Wykorzystanie głębokich sieci neuronowych do uczenia ze wzmocnieniem w grach komputerowych - raport

Łukasz gołębiowski, Wiktor Wojciechowski

Wrocław, Polska

Abstract

Uczenie ze wzmocnieniem jest jedną z części uczenia maszynowego, według niektórych określanych jako najtrudniejszą. Problem polega na tym, że niemożliwe jest natychmiastowa modyfikacja modelu na podstawie wejścia - potencjalna ocena podjętej akcji jest otrzymywana dopiero po pewnym czasie. Opisywane podejście wykorzystuje głęboką sieć neuronową do uczenia ze wzmocnieniem w środowisku gier komputerowych stanowiących dobrą przestrzeń do rozwoju nowych algorytmów. Raport wprowadza w niektóre nowoczesne techniki używane we wspomnianym zastosowaniu opisując jednocześnie zaimplementowane podejście.

Keywords: Uczenie ze wzmocnieniem, Gry, Głębokie sieci neuronowe

1. Wprowadzenie

Wzrost popularności uczenia ze wzmocnieniem nastąpił po opublikowaniu pracy "Human-level control through deep reinforcement learning" [1] przez firmę DeepMind, później przejętą przez Google. Zaprezentowany w niej został system wykorzystujący głębokie sieci neuronowe do uczenia ze wzmocnieniem w grach Atari. Przyjmując tylko same piksele jako wejście, w wielu grach uzyskiwał on wyniki lepsze od człowieka w takiej samej sytuacji. Rok później DeepMind stworzyło system, który po raz pierwszy pokonał profesjonalnego gracza w Go. Nie było to do tej pory możliwe wykorzystując tradycyjne podejścia ze względu na ogromną przestrzeń rozwiązań występującą w grze. Jednak dzięki wykorzystaniu głębokich sieci neuronowych, możliwości sztucznej inteligencji znacznie się zwiększyły.

Bardzo pomocne w rozwoju kolejnych modeli może okazać się Universe od OpenAI. Jest to darmowa platforma służąca do trenowania i oceny mo-

deli sztucznej inteligencji na różnorodnej kolekcji gier, stron internetowych i aplikacji. Udostępniony jest dostęp do prostych gier, takich jak Pong, czy bardziej skomplikowanych gier Flashowych lub na Atari takich, jakie były wykorzystywane w pracy DeepMind. Jest również możliwość trenowania modeli na dużo bardziej skomplikowanych przykładach - złożonych grach takich jak GTA V czy Cywilizacja VI.

Obie wspomniane firmy twierdzą jednak że nauka obsługi gier nie jest celem a jedynie drogą do niego. Głównym dążeniem jest stworzenie modelu, która będzie w stanie szybko zaznajomić się z nieznanym otoczeniem i nauczyć poruszać się w jego zakresie. Będzie to duży krok w kierunku zbudowania silnej sztucznej inteligencji.

2. Q-learning

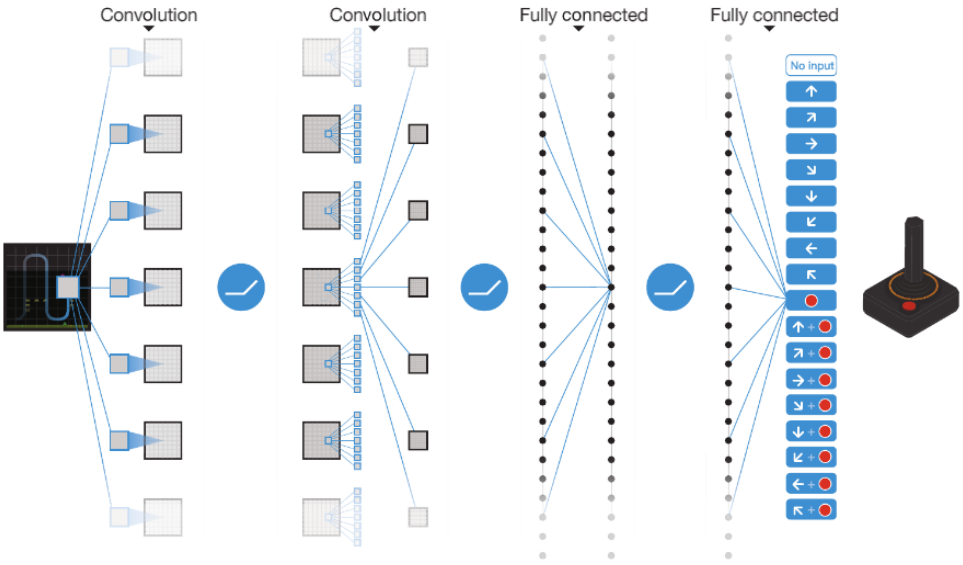
Do uczenie ze wzmocnieniem powszechnie wykorzystywana jest technika zwana *q-learningiem*. Próbuje ona nauczyć się, jaka jest wartość w przebywaniu w danym stanie i podjęcia tam określonej akcji. Funkcja, która określa te wartości nazywana jest funkcją akcji. Aktualizacja wartości funkcji dla danego stanu odbywa się dopiero po poznaniu wyników każdej możliwej akcji - akcje które poprawiły stan są nagradzane, podczas gry te które stan pogorszyły są karane poprzez modyfikację ich wag. Idea tego algorytmu jest oparta o równanie Bellmana, które mówi, że długoterminowa nagroda za daną akcję jest równa natychmiastowej nagrodzie za aktualną akcję połączoną z przewidywaną nagrodą za najlepszą akcję podjętą w kolejnym stanie. Przedstawia to równanie 1.

$$Q(s,a) = r + \gamma(max(Q(s',a')) \tag{1}$$

Oznacza to że, wartość funkcji akcji dla stanu s i akcji a jest reprezentowana przez aktualną nagrodę r plus obniżoną (γ) wartość nagrody za maksymalną wartość funkcji akcji dla następnego stanu s' w którym się znajdziemy. Współczynnik γ określa, jak ważne w ocenie są kolejne wartości funkcji akcji.

Sieć neuronowa pozwala na przechowywanie tych tysięcy jak nie milionów wartości funkcji celu. W analizowanej grze mamy tyle stanów, ile możliwych jest ułożeń pikseli na obrazie, podczas gdy liczba akcji może być różna- dla Ponga wynosi tylko 2 - ruch paletką w górę lub w dół. Wyuczona sieć neuronowa oczywiście nie przechowuje bezpośrednio tych wartości, umożliwia jednak ich otrzymanie poprzez układ wewnętrznych wag.

3. Zaimplementowane podejście



Rysunek 1: Deep Q-Network

Podstawą dla wykorzystywanych modeli jest DQN, Deep Q-Network opisana w pracy DeepMind na temat uczenia w grach Atari [1]. Jej architektura jest wizualizowana na rysunku 1. Można wyróżnić 3 usprawnienia ponad zwykłą sieć MLP która pozwoliły osiągnąć tak dobre wyniki:

- Zastosowanie głębokiej sieci konwolucyjnej
- Implementacji *Odgrywania Doświadczeń (Experience Replay)*, co umożliwia sieci uczenie się na podstawie zapisanych wcześniej doświadczeń.
- Użycie drugiej sieci "celu" do obliczania wartości funkcji akcji.

Jak już zostało wspominane, wejściem do sieci są kolejne ramki z symulacji gry, przedstawione za pomocą tablicy pikseli. Jej rozmiar może być zmienny w zależności od środowiska, ale obraz z Ponga dostępny dzięki OpenAI ma rozmiar 210x160x3. W przetwarzaniu obrazów prym wiodą sieci konwolucyjne. Warstwy konwolucyjne w modelu mają za zadanie wyciągnięcie informacji z obrazu.

Odgrywanie Doświadczeń polega na przechowywaniu doświadczeń modelu i losowe wybieranie z nich paczek do trenowania sieci. Doświadczenia są przechowywane jako krotki <stan, akcja, nagroda, następny stan>. Poprzez wykorzystywanie losowych wspomnień, zapobiegamy uczenia się jedynie tego co jest aktualnie wykonywane w środowisku i umożliwia uczenie na bardziej zróżnicowanych doświadczeniach. Przechowywana jest tylko wcześniej ustalona liczba doświadczeń, które są wraz z upływem kolejnych epok zastępowane nowszymi.

Sieć celu jest wykorzystywana do obliczenia wartości funkcji akcji, która to jest wykorzystywana do obliczenia straty każdej akcji podczas treningu. Innymi słowy służy jedynie do wyznaczenia która z dostępnych akcji jest według sieci optymalna. Niemożliwe jest wykorzystanie jednej sieci, która jednocześnie uaktualnia swoje wagi i generuje nowe wartości. Według cytowanej pracy, ze względu na ciągle zmieniające się wagi, które jednocześnie są pośrednio wykorzystywane do aktualizacji samych siebie (ponieważ wagi sieci wpływają na wynik funkcji akcji) sieć traci stabilność i przewidywane wartości mogą wymknąć się spod kontroli. Aby temu zapobiec, wagi sieci celu są stałe, jedynie co jakiś czas powoli aktualizowane do wag sieci bazowej.

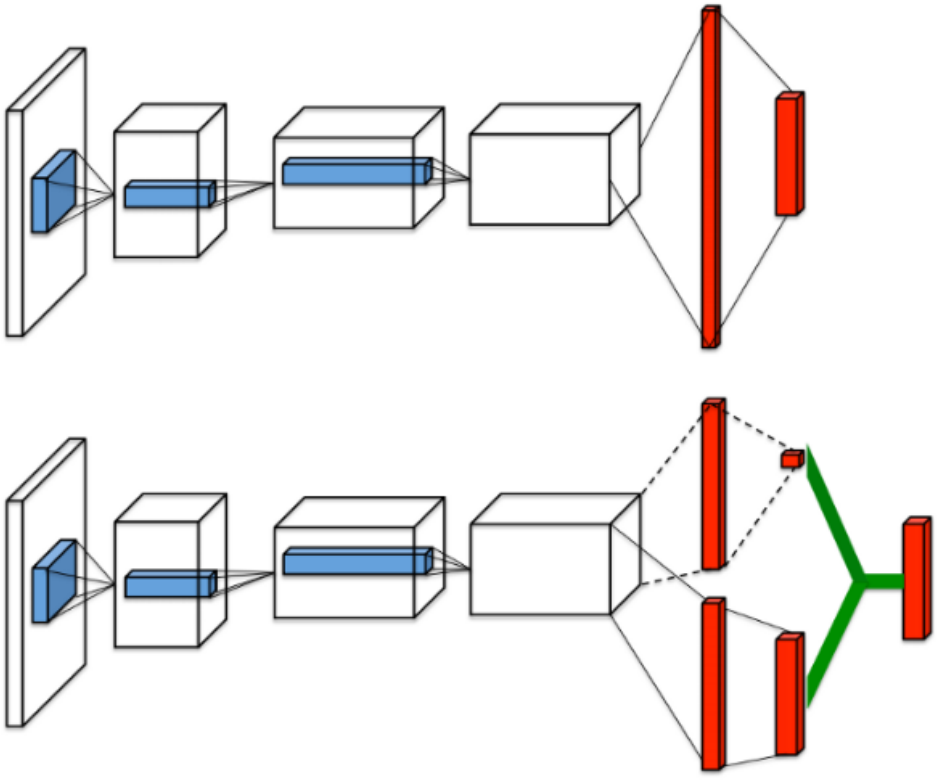
Pracownicy DeepMind zaprezentowali kilka usprawnień do swojego podstawowego modelu. Pierwszym z nich jest określany pojęciem Double DQN [2]. Inspirowany faktem, że w podstawowym modelu często dochodziło do sytuacji, w której wartości funkcji akcji dla poszczególnych akcji w danym stanie były przeszacowane. Nie stanowiłoby to wielkiego problemu, jeśli wszystkie akcje byłyby przeszacowane w jednakowym stopniu, jednak badania potwierdzały, że było inaczej. Częste przyznawanie większych wartości funkcji akcji nieoptymalnym akcjom sprawiało, że model miał problemy z nauczeniem się odpowiedniego postępowania. Aby zapobiec tej sytuacji, autorzy zaproponowali, aby zamiast brać maksymalną wartość funkcji akcji wygenerowaną przez sieć celu(wykorzystywaną w trenowaniu sieci głównej), sieć główna jest wykorzystywana do generowania optymalnej akcji, a sieć celu generuje wartość funkcji akcji dla tej wybranej akcji. Poprzez oddzielenie generacji optymalnej akcji od generowania jej oceny, możemy znacznie przyśpieszyć i uregulować uczenie.

Drugim usprawnieniem jest Dueling DQN[3]. Główną ideą jest rozbicie funkcji akcji na dwie osobne funkcje. Funkcja akcji określa, jak dobre jest podjęcie danej akcji w danym stanie - opisywana jest jako Q(s,a). Pierwszą składową tej funkcji jest funkcja wartości V(s), która określa, jaka jest wartość w przebywaniu w danym stanie. Drugą jest funkcja korzyści A(a),

która określa o ile lepsze jest podjęcie danej akcji w porównaniu do innych. Ostatecznie:

$$Q(s,a) = V(s) + A(a) \tag{2}$$

Zaleta tego podejścia wynika z faktu, że nie w każdym momencie interesuje nas wartość i korzyść. Wykorzystując w przykładzie grę w ponga, w grze istnieją stany, w którym nie trzeba podejmować żadnej decyzji - dopiero odbiliśmy piłeczkę, porusza się ona w kierunku przeciwnika. Jesteśmy w dobrym stanie, nie ma więc sensu myśleć nad podjęciem jakiejkolwiek decyzji. Implementowane jest to przez rozdzielenie sieci na dwa przepływy, który każdy odpowiada za swoją funkcję i są na końcu łączone, co jest widoczne na rysunku 2.



Rysunek 2: Dueling DQN

4. Niektóre szczegóły implementacji

Pierwsze postępy w implementacji modelu były oparte o poradnik [4] Dostępne w nim jest wprowadzenie do Q-learningu jak i do samego Tensorflowa. Początkowe rozdziały skupiają się na implementacji Q-learningu za pomocą tabel, jednak to rozwiązanie może być wykorzystywane tylko w najprostszych środowiskach. Następujące rozszerzenie do płytkich sieci neuronowych nie gwarantuje znacznej poprawy, ale stanowi punkt wejścia do stosowania sieci głębokich.

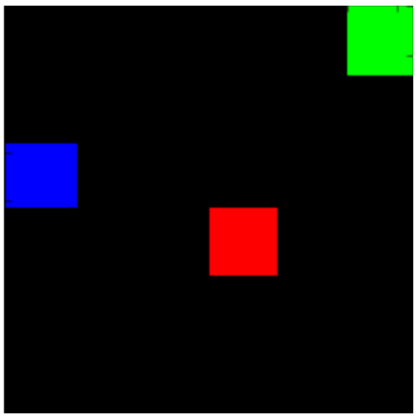
Dużym problemem była instalacja wszystkich zależności modelu - szczególnie Gym wymaga specyficznych warunków, umożliwiając działanie tylko z językiem Python i nie posiadając wsparcia dla systemu Windows. Konieczna była instalacja Linuxa i odpowiednia konfiguracja występujących na nim sterowników. Konieczna była instalacja interfejsu Cuda, który był niezbędny aby uczyć sieć na karcie graficznej - przyspieszając proces. Jest to istotne ze względu na to, że rozpatrywany problem wymaga często kilkudziesięciogodzinnego uczenia, nawet na wydajnych kartach graficznych.

Przy instalacji wykorzystano platformę Anaconda, która umożliwia zarządzanie naukowymi bibliotekami do Pythona. Dodatkowym narzędziem był pip, pełniący podobną funkcję.

Pierwsza implementacja DQN działała na prostym środowisku, aby można było szybko wyuczyć sieć i zauważyć wpływ parametrów na działanie sieci. Kontrolowany w nim był niebieski kwadrat i celem było dotarcie do zielonego kwadratu (nagroda +1) omijając jednocześnie czerwony kwadrat (nagroda - 1), co jest widoczne na rysunku3. Na początku każdego epizodu kwadraty były losowo układane na planszy 5x5.

Mimo że sieć w ciągu uczenia stale polepszała swoje działanie, co było obrazowane przez rosnącą średnią wartość nagrody, to na innych środowiskach, takich jak Pong, sieć nie wykazywała poprawy wraz z rosnącą liczbą epok. Ze względu na to, że istniejąca implementacja nie była napisana zbyt czytelnie, zdecydowano się na przetestowanie innej.

Przystosowano kolejną implementację z internetu[5]. W podanym repozytorium dostępna jest implementacja podstawowej wersji sieci oraz wariantu z rozszerzeniem Dueling. Oparta jest o interfejs Keras działający na bibliotece TensorFlow. Korzysta również z biblioteki Gym od OpenAI, co umożliwia prostą wymianę gry na której uczona jest sieć. Konieczne było jedynie zaktualizowanie projektu na nowe wersje TensorFlowa i Kerasa. Po tych modyfikacjach uruchomiono uczenie na grze Breakout, w której celem jest stero-



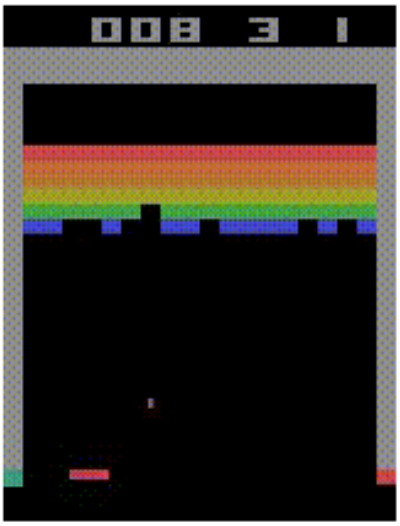
Rysunek 3: Proste środowisko testowe

wanie paletką tak, aby odbijać nadlatującą piłkę w celu niszczenia klocków znajdujących się u góry ekranu. Zobrazowanie widoczne jest na rysunku 4. Niestety, ze względu na ograniczenia czasowe uruchomiono uczenie jedynie na 8 godzin, co nie pozwoliło na weryfikację działania. Artykuły wskazują, że wzrost nagrody występuje dopiero po tym okresie. Jednak inne parametry zachowywały się zgodnie z przewidywaniami - ciągle rosły średnie wartości funkcji akcji sugerując ich modyfikację. Dopiero po pewnym czasie te wartości osiągnęły by taki układ, który pozwala na podejmowanie odpowiednich decyzji.

5. Podsumowanie

Mimo niepowodzenia w kompletnym przetestowaniu modelu, można z projektu wyciągnąć pewne wnioski. Po pierwsze, testowanie DQN na prostym środowisku jest przydatne do optymalizacji parametrów, ale nie może wiarygodnie służyć do przewidywania działania na bardziej skomplikowanym środowisku. Co więcej, aby przetestować działanie sieci w domowych warunkach, konieczna jest implementacja możliwości zapisywania i wczytywania stanu sieci aby móc wznowić uczenie w czasie, gdy dostępne są do tego środki. Innym rozwiązaniem jest posiadanie dedykowanej maszyny.

Dodatkowo, praca z Kerasem okazała się dużo prostsza niż z samym TensorFlowem. Może nie jest to twierdzenie odkrywcze, ale używając jego od początku, można znacznie przyspieszyć zrozumienie modelu ułatwiając modyfikacje.



Rysunek 4: Gra Breakout

Literatura

[1] M. et al., Human-level control through deep reinforcement learning, Nature (2015).

[2] D. S. Hado van Hasselt, Arthur Guez, Deep reinforcement learning with double q-learning (2015).

[3] Z. et al., Dueling network architectures for deep reinforcement learning (2016).

[4] tokb23, Simple reinforcement learning with tensorflow, 2016. Https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0.

[5] tokb23, Dqn implementation in keras + tensorflow + openai gym, 2016. Https://github.com/tokb23/dqn.