Dokumentacja do projektu z AMHE

Roman Moskalenko Pavel Tarashkevich

Spis treści

1 Treść zadania

Zadanie 10.

Zaprojektuj algorytm ewolucyjny i zbadaj jego działanie w jednym z zadań zdefiniowanych w ramach środowiska OpenAI poświęconym kontroli.

Specyfikacja zadań znajduje się pod poniższym linkiem:

gym.openai.com/envs/#classic_control

Porównaj wyniki swojego rozwiązania z dwoma wybranymi rozwiązaniami, które bazują na innych podejściach.

2 Projekt wstępny

2.1 Opis problemu i jego sposób rozwiązania

Zadania kontroli klasycznej zakładają sterowanie obiektu fizycznego w sposób, jak najbardziej efektywny dla jego stanu. Funkcja sterowania może być bardzo skomplikowana, uciążliwe jest jej zaprojektowanie w sposób ręczny.

Algorytmy uczenia się ze wzmocnieniem (dalej RL) są wykorzystywane w zadaniach związanych z kontrolą. Do ewaluacji tych algorytmów powstał zestaw środowisk OpenAI gym.

Alternatywnym podejściem do zadań kontroli jest bezpośrednie stosowanie sieci neuronowych, m.i. podejście neuroewolucji zastosowane w [1], które zakłada połączenie sieci neuronowych i algorytmów ewolucyjnych.

Do rozwiązania tego problemu planujemy zastosować sieć MLP, gdzie wejściem sieci będzie stan rozważanego środowiska, a wyjściem będzie akcja do wyboru przez agenta. Sieć ta będzie optymalizowana za pomocą algorytmu ewolucyjnego CMA-ES.

2.2 Planowane eksperymenty numeryczne

W ramach eksperymentów ocenimy działanie implementacji naszego algorytmu neuroewolucji na różnych środowiskach OpenAI gym.

Następnie porównamy jego działanie z dwoma algorytmami RL pod kątem końcowej nagrody oraz czasu uczenia algorytmu. Wybrane przez nas algorytmy RL to A2C oraz PPO.

2.3 Biblioteki wybrane do realizacji projektu

- **gym** - liczne środowiska, w tym kontroli klasycznej, do trenowania agentów.

- **cmaes** - implementacja algorytmu CMA-ES.
- **stable-baselines** - biblioteka zawierająca gotowe implementację algorytmów RL.

3 Project końcowy

3.1 Uzupełnienie projektu wstępnego

Na dodatek do bibliotek wymienionych w 2.3 użyliśmy biblioteki **PyTorch** do implementacji perceptronu wielowarstwowego.

3.2 Przeprowadzone eksperymenty

Dany rozdział opisuje szczegóły przeprowadzonych w ramach projektu eksperymentów: opis środowisk na których działają agenty, ich uczenie oraz wnioski.

3.2.1 Opis środowisk kontroli klasycznej

Do przetestowania algorytmów rozważanych w ramach projektu wykorzystane zostały środowiska OpenAI gym [3], w szczególności dotyczące kontroli klasycznej: środowiska powszechnie znane, proste ze względu na działanie i rozwiązywalność.

W tabeli 1 znajdują się rozważane przez nas środowiska oraz ich właściwości.

Środowisko	# obs.	# akcje	typ akcji	rozwiązany dla
CartPole-v0	4	2	dyskretna	195
CartPole-v1	4	2	dyskretna	475
MountainCar-v0	2	3	dyskretna	-110
MountainCarContinuous-v0	2	1	ciągła	90
Pendulum-v1	3	1	ciągła	-140
Acrobot-v1	6	3	dyskretna	-60

Tabela 1: Właściwości środowisk kontroli klasycznej. Kolumny "obs." oraz "akcje" wskazują wymiary zbiorów obserwacji oraz akcji odpowiednio w każdym środowisku. Kolumna "rozwiązanie" zawiera wartości, które wskazują granice nagród dla każdego środowiska, powyżej których można uznać, że agent zachowuje się sensownie [2] [3].

3.2.2 Metryki do porównania algorytmów

Zgodnie z rozdziałem 2.2 porównujemy algorytm neuroewolucji - połączenie CMA-ES z siecią MLP z dwoma algorytmami RL - A2C oraz PPO.

Zdajemy sobie sprawę, że nie możemy wprost porównywać tych algorytmów, gdyż charakter uczenia w nich jest różny: algorytm neuroewolucji uczy się przeszukując przestrzeń parametrów sieci MLP, jedyną informacją zwrotną jest końcowa suma nagród otrzymana na danym środowisku. Natomiast algorytmy RL optymalizują swoją politykę zachowania na środowisku, potrafią uwzględniać natychmiastowe nagrody, a także stany i akcje, które do nich prowadzą.

Do porównania algorytmów uwzględniliśmy jak szybko algorytmy osiągają kolejne sumy nagród, ile czasu zajęło uczenie. Najważniejszą metryką porównania jest uśredniona suma nagród uzyskiwana przez modele nauczone.

3.2.3 Opis modeli do nauczenia

Algorytm neuroewolucji: użyliśmy perceptronu dwuwarstwowego z 8 neuronami w warstwie ukrytej. Perceptron ten nie zawiera dodatkowych parametrów bias. Podliczyliśmy, że liczba parametrów perceptronu będzie wystarczająco mała aby móc efektywnie stosować algorytm CMAES. Wybraliśmy ReLU jako funkcję aktywacji. Algorytm CMA-ES automatycznie dobiera liczność populacji zgodnie ze wzorem:

$$population\_size = 4 + \lfloor 3 * \ln(n) \rfloor$$

gdzie n to wymiar przestrzeni parametrów sieci. Przykładowo dla $n = 100$ liczność populacji wynosi 17. Parametr σ został wybrany arbitralnie i wynosi 1.3.

Algorytmy A2C i PPO: użyliśmy domyślnych konfiguracji modeli tych algorytmów z pakietu stable_baselines3 z polityką *MlpPolicy*.

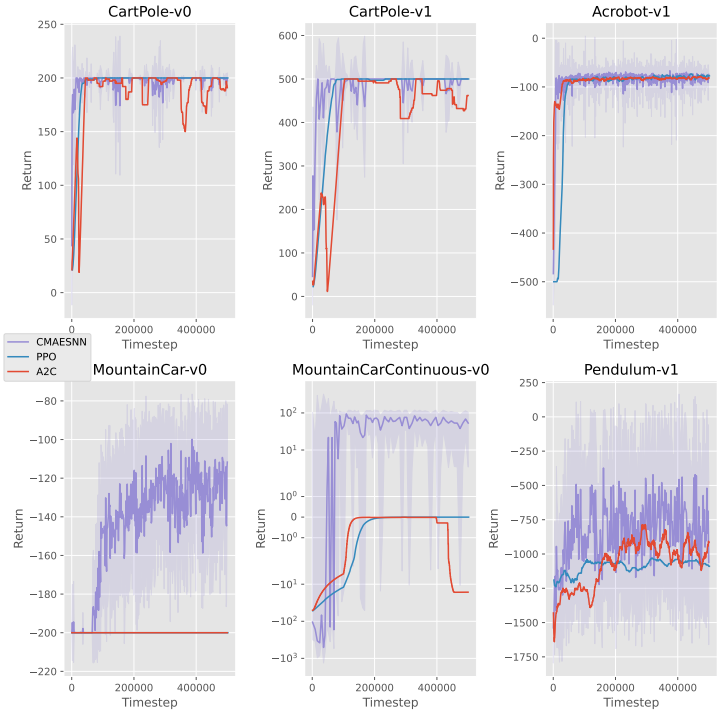
3.2.4 Uczenie agentów

Uczenie agentów dokonywano w ciągu 500 tysięcy kroków, gdzie za krok uznajemy pojedynczą akcją dokonaną przez agenta na danym środowisku.

Warto zaznaczyć, że ze względu na to, że agent algorytmu CMAESNN korzysta z populacji osobników niezależnie działających na środowisku, postanowiliśmy zliczać wykonane kroki tylko przez jednego osobnika, który uzyskał największą sumę nagród w danym epizodzie.

Uczenie jest dokonane łącznie 5 razy dla każdej pary środowisko-algorytm, aby zmniejszyć wpływ losowości na uzyskane wyniki.

Na rysunku 1 pokazano jeden z pięciu przebiegów uczenia algorytmów. Wartości na osi pionowej odpowiadają sumom nagród, które algorytmy uzyskują na końcu epizodu działania na danym środowisku.



Rysunek 1: Przebieg uczenia algorytmów w przeciągu 500,000 kroków. Dla algorytmu CMAESNN pokazano zakres sum nagród, uzyskany przez średnią sumę nagród oraz odchylenia standardowego dla całej populacji.

Jako że algorytm CMAESNN jest algorytmem przeszukiwania postanowiliśmy w trakcie uczenia śledzić najlepszą dotychczas populację poprzez najlepszą uśrednioną wartość sum nagród otrzymanych przez jej osobników. Dla porównania także zapisujemy najlepsze sumę nagród dla algorytmów RL. Wartości te są podane w tabeli 2.

W pierwszych trzech środowiskach - CartPole-v0, CartPole-v1, Acrobot-v1 - wszystkie algorytmy relatywnie szybko osiągają sumy nagród, które odpowiadają rozwiązaniu środowiska (albo bliskości do niego). Kolejne trzy - MountainCar-v0, MountainCarContinuous-v0, Pendulum-v1 - wyraźnie sprawiają kłopoty dla algorytmów RL. Algorytm CMAESNN dzięki wbudowanej właściwości do eksploracji potrafi rozwiązać pierwsze dwa z nich oraz wykazuje się lepszymi wynikami w trzecim środowisku. Warto też zwrócić uwagę na dużo większą wariancję sum nagród w populacji dla tych środowisk.

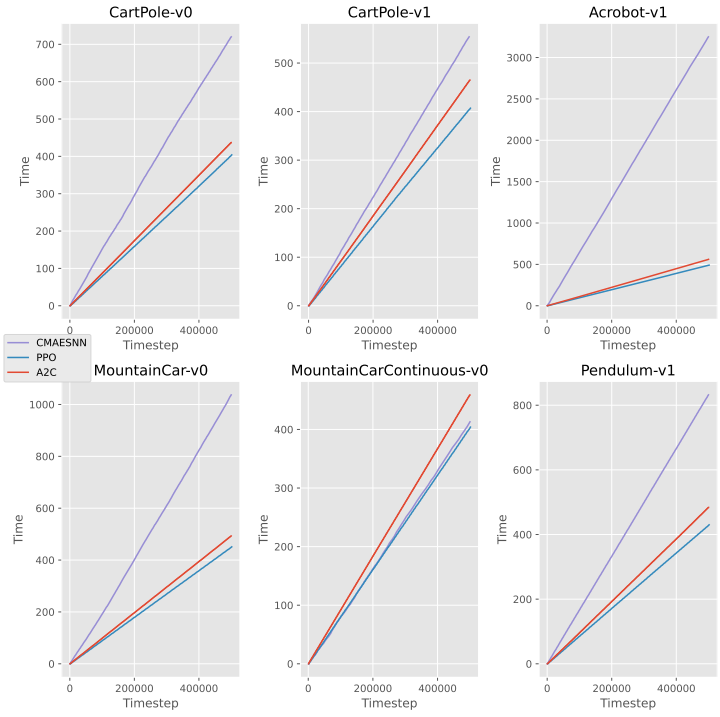
Resztę wykresów uczenia agentów można znaleźć w dodatku A.

Zostały zmierzone czasy uczenia agentów. Podajemy tylko jeden wykres, ponieważ pozostałe pomiary czasu były prawie identyczne. Na rysunku 2 wyraźnie widać liniową zależność czasu uczenia od liczby kroków. Na większości

	A2C	PPO	CMAESNN
CartPole-v0	200	200	**200**
CartPole-v1	500	500	**500**
MountainCar-v0	-200	-200	$\mathbf{-104.54 \pm 8.41}$
MountainCarContinuous-v0	0.77 ± 1.55	0.0	$\mathbf{81.20 \pm 35.97}$
Pendulum-v1	-843.2 ± 237.08	-1060 ± 29.66	$\mathbf{-221 \pm 89.43}$
Acrobot-v1	-80.44 ± 2.90	-73.98 ± 1.21	$\mathbf{-67.56 \pm 0.81}$

Tabela 2: Największę sumę nagród otrzymywane przez agentów w trakcie uczenia, uśrednione po pięciu przebiegach uczenia, oraz uśrednione odchylenia standardowe. Odchylenia standardowe dla CMAESNN nie uwzględniają odchylenia narzucanego przez rozrzut wyników w populacji.

wykresach algorytm CMAESNN spędza dużo więcej czasu na uczenie, co można wytłumaczyć tym, że algorytm w przeciągu epizodu przechodzi środowisko tyle razy, ile osobników występuje w populacji. Z innej strony dana cecha algorytmu umożliwia łatwą jego skalowalność, co jest dobrze opisane w [1].



Rysunek 2: Czasy uczenia (s) agentów w przeciągu 500,000 kroków.

Ciekawostka: obecność wartości bias w implementacji sieci neuronowej

algorytmu CMAESNN negatywnie wpłynęła na działanie na środowiskach MountainCar-v0 oraz MountainCarContinuous-v0. W dodatku B można znaleźć przebiegi uczenia CMAESNN z tymi parametrami i bez nich. Na dane zjawisko zwrócono też uwagę w [2].

3.2.5 Sprawność nauczonych agentów

Po nauczeniu agentów przetestowaliśmy sprawność ich działania. Wszystkie 5 nauczonych wersji agentów z poprzedniego rozdziału zostały uruchomione. Uśrednione (po 10 uruchomieniach) wyniki znajdują się w tabeli 3.

	A2C	PPO	CMAESNN
CartPole-v0	181.28 ± 40.90	**200**	199.7 ± 2.1
CartPole-v1	500	500	**500**
MountainCar-v0	-200	-200	$\mathbf{-125.82 \pm 33.43}$
MountainCarContinuous-v0	-0.01 ± 0.02	0.0	$\mathbf{49.57 \pm 49.74}$
Pendulum-v1	-841.44 ± 579.90	-1180.94 ± 143.50	$\mathbf{-508.85 \pm 602.58}$
Acrobot-v1	-248 ± 206.03	$\mathbf{-73.54 \pm 10.02}$	-82.6 ± 21.53

Tabela 3: Sumy nagród nauczonych agentów uśrednione po 10 uruchomieniach wraz z uśrednionym odchyleniem standardowym.

Algorytm CMAESNN sprawuje się lepiej niż algorytmy A2C i PPO w domyślnych konfiguracjach, a co więcej rozwiązuje bądź jest bliski rozwiązania dla wszystkich środowisk kontroli klasycznej.

3.3 Opis struktury projektu

Kod źródłowy projektu jest umieszczony w repozytorium GitHub na stronie https://github.com/skalermo/AMHE-cmaes-classic-control

Na rysunku 3 pokazana jest struktura projektu.

```
.
├── .gitignore
├── docs
│   └── amhe_docs.tex
├── plotting
│   ├── plot_all.py
│   ├── plot_bias_perf.py
│   └── plots
├── main.py
├── requirements.txt
├── src
│   ├── cmaes_nn.py
│   ├── env_info.py
│   ├── log_utils.py
│   └── nn.py
└── test
    ├── __init__.py
    ├── test_cmaes_nn.py
    ├── test_data
    │   ├── example_log_a2c.txt
    │   └── example_log_cmaesnn.txt
    ├── test_log_extraction.py
    ├── test_log_utils.py
    └── test_nn.py
```

Rysunek 3: Struktura plików projektu

Plik **main.py** odpowiada za główny skrypt, który wykonuje uczenie modeli oraz zapisanie wyników uczenia: logów oraz wytrenowanych modeli do katalogów **.data/logs** oraz **.data/models** odpowiednio.

Kod źródłowy projektu, z którego jest zbudowany **main.py** znajduje się w katalogu **src**:

- **cmaes_nn.py** - zawiera implementację algorytmu do rozwiązywania problemów kontroli klasycznej, jego interfejs jest zbliżony do algoryt-

mów RL z pakietu **stable-baselines3** (tzn. posiada implementacje funkcji *learn()*, *predict()*, *load()*, *save()*).
Łączy w sobie stosowanie algorytmu CMA-ES oraz perceptronu wielowarstwowego.

- **env_info.py** - zawiera listę nazw środowisk do problemów kontroli klasycznej oraz informacje o tym, czy typ akcji jest ciągły bądź dyskretny.
- **log_utils.py** - zawiera funkcje pomocnicze do przetwarzania logów programu.
- **nn.py** - zawiera funkcje do zbudowania perceptronu wielowarstwowego, a także funkcje do przypisania wektora parametrów do wag perceptronu.

Wraz z implementacją projektu powstawały testy do sprawdzenia poprawności jego działania. Znajdują się w katalogu **test**.

Katalog **plotting** zawiera skrypty do generowania wykresów na podstawie przetworzonych logów, a także katalog z wykresami wykorzystanymi w dokumentacji.

Katalog **docs** przechowuje dokumentację projektu. Plik **requirements.txt** zawiera listę wymaganych do zainstalowania bibliotek.

3.4 Uruchomienie projektu

Dla uruchomienia projektu niezbędne jest posiadania interpretera Python wersji co najmniej 3.8 oraz zainstalowanie bibliotek znajdujących w pliku **requirements.txt**:

```
$ pip install -r requirements.txt
```

Następnie można uruchomić główny skrypt za pomocą polecenia:

```
$ python main.py
```

Testy można uruchomić poleceniem:

```
$ python -m unittest discover test
```

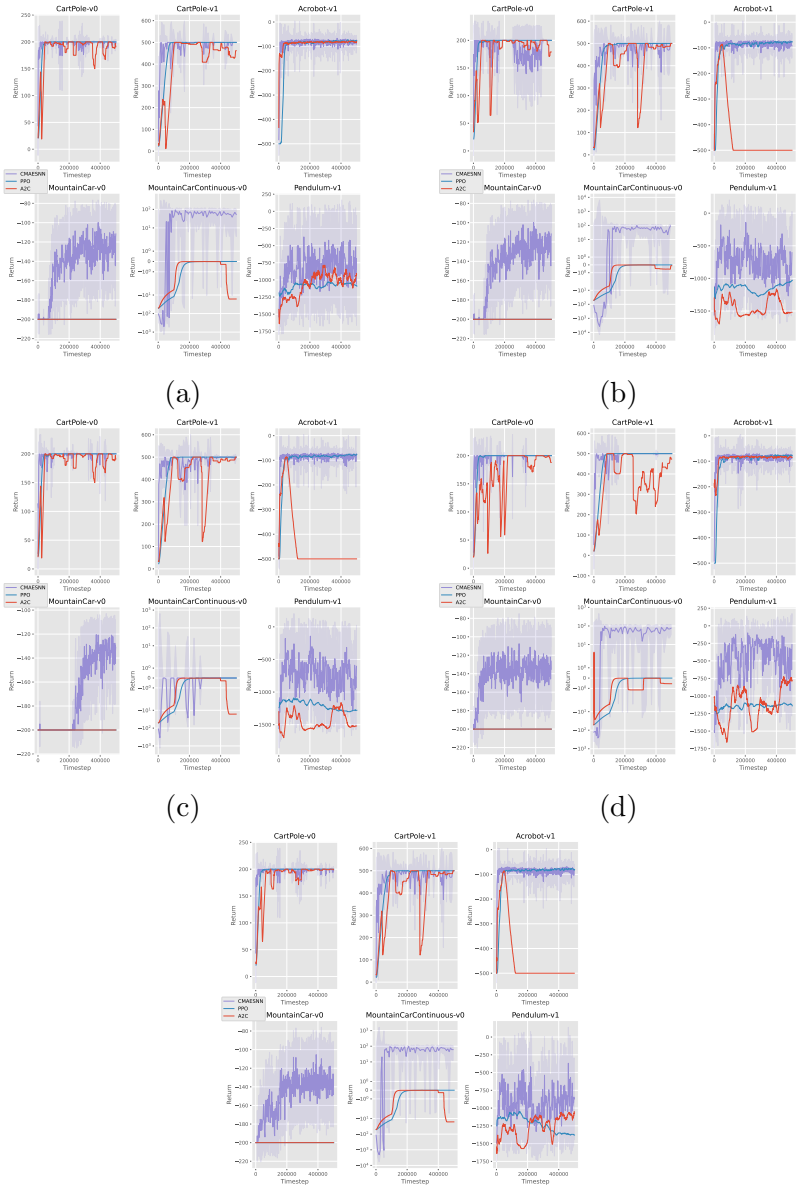
Referencje

[1] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, Ilya Sutskever, Evolution Strategies as a Scalable Alternative to Reinforcement Learning, 2017, https://arxiv.org/pdf/1703.03864v1.pdf

[2] Declan Oller, Tobias Glasmachers, Giuseppe Cuccu, Analyzing Reinforcement Learning Benchmarks with Random Weight Guessing, 2020, https://arxiv.org/pdf/2004.07707.pdf

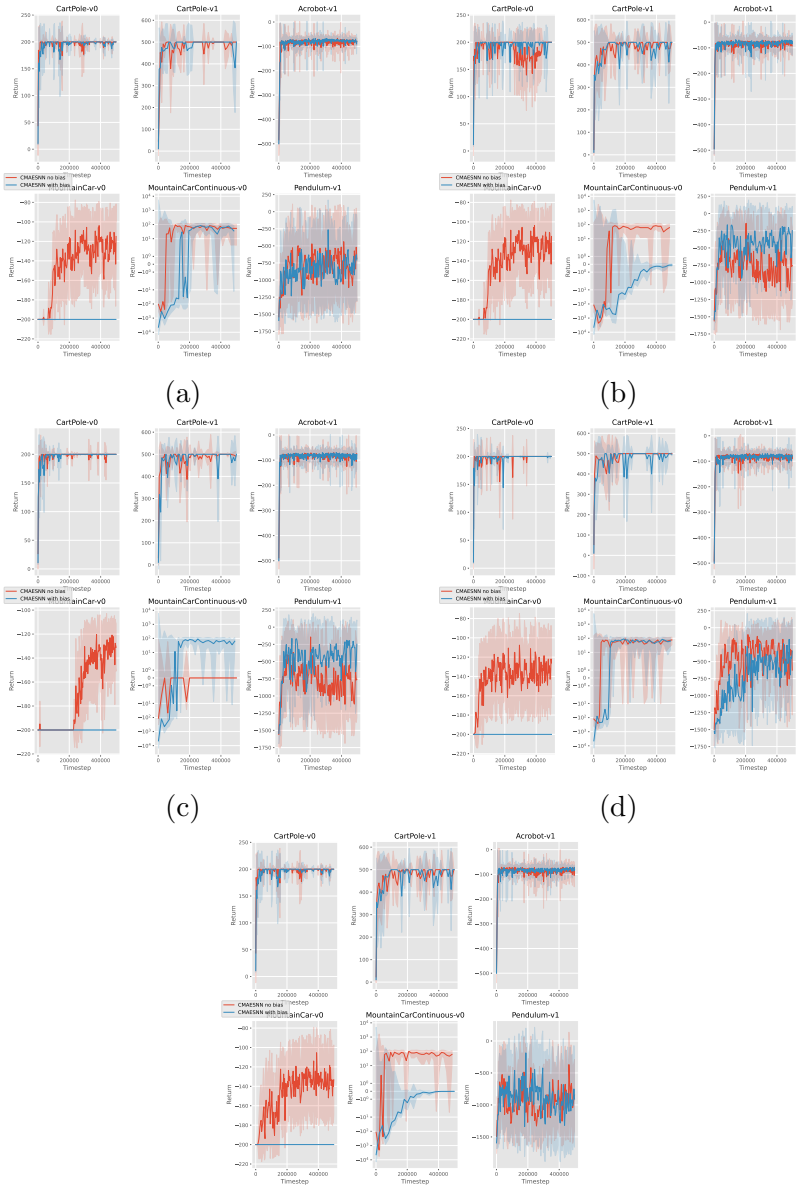
[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba, OpenAI Gym, 2016, https://gym.openai.com/ rejestr środowisk (zawiera m.i. nagrody graniczne środowisk): https://github.com/openai/gym/blob/master/gym/envs/__init__.py

A Wykresy uczenia algorymów



Rysunek 4: Pięć niezależnych przebiegów uczenia algorytmów w przeciągu 500,000 kroków. Algorytmom tym odpowiadają kolory: CMAESNN - fioletowy, PPO - niebieski, A2C - czerwony.

B Wpływ bias'u na uczenie CMAESNN



Rysunek 5: Pięć niezależnych przebiegów uczenia algorytmów CMAESNN z (niebieski) i bez (czerwony) dodatkowymi parametrami bias. Uczenie dokonywało się w przeciągu 500,000 kroków.