Tactibrême

Agents et Réseaux de Neurones

Auteurs: Gros Charlène, Pottier Loïc, Mudoy Jacques,
Horter Louise, Los Thomas, Gambier Clément

Date de rendu: March 18, 2025

Contents

1 Introduction

Ce document fournit une présentation détaillée des architectures de réseaux de neurones utilisées pour jouer au jeu de plateau 'Les tacticiens de Brême'. Il couvre la représentation des données d'entrée, le codage des sorties, le fonctionnement de l'apprentissage par renforcement profond (*Deep Q-Learning*) ainsi que les différentes couches du réseau. Nous étudierons d'abord le focntionnement du DQL avant de détailler les différents modèles utilisés.

2 Le DQL

Le Deep Q-Learning (DQN) est un méthode d'apprentissage par reforcement (Reinforcement Learning) visant à entraîner un agent à interagir avec un environnement pour maximiser une récompense cumulative. L'agent apprend à prendre des décisions en fonction de l'état actuel de l'environnement et des récompenses associées à chaque action. Le DQL est basé sur l'apprentissage profond et utilise un réseau de neurones pour approximer la fonction $Q(s, a)$, qui estime la récompense attendue pour une action donnée dans un état donné.

2.1 Utilisation de datasets

Dans la plupart des applications de l'apprentissage profond, l'entraînement repose sur des ensembles de données statistiques (*offline learning*). Par exemple, les réseaux de neurones convolutionnels (CNN) sont généralement entraînés sur des datasets d'images (*e.g., ImageNet*), tandis que les modèles NLP exploitent des corpus de texte préexistants. De manière similaire, certaines approches de RL exploitent des expériences passées sous forme de datasets pré-enregistrés pour optimiser la politique d'un agent, comme dans l'*Offline Reinforcement Learning*.

2.2 Apprentissage par récompense

Dans notre cas particulier, n'ayant pas accès a un dataset, nous adoptons une approche d'apprentissage plus directe. L'agent apprend à jouer en interagissant directement avec l'environnement, en explorant les différentes actions possibles et en ajustant ses stratégies en fonction des récompenses reçues. L'agent est entraîné à maximiser la récompense cumulative en utilisant une combinaison d'exploration aléatoire et d'exploitation des connaissances acquises.

3 Nécessité de deux modèles: jeu et draft de placement

Au début d'une partie, les deux joueurs doivent placer leurs pions sur le plateau. Ce placement initial est crucial pour la suite de la partie, car il détermine les

options stratégiques disponibles. Chaque joueur doit placer un pion à tour de role jusqu'à ce que les 8 pions soient placés. Pour cette phase de draft, nous utiliserons un modèle spécifique prenant en entrée une représentation partielle du plateau de jeu et produisant une distribution sur les cases disponibles. Ce modèle est entièrement distinct du modèle principal qui sera utilisé pour les mouvements pendant la partie.

3.1 Pourquoi un modèle séparé ?

L'apprentissage d'un modèle unique pour l'ensemble du jeu présente plusieurs inconvénients :

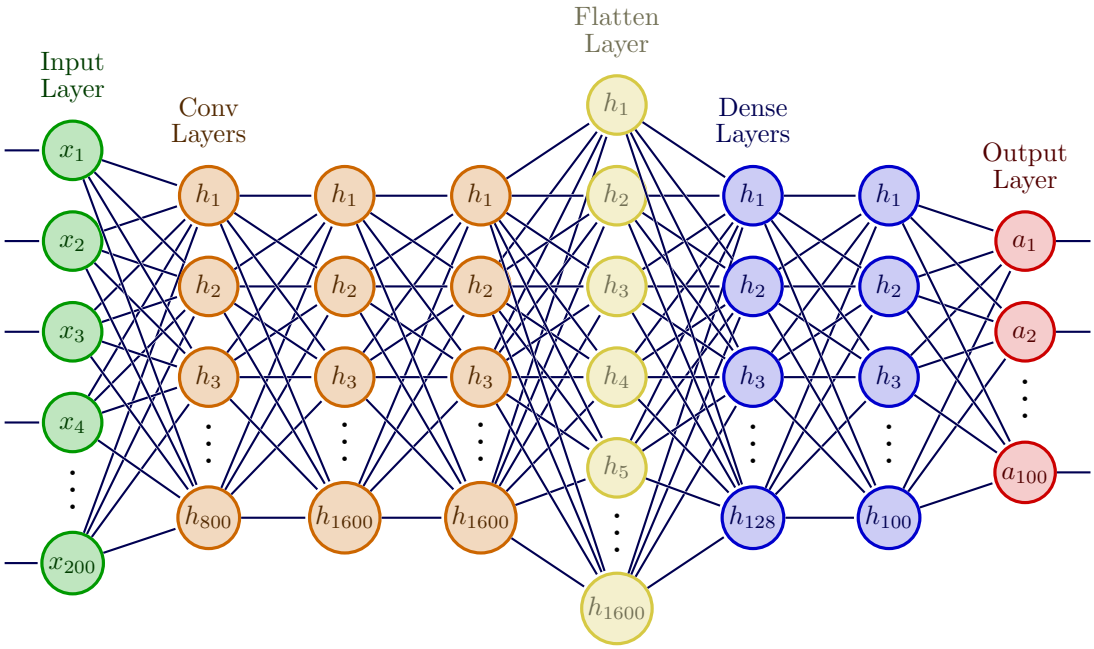
- **Différence d'objectifs** : Les objectifs de la phase de draft et de la phase de jeu sont différents. Le modèle de draft doit apprendre à placer les pions de manière stratégique par rapport aux placements adverses, tandis que le modèle de jeu doit apprendre à prendre des décisions tactiques pour gagner la partie.
- **Représentation différente** : La phase de draft nécessite une représentation différente du plateau, car les pions ne sont pas encore placés. Un modèle distinct permet de gérer cette différence de manière plus efficace. Il est inutile d'utiliser les mêmes dimensions d'entrées/sortie pour les deux modèles.

3.2 Approche d'apprentissage

- **Draft** : L'agent place les pions et est récompensé en fonction de la qualité du placement. La récompense est déterminée en fonction du déroulement de la partie.
- **Jeu** : L'agent prend des décisions pour gagner la partie, en apprenant progressivment à l'aide des récompenses données par l'environnement à chacun de ses mouvements.

4 Modèle 1

Ce premier modèle prend en entrée une représentation du plateau de jeu et produit une distribution sur les mouvements possibles. Le modèle n'ayant pas connaissance des mouvements interdits, on utilise un masque pour les éviter.



4.1 Représentation des Entrées

Le plateau est représenté comme une grille 5×5, où chaque case est encodée par un vecteur à 8 canaux décrivant la présence des différents pions.

- Les 4 premiers canaux : Présence des pions bleues (*âne_bleu*, *chien_bleu*, *chat_bleu*, *coq_bleu*).
- Les 4 canaux suivants : Présence des pions rouges (*âne_rouge*, *chien_rouge*, *chat_rouge*, *coq_rouge*).

Exemple

```
[0, 0, 1, 0, 1, 0, 0, 0] # chat bleu, âne rouge
[0, 0, 0, 0, 0, 0, 0, 0] # case vide
```

4.1.1 Dimensions des Entrées

- Pour un plateau unique : $(8, 5, 5)$ (canaux, lignes, colonnes).
- Pour un lot de plateaux (*batch*) : $(\text{taille\_batch}, 8, 5, 5)$.

4.2 Représentation des Sorties

La sortie du réseau encode tous les déplacements possibles sous forme de liste aplatie de paires $(\text{pion}, \text{destination})$.

4.2.1 Taille de la Sortie

La sortie compte $4 \times 25 = 100$ mouvements possibles :

- 4 pions possibles (par joueur).
- 25 cases possibles (une grille 5×5).

4.2.2 Encodage des Placements

Chaque index i dans la sortie correspond à :

$$\text{index\_pion} = \frac{i}{25}, \quad \text{ligne\_destination} = \frac{i \bmod 25}{5}, \quad \text{colonne\_destination} = i \bmod 5.$$

4.2.3 Masquage

Un masque binaire est appliqué aux 100 scores de sortie pour invalider les mouvements impossibles. Les mouvements invalides reçoivent un score de $-\infty$ avant l'étape *softmax* :

```
Masque = [0, 1, 0, ..., 1]  # 1 pour les mouvements valides, 0 sinon
```

4.3 Architecture du Réseau de Neurones

L'architecture prend la représentation du plateau en entrée et produit une distribution sur les 100 mouvements possibles. Le réseau utilise des couches convolutionnelles pour extraire les caractéristiques spatiales, suivies de couches entièrement connectées pour produire les scores finaux.

4.3.1 Entrée

Dimensions d'Entrée : $(\text{taille\_batch}, 8, 5, 5)$

4.3.2 Couches Convolutionnelles

- Conv2D$(8 \to 32) \to$ ReLU
- Conv2D$(32 \to 64) \to$ ReLU
- Conv2D$(64 \to 64) \to$ ReLU

4.3.3 Couches Entièrement Connectées

- Flatten
- Linear$(5 \times 5 \times 64 \to 128) \to$ ReLU
- Linear$(128 \to 100)$

4.3.4 Sortie

$$(\text{taille\_batch}, 100)$$

Ce vecteur est ensuite combiné avec le *masque* pour garantir que seuls les mouvements valides sont pris en compte.

4.4 Mécanisme de sélection des actions

Après avoir obtenu les scores pour chaque action possible, l'agent doit sélectionner un mouvement à exécuter. Plutôt que de choisir systématiquement l'action avec le score le plus élevé, nous utilisons une approche probabiliste qui favorise l'exploration tout en privilégiant les actions à fort potentiel.

4.4.1 Politique stochastique

Nous convertissons les scores du réseau en une distribution de probabilités à l'aide de la fonction softmax:

$$P(a_i) = \frac{e^{q_i}}{\sum_j e^{q_j}}$$

où q_i est le score attribué à l'action i par le réseau.

4.4.2 Échantillonnage multinomial

Plutôt que de sélectionner l'action avec la probabilité la plus élevée (exploitation pure), nous échantillonnons une action selon la distribution calculée:

```
action_index = torch.multinomial(probabilités, 1).item()
```

Cette approche offre plusieurs avantages:

- **Exploration guidée**: Les actions avec des scores élevés ont plus de chances d'être sélectionnées, mais les actions moins prometteuses peuvent également être explorées.
- **Diversité des stratégies**: L'agent ne joue pas toujours de manière déterministe, ce qui le rend moins prévisible et potentiellement plus robuste.
- **Prévention du sur-ajustement**: La variabilité dans les actions aide à éviter que l'agent se bloque dans des stratégies sous-optimales.

4.4.3 Masquage des actions invalides

Avant l'application de softmax, les actions invalides sont masquées en assignant une valeur très négative à leurs scores (-10^9):

```
q_values[mask == 0] = -1e9
```

Cela garantit que leur probabilité après softmax sera pratiquement nulle, les excluant effectivement de la sélection.

4.4.4 Équilibre exploration-exploitation

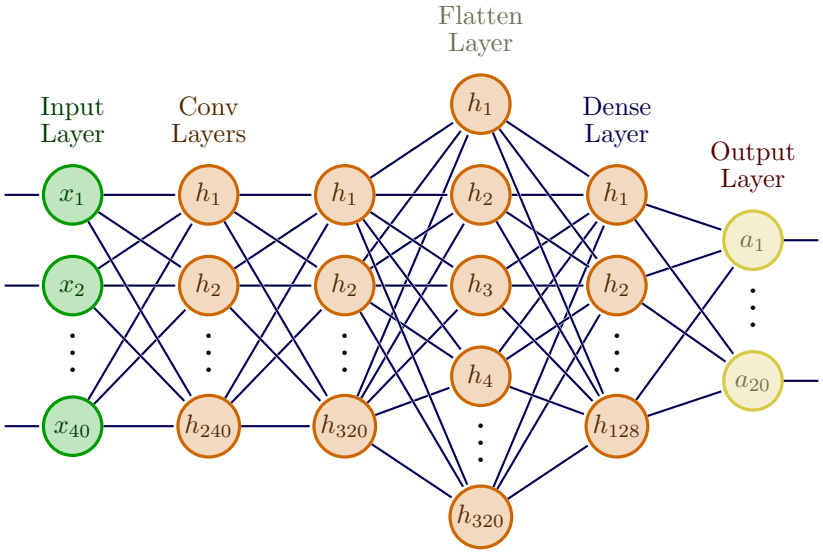
Pendant l'entraînement, nous utilisons une politique ϵ-greedy modifiée:

- Avec une probabilité ϵ, l'agent choisit une action complètement aléatoire parmi les actions valides.
- Avec une probabilité $1 - \epsilon$, l'agent utilise l'échantillonnage multinomial décrit ci-dessus.

Le paramètre ϵ diminue progressivement au cours de l'entraînement, réduisant l'exploration aléatoire au profit de l'exploitation des connaissances acquises.

5 Modèle 2: IA de Draft

Ce second modèle est spécialisé dans la phase initiale de placement des pièces (draft). Il s'agit d'une phase cruciale où les joueurs placent leurs pions à tour de rôle sur leurs lignes de départ respectives. Ce modèle possède une architecture distincte, adaptée à cette phase spécifique du jeu.



5.1 Représentation des Entrées

Dans la phase de draft, seules les lignes de départ sont pertinentes. Le plateau est donc représenté comme une grille 2×5, où chaque case est encodée par un vecteur à 4 canaux décrivant la présence des différents pions.

- Les 4 canaux représentent chaque type de pion (âne, chien, chat, coq).

5.1.1 Dimensions des Entrées

- Pour un plateau unique : $(4, 2, 5)$ (canaux, lignes, colonnes).
- Pour un lot de plateaux : $(\text{taille\_batch}, 4, 2, 5)$.

5.2 Représentation des Sorties

La sortie du réseau encode tous les placements possibles sous forme de liste aplatie de paires $(\text{pion}, \text{colonne})$.

5.2.1 Taille de la Sortie

La sortie compte $4 \times 5 = 20$ placements possibles :

- 4 types de pions.
- 5 positions possibles sur la ligne de départ.

5.2.2 Encodage des Placements

Chaque index i dans la sortie correspond à :

$$\text{type\_pion} = \lfloor \frac{i}{5} \rfloor, \quad \text{colonne} = i \bmod 5$$

5.3 Architecture du Réseau de Neurones

L'architecture prend la représentation du plateau de draft en entrée et produit une distribution sur les 20 placements possibles.

5.3.1 Entrée

Dimensions d'Entrée : $(\text{taille\_batch}, 4, 2, 5)$

5.3.2 Couches Convolutionnelles

- Conv2D$(4 \to 16) \to$ ReLU
- Conv2D$(16 \to 32) \to$ ReLU

5.3.3 Couches Entièrement Connectées

- Flatten
- Linear$(2 \times 5 \times 32 \to 128) \to$ ReLU
- Linear$(128 \to 20)$

5.3.4 Sortie

$$(\text{taille\_batch}, 20)$$

5.4 Mécanisme d'apprentissage spécifique au draft

L'apprentissage du modèle de draft présente des particularités importantes:

5.4.1 Signal de récompense différé

Contrairement au modèle principal où des récompenses peuvent être attribuées à chaque mouvement, le modèle de draft ne reçoit une récompense significative qu'à la fin de la partie:

- Récompense positive (+100) si le joueur gagne la partie.
- Récompense négative (−100) si le joueur perd la partie.
- Récompense faiblement négative (−5) pour les placements invalides.

5.4.2 Stockage des expériences de draft

Pour faciliter l'apprentissage malgré ce signal de récompense différé, nous maintenons un buffer spécifique pour les expériences de draft:

```
self.draft_buffer[color] = []  # Liste pour chaque couleur
# Pour chaque action de draft:
self.draft_buffer[color].append((state_tensor, move_idx, is_valid, next_state_tensor))
```

À la fin de la partie, ces expériences sont utilisées pour entraîner le modèle de draft avec la récompense appropriée.

5.5 Interaction entre les modèles

Les deux modèles (draft et jeu principal) interagissent de la manière suivante:

- Le modèle de draft détermine le placement initial des pièces.
- Le modèle principal prend le relais pour la phase de jeu proprement dite.
- Le résultat final influence l'apprentissage des deux modèles.

Cette séparation permet à chaque modèle de se spécialiser dans son domaine spécifique tout en contribuant à l'objectif global de victoire.

6 Interactions et Améliorations

6.1 Coordination entre les systèmes d'IA

Le succès global de l'agent dépend de la coordination efficace entre les deux systèmes d'IA:

- L'IA de draft doit apprendre à créer des positions favorables que l'IA de jeu pourra exploiter.

- L'IA de jeu doit être capable d'utiliser efficacement les pièces placées par l'IA de draft.

Cette coordination émerge naturellement à travers le processus d'apprentissage, où les récompenses finales guident les deux systèmes vers des stratégies complémentaires.

6.2 Améliorations potentielles

Plusieurs pistes d'amélioration pourraient être explorées:

6.2.1 Architecture des réseaux

- Utilisation d'attention pour mieux capturer les relations entre pièces.
- Réseaux résiduels pour faciliter l'entraînement des modèles profonds.
- Paramétrage dynamique des couches en fonction de la taille du plateau.

6.2.2 Mécanismes d'apprentissage

- Implémentation de méthodes d'apprentissage plus avancées (A3C, PPO).
- Experience replay prioritaire pour favoriser les expériences les plus instructives.
- Ajouter des récompenses plus variées que `victoire/défaite` pour l'agent principal.

6.2.3 Entraînement et évaluation

- Parallélisation de l'entraînement sur plusieurs instances.
- Tournois réguliers entre différentes versions des agents pour mesurer les progrès.
- Visualisation des représentations internes pour mieux comprendre les stratégies apprises.