolution de formules booléennes quantifiées reste un problème complexe. En conséquence, notre algorithme souffre de sérieuses limitations :

1. L'apprentissage effectué est spécifique à une formule particulière et n'est donc pas réexploitable
2. L'exécution est assez lente
3. Le résultat obtenu reste purement heuristique et n'offre pas de certitude quand à la satisfaisabilité de la formule

Pour remédier à ces limitations, par exemple, on pourrait utiliser un modèle de réseaux de neurones plus complexe ou changer l'ensemble sur lequel on effectue l'apprentissage pour tenter de créer un algorithme plus général qui soit réexploitable. On pourrait aussi essayer d'optimiser l'utilisation de la mémoire afin d'accélérer l'exécution.
L'utilisation de l'apprentissage automatique pour la résolution de QBF-SAT reste très prometteuse avec beaucoup de pistes à explorer.

7 Références

[1] Raul Rojas. Neural Networks, 1996.