
	Politechnika Bydgoska im. J. J. Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki		
Przedmiot	Analiza Regresji i Szeregów Czasowych		
Prowadzący	dr inż. Damian Ledziński		
Temat	Gra Atari i Stable-Baselines		
Student	Zuzanna Tarazewicz	Nr Indeksu	116954
Ocena		Data Oddania	

Spis Treści

Cel	3
Wykonanie zadania	3
Wybór gry	3
Wybór modelu	3
Wykonanie	3
Zaimportowanie odpowiednich bibliotek	3
Zdefiniowanie środowiska	4
Trening modelu	4
Testowanie modelu	5
Wynik	6
Wnioski	6

Cel

Celem laboratorium było wyuczenie algorytmu ze wzmocnieniem (RL, Reinforcement Learning) umiejącego grać w wybraną grę Atari.

Uczenie ze wzmocnieniem jest gałęzią uczenia maszynowego. Zajmuje się tym, w jaki sposób agenci mogą podejmować decyzje w środowiskach, aby zmaksymalizować nagrody. Jednym z zastosowań jest trenowanie agentów do gier komputerowych, gdzie agent musi się nauczyć optymalnych strategii na podstawie doświadczeń ze środowiska.

Wykonanie zadania

Wybór gry

W tym przypadku padło na grę Zaxxon z biblioteki OpenAI Gymnasium (znane w starszej wersji jako Gym). Biblioteka Gymnasium zawiera dużo środowisk do tworzenia agentów uczenia ze wzmocnieniem, w tym posiada sporą kolekcję gier Atari.

Wybór modelu

Po zrobieniu wstępnego rozeznania w temacie uczenia ze wzmocnieniem wybór padł na agenta Deep Q Learning (DQN).

Wykonanie

Zaimportowanie odpowiednich bibliotek

```
import gymnasium as gym
from gymnasium.wrappers import AtariPreprocessing, FrameStack
from stable_baselines3 import DQN
from stable_baselines3.common.vec_env import DummyVecEnv
from stable_baselines3.common.vec_env import DummyVecEnv, VecVideoRecorder
from stable_baselines3.common.monitor import Monitor
```

Uczenie następuje za pomocą biblioteki gymnasium i stable_baselines3.

Początkowo wybór padł na starszą wersję biblioteki gymnasium – gym, jednakże ze względu na problemy z utworzeniem środowiska i brak dalszego wsparcia – wybór padł na nowszą wersję biblioteki. Za pośrednictwem tych dwóch bibliotek możliwe było wybór środowiska, wybór trybu renderowania, wyuczenie modelu i zapisanie video modelu podczas gry.

Zdefiniowanie środowiska

```
def makeEnv(envId: str, renderMode = None):
    env = gym.make(envId, render_mode=renderMode)
    env = AtariPreprocessing(env, frame_skip=1, scale_obs=True)
    env = FrameStack(env, 4)

    return env

envId = 'ALE/Zaxxon-v5'
env = DummyVecEnv([lambda: makeEnv(envId)])
```

Środowisko zostaje zdefiniowane za pośrednictwem funkcji makeEnv, która przyjmuje od jednego do dwóch argumentów (drugi argument jest predefiniowany, dlatego można go pominąć) i zwraca utworzone środowisko, które zostaje zapisane pod zmienną env.

Trening modelu

```
model = DQN('MlpPolicy', env, verbose=1, buffer_size=10000,
learning_starts=1000,
            batch_size=32, gamma=0.99, target_update_interval=1000,
train_freq=4,
            gradient_steps=1, exploration_fraction=0.1,
exploration_final_eps=0.01)
model.learn(total_timesteps=50000)
model.save('dqn_zaxxon.pkl')
```

Trening modelu był najdłuższym elementem tworzenia programu. Szkolony agent DQN został wyuczony w 50000 krokach. Po wyuczeniu model został zapisany jako dqn_zaxxon.pkl. Wynik uczenia był następujący:

rollout/	
exploration_rate	0.01
time/	
episodes	56
fps	351
time_elapsed	141
total_timesteps	49603
train/	
learning_rate	0.0001
loss	0.000128
n_updates	12150

Co jest dość zadawalającym wynikiem, ze względu na to, że strata jest bardzo bliska zeru. Oznacza to, że algorytm ma małą stratę – więc uczenie przebiega pomyślnie.

Testowanie modelu

```
videoFolder = 'videos/'

def makeRecEnv(envId: str, renderMode = 'rgb_array'):
    env = gym.make(envId, render_mode=renderMode)
    env = AtariPreprocessing(env, frame_skip=1)
    env = FrameStack(env, 4)
    env = Monitor(env)

    return env

recording_env = DummyVecEnv([lambda: makeRecEnv(envId)])
recording_env = VecVideoRecorder(recording_env, videoFolder,
record_video_trigger=lambda x: x == 0, video_length=10000,
name_prefix="dqn_zaxxon")

modelPath = 'dqn_zaxxon.pkl'
model = DQN.load(modelPath)
obs = recording_env.reset()

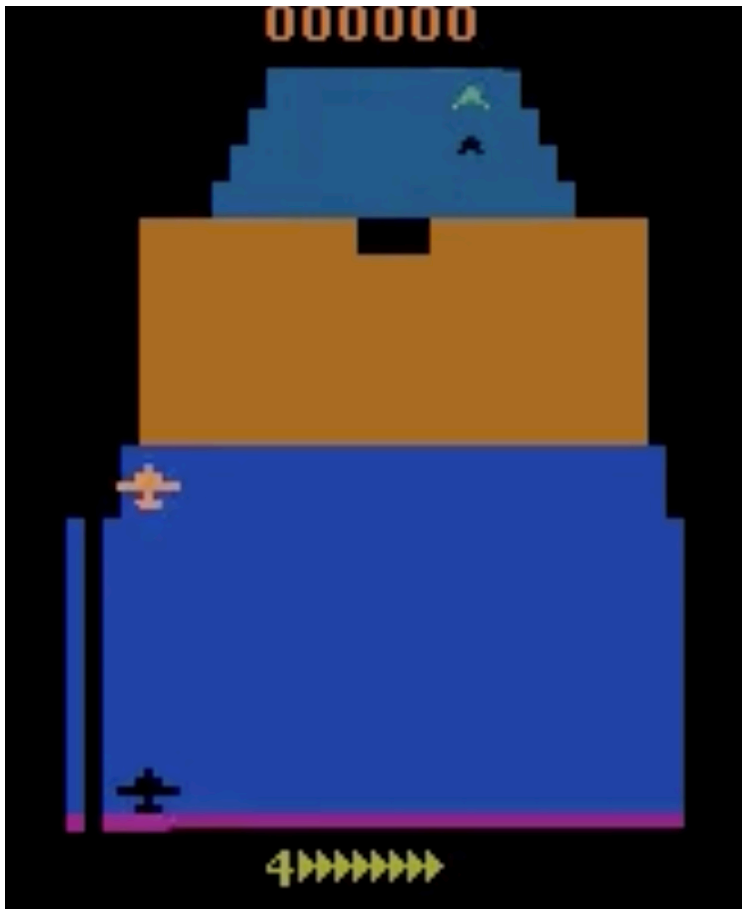
for _ in range(10000):
    action, _states = model.predict(obs)
    obs, rewards, dones, infos = recording_env.step(action)

    if dones:
        obs = recording_env.reset()

recording_env.close()
```

Podczas testowania modelu wykorzystany jest zapisany wcześniej model i utworzenie nowego, testowego środowiska, w którym powstanie nagranie działania programu. Podczas testowania zostały nagrane pliki mp4, w różnych miejscach testu, co prezentuje to jak naprawdę działa. Z tym etapem nastąpiły problemy z brakami pewnych programów. Trzeba było uczyć program na nowo po zainstalowaniu programu ffmpeg i biblioteki moviepy. Wymaganie biblioteki moviepy było szczególnie interesujące, ponieważ nie jest ona w żaden sposób importowana do programu.

Wynik



Pomimo pomyślnych testów, algorytm nie radzi sobie najlepiej, jednakże sukcesem jest to, że udało się przeprowadzić uczenie od początku do końca – włącznie z nagraniem prezentacji działania programu.

Wnioski

- Pomimo użycia mniejszej ilości kodu do napisania algorytmu uczenia ze wzmocnieniem, niż przy sieciach neuronowych, to napisanie tego było bardziej skomplikowane.
- Pomimo interesującego systemu nagród i kar, ciężiej kontrolować to, czy taki algorytm na pewno uczy się dobrze.
- Biblioteka gymnasium od OpenAI zapewnia dużą ilość środowisk, które pomagają zagłębić się w tematykę uczenia ze wzmocnieniem.