Apprentissage par renforcement, La ruée vers l'or

December 5, 2024

Contexte et Objectifs

Le TP s'organise autour du développement d'un agent agissant comme un bandit multi-bras dans le cadre de l'apprentissage par renforcement. L'agent sera confronté à une situation où il doit choisir parmi plusieurs actions possibles, représentées comme des bras de bandit, sans connaître initialement les récompenses associées à chaque bras. Les algorithmes de bandit, notamment Tirage Uniforme, Epsilon-Greedy, Softmax et UCB, seront explorés et étudiés pour leur capacité à équilibrer l'exploration et l'exploitation dans la sélection des actions. Ce projet vise à comprendre comment ces différents algorithmes peuvent influencer les performances de l'agent en termes d'accumulation de récompenses sur un nombre défini d'essais, ainsi qu'à analyser leur efficacité dans des contextes d'apprentissage par renforcement.

Description

L'exercice prend place dans un environnement de jeu destiné à être implémenté en Python pour illustrer les concepts d'apprentissage par renforcement et plus précisément du bandit multi-bras. Le jeu se déroule sur un terrain bidimensionnel de taille (X, Y), où chaque position possède une récompense stochastique associée. L'objectif principal pour l'agent est d'accumuler autant de récompenses que possible en sélectionnant de manière itérative des positions (x, y) sur ce terrain.

Le TP consistera à implémenter le jeu de l'exercice en utilisant le language Python, en mettant en œuvre les mécanismes d'apprentissage par renforcement pour permettre à l'agent d'apprendre et de maximiser ses récompenses accumulées. Le TP comprendra également la création d'un visualiseur pour suivre les performances de l'agent au fil du temps, ainsi que la représentation graphique de la connaissance actuelle du terrain par l'agent.

Le TPs s'articule en trois étapes:

- Le développement de la classe Terrain où chaque cellule de la grille 2D correspond à une distribution Gaussienne de gain prédéfinie.
- Le développement des différents agents suivant une hierarchie de classe (Uniform, Epsilon-greedy, SoftMax, UCT)
- Le developpement d'une classe de visualisation permettant de comparer les performances des différents agent dans le temps en plottant le gain cumulé dans le temps d'une expérience, ainsi que la variance de ce gain en intégrant une série d'experiences pour le même agent.

Les bibliothèques logicielles utilent pour ce TP seront essentiellement Numpy et Matplotlib. Nous démarrerons l'exercice en créant un environment virtuel spécifique où installer ces bibliothèques nécessaires.

Bonus

- Version à deux joueurs : Créer une version du jeu où deux agents peuvent partager leurs connaissances et sélectionner simultanément des positions pour accumuler des récompenses.
- Version non-stationnaire : Créer une version où les paramètres de la distribution (Gaussienne) modélisant l'espérance de gain des cellules du terrain changent périodiquement.

Architecture

Terrain

Cette classe représente le terrain bidimensionnel sur lequel se déroule le jeu. Elle contient les informations sur les récompenses associées à chaque position.

```
class Terrain:
    def __init__(self, size_x, size_y):
        # Initialise le terrain avec la taille (size_x, size_y)
        # G n re al atoirement les r compenses pour chaque position
        pass

    def get_reward(self, x, y):
        # Renvoie la r compense    la position (x, y)
        pass

    def visualize(self):
        # Visualise le terrain et les r compenses associ es
        pass
```

Agent

La classe Agent représente l'agent qui interagit avec le terrain en choisissant des positions pour accumuler des récompenses.

```
class Agent:
    def __init__(self, terrain, trials):
        # Initialise l'agent avec le terrain et le nombre d'essais
        pass

    def select_position(self):
        # S lectionne une position (x, y) sur le terrain
        pass

    def update_knowledge(self, x, y, reward):
        # Met    jour la connaissance de l'agent sur le terrain
        pass
```

Visualizer

Cette classe sert à visualiser les performances de l'agent et la connaissance actuelle du terrain.

```
class Visualizer:
    def __init__(self, agent):
        # Initialise le visualiseur avec l'agent
        pass

    def plot_performance(self, performance_data):
        # Trace les performances de l'agent au fil du temps
        pass

    def draw_terrain_knowledge(self):
        # Dessine la connaissance actuelle du terrain par l'agent
        pass
```