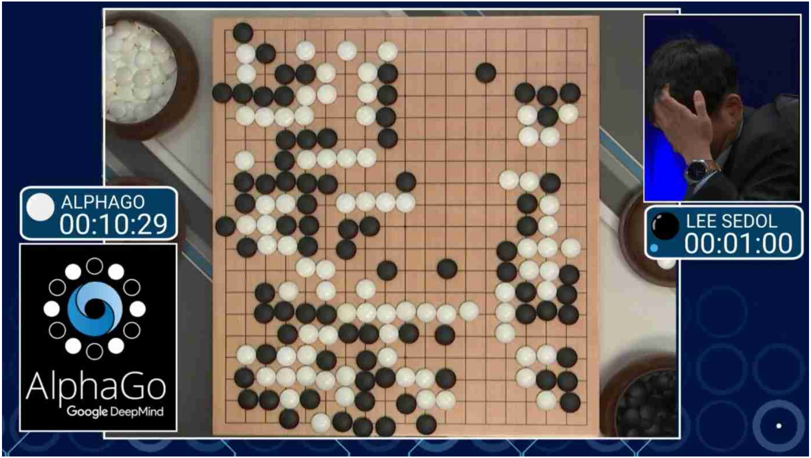
Résolution de Formules Booléennes Quantifiées à l'aide de Réseaux de Neurones

Université Paris-Diderot

Mohammed Younes Megrini
Nicolas Nalpon

10 juin 2016

Introduction : AlphaGo



Introduction : Deep-Q Learning Atari

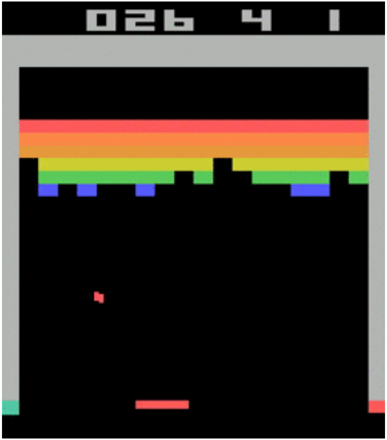
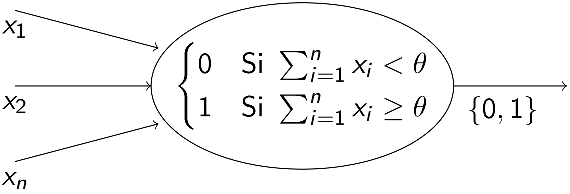


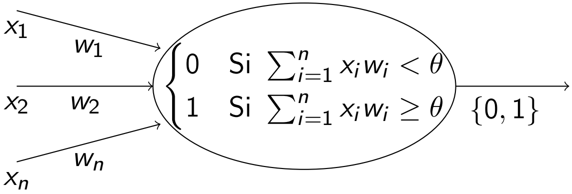
Table des matières

Table des matières

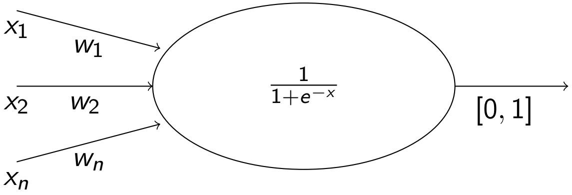
Modèle de McCulloh-Pitts



Neurone Perceptron



Neurone Sigmoïde



Réseau de Neurones

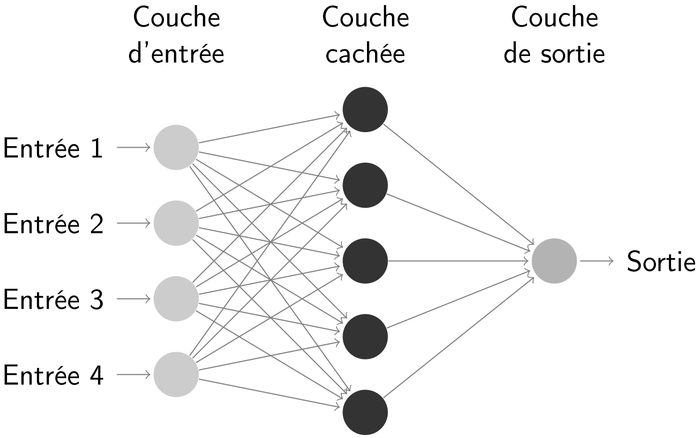


Table des matières

Fonction d'erreur quadratique moyenne

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

Descente de Gradient

Algorithm 1 Descente de Gradient

Require: $x_0 \in \mathbb{R}^{n+1}$, $c \in \mathbb{N}$, $\varepsilon \geq 0$

$k = 0$
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

Descente de Gradient

Algorithm 2 Calcul du pas d'apprentissage

Require: x_0 vecteur poids qu'on choisit initialement

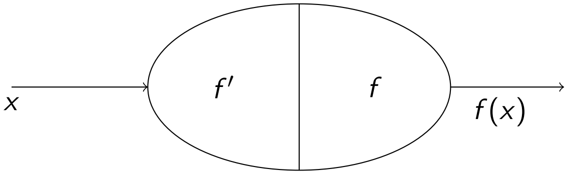
$k = 0$

1: Calculer $\nabla f(x_K)$

2: Choisir α_k afin de minimiser la fonction $h(\alpha) = f(x_k - \alpha \nabla f(x_k))$;

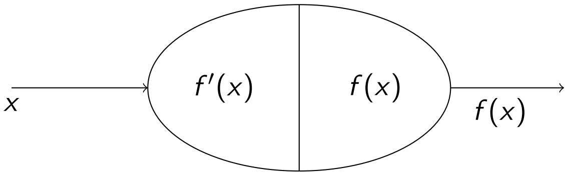
Table des matières

Rétro-propagation

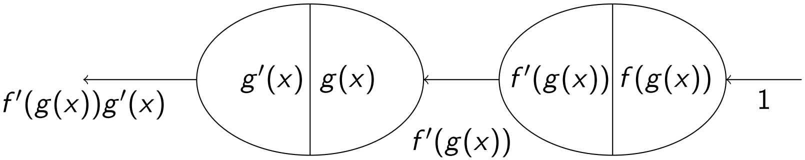


B-Diagramme

Rétro-propagation : *feed-forward*

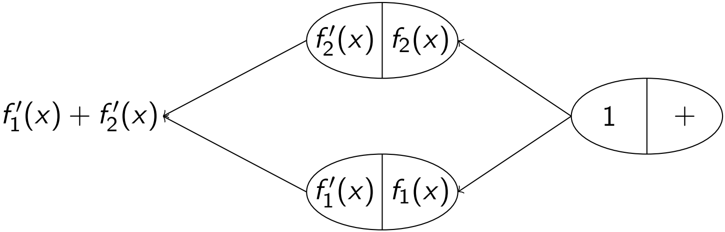


Rétropropagation : *backpropagation*



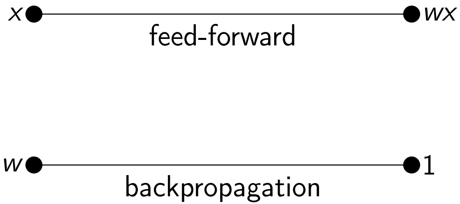
Cas 1: Composition

Rétropropagation : *backpropagation*



Cas 2 : Somme

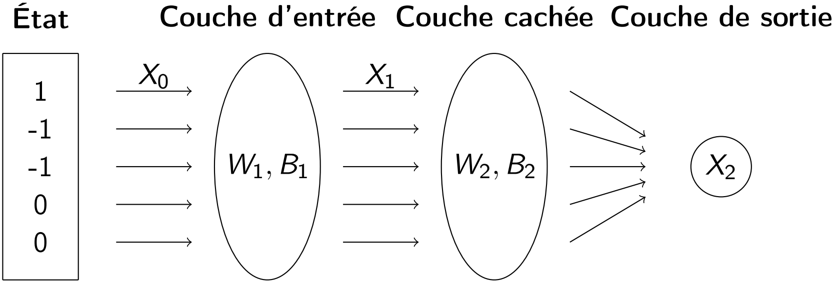
Rétropropagation : *backpropagation*



Cas 3 : Poids

Table des matières

Modèle des deux joueurs



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

Table des matières

Librairie Torch

torch

A SCIENTIFIC COMPUTING FRAMEWORK FOR LUAJIT

Librairie Torch

```
function init()
    local net = nn.Sequential()
    net:add(nn.Linear(variables, variables))
    net:add(nn.Sigmoid())
    net:add(nn.Linear(variables, output))
    net:add(nn.Sigmoid())
    return net
end
```

Format QDimacs

Formule booléenne quantifiée normale conjonctive :

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

Format QDimacs

Formule booléenne quantifiée normale conjonctive :

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

Format QDimacs :

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

Algorithme d'apprentissage de QBF-SAT

Algorithm 3 Deep QBF-SAT Learning

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
```

Conclusion

```
--------------------------------------------------------------------------------

Session 9944:
 1.0000 -1.0000 -1.0000  1.0000
 0.0066  0.9914  0.0055  0.9914
[torch.DoubleTensor of size 2x4]

Result: 0 (var 2)

Training set:

uni:
 1  1  0  0
 1  1 -1  1
[torch.DoubleTensor of size 2x4]

 0.9936
 1.0000
[torch.DoubleTensor of size 2]

exi:
 1  0  0  0
 1  1 -1  0
[torch.DoubleTensor of size 2x4]

0.001 *
 5.8034
 2.7921
[torch.DoubleTensor of size 2]


Loss: uni8.6061716698874e-05 exi1.2046354306021e-05


Percentage:
1%
--------------------------------------------------------------------------------

Session 10000:
 1.0000 -1.0000  1.0000 -1.0000
 0.0065  0.9915  0.0082  0.9915
[torch.DoubleTensor of size 2x4]
```