Apprentissage Automatique, Mini-projet

Mini-projet :

Réseau de neurones appliqué au

« Jeu des bâtons »

GUETTOUCHE Islam

Master 1 Génie de l'informatique logicielle (G.I.L)

01 Mai 2017

Table des matières

Introduction

Le but de ce Mini_Projet est d'apprendre à la machine comment jouer au mieux au "**jeu des bâtons**". Il existe en effet une "technique" pour gagner en fonction du nombre de bâtons à jouer et de quel joueur commence la partie. Nous allons utiliser une méthode de réseau de neurones simple (type perceptron) pour que la machine apprenne à jouer et découvre ainsi la "technique".

Dans notre programme, la machine pourra jouer selon trois modes :

- **Easy :** Elle joue aléatoirement 1, 2 ou 3 bâtons (dans la limite des règles, cf. ci-dessous).
- **Medium :** Elle joue aléatoirement sauf à la fin où elle ne commet pas d'erreurs évidentes.
- **Hard :** Elle joue via son réseau de neurones grâce à l'apprentissage.

Le travail va se diviser en trois phases :

- ✓ Prise en main du code existant, premiers tests de jeu.
- ✓ **Phase d'apprentissage :** on va entrainer le réseau de neurones en jouant un grand nombre de fois.
- ✓ **Phase de tests :** on va créer plusieurs scénarii pour observer les comportements de la machine en fonction de plusieurs paramètres et notamment vérifier si la machine joue bien parfaitement en mode "hard".

Objectifs de ce Mini_Projet :

Offrir une plateforme de jeu qui comporte quinze bâtons ou deux joueurs s'affrontent et doivent, chacun leur tour, prendre un, deux ou trois bâtons. Celui qui prend le dernier bâton a perdu.

Historique :

Le jeu des bâtonnets est un jeu de duel qui demande logique et stratégie. Il est connu sous le nom actuel de "**jeu de Nim**" a été donné par le mathématicien anglais Charles Leonard Bouton en 1901 qui a trouvé un algorithme permettant le gain.

En 1951, le Nimrod a été conçu, c'est le premier ordinateur dont le seul but est de permettre de jouer à un jeu, en l'occurrence le jeu de Nim.

Technologies utilisé :

Python est un langage de programmation objet, multi-paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet.

Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions, il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.

```
def add5(x):
   return x+5

def dotwrite(ast):
   nodename = getNodename()
   label=symbol.sym_name.get(int(ast[0]),ast[0])
   print ' %s [label="%s' % (nodename, label),
   if isinstance(ast[1], str):
      if ast[1].strip():
         print '= %s"];' % ast[1]
      else:
         print '"]'
   else:
      print '"];'
      children = []
      for n, child in enumerate(ast[1:]):
         children.append(dotwrite(child))
      print ' %s -> {' % nodename,
      for name in children:
         print '%s' % name,
```

Les bases et le mode simple :

Analyse rapide des classes:

- **La classe Game :**

Elle contient l'algorithme du déroulement d'une partie du jeu des bâtons.

- Son **constructeur « def __init__(self,nbSticks): self.nbSticks = nbSticks »** prend en paramètre le nombre de bâtons au début du jeu.
- La méthode **start « def start(self,player1,player2,verbose):»** permet d'effectuer une partie, si **verbose == True** et **nbSticks > 0** alors le message est afficher

```
New game
Remaining sticks: 15
Sticks?
```

Et les deux joueurs peuvent disputer une partie, la partie ce termine dès que **nbSticks == 0 « while sticks>0: (sticks = self.nbSticks)»**.

- **La classe Player :**

C'est la super-classe représentant un joueur.

- Son **constructeur « def __init__(self,name): self.name = name self.nbWin = 0»** prend en paramètre le nom du joueur.
- La méthode **getName « def getName(self): return self.name»** elle retourne le nom du joueur.
- La méthode **getNbWin « def getNbWin(self): return self.nbWin»** elle retourne le nombre des partie qui on étais gagner par ce joueur.
- La méthode **addWin « def addWin(self): self.nbWin+=1»** elle augmente le nombre de partie qui on étais gagner par ce joueur.
- La méthode **addLoss « def addLoss(self): pass»** elle ne fait rien.

- **La classe HumanPlayer :**

Elle est utilisée pour faire jouer un joueur humain au jeu des bâtons.

- **La classe CPUPlayer :**

Elle est utilisée pour faire jouer la machine au jeu des bâtons.

- Son **constructeur « def __init__(self,name,mode,nbSticks): super().__init__(name) self.mode = mode self.netw = NeuronNetwork(3,nbSticks) self.previousNeuron = None»** prend en paramètre le nom du joueur, le modes de jeux ainsi que le nombre de battons.

- **La classe NeuronNetwork :**

Elle représente un réseau de neurones, c'est-à-dire un ensemble d'instances de classe Neuron.

- **La classe Neuron :**

Elle représente le neurone de base d'un réseau de neurones. Chaque neurone contient des connexions vers d'autres neurones.

SCRIPT1 : Guettouche VS Machine en mode facile

```
from Game import *
from Neuron import *
from Player import *
import time

une_partie = Game(15)
Guettouche = HumanPlayer("Guettouche Islam")
Machine = CPUPlayer("Machine","easy",15)

une_partie.start(Guettouche,Machine,True)
time.sleep( 10 )
```

```
New game
Remaining sticks: 15
Sticks?
3

Guettouche Islam takes 3
Remaining sticks: 12
Machine takes 1
Remaining sticks: 11
Sticks?
3


Guettouche Islam takes 3
Remaining sticks: 8
Machine takes 1
Remaining sticks: 7
Sticks?
3


Guettouche Islam takes 3
Remaining sticks: 4
Machine takes 1
Remaining sticks: 3
Sticks?
2


Guettouche Islam takes 2
Remaining sticks: 1
Machine takes 1
Guettouche Islam wins!
```

Testes du SCRIPT1 :

L'ordinateur peut faire des erreurs évidentes lors du dernier tour car la version "easy" utilise seulement une fonction **random.randint(1, (sticks%3)+1)** pour jouer cette dernière génère un nombre entier aléatoirement entre **« 1 »** et **« le (reste de la division entre sticks = 15 et 3) plus 1 »**.

Mode intermédiaire : **Est-ce de l'apprentissage ?**

La machine n'a pas appris d'elle-même afin de s'améliorer pour parler d'apprentissage, c'est nous qui ont spécifié une action à réaliser selon une certaine situation pour permettre à la machine de s'adapter celons les cas possible et pour lui éviter de commette une erreur au dernier tour lorsqu'elle a toutes les chances de gagner.

```
def playMedium(self,sticks):
    if sticks > 4:
        return random.randint(1,3)
    elif sticks == 1:
        return 1
    else:
        return sticks - 1
```

SCRIPT 1.2 : Guettouche VS Machine en mode medium

```
from Game import *
from Neuron import *
from Player import *

une_partie = Game(15)
Guettouche = HumanPlayer("Guettouche Islam")
Machine = CPUPlayer("Machine","medium",15)

une_partie.start(Guettouche,Machine,True)
```

Implémentation du réseau de neurone :

- La méthode **testNeuron** de la **classe Neuron** :

```
def testNeuron(self,inValue):
    # TODO renvoie un booléen : True si la différence entre
    #la 'inValue' et la valeur du neurone actuel est comprise entre 1 et 3 inclus

    if inValue-self.index>=1 and inValue-self.index<=3:
        return True
    else:
        return False
```

- La méthode **chooseConnectedNeuron** de la **classe Neuron** :

```
def chooseConnectedNeuron(self,shift):
    neuron = None
    # TODO méthode qui retourne un neurone connecté au neurone actuel en fonction du 'shift' (cf. CPUPlayer).
    # On devra utiliser la méthode self.weighted_choice pour choisir au hasard dans une liste de connexions disponibles en fonction
    #on utilise une var neuron_test afin de gérer la boucle while
    #on utilise une var copie_connections afin d'avoir une copie de la liste des neurones connecté
    neuron_test = False
    copie_connections = self.connections.copy()
    #tant que neuron_test == False choisir au hasard un neuron dans la liste de connexions disponibles en fonction de leurs poids
    #on Enlève de la liste l'élément situé à la position indiquée
    while neuron_test == False:
        neuron = self.weighted_choice(copie_connections)
        #on Enlève de la liste l'élément situé à la position indiquée
        copie_connections.pop(neuron)
        #on teste si la différence entre la 'self.index-shift' et la valeur du neurone actuel est comprise entre 1 et 3 inclus
        neuron_test = neuron.testNeuron(self.index-shift)
        #si cest oui on retourne le neuron
        if neuron_test == True:
            return neuron
        #si le nombre d'éléments de la liste copie_connections est == 0 on retourne None
        if len(copie_connections) == 0:
            return None
```

- La méthode **playHard** de la **classe CPUPlayer** :

```
def playHard(self,sticks):

    # TODO utiliser le réseau neuronal pour choisir le nombre de bâtons à jouer
    # utiliser l'attribut self.previousNeuron pour avoir le neuron précédemment sollicité dans la partie
    # calculer un 'shift' qui correspond à la différence entre la valeur du précédent neurone et le nombre de bâtons encore en jeu
    # utiliser la méthode 'chooseConnectedNeuron' du self.previousNeuron puis retourner le nombre de bâtons à jouer
    # bien activer le réseau de neurones avec la méthode 'activateNeuronPath' après avoir choisi un neurone cible
    # attention à gérer les cas particuliers (premier tour ou sticks==1)
```

Premier Tour :

```
#si cest le premier tour (self.previousNeuron retourne None)
 if self.previousNeuron == None:
# en recupere neurons[sticks-1]
    self.previousNeuron = self.netw.getNeuron(sticks)
    neuron_previous = self.previousNeuron
#on fais appel a la methode chooseConnectedNeuron afin de choisir le nombre de bâtons à jouer
    after_use_of_chooseConnectedNeuron = neuron_previous.chooseConnectedNeuron(0)
    self.previousNeuron = after_use_of_chooseConnectedNeuron
#activer le réseau de neurones avec la méthode 'activateNeuronPath' après avoir choisi un neurone cible (after_use_of_chooseConnectedNeuron)
#self.path[neuron_previous]=after_use_of_chooseConnectedNeuron
    self.netw.activateNeuronPath(neuron_previous,after_use_of_chooseConnectedNeuron)

# Taux erreur
    if(after_use_of_chooseConnectedNeuron.index%4!=1 and neuron_previous.index%4!=1):
        self.nbErreur = self.nbErreur+1
    self.nbTour = self.nbTour+1
    return (neuron_previous.index-after_use_of_chooseConnectedNeuron.index)
```

Dernier Tour :

```
#si cest le dernier tour sticks==1 incrementer le nombre de tour et retourner 1
elif sticks==1:
    self.nbTour = self.nbTour+1
    return 1
```

Tours Intermédiaires :

```
#si self.previousNeuron != None et sticks!=1    alors
else:
#on recupere le neuron précédemment sollicité dans la partie
    neuron_previous = self.previousNeuron
#on fais appel a la methode chooseConnectedNeuron afin de choisir le nombre de bâtons à jouer
    after_use_of_chooseConnectedNeuron = neuron_previous.chooseConnectedNeuron(neuron_previous.index-sticks)
    self.previousNeuron = after_use_of_chooseConnectedNeuron
#activer le réseau de neurones avec la méthode 'activateNeuronPath' après avoir choisi un neurone cible (after_use_of_chooseConnectedNeuron)
#self.path[neuron_previous]=after_use_of_chooseConnectedNeuron
    self.netw.activateNeuronPath(neuron_previous,after_use_of_chooseConnectedNeuron)

# Taux erreur
    if(after_use_of_chooseConnectedNeuron.index%4!=1 and sticks%4!=1):
        self.nbErreur = self.nbErreur+1
    self.nbTour = self.nbTour+1
    return sticks-after_use_of_chooseConnectedNeuron.index
```

- La méthode **recompenseConnection** de la **classe Neuron** :

```
def recompenseConnection(self,neuron):
    # TODO récompenser la connexion entre le neurone actuel et 'neuron'
    self.connections[neuron] = self.connections[neuron]+RECOMPENSE
    pass
```

SCRIPT 1.3 : Machine VS Guettouche en mode difficile

```
from Game import *
from Neuron import *
from Player import *

une_partie = Game(15)
Guettouche = HumanPlayer("Guettouche Islam")
Machine = CPUPlayer("Machine","hard",15)

une_partie.start(Machine,Guettouche,True)
```

Apprentissage :

SCRIPT 2 : Machine1 VS Machine2 en mode difficile

Comment s'appelle cette méthode et pourquoi l'utilise-t-on ?

Le but derrière le fais de laisser l'ordinateur jouer contre lui-même en mode "hard est de faire de l'apprentissage non supervisé. En effet, avec le réseau de neurones, on va privilégier les décisions qui on était prise ainsi que le chemin par lequel il est passé lorsqu'il a gagné. On va donc attribuer des récompenses « **recompenseConnection(self,neuron)**» aux synapses que la machine a empruntées lorsqu'elle a gagnée, et ainsi on peut automatiser facilement cet apprentissage.

On va donc la faire jouer plusieurs centaines de fois voire plusieurs milliers de fois afin qu'elle puisse apprendre d'elle-même et pour la faire familiariser avec le plus de cas possible et ainsi elle gagne de l'expérience.

En laissant l’ordinateur jouer contre lui-même, que constatez-vous à la fin de N parties ?

On peut remarquer qu'après plusieurs centaines de fois voire plusieurs milliers de fois de parties disputer, la machine 1 cumule plus de victoire que la machine 2, vu qu'elle joue en premier, elle a pu acquérir plus d’expériences.

- **Machine1 :**

```
Nombres de victoires de Machine1:
510 / 800.

Connections de Machine1:
Connections of N1:
Connections of N2:
N1 10
Connections of N3:
N2 10
N1 226
Connections of N4:
N3 10
N2 42
N1 1186
Connections of N5:
N4 10
N3 26
N2 90
N1 2394
Connections of N6:
N5 10
N4 42
N3 146
N2 154
N1 242
Connections of N7:
N6 10
N5 82
N4 114
N3 514
N2 98
N1 82
Connections of N8:
N7 10
N6 42
N5 426
N4 1066
N3 170
N2 10
Connections of N9:
N8 10
N7 50
N6 114
N5 1850
N4 522
N3 34
Connections of N10:
N9 10
N8 26
N7 82
N6 146
N5 162
N4 10
Connections of N11:
N10 10
N9 26
N8 50
N7 146
N6 58
N5 10
Connections of N12:
N11 10
N10 34
N9 234
N8 1250
N7 242
N6 202
Connections of N13:
N12 10
N11 34
N10 66
N9 2202
N8 330
N7 178
Connections of N14:
N13 10
N12 74
N11 178
N10 282
N9 18
N8 10
Connections of N15:
N14 338
N13 2490
N12 1282
```

```
Connections neuronales pondérées de Machine1:
N1 4140
N2 404
N3 900
N4 1764
N5 2540
N6 572
N7 708
N8 1676
N9 2490
N10 392
N11 222
N12 1366
N13 2500
N14 338
```

- **Machine1 :**

```
Nombres de victoires de Machine2:
290 / 800.
```

```
Nombres de victoires de Machine
290 / 800.


Connections de Machine2:
Connections of N1:
Connections of N2:
N1 10
Connections of N3:
N2 10
N1 218
Connections of N4:
N3 10
N2 34
N1 490
Connections of N5:
N4 10
N3 26
N2 42
N1 1034
Connections of N6:
N5 10
N4 18
N3 562
N2 362
N1 514
```

```
Connections of N7:
N6 10
N5 34
N4 170
N3 226
N2 330
N1 114
Connections of N8:
N7 10
N6 26
N5 170
N4 506
N3 106
N2 26
Connections of N9:
N8 10
N7 154
N6 282
N5 594
N4 322
N3 74
Connections of N10:
N9 10
N8 250
N7 122
N6 826
N5 306
N4 90
```

```
Connections of N11:
N10 10
N9 10
N8 50
N7 442
N6 82
N5 10
Connections of N12:
N11 170
N10 114
N9 1170
N8 170
N7 26
N6 162
Connections of N13:
N12 394
N11 50
N10 1242
N9 58
N8 90
N7 10
Connections of N14:
N13 370
N12 146
N11 26
N10 10
N9 10
N8 10
Connections of N15:
N14 10
N13 10
N12 10
```

```
Connections neuronales pondérées de Machine2:
N1 2380
N2 804
N3 1004
N4 1116
N5 1124
N6 1388
N7 764
N8 580
N9 1258
N10 1376
N11 246
N12 550
N13 380
N14 10
```

SCRIPT 3 :

Expliquez la déférence de scores entre les 2 joueurs ?

- **Machine1 en mode EASY VS Machine2 en mode EASY|MEDIUM|HARD :**

```
La Machine 1 Easy : 2474 victoires sur 5000 parties.
La Machine 2 Easy : 2526 victoires sur 5000 parties.

La Machine 1 Easy : 881 victoires sur 5000 parties.
La Machine 2 Medium : 4119 victoires sur 5000 parties.

La Machine 1 Easy : 917 victoires sur 5000 parties.
La Machine 2 Hard : 4083 victoires sur 5000 parties.
```

On peut remarquer que la machine 2 cumule plus de victoire que la machine 1.

- ✓ **EASY VS EASY :** On voit que le score est vraiment serré comme les deux machines se pose sur la même stratégie de jeu alors c'est plus question de chance que de stratégie.

- ✓ **EASY VS MEDIUM:** On voit que la machine 2 mène la partie et avec un score large cela revient au fait que cette dernière se pose sur une stratégie plus élaborée que la machine 1.

- ✓ **EASY VS HARD :** On voit que la machine 2 mène la partie et avec un score large cela revient au fait que cette dernière se pose sur une stratégie plus élaborée que la machine 1.

- **Machine1 en mode MEDIUM VS Machine2 en mode EASY|MEDIUM|HARD :**

```
La Machine 1 Medium : 4044 victoires sur 5000 parties.
La Machine 2 Easy : 956 victoires sur 5000 parties.

La Machine 1 Medium : 2435 victoires sur 5000 parties.
La Machine 2 Medium : 2565 victoires sur 5000 parties.

La Machine 1 Medium : 2433 victoires sur 5000 parties.
La Machine 2 Hard : 2567 victoires sur 5000 parties.
```

  - ✓ **MEDIUM VS EASY :** On voit que la machine 1 mène la partie et avec un score large cela revient au fait que cette dernière se pose sur une stratégie plus élaborée que la machine 2.

  - ✓ **MEDIUM VS MEDIUM:** On voit que le score est vraiment serré, comme les deux machines se pose sur la même stratégie de jeu alors c'est plus question de chance que de stratégie.

  - ✓ **MEDIUM VS HARD :** On voit que le score est vraiment serré, mais n'empêche, on voit que la machine qui joue avec le mode hard gagne cela revient au fait que cette dernière se pose sur une stratégie plus élaborée que la machine 1.

- **Machine1 en mode HARD VS Machine2 en mode EASY|MEDIUM|HARD :**

```
La Machine 1 Hard : 3270 victoires sur 5000 parties.
La Machine 2 Easy : 1730 victoires sur 5000 parties.

La Machine 1 Hard : 3223 victoires sur 5000 parties.
La Machine 2 Medium : 1777 victoires sur 5000 parties.

La Machine 1 Hard : 3662 victoires sur 5000 parties.
La Machine 2 Hard : 1338 victoires sur 5000 parties.
```

  - ✓ **HARD VS EASY :** On voit que la machine 1 mène la partie et avec un score large cela revient au fait que cette dernière se pose sur une stratégie plus élaborée que la machine 2.

  - ✓ **HARD VS MEDIUM:** On voit que le score est vraiment serré, mais n'empêche, on voit que la machine qui joue avec le mode hard gagne cela revient au fait que cette dernière se pose sur une stratégie plus élaborée que la machine 2.

  - ✓ **HARD VS HARD :** On voit que le score est vraiment serré, comme les deux machines se pose sur la même stratégie de jeu alors c'est plus question de chance que de stratégie.

Phase d'apprentissage : l'enregistrement du réseau neuronal dans un fichier sérialisé

```
if (Machine1.getNbWin()>Machine2.getNbWin()):
    neurons = Machine1.getNeuronNetwork()
else:
    neurons = Machine2.getNeuronNetwork()

# Phase d'apprentissage : l'enregistrement du réseau neuronal dans un fichier sérialisé
with open('neuronNetwork_By_Guettouche.nnw','wb')
        as output: pickle.dump(neurons,output,pickle.HIGHEST_PROTOCOL)
```

Jeu final :

SCRIPT 4 :

En jouant contre l'ordinateur en mode Hard, en 2 ème joueur et après avoir Charger le réseau de neurones au démarrage pour le mode « **hard** », il est impossible de gagner, on a fait tellement gagner de l'expérience (scripte 3 : apprentissage) à la machine qu'elle arrive maintenant à assurer le résultat de la partie avant même que cette dernière se termine.

Tout au long de la partie, la machine suit des stratégies astucieuses quelle a pu acquérir tout au long de son apprentissage et qu'elle a pu les récupérer du réseau de neurones « **neuronNetwork_By_Guettouche.nnw** ».

Exmpl :

- ✓ S'il reste 5 batons et que le joueur en choisi 1, l'ordinateur en choisira 3 (Le joueur perd)
- ✓ S'il reste 5 batons et que le joueur en choisi 2, l'ordinateur en choisira 2 (Le joueur perd)
- ✓ S'il reste 5 batons et que le joueur en choisi 3, l'ordinateur en choisira 1 (Le joueur perd)

Etc.

Question optionnelle : évaluation :

Pour gagner une partie en étant le premier joueur, nous devons faire en sorte de tomber sur les bâtons restants : 13, 9 et 5. Si la première machine à chaque tour ne tombe pas sur un de ses chiffres après avoir joué (Multiples de 4 + 1), on comptabilise une erreur. A la fin de la partie, **on obtient le taux d'erreur en divisant le nombre d'erreurs par le nombre de coups au total**. Comme la Machine2 n'a pas trop le choix alors on ne compte pas forcément d'erreur.

On remarque bien que le taux d'erreur baisse et que les machines font moins d'erreurs plus les parties n'avancent, est cela, car les machines gagnent de l'expérience et qu'elles apprennent de leurs parties précédentes.

Afin d'améliorer le réseau de neurones un système de punition peux être mis en œuvre sur certaines « synapses » lorsque la machine perd la partie. En effet, à l'heure actuelle, on récompense seulement lorsque la machine gagne. Le fait de punir certains chemins reviendrait à obtenir plus de chance d'emprunter un chemin qui fera gagner la machine.

Conclusion :

Pour terminer on peut dire que ce projet nous a permis de se familiariser avec Python ainsi qu'avec le jeu de bâton qui est un jeu de duel qui demande logique et stratégie.