Documentation Optuna

Utilité:

- C'est une bibliothèque Python dédiée à l'**optimisation automatique des hyper paramètres** d'un modèle de machine learning

- Au lieu de choisir manuellement des paramètres (pouvez vous ajouter les params à tester?) et de tester plusieurs configurations à la main, il nous aide à :
  - générer automatiquement des configurations,
  - entraîner le modèle,
  - évaluer les performances,
  - et utilise les résultats précédents pour proposer de meilleurs paramètres aux essais suivants.

-> **Obtenir de meilleures performances plus rapidement**.

Installation:

Optuna s'installe simplement via pip :

uv pip install optuna

Et pour vérifier:

import optuna
print(optuna.__version__)

Utilisation:

- Le cœur d'Optuna est la fonction "Objectif ". Elle:
  +) reçoit un objet trial,
  +) définit les hyperparamètres à tester,
  +) entraîne le modèle,
  +) retourne une métrique à optimiser.
  Exemple simple:
  def objective(trial):

```
        lr = trial.suggest_float("learning_rate", 1e-4, 1e-2, log=True)
        gamma = trial.suggest_float("gamma", 0.90, 0.99)
        score = train_and_evaluate(lr, gamma)
        return score
#trial: essai
# dans presque tous les cas on utilise scale log pour LR car il est efficace à rechercher
#  gamma ∈ [0.90 ; 0.99]
# train_and_evaluate: notre code
```

- Lancement de l'optimisation
  Exemple:

```
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=50)  #appel fonc obj 50 fois
```

avec -direction="maximize" : on maximise la métrique (ex : reward moyen)

-n_trials : nombre d'essais

Résultats et analyse:

Accès facilement par:

- *study.best_params*
- *study.best_value*

Avantages et limites

Avantage:

- Automatisation complète
- Gain de temps
- Meilleure performance des modèles
- Facile à intégrer dans un projet existant

Limites

- Coût de calcul (plusieurs entraînements)
- Nécessite une métrique bien définie

docu pour piloter

Lien vers la documentation

```
def run_once(dist, ajust):

    env, agents, names = create_race(dist, ajust)
    obs, _ = env.reset()
    done = False
    steps = 0

    max_distance = 0

    while not done and steps < 1500:
        actions = {}
        for i in range(MAX_TEAMS):
            actions[str(i)] = agents[i].choose_action(obs[str(i)])

        obs, _, terminated, truncated, info = env.step(actions)
        done = terminated or truncated
        steps += 1

        distance_i = info['infos']["0"]['distance']
        max_distance = max(max_distance, distance_i)

    env.close()
    return -max_distance
```

But de cette fonction : lancer une course avec un couple de parametres (dist et ajust) et retourner un score que Optuna va optimiser par la suite (-max_distance)

```
def objective(trial):

    dist = trial.suggest_float("dist", 0.2, 1.5)
    ajust = trial.suggest_float("ajust", 0.08, 0.9)

    score = run_once(dist, ajust)

    return score  # plus petit = meilleure position
```

le but des trial c'est de tester tout un tas de valeurs comprises dans un intervalle donné et de cibler de meilleures valeurs intelligemment dans un intervalle plus petit

```
dist = trial.suggest_float("dist", 0.2, 1.5)
ajust = trial.suggest_float("ajust", 0.08, 0.9)
```

ces deux lignes permettent de donner le nom des variables à tester et de donner l'intervalle de test, ensuite la valeur test décidée seront stockées dans dist et ajust.

```
score = run_once(dist, ajust)
```

ici on appelle la fonction qui crée la course et on teste les valeurs de dist et ajust qui ont été choisi la ligne au dessus. Dans score, on stocke alors la distance qui a été parcourue par le kart avec ces valeurs de variables.
On return -max_distance car on a choisi l'optimization "minimize" et optuna va donc essayer de faire en sorte de rendre le score le plus petit possible, or nous on veut une distance qui soit la plus grande possible, car nos kart ne finissent pas toujours la piste il faut donc une distance la plus grande, et pour qu'il minimise la valeur il faut faire -max_distance ce qui mathématiquement maximise max_distance.

(entre temps nos agents sont désormais capables de finir la piste, on peut donc chercher à minimiser le temps et non la distance, à implémenter)

study = optuna.create_study(direction="minimize")
c'est celle ligne qui indique qu'on cherche à minimiser, on pourrait mettre maximiser mais c'était rédiger de cette manière dans l'exemple sur lequel on s'est basé, et on a continué ainsi car ce n'était pas dérangeant.

study.optimize(objective, n_trials=3000)
ici on indique au logiciel qu'il doit faire 3000 essais de valeurs de paramètres avec l'utilisation de la fonction objective

Amélioration de la stratégie:
On a commencé par vouloir optimiser la distance parcourue par le kart, car cela impliquerait que s'il va plus loin c'est qu'il reste mieux sur la piste.
Notre objectif est maintenant différent: on souhaite faire le plus petit temps possible, pour pouvoir battre nos karts adverses.

Pour cela nous avons modifier la fonction objective et utiliser les steps comme base de temps : plus le nombre de steps dont le kart a besoin pour franchir la ligne est bas, plus son temps a été meilleur, on va donc minimiser le nombre de steps, on return alors steps au lieu de -max_distance.

Cependant, il reste des maps où notre agent se retrouve parfois coincé, auquel cas ce temps là ne doit être considéré comme le pire cas possible, car on arriverait assurément dernier en course contre les autres équipes si on se retrouve coincé. Ainsi on va forcer a return un

nombre très grand lorsque la course n'est pas terminée par notre agent:

```
if not terminated:   # l'agent n'est pas allé au bout
   return 4000      # grosse valeur de step
else:
   return steps
```

Cette partie est donc ajoutée dans la fonction run_once :

```
def run_once(dist, ajust):

   env, agents, names = create_race(dist, ajust)
   obs, _ = env.reset()
   done = False
   steps = 0

   while not done and steps < 1500:
      actions = {}
      for i in range(MAX_TEAMS):
         actions[str(i)] = agents[i].choose_action(obs[str(i)])

      obs, _, terminated, truncated, info = env.step(actions)
      done = terminated or truncated
      steps += 1

   env.close()

if not terminated:
        return 4000   # grosse pénalité
else:
        return steps  # temps réel de la course
```