Uniwersytet im. Adama Mickiewicza w Poznaniu
Wydział Nauk Społecznych
Instytut Psychologii

Gracjan Popiółkowski
nr albumu: 464714

Analiza pierwszych 15 minut meczy w grze League of Legends za pomocą sztucznych sieci neuronowych

Analysis of the first 15 minutes of League of Legends matches using artificial intelligence algorithms

Poznań 2023

Spis treści

Wprowadzenie

W tej pracy zaprezentuje wyniki uczenia sztucznej sieci neuronowej przy różnych parametrach, którą wykorzystam do prognozowania wyniku meczu na podstawie danych z pierwszych 15 minut popularnej gry "League of Legends". Pierwszy kwadrans rozgrywki stanowi istotny moment meczu, dlatego stawiam sobie pytania: Jak dużą precyzją możemy przewidzieć rezultat meczu, bazując na kluczowych statystykach obu drużyn? Czy pierwsze 15 minut są decydujące?

ROZDZIAŁ 1

Prezentacja danych

1.1. Zbiór danych

Korzystam z zestawu danych "League of Legends Diamond Games (First 15 Minutes)", udostępnionego przez użytkownika Bena Fattoriego na platformie Kaggle. Dostęp do tego zbioru został umożliwiony przez twórców gry - Riot Games. Dane zostały zebrane z prawie 50000 meczów graczy rangi diamentowej (jednej z najwyższych) na serwerze "NA1" i zawierają kluczowe statystyki dotyczące przebiegu rozgrywki z pierwszych 15 minut gry.

1.2. Opis danych

Spośród 19 kolumn danych, przed dalszą analizą usunięto 4 z nich:

- `id` (numer danych) - mogłoby to zaburzyć pracę algorytmów ML.
- `matchId` (indywidualny identyfikator meczu) - nie ma wpływu na rozgrywkę.
- `blueDragonKills` (ile smoków zabiła drużyna niebieska)
- `redDragonKills` (ile smoków zabiła drużyna czerwona) - wartość zmiennej w obu drużynach w każdym meczu wynosi 0, więc zmienne nie mają znaczenia.

Dalsza analiza obejmuje 15 zmiennych, które są opisane poniżej:

- `blue_win` - zmienna kategoryzująca, przyjmująca wartości 0 i 1. 1 w przypadku, kiedy drużyna niebieska wygrała i 0, kiedy wygrała drużyna czerwona.

Pozostałe kolumny to zmienne numeryczne:

- `blueGold` - suma monet zdobytych przez drużynę niebieską, składającą się z 5 graczy.
- `redGold` - to samo odnośnie drużyny czerwonej.
- `blueMinionsKilled` - liczba potworów zabitych przez drużynę niebieską.
- `redMinionsKilled` - to samo odnośnie drużyny czerwonej.
- `blueJungleMinionsKilled` - liczba potworów zabitych przez *leśnika* drużyny niebieskiej.
- `redJungleMinionsKilled` - to samo odnośnie drużyny czerwonej.
- `blueAvgLevel` - średni poziom doświadczenia bohaterów graczy z drużyny niebieskiej w 15 minucie gry.
- `redAvgLevel` - to samo odnośnie drużyny czerwonej.
- `blueHeraldKills` - liczba potworów o nazwie "Herald" zabitych przez drużynę niebieską.

- `redHeraldKills` - to samo odnośnie drużyny czerwonej.
- `blueTowersDestroyed` - liczba zniszczonych wież drużyny niebieskiej.
- `redTowersDestroyed` - to samo odnośnie drużyny czerwonej.
- `blueChampKills` - ile razy łącznie drużynie niebieskiej udało się zabić bohatera z drużyny przeciwnej.
- `redChampKills` - to samo odnośnie drużyny czerwonej.

Oto jak przedstawia się statystyka danych dla drużyny niebieskiej:

Variable	Mean	Std	Min	Mdn	Max
blueGold	26349.16	2777.30	15930.00	26167.00	39769.00
blueMinionsKilled	335.86	32.67	120.00	337.00	455.00
blueJungleMinionsKilled	83.14	14.79	0.00	83.00	164.00
blueAvgLevel	9.17	0.41	5.40	9.20	10.80
blueChampKills	11.01	4.53	0.00	11.00	50.00
blueHeraldKills	1.32	0.96	0.00	1.00	4.00
blueTowersDestroyed	0.79	1.05	0.00	1.00	11.00

Tabela 1.1: Mean oznacza średnią, std - odchylenie standardowe, min - wartość najmniejsza, mdn - medianę, a max - wartość największa.

1.3. Przygotowanie danych

Tak jak wyżej pisałem 4 kolumny danych zostały usunięte. Zmienną kategoryzującą jest kolumna `blue_win`. Dodatkowo wykonałem zabieg standaryzacji danych na wszystkich 14 numerycznych cechach, czyli przekształciłem je w taki sposób, aby miały średnią wartość równą 0 i odchylenie standardowe równe 1. Może to poprawić efektywność użytych przeze mnie algorytmów uczenia maszynowego.

ROZDZIAŁ 2

Sztuczna sieć neuronowa

`Sztuczna sieć neuronowa` to model inspirowany strukturą mózgu, zbudowany z jednostek nazywanych neuronami, połączonych ze sobą. Neurony są zorganizowane w warstwy, a każde połączenie między nimi posiada wagę. Proces uczenia polega na dostosowywaniu wag, aby model skuteczniej przewidywał wyniki. Parametry, na które możemy wpływać w sztucznych sieciach neuronowych, obejmują:

1. `Wielkość zbioru treningowego` - określa proporcję zbioru treningowego do testowego, definiując, jaką część danych algorytm przeznaczy na uczenie sieci, a jaką na testowanie jej skuteczności. W mojej analizie wielkość zbioru treningowego jest ustawiona na 80% (38920 obserwacji), więc testowego na 20% (9731 obserwacji).

2. `Liczba warstw i neuronów` - decyduje o strukturze sieci neuronowej. Więcej warstw i neuronów może umożliwić modelowi lepsze zrozumienie złożonych zależności, lecz jednocześnie zwiększa ryzyko nadmiernego dopasowania (overfitting).

3. `Funkcje aktywacji` - określają, w jaki sposób aktywacje są przekazywane przez neurony. Funkcje aktywacji wprowadzają nieliniowość do modelu, co umożliwia mu modelowanie bardziej skomplikowanych relacji w danych.

4. `Funkcja straty` - parametr określający, jak model mierzy swoją skuteczność podczas treningu. Funkcja straty jest minimalizowana podczas treningu, a jej wybór zależy od rodzaju problemu (np. klasyfikacja binarna, wieloklasowa, regresja).

5. `Optymalizator` - algorytm używany do aktualizacji wag w trakcie treningu. Optymalizatory regulują, jak szybko model się uczy i jak dobrze dostosowuje się do danych treningowych.

6. `Wskaźniki metryki` - metryki do oceny skuteczności modelu. Na przykład, accuracy mierzy procent poprawnie sklasyfikowanych przykładów.

7. `Rozmiar wsadu (batch size)` - parametr określający liczbę przykładów treningowych używanych do jednej aktualizacji wag. Odpowiedni dobór rozmiaru wsadu wpływa na szybkość treningu i zużycie pamięci.

8. `Liczba epok` - parametr określający liczbę pełnych przejść przez zbiór treningowy. Zbyt duża liczba epok może prowadzić do przeuczenia.

9. `Współczynnik uczenia` - parametr określający, jak szybko model się uczy. Wartość zbyt wysoka może prowadzić do przeuczenia, a zbyt niska do zbyt wolnego postępu w treningu.

10. `Dropout (warstwy Dropout)` - warstwy Dropout pomagają w zapobieganiu nadmiernemu dopasowaniu poprzez losowe wyłączanie pewnych neuronów podczas treningu. To narzędzie regularyzacyjne zmniejsza ryzyko nadmiernego dopasowania.

W dalszej części tekstu przedstawię porównanie wyników przewidywania sieci neuronowej uzyskanych przy różnych zestawach parametrów.

ROZDZIAŁ 3

Analiza wyników

3.1. Najprostszy model sieci neuronowej

`Model 1` - to będzie najprostszy model sieci, który posłuży nam jako punkt odniesienia. Jego specyfikacja:

- `Liczba Warstw` - zastosowałem minimalną liczbę warstw, co oznacza warstwę wejściową, jedną warstwę ukrytą i warstwę wyjściową.
- `Liczba Neuronów` - niewielka liczba neuronów: 14 neuronów warstwy wejściowej, 8 pierwszej ukrytej i 1 neuron drugiej warstwy ukrytej.
- `Funkcje Aktywacji` - zastosowałem funkcję aktywacji ReLU w warstwie ukrytej.
- `Funkcja Straty` - użyłem binarnej klasyfikacji; Binary Crossentropy jest często stosowaną funkcją straty.
- `Optymalizator` - popularny optymalizator Adam.
- `Liczba Epok` - wybrałem stałą liczbę 10 epok, bez funkcji szybszego zatrzymania nauki sieci.
- `Batch Size` - skorzystałem z wartości domyślnej.

Modele będę porównywał głównie według wielkości `Accuracy`, czyli zmiennej oznaczającej procentowo, jaką skuteczność uzyskał model w ocenianiu zwycięzcy meczu na zbiorze testowym. W naszym najprostszym modelu:

- `Accuracy` = 0.7881

Czyli skuteczność przewidywania zwycięskiej drużyny na podstawie pierwszych 15 minut meczu wynosi 78.81%.

3.2. Porównanie 10 modeli ze zmianą jednego parametru

Drugą część mojej analizy postanowiłem poprowadzić w taki sposób, że skonstruuje 9 modeli, które będą się różnić od podstawowej wersji tylko jednym parametrem.

3.2.1. Model 2 - więcej neuronów warstwy ukrytej

W modelu 2 zdecydowałem się zwiększyć liczbę neuronów w warstwie ukrytej. Wybór ten ma na celu zwiększenie zdolności sieci do wykrywania bardziej skomplikowanych wzorców w danych treningowych. Użyto dwóch warstw - pierwsza z 32 neuronami, a druga z 16. Ostateczna warstwa, zawierająca jeden neuron z funkcją aktywacji sigmoid, służy do klasyfikacji binarnej. Model był uczony przez 10 epok. Skuteczność tego modelu wyniosła:

- `Accuracy` = 0.7901

3.2.2. Model 3 - mniej neuronów warstwy ukrytej

W przypadku modelu 3 zastosowałem odwrotne podejście, redukując liczbę neuronów w warstwie ukrytej. Mniejsza liczba neuronów może pomóc w uniknięciu przeuczenia, zwłaszcza w przypadku ograniczonej ilości danych treningowych. Warstwy modelu to kolejno 8, 4 i 1 neuron z funkcją aktywacji sigmoid. Również ten model był uczony przez 10 epok. Skuteczność tego modelu wyniosła:

- `Accuracy` = 0.7872

3.2.3. Model 4 - dodatkowa warstwa ukryta

W modelu 4 wprowadziłem dodatkową warstwę ukrytą w celu zwiększenia pojemności modelu. Dodana warstwa ma 8 neuronów i korzysta z funkcji aktywacji relu. Model składa się z trzech warstw: dwie warstwy ukryte i jedna warstwa wyjściowa z funkcją aktywacji sigmoid. Trenowałem go przez 10 epok. Skuteczność tego modelu wyniosła:

- `Accuracy` = 0.7885

3.2.4. Model 5 - inny współczynnik uczenia

W przypadku modelu 5 zmieniłem współczynnik uczenia algorytmu optymalizacyjnego na 0.001. Współczynnik uczenia reguluje kroki, jakie algorytm podejmuje w celu minimalizacji funkcji kosztu. Niższy współczynnik może skutkować bardziej dokładnymi wagami, ale może również wymagać więcej epok treningowych. Model składa się z jednej warstwy ukrytej (8 neuronów) i jednej warstwy wyjściowej z funkcją aktywacji sigmoid. Skuteczność tego modelu wyniosła:

- `Accuracy` = 0.7874

3.2.5. Model 6 - inna funkcja aktywacji

W modelu 6 zastosowałem inną funkcję aktywacji w warstwie ukrytej. Zamiast relu, użyłem sigmoid w warstwie z 8 neuronami. Funkcje aktywacji decydują, jak sygnały są przekazywane między neuronami. Sigmoid jest często używany w warstwie wyjściowej przy klasyfikacji binarnej, ale jego zastosowanie w warstwie ukrytej może wprowadzić subtelne różnice w uczeniu. Skuteczność tego modelu wyniosła:

- `Accuracy` = 0.7889

3.2.6. Model 7 - inny batch size

Model 7 różni się od poprzednich poprzez zastosowanie innego rozmiaru wsadu (batch size) podczas treningu. Batch size to liczba przykładów treningowych używanych do jednej aktualizacji wag. Zmiana tego parametru może wpłynąć na stabilność procesu uczenia. W tym przypadku użyłem batch size równego 64, a model składa się z jednej warstwy ukrytej (8 neuronów) i jednej warstwy wyjściowej z funkcją aktywacji sigmoid. Skuteczność tego modelu wyniosła:

- `Accuracy` = 0.7867

3.2.7. Model 8 - inna liczba epok

W modelu 8 zmieniłem liczbę epok treningowych na 20. Liczba epok określa, ile razy model przejdzie przez cały zbiór treningowy. Zwiększenie liczby epok może prowadzić do dalszego dopasowania modelu do danych treningowych, ale jednocześnie zwiększa ryzyko przeuczenia. Model ten składa się z jednej warstwy ukrytej (8 neuronów) i jednej warstwy wyjściowej z funkcją aktywacji sigmoid. Skuteczność tego modelu wyniosła:

- `Accuracy` = 0.7883

3.2.8. Model 9 - warstwy dropout

Model 9 używa warstw dropout, które pomagają w zapobieganiu nadmiernemu dopasowaniu poprzez losowe wyłączanie pewnych neuronów podczas treningu. W tym przypadku zastosowałem warstwę dropout z prawdopodobieństwem 0.2. Model składa się z dwóch warstw ukrytych (16 i 8 neuronów) oraz warstwy wyjściowej z funkcją aktywacji sigmoid. Skuteczność tego modelu wyniosła:

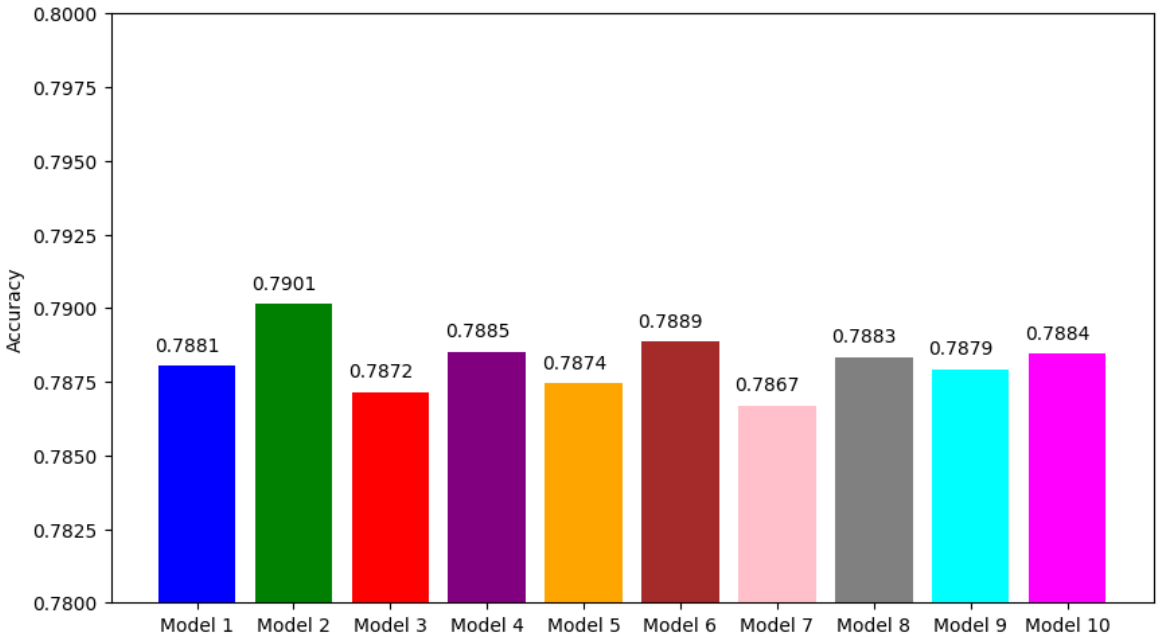
- `Accuracy` = 0.7879

3.2.9. Model 10 - inna funkcja straty

Ostatni model, numer 10, różni się funkcją straty. Zamiast binary_crossentropy, użyto mean_squared_error. Funkcja straty określa, jak skutecznie model jest w stanie oszacować, jak bardzo jego przewidywania różnią się od rzeczywistych wartości. Ten model składa się z jednej warstwy ukrytej (8 neuronów) i jednej warstwy wyjściowej z funkcją aktywacji sigmoid. Skuteczność tego modelu wyniosła:

- `Accuracy` = 0.7884

Oto porównanie na wykresie słupkowym trafności moich modelów:



Rysunek 3.1: Wykres słupkowy porównujący 10 naszych modeli. Model 1 był naszym modelem podstawowym, a modele 2 do 10 różniły się tylko jednym parametrem w stosunku do modelu 1.

Analiza wykresu wskazuje, że największą skuteczność osiągnął model 2, w którym zwiększono liczbę neuronów w warstwie ukrytej. Z kolei najniższy wynik uzyskano dla modelu 7, w którym zmieniono rozmiar wsadu (*batch size*). Również modele 3, 5 i 9 osiągnęły gorszy wynik niż model podstawowy oznaczony numerem 1.

3.3. Trzy bardziej zaawansowane modele

W trzeciej części mojej analizy porównuję trzy modele, które różnią się od mojego najprostszego modelu o więcej niż jeden parametr.

3.3.1. Model 11

Model 11 to sieć z dwiema warstwami ukrytymi o 32 i 16 neuronach, odpowiednio. Wykorzystuje on funkcję aktywacji `relu` w warstwach ukrytych i `sigmoid` w warstwie wyjściowej, co jest powszechnie stosowane w problemach klasyfikacji binarnej. Kompilacja modelu wykorzystuje `binary_crossentropy` jako funkcję straty. W celu zabezpieczenia przed przeuczeniem zastosowano wczesne zatrzymywanie (*EarlyStopping*) z monitorem `monitor='val_loss'`, cierpliwością `patience=3` i przywracaniem najlepszych wag `restore_best_weights=True`. Model ten był ustawiony na 50 epok, jednak zatrzymał się po 8 epokach. Skuteczność tego modelu wyniosła:

- `Accuracy` = 0.7902

3.3.2. Model 12

Model 12 jest rozwinięciem poprzedniego, zwiększając liczbę neuronów do 64 w pierwszej warstwie ukrytej i 32 w drugiej. To zwiększa pojemność modelu, umożliwiając mu bardziej skomplikowane nauki na danych treningowych. Zastosowano te same funkcje aktywacji, funkcję straty i optymalizator, co w Modelu 11. Również tutaj użyto wczesnego zatrzymywania (*EarlyStopping*), a model zatrzymał się już przy 7 epoce. Skuteczność tego modelu wyniosła:

- `Accuracy` = 0.7900

3.3.3. Model 13

Model 13 to sieć neuronowa z dwiema warstwami ukrytymi o 16 i 8 neuronach. Model ten ma mniejszą liczbę neuronów niż poprzednie dwa modele, co może pomóc w uniknięciu przeuczenia, zwłaszcza w przypadku ograniczonej ilości danych treningowych. Funkcje aktywacji, funkcja straty i optymalizator pozostają takie same jak w poprzednich modelach. Trenowano go przez 20 epok, bez funkcji *EarlyStopping*. Skuteczność tego modelu wyniosła:

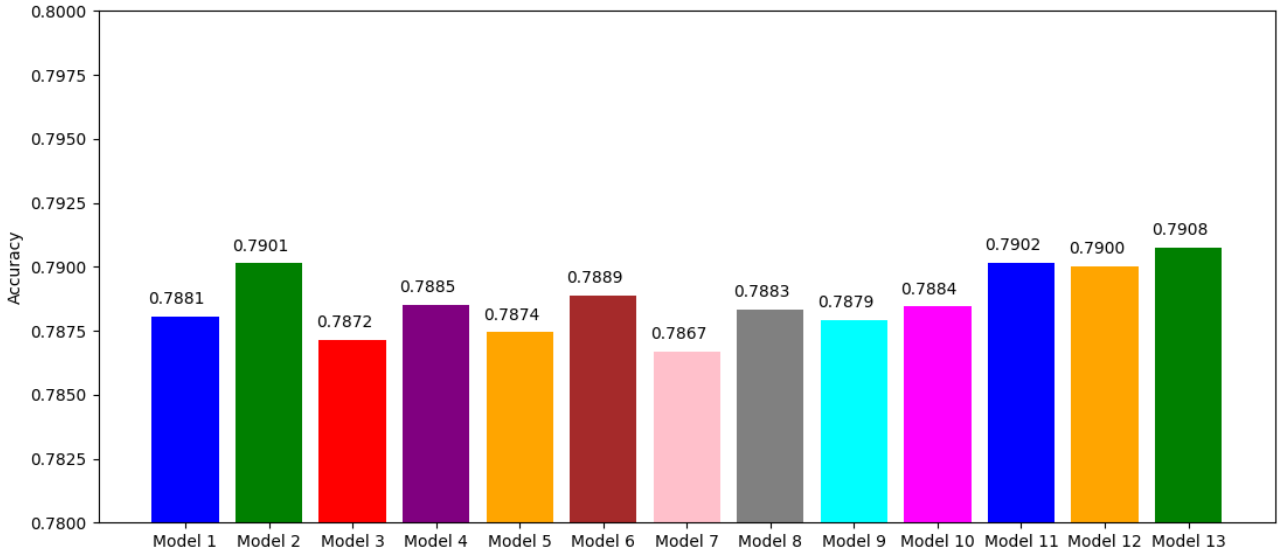
- `Accuracy` = 0.7908

Podsumowanie

Reasumując, w mojej pracy użyłem różnych modeli sieci neuronowych do przewidywania wyników meczów w grze "League of Legends", wykorzystując dane z pierwszych 15 minut gry. Najlepszy wynik, jaki udało mi się osiągnąć, został uzyskany w przypadku modelu 13, z osiągniętą skutecznością:

- `Accuracy` = 0.7908

Powyżej przedstawiam porównanie wszystkich 13 modeli:



Rysunek 3.2: Wykres słupkowy porównujący wszystkie 13 modeli opisane w tym raporcie. Model 1 był naszym modelem podstawowym, 2 do 10 to były modele różniące się tylko jednym parametrem w stosunku do modelu 1, a 11, 12 i 13 to były najbardziej zaawansowane modele.

Uważam, że sieć neuronowa poradziła sobie dobrze, osiągając satysfakcjonujące wyniki predykcji. Niemniej jednak, przy żadnych ustawieniach nie udało mi się przekroczyć 80%, co sugeruje, że na podstawie pierwszych 15 minut meczu trudno jest przewidzieć wynik z dokładnością większą niż 80%. Być może przekroczenie tej granicy wymagałoby dodania kolejnych cech do zbioru danych. Jednakże 14 cech, które zostały uwzględnione, to najważniejsze parametry rozgrywki, na które gracz może wpłynąć w pierwszych 15 minutach gry. Warto zauważyć, że cały mecz w grze "League of Legends" trwa średnio od 20 do 50 minut, więc osiągnięta skuteczność na poziomie 80% jest już wysoka.

Bibliografia

- https://pl.wikipedia.org/wiki/Sie%C4%87_neuronowa
- https://www.statsoft.pl/textbook/stneunet.html#artificial