Rozwiązywanie Sapera z pomocą sztucznej inteligencji

Mateusz Buchajewicz, Dominika Zembrzuska, Kacper Ziółkowski

04.05.2019

1 Wprowadzenie

Celem projektu jest utworzenie trzech algorytmów rozwiązujących popularną grę Saper.

2 Zasady sapera

3 Wykorzystane technologie

Projekt został napisany w języku Python, korzysta z wersji online gry Saper znajdującej się pod adresem *minesweeperonline.com*. Obsługa okna przeglądarki zrealizowana została z pomocą pakietu selenium. Sieć neuronowa została utworzona przy pomocy pakietu tensorflow, a do uzyskiwania rozwiązań przy zastosowaniu zasad napisanych w Prologu użyto pakietu PyDatalog.

4 Opisy algorytmów

4.1 Bot generujący dane

Bot działa w oparciu o 3 główne metody rozwiązujące:

- Rozwiązywanie za pomocą prostej logiki
- Rozwiązywanie za pomocą układów równań
- Strzelanie w losowe pole

Rozwiązywanie za pomocą prostej logiki

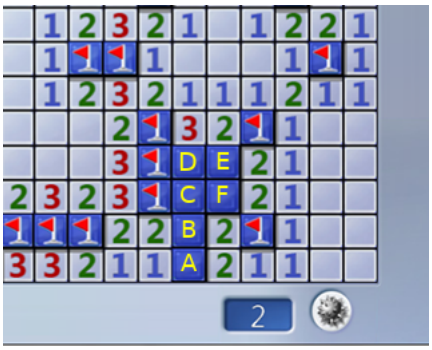
Sprawdza po kolei wszystkie pola planszy, zawierające wartość numeryczną. Jeśli liczba pól sąsiadujących ze sprawdzanym polem, a nieoznaczonych jako "bez miny" (czyli zarówno oznaczonych jako "miny", jak i pustych) jest taka jak wartość pola, to wszystkie te pola są oznaczane jako miny. Natomiast jeżeli liczba oznaczonych min na polach sąsiadujących jest równa wartości sprawdzanego pola, to wszystkie pozostałe pola sąsiadujące zostają oznaczone jako "bez miny"

Rozwiązywanie za pomocą układów równań

Metoda opiera się na ułożeniu układu równań z pól na planszy oraz rozwiązaniu go. Algorytm przekształca układ równań na macierz n x m, gdzie n jest liczbą oznaczonych pól z wyznaczoną wartością, a m jest liczbą różnych pól nieoznaczonych z nimi sąsiadujących.
Dla przykładu, poniższa plansza:

Rysunek 1: Przykładowa plansza sapera

Zostanie przekształcona do postaci:



Rysunek 2: Przykładowa plansza sapera z polami oznaczonymi jako zmienne

Otrzymany układ równań:

$$\begin{cases} A + B = 1 \\ A + B + C = 1 \\ D + E = 1 \\ E + F = 1 \\ A + B + C + F = 1 \end{cases}$$

Co z kolei można zapisać w postaci macierzy:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Macierz główna nie jest macierzą kwadratową (a przynajmniej nie zawsze), więc algorytm nie może określić dokładnych wartości wszystkich zmiennych. Może jednak przekształcić tę macierz za pomocą operacji elementarnych do takiej postaci, żeby mógł odczytać z niej jak najwięcej wartości zmiennych.

$$\left[\begin{array}{cccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 \end{array}\right] \longrightarrow \left[\begin{array}{cccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right]$$

Co jest tożsame z układem równań:

$$\begin{cases} A + B = 1 \\ C = 0 \\ D = 0 \\ E = 1 \\ F = 0 \end{cases}$$

W tym przypadku doskonale widać, że pole E zawiera minę, natomiast pola C, D, F jej nie zawierają. O polach A i B nic nie wiadomo, co wynika z faktu iż macierz główna nie jest kwadratowa.
Warto jednak zauważyć, że operacje podstawowe nie zawsze zwrócą wartość 0 lub 1. W ogólności algorytm sprawdza otrzymane wyniki za pomocą prostych warunków:

$$\begin{cases} \text{if } \forall(a_i \in M_j^W) \mid \sum a_i \mid = \sum \mid a_i \mid = \mid b_j \mid \text{ then } x_j = 1 \\ \text{if } \forall(a_i \in M_j^W) \mid \sum a_i \mid = \sum \mid a_i \mid = 0 \text{ then } x_j = 0 \end{cases}$$

Analogicznie jak wyżej, wszystkie pola z wartością 1 algorytm oznacza jako pola "z miną", a te z wartością 0 jako "bez miny".

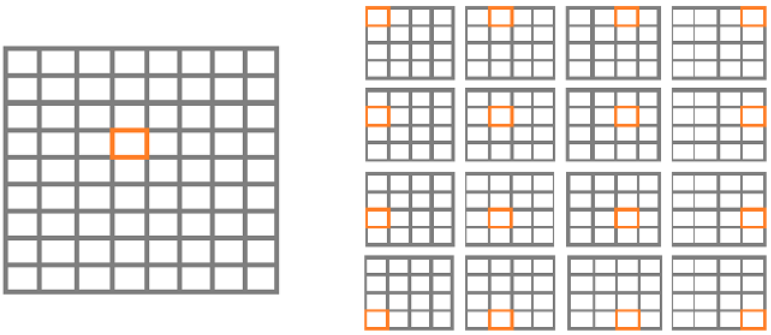
Strzelanie w losowe pole

Ta metoda jest używana wtedy i tylko wtedy, gdy obie powyższe po przejrzeniu całej planszy nie dadzą rady oznaczyć ani jednego pola. Algorytm wybiera losowo jedno z nieoznaczonych pól i oznacza je jako pole "bez miny". Jeśli oznaczy poprawnie gra jest kontynuowana z wykorzystaniem powyższych metod. Jeśli pole nie zostało poprawnie oznaczone, gra jest zakończona porażką.

4.2 Sieć neuronowa

Dane uczące

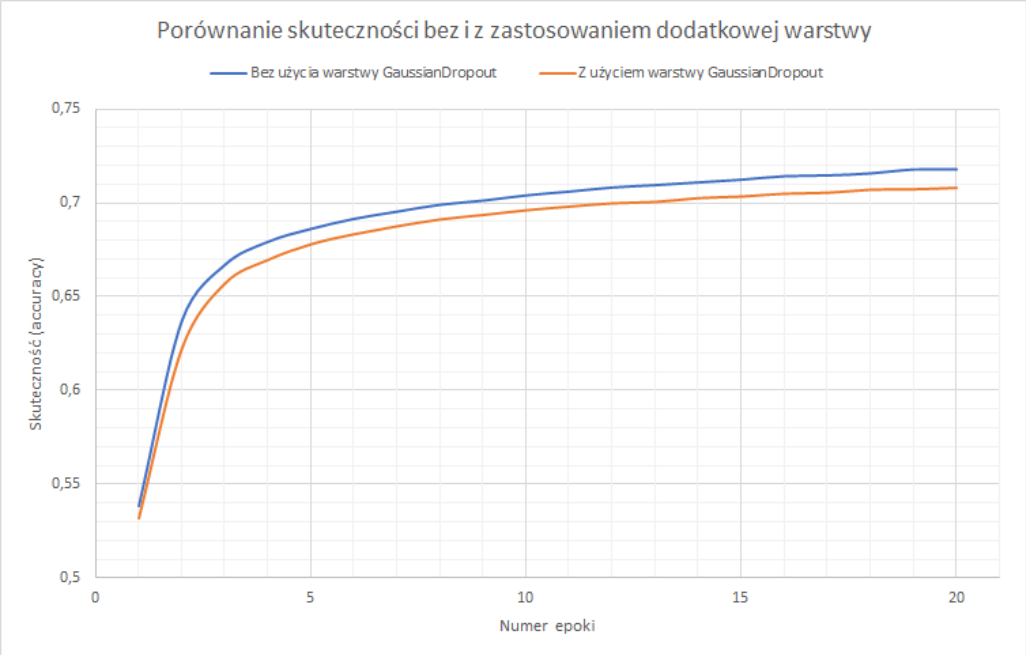
Dane uczące wygenerowane zostały przy pomocy bota. Po wykonaniu obliczeń oznacza on miny w sposób poprawny, błędy występują w przypadku, gdy obecny stan planszy nie pozwala na wyliczenie ich położenia. Pomyłki występują wyłącznie w przypadku oddania strzału, dane po niecelnym strzale nie zostawały uwzględnione. Po wybraniu pola na planszy, tworzono do 16 macierzy rozmiaru 4x4, które uwzględniały położenie pola na każdej z 16 możliwych pozycji. Miało to na celu nauczenie sieci odpowiednich wzorców z uwzględnieniem otoczenia wybranego pola. Rozmiar macierzy 5x5 mogłby być zbyt duży w przypadku małej planszy (10x10), a wymiar 3x3 mógłby nie dostarczać odpowiedniej ilości informacji. Macierze przekształcano na wektory o długości 16, które stanowiły dane do trenowania sieci. Sieć wykorzystano do klasyfikacji. Etykiety stanowiło położenie miny w wektorze (0-15) lub liczba 16 w przypadku braku miny na przekazanym fragmencie planszy. Wygenerowano około 1.000.000 wektorów treningowych, w tym po 500.000 przypadków danych pozytywnych i negatywnych.



Rysunek 3: Sposób generowania danych wejściowych.

Opis sieci

Sieć składa się z czterech warstw gęstych o 512 neuronach i warstwy gęstej wyjściowej o 17 neuronach. Zastosowanie warstwy zapobiegającej przetrenowaniu nieznacznie pogarsza skuteczność sieci przy zastosowaniu danych testowych (ilość 50.000), więc finalnie nie została ona użyta.
Jako funkcji straty użyto funkcji sparse categorical crossentropy, jako funkcji optymalizującej funkcji Adam. Wykresy skuteczności i wartości funkcji straty dla kolejnych epok podczas treningu:



Rysunek 4: Wykresy skuteczności z zastosowaniem warstwy zapobiegajacej przetrenowaniu i bez jej użycia.

Rozwiązywanie

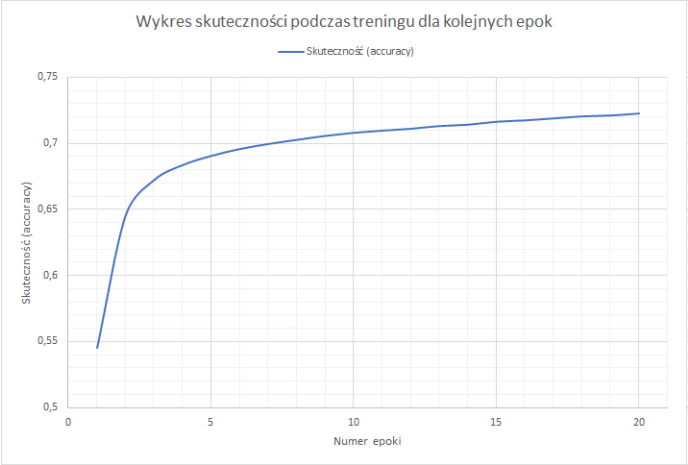
Dla każdego odkrytego pola z wartością numeryczną generowane jest do 16 macierzy 4x4. Korzystając z prawdopodobieństw przynależności do każdej z klas (oprócz ostatniej) zwróconych przez sieć neuronową, uzupełniana jest tablica wielkości planszy, w której zawarte są prawdopodobieństwa wystąpenia miny na danym polu. Po oznaczeniu pola, na którym z największym prawdopodobieństwem znajduje się mina algorytm sprawdza, czy występują jakiekolwiek pola, dla których wszystkie miny zostały już znalezione. W przypadku sukcesu, odsłaniane są pozostałe pola z nimi graniczące.

4.3 Prolog

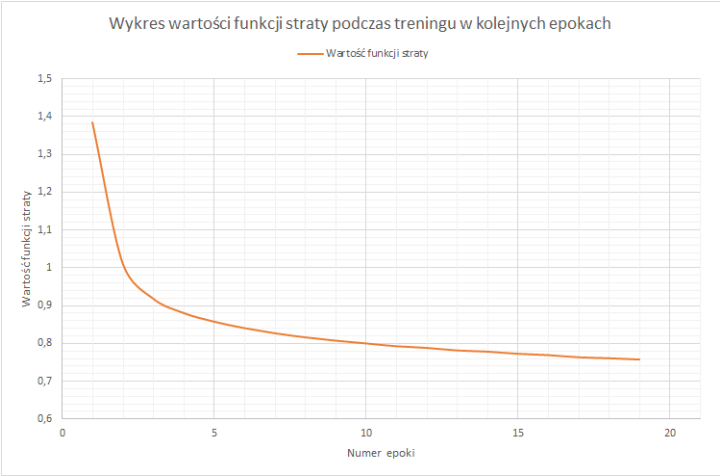
Dodana została jeszcze metoda oparta o proste reguły logiczne napisane w języku logicznym bazującym na prologu (pyDataLog). Metoda iteruje po wszystkich polach sąsiadujących z miejscami, gdzie potencjalnie mogłaby być bomba. Dla każdego pola zlicza ile jest sąsiadujących nieodkrytych pól oraz oznaczonych flagą, i na tej podstawie wnioskuje czy w sąsiadujących nieodkrytych polach jest bomba czy nie.

5 Porównanie metod

Bot osiąga około 85-90% skuteczności na małych planszach (10x10, 10 min) oraz około 10-15% na dużych planszach (30x15, 99 min).



Rysunek 5: Wykres skuteczności w czasie treningu w kolejnych epokach.



Rysunek 6: Wykres wartości funkcji straty podczas treningu w kolejnych epokach.

Algorytm	Mała	Duża
Bot	85-90	10-15
Sieć neuronowa	?	?
Logika / prolog	?	?

6 Wnioski