

Minigrid – Door Key – Uma análise de modelos (DQN x PPO)

Giancarlo Vanoni Ruggiero, Luciano Felix Dias, Tales Ivalque

Engenharia da Computação, INSPER

Prof. Fabrício Barth

Abstract—Neste projeto foi desenvolvido e validado um agente capaz de atuar no ambiente Door Key. O método de treinamento empregado foi modificar o ambiente para receber recompensas em estágios intermediários, como abrir a porta e pega a chave. Os resultados obtidos foram que utilizando o DQN o agente não foi capaz de aprender, enquanto com o PPO e com as modificações no ambiente, o agente conseguiu aprender.

keywords—Reinforcement Learning, DQN, PPO, Minigrid, Door Key

1. Introdução

O projeto tem como objetivo comparar o desempenho do ambiente single-agent *Minigrid – Doorkey* utilizando os algoritmos DQN (*Deep Q-Network*) e PPO (*Proximal Policy Optimization*). Neste ambiente, há uma chave que o agente deve adquirir para destrancar uma porta e chegar ao objetivo no quadrado verde.

2. Ambiente

Table 1. Direções do espaço de observação

ID	Vetor direção
0	(1, 0)
1	(0, 1)
2	(-1, 0)
3	(0, -1)

O espaço de observação é descrito por um dicionário contendo a direção do agente *direction*, a imagem observável *image* e o espaço de missão *mission*. Sendo: *direction* um atributo discreto de valor como descrito na tabela 1; *image* sendo uma matriz RGB quadrada de valores inteiros contidos no intervalo [0, 255] do tamanho do campo de visão do agente, e sendo este parametrizável como numero inteiro ímpar maior que 2 por padrão 7; e *mission* sendo o universo de ação do agente como uma matriz retangular de tamanho arbitrário contendo trincas seguindo a seguinte estrutura, respectivamente:

Table 2. Objetos do espaço de observação

ID	Nome	Objeto
0	UNSEEN	Não visível
1	EMPTY	Vazio
2	WALL	Parede
3	FLOOR	Chão
4	DOOR	Porta
5	KEY	Chave
6	BALL	Bola
7	BOX	Caixa
8	GOAL	Objetivo
9	LAVA	Lava
10	AGENT	Agente

O espaço de ação é discreto com ações de movimento e poucas interações com elementos do ambiente. O espaço de ação é implementado conforme especificado na tabela 5.

A função de recompensa é definida no domínio contínuo do intervalo [0, 1] e proporcional a razão entre o número de passos e o número máximo de passos permitido para o agente atingir o estado final com sucesso, caso contrário a recompensa é anulada. Esta razão é definida com maior rigor na equação 1.

Table 3. Cores do espaço de observação

ID	Nome	Vetor RGB
0	RED	(255, 0, 0)
1	GREEN	(0, 255, 0)
2	BLUE	(0, 0, 255)
3	PURPLE	(112, 39, 195)
4	YELLOW	(255, 255, 0)
5	GREY	(100, 100, 100)

Table 4. Estados do espaço de observação

ID	Nome	Estado
0	OPEN	Aberto
1	CLOSED	Fechado
2	LOCKED	Trancado

$$\text{RECOMPENSA} = \begin{cases} 1 - 0.9 \cdot \frac{\# \text{PASSOS}}{\max(\# \text{PASSOS})}, & \text{se sucesso} \\ 0, & \text{caso contrário} \end{cases} \quad (1)$$

3. Método

Para conseguir comparar bem o desempenho dos dois algoritmos no ambiente, será analisado o crescimento da recompensa de um agente treinado em cada um desses algoritmos, utilizando a *MlpPolicy*, e 200.000 passos de aprendizado. O ambiente utilizado dentro do *Minigrid - Doorkey* será o "MiniGrid-DoorKey-6x6-v0", que já oferece uma complexidade razoável para o agente. Mas, para obter uma comparação satisfatória, é necessário primeiramente alterar a função recompensa de modo a ficar mais nítido o aprendizado de cada agente.

Atenção

Para treinamento será utilizado a biblioteca *stable baselines*.

Inicialmente, para conseguir utilizar a biblioteca *stable baselines*, foi necessário utilizar do *ImgObsWrapper*, que é um *Wrapper* específico do ambiente, que transforma a observação do *environment* em um formato de Imagem, próprio para interpretação da biblioteca. Junto a isso, foi necessário utilizar dois outros *Wrappers* que alterassem a recompensa entregue ao agente para que o treinamento pudesse ser realizado e analisado. Desses, um é o *PositionBonusWrapper*, que também é específico do ambiente *Minigrid* e permite a recompensa

Table 5. Espaço de ação

Número	Nome	Ação
0	LEFT	Vira para a esquerda
1	RIGHT	Vira para a direita
2	FORWARD	Move para a frente
3	PICKUP	Pega um objeto
4	DROP	Não utilizado
5	TOGGLE	Alternar/ativar um objeto
6	DONE	Não utilizado

da exploração do agente, o outro é um *Wrapper* original feito pela equipe de forma para recompensar as ações intermediárias antes de chegar ao objetivo final.

O *Wrapper* original implementado tem o seguinte funcionamento:

```
1 from gymnasium import Wrapper
2 from minigrid.core.world_object import Door, Goal, Key
3
4 class CustomRewardWrapper(Wrapper):
5     def __init__(self, env):
6         super().__init__(env)
7         self.key_was_picked = False
8         self.door_was_unlocked = False
9
10    def step(self, action):
11        obs, reward, terminated, truncated, info = self
12        .env.step(action)
13
14        if action == self.unwrapped.actions.pickup and
15        self.unwrapped.carrying and isinstance(self.
16        unwrapped.carrying, Key) and not self.
17        key_was_picked:
18            reward = 0.5
19            self.key_was_picked = True
20
21        elif action == self.unwrapped.actions.toggle:
22            front_cell = self.unwrapped.grid.get(*self.
23            unwrapped.front_pos)
24
25            if front_cell and isinstance(front_cell,
26            Door):
27                if front_cell.is_locked and self.
28                unwrapped.carrying and isinstance(self.unwrapped.
29                carrying, Key) and not self.door_was_unlocked:
30                    reward = 0.5 # Assign a reward for
31                    unlocking a door
32                    self.door_was_unlocked = True
33                    elif not front_cell.is_locked:
34                        reward = -0.25
35                    else:
36                        reward = -0.1
37                else:
38                    reward = -0.1
39            elif terminated and self.unwrapped.agent_pos ==
40            (self.unwrapped.width - 2, self.unwrapped.height
41            - 2):
42                reward = 2
43            else:
44                reward = -0.1
45        return obs, reward, terminated, truncated, info
```

Code 1. Código do Wrapper

Agora, a função recompensa é definida no domínio contínuo do intervalo $[-0.25, 3]$, levando em conta as ações pegar a chave e destrancar a porta, além de punir o número de passos do agente, isso na sua primeira parcela. O *PositionBonusWrapper* provê uma segunda parcela inversamente proporcional à raiz do número de vezes que uma determinada posição foi visitada. O cálculo da recompensa final pode ser visto nas equações 2 e 3.

$$\text{NEW REWARD} = \begin{cases} -0.1, & \text{Por cada step} \\ 0.5, & \text{Pegar a chave ou destrancar a porta} \\ -0.25, & \text{Caso feche uma porta} \\ 2, & \text{Ao chegar ao destino final} \end{cases} \quad (2)$$

$$\text{NEW REWARD} += \frac{1}{\sqrt{\#\text{VISITAS}}} \quad (3)$$

Após a configuração dos rewards, vem a etapa de treinamento dos agentes. O setup é feito da seguinte maneira para o agente DQN:

```
1 from minigrid.wrappers import ImgObsWrapper,
2   PositionBonus
3 from stable_baselines3 import DQN
4 from stable_baselines3.common.evaluation import
5   evaluate_policy
6 from stable_baselines3.common.logger import configure
7 from MinigridFeaturesExtractor import
```

```
MinigridFeaturesExtractor
6 import gymnasium as gym
7
8 from CustomRewardWrapper import CustomRewardWrapper
9
10 results_path = "./results/dqn"
11
12 new_logger = configure(results_path, ["stdout", "csv",
13   "tensorboard"])
14
15 policy_kwargs = dict(
16     features_extractor_class=MinigridFeaturesExtractor,
17     features_extractor_kwargs=dict(features_dim=128),
18 )
19
20 env = gym.make("MiniGrid-DoorKey-6x6-v0", render_mode="
21   rgb_array")
22 env = ImgObsWrapper(env)
23 env = CustomRewardWrapper(env)
24 env = PositionBonus(env)
25
26 model = DQN("MlpPolicy",
27   env,
28   policy_kwargs=policy_kwargs,
29   verbose=0,
30   exploration_fraction=0.001,
31   exploration_final_eps=0.5)
32
33 model.set_logger(new_logger)
34 model.learn(500_000, progress_bar=True)
35 model.save(results_path + "/dqn_minigrid")
36
37 mean_reward, std_reward = evaluate_policy(model, model.
38   get_env(), n_eval_episodes=10)
39 print(f'Mean reward: {mean_reward} +/- {std_reward:.2f}')
```

Code 2. Modelo DQN

E para o modelo PPO:

```
1 from minigrid.wrappers import ImgObsWrapper
2 from stable_baselines3 import PPO
3 from stable_baselines3.common.evaluation import
4   evaluate_policy
5 from stable_baselines3.common.logger import configure
6 from minigrid.wrappers import PositionBonus
7 from MinigridFeaturesExtractor import
8   MinigridFeaturesExtractor
9 import gymnasium as gym
10
11 from CustomRewardWrapper import CustomRewardWrapper
12
13 results_path = "./results/ppo"
14
15 new_logger = configure(results_path, ["stdout", "csv",
16   "tensorboard"])
17
18 policy_kwargs = dict(
19     features_extractor_class=MinigridFeaturesExtractor,
20     features_extractor_kwargs=dict(features_dim=128),
21 )
22
23 env = gym.make("MiniGrid-DoorKey-6x6-v0", render_mode="
24   rgb_array")
25 env = ImgObsWrapper(env)
26 env = CustomRewardWrapper(env)
27 env = PositionBonus(env)
28
29 model = PPO("MlpPolicy",
30   env=env,
31   policy_kwargs=policy_kwargs,
32   verbose=0,
33 )
34
35 model.set_logger(new_logger)
36 model.learn(200_000, progress_bar=True)
37 model.save(results_path + "/ppo_minigrid")
38
39 mean_reward, std_reward = evaluate_policy(model, model.
40   get_env(), n_eval_episodes=10)
41 print(f'Mean reward: {mean_reward} +/- {std_reward:.2f}')
```

Code 3. Modelo PPO

4. Resultados

Através dos treinamentos foi possível obter a curva de aprendizagem mostrada na Figura 1. Através dela é possível notar que após 100000 episódios o PPO consegue convergir, o DQN não converge em nenhum momento. Também é possível notar que o *reward* começa alto em ambos e vai decaindo, até se estabilizar.

Outro resultado que encontramos foi que mesmo treinando no ambiente 6x6 o treinamento funcionou no ambiente 8x8. Isso se deve ao fato de que como o ambiente não é totalmente observável o agente aprendeu a procurar os objetos para chegar em seu objetivo.

5. Considerações finais

O objetivo deste trabalho foi explorar o ambiente *Minigrid-Doorkey*, que possui características únicas, como o fato de ter um espaço de observação limitado e também depender de aleatoriedade, já que originalmente o agente só iria receber recompensa caso chegasse no final. Ao longo do desenvolvimento foi possível modificar o ambiente para que o agente tenha mais oportunidades de aprendizado, o que permitiu, através do algoritmo PPO, o agente

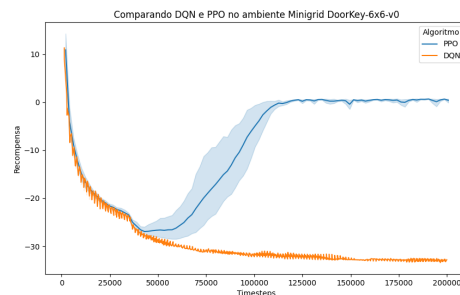


Figure 1. Curva de aprendizado MiniGrid-6x6-v0, comparando DQN e PPO

aprender como chegar no objetivo.