Szkoła Główna Gospodarstwa Wiejskiego
w Warszawie

Wydział Zastosowań Informatyki i Matematyki

Maciej Pleban
196195

Aplikacja do predykcji wyników meczów tenisowych w oparciu o dane historyczne z wykorzystaniem sieci neuronowych i języka programowania Python

Application for predicting of tennis matches results based on the historical data with the use of neural networks and the Python programming language

Praca dyplomowa licencjacka
na kierunku – Informatyka i Ekonometria

Praca wykonana pod kierunkiem
dr Marcina Ziółkowskiego
Instytut Informatyki Technicznej
Katedra Systemów Informatycznych

Warszawa, 2021

Oświadczenie promotora pracy

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia warunki do przedstawienia tej pracy w postępowaniu o nadanie tytułu zawodowego.

Data ..... Podpis promotora pracy.....

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej, w tym odpowiedzialności karnej za złożenie fałszywego oświadczenia, oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami prawa, w szczególności ustawą z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. Nr 90 poz. 631 z późn. zm.).

Oświadczam, że przedstawiona praca nie była wcześniej podstawą żadnej procedury związanej z nadaniem dyplomu lub uzyskaniem tytułu zawodowego.

Oświadczam, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną. Przyjmuję do wiadomości, że praca dyplomowa poddana zostanie procedurze antyplagiatowej.

Data ..... Podpis autora pracy.....

Streszczenie

Aplikacja do predykcji wyników meczów tenisowych w oparciu o dane historyczne z wykorzystaniem sieci neuronowych i języka programowania Python

Celem niniejszej pracy było stworzenie aplikacji służącej do predykcji wyników meczów tenisowych w oparciu o dane historyczne z wykorzystaniem sieci neuronowych i języka programowania Python. Praca została podzielona na trzy części. W pierwszej z nich poruszone zostały zagadnienia teoretyczne związane z typami uczenia maszynowego oraz sieciami neuronowymi i ich istotą. Przedstawiona została także historia rozwoju uczenia maszynowego. W drugiej części pracy przeanalizowany został zbiór danych wejściowych oraz przedstawiono metody radzenia sobie z brakującymi danymi i reprezentację danych, a także wprowadzono odpowiednie zmienne. Przybliżono także budowę sieci neuronowych, sposób ich działania, a także związane z tym problemy. W ostatniej części zaprezentowana została stworzona aplikacja, narzędzia użyte w trakcie jej tworzenia, przybliżono również charakterystykę implementacji, a także analizę otrzymanych wyników.

Słowa kluczowe – sieci neuronowe, ReLu, Keras, sklearn, Python, predykcja, uczenie maszynowe, Tensorflow

Summary

Application for predicting of tennis matches results based on the historical data with the use of neural networks and the Python programming language

The aim of this study was to create an application for predicting of tennis matches results based on historical data with the use of neural networks and the Python programming language. The thesis consists of three parts. The first part deals with theoretical issues related to the types of machine learning, neural networks and their essence. The history of the development of machine learning was also presented in it. In the second part, a set of input data was analysed, methods of dealing with missing data and data representation were presented as well as appropriate variables were introduced. The structure of neural networks, the way they work and the problems related to them were also discussed. In the last part, the developed application was presented together with the tools used during writing the application, characteristics of the implementation as well as the analysis of the obtained results.

Keywords – neural networks, ReLu, Keras, sklearn, Python, prediction, machine learning, Tensorflow

Spis treści

Wstęp

Problem prognozowania zjawisk jest poruszany od zamierzchłych czasów. Starożytne cywilizacje były w stanie przewidzieć pogodę na podstawie różnych przesłanek takich jak: rodzaje chmur na niebie, zmiany temperatury czy obserwację zachowania zwierząt. Na podstawie takich danych możliwe jest, w pewnym zakresie, przewidzenie jej w dniu następnym. Predykcja przyszłości nie ogranicza się jednak tylko do prognozowania pogody. Znajduje ona zastosowanie również w innych dziedzinach życia. Dla przykładu Warren Buffett (dyrektor generalny Berkshire Hathaway) szacuje cenę akcji na podstawie analizy wyników finansowych spółek — stara się ustalić zależność między rynkową ceną akcji a wynikami przedsiębiorstw [1]. Jednakże nie ma sytuacji idealnych, w których można wszystko dokładnie przewidzieć, ponieważ może wydarzyć się coś nieprzewidywalnego, co normalnie nie byłoby brane pod uwagę w procesie prognozy. Kiedy w 1929 roku maklerzy prognozowali ceny akcji notowanych na Nowojorskiej Giełdzie Papierów Wartościowych, nie przewidywano, że ich wartości gwałtownie spadną, co zapoczątkuje wielki kryzys gospodarczy. Znacznie stabilniejszym środowiskiem, w którym można zastosować predykcję, są zakłady bukmacherskie.

Zakłady na wybrane wydarzenia sportowe są praktykowane już od ponad 100 lat. Najchętniej wybieranymi sportami są piłka nożna, koszykówka czy tenis ziemny, na którym to sporcie oparta jest niniejsza praca. Jest on najpopularniejszym sportem indywidualnym na świecie, który przyciąga wielu fanów i sponsorów. Jego początki sięgają VIII wieku, ale forma, w jakiej znamy go obecnie, obowiązuje od roku 1859 [2]. Do grona najznamienitszych graczy można zaliczyć obecnie Rogera Federera, Rafaela Nadala czy Novaka Đjokovića.

Celem niniejszej pracy jest stworzenie aplikacji, która będzie prognozowała wynik hipotetycznego spotkania między dwoma zawodnikami na wybranej powierzchni. Aby zagłębić się w ten temat, należy skorzystać z dorobku dziedziny sztucznej inteligencji.

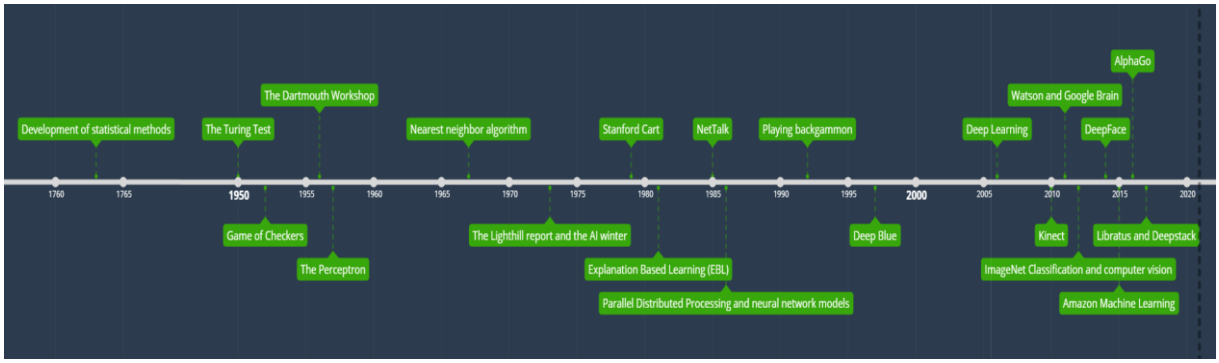
W tego typu prognozowaniu często używa się modeli ekonometrycznych takich jak: Regresja Logistyczna, Maszyny Wektorów Nośnych (SVC), Lasy Losowe czy Sieci Neuronowe, które są wykorzystywane w tej pracy. Implementacja wybranych rozwiązań została przeprowadzona z użyciem języka programowania Python.

Niniejsza praca składa się z trzech rozdziałów. W pierwszym rozdziale przedstawione zostały zagadnienia związane z uczeniem maszynowym i sieciami neuronowymi, a także ich historia oraz koncepcja służąca wprowadzeniu do dalszej części pracy. W kolejnym rozdziale przedstawiony został zbiór danych, metody zastosowane w celu radzenia sobie z brakującymi danymi, dobór odpowiednich wskaźników, jak i sposób obliczania nowych. Na szczególną uwagę zasługuje wskaźnik Elo, zainspirowany grą w szachy. Została także przedstawiona struktura użytej sieci neuronowej. Ostatni rozdział został poświęcony opisowi wykorzystanych narzędzi, sposobowi implementacji, jak i analizie otrzymanych wyników. W części końcowej pracy znalazło się odniesienie do celów, problemów badawczych i hipotez, umieszczone w podsumowaniu.

1 Zagadnienia wstępne

Poniższy rozdział przedstawia zagadnienia teoretyczne związane z uczeniem maszynowym i sieciami neuronowymi. Na początku przedstawiona jest historia powstania i rozwoju sztucznej inteligencji. Następnie przybliżone są typy uczenia maszynowego razem z przykładami zastosowań. Na koniec pokazana jest istota sieci neuronowych.

1.1 Historia uczenia maszynowego



Rysunek 1. Oś czasu przedstawiająca najważniejsze wydarzenia uczenia maszynowego

Opracowanie własne na podstawie [3]

Początek uczenia maszynowego przypisuje się wydaniu *Essay Towards Solving a Problem in the Doctrine of Chance* [4] w roku 1763, gdzie Thomas Bayes, angielski matematyk, uważany za twórcę statystyki, sformułował twierdzenie o prawdopodobieństwie, dzisiaj znane jako Twierdzenie Bayesa, które jest główną ideą w niektórych podejściach nauczania maszynowego.

Tw. Niech $\{H_i\}_{i=1\ldots,n}$ będzie rozbiciem przestrzeni Ω na zdarzenia o dodatnich prawdopodobieństwach i $P(A) > 0$. Wówczas dla dowolnego $j \in \{1 \ldots, n\}$ mamy ([5])

$$P(H_j|A) = \frac{P(A|H_j)P(H_j)}{\sum_{i=1}^{n} P(A|H_i)P(H_i)}$$

1. Wzór Bayesa

Następny przełomowy krok wydarzył się dopiero w 1950 roku, kiedy Alan Turing w artykule *Computing Machinery and Intelligence* [6] zadał pytanie „Czy maszyny potrafią myśleć?”. Zaproponował grę, nazywaną później Testem Turinga, w której biorą udział trzy osoby. Na konwersację z maszyną każdy z ludzkich „sędziów” miał pięć minut. Jeśli co najmniej jeden na trzech rozmówców będzie przekonany, że rozmawia z człowiekiem, taką

maszynę należy uznać za inteligentną. Zaproponowany przez Turinga test był pierwszą próbą zdefiniowania tego, co można uważać za sztuczną inteligencję. Dwa lata później Arthur Samuel, amerykański pionier w dziedzinie gier komputerowych i sztucznej inteligencji, zbudował maszynę zdolną nauczyć się gry w warcaby [7]. Wykorzystywała opatrzone komentarzami ekspertów poradniki i grała przeciwko samej sobie, aby nauczyć się odróżniać właściwe posunięcia od niewłaściwych.

Nazwa „sztuczna inteligencja" powstała sześć lat później, na letnich warsztatach *Dartmouth Summer Research Project on Artificial Intelligence* [8]. To wydarzenie jest uznawane za narodziny sztucznej inteligencji jako nauki. Warsztaty trwały od sześciu do ośmiu tygodni i uczestniczyli w nich wybitni matematycy i naukowcy w tym: John McCarthy, Marvin Minsky, Nathaniel Rochester i Claude Shannon.

Pierwszą próbą zbudowania sieci neuronowej było stworzenie perceptronu [9] przez Franka Rosenblatta, amerykańskiego psychologa znanego w dziedzinie sztucznej inteligencji, rok po wydarzeniach z Darthmouth. Perceptron był inspirowany sposobem, w jaki neurony współpracują ze sobą w mózgu. Działał za pomocą rezystora obrotowego (potencjometru) napędzanego silnikiem elektrycznym. Jego działanie polegało na klasyfikowaniu danych pojawiających się na wejściu i ustawianiu stosownie do tego wartości wyjścia.

W latach 60 opublikowano wiele prac naukowych na temat metody najbliższych sąsiadów, szczególnie poruszonej w *Nearest neighbor pattern classification* [10], artykule napisanym w 1967 roku, przez T. Covera i P. Harta. Ta metoda przechowuje wszystkie dostępne przypadki i klasyfikuje nowe na podstawie miary podobieństwa. We wcześniej wspomnianym artykule, wyznaczyli oni trasę dla podróżujących sprzedawców, zaczynając od losowego miasta, ale upewniając się, że odwiedzą wszystkie miasta podczas możliwie najkrótszej podróży. Jest to klasyczna metoda używana przy rozpoznawaniu wzorców.

W roku 1973 nastąpiły trudne czasy dla dziedziny sztucznej inteligencji, Brytyjska Rada ds. Badań Naukowych opublikowała raport Jamesa Lighthilla [11], w którym przedstawiona była bardzo pesymistyczna prognoza, jeśli chodzi o rozwój głównych aspektów badań nad sztuczną inteligencją, co doprowadziło do zaprzestania ich finansowania na wszystkich uniwersytetach z wyjątkiem dwóch, a okres ten jest nazywany *AI Winter*. Mimo tego prace badawcze trwały. Kilka lat później studenci z uniwersytetu Stanford stworzyli robota o nazwie Cart [12], połączonego drogą radiową do komputera, który mógł samodzielnie

omijać przeszkody znajdujące się w pomieszczeniu. Mimo że przejście całego pokoju zajęło pięć godzin, wynalazek był wówczas uważany jako szczyt techniki.

W 1981 Gerald Dejong, profesor Uniwersytetu Illnois w Urbanie i Champaign, przedstawił koncepcję *Explanation Based Learning* (EBL) [13], która analizuje dane i tworzy ogólną zasadę, której można przestrzegać, odrzucając nieważne dane. Tej metody użyto na przykład do przetwarzania słów lub nauki gry w szachy. Cztery lata później profesor centrum biomedycznego Francis Crick Institute T. Sejnowski i profesor z Uniwersytetu Harvard Charles Rosenberg stworzyli NetTalk [14], czyli program, który uczy się wymowy tekstu, pisanego w języku angielskim, na podstawie tego, który otrzymał jako dane wejściowe i dla porównania dopasowując transkrypcje fonetyczne. Celem przyświecającym stworzeniu tego oprogramowania było stworzenie uproszczonego modelu, który pomoże zobrazować złożoność uczenia się zadań poznawczych na poziomie ludzkim.

W roku 1986 amerykańscy naukowcy, David Rumelhart, James McClelland i Geoffrey Hinton opublikowali *Parallel Distributed Processing* [15], w którym udoskonalono wykorzystanie modeli sieci neuronowych do uczenia maszynowego, co pozwoliło na rozwój w dziedzinie sztucznej inteligencji.

Sześć lat później, badacz Gerald Tesauro stworzył sieć neuronową TD-Gammon [16], która była w stanie grać w tryktraka (*ang. backgammon*) na poziomie najlepszych ludzkich graczy. Kolejną innowacją w sferze gier był pierwszy system komputerowy do gry w szachy, który został stworzony przez IBM i nosił nazwę Deep Blue [17]. Powstał w roku 1997 i był w stanie pokonać ówczesnego mistrza świata w szachach – Garry'ego Kasparov'a. System wykorzystał moc obliczeniową do przeszukiwania na dużą skalę potencjalnych ruchów i wybrania najlepszego z nich. Pośrednio dzięki niemu stworzono aktualnie najlepszy silnik szachowy Stokfish.

W 2006 roku, za sprawą brytyjskiego naukowca Geoffreya Hinton'a, powstał termin *deep learning* [18], w którego obszar wchodzą nowe algorytmy, pomagające komputerom rozróżniać obiekty i tekst w obrazach i filmach. Krótko potem, firma Microsoft stworzyła urządzenie o nazwie Kinect [19], które było w stanie wykryć ruch i opisać jego parametry za pomocą układu składającego się ze zwykłej kamery i uproszczonej kamery głębi. Pozwoliło to na interakcję z komputerem za pomocą ruchów i gestów. Następnym znaczącym wydarzeniem była wygrana superkomputera Watson [20] stworzonego przez IBM w 2011 roku, grającego w Jeopardy, z dwoma jej mistrzami. W tym samym roku

powstał Google Brain [21], czyli sieć neuronowa, która mogła odkrywać i kategoryzować obiekty w sposób podobny do kota. Rok później ukazał się artykuł badawczy [22] Alexa Krizhevsky'ego, Geoffreya Hintona i Ilyi Sutskevera opisujący model, który może radykalnie zmniejszyć współczynnik błędów w systemach rozpoznawania obrazów. W tym samym czasie firma Google opracowała algorytm uczenia maszynowego, który może autonomicznie przeglądać filmy w serwisie YouTube, aby identyfikować filmy zawierające koty, dzięki czemu w przyszłości filmy mogłyby być automatycznie sprawdzane pod kątem grupy wiekowej lub niestosownych treści łamiących regulamin YouTube, co skutkowałoby usunięciem filmu.

W 2014 roku Facebook stworzył algorytm DeepFace [23], który był w stanie rozpoznawać i identyfikować osoby znajdujące się na zdjęciach z dokładnością człowieka, dzięki czemu użytkownicy obecni na zdjęciach byli automatycznie oznaczani. Rok później Andy Jassy, dyrektor generalny Amazon Web Services, uruchomił usługi zarządzane przez sieci neuronowe, które analizują historyczne dane użytkowników w celu wyszukiwania wzorców i oferowania atrakcyjnych produktów. Jednocześnie firma Microsoft stworzyła Distributed Machine Learning Toolkit, czyli narzędzie, które umożliwia wydajną dystrybucję problemów uczenia maszynowego na wielu komputerach. Zostało użyte między innymi do stworzenia modelu rozpoznawania mowy Cortana, który został zaimplementowany w systemie operacyjnym Windows 10 i pozwala na sterowanie komputerem z wykorzystaniem mowy.

Dwa lata później, naukowcy z Google DeepMind stworzyli program AlphaGo [24], który był zdolny grać w starożytną chińską grę Go i wygrać cztery na pięć gier przeciwko Lee Sedol’owi, który był najlepszym graczem w Go przez ponad dekadę. W 2017 roku badacze z uniwersytetu Carnegie Mellon w Pitsburghu, stworzyli system Libratus [25], który pokonał czterech topowych graczy w Texas Hold’em po 20 dniach gry. Podobny rezultat odnieśli naukowcy z uniwersytetu w Albercie, używając swojego systemu Deepstack [26].

1.2 Typy uczenia maszenowego

Uczenie maszynowe można podzielić na różne rodzaje, w zależności od tego, co chce się uzyskać. Do najczęściej definiowanych należą: uczenie nadzorowane, uczenie nienadzorowane oraz uczenie ze wzmocnieniem.

Każdy model uczenia maszynowego opiera się na statystyce, dzięki której tworzy algorytm, który pozwala komputerowi „uczyć się". Uczenie się jest bardziej kwestią znalezienia statystycznych prawidłowości lub innych wzorców w danych niż posiadania jakiejś świadomości. Dlatego też uczenie się algorytmu rzadko przypomina sposób, w jaki zrobiłby to człowiek.

Pierwszym typem jest uczenie nadzorowane, najczęściej wykorzystywane w problemach klasyfikacyjnych, ponieważ głównym celem modelu jest nauczenie się, w jaki sposób dane powinny zostać zaklasyfikowane na podstawie zbudowanego systemu. Dla tego sposobu potrzebne są dane uczące dla pewnej liczby obiektów, zwanych obiektami uczącymi, na które składają się cechy każdego z obiektów i wynik klasyfikacji, zwany etykietą. Wszystkie te dane muszą być kompletne i zasadniczo powinny być one podane w formie liczbowej. Jest to najczęściej stosowana metoda dla sieci neuronowych i drzew decyzyjnych, ponieważ obie techniki są uzależnione od informacji podanych przez z góry określone klasyfikacje. W przypadku sieci neuronowych klasyfikacja jest używana do określenia błędu sieci i dopasowania jej tak, aby go zminimalizować; w drzewach decyzyjnych służy do określenia, jakie atrybuty dostarczają najwięcej informacji, które można wykorzystać do rozwiązania problemu klasyfikacyjnego [27].

Kolejnym typem jest uczenie nienadzorowane, które powala maszynie działać bez nadzoru, tak aby sama znalazła relacje w zbiorze danych i informacje nie podane jej jako dane. W tej metodzie dla uczącego zbioru obiektów podaje się tylko cechy, bez etykiet. Maszyna ma za zadanie odnaleźć w zbiorze obiektów uczących podzbiory obiektów podobnych, według podanego przez człowieka kryterium, i w ten sposób dokonać podziału tego zbioru na klasy. Można to łatwo wytłumaczyć na przykładzie dziecka i psa. Dziecko kojarzy, jak wygląda pies,. Kilka tygodni później znajomi rodziców przyjeżdżają w odwiedziny ze swoim psem. Mimo że dziecko wcześniej nie widziało tego psa, to rozróżnia cechy charakterystyczne dla psów takie jak dwoje uszu czy chodzenie na czterech łapach. Gdyby znajomy rodziców powiedział dziecku, że jest to pies, byłby to przykład uczenia

nadzorowanego. Uczenia nienadzorowanego używa się głównie w celu znalezienia wcześniej nieznanych wzorców i cech, które mogą być przydatne w przypadku kategoryzacji [28].

Ostatnim typem jest uczenie ze wzmocnieniem. Jest to sposób na trenowanie modelu tak, aby wykonywał odpowiednie sekwencje decyzyjne. Agenta, bo tak nazywa się algorytm wykonujący zadanie, umieszcza się w pewnym środowisku, w którym istnieją określone zasady. Ma on za zadanie podejmować sekwencje decyzji, których konsekwencje, np. wygrana/przegrana, są odsunięte w czasie. Aby nakierować maszynę na robienie tego, co zostało założone, agent otrzymuje nagrody lub kary za wykonywane czynności. Jego celem jest maksymalizacja całkowitej nagrody. Mimo że system nagród i kar jest z góry ustalony, agent nie dostaje żadnych podpowiedzi, np. w jaki sposób powinien postępować, musi sam dojść do rozwiązania zadania. Agent zaczyna od losowych parametrów, a kończy na wyrafinowanej taktyce. Tego podejścia często używa się w grach, np. League of Legends, Counter Strike: Global Offensive, szachy [29].

1.3 Istota sieci neuronowych

Sieci neuronowe to podzbiór uczenia maszynowego i podstawa uczenia głębokiego. Ich nazwa zainspirowana została połączeniami neuronów w ludzkim mózgu, jednakże poza tym nie mają ze sobą wiele wspólnego. Dla rozróżnienia używa się nazwy „sztuczna sieć neuronowa".

Sieci neuronowe wydają się nowym zagadnieniem, w rzeczywistości jednak ich historia sięga lat czterdziestych, kiedy Warren S. McCulloch i Walter Pitts opublikowali *A Logical Calculus of the Ideas Immanent in Nervous Activity* [30]. Praca ta dotyczyła badań nad ludzkim mózgiem i była próbą zrozumienia, w jaki sposób wytwarza on złożone wzorce dzięki połączeniom komórek mózgowych. Jednym z głównych pomysłów było porównanie neuronów z progiem binarnym do logiki zero jedynkowej. W 1958 roku Frank Rosenblatt stworzył perceptron, który udokumentowany został w jego pracy *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain* [9]. Usprawnił pomysł McCullocha i Pittsa wprowadzając wagi do równania, co sprawiło, że komputer IBM 704 [31] był w stanie odróżniać karty zaznaczone po lewej stronie od tych zaznaczonych po prawej stronie. Po niespełna dwudziestu latach Paul Werbos w swojej pracy doktorskiej [32] jako pierwszy zwrócił uwagę na zastosowanie propagacji wstecznej w sieciach neuronowych. Następnie w 1989 roku, Yann LeCun opublikował pracę *Backpropagation*

Applied to Handwritten Zip Code Recognition [33], w której pokazał, w jaki sposób wykorzystanie ograniczeń w propagacji wstecznej może być zastosowane do uczenia algorytmów. Wykorzystał sieć neuronową do rozpoznawania odręcznych cyfr kodu pocztowego dostarczanych przez US Postal Service. Bez wątpienia, dużą rolę w rozwoju sieci neuronowych odegrała firma Google, tworząc otwartoźródłową bibliotekę programistyczną TensorFlow [34], która jest powszechnie stosowana przez największych producentów na rynku technologicznym takich jak Nvidia, Apple czy USAA. [35]

Typy sieci

Do najprostszych należą sieci jednokierunkowe (*ang. FeedForward Neural Networks*), w których dane są przesyłane w jednym kierunku. Charakteryzują się pełnym połączeniem między neuronami i mogą, choć nie muszą, posiadać warstwę ukrytą, a ich ilość zależy od złożoności poruszanego problemu. Funkcja aktywacji pobiera dane wejściowe pomnożone przez wagi, które są statyczne. Ten typ sieci nie jest zbyt skomplikowany, łatwy w projektowaniu i dosyć szybki, jednak w przypadku braku warstw ukrytych nie może zostać użyty w nauczaniu.

Następnym typem sieci jest Perceptron Wielowarstwowy (*ang. Multi-Layer Perceptron*), który składa się z wielu warstw ukrytych, a wszystkie neurony są ze sobą połączone, do uczenia używa algorytmu propagacji wstecznej. Na początku dane pomnożone przez wagi są przesyłane do funkcji aktywacji, a następnie dzięki propagacji wstecznej są modyfikowane tak, aby zmniejszyć stratę, która jest obliczana na podstawie różnicy między wartościami przewidzianymi i treningowymi. Jako funkcji aktywacji w warstwie wyjściowej używa się zwykle funkcji nieliniowych takich jak softmax czy ReLu. Tę sieć można zastosować do rozpoznawania mowy lub bardzo złożonej klasyfikacji. Jest użyteczna w nauczaniu głębokim, ale nie jest łatwo dobrać odpowiednią strukturę sieci, a w dodatku wytrenowanie jej jest czasochłonne.

Do głównych typów sieci neuronowych można zaliczyć także sieci konwolucyjne (*ang. Convolutional Neural Network*), są one stosowane głównie do przetwarzania obrazów. Składają się z trójwymiarowego układu neuronów (wysokość, szerokość, głębia), zamiast dwuwymiarowej tablicy, tak jak było to w poprzednich przypadkach. Pierwsza warstwa nazywa się konwolucyjną, celem jej działania jest konwersja obrazu do formy, która jest łatwiejsza do przetworzenia, jednocześnie nie usuwając cech, które są istotne dla poprawnej predykcji, zwykle stosuje się w niej funkcję aktywacji ReLu. Następnie dane są przesyłane

do warstwy łączenia (*ang. pooling layer*), której celem, podobnie jak w warstwie konwolucyjnej, jest zmniejszenie rozmiaru przestrzennego, co skutkuje zmniejszeniem wymaganej mocy obliczeniowej. W tej warstwie używa się dwóch metod, tzw. Max Pooling i Average Pooling. Ta pierwsza zwraca maksymalną wartość z próbki pobranej przez model, z kolei druga metoda zwraca wartość średnią. Następnie warstwa spłaszczająca redukuje wymiary danych do żądanego kształtu i ostatecznie są one przesyłane do wspomnianej wcześniej sieci wielowarstwowej.

Ostatnim typem są sieci rekurencyjne (ang. *Recurrent Neural Networks*), które znalazły zastosowanie w przetwarzaniu tekstu na mowę, podpowiadaniu tekstu czy sprawdzaniu poprawności gramatycznej. Są zaprojektowane tak, aby zachować wyjściowe warstwy, RNN jest następnie przekazywana z powrotem do danych wejściowych, aby pomóc w przewidywaniu. Pierwsza warstwa jest zwykle siecią jednokierunkową, po której następuje warstwa sieci rekurencyjnych, w której niektóre informacje z poprzedniego kroku są zapamiętywane. Sprawia to, że każdy neuron zachowuje się jak komórka pamięci, podczas wykonywania obliczeń. Jeśli prognoza jest błędna, używany jest współczynnik uczenia, aby wprowadzić małe zmiany, które będą stopniowo działały w kierunku prawidłowego przewidywania podczas propagacji wstecznej.

2 Dane wejściowe i budowa modelu teoretycznego na podstawie dotychczasowych badań

Poniższy rozdział przedstawia zagadnienia związane z przetwarzaniem danych wejściowych, doborem zmiennych objaśniających, a także budową sieci neuronowej i sposobami jej optymalizacji.

2.1 Analiza danych wejściowych

Największym ogólnodostępnym zbiorem danych tenisowych jest repozytorium Jeffa Sackmanna [36] w serwisie Github, które zawiera dane o meczach od roku 1968, czyli tzw. Open Ery aż do dziś. Jednakże bardziej szczegółowe dane dostępne są dopiero od roku 1997. Co roku odbywa się około 2800 meczów, dlatego w celu optymalizacji zostały wybrane mecze tylko z okresu 2000 – 2019. Zbiór ten należało wcześniej wyczyścić, ponieważ nie był on do końca kompletny. Istnieje kilka sposobów radzenia sobie z tego typu problemami.

Średnia

Brakujące dane można zastąpić średnią, ponieważ uważa się ją za rozsądne oszacowanie dla losowo wybranej obserwacji z rozkładu normalnego. Jednakże w przypadku dużej dysproporcji brakujących danych, może to prowadzić do niespójności danych i dużego błędu systematycznego [37].

Moda

Kiedy dane są wypaczone, warto rozważyć zastąpienie pustych miejsc modą (dominantą), czyli najczęściej występującą wartością. Może ona zostać zastosowana dla danych numeryczny jak również kategorycznych [38].

Regresja

Metoda ta polega na zastąpieniu pustych miejsc wartościami przewidywanymi, na podstawie wcześniejszych danych. Pozwala uniknąć znaczącej zmiany odchylenia standardowego i kształtu rozkładu [37].

Ostatnia obserwacja

To podejście znalazło swoje zastosowanie w anestezjologii, gdzie badania robione są okresowo, więc gdy w danym momencie brakuje danych, zastępuje się je tymi z ostatnich badań. Zakłada ono jednak, że wartość pozostaje niezmienna, co jest mało prawdopodobne w wielu sytuacjach, dlatego nie jest to powszechnie stosowane podejście [37].

Usuwanie obserwacji

Istnieją sytuacje, w których zmienna nie posiada wielu wartości. W tym przypadku, jeśli nie jest ona bardzo istotna, można ją całkowicie odrzucić. Analogicznie postępuje się z pojedynczą obserwacją, którą można usunąć, jeżeli posiada wiele niewiadomych wartości. Należy jednak pamiętać, aby zachować rozwagę przy tej metodzie, ponieważ jeśli usuniemy za dużo danych, model nie będzie posiadał wystarczającej liczby danych, aby prawidłowo przewidywać przyszłe wartości [39].

2.2 Dobór zmiennych objaśniających

W przypadku uczenia nadzorowanego dane muszą mieć odpowiednią postać, dlatego też wektor X reprezentuje cechy graczy, a wartość docelowa y jest ustalona w następujący sposób:

$$y = \begin{cases} 1, \ jeśligracz1wygrał \\ 0, jeśligracz1przegrał \end{cases}$$

Mecze nierozstrzygnięte nie są brane pod uwagę, więc są to jedyne możliwe wyniki.

Aby model był efektywny w przewidywaniu, musi brać pod uwagę cechy obu graczy, dlatego zmienne w modelu zostały przedstawione w postaci różnic między cechami gracza 1 i gracza 2 np. $diff_{win_{overall}} = win_{overall_1} - win_{overall_2}$, gdzie $win_{overall_1}$ i $win_{overall_2}$ to odpowiednio średnia wygranych gracza 1 i 2 w momencie rozgrywania meczu. Te cechy można przedstawić jako osobne zmienne, lecz zwiększy to dwukrotnie ich ilość w modelu, przez co trening będzie trwał dłużej, a także zwiększy wariancję, i może spowodować mylne przypisanie większej wagi jednej ze zmiennych, co nie powinno mieć miejsca [40].

Elo Rating

Elo rating to wskaźnik, który powstał dzięki Arpadowi Elo, węgiersko-amerykańskiemu profesorowi fizyki, a jednocześnie mistrzowi szachowemu, w roku 1978

[41]. Elo Rating System został wprowadzony w celu oceny siły zawodników w grach takich jak szachy. W 2016 roku rozgłos zyskała zmodyfikowana wersja Elo, stworzona przez stronę FiveThirtyEight, która użyła jej do predykcji turnieju U.S. Open [42].

Dla meczu w czasie t pomiędzy p_ia p_j, ze wskaźnikiem Elo odpowiednio $E_i(t)$ i $E_j(t)$, przewiduje się, że p_i wygra mecz z prawdopodobieństwem:

$$\text{ft}_{ij} = \left(1 + 10^{\frac{E_j(t) - E_i(t)}{400}}\right)^{-1}$$

2. Prawdopodobieństwo wygrania meczu

Następnie wskaźnik jest aktualizowany według poniższej formuły:

$$E_i(t+1) = E_i(t) + K_{it} * \left(W_i(t) - \text{ft}_{ij}\right)$$

3. Wskaźnik Elo po zakończonym meczu

gdzie:

$W_i(t)$ to wskaźnik, który jest równy 1 lub 0, w zależności od tego, czy gracz p_i wygrał lub przegrał mecz.

K_{it} to współczynnik uczenia się dla p_i w czasie t, który opisany jest wzorem :

$$K_{it} = \frac{250}{\left(5 + m_i(t)\right)^{0.4}}$$

4. Współczynnik uczenia

gdzie:

$m_i(t)$ reprezentuje ilość meczy gracza p_i rozegranych w czasie t.

Aby zastosować te wzory, początkowe wartości ustala się następująco: $E_i(0) = 1500$ i $m_i(0) = 0$.

Wskaźnik ten został użyty zamiast różnicy miejsc w rankingu ATP, ponieważ zmienne te wykazywały silną korelację, a według Jacoba Golluba, Elo odznaczało się lepszą dokładnością [43]. Dobrą praktyką jest stworzenie macierzy korelacji, dzięki której można stwierdzić zależności liniowe zmiennych na podstawie współczynnika korelacji. Gdy jest on wysoki, oznacza to, że zmienne mają prawie taki sam wpływ na zmienną zależną, tak więc można odrzucić jedną z nich [44].

Zmienne użyte do trenowania modelu zostały zebrane w tabeli 1:

Tabela 1. Zmienne użyte w modelu

Nazwa zmiennej	Opis zmiennej
$diff_{ht}$	Wzrost
$diff_{bp}$	Średnia liczba wygranych punktów przełamań
$diff_{sp}$	Średnia liczba wygranych punktów serwisowych
$diff_{elo}$	Wskaźnik Elo
$diff_{hand}$	Średnia liczba wygranych przeciwko zawodnikowi grającemu daną ręką
$diff_{win_h 2h}$	Bilans między graczami
$diff_{win_{last5}}$	Ilość wygranych w ostatnich pięciu meczach
$diff_{win_{overall}}$	Średnia wygranych ogółem
$diff_{surface_{overall}}$	Średnia wygranych na danej powierzchni

Wszystkie zmienne w tabeli przedstawiono w postaci różnicy między odpowiednimi cechami graczy.

Kiedy dane są już przygotowane, trzeba je podzielić na zbiór treningowy, na którym model będzie uczył się zależności między zmiennymi i testowy, dzięki któremu będzie w stanie ocenić dopasowanie i polepszyć swoją wydajność w następnym kroku. Zbiór powinien zostać podzielony losowo, a dane testowe powinny wynosić około 1/3 całego zbioru. Jeśli zbiór testowy będzie zbyt mały, nie będzie można ocenić czy model dobrze generalizuje, ponieważ próba będzie niewystarczająca, z drugiej strony nie może też być zbyt duży względem zbioru treningowego, ponieważ można wtedy doprowadzić do sytuacji, w której model nie będzie miał się na czym uczyć. Gdy zbiór danych jest stosunkowo mały, zaleca się użycie walidacji krzyżowej [45].

Kolejnym krokiem jest poddanie danych procesowi transformacji normalizacyjnej, która ma na celu ujednolicenie jednostek miary i rzędów wielkości wszystkich zmiennych. Na ogół stosuje się dwa przekształcenia:

Standaryzacja, powoduje wyrównanie dyspersji i poziomu wartości cechy, co sprawia, że każda zmienna ma jednakowy wpływ na końcowy wynik. Wykorzystuje się poniższy wzór:

$$z_{ij} = \frac{x_{ij} - \overline{x_j}}{s_j}$$

5. Standaryzacja

Normalizacja, to przeskalowanie danych w zakres $[0,1]$. W tym procesie wykorzystujemy wzór poniższy;

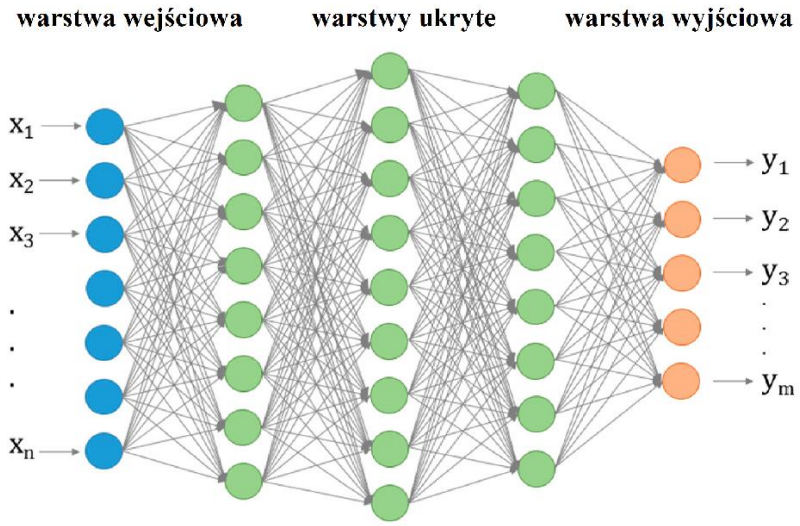
$$z_{ij} = \frac{x_{ij} - min\{x_j\}}{max\{x_j\} - min\{x_j\}}$$

6. Normalizacja

Należy pamiętać, aby nie dokonywać transformacji na zmiennej docelowej, ponieważ chcemy, aby model przewidział dokładnie te wartości, które są tam zawarte.

2.3 Budowa sieci neuronowej

Sieć składa się z warstwy wejściowej, co najmniej jednej warstwy ukrytej i warstwy wyjściowej. Każdy neuron jest połączony z neuronami w kolejnej warstwie, ma przypisaną wagę i próg klasyfikacji. Jeśli wynik z pojedynczego neuronu ma wartość powyżej ustalonego progu, neuron przesyła informacje do następnej warstwy sieci. W przeciwnym wypadku, neuron jest odrzucany [46]. Przykładowa struktura sieci neuronowej jest przedstawiona na rysunku (Rys. 2):

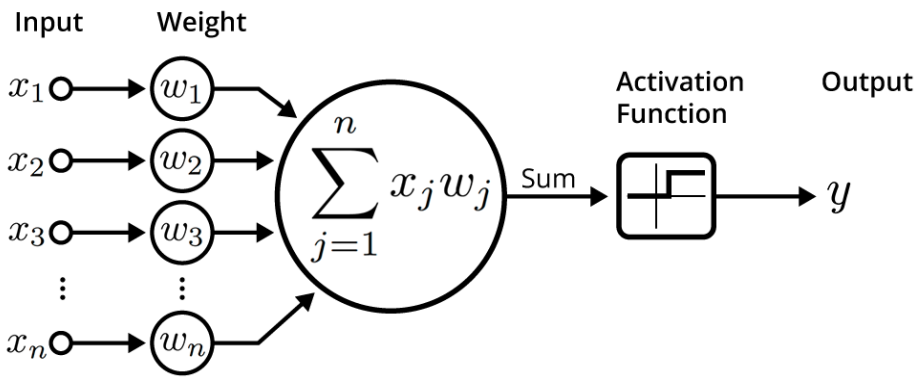


Rysunek 2. Schemat budowy głębokiej sieci neuronowej

Opracowanie własne na podstawie [47]

Sieci neuronowe opierają się na danych treningowych, na których „uczą się" i udoskonalają swoją dokładność. Dostrajanie się modelu może trwać kilku minut, godzin lub nawet dni, w zależności od ilości danych i złożoności modelu. Jednakże, kiedy model będzie już dostrojony, stanie się potężnym narzędziem, dzięki któremu będzie można klasyfikować dane o wiele szybciej, niż robiąc to „ręcznie".

Połączenia między neuronami są reprezentowane jako wcześniej wspomniane wagi, które mogą być zarówno pozytywne jak i negatywne. Im większa waga, tym większy wpływ jednego neuronu na drugi. Ich odpowiednie dobranie jest podstawowym sposobem szkolenia modelu uczenia sieci neuronowych. Gdy neuron otrzyma dane od neuronów z poprzedniej warstwy modelu, sumuje każdy sygnał pomnożony przez odpowiednią wagę i przekazuje je do funkcji aktywacji (Rys. 3).



Rysunek 3. Schemat działania sieci neuronowej

Źródło: [48]

Funkcja aktywacji oblicza wartość wyjściową dla neuronu. Jest ona następnie przekazywana do kolejnej warstwy sieci neuronowej przez inną synapsę. Istnieje wiele funkcji aktywacji, natomiast zwykle używa się tylko kilku z nich.

Funkcja liniowa, $Y = aX + b$, bierze pod uwagę zakres. Można zauważyć, że pochodna względem X to a czyli stała, co oznacza, że gradient nie ma związku z X, więc będzie on stały. Jeśli w przewidywaniu wystąpi błąd, zmiany wprowadzane przez algorytm propagacji wstecznej są stałe i nie zależą od zmiany $delta(X)$. Może ona sprawiać problemy przy użyciu metody gradientu prostego.

Następną funkcją jest funkcja sigmoidalna $Y = \frac{1}{1+e^{-X}}$, która jest nieliniowa, jej wartości znajdują w zakresie $(0,1)$, a małe zmiany X powodują znaczące zmiany Y. Jednakże, tak jak wcześniej wspomniana funkcja liniowa, nie jest idealna. Na końcu funkcji sigmoidalnej wartości Y reagują słabiej na większe wartości X, co oznacza, że gradient w tych miejscach będzie mały lub całkowicie znika i sieć przestaje się uczyć lub dzieje się to bardzo powoli. Stwarza to problem zanikającego gradientu.

Kolejną jest funkcja tanh: $Y = tanh(x) = \frac{2}{1+e^{-2X}} - 1$, która w rzeczywistości jest przeskalowaną funkcją sigmoidalną. Ona także jest nieliniowa, jednakże jej wartości mieszczą się w zakresie $(-1,1)$, a gradient jest silniejszy, ponieważ pochodne mają duże nachylenie. Podobnie jak funkcja sigmoidalna, tanh ma problem zanikającego gradientu.

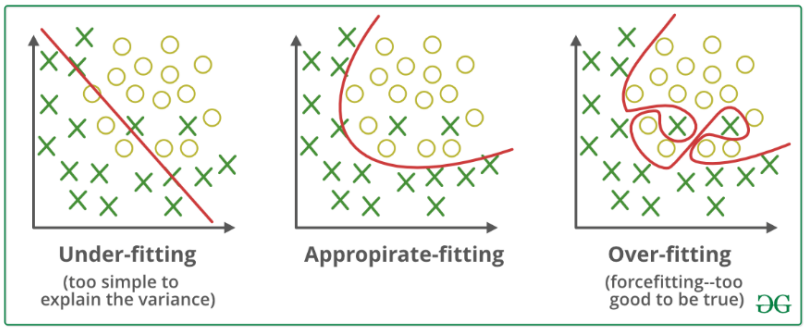
Ostatnią funkcją jest ReLu (*ang. Rectified Linear unit*), $Y = max(0, X)$. Jest najczęściej stosowaną funkcją aktywacji, potrzebuje mniej zasobów do obliczeń niż funkcje sigmoidalna i tanh, ponieważ używa prostszych operacji matematycznych, co jest istotne

przy projektowaniu głębokich sieci neuronowych, gdzie uczenie jest bardzo czasochłonne. Jednakże tak jak poprzednie funkcje, ta także ma wady. Można się spotkać z tzw. problemem umierającego ReLu. Ze względu na poziomą linię ReLu dla ujemnych X, gradient może dążyć do 0, gdzie w przypadku aktywacji w tym obszarze gradient także będzie wynosił 0, przez co wagi nie zostaną dostosowane. Ten problem może spowodować, że niektóre neurony przestaną reagować, czyniąc znaczną część sieci pasywną.

Kolejnym aspektem jest dobranie odpowiedniej liczby warstw ukrytych i neuronów. W większości problemów wystarczy jedna warstwa ukryta, dlatego ważniejsza jest ilość neuronów w warstwie. Zwykle stosuje się metodę prób i błędów, natomiast można spotkać się z tzw. *regułą kciuka*. Istnieją różne jej wariacje, według których liczba neuronów powinna być pomiędzy wielkością warstwy wejściowej a wyjściowej, innym podejściem jest 2/3 warstwy wejściowej plus wielkość wyjściowej lub też zasada, że wartość ta powinna być mniejsza niż dwukrotność warstwy wejściowej [49]. Sarle [50] natomiast twierdzi że te zasady są bezsensowne, ponieważ ignorują ilość szumu danych i złożoność modelu.

Często można napotkać sytuację, kiedy zbudowana sieć nie jest w stanie w wystarczającym stopniu nauczyć się problemu i działa słabo na obu zestawach danych. Zjawisko to nazywa się niedotrenowaniem (ang. *underfitting*), które charakteryzuje się dużym odchyleniem i małą wariancją. Zwykle jego przyczyną jest zbyt mało złożony model lub zbyt mała ilość danych. W tym przypadku najlepiej zwiększyć ilość warstw ukrytych i zamieszczonych w nich neuronów, powiększyć zbiór, na którym model jest uczony lub dodać nowe zmienne.

Znacznie częstszym przypadkiem jest przetrenowanie (ang. *overftting*), czyli sytuacja, w której model dobrze spisuje się na danych treningowych, ale na danych testowych już nie, co jest istotne, ponieważ celem modelu jest dobra generalizacja. Istnieje wiele metod, aby poradzić sobie z tym problemem. Tak jak w przypadku niedotrenowania można zmienić strukturę sieci, lecz w tym wypadku, zamiast zwiększać liczbę warstw ukrytych i neuronów, powinno się je zmniejszyć [51]. Aby lepiej zobrazować zjawisko przetrenowania i niedotrenowania został przedstawiony rysunek (Rys. 4):



Rysunek 4. Niedotrenowanie, idealne dopasowanie i przetrenowanie

Źródło: [52]

Przy problemie przetrenowania często używa się metod regularyzacji, które zmniejszają błąd generalizacji, utrzymując małe wagi w sieci. Są one tak szeroko stosowane w celu zmniejszenia overfittingu, że termin „regularyzacja" może być używany w odniesieniu do każdej metody, która poprawia błąd generalizacji modelu sieci neuronowej.

Pierwszą metodą jest regularyzacja aktywności, która zapewnia podejście „zachęcające" sieć do uczenia się rzadkich cech poprzez nałożenie tzw. kar. L1, gdzie aktywność jest obliczana jako suma wartości bezwzględnych, L2 jako suma kwadratów wartości i L1_L2 jako połączenie dwóch poprzednich. Przyjmują one parametr (a w przypadku L1_L2, dwa parametry), który kontroluje miarę, jaką każde działanie wnosi do sumy [53].

Następna metoda to Dropout, która losowo dezaktywuje niektóre neurony, co powoduje, że przy każdej aktualizacji warstwy, połączenia w sieci wyglądają inaczej. Taka koncepcja sugeruje, że Dropout nie pozwala zaistnieć sytuacjom, w których warstwy wspólnie by się dostosowywały, co czyni model niezawodnym. Tę metodę najczęściej stosuje się w głębokich sieciach neuronowych, które mają tendencję do szybkiego przetrenowania [54].

Ostatnią metodą jest tzw. *Early Stopping*, która kończy proces uczenia, jeśli jakość modelu nie zwiększyła się przez ustalony czas. Podczas uczenia dużej sieci, w końcu nastąpi moment, gdy model przestanie dobrze generalizować i zacznie uczyć się szumu statystycznego, co spowoduje wzrost błędu generalizacji [55].

Kiedy zostanie wybrana struktura sieci, należy ją skompilować. W tym miejscu należy wybrać funkcję straty, optymalizator i miarę, pod kątem której chcemy

optymalizować naszą sieć. Funkcja straty ma za zadanie ocenić, jak dobrze algorytm opisuje zbiór danych. Jeśli przewidywania są całkowicie błędne, funkcja straty wyświetli wyższą wartość, a jeśli są całkiem niezłe, zwróci niższą, dlatego też celem jest jej minimalizacja. Istnieją różne rodzaje funkcji straty wybierane w zależności od poruszanego problemu. Do regresjii używa się MAE, MSE lub MAPE, przy problemie klasyfikacji wieloetykietowej stosuje się Categorical Cross Entropy, a do klasyfikacji binarnej Binary Cross Entropy [56].

Następnym krokiem jest wybranie optymalizatora, którego zadaniem jest zmniejszenie strat i zapewnienie możliwie jak najdokładniejszych wyników. Najczęściej stosowanym jest Adam (Adaptive Moment Estimation), który używa momentum – współczynnika wnoszącego bezwładności [57] – pierwszego i drugiego rzędu. Jego konwencją jest, aby optymalizacja przebiegała powoli, co zmniejszy szansę na ominięcie minimum. Wadą tego podejścia jest duża wariancja i zużycie zasobów obliczeniowych.

Warto też wspomnieć o optymalizatorach opartych o metodę gradientu, a konkretnie o Mini-Batch Gradient Descent, która jest ulepszoną wersją gradientu prostego i jej odmiany stochastycznej. Nie potrzebuje ona sporej ilości pamięci, a także charakteryzuje się niską wariancją. Problemem jest natomiast dobranie odpowiedniej wartości współczynnika uczenia, ponieważ jeśli tempo uczenia jest zbyt niskie, proces uczenia będzie trwał bardzo długo [58].

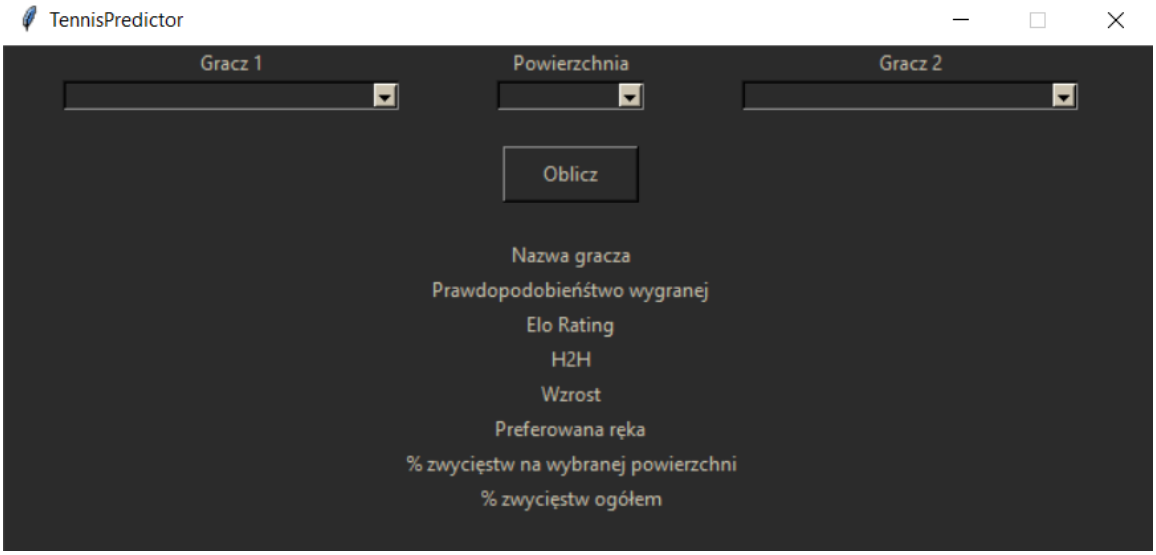
Kiedy nasza sieć jest już gotowa, można przejść do jej trenowania. Podczas tego procesu należy zdecydować o ilości tzw. *epok*. Jedna *epoka* to sytuacja, w której cały zbiór danych zostanie przetworzony przez model jeden raz. Należy pamiętać, aby ilość *epok* była optymalna, ponieważ dopasowywanie parametrów, zwykle niepotrzebnie, trwa zbyt długo. W takim przypadku można wykorzystać wcześniej wspominany *Early Stopping*. *Epokę* dzieli się na tzw. *batche*, czyli mniejsze części zajmujące mniej pamięci niż cały zbiór danych [59].

3 Predykcja wyników meczów tenisowych w oparciu o stworzoną aplikację

W poniższym rozdziale zaprezentowano stworzoną aplikację, sposób jej działania, a także narzędzia i biblioteki wykorzystane w trakcie pisania programu. Następnie przybliżono charakterystykę implementacji, a na końcu zaprezentowano analizę otrzymanych wyników.

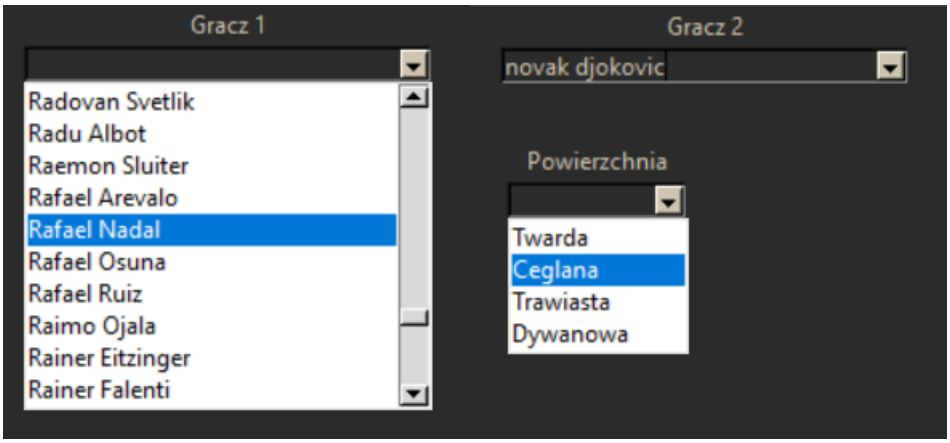
3.1 Prezentacja stworzonej aplikacji

Stworzona aplikacja została nazwana *TennisPredictor* i była wzorowana na porównywarce zawodników ze strony Ultimate Tennis Statistics [60]. Zawiera ona podobne funkcjonalności, lecz zdecydowanie prostsze, ponieważ pełnią one rolę przykładów.



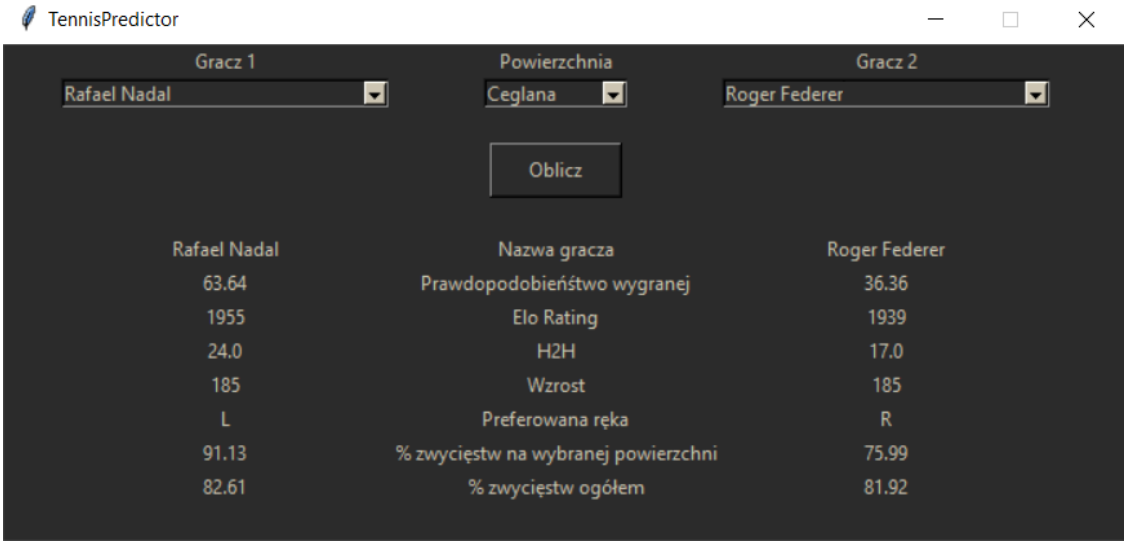
Rysunek 5. Startowy wygląd aplikacji.

Użytkownik posiada możliwość wyboru nazwy zawodnika z listy w polu *Gracz 1* i *Gracz 2*, lub wpisaniu jej ręcznie, a także wybór nawierzchni hipotetycznego spotkania z listy w polu *Powierzchnia* (Rys. 6).



Rysunek 6. Dostępne możliwości wyboru

Po wciśnięciu przycisku ***Oblicz***, poniżej pól wyboru wyświetlą się podstawowe informacje na temat wybranych zawodników, jak i prawdopodobieństwo wygranej w tym hipotetycznym starciu obliczone przez sieć neuronową (Rys. 7).



Rysunek 7. Hipotetyczny starcie między wybranymi zawodnikami.

H2H odpowiada liczbie wygranych meczy między tymi zawodnikami, a *Elo Rating* oznacza wskaźnik Elo obliczony dla gracza na podstawie jego historii meczy.

Aplikacja nie dokona obliczeń, jeśli dane w którymkolwiek z pól zostaną wprowadzone niepoprawnie lub w polu *Gracz 1* i *Gracz 2* zostaną wybrani ci sami gracze. Dodatkowo, jeśli jeden z graczy nie grał na danej powierzchni, w wierszu *% zwyciestw na wybranej powierzchni* po jego stronie wyświetli się komunikat *brak danych*.

3.2 Przedstawienie wykorzystanych narzędzi

Do implementacji został wykorzystany język programowania *Python*, ponieważ jego składnia jest bardzo prosta, dodatkowo zostało stworzone wiele bibliotek dla dziedziny sztucznej inteligencji, a w dodatku posiada on bardzo dużą bazę użytkowników.

Do pracy nad skryptami wykorzystany został *Jupyter Notebook*, czyli edytor który jest powszechnie stosowany w dziedzinie sztucznej inteligencji, ponieważ pozwala na uruchamianie wybranych fragmentów kodu w blokach, co może przypominać programowanie w języku R. Dzięki temu nie trzeba za każdym razem uruchamiać całego skryptu, którego wykonanie czasem może trwać nawet kilka godzin.

Bardzo pomocna okazała się biblioteka *pandas*, która pozwalała na operacje na ramkach danych, oraz na wczytywanie i zapisywanie ich do plików z rozszerzeniami .csv. Jest ona zbudowana w oparciu o bibliotekę *numpy*, także używaną w projekcie, która pozwala na obsługę wielowymiarowych tablic i macierzy, a także posiada duży zbiór funkcji matematycznych do ich obsługi. Jest bardzo szybka, ponieważ została napisana w oparciu o język C, a jej tablice i macierze zajmują o wiele mniej miejsca, niż listy stworzone w Pythonie.

W celu wyznaczenia funkcji gęstości dla każdej zmiennej, aby następnie stworzyć histogramy i ocenić rozkład danych, użyto biblioteki *scipy*, dzięki której możliwe było szybkie wyliczenie wartości oczekiwanej i odchylenia standardowego potrzebnych jako parametry rozkładu normalnego.

Do podziału zbioru danych na treningowy i testowy użyto biblioteki *sklearn* i pochodzącego z niej modułu *train_test_split*. Oferuje ona także moduły służące do transformacji danych takie jak *MinMaxScaler* i *StandardScaler*, a dzięki bibliotece *pickle* można używać go w innych skryptach.

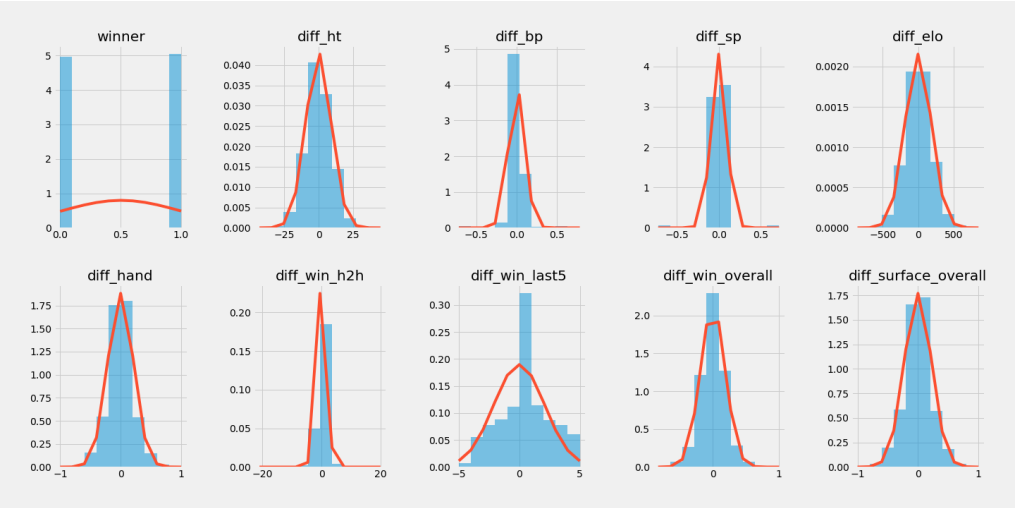
Sieć neuronowa została stworzona w oparciu o model *Sequential()* z frameworka *Keras*, który z kolei został zbudowany na podstawie biblioteki TensorFlow, stworzonej przez Google. Framework oferuje proste i naturalne tworzenie różnego rodzaju zaawansowanych sieci neuronowych. Aby stworzyć strukturę sieci użyto modułów *Input, Dense, Dropout*, a także tzw. odwołań (*ang. callbacks*) *ModelChechpoint* i *EarlyStopping*, aby usprawnić proces uczenia. Dzięki modułowi *load_model* można używać modelu w innych skryptach.

Aby zobrazować rozkład danych i pokazać jak przebiegał proces uczenia sieci użyto biblioteki *matplotlib* i modułu *pyplot*, które służą do tworzenia wykresów w Pythonie, a do stworzenia wyglądu aplikacji użyto biblioteki *tkinter,* która pozwala tworzyć proste interfejsy graficzne.

3.3 Charakterystyka implementacji

Cały projekt opiera się na pliku *all_matches.xlsx*, który zawiera wszystkie mecze z lat 1968-2020. W skrypcie *preprocessing.ipynb* następuje pozbycie się niepotrzebnych kolumn takich jak *id meczu*, *narodowość gracza* itp. Następnym krokiem było zajęcie się brakującymi danymi. Zastosowano najczęściej stosowane metody, takie jak zastąpienie danych średnią, w przypadku np. wzrostu, modą, gdy chodziło o preferowaną rękę lub zerami, gdy brakowało danych dotyczących statystyk graczy. W pliku *all_matches.xlsx,* zwycięzca znajdował się w kolumnie *winner,* a przegrany w kolumnie *loser,* dlatego też zmieniono ich nazwy odpowiednio na *p1* i *p2*, a następnie wymieszano miejscami cechy im odpowiadające, aby mieć pewność, że model nie uzna pozycji gracza w tabeli za zależność. Następnie stworzono pomocnicze ramki danych *df_player, df_surface* i *df_versus,* w których przechowywano odpowiednio: podstawowe informacje o graczach, liczbę zwycięstw na danej powierzchni i bilans między graczami. Stworzono funkcje *fill_params* i *calc_params,* które realizowały wypełnianie i zliczanie danych. Na samym końcu stworzona została ramka danych *result*, która zawierała wszystkie interesujące zmienne, a następnie zapisano ją, jak i ramki pomocnicze, do plików *.csv.*

W pliku *neural_network.ipynb* na początku wczytano plik result.csv, dane przekonwertowano na odpowiednie formaty i wybrano zakres meczy od roku 2000 do 2019. Następnie stworzona została macierz korelacji i mimo, że niektóre z nich wskazywały wysoki współczynnik korelacji, żadna nie osiągnęła wartości 0.9. Dlatego też wszystkie były brane pod uwagę. Kolejnym krokiem było sporządzenie histogramów i nałożenie na nie funkcji gęstości. Na rysunku (Rys. 8) można zauważyć, że rozkład danych przypomina rozkład normalny:



Rysunek 8. Histogramy dla zbioru danych

Dlatego też po podzieleniu danych na zbiór treningowy i testowy, zostały one zestandaryzowane za pomocą modułu *StandardScaler*, a następnie zbiór został zapisany do pliku *scaler.pkl* w celu późniejszego użycia. Utworzono dodatkowy zbiór walidacyjny, który obejmował sezon 2020 w celu sprawdzenia na nim dokładność modelu. Nie był on używany podczas treningu, dlatego mamy pewność że dla modelu są to nowe dane. Po wielu testach, finalna struktura sieci składała się z warstwy wejściowej z liczbą neuronów odpowiadającą liczbie zmiennych, jednej warstwy ukrytej z 64 neuronami i z funkcją aktywacji *ReLu*, warstwy *Dropout* ze współczynnikiem 0.5 oraz warstwy wyjściowej z jednym neuronem i funkcją aktywacji *sigmoid* w celu uzyskania wyniku w przedziale [0,1]. Następnie sieć skompilowano, przy użyciu funkcji straty *Binary Cross-Entropy*, ponieważ używa się jej w problemach klasyfikacji binarnej i celem jest jej minimalizacja. Jako optymalizatora użyto metody *Adam*, ponieważ łączy ona najlepsze cechy algorytmów *AdaGrad* i *RMSProp*, a miarą maksymalizacji jest *accuracy*, czyli dokładność modelu. Na histogramie zmiennej wyjściowej *winner* (Rys. 8), można zauważyć, że dane są zbalansowane, dlatego do zmierzenia dokładności wystarczy użyć –miar *accuracy* i *loss*, które wyrażają się poprzez poniższe wzory:

$$loss = \frac{-1}{N}\sum_{i=1}^{N} y_i * log\big(p(y_i)\big) + (1 - y_i) * log\big(1 - p(y_i)\big)$$

7 Wzór funkcji straty

$$accuracy = \frac{Liczba\ poprawnych\ klasyfikacji}{Ogólna\ liczba\ klasyfikacji}$$

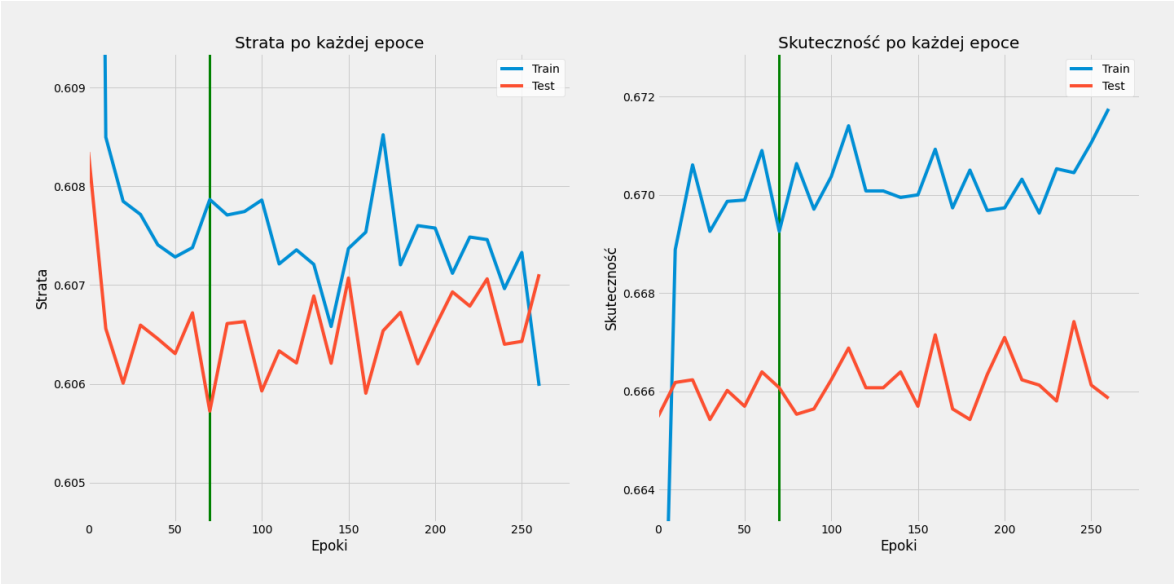
8 Wzór dokładności modelu

Uczenie sieci odbywało się na podstawie algorytmu propagacji wstecznej, a liczba epok została domyślnie ustawiona na 1000, aby optymalizator miał czas znaleźć minimum globalne dla miary *loss*, natomiast dzięki odwołaniu *EarlyStopping* proces ten kończył się, jeśli wartość straty nie zmniejszyła się przez 200 epok. Ostatecznie, aby z całego procesu pobrać jak najlepszy model, zastosowano odwołanie *ModelCheckpoint*, które monitorowało cały przebieg nauczania i zapisywało tylko najlepsze parametry sieci.

W skrypcie *main.py* zaimplementowany został interfejs graficzny dla aplikacji, w którym wybrane zostają nazwy zawodników i powierzchnia hipotetycznego meczu, zaś po wciśnięciu przycisku Oblicz pobierane są dane z wczytanych ramek pomocniczych. Są one odpowiednio przeliczane, transformowane przez *skaler*. Ostatecznie, zwracane są wyniki obliczone przez model.

3.4 Analiza otrzymanych wyników

Na rysunku (Rys. 9) przedstawiono proces uczenia modelu. Linia pionowa wskazuje moment, w którym model osiągnął minimum dla funkcji straty, dlatego też algorytm *EarlyStopping* zakończył proces uczenia 200 epok później.



Rysunek 9. Wykresy przedstawiające proces uczenia modelu

Wyniki otrzymane na podstawie zbioru testowego zostały przedstawione w tabeli (Tab. 2)

Tabela 2. Dokładność modelu na zbiorze testowym i walidacyjnym

zbiór testowy		**zbiór walidacyjny**	
strata	dokładność	strata	dokładność
0.6056	66,66 %	0.6097	64,58 %

Wyniki mogą się różnić ze względu na stochastyczny charakter algorytmu lub procedury oceny albo różnice w dokładności numerycznej.

W pracy Michala Sipko [61] wartość straty dla danych testowych wynosiła 0.6111, a dla walidacyjnych 0.5766. Można zatem stwierdzić, że udało się stworzyć model podobny do tego zaprojektowanego przez Michala Sipko. Procent poprawnie przewidzianych meczy nie został do końca sprecyzowany w pracy [61], dlatego też nie ma możliwości, aby porównać te miary.

Mimo tego, że ogólna dokładność modelu nie jest zbyt wysoka, dla niektórych turniejów z sezonu 2020, jak na przykład dla turnieju w Dubaju, wyniosła aż 80,65 %, co można uznać za zadowalający wynik.

Podsumowanie i wnioski

Celem pracy było stworzenie aplikacji, która przewiduje wyniki meczów tenisowych na podstawie danych historycznych. Do predykcji wykorzystano odpowiednio zaprojektowaną sieć neuronową i narzędzia obliczeniowe zawarte w modułach języka programowania Python. Cel pracy został osiągnięty. Stworzona aplikacja zwraca obiecujące prognozy. Po przeprowadzeniu uczenia perceptronu wielowarstwowego na meczach tenisowych z lat 2000 – 2019, uzyskano dokładności około 65 %. Porównując wyniki z danymi z literatury, które wynoszą 60 – 70 %, można stwierdzić że sieć neuronowa tego typu (oraz wykorzystująca ją aplikacja) może być stosowana do prognozowania wyników meczy tenisowych z umiarkowanym powodzeniem. Przewidywanie przyszłości to niezwykle trudne, zasadniczo nierozwiązywalne zagadnienie, w szczególności gdy mamy do czynienia z czynnikiem ludzkim, więc każde znaczne przekroczenie progu 50% należy uważać za sukces. Stworzona aplikacja jest zaledwie prototypem i istnieje oczywiście możliwość jej rozwoju, zaczynając chociażby od zmiany zbioru danych na bardziej rozbudowany, wprowadzając modyfikacje struktury sieci lub używając metod uczenia maszynowego takich jak *las losowy* czy *maszyny wektorów nośnych*. Warto również rozpatrzyć użycie tzw. *Ensemble Methods,* w których wykorzystuje się wyniki klasyfikacji pochodzące z użycia wielu metod, aby wyciągnąć wnioski ogólniejsze i dzięki temu osiągnąć lepsze wyniki.

Bibliografia

[1] „Jak przewidywać przyszłe ceny akcji,” 1 Marzec 2011. [Online]. Available: https://www.bankier.pl/wiadomosc/Jak-przewidywac-przyszle-ceny-akcji-2295313.html. [Data uzyskania dostępu: 6 Kwiecień 2021].

[2] Goget, „Historia tenisa ziemnego,” [Online]. Available: https://www.tenis.net.pl/historia-tenisa-ziemnego/. [Data uzyskania dostępu: 6 Kwiecień 2021].

[3] B. NT, „Machine Learning: The complete history in a timeline,” 7 Grudzień 2019. [Online]. Available: https://roboticsbiz.com/machine-learning-the-complete-history-in-a-timeline/. [Data uzyskania dostępu: 6 Kwiecień 2021].

[4] F. BAYES, „AN ESSAY TOWARDS SOLVING A PROBLEM IN THE DOCTRINE OF CHANCES,” tom 45, nr 3-4, p. 296–315, 23 Grudzień 1958.

[5] W. Zieliński, „Rachunek_wyklad_part1,” [Online]. Available: http://wojtek.zielinski.statystyka.info/Probabilistyka/Rachunek_wyklad_part1.pdf. [Data uzyskania dostępu: 6 Kwiecień 2021].

[6] A. M. TURING, „COMPUTING MACHINERY AND INTELLIGENCE,” *Mind,* tom 49, nr 236, pp. 433-460, 1950.

[7] J. McCarthy, „Arthur Samuel: Pioneer in Machine Learning,” [Online]. Available: http://infolab.stanford.edu/pub/voy/museum/samuel.html. [Data uzyskania dostępu: 6 Kwiecień 2021].

[8] J. Moor, „The Dartmouth College Artificial Intelligence Conference: The Next Fifty years,” *AI Magazine,* tom 27, nr 4, pp. 87-90, 2006.

[9] F. ROSENBLATT, „THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN,” *Psychological Review,* tom 65, nr 6, 1958.

[10] T. C. a. P. Hart, „Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory,* tom 13, nr 1, pp. 21-27, 1967.

[11] J. Lighthill, „Artificial Intelligence: A General Survey,” 1973.

[12] „First Robot 3D Mapping and Navigation in Ordinary Settings,” [Online]. Available: https://frc.ri.cmu.edu/~hpm/project.archive/robot.papers/2002/Commercial/1979.MRL.html. [Data uzyskania dostępu: 6 Kwiecień 2021].

[13] R. M. GERALD DEJONG, „Explanation-Based Learning: An Alternative View,” Kluwer Academic Publishers, Boston, 1986.

[14] C. R. R. Terrence J. Sejnowski, „NETtalk: a parallel network that learns to read aloud,” 1986.

[15] D. E. R. G. E. H. J. L. McCLELLAND, „The Appeal of Parallel Distributed Processing,” MIT Press, 1987.

[16] G. Tesauro, „TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play,” *Neural Computation,* tom 6, nr 2, pp. 215-219, 1994.

[17] C. HIGGINS, „A Brief History of Deep Blue, IBM's Chess Computer,” 29 Lipiec 2017. [Online]. Available: https://www.mentalfloss.com/article/503178/brief-history-deep-blue-ibms-chess-computer. [Data uzyskania dostępu: 6 Kwiecień 2021].

[18] „Geoffrey Hinton,” [Online]. Available: https://en.wikipedia.org/wiki/Geoffrey_Hinton. [Data uzyskania dostępu: 6 Kwiecień 2021].

[19] „Kinect,” [Online]. Available: https://pl.wikipedia.org/wiki/Kinect#cite_note-6. [Data uzyskania dostępu: 6 Kwiecień 2021].

[20] E. Burns, „IBM Watson supercomputer,” [Online]. Available: https://searchenterpriseai.techtarget.com/definition/IBM-Watson-supercomputer. [Data uzyskania dostępu: 6 Kwiecień 2021].

[21] „Google Brain,” [Online]. Available: https://en.wikipedia.org/wiki/Google_Brain. [Data uzyskania dostępu: 6 Kwiecień 2021].

[22] I. S. G. E. H. Alex Krizhevsky, „ImageNet Classification with Deep Convolutional Neural Networks,” *Advances in Neural Information Processing Systems,* 2012.

[23] M. Y. M. R. L. W. Yaniv Taigman, „DeepFace: Closing the Gap to Human-Level Performance in Face Verification,” 2014.

[24] „AlphaGo,” [Online]. Available: https://deepmind.com/research/case-studies/alphago-the-story-so-far. [Data uzyskania dostępu: 6 Kwiecień 2021].

[25] „Libratus,” [Online]. Available: https://pl.wikipedia.org/wiki/Libratus. [Data uzyskania dostępu: 6 Kwiecień 2021].

[26] „DeepStack,” [Online]. Available: https://www.deepstack.ai/. [Data uzyskania dostępu: 6 Kwiecień 2021].

[27] T. Ayodele, „Types of Machine Learning Algorithms,” w *New Advances in Machine Learning,* United Kingdom, 2010.

[28] „Unsupervised Machine Learning: What is, Algorithms, Example,” [Online]. Available: https://www.guru99.com/unsupervised-machine-learning.html#2. [Data uzyskania dostępu: 6 Kwiecień 2021].

[29] K. B. Błażej Osiński, „What is reinforcement learning? The complete guide," 5 Lipiec 2018. [Online]. Available: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/. [Data uzyskania dostępu: 6 Kwiecień 2021].

[30] W. M. a. W. Pitts, „A Logical Calculus of the Ideas Immanent in Nervous Activity," *Brain Theory*, pp. 229-230.

[31] „IBM 704," [Online]. Available: https://en.wikipedia.org/wiki/IBM_704. [Data uzyskania dostępu: 6 Kwiecień 2021].

[32] P. Werbos, „Beyond regression : new tools for prediction and analysis in the behavioral sciences," 1974.

[33] Y. LeCun, „Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation,* tom 1, nr 4, pp. 541-551, 1989.

[34] „TensorFlow," [Online]. Available: https://www.tensorflow.org/. [Data uzyskania dostępu: 6 Kwiecień 2021].

[35] „Companies Currently Using TensorFlow," [Online]. Available: https://discovery.hgdata.com/product/tensorflow. [Data uzyskania dostępu: 6 Kwiecień 2021].

[36] J. Sackmann, „tennis_atp," [Online]. Available: https://github.com/JeffSackmann/tennis_atp. [Data uzyskania dostępu: 7 Kwiecień 2021].

[37] H. Kang, „The prevention and handling of the missing data," 23 Maj 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3668100/. [Data uzyskania dostępu: 7 Kwiecień 2021].

[38] A. Kumar, „Data Analytics," 23 Lipiec 2020. [Online]. Available: https://vitalflux.com/pandas-impute-missing-values-mean-median-mode/. [Data uzyskania dostępu: 13 Maj 2021].

[39] D. Berdikulov, „Dealing with Missing Data," 3 Kwiecień 2019. [Online]. Available: https://medium.com/@danberdov/dealing-with-missing-data-8b71cd819501. [Data uzyskania dostępu: 7 Kwiecień 2021].

[40] G. S. a. D. W. Andre Cornman, „Machine Learning for Professional Tennis Match Prediction and Betting," 2017.

[41] A. E. Elo, The rating of chessplayers, past and present, New York: Arco Pub., 1978.

[42] C. B. J. B. Benjamin Morris, „How We're Forecasting The 2016 U.S. Open," FiveThirtyEight, [Online]. Available: https://fivethirtyeight.com/features/how-were-forecasting-the-2016-us-open/. [Data uzyskania dostępu: 7 Kwiecień 2021].

[43] J. Gollub, Producing Win Probabilities for Professional Tennis Matches from any Score, 2017.

[44] V. R, „Feature selection — Correlation and P-value,” 11 Wrzesień 2018. [Online]. Available: https://towardsdatascience.com/feature-selection-correlation-and-p-value-da8921bfb3cf. [Data uzyskania dostępu: 7 Kwiecień 2021].

[45] J. Brownlee, „Train-Test Split for Evaluating Machine Learning Algorithms,” Machine Learning Mastery, 26 Sierpień 2020. [Online]. Available: https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/. [Data uzyskania dostępu: 7 Kwiecień 2021].

[46] „Neural Network,” IBM, [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks. [Data uzyskania dostępu: 7 Kwiecień 2021].

[47] V. M. C. M. Alberto de la Fuente, Hydrological Early Warning System Based on a Deep Learning Runoff Model Coupled with a Meteorological Forecast, 2019.

[48] N. McCullum, „Deep Learning Neural Networks Explained in Plain English,” [Online]. Available: https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/. [Data uzyskania dostępu: 7 Kwiecień 2021].

[49] J. Heaton, „The Number of Hidden Layers,” Heaton Research, 1 Czerwiec 2017. [Online]. Available: https://www.heatonresearch.com/2017/06/01/hidden-layers.html. [Data uzyskania dostępu: 7 Kwiecień 2021].

[50] W. Sarle, „How many hidden units should I use?,” [Online]. Available: http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html. [Data uzyskania dostępu: 13 Maj 2021].

[51] J. Brownlee, „How to Avoid Overfitting in Deep Learning Neural Networks,” Machine Learning Mastery, 17 Gtudzień 2018. [Online]. Available: https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/. [Data uzyskania dostępu: 7 Kwiecień 2021].

[52] „Underfitting and Overfitting in Machine Learning,” 18 Maj 2020. [Online]. Available: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/. [Data uzyskania dostępu: 7 Kwiecień 2021].

[53] J. Brownlee, „How to Reduce Generalization Error With Activity Regularization in Keras,” Machine Learning Mastery, 30 Listopad 2018. [Online]. Available: https://machinelearningmastery.com/how-to-reduce-generalization-error-in-deep-neural-networks-with-activity-regularization-in-keras/. [Data uzyskania dostępu: 7 Kwiecień 2021].

[54] J. Brownlee, „A Gentle Introduction to Dropout for Regularizing Deep Neural Networks,” Machine Learning Mastery, 3 Grudzień 2018. [Online]. Available: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. [Data uzyskania dostępu: 7 Kwiecień 2021].

[55] J. Brownlee, „A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks,” Machine Learning Mastery, 6 Sierpień 2019. [Online]. Available: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/. [Data uzyskania dostępu: 7 Kwiecień 2021].

[56] D. Brzeziński, Uczenie głębokie cz. I, podstawy teoretyczne, konwolucje, keras i tensorflow.

[57] R. Tadeusiewicz, Najważniejszy element wiedzy o sieciach neuronowych, 2013.

[58] S. Doshi, „Various Optimization Algorithms For Training Neural Network,” Towards Data Science, 13 Styczeń 2019. [Online]. Available: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6. [Data uzyskania dostępu: 7 Kwiecień 2021].

[59] S. Sharma, „Epoch vs Batch Size vs Iterations,” Towards Data Science, 23 Wrzesień 2017. [Online]. Available: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9. [Data uzyskania dostępu: 7 Kwiecień 2021].

[60] M. Cekovic, „Ultimate Tennis Statistics,” [Online]. Available: https://www.ultimatetennisstatistics.com/headToHead. [Data uzyskania dostępu: 16 Kwiecień 2021].

[61] M. Sipko, „Machine Learning for the Prediction of Professional Tennis Matches,” Imperial College London, London, 2015.

[62] „AlphaGo,” [Online]. Available: https://deepmind.com/research/case-studies/alphago-the-story-so-far. [Data uzyskania dostępu: 6 Kwiecień 2021].

[63] A. Tch, „The mostly complete chart of Neural Networks, explained,” 4 Sierpień 2017. [Online]. Available: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464. [Data uzyskania dostępu: 7 Kwiecień 2021].

[64] G. F. M. I. E. P. A. L. Torre Ana, „Prediction of compression strength of high performance concrete using artificial neural networks,” *Journal of Physics Conference Series* , 2015.

[65] Y. Kinha, „How to Deal with Missing Values in Your Dataset,” [Online]. Available: https://www.kdnuggets.com/2020/06/missing-values-dataset.html. [Data uzyskania dostępu: 7 Kwiecień 2021].

[66] S. D. M. A. Knottenbelt William, „A common-opponent stochastic model for predicting the outcome of professional tennis matches,” *Computers & Mathematics with Applications.,* 2012.

[67] M. Chrzanowska, „Ekonometria Przestrzenna”.

[68] A. S. V, „Understanding Activation Functions in Neural Networks,” 30 Marzec 2017. [Online]. Available: https://medium.com/the-theory-of-everything/understanding-

activation-functions-in-neural-networks-9491262884e0. [Data uzyskania dostępu: 7 Kwiecień 2021].

[69] J. Brownlee, „A Gentle Introduction to Weight Constraints in Deep Learning,” Machine Learning Mastery, 23 Listopad 2018. [Online]. Available: https://machinelearningmastery.com/introduction-to-weight-constraints-to-reduce-generalization-error-in-deep-learning/. [Data uzyskania dostępu: 7 Kwiecień 2021].

[70] J. Brownlee, „Train Neural Networks With Noise to Reduce Overfitting,” Machine Learning Mastery, 12 Sierpień 2019. [Online]. Available: https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/. [Data uzyskania dostępu: 7 Kwiecień 2021].

Spis rysunków

Spis wzorów

Spis tabel

Wyrażam zgodę na udostępnienie mojej pracy w czytelniach Biblioteki SGGW w tym w Archiwum Prac Dyplomowych SGGW

.....

(czytelny podpis autora pracy)