Estymacja i generacja samopodobnych szeregów czasowych z wykorzystaniem sieci neuronowych

Autorzy: inż. Katarzyna Filus, inż. Anna Długosz
Konsultanci: mgr inż. Dariusz Marek, mgr inż. Jakub Szyguła

Studenckie Koło Naukowe Linuksa i Wolnego Oprogramowania
Politechnika Śląska w Gliwicach

1 Wstęp

Coraz więcej odpowiedzialności kładzie się na urządzenia podłączone do internetu. Krytyczny staje się problem niezawodności tych urządzeń i samych sieci komputerowych. Z tego powodu niezbędna jest poprawna estymacja właściwości ruchu sieciowego oraz jego jakości przesyłu. Charakter internetu zmienia się i na rok 2023 przewidywane jest, że połowę wszystkich urządzeń będą stanowiły połączenia maszyna do maszyny, co oznacza, że internet nie będzie już zdominowany przez międzyludzkie interakcje [4]. To wydarzenie postrzegane jest jako powód do powtórnego zbadania właściwości ruchu sieciowego [21].

Z drugiej strony, konieczna jest efektywna ocena właściwości ruchu sieciowego w celu odpowiedniego sterowania mechanizmami mającymi na celu zwiększenie wydajności sieci, a co za tym idzie spełnieniu wymagań QoS (ang. Quality of Service). Takim mechanizmem jest na przykład mechanizm aktywnego zarządzania kolejkami (ang. Active Queue Management, AQM). Mechanizm ten współpracuje z mechanizmem okien przeciążenia (ang. congestion window) adaptując intensywność przesyłania danych aby minimalizować szansę wystąpienia zatorów w sieci [7].

Tradycyjny ruch internetowy został już w dużym stopniu zbadany i znana jest jego natura. W latach dziewięćdziesiątych, kiedy rozpoczął się jego eksponencjalny wzrost, udowodniono, że rzeczywisty ruch w sieci internet nie może być modelowany za pomocą wykorzystywanych w tamtych czasach do modelowania sieci telefonicznych źródła ruchu, np. Poissonowskie i markowowskie o lekkich ogonach [20]. Naturę rzeczywistego ruchu w sieci internet cechuje samopodobieństwo, a także zachowania długo-ogonowe [20].

Stopień samopodobieństwa może być szacowany za pomocą wykładnika Hursta. Jest to miara przyjmująca wartości o 0 do 1. Jej szacowanie możliwe jest za pomocą różnych metod, m.in. Wariacji Zagregowanej, metody Beztrendowej Analizy Fluktuacyjnej (ang. Detrendend Fluctuation analysis, DFA), a także najstarszej metody - R/S. Każda z metod daje inną wartość estymowanej wartości wykładnika Hursta, a także czasem obliczeń. Metody te są niejednokrotnie bardzo czasochłonne i niepraktyczne dla zastosowań czasu rzeczywistego (np. we wspomnianych wcześniej mechanizmach AQM).

Dużą popularność w domenie sieciowej zdobyły rozwiązania bazujące na uczeniu maszynowym, w szczególności na sieciach neuronowych. Modele te mają olbrzymią moc odnajdywania wzorców w danych i dlatego są wykorzystywane do rozwiązywania szerokiej gamy problemów.: od klasyfikacji ruchu IP [2] do detekcji ataków sieciowych [19]. Do trenowania modeli sztucznych sieci neuronowych konieczne jest zgromadzenie dużej ilości danych. Dostęp do rzeczywistych śladów sieciowych jest ograniczony z powodów polityki prywatności [17]. Z tego powodu powszechne jest wykorzystanie generatorów samopodobnego ruchu sieciowego opartych na FGN (ang. Fractional Gaussian Noise) [12], a także możliwe jest zastosowanie generatorów opartych na rozkładzie Pareto [8].

Z powodu konieczności estymacji właściwości ruchu sieciowego (często w czasie rzeczywistym) oraz coraz większej popularności sieci neuronowych w domenie sieciowej oraz ich dużej zdolności wykrywania wzorców w wielowymiarowych danych w pracy stworzono i przetestowano zdolność sieci neuronowych do szacowania wartości parametru Hursta na danych oznaczonych za pomocą różnych metod klasycznych do estymacji wartości parametru Hursta. Takie podejście do problemu sprawia, że sieć neuronowa pełni rolę aproksymatora konkretnej metody do estymacji parametru Hursta (ale wartości danej funkcji przyjmują wartości dyskretne - przedziały wartości wykładnika Hursta, bo rozpatrywane zadanie to klasyfikacja). Dużą zaletą rozwiązania jest taka sama złożoność obliczeniowa dla wszystkich metod przy tej samej architekturze sieci neuronowej (różnią się tylko etykiety podczas treningu, a co za tym idzie wartości przewidywane dla nieznanych danych). Ponadto, sieci neuronowe małych rozmiarów cechują się szybkością predykcji, więc rozwiązanie jest bliższe zastosowaniom w czasie rzeczywistym. W pracy do trenowania sieci wykorzystywane są próbki samopodobne imitujące ruch sieciowy stworzone za pomocą samopodobnego generatora opartego na FGN. Takie podejście pozwala na wygenerowanie równej liczby próbek dla wszystkich klas treningowych, przez co nie ma konieczności zwalczania problemu niezbalansowania, na który sieci neuronowe są wrażliwe (dla ruchu rzeczywistego nie można zakładać, że uda się zebrać równa liczbę próbek dla wszystkich klas decyzyjnych. Wymagałoby to zebrania olbrzymich ilości ruchu, do którego dostęp, jak wcześniej wspomniano, jest ograniczony.).

2 Cel projektu

Głównym celem projektu było stworzenie sieci neuronowej, która klasyfikowałaby ruch sieciowy pod kątem stopnia jego samopodobieństwa. Docelowo model ma spełniać zadanie klasyfikatora danych pochodzących z rzeczywistego ruchu sieciowego. Na potrzeby projektu wykorzystano dane syntetyczne, które zostały wygenerowane za pomocą generatora. Z tego powodu konieczne było stworzenie takiego generatora. Dodatkowo zaimplementowano procedury pozwalające na estymację wartości parametru Hursta do oznaczenia próbek wykorzystywanych w eksperymentach. Jako dane reprezentujące ruch pakietów w sieci wykorzystano dane wygenerowane za pomocą generatora opartego na Fractional Gaussian Noise (FGN).

3 Samopodobieństwo ruchu sieciowego

Samopodobieństwo to termin wprowadzony przez Mandelbrota [13] w latach sześćdziesiątych dwudziestego wieku. Samopodobieństwo w matematyce oznacza, że niezależnie od skalowania fragment całości w sensie statystycznym odpowiada całości (czyli uzyskane w procesie skalowania fragmenty posiadają takie same właściwości statystyczne jak obraz całości) [13]. Przykładem tworów samopodobnych są dobrze znane fraktale. Także w naturze zjawisko samopodobieństwa jest bardzo powszechne. Sam B. Mandelbrot opisał samopodobieństwo na przykładzie obrazu linii wybrzeża [13].

Samopodobieństwo można także zaobserwować w szeregach czasowych. Stopnień samopodobieństwa w tym przypadku określa fakt występowania zależności długo- i krótkoterminowych oraz ich stopnień. Zależności długoterminowe są określane w literaturze jako LRD, ang. Long-Range Dependence, a krótkoterminowe SRD, ang. Short-Range Dependence. Procesy LRD dla różnych skali czasu posiadają podobną zmienność [8]. Zależności te zauważono już w połowie dwudziestego wieku, kiedy to sir H. E. Hurst opisał występowanie zależności długoterminowych na podstawie wartości wahań poziomu wody w Nilu [10].

W latach dziewięćdziesiątych dwudziestego wieku rozpoczął się eksponencjalny rozwój internetu. Początkowo jego natura była niezbadana, nie było dostępu do dużych ilości wysokiej jakości śladów ruchu sieciowego, a do modelowania ruchu wykorzystywano proste modele ruchu charakterystyczne dla sieci telefonicznych: markowowskie i poissonowskie [21]. Artykuł [20] wprowadził pierwszy zbiór danych opisujący ruch sieciowy (ruch pakietów w sieci LAN Ethernet) o wysokiej jakości, dużej ilości zebranych danych i wysokiej rozdzielczości. W artykule udowodniono także, że ruch ten jest statystycznie samopodobny i wykazuje właściwości długo-ogonowe [20] i wykazuje zależności długoterminowe przez dużą liczbę skali czasu.

Na cześć wcześniej wspomnianego H. E. Hursta stopień samopodobieństwa jest określany za pomocą wykładnika/parametru Hursta. Parametr ten przyjmuje wartości od 0 do 1 i spełnia następujące właściwości:

- wartość parametru w przedziale $\langle 0; 0.5)$ oznacza występowanie zależności krótkoterminowych
- wartość 0.5 oznacza brak zależności w ogóle
- wartość parametru w przedziale $(0.5; 1.0\rangle$ oznacza występowanie zależności długoterminowych

Szacowanie wartości parametru Hursta jest możliwe za pomocą różnych metod:

- metody R/S
- metody Zagregowanej Wariancji
- metody Beztrendowej Analizy Fluktuacyjnej (ang. Detrended Fluctuation analysis, DFA)
- metody falkowej
- metody wykorzystującej lokalny estymator Whittle'a.

Na potrzeby pracy wykorzystano pierwsze trzy metody.

Metoda R/S - jest to miara statystyczna wprowadzona przez H.E. Hursta i określa stopień zmienności funkcji [10]. Jest to zarazem najstarsza metoda szacowania wartości wykładnika Hursta. Dla serii czasowej $X = X_1, X_2, .., X_n$, gdzie n to długość serii czasowej wartość estymuje się następująco: najpierw należy wyznaczyć wartość średnią m dla szeregu czasowego , później stworzyć serię wartości skorygowanych średnią [16]:

$$Y_t = X_t - m, \quad t = 1, 2, ..., n \tag{1}$$

, następnie należy wyznaczyć serię zawierająca wartości sumy skumulowanej Z_t i zakres danych zagregowanych w blokach R:

$$R(n) = max(Z_1, Z_2, ..., Z_n) - min(Z_1, Z_2, ..., Z_n) \tag{2}$$

oraz obliczyć wartości odchylenia standardowego:

$$S(n) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - m)^2} \tag{3}$$

Na sam koniec wyznacza się iloraz $R(n)/S(n)$ i otrzymane wyniki wykreśla się w funkcji czasu t w skali podwójnie logarytmicznej, dopasowuje się do niej prostą metoda najmniejszych kwadratów, a przybliżenie wartości parametru Hursta oblicza się jako wartość współczynnika nachylenia tej prostej.

Metoda DFA - jest to także metoda statystyczna, która jest wykorzystywana do badania samopodobieństwa serii czasowej. Duża zaletą tej metody jest fakt, że może być ona stosowana także do procesów niestacjonarnych. Wyznaczona wartość estymowana α to uogólnienie parametru Hursta, ale w przedziale od 0 do 1 pokrywa się z wartością parametru Hursta ($\alpha = 1$ oznacza szum różowy, a wartości większe od 1 - proces niestacjonarny). Dla serii czasowej x_t o długości N, gdzie $t \in \mathbb{N}$, należy [15] wyznaczyć serię zawierającą wartości sumy skumulowanej X_t. Później X_t dzieli się na przedziały zawierające n próbek i dla każdego z przedziałów dopasowuje się prostą za pomocą metody najmniejszych kwadratów. Następnie wyznacza się pierwiastek z kwadratu średniej błędu, czyli fluktuację, opisywaną wzorem:

$$F(n) = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(X_t - Y_t)^2} \tag{4}$$

Dla różnych długości okna interpolacji proces powtarza się i wykreśla w podwójnej skali logarytmicznej wykres $F(n)$ od n [1]. Dopasowana do tego wykresu linia prosta służy do wyznaczenia estymowanej wartości parametru (wartość nachylenia prostej).

Metoda Zagregowanej Wariancji - w tej metodzie wykładnik Hursta jest szacowany na podstawie własności, że szereg czasowy stacjonarny (długość N) wykazujący zależności długoterminowe, charakteryzuje się uśrednioną wariancją

próbek rzędu N^{2H-2} [8]. Szereg czasowy dzieli się na na bloki o długości m dla $m \in (2; \frac{N}{2})$ i oblicza dla nich średnią $\overline{X}^{(m)}(k)$, k oznacza numer bloku. Następnie dla każdego m wyznacza się wariancję procesu uśrednionego $\overline{X}(k)$ i później wykreślić w skali podwójnie logarytmicznej wykres σ^2 od m i dopasować do niego prostą. Dla wystarczająco dużej wartości m nachylenie prostej jest równe $2H - 2$, więc:

$$H = \frac{a+2}{2}, \tag{5}$$

gdzie a oznacza współczynnik nachylenia dopasowanej prostej.

Opisane metody różnią się działaniem i wyznaczone przez nie wartości parametru Hursta zazwyczaj różnią się. Różnią się także ich czasy obliczań.

W dziedzinie sieciowej coraz częściej wykorzystywane są rozwiązania bazujące na algorytmów uczenia maszynowego, a w szczególności na sztucznych sieciach neuronowych. Jest to metoda zainspirowana biologicznymi sieciami neuronowymi występującymi w organizmach ssaków. Zazwyczaj sieci neuronowe składają się z warstw: wejściowej, wyjściowej i warstw ukrytych, które poprzez nieliniowe funkcje aktywacji poszczególnych neuronów mapują dane wejściowe danej warstwy do innej przestrzeni cech. Modele z jedną warstwą ukrytą to przykłady uczenia płytkiego (są to także algorytmy takie jak np. KNN i SVM) , a gdy warstw jest więcej, mowa o uczeniu głębokim. Sieci neuronowe to bardzo rozległa dziedzina, która obejmuje różnorodne sieci, warstwy, algorytmy optymalizacyjne i funkcje aktywacji neuronów.

Ruch sieciowy opisywany jest często w formie szeregów czasowych (tak tez założono w pracy), a do ich przetwarzania wykorzystuje się zazwyczaj warstwy rekurencyjne: LSTM (ang. Long-Short Term Memory) oraz GRU (ang. Gated Recurrent Unit). Pojedyncze warstwy LSTM rozwiązują problem zanikającego gradientu charakterystycznego dla prostych warstw rekurencyjnych i są w stanie propagować gradient przez dłuższe jednostki czasu [11]. Ich cechą charakterystyczna jest fakt, że przechowują one stan wewnętrzny, który pozwala im na „pamiętanie" informacji z przeszłości [5]. Z tego powodu ich „pamięć" jest dłuższa niż dla prostych warstw rekurencyjnych i są przystosowane do badania zależności długoterminowych. Alternatywą dla tych warstw są warstwy GRU, które działają w bardzo podobny sposób, ale ze względu na ich prostszą implementację i obliczenia są bardziej efektywne niż warstwy LSTM [9].

Kolejnym rodzajem warstwy odpowiednim do przetwarzania danych serii czasowej jest warstwa konwolucyjna (jednowymiarowa). W tym wypadku czas jest traktowany jako wymiar przestrzenny [3]. Jest to szybka alternatywa dla warstw rekurencyjnych. w takiej sieci dane serii czasowej o zmienionym wymiarze (aby zapewnić wspomniane wcześniej traktowanie czasu jako wymiaru przestrzennego) są przetwarzane na zmianę przez warstwy konwolucyjne i warstwy „pooling" (maja one na celu zmniejszenie wymiarowości). Jako rezultat, tworzone są bardziej abstrakcyjne reprezentacje danych. Na końcu modelu znajduje się częścią klasyfikującą, którą jest klasyczny perceptron wielowarstwowy (sieć, w której neurony w następujących po sobie warstwach są gęsto połączone).

W procesie treningu sieci neuronowych potrzebne są duże ilości danych, najlepiej zbalansowanych pod względem liczby próbek pomiędzy klasami. Dostęp do rzeczywistych śladów ruchu czasowego jest ograniczony ze względu na prywatność [17]. Ponadto, w badanych problemie zależy nam na uzyskaniu równej liczby próbek dla konkretnych wartości przedziałów parametru Hursta. Korzystając z gotowych baz nie mamy wpływu na wartości parametru Hursta jakie przyjmują serie czasowe stworzone na podstawie zawartych w nich śladów. z tego powodu dla badanego problemu naturalnym podejściem jest skorzystanie z generatorów ruchu samopodobnego i wygenerowanie zbiorów treningowych, ponieważ pozwala to na wygenerowanie zbalansowanych zbiorów treningowych o dużej liczbie próbek.

Do generacji przebiegów samopodobnych można wykorzystać różne metody:

- FGN (ang. Fractional Gaussian Noise) - gaussowski proces dokładnie samopodobny, który jest często wykorzystywany do ewaluacji wydajności sieci [6]. Jeśli zdefiniujemy $B_H(t)$ jako proces FBM (ang. Fractional Brownian Motion), to proces zinkrementowany $X(t) = B_H(t+1) - B_H(t)$ nazywany jest FGN. Warunkiem wystarczającym dla występowania samopodobieństwa drugiego stopnia jest postać funkcji autokorelacji tego procesu definiowana jako: [6]:

$$\rho^{(m)}(k) = \rho(k) = \frac{1}{2}[(k+1)^{2H} - 2k^{2H} + (k-1)^{2H}], \tag{6}$$

- Rozkład Pareto - jest to rozkład, który także można wykorzystywać do modelowania ruchu samopodobnego [8] i można go parametryzować za pomocą „ciężkości" ogona, która to jest związana z wartością parametru Hursta [14]
- Klasę procesów FARIMA (ang. Fractional Autoregressive Integrated Moving-Average) - uogólnienienie modeli ARMA. Są to procesy opisywane przez trzy parametry, dwa wspólne z ARMA plus jeden dodatkowy, zależny od wartości parametru Hursta[18]:

$$d = H - \frac{1}{2} \tag{7}$$

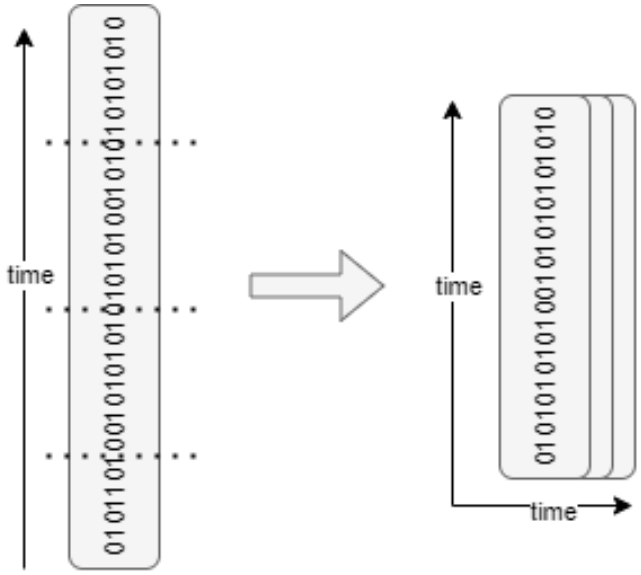
4 Metodologia uzyskania danych uczących

W celu generacji próbek treningowych zaimplementowano funkcję generującą sztuczny ruch sieciowy z wykorzystaniem FGN (generator zbudowano w oparciu o funkcję z biblioteki języka Python - stochastic - i odpowiednio przetworzono uzyskane za jej pomocą dane). Wygenerowanie serii z wykorzystaniem rozkładu FGN wymaga w pierwszej kolejności inicjalizacji generatora o argumentach: zadanej wartości wykładnika Hursta oraz długości próbki, a następnie stworzenia próbki i określeniu algorytmu, za pomocą którego zostanie wygenerowana. Na potrzeby projektu wykorzystano algorytm Daviesa-Harte'a.

Wygenerowane dane są zapisywane do plików tekstowych. Dane z poszczególnych plików (o zadanych wartościach wykładnika Hursta) są następnie łączone w jeden plik. Dane uczące oraz odpowiadające im etykiety są przechowywane w

osobnych plikach tekstowych i wczytywane tuż przed rozpoczęciem procesu trenowania sieci neuronowych. Serie czasowe są reprezentowane przez ciągi jedynek i zer (1 oznacza nadejście pakietu, 0 jego brak). Dane w plikach z etykietami znajdują się na tych samych pozycjach co odpowiadające im próbki i oddzielone znakiem nowej linii. Etykiety to wartości współczynnika Hursta pomnożone przez sto i zaokrąglone do wartości całkowitych. Przyjmują wartości z zakresu (0 - 93).

Dane treningowe wczytane z plików tekstowych są w postaci jednowymiarowych wektorów. Aby możliwe było podanie ich na wejście sieci konwolucyjnych, próbki należy przekształcić tak, aby reprezentowały dane w czasie jako wymiarze przestrzennym. Praktycznie, należy zmienić kształt badanych próbek do postaci dwuwymiarowej (na potrzeby projektu dane wczytywano do tablic biblioteki NumPy i wykorzystywano funkcję *reshape*). W eksperymentach zbadano wpływ przekształcenia próbek do różnych wymiarów. Na potrzeby podziału danych na treningowe i testowe wykorzystywana jest funkcja *train_test_split*, która znajduje się w module *model_selection* biblioteki *sklearn*.



Rysunek 1: Proces przekształcenia próbek wejściowych

4.1 Metody estymacji Hursta

W celu oznaczenia próbek treningowych dla sieci neuronowych zaimplementowano trzy metody estymacji wartości parametru Hursta. Metody zastosowane na potrzeby projektu:

– Metoda wariancji zagregowanej - algorytm wyznaczania wartości parametru Hursta tą metodą polega na obliczeniu wartości wariancji średniej dla próbek zagregowanych o różnej skali. Do otrzymanego wykresu w skali podwójnie logarytmicznej wariancji od skali dopasowuje się prostą za pomocą funkcji polyfit z biblioteki NumPy. Współczynnik Hursta jest równy połowie zwróconej przez funkcję wartości (jako 1 element), która jest nachyleniem prostej dopasowanej.
– Metoda R/S - do wyznaczenia wartości wykładnika Hursta tą metodą wykorzystywana jest funkcja *RS*. Do wyznaczenia współczynnika nachylenia prostej dopasowanej do wykresu stworzonego przy pomocy tej metody wykorzystano wcześniej wspomnianą funkcję *numpy.polyfit*. Wartość współczynnika Hursta jest równa wartości współczynnika nachylenia prostej.
– Metoda beztrendowej analizy fluktuacyjnej - do wyznaczenia wartości wykładnika Hursta tą metodą wykorzystywana jest funkcja, która korzysta głównie z funkcji matematycznych z biblioteki NumPy. Do wyznaczenia współczynnika nachylenia prostej (wartość parametru Hursta równa jest jej nachyleniu) dopasowanej do wykresu fluktuacji od rozmiaru okna interpolacji w skali podwójnie logarytmicznej wykorzystywana jest także funkcja *numpy.polyfit*).

4.2 Szkolenie sieci neuronowych

Do stworzenia sieci neuronowych na potrzeby eksperymentów skorzystano z biblioteki Keras i języka Python. W tym celu wykorzystano warstwy konwolucyjne (jednowymiarowe) - *Conv1D* oraz dodatkowe warstwy (m.in. *Dropout*, *BatchNormalization*, *AvgPool1D*) oraz warstwy ściśle połączone - *Dense* (w finalnym klasyfikatorze). Wykorzystano funkcję ReLU jako funkcję aktywacji i porównano wyniki uzyskane dla różnych optymalizatorów funkcji kosztu (wykorzystana funkcja kosztu: *categorical_crossentropy*). Przykładową implementację sieci z dwiema warstwami konwolucyjnymi przedstawiono na Listingu 1.1. W celu polepszenia wyników trenowania skorzystano z mechanizmu kontroli procesu trenowania (moduł *keras.callbacks* - *ReduceLROnPlateau*. Wywołanie tej metody przedstawiono na 1.2.

Listing 1.1: Przykładowy model sekwencyjny konwolucyjnej sieci neuronowej zaimplementowany za pomocą biblioteki Keras

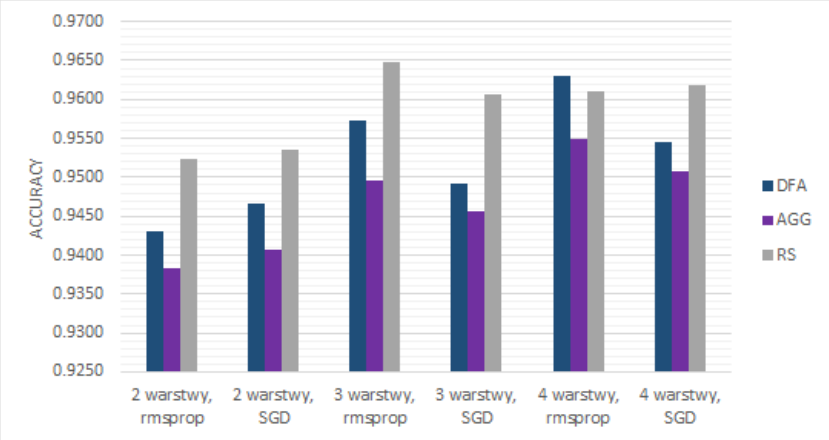
```
1     network = models.Sequential()
2
3     # pierwsza warstwa konwolucyjna
4     network.add(keras.layers.Conv1D(64, 10, activation=
          'relu', padding='same',
5                                     input_shape=(
                                          x_train.shape
                                          [1:])))
6     network.add(BatchNormalization())
7     network.add(keras.layers.AvgPool1D(3))
```

```
8       network.add(Dropout(0.25))
9
10      # druga warstwa konwolucyjna
11      network.add(keras.layers.Conv1D(128, 10, activation
            ='relu', padding='same'))
12      network.add(BatchNormalization())
13      network.add(keras.layers.GlobalAvgPool1D())
14      network.add(Dropout(0.25))
15
16      # klasyfikator finalny
17      network.add(Dense(128, activation='relu'))
18      network.add(Dropout(0.4))
19      network.add(Dense(9, activation='softmax'))
```

Listing 1.2: Zastosowanie mechanizmu biblioteki Keras - callbacks

```
1       callbacks_list = [keras.callbacks.ReduceLROnPlateau
            (monitor='val_loss', factor=0.1, patience=3)]
```

5 Uzyskane wyniki



Rysunek 2: Uzyskane średnie wyniki dokładności klasyfikacji dla przeprowadzonych eksperymentów na danych oznaczonych trzema metodami estymacji Hursta (RS, Agregated Variance i DFA) oraz sieci neuronowych o różnych rozmiarach i optymalizatorach wykorzystywanych w procesie uczenia (RMSProp i SGD), dla danych wejściowych o wymiarach: 1000x10

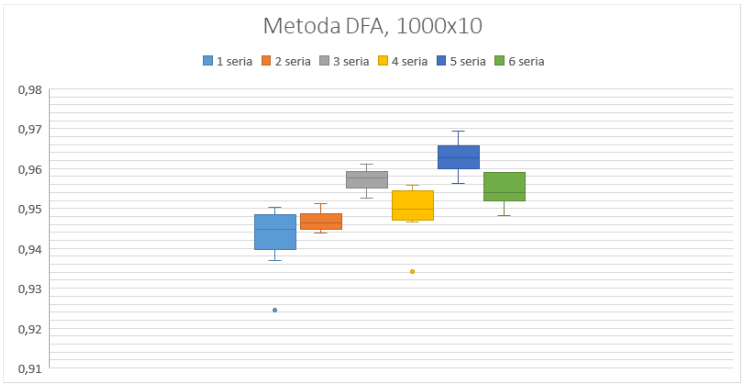
Uzyskane średnie rezultaty dokładności klasyfikacji sieci neuronowych dla danych wejściowych o wymiarach 1000x10 przedstawiono na Rys. 2. W badaniach uwzględniono różne liczby warstw ukrytych oraz zbadano wpływ zastosowanych optymalizatorów na otrzymywane rezultaty. Brano pod uwagę Stochastyczny Spadek Gradientowy (SGD) oraz RMSProp. Najniższe miary dokładności uzyskano w przypadku zastosowania 2 warstw RMSProp (93,8%), dla danych etykietowanych za pomocą metody zagregowanej wariancji (AGG). Natomiast najwyższe średnie wartości zaobserwowano w przypadku 3 warstw RMSProp (96,5 %), dla danych przygotowanych za pomocą metody R/S.

Eksperymenty przeprowadzone w czasie trwania realizacji projektu zostały wielokrotnie powtórzone w celu uzyskania jak najbardziej wiarygodnych rezultatów. Szczegółowy zakres uzyskanych wyników dla danych wejściowych o wymiarach 1000x10, przedstawiono na rysunku 3, z wykorzystaniem wykresów pudełkowych. Podpunkt 3a ilustruje rezultaty dla danych etykietowanych metodą DFA. Najniższe pojedyncze wartości na poziomie 92,5 % zaobserwowano w 1 serii (oznaczonej kolorem jasnoniebieskim na wykresie), natomiast najwyższe wartości dokładności klasyfikacji modelu uzyskano w 5 serii (kolor ciemnoniebieski), gdzie odnotowano 96,9 % dokładności klasyfikacji. Rezultaty dla danych etykietowanych metodą zagregowanej wariancji (AGG) przedstawia podpunkt 3b. W tym przypadku dokładność predykcji modelu mieściła się w zakresie od 92 % (uzyskane w 1 serii) do 95,8 % (dla serii 5). Spośród wybranych do badań metod do etykietowania danych najlepsze średnie wartości uzyskano dla metody R/S (podpunkt 4c). Najniżej odnotowany rezultat to 94,9 % (w 1 serii), a najwyższa dokładność klasyfikacji przekroczyła 96,6 % w przypadku 3 serii.

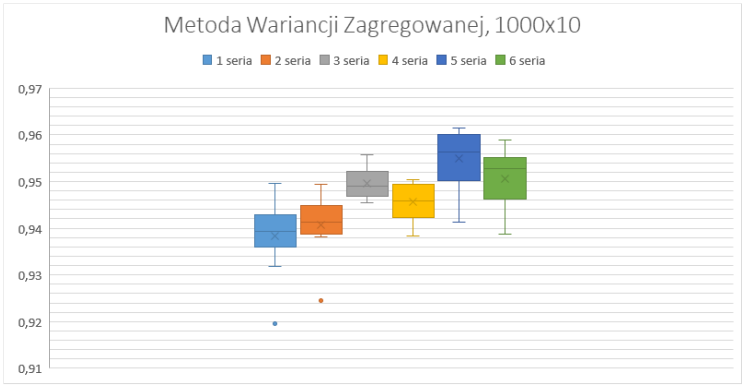
Analogiczny zestaw badań, przeprowadzono dla danych wejściowych o wymiarach 500x20. Wyniki przedstawiono na rysunku 4. W przypadku danych etykietowanych metodą DFA (podpunkt 4a), najniżej uzyskany rezultat dokładności klasyfikacji to 83 % (seria 1). Najwyższą wartość odnotowano w serii 5 na poziomie 91,9 %. Dla danych o wymiarach 500x20 były to zarazem najlepsze uzyskane średnie wartości dokładności klasyfikacji.

Podpunkt 4b przedstawia rezultaty dla danych etykietowanych metodą zagregowanej wariancji (AGG). W przypadku tej metody zaobserwowano rezultaty w zakresie od 85,5 % (w przypadku serii 1) do 90,8 % (w serii 5). W tym przypadku najniższe średnie wartości uzyskano dla metody R/S (podpunkt 4c). Najniższa uzyskana wartość to 79,8 % (w serii 4), a najwyższa to 85,5 % (seria 1).

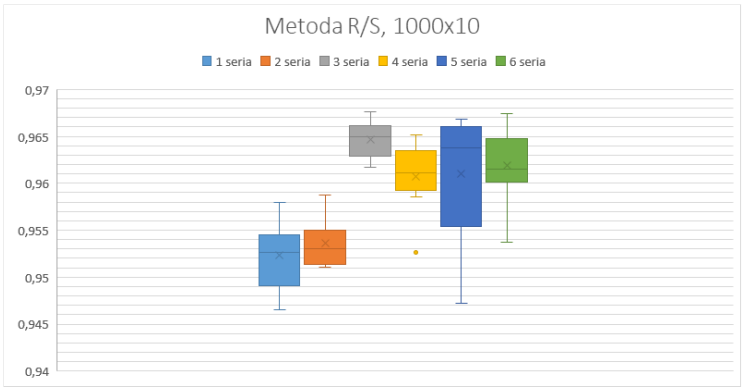
W celu lepszego zobrazowania opisanych powyżej wyników eksperymentów badawczych oraz uwidocznienia różnic, jakie wynikały z zastosowania danych o wymiarach 1000x10 oraz 50x20, przygotowano Rys. 5. Podpunkt 5c obrazuje jak duże różnice w dokładności klasyfikacji uzyskano w przypadku danych etykietowanych metodą R/S. Odnotowano spadek dokładności klasyfikacji nawet na poziomie 15 % w przypadku danych o wymiarach 50x200 dla modelu z 4 warstwami SGD. W przypadku metody DFA (podpunkt 4a) oraz metody zagregowanej wariancji (AGG), podpunkt 4b, różnice te były zdecydowanie mniejsze oraz były na stałym poziomie - ok. 4%. Lepsze rezultaty uzyskano dla danych o wymiarach 1000x10.



(a)

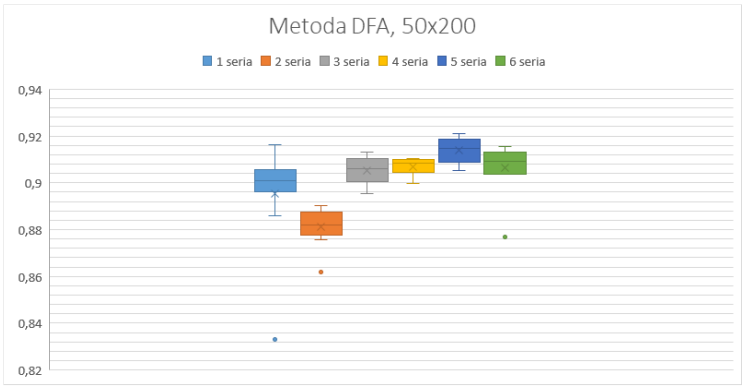


(b)

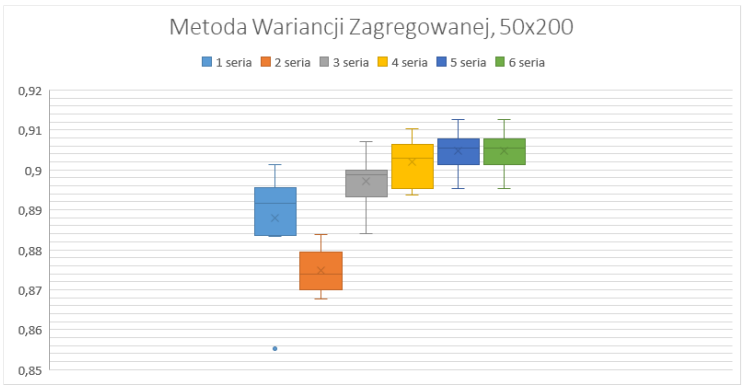


(c)

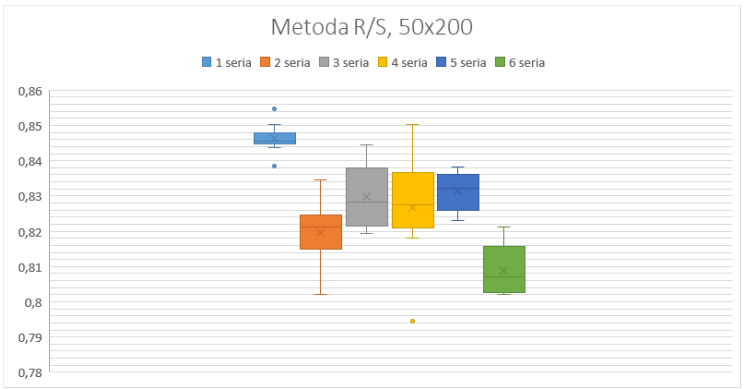
Rysunek 3: Porównanie wykresów pudełkowych przedstawiających wyniki dokładności modeli sieci neuronowych dla danych wejściowych o wymiarach 1000x10 o różnej liczbie warstw ukrytych, różnych optymalizatorach wykorzystanych w treningu (Stochastyczny Spadek Gradientowy oraz RMSProp) i różnych metodach szacowania wartości parametru Hursta wykorzystanych do etykietowania danych: (a) DFA, (b) Metoda Wariancji Zagregowanej, (c) Metoda R/S



(a)

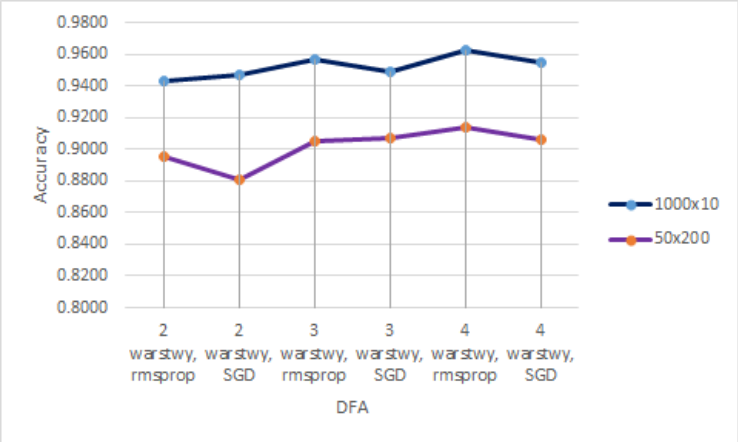


(b)

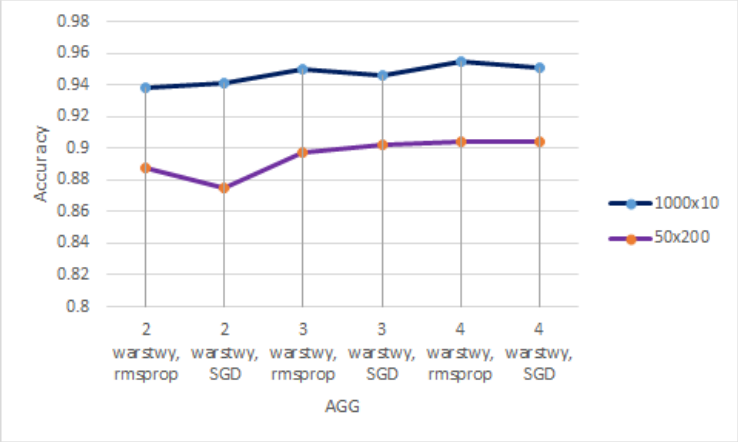


(c)

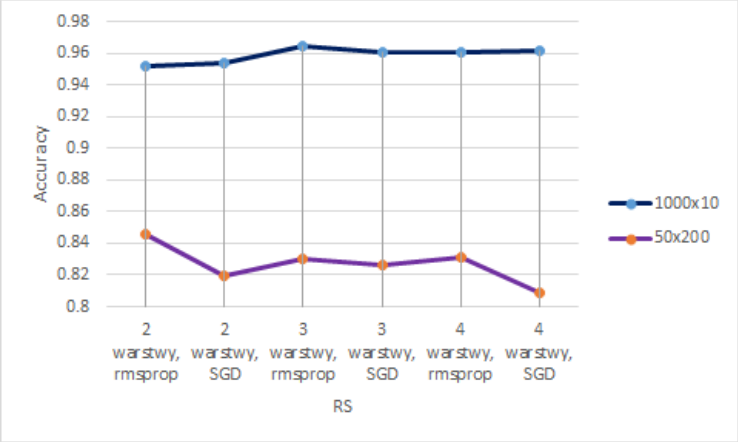
Rysunek 4: Porównanie wykresów pudełkowych przedstawiających wyniki dokładności modeli sieci neuronowych dla danych wejściowych o wymiarach 500x20 o różnej liczbie warstw ukrytych, różnych optymalizatorach wykorzystanych w treningu (Stochastyczny Spadek Gradientowy oraz RMSProp) i różnych metodach szacowania wartości parametru Hursta wykorzystanych do etykietowania danych: (a) DFA, (b) Metoda Wariancji Zagregowanej, (c) Metoda R/S



(a)



(b)



(c)

Rysunek 5: Porównanie dokładności modeli sieci neuronowych dla danych wejściowych o wymiarach 1000x10 oraz 50x200, a także o różnej liczbie warstw ukrytych, różnych optymalizatorach wykorzystanych w treningu (Stochastyczny Spadek Gradientowy oraz RMSProp, różnych metodach szacowania wartości parametru Hursta wykorzystanych do etykietowania danych: (a) DFA, (b) Metoda Wariancji Zagregowanej, (c) Metoda R/S

6 Podsumowanie

Proces realizacji projektu składał się z trzech głównych etapów, które przestawione zostały jako kamienie milowe: dobór odpowiednich metod generowania ciągów samopodobnych oraz wygenerowania danych uczących, rozważania dotyczące koncepcji architektury sieci neuronowych które realizują postawione cele oraz dokonanie oceny zaproponowanych rozwiązań.

Każdy z etapów realizacji projektu był bardzo ważny dla projektu. Rezultatem było stworzenie modeli uczenia maszynowego o dużej dokładności.

Celem projektu było stworzenie klasyfikatora ruchu sieciowego na podstawie stopnia jego podobieństwa. Jego realizacja, oprócz implementacji mechanizmów sieci neuronowych, wymagała również stworzenia oprogramowania koniecznego do przetwarzania i etykietowania danych wykorzystanych w eksperymentach. Z tego powodu konieczne było zapoznanie się z bibliotekami zawierającymi metody statystyczne oraz złożone funkcje matematyczne.

Stworzone na potrzeby projektu sieci neuronowe i mechanizmy ich uczenia zaimplementowano za pomocą modułów biblioteki Keras, która jest biblioteką często wykorzystywaną w rozwiązaniach komercyjnych ze względu na wygodę użytkowania oraz łatwe wdrażanie modeli. Z drugiej strony stworzenie dobrego modelu sieci neuronowej o wysokiej dokładności, wymagało uzyskania dużej wiedzy dotyczącej wszystkich warstw zawartych w sieci, interakcji między nimi oraz poznania mechanizmów pozwalających na efektywne trenowanie i testowanie modeli. Warte podkreślenia jest, że samo napisanie kompilującego się i działającego kodu nie oznacza jeszcze sukcesu. Koniecznym jest, by poszczególne składniki sieci neuronowej właściwie ze sobą współpracowały oraz były odpowiednio dobrane do rozwiązywanego problemu. Zadanie to okazało się więc złożone i wymagające poświęcenia dużej ilości czasu na analizę badanej problematyki. Jednak zdobyta w ten sposób wiedza, umiejętności i doświadczenie z pewnością będzie przynosić efekty również w kolejnych projektach.

Często spotykanym problemem w rozwiązaniach wykorzystujących metody uczenia maszynowego jest niska i niewystarczająca dokładność. Badania literaturowe oraz analiza opisanych tam metod dostarcza jednak pewne zasady jak postępować z różnego typu warstwami, architekturami sieci itd. Jako przykład można podać zwiększanie liczby warstw w badanej sieci oraz rozmiarów warstw. Jednakże, kluczem do stworzenia wydajnych i dokładnych modeli sieci neuronowych dla danej problematyki są liczne eksperymenty wykonane z różnymi optymalizatorami, rozmiarami sieci. W projekcie zbadano wpływ doboru różnych optymalizatorów i liczby warstw ukrytych na wynik trenowania. Pozwoliło to na uzyskanie dokładności predykcji dla architektury konwolucyjnej sieci neuronowej na poziomie przekraczającym 96 %, co uznano za satysfakcjonujący wynik. Projekt ten dowodzi, że sztuczne sieci neuronowe są w stanie wykrywać stopień samopodobieństwa, przez co można je stosować do estymacji wartości parametru Hursta.

Literatura

1. Bryce, R., Sprague, K.: Revisiting detrended fluctuation analysis. Scientific reports 2, 315 (March 2012)
2. Chen, Z., He, K., Li, J., Geng, Y.: Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks. In: 2017 IEEE International conference on big data (big data). pp. 1271–1276. IEEE (2017)
3. Chollet, F.: Deep Learning with Python . Manning (2017)
4. Cisco annual internet report (2018–2023). [online] (Cisco 2020), dostępne: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf [Dostęp: 2020-12-4]
5. D'Angelo, G., Palmieri, F.: Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features extraction. Journal of Network and Computer Applications 173, 102890 (2021)
6. Domańska, J., Domański, A., Czachórski, T.: Estimating the intensity of long-range dependence in real and synthetic traffic traces. In: Computer Networks. Springer International Publishing. vol. 522, pp. 11–22 (06 2015)
7. Domańska, J., Domański, A.: Adaptive red in aqm. In: International Conference on Computer Networks. pp. 174–183. Springer (2009)
8. Domańska, J.: Markowowskie modele natężenia przesyłów internetowych. Instytut Informatyki Teoretycznej i Stosowanej Polskiej Akademii Nauk (2014)
9. Fu, R., Zhang, Z., Li, L.: Using lstm and gru neural network methods for traffic flow prediction. In: 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC). pp. 324–328. IEEE (2016)
10. Hurst, H.E.: Long-term storage capacity of reservoirs. Transactions of American Society of Civil Engineers 55(1), 400–414 (1951)
11. Li, J., Liu, C., Gong, Y.: Layer trajectory lstm. arXiv preprint arXiv:1808.09522 (2018)
12. Li, M.: Fractional gaussian noise and network traffic modeling. In: Proceedings of the 8th International Conference on Applied Computer and Applied Computational Science (2009)
13. Mandelbrot, B.: How long is the coast of britain? statistical self-similarity and fractional dimension. Science 156(3775), 636–638 (May 1967)
14. Ntlangu, M.B.: Modelling Computer Network Traffic Using Wavelets and Time Series Analysis. Master's thesis, University of Cape Town (2019)
15. Peng, C.K., Buldyrev, S., Havlin, S., Simons, M., Stanley, H., Goldberger, A.: Mosaic organization of dna nucleotides. Physical review. E, Statistical physics, plasmas, fluids, and related interdisciplinary topics 49, 1685–9 (March 1994)
16. Qian, B., Rasheed, K.: Hurst exponent and financial market predictability. In: IASTED conference on Financial Engineering and Applications (FEA 2004). p. 203–20 (2004)
17. Ring, M., Schlör, D., Landes, D., Hotho, A.: Flow-based network traffic generation using generative adversarial networks. Computers & Security 82, 156–172 (2019)
18. Rose, O.: Estimation of the hurst parameter of long-range dependent time series. Tech. rep., University of Wuerzburg, Institute of Computer Science (February 1996)
19. Saharkhizan, M., Azmoodeh, A., Dehghantanha, A., Choo, K.K.R., Parizi, R.M.: An ensemble of deep recurrent neural networks for detecting iot cyber attacks using network traffic. IEEE Internet of Things Journal 7(9), 8852–8859 (2020)
20. Willinger, W., Leland, W., Taqqu, M.: On the self-similar nature of traffic. IEEE/ACM Transactions on Networking (February 1994)

21. Willinger, W., Taqqu, M.S., Wilson, D.V.: Lessons from" on the self-similar nature of ethernet traffic". ACM SIGCOMM Computer Communication Review 49(5), 56–62 (2019)