UNIWERSYTET ŚLĄSKI
INSTYTUT INFORMATYKI

Laura Dymarczyk, Adrian Rupala

Uczenie maszynowe do gry PacMan za pomocą OpenAI Gym

(projekt zaliczeniowy ze Sztucznej Inteligencja w Grach Komputerowych)

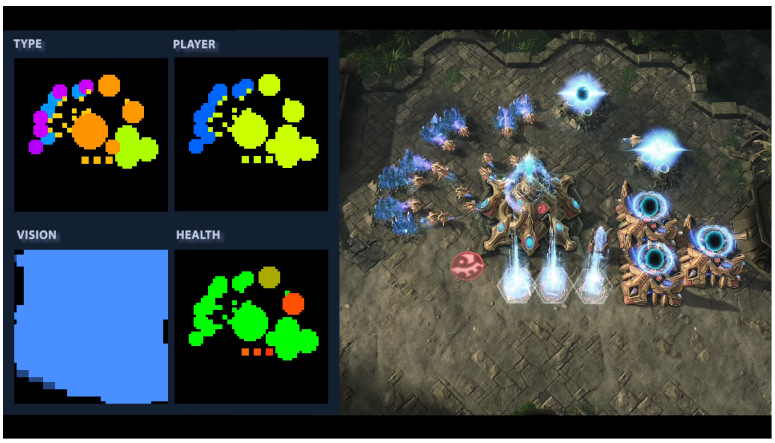
Sosnowiec 2019

1. Wstęp

Uczenie maszynowe oraz sztuczna inteligencja to jeden z najgorętszych tematów pojawiających się na ustach wielu firm. Głównym celem uczenia maszynowego jest praktyczne zastosowanie dokonań w dziedzinie sztucznej inteligencji do stworzenia autonomicznego systemu umiejącego doskonalić się przy pomocy zgromadzonych doświadczeń i nabywania na tej podstawie nowej wiedzy. Ma ono zapewnić efektywność, wydajność oraz bezawaryjność produktu.

Uczenie maszynowe jest aktualnie ściśle powiązane z grami wideo. Począwszy od pierwszej serii zwycięstw w latach 1952-1962 w turniejach szachowych, przechodząc do aktualnych osiągnięć superkomputera Watson firmy IBM, który to jest w stanie wygrać telewizyjne teleturnieje, pokonując dwóch najlepszych graczy w historii.

Aktualnymi projektami firm zajmujących się uczeniem maszynowym są systemy pokonujące profesjonalnych graczy esportowych w danych grach. Jednym z takich przykładów może zostać firma Google tworząca oraz ucząca specjalną sieć neuronową grającą w grę Starcraft 2 przy użyciu technologii DeepMind (Rysunek 1.1).



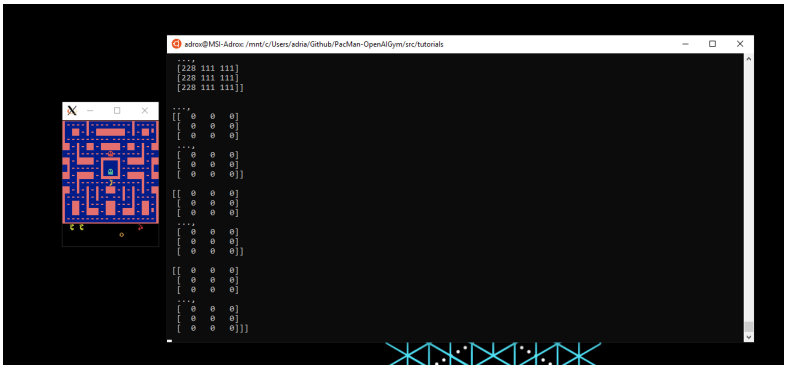
Rysunek 1.1. Starcraft 2 DeepMind

2. Teoria i algorytmy

Aktualnie na rynku istnieje wiele bibliotek oraz rozwiązań oferujących możliwość uczenia maszynowego w grach komputerowych. Dwoma najpopularniejszymi oraz najbardziej rozwijanymi projektami Open Source są właśnie zastosowane przez nas w projekcie: Tensorflow, oraz OpenAI Gym.

W naszym projekcie wykorzystaliśmy uczenie maszynowe ze wzmocnieniem. W przeciwieństwie do klasyfikacji oraz regresji, jego celem nie jest aproksymacja pewnego nieznanego odwzorowania przez generalizację zbioru. Systemowi uczącemu się ze wzmocnieniem nie są dostarczane żadne przykłady trenujące, a jedynie informacja o wartości oceniającej jego skuteczność. Cel uczenia pośrednio określony jest przez wartości wzmocnienia. W przypadku tej aplikacji został wykorzystany algorytm Q-learning. Jest to technika uczenia maszynowego która mówi agentowi jakie działania ma podjąć w określonych okolicznościach. Q-learning może zidentyfikować optymalne rozwiązanie, biorąc pod uwagę nieskończony czas poszukiwania. Wartość „Q" oznacza funkcję, która zwraca „nagrodę", którą można określić jako premię za działania podjętego w danym stanie.

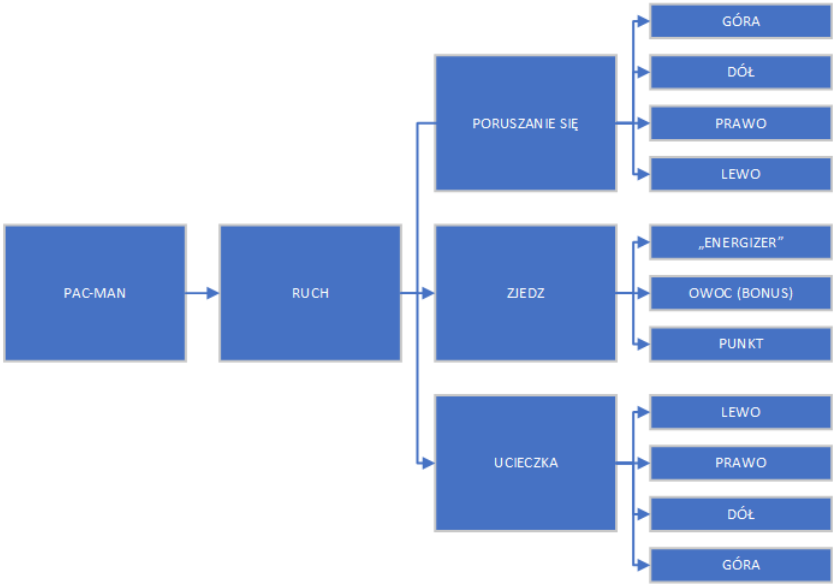
Podczas tworzenia projekt zdecydowaliśmy się na wykorzystanie biblioteki OpenAI Gym (Rysunek 2.1) dostarczającej wiele środowisk uczących. Pozwala ona na emulowanie, między innymi danej gry w pełni legalny sposób oraz interakcję z jej środowiskiem bez konieczności modyfikowania gry.



Rysunek 2.1. OpenAI Gym Atari ze środowiskiem MsPacman

Biblioteka Tensorflow została wykorzystana do zbudowania sieci neuronowej, której głównym zadaniem była obserwacja wysyłania oraz odbierania informacji o akcjach, które są w danej chwili wykonywane. Jest to sieć neuronowa deep Q-network (DQN) posiadająca 4 warstwy - warstwę „Kernel", jako najniższą warstwę biblioteki TensorFlow, warstwę „prev_layer", której zadaniem jest

przekształcenie danych wejściowych, warstwę „hidden” oraz „outputs”, jako warstwy interakcji gęsto połączone ze sobą. Wejściowe wartości neuronów są generowane losowo za pomocą biblioteki TensorFlow. Aby wybrać, jaką akcję w danej chwili PacMan ma wykonać, sieć neuronowa na początku określa prawdopodobieństwo każdej z akcji (Rysunek 2.2), a następnie wybiera akcję losowo zgodnie z oszacowanym prawdopodobieństwem. Na przykład, PacMan może iść w jednym z czterech kierunków, istnieje więc szansa, że znajdując się w rogu planszy dalej będzie próbował „wejść w ścianę”, ponieważ prawdopodobieństwo ruchu jest takie samo dla ruchu w każdym z kierunków.



Rysunek 2.2. Struktura decyzji którą może podjąć Pac-Man

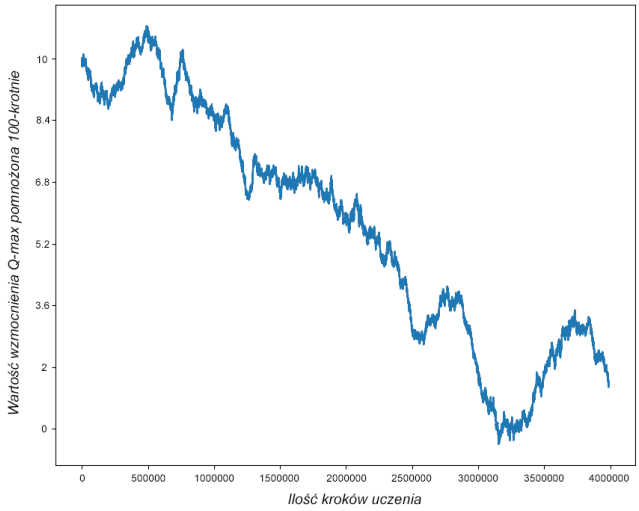
Modyfikując wartości „Q” z algorytmu Q-learning jesteśmy w stanie wpływać na proces uczenia się oraz jego jakość. Algorytm Q-learning jest przykładem algorytmu uczącego się funkcji wartości akcji. Uczy się on optymalnej funkcji wartości akcji, tak aby móc uzyskać strategię optymalną dla danego zadania. Metoda Q-learning uczy się reprezentowania zależności akcja-wartość w postaci funkcji $Q(s,a)$. Funkcja ta wyraża wartości wykonania akcji a w stanie s i jest związana z użytecznościami stanów wzorem:
$U(s,a) = \max_{\mathrm{a}} Q(s,a)$
Docelowo wartość Q spełnia równanie:
$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$
Wzór też można wykorzystać w celu aktualizacji wartości Q. Wymaga to jednoczesnego uczenia się wartości Q i modelu postaci funkcji P, która występuje we wzorze. Algorytm Q-learning można więc opisać za pomocą następującej listy kroków:

1. Obserwacja aktualnego stanu s
2. Wybierz akcję a do wykonania w stanie s
3. Wykonaj akcję a
4. Obserwuj wzmocnienie $R(s)$ i następnie stan s'
5. $Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$

Powyższy algorytm nie definiuje wyboru akcji na postawie wartości Q. Pomimo tego, że do wyboru optymalnej funkcji wystarczy dowolny sposób wyboru akcji zapewniający, że każda akcja zachowuje niezerowe prawdopodobieństwo, w praktyce najlepszym wyborem są rozwiązania faworyzujące akcje o większych wartościach Q, ponieważ pozwalają agentowi poprawiać swoje działania. W naszym przypadku algorytm ten został zaimplementowany oraz zoptymalizowany w bibliotece TensorFlow, dzięki czemu nie ma konieczności ponownej implementacji kodu.

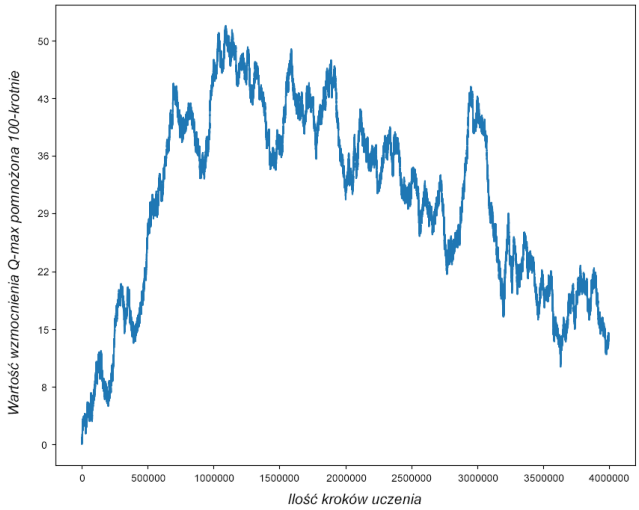
Z przeprowadzonych przez nas badań wynika, że im większą wartość „Q" wprowadzimy do programu tym PacMan lepiej oraz efektywniej będzie w stanie się uczyć. Oczywiście cały proces zależy również od innych wartości, między innymi całkowitej liczby kroków uczenia, w przypadku naszego doświadczenia wynosiła ona 4000000, z założeniem, że PacMan ma dopiero zapisywać postęp uczenia od 10000 kroku. Również nasz skrypt pomija początkowe oczekiwanie 90 sekund na początku gry w celu oszczędzenia całkowitego czasu uczenia. Czas uczenia zależny jest również od możliwości urządzanie na jakim uruchamiany jest projekt. Nasze testy zostały przeprowadzone na komputerze wyposażonym w procesor Intel Core i7-8700, 16GB pamięci ram DDR4, a całość zapisywana była oraz przetwarzana na dysku podpiętym do interfejsu x4 PCIe 3.0 NVM-e. Nasze testy również zostały przeprowadzone dla trzech określonych wartości „Q": 0.99 (dążącym do maksymalnego 1), 0.5 (wartość „średnia') oraz 0.01 (wartość najmniejsza).

Jesteśmy w stanie zaobserwować, że dla wartości „Q" równiej 0.01 dla wcześniej określonej ilości kroków program traci „motywację" do uczenia się (Rysunek 2.3). Oznacza to, że niska wartość nagrody nie jest w stanie odpowiednio stymulować sieci neuronowej do dalszych, lepszych działań i poprawiania swoich osiągnięć. Po osiągnięciu określonej ilości kroków PacMan porusza się, lecz dalej robi to w chaotyczny sposób, nie jest w stanie określić swoich zachowań ani w sensowny sposób unikać przeciwników czy zbierać rozrzuconych po mapie bonusów.



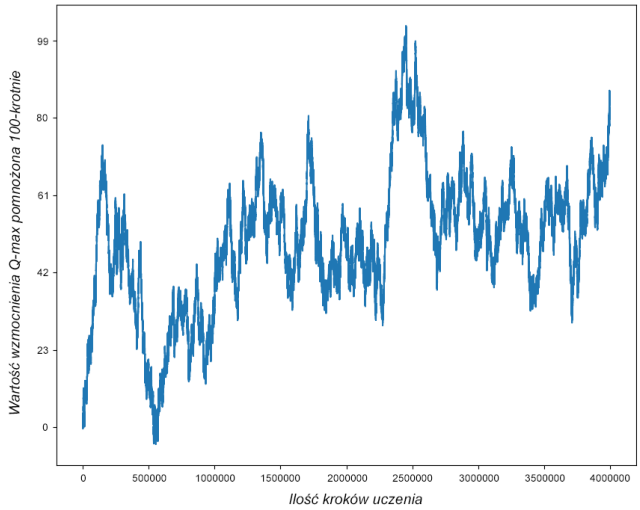
Rysunek 2.3. „Q” równa 0.01

Wzrost zainteresowania uczeniem przejawia wartość „Q” równa 0.5 (Rysunek 2.4). Sieć neuronowa z taką wartością jest odpowiednio pobudzona, aby kontynuować proces uczenia. Również jesteśmy w stanie zaobserwować, że po osiągnięciu najlepszych wyników dla danej zachęty, sieć neuronowa wykorzystywała już zdobytą przez siebie wiedzę, wykorzystując ją do gry, zamiast kontynuować proces uczenia. Idealnie reprezentuje to malejący wykres pod koniec procesu uczenia. Skok w przedziale 3000000 kroków spowodowany jest wystąpieniem nowej zmiennej - PacMan przeszedł do nowej planszy, lecz sieć neuronowa była w stanie określić i zastosować znane już rozwiązania z powodu występowania dalej tych samych przeciwników oraz bonusów.



Rysunek 2.4. „Q” równa 0.5

Ostatnim doświadczeniem było ustawienie wartości „Q” na wartość 0.99 (Rysunek 2.5), czyli naszą wartość maksymalną. Jesteśmy w stanie zaobserwować, że nasza sieć ciągle pobudzana dużą wartością jest w stanie ciągle znajdować nową potrzebę oraz „motywację” do uczenia się. Odpowiednio stymulowana sieć jest w stanie ciągle poszukiwać nowych rozwiązań nawet na znanym jej już środowisku. Na wykresie jesteśmy w stanie zauważyć spadki dla poszczególnych przedziałów. Są one w tym przypadku spowodowane wykorzystywaniem znanych już rozwiązań, lecz gdy zmieni się przynajmniej jedna ze zmiennych, proces uczenia na nowo zostaje uruchomiony.



Rysunek 2.5. „Q” równa 0.99

Chcielibyśmy również zaznaczyć, że ze względu, iż do uczenia zostało wykorzystane prywatne urządzenie, było ono eksploatowane w trakcie procesu uczenia, co również mogło mieć wpływ na ostateczny wynik ze względu na różne zużycie podzespołów i obciążenie ich innymi zadaniami. Całkowity proces uczenia dla 4000000 kroków wynosił w każdym przypadku około 55 godzin, przy czym każda zmiana związana z siecią neuronową wymuszała ponawianie całości uczenia od nowa. Istnieje możliwość uruchomienia biblioteki Tensorflow na kartach graficznych Nvidia GeForce oraz Quadro posiadających rdzenie CUDA, lecz ze względu na brak odpowiedniego sprzętu komputerowego nasz algorytm wykorzystuje uczenie wykorzystujące procesor komputera.

3. Opis programu

Program został napisany w języku Python 3.6 z wykorzystaniem biblioteki Tensorflow oraz OpenAI Gym Atari. Do uruchomienia programu potrzebne są odpowiednio zainstalowane oraz skonfigurowane wyżej wymienione biblioteki wraz z innymi pakietami systemowymi.

Aby ułatwić przygotowanie potrzebnych bibliotek dla systemów opartych na dystrybucji Debian GNU/Linux w tym Ubuntu GNU/Linux został również przygotowany skrypt instalacyjny, który po wywołaniu instaluje wszystkie potrzebne biblioteki oraz pakiety systemowe.

Do edycji kodu źródłowego został wykorzystany program Visual Studio Code wraz z zainstalowanym dodatkiem Microsoft Python.

3.1. Możliwości programu

Program po uruchomieniu automatycznie i samodzielnie uczy się grać w grę MsPacman. Program również zapisuje oraz odczytuje stan gry po wyłączeniu aplikacji.

3.2. Opis programu

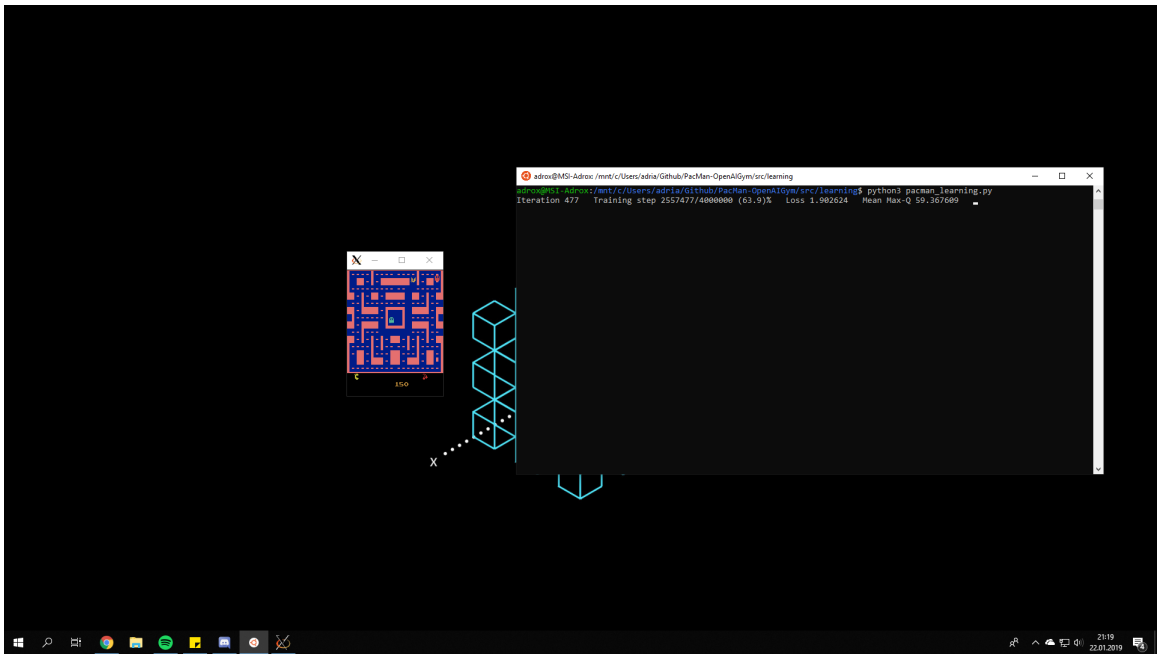
Aplikacja została przygotowana z myślą o wywołaniu jej z poziomu powłoki Bash. W celu uruchomienia aplikacji należy uruchomić konsolę, a następnie przejść do folderu z aplikacją za pomocą komendy `$ cd sciezka/do/pliku`. Gdy znajdujemy się w katalogu z programem, aby uruchomić aplikację, należy użyć polecenia `$ python3 pacman_learning.py`.

Aby uruchomić plik postinstalacyjny mający na celu przygotować wszystkie potrzebne biblioteki należy przejść do katalogu `scripts`, nadać uprawnienia do wykonywania pliku za pomocą polecenia `$ chmod +x post.sh` a następnie wykonać skrypt za pomocą komendy `./post.sh`. Skrypt do działania wymaga podania hasła administratora ze względu na instalację bibliotek z różnych repozytoriów.

Istnieje możliwość uruchomienia aplikacji w środowisku Windows Subsystem for Linux (dla podsystemu Ubuntu) (Rysunek 3.1). W tym celu należy zainstalować dodatkowo aplikację `vcxsrv` emulującą X Server dla systemów Windows oraz po uruchomieniu podsystemu należy wykonać komendę `$ export DISPLAY=localhost:0.0`, która to umożliwia wyexportowanie wyświetlanego obrazu na zewnętrzny system (Windows 10). Należy jednak pamiętać, aby wcześniej uruchomić aplikację `vcxsrv` z następującymi parametrami:

— Multiple windows

— Start no client
— Clipboard, Primary selection (odznaczyć Native opengl)



Rysunek 3.1. Działający, gotowy program na systemie Windows 10

Bibliografia

[1] Machine Learning [ENG] - Wikipedia
https://en.wikipedia.org/wiki/Machine_learning

[2] Dokumentacja Python 3
https://docs.python.org/3/

[3] Dokumentacja Tensorflow
https://www.tensorflow.org/api_docs/

[4] Dokumentacja OpenAI Gym
https://gym.openai.com/

[5] Dokumentacja OpenAI Gym Atari
https://gym.openai.com/envs/#atari

[6] Uczenie się ze wzmocnieniem
http://wazniak.mimuw.edu.pl

[7] Day 22: How to build an AI Game Bot using OpenAI Gym and Universe
https://medium.freecodecamp.org/

[8] Aurélien Géron: Hands-On Machine Learning with Scikit-Learn and TensorFlow (March 2017)