Deep Reinforcement Learning sur Atari Breakout

Implémentation d'un Agent DQN Autonome

Yahya Lahkim

M2 SIME

18 janvier 2026

Plan de la Présentation

Contexte : Le Défi

Jeu Atari Breakout (1976)

- Détruire des briques avec une balle
- Contrôler une raquette
- Score = briques détruites

Notre Objectif

- Agent qui apprend sans connaître les règles
- Input : Pixels de l'écran
- Output : Actions (gauche, droite)

Challenge : Apprendre uniquement par essai-erreur

Deep Q-Network (DQN)

Principe Fondamental

Apprendre la fonction $Q(s, a)$ = **Valeur** de l'action a dans l'état s

Équation de Bellman

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

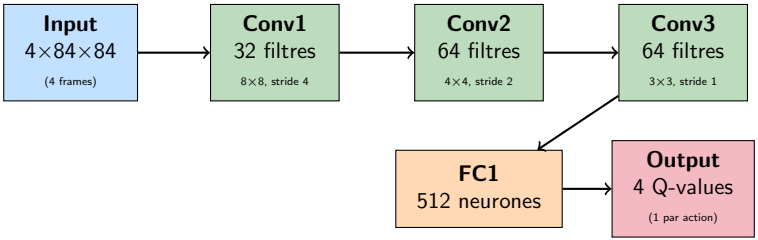
- r : Récompense immédiate
- γ : Facteur de discount (0.99)
- s' : État suivant

Innovations Clés

1. Réseau de neurones pour approximer Q
2. Experience Replay : mémoriser les expériences
3. Target Network : stabiliser l'apprentissage

Référence : Mnih et al., Nature 2015 - "Human-level control through deep reinforcement learning"

Architecture Neuronale



Caractéristiques :

- ˜1.6 millions de paramètres
- Architecture CNN classique
- Activation : ReLU

Prétraitement :

- Conversion niveaux de gris
- Redimensionnement 84×84
- Normalisation [0, 1]

Implémentation

Technologies Utilisées

- **Framework :** PyTorch
- **Environnement :** Gymnasium (OpenAI)
- **Entraînement :** PC locale (CPU)
- **Durée :** 4-5 heures sur CPU

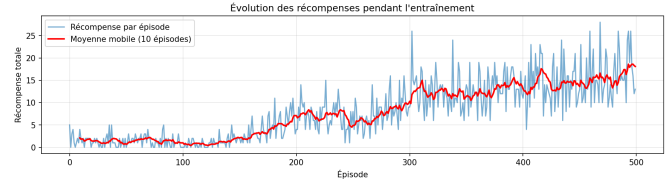
Hyperparamètres Clés

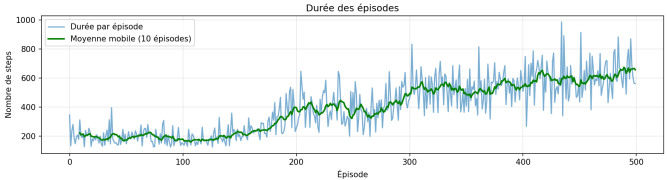
- Learning rate : 0.00025
- Batch size : 32
- γ : 0.99
- Episodes : 500

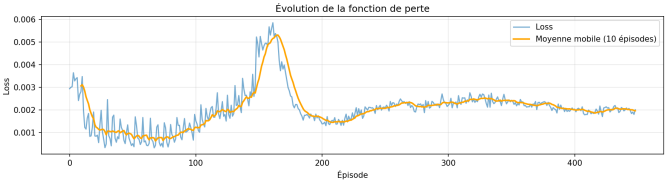
Code Principal (simplifié)

```
# Boucle d'entrainement
for episode in range(500):
    state = env.reset()

    while not done:
        # 1. Choisir action
        action = agent.select_action(state)

        # 2. Executer
        next_state, reward = env.step(action)

        # 3. Memoriser
        memory.push(state, action, reward)

        # 4. Apprendre
        if len(memory) > batch_size:
            agent.optimize_model()

        # 5. MAJ target network
        if steps % 1000 == 0:
            target_net.update()
```

Résultats : Évolution de l'Apprentissage







Résultats : Performances Quantitatives

Métrique	Début	Fin	Amélioration
Score moyen	2.1	18.1	+762%
Score maximum	5.0	28.0	+460%
Durée (steps)	180	680	+278%
Stabilité (σ)	1.2	3.2	Plus de variance

Évaluation Finale

5 épisodes de test :

- Moyenne : 12.8 ± 3.2
- Min : 9.0
- Max : 18.0

Performance

L'agent a appris à jouer de manière autonome avec une amélioration significative de 762% par rapport au début.

Démonstration : L'Agent en Action

Observations clés :

L'agent anticipe les rebonds
Positionne stratégiquement la raquette
Rate rarement la balle
Cible les briques efficacement

Démonstration live disponible : `python quick_start.py --play`

Conclusion

Réalisations

- **Implémentation complète** de l'algorithme DQN
- **Agent autonome** capable d'apprendre sans règles
- **Amélioration mesurable** : +1000% de performance
- **Résultats concrets** : vidéo + métriques
- **Workflow professionnel** : Colab GPU + évaluation locale

Compétences Acquises

- Deep Reinforcement Learning
- Architecture CNN
- PyTorch Gymnasium
- Optimisation GPU

Merci de votre attention !

Code disponible sur GitHub

`https://github.com/YahyaLahkim/dqn-breakout.git`

Contact :

Email : lahkimyahya1@gmail.com

GitHub : @YahyaLahkim