Warsztaty z technik uczenia maszynowego - Halite

Wojciech Drochomirecki
Maciej Ejduk
Aleksandra Gajda

Marzec 2022

1 Temat

Celem projektu jest stworzenie algorytmu, który będzie znajdował optymalną strategię ruchów dla gry Halite. Opis gry na Kaggle pod linkiem: Kaggle. Opis Halite na oficjalnej stronie twórcy: Halite. Jest to projekt, który, dzięki dużemu wachlarzowi dotykanych problemów informatycznych, pozwoli na zastosowanie różnych algorytmów i wykorzystanie jak największej ilości informacji zdobytych na przedmiocie. Jednocześnie dzięki swojej postaci pozwala na stworzenie rozwiązania zarówno banalnie prostego, jak i wysoce wysublimowanego, dzięki czemu można dostosować poziom zaawansowania do możliwości w trakcie semestru.

2 Zadania w zespole

Gra udostępnia dosyć rozbudowany interfejs pozwalający na komunikację z jej środowiskiem, wizualizację rezultatów, jak i zbieranie danych z wielu symulacji. Aby efektywnie rozpocząć pracę związaną z implementacją algorytmów ML w rozwiązaniu, trzeba dobrze zapoznać się z tym jaki jest sposób jego funkcjonowania. W tym celu wykonane zostaną różne wersje podstawowego bota prowadzącego rozgrywkę, zaczynając od algorytmu całkowicie losowego, aż do bardziej zaawansowanych algorytmów wykorzystujących proste drzewa decyzyjne. Następnie rozwiązania zostaną

porównane z bazą już istniejących rozwiązań w celu znalezienia najodpowiedniejszego, które później stanie się podstawą do dalszych prac nad projektem.

Następnym krokiem będzie wdrożenie algorytmów ML w konkretnych decyzjach podejmowanych przez bota. Gra udostępnia rozmaite przestrzenie, w których mogą one zostać wykorzystane. Z pośród już rozpoznanych przez nas dziedzin można wyróżnić takie jak sterowanie statkami, interakcje z nieprzyjacielem oraz makrostrategie takie jak tworzenie nowych statków czy gospodarowanie zasobami. Dodatkowo podczas przeprowadzonych eksperymentów planujemy zbierać dane z przykładowych rozgrywek, ilustrujące typowe dla danego problemu i rozwiązania cechy, które pozwolą na końcu projektu wysnuć wnioski na temat poszczególnych algorytmów.

W tym etapie planowane jest przydzielenie każdemu z członków zespołu jednej z dziedzin w której rozwinie algorytm ML spełniający którąś z wymienionych wyżej funkcji, lub inną która okaże się bardziej interesująca dla wyników eksperymentu. Dzięki temu członkowie zespołu otrzymają pewnego rodzaju niezależność, tworząc wspólnie ostateczne rozwiązanie.

Dodatkowo wspólnie będziemy tworzyć dokumentacje projektu związaną z odpowiadającymi nam częściami projektu.

3 Język programowania

W celu wykonania projektu zaplanowane jest użycie języka *Python*. Jest on, tak jak język *R*, narzędziem powszechnie używanym we wszystkich dziedzinach Data Science, a także najpopularniejszym do zastosowań związanych z ML oraz AI. Sposób jego zaprojektowania pozwala na łatwe prototypowanie rozwiązań w celu przeprowadzenia doświadczeń. Dodatkowo posiada wiele bibliotek oraz rozszerzeń zaprojektowanych do wydajnego implementowania algorytmów ML. Ostatecznym argumentem dla użycia tego języka jest jego pełna integracja ze środowiskiem gry Halite oraz fakt że jest on wyróżniony jako polecany przez twórców gry.

4 Narzędzia

Narzędziami, które będą używane w rozwoju aplikacji, będą głównie biblioteki Pythona. Najprawdopodobniej będą to rozwiązania należące do podstawowego kanonu narzędzi wykorzystywanych w zadaniach z zakresu Data Science, takie jak pyTorch, numPy, matplotlib, tensorflow. Dzięki ich powszechnej dostępności i szerokim zastosowaniom korzystać będzie można z ich rozbudowanych dokumentacji oraz tutoriali pomagających w zrozumieniu sposobu działania, jak i metod implementacji konkretnych algorytmów.

Są to też doskonałe narzędzie pomagające w wizualizacji otrzymanych wyników, co pomoże w wyciągnięciu konstruktywnych wniosków z eksperymentu.

Do tworzenia wizualnych reprezentacji naszych rozgrywek, jak i do interakcji ze światem gry, wykorzystane będzie środowisko udostępnione przez twórców gry. Dzięki temu zredukujemy ilość pracy niezwiązanej bezpośrednio z przedmiotem polegającej na samodzielnej implementacji środowiska gry.

5 Przetwarzanie danych

Ze względu na to że zaproponowany projekt ma dosyć nietypowy charakter, wszystkie przetwarzane dane będą wygenerowane na podstawie eksperymentów. Algorytmy, które będziemy stosować, będą w większości należały do rodziny Unsupervised Learning takich jak Reinforced Learning czy Evolutionary Alghoritms. Dodatkowo dzięki popularności konkursu będzie można przetestować utworzone rozwiązania w rywalizacji z botami zaprojektowanymi przez inne zespoły, często profesjonalistów. Pozwoli to na uwypuklenie ich cech w różnych środowiskach.

Dodatkowo wykorzystując dane z poprzednich rozgrywek innych graczy można wykorzystać rozwiązania takie jak regresja liniowa do ekstrakcji pewnych zwycięskich strategii.

6 Wstępne interakcje z danymi

Pierwszym krokiem do uzyskania danych potrzebnych w naszym eksperymencie jest podstawowa integracja ze środowiskiem. W tym celu przed pierwszym checkpointem projektu uwaga skupiona została na zapoznaniu ze środowiskiem gry.

Rozpoczęto od pobrania i skonfigurowania plików gry na maszynie lokalnej wykorzystując instrukcje twórców konkursu. Następnie uruchomiono podstawowy algorytm dostarczony w pakiecie startowym. Zaznajomiono się z API udostępnianym przez środowisko gry, a także zapoznano się z możliwościami interakcji z nią poprzez inspekcję plików gry.

Ostatecznie podjęto się prostych modyfikacji podstawowego algorytmu, modyfikując sposób sterowania statków, a także próbując zapobiec kolizjom. Halite udostępnia własną bibliotekę operacji pozwalających na wykonywanie działań na planszy i uzyskiwanie informacji, co okazało się kluczową częścią projektu, której opanowanie decyduje o zrozumieniu reszty oprogramowania. Dzięki rozegraniu pierwszych rozgrywek można było zapoznać się z plikami replay oraz errorlog które są podstawowymi źródłami informacji o przebiegu rozgrywki. Można było też przetestować różne sposoby produkcji logów, dzięki czemu w przyszłości istniał będzie sposób szybkiego zbierania dan z dużej ilości symulacji.

7 Analiza istniejących rozwiązań

Kolejnym krokiem do przygotowania pod utworzenie sieci neuronowej była analiza istniejących rozwiązań. Najlepiej działające rozwiązania wykorzystywały algorytmy i drzewa decyzyjne w celu wybrania najlepszego ruchu. Często działały na zasadzie maszyny stanowej przypisując każdemu statkowi operację, którą w danej chwili ma wykonywać i potem wybierając jedną z wielu opcji wykonania ruchu.

Rozwiązania wykorzystujące sieci istniały, jednak nie osiągały zadowalających wyników. Popularną metodą przy wykorzystaniu sieci neuronowych było podejście hybrydowe. Program wykonywał kilka początkowych ruchów za pomocą drzew decyzyjnych, w większości przypadków aby zredukować liczbę kolizji statków w okolicy bazy, po czym przechodził do wykorzystania sieci neuronowej. Na koniec rozgrywki

wracał do algorytmu w celu wykonania optymalnych kroków, które powtarzały się w innych rozwiązaniach.

Rozwiązania miały różne pomysły u swoich podstaw. Niektóre oczyszczały okolicę bazy w celu minimalizacji czasu powrotu, inne wyszukiwały miejsca o największym zgromadzeniu surowca w celu maksymalizacji zysków.

Większość rozwiązań nie starała się unikać przeciwnika, a jedynie minimalizować liczbę kolizji w obrębie własnej floty. W analizie rozgrywek najbardziej rozbudowanych rozwiązań zauważyć można, że zderzeń w obrębie własnej floty w ogóle nie ma aż do końcówki rozgrywki, jednak nawet 85% statków niszczona jest w czasie kolizji z flotą przeciwnika. Oznacza to, że takie rozwiązania pomimo tworzenia nawet do 70 statków w czasie 400 rund rozgrywki przeważnie aktywnie wykorzystywały koło 10, pozostałe wysyłając na obszar zajęty przez przeciwnika.

Rozwiązania zbierające surowce w okolicy bazy nie wykorzystywały dodatkowych punktów dostawczych, podczas gdy rozwiązania o szerszym polu działania rozstawiały punkty bardziej losowo, często przekształcając pojazdy, które nie były się w stanie dalej przemieścić w celu redukcji zużycia surowców na nowe, specjalnie przeznaczone do tego statki.

Większość algorytmów, niezależnie od wcześniejszego sposobu działania, w końcowych turach rozgrywki rozbijała własne statki w punktach dostawczych. Strategia taka może wydawać się dziwna, jednak statki trafiające w ten sposób do baz oddawały zebrane surowce, a następnie przy zniszczeniu oddawały część surowców zużytych przy budowie statku. Większość z 50 najlepiej działających algorytmów działa w ten sposób. Różnice polegały najczęściej na określeniu czasu wykonania tej części. Część algorytmów uzależniała to od długości gry, inni od odległości statków od bazy. To podejście wymagało również, by statki tworzone były w miarę możliwości parami, w celu usunięcia wszystkich statków na koniec gry. Rozwiązania tworzące nowe punkty dostaw musiały zmierzyć się z problemem, gdy statki trafiały do różnych punktów dostaw i nie rozbijały się ze względu na to.

Porównując rozwiązania nazywane agresywnymi, ze względu na globalność wykonywanych operacji i zwiększenie ilości kolizji w stosunku do gier bez ich udziału, z rozwiązaniami defensywnymi, charakteryzującymi się wykonywaniem ruchów przede wszystkim w okolicy bazy, te pierwsze osiągały często lepsze wyniki, jednak zdarzały

się sytuacje, w których wszystkie statki rozbijały się o statki gracza defensywnego, uniemożliwiając kontynuację gry. Rozwiązania defensywne wykonywały obliczenia znacząco szybciej niż ich agresywne odpowiedniki.

Ze względu na powyższe informacje, przy tworzeniu sieci neuronowych trzeba będzie zwrócić uwagę na ilość danych wejściowych. Agresywne rozwiązania algorytmiczne często dla każdego statku przetwarzały wszystkie dane z istniejącej planszy, podczas gdy defensywne działały na znacząco mniejszej ilości danych wejściowych.

8 Uczenie maszynowe

Po analizie rozwiązań algorytmicznych nastąpił etap prac nad rozwiązaniem wykorzystującym sieci neuronowe. Podjęto próby rozwinięcia dwóch z możliwych rozwiązań, to jest podejścia ewolucyjnego i próby stworzenia klasyfikatora trenowanego konkretną strategią.

8.1 Podejście ewolucyjne

Aby rozpocząć pracę nad algorytmem ewolucyjnym należało zastanowić się nad sposobem uzyskiwania informacji i narzędziami do tworzenia sieci. Do tworzenia sieci zdecydowano się wykorzystać pakiet *Keras* z biblioteki *Tensorflow*. Wybranym typem sieci było *CNN* o trzech warstwach. Pociągnęło to za sobą kilka decyzji projektowych. Keras najlepiej funkcjonuje dla danych wejściowych składających się z ponad 32 punktów wejściowych. W celu optymalizacji zdecydowano się na pobieranie kwadratu 7×7 wycentrowanym na statku. W ten sposób statek w podejmowaniu decyzji patrzy na odległość 3 pól w każdą stronę. Każde pole zostało scharakteryzowane następującymi parametrami:

- Ilością niezebranego halite
- Obecnością statku, rozróżniając wrogie i przyjazne
- W przypadku istnienia statku na danym polu, ilością halite posiadanego przez dany statek

- Obecnością innej struktury (Bazy, punktu zrzutu, swojego lub wrogiego)

Następnie rozpoczęto generowanie kolejnych pokoleń. Ze względu na rozmiar danych i sieci, pojedynczy plik danych treningowych ważył ponad 30MB. Ograniczono liczbę plików do 250 i liczbę sieci do 10. Każdy plik testowy składał się z ponad 400 testów, więc nawet przy niedużej ilości plików, sieci miały do dyspozycji ponad 1000000 testów ustawionych w losowej kolejności w celu wprowadzenia różnorodności między sieciami. Pojedyncza generacja potrzebowała około doby na ukończenie obliczeń. Początkowe generacje uczyły się stosunkowo szybko, podnosząc średnią dokładność z 0.2 do 0.21, jednak od czwartej generacji tempo wzrostu zmalało do około 0.001. Na koniec dziesiątej generacji trafność sieci wynosiła około 0.22 co może się wydawać optymalne, jednak starając się wnioskować, do uzyskania dokładności na poziomie 0.5, czyli trafienia ponad połowę razy w dobry z 5 możliwych ruchów potrzeba by około 280 dni, na co długość semestru i zasoby obliczeniowe nie pozwalają. Prawdopodobnie wykorzystanie lepszego sprzętu pozwoliłoby na przyspieszenie pracy, jednak nie ma gwarancji, że kolejne generacje nie przetrenowałyby się, co spowodowałoby kompletny zastój. Rozgrywki dla kolejnych generacji generowane były na podstawie rozgrywek najlepszego poprzedniego przodka, który miał między 15% a 35% szansy na wykonanie losowego ruchu zamiast wytrenowanego, z gier w których uzyskał więcej niż ustalona ilość halite. Na początku wartość ta wynosiła 100, co oznacza, że każda sieć, która zebrała minimalną ilość i oddała ją do bazy miała szansę być uznana za wartościową w testach. Z czasem ta wartość się zwiększała.

8.2 Klasyfikator

Ze względu na niską wydajność przygotowania rozwiązania ewolucyjnego zdecydowano się spróbować utworzyć klasyfikator, który, wyuczony większą liczbą rozgrywek utworzonego wcześniej rozwiązania na drzewie decyzyjnym, postara się kopiować poprawne działania i być może jakoś je usprawnić.

Wykorzystano podobny model danych jak poprzednio, zmieniając tylko źródło danych i ich ilość. Wygenerowano ponad 1250 plików testowych, zawierających ponad 3000 testów każdy. Dało to 3750000 testów, na których wytrenowana została sieć. Podjęto kilka prób trenowania sieci. Żadna nie dała dokładności większej niż 0.65.

Wydawać by się mogło, że to całkiem zadowalający wynik. I faktycznie taka sieć znajdowała często najlepsze pola halite dokoła bazy, jednak nie była w stanie oddać ich do punktu zrzutu. W przypadku zapełnienia statku materiałem zastygał on w bezruchu, nie wiedząc co ma zrobić. Próbowano modyfikować dane wejściowe, jednak zawsze dawało to podobny efekt. Aby uzyskać lepsze można by spróbować zmodyfikować dane wejściowe, aby ilość zasobu na statku miała większy wpływ. Można by również przygotować oddzielnie zbiór danych odpowiedzialnych za drogę do bazy, jak i powrót z niej, w celu wyrównania liczby testów. Jednak każda taka zmiana wiąże się potrzebą wykonania nowych gier, które w tym przypadku potrzebują więcej czasu – 1250 gier potrzebowało ponad 2.5 dnia, aby się zakończyć.

9 Podsumowanie

Na podstawie wykonanych działań i eksperymentów można stwierdzić, że przypadku rozgrywek w Halite, sieci neuronowe są zbyt nieopłacalne, żeby wykorzystać pełnię ich potencjału. Użycie klasyfikatora mogło wydawać się nietrafione, jednak klasyfikatory świetnie nadają się do analizy zdjęć, a przekazywane przez nas dane były bardzo podobne do obrazu, dlatego podjęliśmy próbę wykorzystania takiego podejścia.

Aby móc lepiej wykorzystać sieć neuronową, prawdopodobnie potrzebne by były większe ilości zasobów, aby przyspieszyć proces uczenia. Należałoby również poświęcić więcej czasu na analizę poprawności przyjmowanego modelu danych do uczenia sieci w celu uzyskania lepszych wyników dotyczących powrotu statku do bazy.

Ze względu na problemy ze sterowaniem statkiem nie zostały podjęte próby używania sieci do zarządzania produkcją nowych statków. Gdyby udało się poprawnie sterować statkami, kolejnym krokiem byłoby właśnie takie makrozarządzanie.