Reinforcement Learning & Advanced Deep

M2 DAC

2020-2021

TME 11 : Imitation Learning

Les approches de renforcement fonctionnent particulièrement bien lorsque l'environnement considéré donne accès à des récompenses denses qui permettent de guider efficacement l'agent dans sa recherche de politique. Cependant, pour de nombreuses applications ce genre de récompense n'est pas disponible (ou à un coût élevé). Pour ce genre de problème, si l'on dispose des traces d'un expert (agent automatique efficace ou humain), il est possible de procéder par imitation pour apprendre une politique qui vise à prendre des décisions similaires à celles observées dans les traces de l'expert. Diverses approches d'Imitation Learning existent :

- Behavioral Cloning : transposition du problème en un simple problème d'apprentissage supervisé, où l'on cherche à maximiser la vraisemblance des traces de l'expert selon notre politique
- Interactive Policy Learning : le problème du behavioral cloning est que l'on ne dispose d'information qu'au niveau des traces de l'expert et que l'on peut avoir du mal à extrapoler dans les zones trop éloignées de ces situations observées. L'Interactive Policy Learning regroupe toutes les approches, type Dagger, qui comblent les zones d'ombre atteintes par l'agent en questionnant l'expert. Ce genre de procédure est efficace mais requiert d'avoir un expert à disposition (peut être très coûteux)
- Apprenticeship Learning : forme d'Inverse Reinforcement Learning où l'objectif est d'apprendre une politique aussi efficace que les traces de l'expert selon une fonction de récompense à découvrir (on suppose que l'expert maximise cette fonction de récompense)

Dans ce TP nous allons mettre en oeuvre l'algorithme GAIL (Generative Adversarial Imitation Learning) [1], qui définit l'Imitation Learning comme un problème adverse où l'on cherche à générer des trajectoires indiscernables des trajectoires de l'expert selon un discriminateur appris simultanément.

Nous nous intéressons dans la suite au problème LunarLander (version discrète), dont des décisions expertes sont disponibles sur le site de l'UE (fichier expert.pkl). Le fichier fourni contient en fait une seule trajectoire de l'expert (trajectoire d'un agent PPO dont la récompense cumulée excède 230), sous la forme d'un tableau numpy etat-action sérialisé. Le tenseur contenu dans le fichier se charge de la façon suivante (avec self.longTensor et self.floatTensor respectivement un LongTensor et un FloatTensor chargé sur CPU ou GPU selon la ressource utilisée) :

```
import pickle
def loadExpertTransitions(self,file):
  with open(file, 'rb') as handle:
    expert_data = pickle.load(handle).to(self.floatTensor)
    expert_states=expert_data[:,:self.nbFeatures]
    expert_actions=expert_data[:,self.nbFeatures:]
    expert_states=expert_states.contiguous()
    expert_actions=expert_actions.contiguous()
```

Les quatre dernières lignes permettent de séparer les états et les actions associées dans deux tenseurs individuels, avec $self.nbFeatures$ la taille des observations de l'environnement LunarLander. Les actions sont représentées sous la forme de vecteurs one hot : 1 à l'index de l'action choisie, 0 partout ailleurs.

Selon les cas vous aurez besoin de la représentation one hot, ou bien de l'index de l'action choisie. Les méthodes suivantes permettent de passer d'une représentation à l'autre (avec self.nbActions le nombre d'actions de l'environnement) :

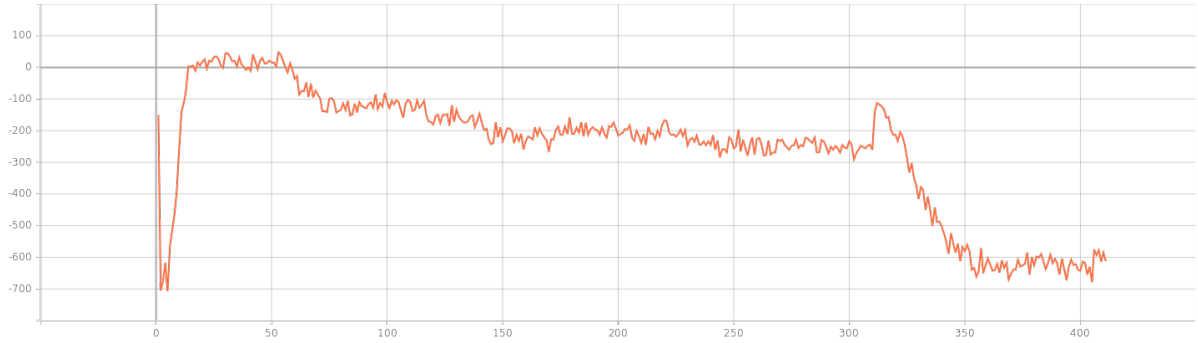
```
def toOneHot(actions):
    actions=actions.view(-1).to(self.longTensor)
    oneHot=torch.zeros(actions.size()[0],self.nbActions).to(self.floatTensor)
    oneHot[range(actions.size()[0]), actions] = 1
    return oneHot

def toIndexAction(oneHot):
    ac = self.longTensor.new(range(self.nbActions)).view(1, -1)
    ac = ac.expand(oneHot.size()[0], -1).contiguous().view(-1)
    actions=ac[oneHot.view(-1)>0].view(-1)
    return actions
```

1 Behavioral Cloning

Il s'agit de maximiser la probabilité de choisir, pour chaque exemple i de l'ensemble expert $\mathcal{E}$, l'action a_i associée à l'état s_i selon la politique de l'agent π_θ :

$$\max_\theta \sum_{(a_i,s_i)\in\mathcal{E}} \log \pi_\theta(a_i|s_i)$$

Avec une politique π_θ représentée par un réseau de neurones à deux couches cachées de 64 et 32 neurones chacune et fonction d'activation *tanh*, on obtient la courbe de résultats suivants en appliquant une montée de gradient à notre problème de maximisation de vraisemblance (optimisation sur tout le batch et affichage des résultats moyens calculés sur 100 épisodes tous les 100 pas de gradient, pas d'apprentissage = 0.003) :



On note que le maximum est aux alentours d'une moyenne de récompense de 50 points par épisode, atteint après environ 5000 pas d'apprentissage. Ensuite, l'algorithme se met à sur-apprendre les transitions de l'expert et n'est plus capable de généraliser sur les états proches (d'où la baisse de performances). Un maximum de 50 points paraît assez éloigné des 230 de la trajectoire de l'expert. Cette performance relativement peu satisfaisante s'explique par la difficulté de fournir des décisions de bonne qualité dans les zones quelque peu éloignées des états observés en apprentissage.

2 GAIL

L'objectif de GAIL est d'interagir avec l'environnement pour apprendre à s'orienter vers des zones connues de l'espace, sur lesquelles on sait imiter l'expert. L'algorithme est composé de :

— Un discriminateur D_ω qui est appris à distinguer les couples état-action issus de l'expert de ceux générés par la politique, via régression logistique (ligne 4 de l'algorithme ci-dessous). Il s'agit d'un réseau de neurones prenant en entrée un vecteur résultant de la concaténation d'un état s avec

l'action correspondante a en représentation one-hot, dont la sortie correspond à une probabilité issue d'une fonction d'activation finale sigmoïde. Le coût donné dans l'algorithme possède deux termes : celui de gauche vise à réduire la probabilité fournie pour les couples issus des trajectoires échantillonnées selon la politique courante (échantillonnées à la ligne 3 de l'algorithme), celui de droite vise à accroître la probabilité produite pour les couples état-action issus des trajectoires de l'expert. Il est conseillé de travailler par mini-batchs en considérant autant de couples de l'expert que de couples générés à chaque étape de l'optimisation.

— Une politique π_θ que l'on apprend à générer des couples état-action indistinguables de couples issus de l'expert selon le discriminateur. Dans l'algorithme original de GAIL donné ci-dessous, l'approche d'optimisation RL utilisée à chaque étape est TRPO. Cependant, du fait de la complexité algorithmique de TRPO, on considère dans ce TP l'utilisation de PPO développé dans un TP précédent. Il s'agit à chaque étape de réaliser plusieurs pas de gradient via PPO, avec pour objectif de maximiser la probabilité des trajectoires de récompense moyenne élevée (selon la log-probabilité fournie par le discriminateur pour chaque couple de la trajectoire). Pour ne pas converger trop rapidement sur des politiques déterministes, un coût d'entropie est ajouté au coût à minimiser à chaque itération de PPO.

Algorithm 1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \tag{17}$$

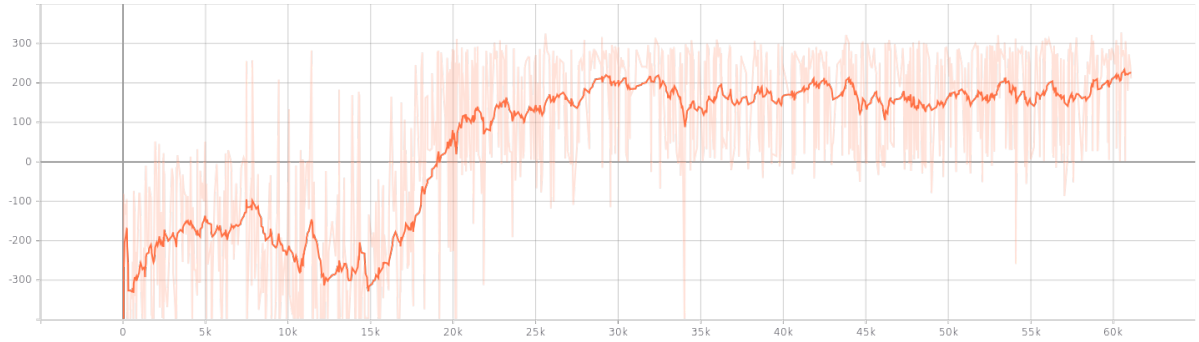
5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\begin{aligned} &\hat{\mathbb{E}}_{\tau_i}\left[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\right] - \lambda \nabla_\theta H(\pi_\theta), \\ &\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \,|\, s_0 = \bar{s}, a_0 = \bar{a}] \end{aligned} \tag{18}$$

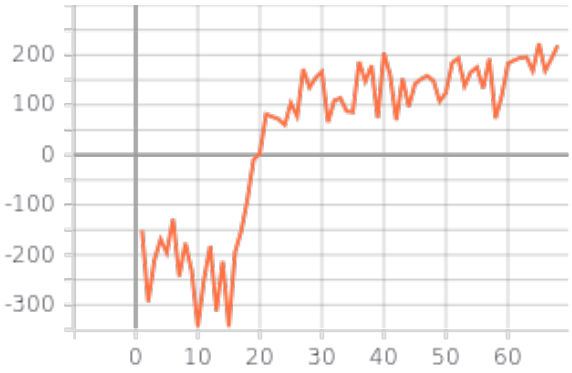
6: **end for**

Une particularité de cet algorithme est que l'on considère un retour moyen plutôt que le retour cumulé discounté classique. Comme en RL discounté classique, il est possible de se concentrer sur les récompenses futures uniquement à chaque étape de la trajectoire pour réduire la variance (causalité). La récompense R_t considérée par PPO à chaque étape t d'une trajectoire (notée $Q(s_t, a_t)$ dans l'algorithme ci-dessus) est donc donnée par : $R_t = \frac{1}{T-t}\sum_{t'=t}^{T} r_{t'} = \frac{1}{T-t}\sum_{t'=t}^{T} \log D_\omega(s_{t'}, a_{t'})$. On pourra également classiquement considérer une baseline donnée par un réseau critique V_ϕ pour chaque état s rencontré dans les trajectoires générées (par exemple appris selon moindres carrés avec cible $y_t = \frac{1}{T-t}\sum_{t'=t}^{T} r_{t'}$, d'autres possibilités d'acteur-critique pour retours moyens sont donnés dans [2]).

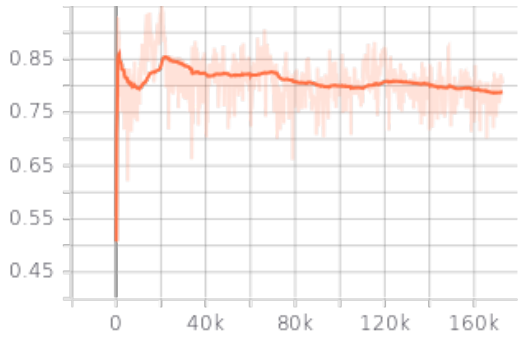
Avec une optimisation du discriminateur, de la politique et de la critique tous les 1000 évènements selon un pas d'apprentissage de 3E-4 (sûrement possible d'aller plus vite), un poids de l'entropie de 1E-3, 10 étapes d'optimisation du discriminateur et de PPO à chaque itération, un clip PPO à $\epsilon = 0.2$, des bruits $\eta \sim \mathcal{N}(0, 10^{-2})$ ajoutés à toutes les dimensions des couples (s, a) lors de l'optimisation du discriminateur et surtout un clip des récompenses $r_t \in [-100; 0]$ pour stabiliser l'optimisation, on obtient la courbe d'apprentissage suivante (récompenses de LunarLander en ordonnée) :

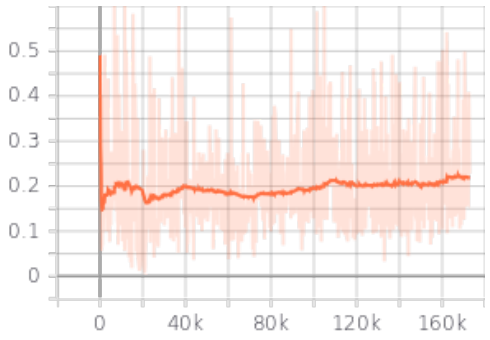


Si on trace les moyennes de récompenses de LunarLander calculées sur 100 trajectoires tous les 1000 épisodes d'apprentissage, on obtient :



On obtient donc une moyenne en test supérieure à 200, ce qui est bien supérieur à ce que l'on obtenait via Behavioral Cloning. On peut aussi s'intéresser aux scores donnés par le discriminateur pour les couples (s, a) issus de l'expert (à gauche) et ceux pour les couples de la politique (à droite) :





On note que ces courbes ne convergent pas vers 1 pour les couples de l'expert et 0 vers ceux de la politique, comme c'est le cas lorsque l'on ne stabilise pas bien les choses. Il faut trouver un point d'équilibre pour lequel les couples de la politique sont suffisamment proches de ceux de l'expert pour que le discriminateur ait des difficultés à discerner un certain nombre d'entre eux, comme c'est le cas ici.

Notons que, plutôt qu'un retour moyen, il aurait pu être possible de considérer un retour cumulé discounté comme exploré par exemple dans une version étendue de GAIL dans [3]. Cependant, cela se heurte à diverses difficultés :

— Avec une récompense $r_t = \log D(s_t, a_t)$, l'algorithme n'obtient que des retours négatifs. Si on considère une somme cumulée de ces différents retours, la politique tend à raccourcir les trajectoires générées (soit couper les moteurs au plus vite dans notre cas) pour collecter un minimum de récompenses ;

— Un schéma alternatif est de prendre $r_t = -\log(1 - D(s_t, a_t))$, qui fonctionne mieux dans notre cas en attribuant uniquement des récompenses positives. Au lieu de pénaliser moins fortement lorsqu'on ressemble à l'expert, on récompense positivement lorsqu'on ne paraît pas trop artificiel.

Néanmoins, cela conduit souvent au problème inverse : la politique tend à faire voler la fusée indéfiniment pour collecter un maximum de récompenses.

Le travail dans [3] propose de considérer une version hybride $r_t = \log D(s_t, a_t) - \log(1 - D(s_t, a_t))$ pour répondre à ces problèmes, mais nos expérimentations nous ont donné de bien meilleurs résultats en utilisant le retour moyen proposé dans le papier original de GAIL.

Références

[1] J. Ho and S. Ermon, "Generative adversarial imitation learning," *CoRR*, vol. abs/1606.03476, 2016. [Online]. Available : http://arxiv.org/abs/1606.03476

[2] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning : Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[3] I. Kostrikov, K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson, "Discriminator-actor-critic : Addressing sample inefficiency and reward bias in adversarial imitation learning," *arXiv preprint arXiv :1809.02925*, 2018.