Cahier des Charges Technique : IA Snake From Scratch

Projet Personnel - Implémentation Deep Q-Learning (NumPy)

8 décembre 2025

Table des matières

1 Objectif du Projet

L'objectif est de développer un agent autonome capable de jouer au jeu Snake en apprenant par renforcement. **Contrainte technique majeure :** Aucune bibliothèque de Deep Learning (Keras, PyTorch, TensorFlow) n'est autorisée. Le réseau de neurones et l'algorithme de rétropropagation doivent être codés à la main en utilisant uniquement l'algèbre linéaire (bibliothèque `numpy`).

2 Module 1 : Environnement et Données

Pour que le modèle converge rapidement, nous simplifions la vision du jeu en un vecteur de caractéristiques relatives (au lieu de l'image brute de la grille).

2.1 Vecteur d'État en Entrée (X)

Le réseau prend en entrée un vecteur colonne $X \in \{0, 1\}^{11}$. Chaque composante est un booléen (0 ou 1).

1. **Danger Immédiat (3 neurones) :**
   - Danger Tout Droit (Mur ou Queue) ?
   - Danger à Droite ?
   - Danger à Gauche ?
2. **Direction Actuelle (4 neurones) :**
   - Le serpent va vers la Droite ?
   - Le serpent va vers la Haut ?
   - Le serpent va vers le Gauche ?
   - Le serpent va vers le Bas ?
3. **Position de la Pomme (4 neurones) :**
   - Pomme est à Droite (de la tête) ?
   - Pomme est à Haut ?
   - Pomme est en Gauche ?
   - Pomme est en Bas ?

2.2 Vecteur de Sortie (Q)

Le réseau retourne un vecteur $Q \in \mathbb{R}^3$ représentant les Q-values (espérance de gain) pour les 3 actions possibles :

$$Actions = [\text{Tout Droit}, \text{Tourner Droite}, \text{Tourner Gauche}]$$

2.3 Système de Récompense (R)

— $+$**10** : Mange une pomme.
— **-10** : Game Over (Mur ou Queue).
— **0** : Rien ne se passe (ou petite pénalité temporelle optionnelle).

3 Module 2 : Modèle Mathématique (Forward)

Nous utilisons un Perceptron Multicouche (MLP) à une couche cachée.

3.1 Dimensions et Variables

— $N_{in} = 11$ (Entrées)
— $N_h = 256$ (Neurones cachés)
— $N_{out} = 3$ (Sorties)

Les paramètres à apprendre sont :
— $W_1 \in \mathbb{R}^{N_h \times N_{in}}$ (Poids couche 1)
— $B_1 \in \mathbb{R}^{N_h \times 1}$ (Biais couche 1)
— $W_2 \in \mathbb{R}^{N_{out} \times N_h}$ (Poids couche 2)
— $B_2 \in \mathbb{R}^{N_{out} \times 1}$ (Biais couche 2)

3.2 Propagation Avant (Forward Pass)

Pour une entrée X, le calcul est le suivant :

1. **Couche Cachée :**

$$Z_1 = W_1 \cdot X + B_1$$
$$A_1 = \text{ReLU}(Z_1) = \max(0, Z_1)$$

(La fonction max est appliquée élément par élément).

2. **Couche de Sortie :**

$$Z_2 = W_2 \cdot A_1 + B_2$$
$$Q_{pred} = Z_2$$

(Pas d'activation finale car nous voulons régresser une valeur réelle, pas une probabilité).

4 Module 3 : Apprentissage (Backward)

L'objectif est de minimiser l'erreur entre la prédiction Q_{pred} et la cible Q_{target} (calculée via Bellman). Fonction de Coût (Loss) : MSE

$$L = \frac{1}{2}(Q_{pred} - Q_{target})^2$$

4.1 Calcul des Gradients (Règle de la Chaîne)

Nous devons calculer les dérivées partielles de L par rapport aux poids et biais.

1. Erreur en Sortie (δ_{out}) On calcule l'écart uniquement sur l'action qui a été jouée (notée a).

$$\delta_{out} = Q_{pred} - Q_{target}$$

Note d'implémentation : Si Q_{target} a été correctement construit (voir Module 4), les composantes correspondant aux actions non jouées sont égales dans Q_{pred} et Q_{target}, donc leur différence est 0.

2. Gradients Couche 2 (Sortie)

$$\frac{\partial L}{\partial W_2} = \delta_{out} \cdot A_1^T \quad \in \mathbb{R}^{3\times 256}$$

$$\frac{\partial L}{\partial B_2} = \delta_{out} \quad \in \mathbb{R}^{3\times 1}$$

3. Rétropropagation vers la couche cachée Il faut traverser W_2 et la fonction d'activation ReLU. La dérivée de la ReLU, notée $\sigma'(Z_1)$, vaut 1 si $Z_1 > 0$, et 0 sinon.

$$\delta_{hidden} = (W_2^T \cdot \delta_{out}) \odot \sigma'(Z_1) \quad \in \mathbb{R}^{256\times 1}$$

(Le symbole $\odot$ désigne le produit terme à terme / Hadamard).

4. Gradients Couche 1 (Cachée)

$$\frac{\partial L}{\partial W_1} = \delta_{hidden} \cdot X^T \quad \in \mathbb{R}^{256\times 11}$$

$$\frac{\partial L}{\partial B_1} = \delta_{hidden} \quad \in \mathbb{R}^{256\times 1}$$

4.2 Mise à jour des Paramètres (Optimizer)

Avec un taux d'apprentissage α (Learning Rate, ex : 0.001) :

$$W_1 \leftarrow W_1 - \alpha \cdot \frac{\partial L}{\partial W_1}$$

$$B_1 \leftarrow B_1 - \alpha \cdot \frac{\partial L}{\partial B_1}$$

(Et idem pour W_2, B_2).

5 Module 4 : Algorithme d'Entraînement Complet

Cette procédure est exécutée à chaque étape du jeu.

5.1 Étape A : Décision

1. Obtenir l'état actuel sous forme de vecteur : X_{old}.
2. Calculer $Q_{valeurs} = \text{Forward}(X_{old})$.
3. Choisir l'action a :
   — Hasard (Exploration) si $random() < \epsilon$.
   — argmax($Q_{valeurs}$) (Exploitation) sinon.

5.2 Étape B : Action

1. Jouer l'action a dans le moteur du jeu.
2. Observer :
   — La récompense r.
   — Le nouvel état transformé en vecteur : X_{new}.
   — Le booléen de fin de partie : *done*.

5.3 Étape C : Construction de la Cible (Bellman)

C'est ici que l'IA apprend à anticiper le futur.

1. On crée une copie du vecteur prédit : $Q_{target} = Q_{valeurs}$.
2. On calcule la "vraie valeur" attendue pour l'action a :

$$Val_{opti} = \begin{cases} r & \text{si } done = \text{True} \\ r + \gamma \cdot \max(\text{Forward}(X_{new})) & \text{si } done = \text{False} \end{cases}$$

   (γ est le facteur de discount, ex : 0.9).
3. On insère cette valeur dans le vecteur cible à l'indice de l'action jouée :

$$Q_{target}[a] = Val_{opti}$$

5.4 Étape D : Backpropagation

Appeler la fonction `backward` avec :

— Entrée : X_{old}
— Cible : Q_{target}

Le réseau met à jour W et B pour réduire l'écart entre sa prédiction et la cible.