Stratégie d’optimisation

🎯 Objectif de l’optimisation

Le but de l’optimisation est de **trouver les meilleurs hyperparamètres** pour une implémentation d’un algorithme d’apprentissage par renforcement, afin d’améliorer les performances de l’agent.

L’utilisateur peut choisir entre deux stratégies :

- **Optimisation par variations aléatoires**
- **Optimisation bayésienne (via `Hyperopt`)**

⚙️ Paramètres optimisables

Les paramètres considérés pour l’optimisation sont les suivants :

Paramètre	Description	Intervalle de variation (Bayesien)
`learning_rate_a`	Taux d’apprentissage (alpha)	`[0.01, 1.0]`

Paramètre	Description	Intervalle de variation (Bayesien)
`discount_factor_g`	Facteur de réduction des récompenses futures	`[0.5, 1.0]`
`epsilon`	Taux d'exploration initial	`[0.1, 1.0]`
`epsilon_decay_rate`	Taux de décroissance de `epsilon`	`[0.00001, 0.01]`

Ces paramètres sont initialement extraits via argparse (depuis la ligne de commande), puis ajustés par l'algorithme d'optimisation si demandé.

🔄 1. Optimisation par variations aléatoires

Implémentée dans la méthode `optimize_variations()`.

🧠 Principe

L'algorithme crée plusieurs **variations aléatoires autour des paramètres de base**, évalue chaque configuration, et conserve celle qui obtient le **meilleur score**.

📐 Détails

- Nombre de variations = `variations_number`
- Pour chaque variation :
  - Chaque hyperparamètre est modifié légèrement par un facteur aléatoire :
    - `±5%` pour les taux (`alpha`, `gamma`, `decay`)
    - `±10%` pour `epsilon`
  - La configuration est évaluée via la méthode `_evaluate_model()`
  - Si le score obtenu est meilleur que le meilleur précédent, on le conserve.

🔍 2. Optimisation bayésienne

La stratégie **d'optimisation bayésienne** utilisée ici repose sur le concept de **modéliser la fonction objectif** (celle que l'on cherche à maximiser ou minimiser) de manière probabiliste. L'idée centrale est que **chaque évaluation du modèle coûte cher** (par exemple, en temps d'entraînement), donc on veut **minimiser le nombre d'essais** tout en **trouvant de bons hyperparamètres**.

https://hyperopt.github.io/hyperopt/#algorithms

⚙️ Objectif

Trouver les meilleurs hyperparamètres (tels que `learning_rate`, `discount_factor`, etc.) pour maximiser une fonction objectif — ici, une combinaison pondérée de mesures de performance comme la récompense moyenne, la médiane, etc.

🧠 Principe de base

Contrairement à une recherche exhaustive ou aléatoire, l'optimisation bayésienne **ne choisit pas les paramètres au hasard**. Elle construit un **modèle probabiliste de la fonction objectif** (appelé "modèle de substitution") et utilise ce modèle pour **sélectionner intelligemment les prochains paramètres à tester**.

🔄 Étapes du processus

1. **Initialisation** :

   On commence par tester quelques combinaisons d'hyperparamètres **au hasard** pour avoir des données initiales.

2. **Modélisation de la fonction objectif** :

   On suppose que la fonction `f(hyperparams)` qu'on cherche à optimiser est inconnue mais qu'on peut **l'approximer** à l'aide d'un **modèle probabiliste**, souvent un **processus gaussien (GP)** ou un **modèle d'arbre de décision** comme dans l'algorithme TPE (Tree-structured Parzen Estimator, utilisé ici).

3. **Acquisition function** :

   Grâce au modèle, on détermine **où il est intéressant d'évaluer la fonction objectif ensuite**.

   On cherche un bon compromis entre :

   - **Exploitation** : tester des zones déjà connues comme prometteuses.

- **Exploration** : tester des zones encore peu explorées mais potentiellement meilleures.

L'**acquisition function** (par exemple `Expected Improvement`) estime donc **le gain attendu** si on évalue la fonction objectif dans une nouvelle zone de l'espace.

4. **Sélection et évaluation** :

   On choisit les hyperparamètres qui **maximisent la fonction d'acquisition**. On évalue ensuite la vraie fonction objectif avec ces paramètres (i.e. on entraîne et teste le modèle).

5. **Mise à jour du modèle** :

   Le modèle probabiliste est mis à jour avec ce nouveau point d'évaluation (paramètres + score obtenu).

6. **Répétition** :

   On recommence les étapes 3 → 5 pendant un certain nombre d'itérations ou jusqu'à un critère d'arrêt (temps, nombre d'essais, etc.).

🧪 Évaluation des modèles

Utilisée dans la méthode `_evaluate_model()`.

⚖️ Métriques calculées

- Moyenne des récompenses
- Médiane des récompenses
- Récompense max et min
- Score global calculé selon une **formule pondérée** :

```
score = avg_reward * 0.5 + median_reward * 0.3 + max_reward * 0.2 - (max_r
```

Cela favorise :

- Une bonne moyenne,
- Une bonne régularité (écart max-min faible),
- Des pics de performance (max élevé).

Les résultats sont affichés dans la console, et envoyés à une fonction de visualisation : `plotResults()`.

🧾 Règles d'utilisation côté ligne de commande

- `-optimize-with-bayesian` : active l'optimisation bayésienne
- `-optimize-with-variations` : active les variations aléatoires
- `-variations_number` : utilisé uniquement si l'une des deux optimisations est activée. Sinon, une erreur est levée.

✅ Résumé

Stratégie	Méthode	Caractéristique principale
Variations aléatoires	`optimize_variations()`	Teste des perturbations autour des paramètres de base
Optimisation bayésienne	`optimize_bayesian()`	Exploration intelligente via TPE

Les deux stratégies évaluent chaque configuration selon une formule de score personnalisée fondée sur les récompenses de l'agent.