

Deep Checker

APPRENTISSAGE STATISTIQUE ET INTELLIGENCE
ARTIFICIELLE

Arthur Correnson, Igor Martayan, Manon Sourisseau

Projet de Statistiques, ENS, 2021

- Construction d'une **heuristique** évaluant la qualité des coups
- 3 problématiques :
 - Génération des données
 - Choix des modèles
 - Mise en place des modèles



Plan de la présentation

1. Génération de données et simulateur
2. Modèles et heuristiques
3. Régression aux k plus proches voisins
4. Amélioration de l'heuristique
5. Perceptron multicouche

Génération de données et simulateur

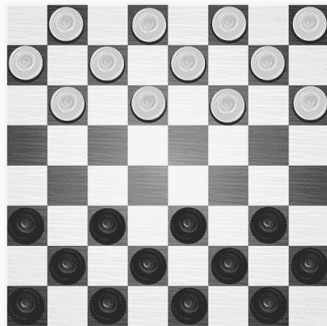
- Besoin d'un grand nombre de données
- Générer beaucoup de parties rapidement et de manière compacte en mémoire
- Écriture d'un simulateur dans le langage C

Création d'un simulateur

- 64 cases, 32 cases possibles
- 3 états possibles par cases :
Vide, pion blanc, pion noir

Représentation de l'état du plateau par
un entier de 64 bits (2×32 bits)

exemple : 1111 1111 1111 0000 0000
0000 0000 0000 // 0000 0000 0000
0000 0000 1111 1111 1111



Données et performances

- Les données sont stockées dans un fichier texte.
(perte d'efficacité contre simplicité de traitement des données)
- Performances très satisfaisantes :
10 000 parties générées en environ 1 secondes, sur un ordinateur ordinaire.

```
unsigned do_move_left(unsigned pos,  
    unsigned *player, int direction) {  
    unsigned pos_next = next_left(pos, direction);  
    *player ^= (1 << pos) | (1 << pos_next);  
    return pos_next;  
}
```

Modèles et heuristiques

On souhaite construire une heuristique qui attribue un **score** à un coup donné, selon la **qualité** du coup. Plusieurs approches pour déterminer l'heuristique :

- Régression par les K plus proches voisins (KNN)
- Réseau de neurones type perceptron multicouche (MLP)

Le but est de jouer le coup possible ayant le meilleur score.

Étant donné un ensemble \mathcal{DB} de parties simulées, on souhaite donner une première approximation de l'heuristique h .

- On introduit une fonction d'évaluation $\|\cdot\|_i : C_i \rightarrow [-1, 1]$ définie comme $\|c\|_i = \frac{1}{\sqrt{d(c)}} \cdot v(c)$
 - C_i l'ensemble des coups dans une partie $P_i \in \mathcal{DB}$
 - $d(c) \in \mathbb{N}$ le nombre de coups qui séparent c de la fin de partie
 - $v(c) = 1$ ou $v(c) = -1$ selon que la partie est gagnée ou perdue
- Le score final d'un coup c est la moyenne des scores qui lui sont attribués sur l'ensemble des parties dans \mathcal{DB}

Régression aux k plus proches voisins

KNN : méthode de régression aux **K plus proche voisins**. On définit la distance entre deux coups par la **distance d'édition** :

$$\langle c_1, c_2 \rangle_{KNN} = \|c_1 \oplus c_2\|_1$$

→ Les coups sont représentés comme des entiers de 128 bits.

- On calcule la distance du coup donné avec tous les autres coups.
- Le score attribué au coup donné correspond à **la moyenne des scores** des K plus proches voisins.

Résultat et performances de KNN

Victoires du joueur 1 (KNN)	Victoires du joueur 2 (Aléatoire)
18	32

Cette version de l'heuristique est peu satisfaisante :

- La notation d'un coup met l'accent sur les variations très **locales**
- La distance choisie rapproche uniquement les coups qui se ressemblent en terme d'état **global** du jeu

Amélioration de l'heuristique

- Nouvelle fonction d'évaluation :
 $\|\cdot\|_i : D_i \rightarrow \mathbb{N}$ définie comme : $\|d\|_i = p(d).v(d)$
 - D_i : Ensemble des états du damier vu par le joueur 1
 - $p(d) \in \mathbb{N}$: Nombre de pions mangé depuis l'état d
 - $v(d) = 1$ si le joueur 1 gagne, $v(d) = 0$ sinon
- On construit maintenant une fonction $w_i(c)$ telle que $w_i(d) = \|c\|_i$ si $d \in D_i$ et $w_i(d) = 0$ sinon
- Pour chaque état de damier d apparaissant dans l'ensemble des parties de \mathcal{DB} , $h(d) = \frac{1}{N} \sum w_i(d)$ avec N le nombre de parties P_i tels que $w_i(d) \neq 0$ (d est l'un des états pris par le damier dans P_i)

Victoires du joueur 1 (KNN)	Victoires du joueur 2 (Aléatoire)
34	16

Cette version de l'heuristique est plus satisfaisante.
Le temps de calcul reste toutefois assez élevé.

Perceptron multicouche

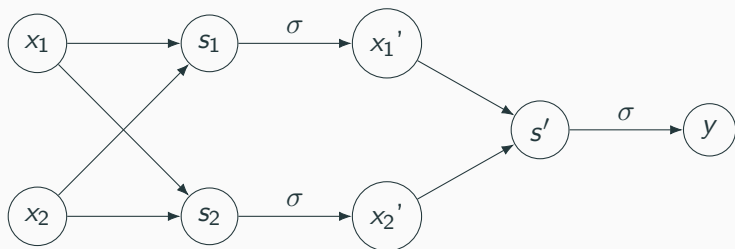


Figure 1: Perceptron avec une couche cachée

- propagation : $s = W.x + b$, $x' = \sigma(s)$
- descente de gradient : $W \leftarrow W - \alpha \frac{\partial \Delta}{\partial W}$

Perceptron multicouche (MLP)

- Entrées : vecteurs de 64 bits (représentant un unique damier)
- Coeur du réseau : 4 couches intermédiaires (64, 64, 32, 16)
- Noeuds du réseau : fonction d'activation **relu**
- Sortie du réseau de dimension 1 (régression) : combinaison linéaire des 16 sorties de la dernière couche puis d'une application de **relu**

Résultats du perceptron multicouche

Victoires du joueur 1 (MLP)	Victoires du joueur 2 (Aléatoire)
50	0

→ Heuristique bien plus satisfaisante, temps de calcul plus rapide

$$R^2 = 1 - \sum_{i=1}^n \frac{y_i - \hat{y}_i}{y_i - \bar{y}}$$

Heuristique	R^2
Distance à la victoire	0.46
Nombre de prises	0.68

Heuristiques + Models	Victoires	Défaites
Distance à la victoire + KNN	18	32
Nombre de prises + KNN	36	14
Distance à la victoire + MLP	39	11
Nombre de prises + MLP	50	0

Conclusion

Pistes d'amélioration :

- Heuristiques plus fines
- Techniques d'apprentissage par renforcement
- Réseaux de neurones convolutifs
- Arbre de Monte-Carlo

Questions ?

Références

1. Multi Layer Perceptron, https://en.wikipedia.org/wiki/Multilayer_perceptron, Wikipédia
2. K-nearest neighbors algorithm, https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm, Wikipédia
3. Scikit-learn: Machine Learning in Python
<https://scikit-learn.org/stable/> Pedregosa et al. ,
Journal of Machine Learning Research, 2012
4. Levenshtein distance, https://en.wikipedia.org/wiki/Levenshtein_distance, Wikipédia
5. Coefficient of determination, https://en.wikipedia.org/wiki/Coefficient_of_determination, Wikipédia