Rapport de TP IA-Jeux Numba

Réalisé par Hugo Maitre

Abstract

Nous présentons dans ce rapport les deux algorithmes utilisés pour les développements des IA ainsi que les résultats obtenus pour chaque IA développées. Enfin nous nous intéressons au développement du réseau de l' "IA Deep" ceci inclut la recherche des hyperparamètres, constitution du dataset... Toutes informations relatives à l'installation et prise en main du code source est donnée dans le fichier README.

Algorithme MCTS

Les précédentes démarches consistaient à simuler le même nombre de simulations pour chaque coup possible. La fonction MCTS programmé, se base désormais sur l'algorithme de Monte Carlo de recherche d'arbre. Pour choisir les prochains coups à simuler, on garde trace des scores UCB pour chaque coups possibles et on choisit le coup au plus gros score associé. La fonction présente ici 2 boucles: Chaque itération effectue une simulation et l'on boucle ensuite sur tous les coups possibles pour mettre à jour les scores UCB de chaque coups. Puisque les scores UCB se basent en partie sur les résultats de la fonction GetScore(), qui retournent un nombre positif pour le joueur 0 et un nombre négatif pour le joueur 1, selon qui est le joueur on récupère soit l'argmin pour le joueur 1 soit l'argmax pour le joueur 0.

Algorithm 1 Algorithme d'exploration MCTS

function MCTS($nb_{SimulationInitial}, Board, c, player$)
  $nb_{Simulations} \leftarrow nb_{PossibleMove} * nb_{SimulationInitial}$
  $UCB\_Score[] \leftarrow [0] * nb_{PossibleMove}$
  $scores[] \leftarrow [0] * nb_{PossibleMove}$
  $n\_try[] \leftarrow [0] * nb_{PossibleMove}$
  $means[] \leftarrow [0] * nb_{PossibleMove}$

  $simu_idx \leftarrow 1$
  while $simu_i < nb_{SimulationInitial}$ **do**
    $Bcopy \leftarrow copy(Board)$
    $best_{move\_idx} \leftarrow argmax(UCB\_Score[])$
    $best_{move\_UCB} \leftarrow Bcopy[best_{move\_idx}]$
    $scores[best_{move\_idx}] \stackrel{+}{=} GetScore(Bcopy)$
    $n\_try[best_{move\_idx}] \stackrel{+}{=} 1$
    $means[best_{move\_idx}] \leftarrow scores[best_{move\_idx}]/(n\_try[best_{move\_idx}] - 1)$

    for i *in* $0..nb_{PossibleMove}$ **do**
      $UCB\_Score[] \leftarrow means[i] + c * \sqrt{\frac{\log(simu)}{n\_try[i]}}$
    end for
    $simu_i \stackrel{+}{=} 1$
  end while

  if $player = 0$ **then**
    $id = argmax(UCB\_Score[])$
  end if

  if $player = 1$ **then**
    $id = argmin(UCB\_Score[])$
  end if
return id

Algorithme IA Deep

Constitution du jeu de données

Nous générons 10000 examples. Chaque example est composé d'un tableau de dimension (3,64) avec l'information sur le plateau, son négatif et le numéro du joueur ainsi qu'un tableau de dimension (1,64) composé de 0 sauf à l'indice du meilleur coup qui nous est calculé par la fonction MCTS. Le meilleur coup est calculé en effectuant *nb_simus* simulations à l'aide de la fonction MCTS qui va générer $nb\_simus * nb\_move\_possible$ simulations de jeu. Dans la suite on a choisis $nb\_simus = 100$. On voit bien dans l'algorithme 3 que l'on génère de manière équilibrée des coups gagnants pour chaque état d'avancement de la partie (début et fin de partie) puisque l'on inscrit pour chaque tour les coups gagnant dans notre jeu d'apprentissage. On génère aussi le même nombre d'exemple pour le player0 et le player 1.
Le jeu à été divisé en 3 sous jeux un jeu d'entraînement et de validation pour la partie entraînement du réseau et un jeu de test pour la partie évaluation. Le réseau est entraîné ainsi sur

60% du jeu total et testé à chaque Epoch (étape de validation) sur 20% du jeu total. Enfin on réserve 20% pour l'ensembe de test.

Algorithm 2 Génération du dataset

```
function ADD_TO_DATABASE(Board, idx_start, nb_simus, c)
    turn = 0
    while possibleMoveLeft > 0 do                ▷ B[−1] > 0
        idx ← idx_start + turn

        if idx_start >= size_dataset then
            return size_dataset
        end if

        if player0.isTurn() then
            best = MCTS(nb_simus, Board, c, player = 0)
            X[idx, 2][] ← [0] * 64
        end if

        if player1.isTurn() then
            best = MCTS(nb_simus, Board, c, player = 1)
            X[idx, 2][] ← [0] * 64
        end if
        idMove = Board[best]
        _, x, y ← DecodeIDmove(idMove)
        idx_move ← 8 * y + x
        Y[idx, idx_move] = 1
        Play(Board, idMove)
        X[idx, 0][] ← Board[64 : 128]
        X[idx, 1][] ← −(Board[64 : 128] − 1)
        turn += 1
    end while
    return turn
end function

function MAIN(Board, size_dataset)
    idx_start ← 0
    count ← 0
    X ← allocate([0] * (size_dataset, 3, 64))
    Y ← allocate([0] * (size_dataset, 64))

    while True do
        B_copy ← Board
        idx_start ← add_to_database(B_copy, idx_start, nb_simus, c)
        if idx_start >= size_dataset then
            break
        end if
    end while
return X, Y
```

Algorithme Général

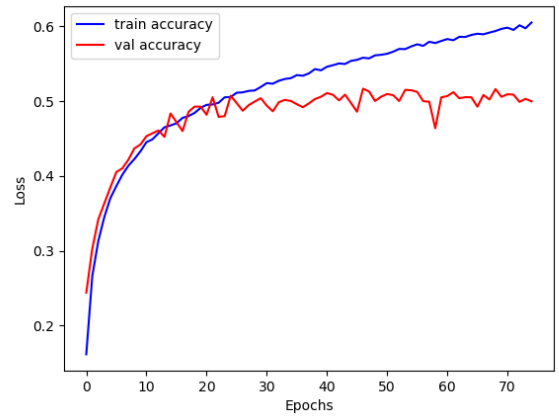
L'algorithme ci-dessous s'occupe de transformer les données en entrée de la manière attendue par le réseau . Les données sont ensuite passées en entrée au modèle entraîné pour produire la prédiction. A noter qu'ici le cas des différents joueurs est traité seulement en modifiant légèrement les entrées du réseau (on récupèrera toujours l'argmax sur les predictions). On calcule ensuite les coups possibles pour le joueur et pour chacun de ses coups l'on récupère la probabilité correspondante prédites par le réseau. Le réseau est entraîné sur un jeu de données donnant les bons coups à jouer, une probabilité élevée est donc associé à "un bon coup". On récupère donc la plus forte probabilité associé parmi tous celles possibles pour le joueur.

Algorithm 3 Tour IA Deep

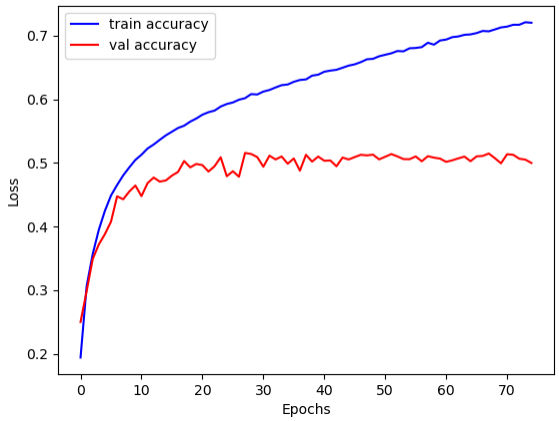
```
function DECODEIDMOVEDEEP(IDmove)
    x = IDmove%10
    y = int(IDmove/10)%10
return 8 * y + x
end function

function _POSSIBLEMOVESDEEP(nb_move, Board_move)
return map(Board_mv[0 : nb_mv], DecodeIDmoveDeep)
end function

function TOURIADEEP(Board, player)
    entry[0, :] ← Board[64 : 128]
    entry[1, :] ← −(Board[64 : 128] − 1)

    if player = 0 then
        entry[2, :] ← [0] * 64
    end if
    if player = 1 then
        entry[2, :] ← [1] * 64
    end if

    prediction[] ← model(entry[3][])
    possible_move_ids[] ← _PossibleMovesDeep(Board[−1], Board[0 : 64])

    assert(size(prediction[]) = Board[−1])
    prediction[] ← predictions[possible_move_ids[]]
    id ← argmax(prediction[])
    idMove ← Board[id]
    Play(B, idMove)
```

Recherche des Hyperparamètres

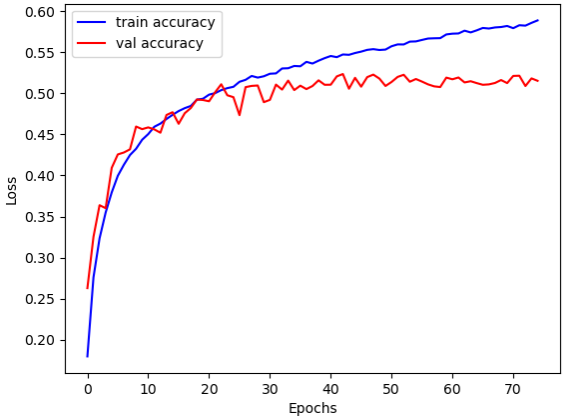
Nous avons tester 2 réseaux différents en faisant varier les paramètres généraux : N_Levels , dropout, N_res_blocks. Le premier réseau est composé de 2 niveaux (6 layers de convolution), il possède ainsi plus de paramètres mais est aussi plus facilement sujet au surapprentissage. On a ainsi pu observer que le dropout avait un impact posi-



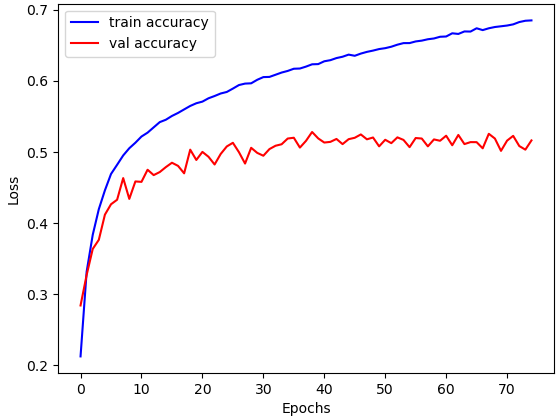
(a) Accuracy Réseau 1 avec dropout



(b) Accuracy Réseau 1 sans dropout



(c) Accuracy Réseau 2 avec dropout



(d) Accuracy Réseau 2 sans dropout

Figure 1: Val Train Accuracy Curve for Different Networks

tif sur ces performances avec un léger gain sur le score de validation. Le deuxième réseau est "moins profond" puisqu'il possède seulement un niveau, ses performances sont meilleures que le premier et le dropout cette fois ci n'est pas vraiment bénéfique ce qui s'explique certainement par la petit nombre de paramètre de ce dernier: le dropout apporte un véritable avantage sur des réseau un minimum conséquent.

$Accuracy = (TP + TN)/All$

On trouve en moyenne une accuracy autour de 50% ce qui est un score relativement bon étant donné le nombre elevé de classes (64) l'aléatoire étant à 1.5%.

IA N1p vs N2p sur 100 jeux				
Réseaux↓ *Paramètres* →	Res_Blocks	Levels	Dropout	Test Acc
Réseau1	3	2	0.15	0.49
Réseau2	3	2	0	0.47
Réseau3	3	1	0.15	0.51
Réseau4	3	1	0	0.52

Résultats entre IA

Ci dessous on peut voir les résultats exprimés en % des gains IA N1 vs IA N2. Il y'a une corrélation entre le nombre de simulations effectuées et la performances des IA sauf pour le nombre de 10K où l'on observe une baisse, ce qui laisse penser qu'il existerait un nombre optimal de simulations (situé entre 1000 et 10k).

IA N1p vs N2p sur 100 jeux					
N1↓ $N2 \rightarrow$	Random	10	100	1K	10K
Random	50	-	0	0	0
10	-	-	6	-	-
100	100	94	-	26	-
1K	100	-	74	-	88
10K	100	-	-	12	-

Ce second test nous permet de mieux cerner la valeur du coefficient c (exploration/exploitation) optimale. Les IA NP et l'IA MCTS avec N simulations sont programmées pour effectuer exactement le même nombre total de simulations c'est à dire $nb\_coups\_possibles * N$. La valeur optimale de c est donc environ égale à 0.4. Nous choisirons 0.3 pour la génération du dataset de IADeep.

IA 100p vs IAMCTS sur 100 jeux									
c	0.1	0.2	0.4	0.8	1.0	1.2	1.4	1.6	1.8
Gain IAM-CTS	54	72	74	50	38	38	22	14	10

IADeep est entraîné avec une valeur de c de 0.3 elle se trouve mis en défault là où IA MCTS est la plus performante c'est à dire aux valeurs optimales de c.

IADeep vs IAMCTS sur 100 jeux									
c	0.1	0.2	0.4	0.8	1.0	1.2	1.4	1.6	1.8
Gain IADeep	36	18	34	50	62	62	78	86	90