TP apprentissage par renforcement

Yann Chevaleyre

Installations locales les packages python necessaires

In [ ]:
```
!pip install --user pyglet==1.2.4
```

In [ ]:
```
!pip install gym
```

Test de l'installation

Exécutez (et comprenez) le code ci-dessous

In [ ]:
```
import numpy as np
```

In [ ]:
```
import gym
```

In [ ]:
```
env = gym.make('CartPole-v0')
```

In [ ]:
```
for i_episode in range(3):
    observation = env.reset()
    reward = 0
    for t in range(20):
        # affichage graphique de l'environnement
        env.render()
        # tirage d'une action au hasard entre {0,1}
        action = np.random.randint(2)
        # cet action est faite, et on recupere
        # le prochain etat et la recompense
        observation, reward, done, info = env.step(action)

        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
```

Cet environnement `Cart-Pole` consiste à déplacer un chariot pour faire tenir en équilibre une poutre. Plus précisément:

- Il y a deux actions : gauche et droite (représentées par 0 et 1)
- L'observation reçue (c'est à dire l'état) est un tableau numpy comprenant 4 variables: la position du chariot, la velocite, l'angle a la verticale et la position du haut de la poutre
- L'épisode se termine lorsque l'angle de la poutre à la verticale dépasse 12 degré
- Les récompenses recues sont égales à 1 sauf si l'angle dépasse 12 degrés.

Pour afficher plus d'informations sur cet environnement, tapez `env.env?`

Exercice : dans le code précédent, l'action est choisie au hasard. Modifiez ce code pour que l'action choisie soit "va a droite" si l'angle de la poutre est inferieur a 2 degres, et "va a gauche" sinon.

Première partie: Implémentation du Q-Learning Tabulaire

dans cette partie, vous devrez discrétiser l'espace d'état, et implémenter l'algorithme du Q-learning sur le tableau Q(s,a)

Pour commencer, voici un rappel de l'algorithme du Q-Learning:

1. Répéter durant M épisodes:
   (a) Pour $t = 1 \ldots T$:
      i. Faire l'exploration ϵ-greedy de la façon suivante:
         A. avec une probabilité ϵ, on choisit une action a aléatoirement
         B. avec une probabilité $1-\epsilon$, on choisit $a = \arg\max Q(s,a)$
      ii. Exécuter l'action a, et reçoit la récompense et l'état suivant r, s'
      iii. Mettre à jour de la fonction Q:
$$Q(s,a) := Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \right]$$

On va donc devoir discrétiser l'espace d'états. Pour cela, on cree une fonction `discretise(x)`, puis une fonction `observation_vers_etat` qui renvoie un état (nombre entre 0 et N-1) en fonction de l'observation. Lisez le code ci-dessous.

In [ ]:
```
# nval est le pas de discretisation par variable
nval = 5         # nombre de valeur discretes qu'une variable peut prendre
N    = nval ** 4 # taille de l'espace d'etat

def discretise(x,mini,maxi):
    # discretise x
    # renvoie un entier entre 0 et nval-1
    if x < mini:  x = mini
    if x > maxi:  x = maxi
    return int(np.floor((x-mini)*nval/(maxi-mini+0.0001)))

def observation_vers_etat(observation):
    pos   = discretise(observation[0],mini=-1,maxi=1)
    vel   = discretise(observation[1],mini=-1,maxi=1)
    angle = discretise(observation[2],mini=-1,maxi=1)
    pos2  = discretise(observation[3],mini=-1,maxi=1)
    return pos + vel*nval + angle*nval*nval + pos2*nval*nval*nval
```

Maintenant, on peut donc récupérer à partir d'une observation le numéro de l'état associé:

In [ ]:
```
observation = env.reset()
s = observation_vers_etat(observation)
print("le numero de l'etat de depart est le :",s)
```

Exercice :

- Creez un tableau numpy `Q` de dimension `N,2` initialisé aléatoirement. Ce tableau sera utilise dans l'algorithme du q-learning
- Implémentez l'algorithme du Q-Learning. Ensuite, ajustez les paramètres de l'algorithme pour qu'il converge correctement

Seconde partie: Implémentation du Q-Learning avec approximation de fonction

dans cette partie, vous devrez d'abord créer un représentation de chaque état, avant d'implémenter le deep-q-learning

Exercice:

- Creez une fonction `K(x)` qui renvoie `1/(1+x**2)`
- Créez un fonction qui transforme l'observation o en une représentation $\psi(o)$, en utilisant la fonction `K`, comme vu en cours
- Implémentez l'algorithme du Q-learning avec approximation de fonction (deep q-learning)
- **Bonus**: Implémentez l' *Experience-replay*, et comparez-le a l'algorithme de base (l'algorithme est donné ci-dessous)

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for