Apprentissage par Renforcement

M2 DAC

TME 3. Q-Learning

Ce TME a pour objectif d'expérimenter les approches de renforcement value-based vues en cours.

1 Q-Learning

Implémenter l'algorithme Q-Learning (version tabulaire) et l'appliquer au problème du GridWorld considéré la semaine dernière. Par contre attention, la commande getMDP est maintenant interdite: on se met dans le cas de figure où les dynamiques de l'environnement sont inconnues. Il faut explorer pour apprendre les trajectoires efficaces.

2 DQN

Implémenter en pyTorch l'algorithme DQN avec Experience replay et Target network. Pour ce faire, vous aurez peut-être besoin de mettre à jour pytorch par la commande:

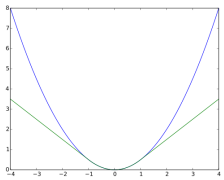
```
pip3 install torch torchvision --proxy=proxy:3128 --user -U
```

Pour la fonction Q, vous pouvez utiliser le code suivant:

```
class NN(nn.Module):
    def __init__(self, inSize, outSize, layers=[]):
        super(NN, self).__init__()
        self.layers = nn.ModuleList([])
        for x in layers:
            self.layers.append(nn.Linear(inSize, x))
            inSize = x
        self.layers.append(nn.Linear(inSize, outSize))
    def forward(self, x):
        x = self.layers[0](x)
        for i in range(1, len(self.layers)):
            x = torch.nn.functional.leaky_relu(x)
```

```
            x = self.layers[i](x)
        return x
```

Pour l'apprentissage de la fonction Q, il est préférable d'utiliser un coût Huber (torch.nn.SmoothL1Loss) plutôt qu'un coût moindre carré classique, afin d'éviter l'explosion des gradients:



$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

Testez votre code sur les trois jeux de randomAgentVariousGames.py:

- option –jeu 0: Gridworld;
- option –jeu 1: CartPole;
- option –jeu 2: LunarLander

Pour les jeux Cartpole et LunarLander, lancer les commandes suivantes pour installer les librairies requises:

```
pip3 install Box2D --proxy proxy:3128 --user
pip3 install Box2D-kengz --proxy proxy:3128 --user
```

Cartpole est un jeu où l'on cherche à stabiliser une barre verticale sur un chariot roulant, en fonction de ses mouvements. Deux actions possibles à chaque itération: gauche ou droite. Le jeu est perdu si la barre verticale s'incline de plus de 15 % (elle tombe alors inévitablement) ou si le chariot sort de l'écran. Le reward cumulé correspond au nombre de pas de temps sans terminaison du jeu (maximum 500 pas de temps). Les observations sont des vecteurs de 4 réels donnant la position du chariot, sa vitesse, l'inclinaison de la barre et sa vitesse de rotation. C'est a priori le jeu le plus simple. Il est donc conseillé de commencer par ce jeu là.

LunarLander est un jeu où l'objectif est de faire alunir une fusée, qui a une vitesse et une direction initiales aléatoires. Le reward obtenu dépend de la vitesse à laquelle la fusée s'est posée et de la distance de la cible (ainsi que du temps mis à alunir, du fait que les deux pieds touchent le sol, du fuel dépensé, etc.). 4 actions possibles par pas de temps: ne rien faire, allumer le moteur de gauche, le moteur central ou le moteur de droite. Les observations sont des vecteurs de 8 réels donnant diverses informations sur la position, la direction et la vitesse de la fusée.

Pour GridWorld qui est plus complexe, il est conseillé de simplifier les choses en cherchant une représentation plus simple des observations. Vous pourrez utiliser le transformateur suivant:

```
class FeaturesExtractor(object):
    def __init__(self,outSize):
        super().__init__()
        self.outSize=outSize*3
    def getFeatures(self, obs):
        state=np.zeros((3,np.shape(obs)[0],np.shape(obs)[1]))
        state[0]=np.where(obs == 2,1,state[0])
        state[1] = np.where(obs == 4, 1, state[1])
        state[2] = np.where(obs == 6, 1, state[2])
        return state.reshape(1,-1)
```

La fonction getFeatures prend en argument la carte du jeu originale et retourne un vecteur de features contenant la position de l'agent, celles des éléments jaunes ainsi que celles des éléments roses (les éléments rouges et verts étant fixes, pas besoin de les représenter dans le vecteur de caractéristiques).