MAP556 CMC2 : Angry Birds 2.0

Yunhao CHEN, Xinkai TANG

30 novembre 2021

1 Introduction

Dans ce projet, on présente une fonction de contrôle qui permet au mieux d'atteindre la cible en présence de vent aléatoire. On construit cette fonction en utilisant des simulations Monte Carlo du vent et des techniques d'optimisation et d'apprentissage sur ces trajectoires.

2 Simulation du vent

On a utilisé schéma d'Euler pour la simulation du vent. On présente ici un exemple de la simulation.

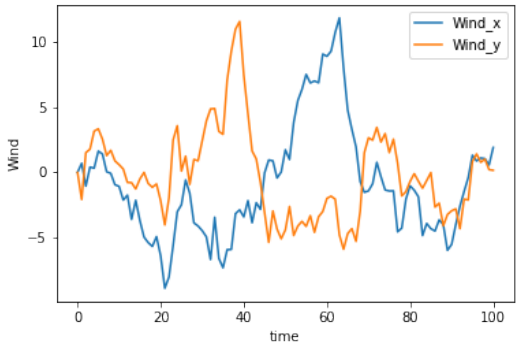

FIGURE 1 – La simulation du vent

3 Chemin d'idées pour résoudre le problème

3.1 L'apprentissage par renforcement

Dans un premier temps, on a pensé à l'apprentissage par renforcement. En simulant un environnement qui peut interagit avec l'agent, le dernier arrive à obtenir les connaissances de comment prendre la décision au travers de plein d'interactions. On modélise le contrôles à prendre comme action de l'agent. Une idée naturelle est de modéliser le temps, la position actuelle et la densité du vent comme observations de l'agent. Cependant, on se rend compte que les actions doivent être indépendantes du vent. En plus, la fonction perte fournie est régulière. Afin de l'optimiser, on peut emprunter l'idée de rétro-propagation du gradient en employant un réseau de neurones.

3.2 Réseau de neurones

Intuitivement, le réseau prend 3 valeurs comme entrée : temps actuel et position en axe x et y. Il a 2 valeurs comme sortie : contrôle à prendre en axe x et y. Cependant, ce n'est pas un problème standard de l'apprentissage supervisé. Il n'y a pas de perte naturelle pour chaque entrée et sortie. On a d'abord pensé à définir une fonction perte soi-même.

4 Concevoir les fonctions de coût

L'une des difficultés est comment définir la fonction de coût. On a proposé plusieurs fonctions.

4.1 Option 1

Comme le contrôle est adapté au temps $t = 0, 1, ..., 9$, donc on met à jour les paramètres chaque seconde. La fonction 1 proposée est de la forme :

$$f_{loss}(t, x_t, u_t) = |u_t|^2 + L(\hat{X}_{10})$$

avec $\hat{X}_{10}$ est la position terminale en supposant que le contrôle reste invariant dans les étapes suivantes. La fonction $L(x)$ est la même que l'énoncé.
Mais en fait, cette fonction perte n'est pas appropriée parce qu'elle ne balance pas les étapes en différents moments. A priori, l'étape en $t = 1$ aurait une perte beaucoup plus importante que l'étape en $t = 9$. A la fin, cette idée est abandonnée.

4.2 Option 2

En condition d'absence du vent et du résistance, la trajectoire est bien définie étant donné la vitesse initiale. On peut alors calculer une trajectoire "idéale" qui est sous forme de parabole standard, et qui donne la position idéale à chaque instant.
La fonction perte 2 vise à minimiser la distance entre la trajectoire réel est la trajectoire théorique. C'est-à-dire que l'on minimise la fonction :

$$f_{loss}(t, x_t, u_t) = |u_t|^2 + |x_t - x_t^*|^2$$

avec x_t^* est la position théorique. La position théorique x_t^* est calculée en supposant qu'il n'y a pas de vent, de contrôle et de résistance de l'air.
Cependant, on réalise que cette fonction perte a tendance de guider l'agent à prendre la trajectoire idéale pour arriver à destination. Mais l'agent peut très bien prendre une autre trajectoire pour arriver à destination. On trouve aussi que la somme de norme au carré des contrôles tout au long une trajectoire est énorme. Cela signifie que l'agent a fait trop d'effort afin de suivre la trajectoire idéale.
Du coup, même si le point d'arrivé entraîné par cette fonction perte est extrêmement proche du point de destination, on a décidé de choisir une autre forme de perte. On pense qu'il est possible d'arriver au point de destination tout en économisant les efforts de contrôles.

4.3 Option 3

On revient donc à la fonction perte proposée dans l'énoncé. La fonction perte 3 est de la forme :

$$f_{loss}(u) = \int_0^{10} |u_t|^2 \, \mathrm{dt} + L(X_{10})$$

qui est la même que la fonction de performance. Dans ce cas, on met a jour les paramètres quand chaque trajectoire est finie.

5 Implémentation

On construit un réseau en utilisant le package Tensorflow. Parmi les fonctions de coûts proposées, on a choisi la fonction 3 pour l'apprentissage. Le réseau contient trois couches cachées avec respectivement **24**, **12** et **6** neurones. On utilise fonction ReLu comme fonction d'activation pour ces trois couches. L'optimizer SGD a été choisi, avec learning rate 1e-7.
Après l'entraînement, on a sauvegardé la structure du réseau et les paramètres dans le fichier "savedModel". Dans le fichier "mon_controle.py", il suffit de recharger le modèle en appelant les fonctions dans le package Tensorflow.

Algorithm 1:

input : Les parametres de la dynamique, m, X_0, g, λ, V_0, nombre d'apprentissage N,etc.
output: La fonction de contrôle $u(t, x)$

Simuler le vent V_t;
for $k \leftarrow 0$ **to** N **do**
    for $i \leftarrow 0$ **to** *99* **do**
        if $i\%10 == 0$ **then**
            $u \leftarrow \mathbf{NN}(t, x_t)$
        Simuler X_t selon le vent V_t, le controle $u(t, x)$;
    Calculer la fonction de coût et les gradients;
    Faire les variables du réseau mis à jours selon les gradients.

5.1 Algorithme

On présente ici pseudo-code de notre algorithme.

5.2 Évaluer la performance

Pour évaluer la performance, on a simulé 1000 trajectoires avec chaque fois différents vents. On prend ensuite la moyenne des 1000 pertes comme la performance de ce modèle. Plus la perte est petite, plus la performance est bonne. Le modèle déposé a une performance environ **700**. On croit qu'il est possible d'améliorer ce résultat en "fine tuning" les hyperparamètres (par exemple, le type d'optimizer, learning rate, le type d'initializer de paramètres, la fonction d'activation et même la structure du réseau).