
	Politechnika Bydgoska im. J. J. Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki		
Przedmiot	Analiza Regresji i Szeregów Czasowych		
Prowadzący	dr inż. Damian Ledziński		
Temat	RL Stable Baselines i Własny model		
Student	Cezary Tytko		
Nr lab.	5 i 6	Data wykonania	24.06.2024
Ocena		Data oddania spr.	

Zadaniem jest rozwiązanie problemu RL dla gry z Atari z wykorzystaniem bibliotek Gymnasium i Stable Baselines.

Lab 5. Model Predefiniowany

Jako grę w którą model będzie uczył się grać wybrałem FrozenLake w wersji planszy 8x8: „FrozenLake8x8-v1”.

Najpierw wybrałem Algorytm PPO i model predefiniowany MlpPolicy i trenowałem go w specjalnie przygotowanej pętli, która co iterację treningu, generowała kilka testowych klatek działania algorytmu, tym samym obrazując zmianę w podejściu modelu do problemu w czasie

```

1. # Wyświetl wszystkie dostępne środowiska
2. print(gym.envs.registry.keys())
3.
4. # Utwórz środowisko
5. env = gym.make('FrozenLake8x8-v1', render_mode='rgb_array')
6. env = DummyVecEnv([lambda: env])
7.
8. # Utwórz model
9. model = PPO('MlpPolicy', env, verbose=1)
10.
11. timesteps_per_iteration = 8000
12. iterations = 100
13. # Trenuj model
14. frames = []
15. for _ in range(iterations):
16.     obs = env.reset()
17.     model.learn(total_timesteps=timesteps_per_iteration, reset_num_timesteps=False)
18.     obs = env.reset()
19.     for _ in range(10):
20.         action, _states = model.predict(obs)
21.         obs, rewards, dones, info = env.step(action)
22.         frame = env.render("rgb_array")

```

```

23.         frames.append(frame)
24.
25. # Zapisz model
26. model.save("ppo_FrozenLake")
27.
28. output_file_avi = 'testowanie_modelu.avi' # Przykładowe rozszerzenie AVI
29. fps = 10 # Szybkość klatek na sekundę
30.
31. # Zapisz animację do pliku AVI
32. imageio.mimsave(output_file_avi, frames, fps=fps)
33.

```



testowanie_modelu.avi



PPO_testowanie_modelu.mp4

Wytrenowałem również inny algorytm uczenia RL: A2C dla tego samego problemu i z tym samym domyślnym modelem MlpPolicy:

```

1. print(gym.envs.registry.keys())
2.
4. env = gym.make('FrozenLake8x8-v1', render_mode='rgb_array')
5. env = DummyVecEnv([lambda: env])
6.
8. model = A2C('MlpPolicy', env, verbose=1)
9.
10. timesteps_per_iteration = 8000
11. iterations = 100
12. frames = []
13. for _ in range(iterations):
14.     obs = env.reset()
15.     model.learn(total_timesteps=timesteps_per_iteration, reset_num_timesteps=False)
16.     obs = env.reset()
17.     for _ in range(10):
18.         action, _states = model.predict(obs)
19.         obs, rewards, dones, info = env.step(action)
20.         frame = env.render("rgb_array")
21.         frames.append(frame)
22.
23.
25. model.save("A2C_FrozenLake")
26.
27. output_file_avi = 'testowanie_modelu_A2C.avi'
28. fps = 10
29.
31. imageio.mimsave(output_file_avi, frames, fps=fps)
32.

```



testowanie_modelu_A2C.avi



testowanie_modelu_A2C.mp4

Lab 6. Własny Model do RL

Własne rozwiązania w przypadku RL można implementować na różnych poziomach: Analizy wejścia (np. obrazu) i ekstrakcji nowych cech które służą do uczenia Modelu odpowiadającego za podejmowanie decyzji, którego wyjściem są akcje jakie możemy podjąć, oraz model do oceny wartości stanu gry (jak dobry jest stan na planszy).

W ramach ćwiczenia przygotowałem przykład jak własnoręcznie utworzyć model do podejmowania akcji i analizy wartości:

```
1. class CustomNetwork(nn.Module):
10.
11.     def __init__(
12.         self,
13.         feature_dim: int,
14.         last_layer_dim_pi: int = 64,
15.         last_layer_dim_vf: int = 64,
16.     ):
17.         super().__init__()
18.
19.         # IMPORTANT:
20.         # Save output dimensions, used to create the distributions
21.         self.latent_dim_pi = last_layer_dim_pi
22.         self.latent_dim_vf = last_layer_dim_vf
23.
24.         # Policy network
25.         self.policy_net = nn.Sequential(
26.             nn.Linear(feature_dim, last_layer_dim_pi), nn.ReLU()
27.         )
28.         # Value network
29.         self.value_net = nn.Sequential(
30.             nn.Linear(feature_dim, last_layer_dim_vf), nn.ReLU()
31.         )
32.
33.     def forward(self, features: torch.Tensor) -> Tuple[torch.Tensor, torch.Tensor]:
34.         return self.forward_actor(features), self.forward_critic(features)
35.
36.     def forward_actor(self, features: torch.Tensor) -> torch.Tensor:
37.         return self.policy_net(features)
38.
39.     def forward_critic(self, features: torch.Tensor) -> torch.Tensor:
40.         return self.value_net(features)
41.
42. class CustomActorCriticPolicy(ActorCriticPolicy):
43.     def __init__(
44.         self,
45.         observation_space: spaces.Space,
46.         action_space: spaces.Space,
47.         lr_schedule: Callable[[float], float],
48.         *args,
49.         **kwargs,
50.     ):
51.         # Disable orthogonal initialization
52.         kwargs["ortho_init"] = False
53.         super().__init__(
54.             observation_space,
55.             action_space,
56.             lr_schedule,
57.             # Pass remaining arguments to base class
58.             *args,
59.             **kwargs,
60.         )
```

```

65.         )
66.
67.         def _build_mlp_extractor(self) -> None:
68.             self.mlp_extractor = CustomNetwork(self.features_dim)
69.
70.
71. # Tworzenie środowiska
72. env = gym.make('FrozenLake-v1', render_mode='rgb_array')
73. env = DummyVecEnv([lambda: env])
74.
75. model = PPO(CustomActorCriticPolicy, env, verbose=1)
76.
77. frames = []
78.
79. timesteps_per_iteration = 8000
80. iterations = 100
81.
82. # Trenuj model
83. frames = []
84. for _ in range(iterations):
85.     obs = env.reset()
86.     model.learn(total_timesteps=timesteps_per_iteration, reset_num_timesteps=False)
87.     obs = env.reset()
88.     for _ in range(30):
89.         action, _states = model.predict(obs)
90.         obs, rewards, dones, info = env.step(action)
91.         frame = env.render("rgb_array")
92.         frames.append(frame)
93.
94. # Zapisz model
95. model.save("ppo_FrozenLake_custom")
96.
97. # Zapisz animację do pliku AVI
98. output_file_avi = 'PPO_Custom_testowanie_modelu.avi'
99. fps = 10 # Szybkość klatek na sekundę
100. # Zapisz animację do pliku AVI
101. imageio.mimsave(output_file_avi, frames, fps=fps)
102.
103. print(f"Zapisano animację do pliku {output_file_avi}")
104.

```



PPO_Custom_testowanie_modelu.avi

Klasa „CustomNetwork” stanowi nasz model, natomiast „CustomActorCriticPolicy” opakowuje go i pozwala przekazać jako własną politykę do algorytmu uczenia RL PPO.

Wnioski:

Uczenie ze wzmocnieniem jest bardzo trudnym i skomplikowanym zagadnieniem, nie tylko pod kątem samego modelu gdzie mamy elementy takie jak aktor, środowisko i dostępne akcje, ale implementacja w środowisku gymnasium i stable baselines również nie należy do najłatwiejszych, głównie przez mnogość funkcji, dlatego zaprezentowane rozwiązania działają, ale modele słabo się uczą i nie prezentują sobą idealnego rozwiązania, jednak na podstawie przedstawionych przykładów, można starać się je rozbudować.