WYDZIAŁ ELEKTRYCZNY
KIERUNEK: ELEKTROTECHNIKA
Specjalność:

Rafał Zaręba
Nr albumu: 238657

Praktyczna implementacja sztucznych sieci neuronowych w celu diagnostyki aparatury przeciwprzepięciowej w czasie rzeczywistym

Practical implementation of artificial neural networks for the purpose of real-time surge protectors *diagnostics*

INŻYNIERSKA PRACA DYPLOMOWA

Studia: niestacjonarne
Opiekun: **Dr inż. Paweł Kostyła**
Katedra K38W05D02

....
ocena *data, podpis opiekuna*

Wrocław, 2020

Spis treści

1. WSTĘP

W dobie powszechnego dostępu do zasobów komputerowych wiele gałęzi gospodarki uległo cyfryzacji. Wyjątkiem nie jest także cały sektor elektroenergetyczny. Niezawodność jego elementów, dobrze zaprojektowana automatyzacja oraz umiejętna diagnostyka działających w nim urządzeń jest kluczowa dla poprawnego funkcjonowania całego systemu. Wszechobecność wszelkiego rodzaju aparatury pomiarowej oraz czujników, nierzadko komunikujących się ze sobą powoduje, że w każdej chwili generowane są ogromne ilości danych. Rozwój technologii komputerowych umożliwił nie tylko ich łatwe pozyskanie, lecz także niemal nieograniczone i nieprzerwane gromadzenie. Podczas gdy dane same w sobie okazują się być bardzo wartościowe, istnieje duża potrzeba rozwijania nowoczesnych technik ich przetwarzania. Dopiero odpowiednio przygotowane i ustrukturyzowane dane mogą wspomóc wiele procesów technologicznych i biznesowych. Coraz częściej wykorzystuje się w tym celu algorytmy uczenia maszynowego i sztuczne sieci neuronowe.

Szczególnie interesującą i innowacyjną dziedziną uczenia maszynowego, która prężnie rozwija się na fali zainteresowania sztucznymi sieciami neuronowymi jest detekcja anomalii (ang. anomaly detection). Obejmuje między innymi takie zagadnienia jak wykrywanie podejrzanych transakcji płatniczych (ang. fraud detection), detekcję niebezpiecznych zdarzeń w ruchu internetowym, diagnozę chorób, czy wykrywanie nadchodzących nieprawidłowości w działaniu urządzeń lub maszyn, których parametry mierzone są w czasie rzeczywistym.

Zagadnieniem wykorzystania sztucznych sieci neuronowych w celu diagnostyki urządzeń w czasie rzeczywistym oraz przykładem praktycznej implementacji zajęto się w niniejszej pracy. Jako badane urządzenie wykorzystano warystor tlenkowo-cynkowy spełniający rolę ogranicznika przepięć w systemie elektroenergetycznym.

2. CEL I ZAKRES PRACY

2.1. Cel pracy

Celem pracy jest zaprojektowanie i praktyczna implementacja modelu uczenia maszynowego opartego o architekturę sztucznych sieci neuronowych. Zadaniem modelu jest monitorowanie stanu technicznego warystora tlenkowo-cynkowego w czasie rzeczywistym na podstawie odczytanych próbek napięcia oraz natężenia prądu.

2.2. Architektura modelu

Model zbudowany został w oparciu o architekturę głębokiego autoenkodera z rekurencyjnymi warstwami LSTM. Detekcja występujących w urządzeniu anomalii oparta jest o błąd rekonstrukcji sekwencji danych wejściowych.

2.3. Użyte narzędzia i technologie

Do realizacji projektu użyto wysokopoziomowego języka programowania Python w wersji 3.8. Do przygotowania i eksploracji danych użyto zoptymalizowanych bibliotek do obliczeń numerycznych – numpy, pandas, matplotlib, a sam model sieci neuronowej został napisany w bibliotece tensorflow w wersji 2.1.

3. WSTĘP TEORETYCZNY

3.1. Konserwacja predykcyjna

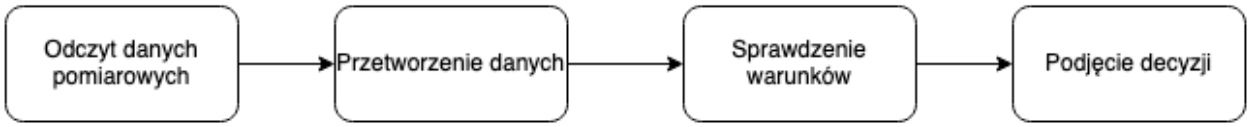
Wraz ze przyrostem obecności nowej technologii, konserwacja i utrzymanie ruchu odgrywają coraz ważniejszą rolę w wielu gałęziach przemysłu. W elektroenergetyce niezawodność i poprawność działania są szczególnie istotne, ponieważ nawet drobna awaria może skutkować nieocenionymi stratami, a niekiedy i groźnymi w skutkach katastrofami. By temu zapobiec stan poszczególnych elementów systemu musi być regularnie badany. Wszystkie naprawy i konserwacje powinny być wykonywane na czas, uprzedzając awarie i w konsekwencji zniszczenie urządzeń i maszyn.

Głównym celem jest wypracowanie oszczędności związanych z wyeliminowaniem zbędnych interwencji serwisowych. Jednakże oprócz wymiernych korzyści materialnych, odpowiednia konserwacja pozwala na zwiększenie bezpieczeństwa, szczególnie w sytuacjach, gdzie awaria danego komponentu może skutkować zagrożeniem życia lub zdrowia.

Tradycyjna metoda konserwacji maszyn i urządzeń opiera się w dużej mierze na eliminowaniu zaistniałych usterek po fakcie. Zazwyczaj uzupełniana jest o wykonywanie przeglądów w ściśle określonych terminach, tworząc tym samym rodzaj konserwacji prewencyjnej (zapobiegawczej). Okresy te ustalane są wykorzystując wiedzę ekspercką i niekiedy uzupełniane są o stosowne adnotacje, jak skrócenie czasu pomiędzy kolejnymi przeglądami wraz z przebiegiem maszyny lub założenie, że po określonym czasie maszyna powinna zostać wymienioną na nową, czy przejść generalny remont. Określanie z góry ustalonych terminów konserwacji urządzeń, stosując nawet bardzo zaostrzone kryteria nie eliminuje występowania poważnych awarii. Ponadto konserwacja i utrzymanie ruchu stanowią znaczną cześć budżetu każdego przedsiębiorstwa i skrócenie okresów pomiędzy przeglądami lub bardziej restrykcyjne podejście w postaci częstszej wymiany lub regeneracji poszczególnych komponentów generuje dodatkowe koszty, które nie zawsze są do zaakceptowania z punktu widzenia biznesu.

Konserwacja predykcyjna (ang. predictive maintenance) to jeden z obszarów innowacji, który odgrywa coraz większą rolę w gałęziach sektora elektroenergetycznego, jak i całego przemysłu. Założeniem są systemy informatyczne, które określają stan techniczny danego urządzenia w czasie rzeczywistym. Wykorzystują w tym celu nieprzerwane pomiary jego parametrów. Na podstawie odpowiedniej kombinacji tych parametrów, sprawdzane są warunki określające konieczność wykonania konserwacji. Odpowiednio zaprojektowany i działający system konserwacji predykcyjnej sam zasygnalizuje wcześniej o zbliżającym się czasie wykonania niezbędnej naprawy urządzenia, zanim dojdzie do pogorszenia się jego parametrów lub całkowitego zniszczenia.

Klasyczna konserwacja prewencyjna skupiona jest zatem na zaprojektowaniu najbardziej optymalnych odstępów czasowych pomiędzy wykonywanymi przeglądami i zakresami tych przeglądów. Konserwacja predykcyjna opiera się na zaprojektowaniu optymalnych warunków, których spełnienie wygeneruje sygnał do konieczności wykonania konserwacji. Warunki oparte są na konkretnych kombinacjach wielkości parametrów urządzenia, które mierzone są w czasie rzeczywistym. Schemat blokowy działania konserwacji predykcyjnej przedstawiono na rysunku 3.1.



Rys 3.1. Schemat działania konserwacji predykcyjnej

Typowym przykładem zastosowania systemu konserwacji predykcyjnej w przemyśle jest monitorowanie dużych maszyn wirujących. Maszyna wyposażona jest w aparaturę pomiarową mierzącą jednocześnie wartości temperatur, drgań, poziomu hałasu, zawartości odpowiednich harmonicznych w prądzie, napięciu i innych parametrów. Analizowanie tych wartości w czasie rzeczywistym przez personel byłoby bardzo trudne do osiągnięcia. Co więcej, nawet zastosowanie systemu informatycznego porównującego te parametry często nie wystarcza. Zaimplementowanie logiki polegającej na reakcji na poszczególne poziomy graniczne mierzonych wielkości może nie dostarczyć zadowalających rezultatów lub co więcej powodować nadmierne fałszywe zadziałania (false positives). Nieliniowość niektórych parametrów i ich zmienna korelacja względem siebie w dynamicznych warunkach pracy maszyny jest bardzo trudna do zamodelowania stosując zwykłą logikę programistyczną lub klasyczne modelowanie statystyczne. Zaprojektowanie warunków, przy których system miałby wygenerować sygnał o zbliżającej się awarii, staje się w takim przypadku dużym wyzwaniem. Coraz częściej wykorzystuje się w tym celu sztuczne sieci neuronowe. Okazuje się, że potrafią one być bardzo dobrymi analizatorami, szczególnie jeżeli należy poradzić sobie z problemem wielu zmiennych oraz zamodelować nietrywialne powiązania między nimi.

3.2. Sztuczne sieci neuronowe

Postęp jaki uczyniono w zakresie pozyskiwania danych, ich gromadzenia oraz sprawnego przetwarzania spowodował silny rozwój dziedzin uczenia maszynowego, w tym sztucznych sieci neuronowych.

Sieci jako wyrafinowana technika modelowania, zdolna do odwzorowywania nawet nadzwyczaj złożonych i nieliniowych funkcji potrzebuje do swej pracy dużo większej ilości obserwacji niż popularne modele analityczne i statystyczne. Stąd teraz, w erze niemal nieograniczonego dostępu do danych i mocy obliczeniowych infrastruktur komputerowych metody oparte o sztuczne sieci neuronowe cieszą się coraz większą popularnością. Sieci neuronowe mogą być stosowane w praktycznie każdej sytuacji, gdzie pomiędzy zmiennymi zależnymi i niezależnymi istnieje rzeczywista zależność, nawet jeśli jest ona bardzo skomplikowana i niewyrażalna w klasyczny sposób.

Sieci umożliwiają także kontrolę nad złożonym problemem wielowymiarowości, który przy stosowaniu pospolitych metod statystycznych znacząco utrudnia lub nawet uniemożliwia próby modelowania funkcji z dużą ilością zmiennych niezależnych. Z tego też powodu nierzadko bywają fundamentem modeli opisujących bardzo złożone systemy jak elektrownie czy hale produkcyjne [1].

3.2.1. Sieci neuronowe, a detekcja anomalii.

Obecnie znacząca większość rozwiązań otrzymanych przy użyciu sztucznych sieci neuronowych wykorzystuje tak zwane uczenie nadzorowane (ang. supervised learning). Polega ono na dostarczeniu wraz z danymi uczącymi, wartości prawdziwych, które sieć ma przewidzieć. Jeśli na przykład sieć miałaby nauczyć się rozróżniać, czy na podanym na zdjęciu znajduje się kot czy pies, podczas etapu uczenia każdy obraz, oprócz jego numerycznej reprezentacji posiadałby także jedną z etykiet: pies lub kot. Sieć podczas uczenia znałaby prawdziwą etykietę, do której powinna się zbliżyć, aktualizując swoje wewnętrzne połączenia. Uczenie nadzorowane odgrywa w uczeniu maszynowym olbrzymią rolę. To tej metodzie zawdzięcza się postęp w przetwarzaniu mowy czy obrazów. Wymaga ona jednak dużej ilości wstępnie etykietowanych danych.

Wykorzystanie takiego podejścia w kontekście detekcji anomalii może być problematyczne. Jeśli sieć miałaby nauczyć się, które kombinacje danych wejściowych (natężenie prądu, temperatura, poziom hałasu itd.) odpowiadają awarii danego urządzenia, to podczas uczenia należałoby przedstawić sieci takie reprezentatywne przypadki. To oznacza, że musielibyśmy posiadać zbiory danych, które reprezentują awarię konkretnych urządzeń. Uzyskanie takich danych, pomimo że możliwe, byłoby kosztowne i niepraktyczne. Nawet jeśli uwzględnimy przypadki, powstałe wskutek długo monitorowanego urządzenia, które po czasie uległo awarii okaże się, że takie dane, choć niewątpliwie przydatne, nie są tak łatwo dostępne. Awarie urządzeń w stosunku do czasu ich normalnej pracy są naturalnie dużo rzadszym zjawiskiem. Dane więc byłyby bardzo niezbalansowane – znacznie więcej próbek określających normalną prace, a tylko minimalna ilość próbek określających stan awarii. Problem niezbalansowanych danych znacznie utrudnia prawidłowe uczenie sieci.

Jak wspomniano, duże etykietowane zbiory danych monitorowanych maszyn są trudne do uzyskania, a proces tworzenia ich od podstaw jest bardzo kosztowny i wymaga dużej wiedzy eksperckiej. Z tego też powodu podczas wykrywania anomalii w pracy urządzeń przy użyciu sztucznych sieci neuronowych nierzadko wykorzystuje się podejście nienadzorowane (ang. unsupervised learning).

Uczenie nienadzorowane zakłada brak obecności dokładnego wyjścia z danych uczących. Przytaczając przykład naszej maszyny wirującej, oprócz danych opisujących parametry maszyny – poziom drgań, natężenie prądu itd. nie posiadamy żadnej informacji w jakim obecnie stanie znajduje się maszyna ani tym bardziej, jaki jest stopień jej zużycia czy degradacji poszczególnych komponentów. Dysponujemy tylko danymi zgromadzonymi podczas jej pracy.

Uczenie nienadzorowane jest bardzo specyficzną i trudną dziedziną uczenia maszynowego. Wymaga starannego doboru algorytmów, a wysnucie wniosków z efektów końcowych wymaga dużo więcej pracy i znajomości dziedziny rozwiązywanego problemu. Uczenie nienadzorowane skupia jednak coraz większą uwagę społeczności naukowej i stosowane jest coraz szerzej w świecie techniki i biznesu. Ta metoda potrafi wydobyć z danych ukryte informację, których istnienia można by się nie spodziewać analizując początkowo problem. W środowisku naukowym istnieje przekonanie, że algorytmy uczenia nienadzorowanego często potrafią odpowiedzieć na niezadane pytanie. Klasycznym przypadkiem jest analiza konsumencka wielkich marketów spożywczych. Koncerny spożywcze od lat stale monitorują preferencje grup konsumenckich podczas robienia zakupów wykorzystując do tego techniki uczenia nienadzorowanego. Wyjściem sieci analizującej całe grupy konsumentów i kupowanych przez nich produktów są ukryte, niewidzialne na pierwszy rzut oka zależności,

które odpowiednio przeanalizowane doprowadzają do rozmieszczenia produktów na hali sklepowej w precyzyjnie przemyślany sposób.

Uczenie nienadzorowane w kontekście detekcji anomalii może być rozumiane jako sposób na znalezienie ukrytych zależności opisujących dane, a następnie monitorowaniu ich ewentualnych zmian – wystąpienia anomalii.

3.3. Warystory tlenkowo-cynkowe

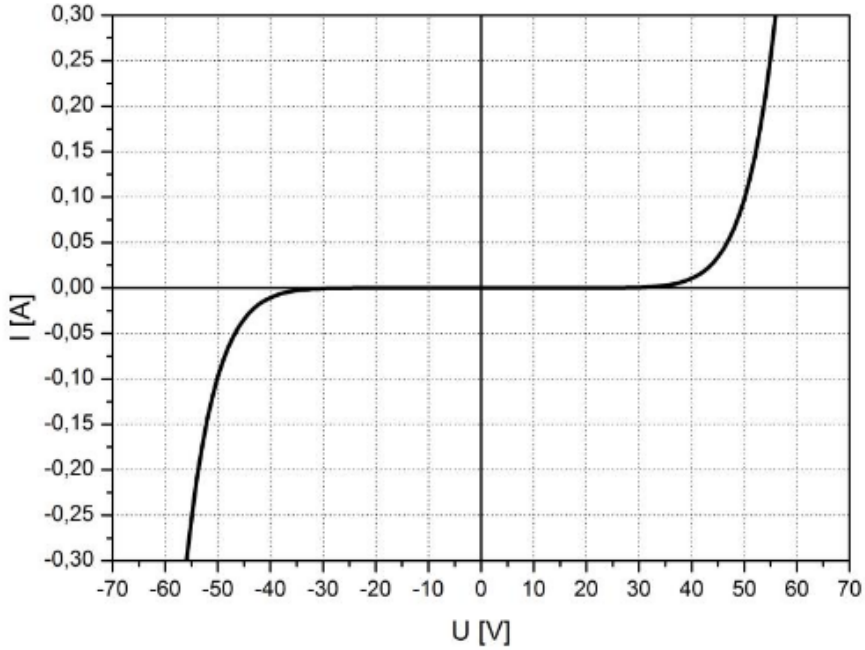
Ograniczniki przepięć zbudowane z warystorów tlenkowo-cynkowych stanowią istotny element systemu elektroenergetycznego. Ich rolą jest ograniczenie skutków przepięć zewnętrznych, jak i wewnętrznych (łączeniowych). Jednakże podczas ich awarii nierzadko dochodzi do zwarć czy wybuchów, które są niebezpieczne zarówno dla samego systemu, jak i personelu. Ponadto wskutek postępującej w czasie degradacji ich struktury wewnętrznej, mogą powodować nieplanowane przerwy w zasilaniu. Z tych powodów stan ograniczników przepięć musi być regularnie badany, aby nie dopuścić do ich niebezpiecznego w skutkach zniszczenia.

3.3.1. Budowa

Warystory tlenkowo-cynkowe są ceramicznymi materiałami polikrystalicznymi charakteryzującymi się silnie nieliniową charakterystyką napięciowo-prądową. Procesy technologiczne wytwarzania ceramiki warystorowej powodują powstanie struktury składającej się z matrycy przewodzących ziaren ZnO otoczonych cienką warstwą międzyziarnową o właściwościach półprzewodzących. Tak ukształtowana mikrostruktura powoduje formowanie się podwójnych barier potencjału na złączach międzyziarnowych. Decydują one o powstaniu nieliniowego przewodnictwa elektrycznego struktury wewnętrznej warystora. [2]

Silna nieliniowość charakterystyki napięciowo-prądowej wykorzystywana jest jako główny czynnik ochrony przeciwprzepięciowej. W zakresie napięcia znamionowego warystor posiada bardzo dużą impedancją wewnętrzną, co przekłada się na bardzo niskie natężenie prądu płynącego przez jego strukturę wewnętrzną. Gdy napięcie na jego zaciskach osiągnie wartość graniczną, specyficzną dla danego warystora – prąd rośnie w bardzo szybkim tempie osiągając wartości znacznie wyższe od znamionowych. Powoduje to przejęcie energii przepięcia przez warystor w postaci przepływu przez jego strukturę wewnętrzną prądu udarowego. Ponadto wartość prądu może być na tyle duża, że spowoduje odpowiednią reakcję zabezpieczeń nadprądowych i wyłączenie danego urządzenia spod zbyt dużego napięcia.

Typową charakterystyką warystora tlenkowo-cynkowego pokazano na rysunku 3.2.



Rys 3.2. Charakterystyka prądowo-napięciowa warystora tlenkowo-cynkowego

Charakterystyka napięciowo-prądowa warystora ZnO opisana jest następującym równaniem:

$$J = cU^{\alpha} \tag{1}$$

Gdzie:
J – gęstość prądu,
U – napięcie,
c – stała
α – współczynnik nieliniowości

Współczynnik nieliniowości **α** stanowi główny parametr określający dany warystor. Jego wartość zależy między innymi od gęstości prądu płynącego przez jego strukturę jak i parametrów procesów technologicznych podczas jego wytwarzania – temperatury spiekania czy zawartości dodatków [2].

Wartość współczynnika **α** zmienia się razem z warunkami pracy warystora i osiąga największą wartość po przekroczeniu napięcia granicznego. Przepływ prądu odbywa się poprzez ruch elektronów, wzbudzonych do odpowiedniego poziomu za pośrednictwem jonizacji termicznej. Stąd silna zależność współczynnika **α**, więc charakterystyki prądowo-napięciowej od temperatury [2].

3.3.2. Diagnostyka

Degradacja struktury wewnętrznej powoduje zmianę współczynnika nieliniowości **α**, a tym samym łączy się ze zmianą jego charakterystyki napięciowo-prądowej. Głównymi czynnikami są:

- wysoka temperatura,
- podwyższone napięcie pracy,
- wysoka wilgotność,
- piki przepięciowe

Często powyższe czynniki występują jednocześnie potęgując efekt degradacji.

Istnieje wiele technik diagnostyki ograniczników przepięć. Poniżej wymieniono niektóre z nich.

1. Pomiar prądu.

Podstawowa metoda diagnostyki zakłada pomiar natężenia prądu płynącego przez warystor. Pierwszym symptomem sugerującym początki degradacji warystora jest wzrost prądu płynącego przez jego strukturę przy określonym napięciu roboczym. Z punktu widzenia obwodu elektrycznego degradacja powoduje zmiany w wartościach rezystancji i pojemności schematu zastępczego służącego do modelowania pracy warystora. Sumaryczny prąd płynący przez warystor w znacznie większej mierze składa się jednak ze składowej pojemnościowej, której wielkość praktycznie nie ulega zmianie podczas degradacji. Metodę tę usprawnia się mierząc tylko składową rezystancyjną prądu [2]. Ponadto pomiar prądu lub jego poszczególnych składowych powinien być także skorygowany o odpowiednie współczynniki poprawkowe, głównie temperaturowe i napięciowe [3].

2. Pomiar temperatury

Wzrost prądu powoduje także wydzielanie się na warystorze większej ilości ciepła, co skutkuje podwyższeniem temperatury warystora. Tą zależność wykorzystuje się w metodzie opartej o pomiary temperatury. Do pomiarów wykorzystuje się czujniki bądź zaawansowane kamery termowizyjne [4].

3. Pomiar zawartości wyższych harmonicznych w prądzie.

Metoda ta uwzględnia zawartość trzeciej i dalszych harmonicznych w prądzie oraz ich wpływ na składową rezystancyjną i temperaturę warystora [5]. Zawartość harmonicznych nie jest jednak zawsze jednakowo skorelowana ze wzrostem prądu, szczególnie w warunkach o podwyższonej wilgotności. Ponadto wartość może być zakłamana występowaniem wyższych harmonicznych w napięciu zasilającym [2].

4. Monitorowanie parametrów wewnętrznych modelu warystora.

Metoda polega na wyznaczeniu parametrów wewnętrznych R i C modelu zastępczego warystora jeszcze nie eksploatowanego, a następnie monitorowaniu tych parametrów w czasie pracy. Sygnał o postępującej degradacji wynika, ze zmiany tych parametrów względem wartości otrzymanych w warunkach nominalnych.

W celu wyznaczenia parametrów wewnętrznych warystora dokonuje się analizy budując ich schematy zastępcze. Dla napięć przemiennych przyjmuje się najczęściej elementy R, C. Wyznaczenie parametrów może odbywać się drogą analityczną oraz przy użyciu sztucznych sieci neuronowych [6].

4. ZAPROPONOWANE ROZWIĄZANIE

W pracy zaproponowano model sieci neuronowej oparty o architekturę autoenkodera z warstwami rekurencyjnymi LSTM (Long Short Term Memory).

4.1. Rekurencyjne sieci neuronowe

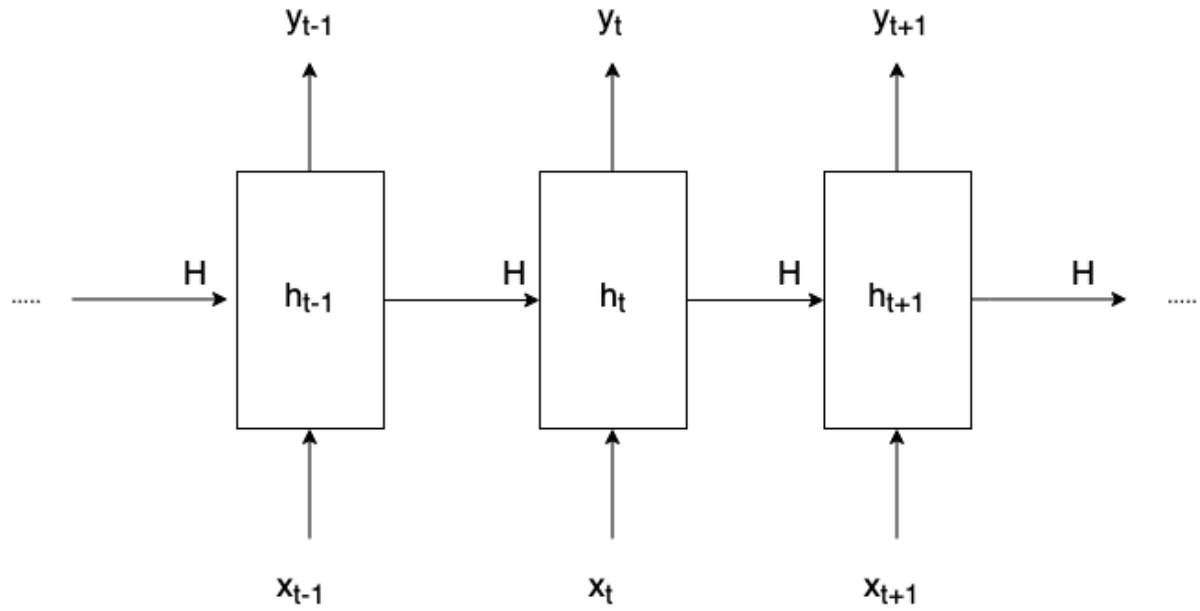
Klasyczne modele uczenia maszynowego i podstawowe struktury sieci neuronowych zakładają niezależność danych wejściowych. To znaczy, że konkretna próbka danych podawana na wejście modelu jest całkowicie niezależna od innych próbek występujących w zbiorze uczącym. Założenie to jest błędne w momencie, gdy dane reprezentowane są za pomocą szeregów czasowych. W przypadku gdy kolejność wystąpienia obserwacji w zbiorze ma znaczenie, potrzebne jest przetwarzanie danych w sekwencjach, czyli ciągach próbek o ściśle określonej kolejności.

Dobrze ilustrującym to przykładem są modele przetwarzania języka (ang. Natural Language Processing). Aby zrozumieć dobrze ludzki język, musimy znać kontekst w jakim wypowiadane są słowa. Czytając pojedyncze słowo lub wycinek tekstu z losowo umieszczonymi słowami trudno jest wysnuć odpowiednie wnioski. Model do przetwarzania ludzkiego języka musi być zatem uczony na całych sekwencjach słów, które występują w kolejności.

W przypadku analizowania wielkości elektrycznych sytuacja jest podobna. Ciężko jest zbadać sygnał napięcia czy natężenia prądu analizując tylko pojedyncze próbki. Dopiero zamodelowanie całej, odpowiednio długiej sekwencji może pozwolić na trafniejszą analizę.

Rekurencyjne sieci neuronowe umożliwiają przetwarzanie danych ciągami, budując w swoich połączeniach coś na kształt wewnętrznej pamięci. Przyjmują jako wejście ciąg d-wymiarowych wektorów i przetwarzają je rekurencyjnie, wykonując te same operacje dla każdego elementu sekwencji. Wyjście każdej próbki zależy nie tylko od zmiennych ją opisujących, ale także od stanu z poprzednich kroków. W ten sposób można zamodelować zależności, które uwidaczniają się dopiero po przeanalizowaniu całego ciągu danych, a nie pojedynczych próbek. Taka struktura umożliwia, aby informacja zawarta w pierwszej próbce miała wpływ nawet na ostatnią w sekwencji.

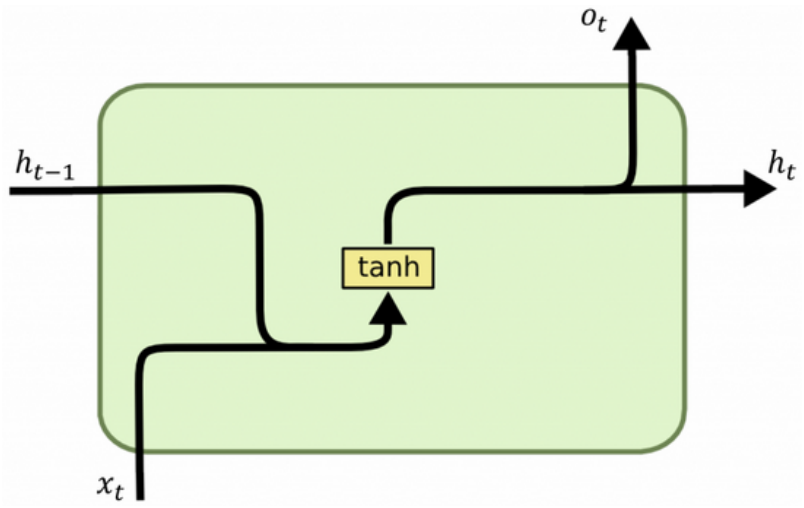
4.1.1. Budowa



Rys. 4.1. Schemat rekurencyjnej sieci neuronowej

Na powyższym rysunku przedstawiono uproszczony schemat działania rekurencyjnej sieci neuronowej. Sieć została symbolicznie rozwinięta do trzech sieci dla lepszego zilustrowania zachodzących operacji. W każdej chwili czasowej t na wejście sieci podawany jest wektor danych x_t i stan ukryty H. Stan ukryty jest w istocie funkcją wartości z wszystkich dotychczas przetworzonych chwil w sekwencji. Macierz H aktualizowana jest po każdej chwili czasowej i jest współdzielona przez całą sieć. Każdy stan zależy więc od wszystkich poprzednich obliczeń.

Konsekwencją takiego przetwarzania danych jest wspomniana wcześniej struktura pamięci. Stany ukryte zawierają informacje bazujące na wcześniejszych krokach. Podczas treningu sieć dąży do takiej kombinacji parametrów ukrytych, aby wyjście było jak najbardziej zbliżone do przewidywanej wartości, uwzględniając przy tym sekwencyjny charakter danych. W praktyce sprowadza się to do takiego dostrojenia wag, aby były one aktualizowane nie tylko w oparciu o wektor wejściowy, ale także o poprzednie próbki w sekwencji. Może to przykładowo oznaczać, że próbka z chwili czasowej t-10 ma dużo większy wpływ na wynik niż pozostałe próbki, ponieważ zawiera kluczową dla predykcji informację. Sieć sama znajdzie w tym wypadku istotne elementy sekwencji i nada im odpowiednie wagi.



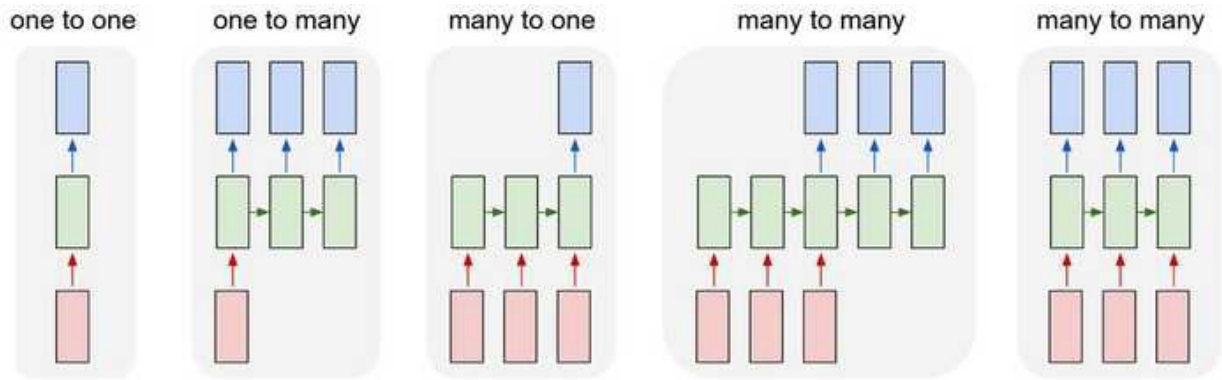
Rys. 4.2. Schemat rekurencyjnego neuronu
http://dprogrammer.org/rnn-lstm-gru

W matematycznym ujęciu, wyjście w bieżącej chwili czasowej składa się z dodania wektorów - wyjściowego z poprzedniej chwili t_{-1} i wejściowego z bieżącej chwili t. Całość przepuszczana jest przez nieliniową funkcję aktywacji tanh (tangens hiperboliczny), która sprowadza wartości do przedziału -1; 1.

$$h(t) = \tanh(i_t) = \tanh(U_h x_t + V_h h_{t-1} + b_h) \quad (2)$$

4.1.2. Wyjście sieci rekurencyjnej

W podstawowej konfiguracji sieć przetwarza całą sekwencję danych i generuje na swoim wyjściu sekwencje o tej samej długości. Wyjście jest generowane dla każdej próbki w czasie, więc gdy przykładowo na wejście zostanie podana sekwencja składająca się z 3 próbek, wyjście także będzie ciągiem 3 próbek. Nie jest to jedyna możliwa konfiguracja. Sieci rekurencyjne mogą przetwarzać dane na wiele sposobów. Gdy modelują na przykład klasyfikator wyjściem może być jedna próbka. Ponadto podawane na wejście sekwencje mogą być różnej długości. Schematy niektórych konfiguracji pokazano na rysunku 4.3.



Rys. 4.3. Konfiguracje pracy sieci rekurencyjnych
http://dprogrammer.org/rnn-lstm-gru

4.1.3. Problem zanikających gradientów.

Trening sieci neuronowych składa się z następujących etapów:

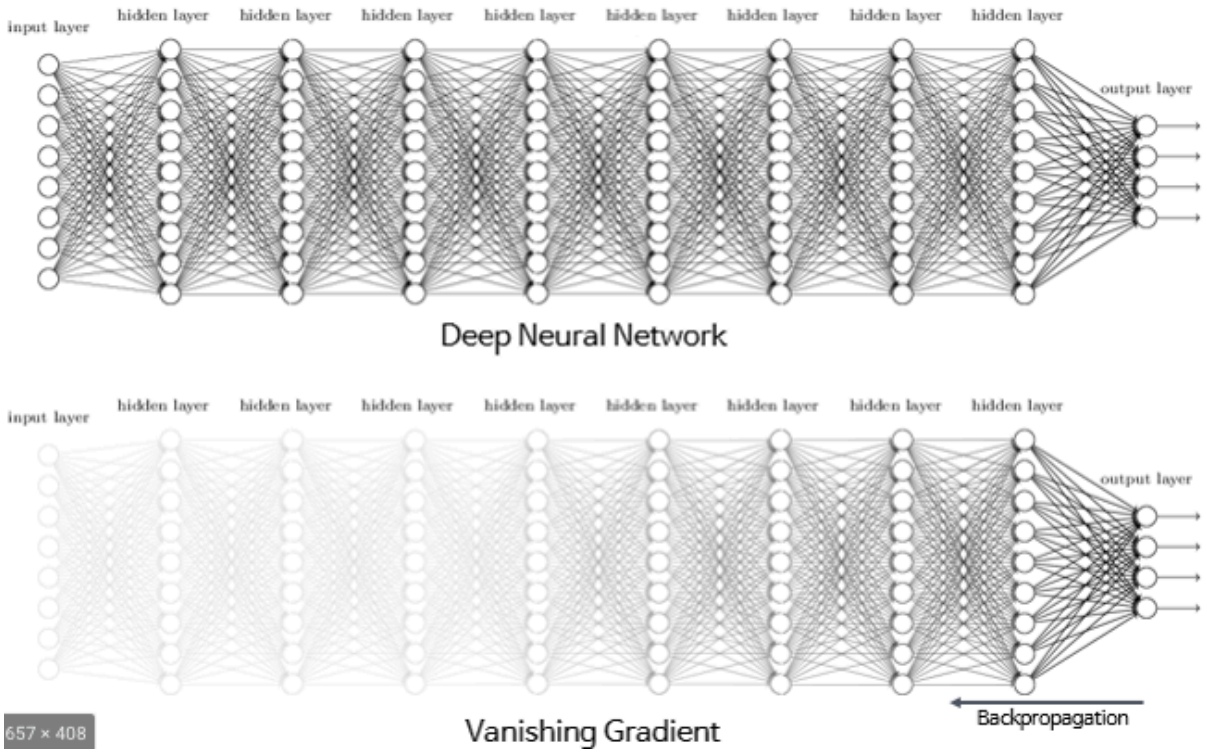
- przejście wektora danych wejściowych przez sieć (forward pass),
- porównanie wyjścia z sieci (predykcji) z wartością prawdziwą – obliczenie błędu sieci,
- obliczenie gradientów parametrów sieci względem otrzymanego błędu przy użyciu algorytmu wstecznej propagacji,
- korekcja parametrów sieci w oparciu o obliczone gradienty – zmniejszenie błędu sieci.

Sieci rekurencyjne mogą być znacznie trudniejsze do trenowania niż klasyczne sieci neuronowe. Sekwencyjny charakter przetwarzania danych niesie ze sobą dodatkowy koszt – konieczność uwzględnienia każdej chwili czasowej również podczas treningu. To powoduje, że ilość operacji obliczeniowych niezbędnych do wykonania w celu propagacji wstecznej błędu rośnie wraz z długością sekwencji. Z tego powodu sieci rekurencyjne, podobnie jak bardzo głębokie i złożone sieci neuronowe cierpią na przypadłość zanikających gradientów.

Sieć neuronowa jest w istocie bardzo złożoną funkcją matematyczną składającą się z wielu podfunkcji. Obliczenie gradientów jej poszczególnych parametrów w celu ich późniejszej korekcji realizowane jest przy użyciu znanej w matematyce reguły łańcuchowej. Łatwo wyobrazić sobie, że parametr występujący w sieci wcześniej – bliżej warstwy wejściowej, a dalej wyjściowej, będzie podczas kalkulacji gradientu obarczony większą ilością operacji matematycznych do wykonania. Jeśli dodatkowo poszczególne składowe tych operacji będą miały wartość mniejszą od 1, finalnie wynik dążył będzie do 0.

Z tego też powodu przy głębokich sieciach, czyli takich, w których dane wejściowe są przetwarzane przez wiele warstw zanim dojdą do neuronów wyjściowych, istnieje ryzyko, że parametry pierwszych warstw zostaną zignorowane, ponieważ ich gradient, obliczony względem błędu sieci będzie miał wartość bliską 0.

Poniższy rysunek poglądowo przedstawia problem zanikających gradientów w głębokich sieciach neuronowych.



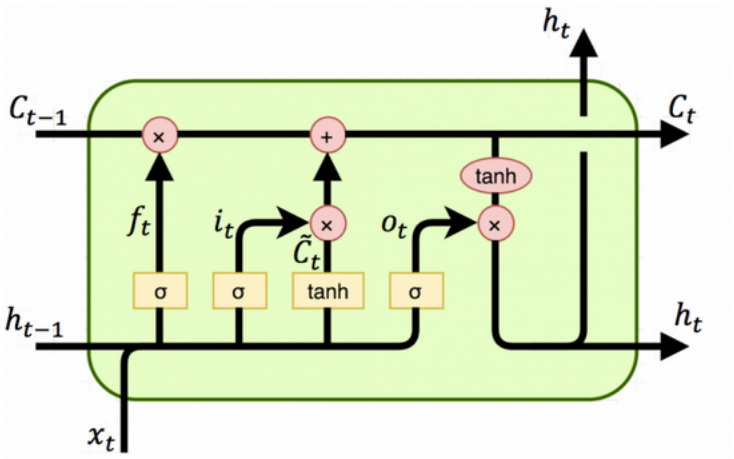
Rys. 4.4. Zanikające gradienty w głębokiej sieci neuronowej
https://trendy00develope.tistory.com/37

Jak widać, pierwsze warstwy sieci podczas operacji wstecznej propagacji błędu zostają wygaszone, czyli ich gradienty są bliskie zeru. Podczas treningu nie zostają one więc korygowane, wskutek czego pozostają dla sieci niewykorzystaną przestrzenią do modelowania informacji. W ten sposób sieć, podczas swojej pracy weźmie pod uwagę tylko ostanie, najbliższe wyjściu warstwy.

W sieciach rekurencyjnych iteracyjne przejście przez całą sekwencję powoduje ten sam efekt – zanikanie gradientów w początkowych próbkach. Skutkuje to znacznym ograniczeniem wykorzystania takich sieci przy pracy z długimi sekwencjami. Gradienty we wcześniejszych stanach stają się bardzo małe i znika możliwość zachowania historii tych stanów. Sieć traci w tym wypadku swoje zdolności do zapamiętania i modelowania sekwencji z uwzględnieniem wszystkich, także początkowych próbek. Propagacja wsteczna w czasie jest zbyt wrażliwa na niedawne zakłócenia. Choć w teorii proste sieci rekurencyjne są w stanie uczyć się zależności długoterminowych, w praktyce ograniczają się tylko do zależności krótkoterminowych. Z tego powodu powstały ich bardziej rozbudowane odpowiedniki.

4.1.4. Sieci LSTM

Sieci LSTM (ang. Long Short Term Memory) to sieci rekurencyjne, które powstały w celu wyeliminowania problemu zanikającego gradientu oraz poprawy pracy z długimi sekwencjami danych. Względem klasycznej sieci neuronowej, ich struktura została uzupełniona o tzw. bramki oraz dodatkowy stan ukryty przechowujący pamięć długoterminową.



Rys. 4.5. Komorka sieci LSTM
http://dprogrammer.org/rnn-lstm-gru

Stan ukryty jest strukturą przechowującą długoterminowe zależności. Informacje w nim zawarte mogą być dynamicznie zapisywane i usuwane. Jeśli nie ma ingerencji z zewnątrz, stan komórki pozostaje bez zmian. Jest to element zapewniający sieci pamięć. Bramki są zaś swojego rodzaju zaworami. Regulują przepływ informacji w sieci, umożliwiając stanowi ukrytemu zachowanie najcenniejszych informacji w sekwencji.

Typowa sieć LSTM zawiera 3 bramki:
- zapominania (ang. forget gate), oznaczona na rysunku literą f,
- wejścia (ang. input gate), oznaczona literą i,
- wyjścia (ang. output gate) oznaczona literą o

Bramka zapominania swoją nazwę zawdzięcza temu, że decyduje o ilości informacji, które mają zostać wykasowane ze stanu ukrytego. Opiera swoją decyzję na poprzednim wyjściu h_{t-1} i bieżącym wejściu x_t. Łączy te informację i kompresuje za pomocą funkcji logistycznej, a następnie przemnaża otrzymane wartości z wektorem stanu ukrytego. Dzięki temu, że funkcja logistyczna zwraca wartości z przedziału 0; 1 informację mogą być całkowicie zachowane – mnożenie przez 1 lub zapomniane - mnożenie przez 0. Oznacza to, że pamięć LSTM może się samoistnie pozbyć niepotrzebnych informacji z wektora stanu ukrytego.

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \tag{3}$$

Bramka wejściowa decyduje o tym, jakie nowe informacje zostaną dodane do komórki pamięci. Odbywa się to w dwóch etapach. W pierwszym podejmowana jest decyzja o tym, czy informacje w ogóle zostaną dodane. Tak, jak w przypadku bramki zapominania, decyzja bazuje na wartościach h_{t-1} i $x_{t.}$. Funkcja zwraca wynik 0 lub 1 za pośrednictwem funkcji logistycznej dostępnej dla każdego bloku wektora. W rezultacie sieć LSTM, może dodawać do swojej pamięci, tylko specyficzne informację.

$$i_t = \sigma(W_i h_{t-1} + U_f x_t + b_i) \tag{4}$$

Wektor informacji, który został zakwalifikowany do dodania jest obliczany na podstawie poprzedniego wyjścia h_{t-1} i skompresowany za pomocą funkcji tanh

$$a_t = tanh(W_c h_{t-1} + U_c x_t + b_c) \tag{5}$$

Ostatecznie bramka zapominania i bramka wejściowa decydują o zawartości stanu ukrytego – pamięci sieci.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot a_t \tag{6}$$

Ostatnia bramka decyduje z czego składać ma się wyjście w bieżącej chwili czasowej. Pobiera ona wartości h_{t-1} i $x_{t.}$ jako swoje wejście i zwraca wektor liczb z przedziału 0; 1 za pomocą funkcji logistycznej. Wyjście 0 oznacza, że blok komórki nie zwraca w bieżącej chwili żadnej informacji, a 1 że na wyjście przekazywana jest pełna informacja z bieżącej próbki.

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o) \tag{7}$$

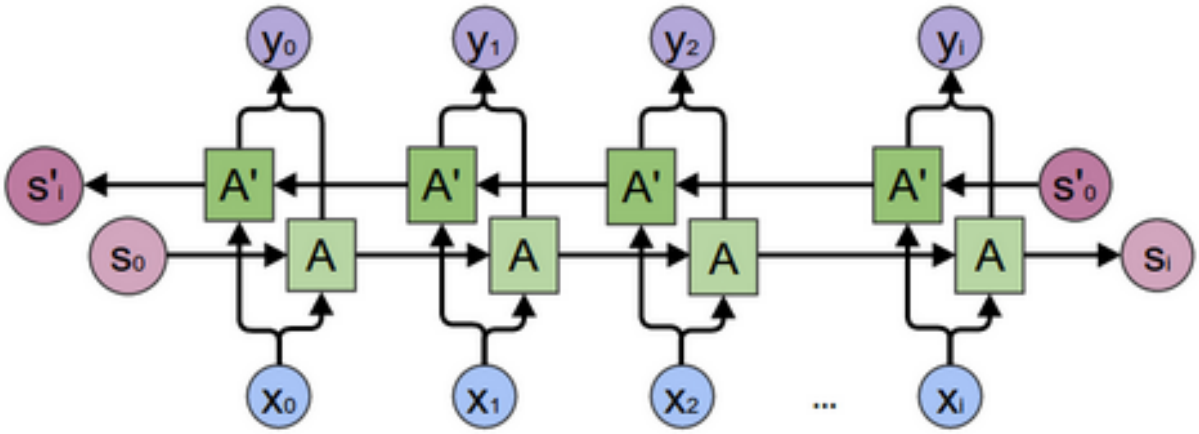
Ostateczna wartość wyjścia to kombinacja otrzymanego powyżej wyjścia z bieżącej komórki oraz dotychczas zarejestrowanego stanu ukrytego – pamięci długotrwałej sieci.

$$h_t = o_t \cdot \tanh(C_t) \tag{8}$$

Ponieważ wszystkie te wielkości są dostępne na zewnątrz komórki, tak jak w przypadku klasycznych neuronów rekurencyjnych, istnieje możliwość ich połączenia, aby przetwarzały całe sekwencje danych i trenowania całej sieci za pomocą wstecznej propagacji w czasie. Dodatkowy stan ukryty jest niezależnym wektorem, magazynującym dotychczasowe informacje. Tylko bramka zapominania jest w stanie wymazać z niech pamięć, co czyni sieć LSTM zdolną do przechowywania długoterminowych zależności, a informację w niej zawarte mogą pozostać niezmienione przez długi czas.

4.1.5. Sieci dwukierunkowe LSTM

Sieci rekurencyjne umożliwiają także pracę w trybie dwukierunkowym. Polega to na zdublowaniu warstwy komórek rekurencyjnych, tak aby jedna przetwarzała sekwencje danych licząc od pierwszej próbki do ostatniej, a druga od ostatniej do pierwszej. Istnieją dwa stany ukryte w danej chwili czasowej otrzymane w wyniku tych dwóch iteracji. Ostateczne wyjście składa się z odpowiedniej kombinacji tych dwóch stanów ukrytych – najczęściej połączenia (konkatenacji), lecz niekiedy ich sumy lub średniej.



Rys. 4.6. Sieć rekurencyjna dwukierunkowa
https://colah.github.io/posts/2015-09-NN-Types-FP/

Sieci dwukierunkowe zostały stworzone z myślą o zwiększeniu ilości informacji dostępnej dla sieci podczas przetwarzania sekwencji. Sieć w danej chwili czasowej ma do dyspozycji informację przetworzone od początku sekwencji, a także od jej końca. Jest to szczególnie istotne, w przypadkach, kiedy modelowane zjawisko nie jest zależne tylko od przeszłych próbek, ale także od przyszłych. Uzyskana w ten sposób informacja stanowi swojego rodzaju kontekst, dostępny w każdej chwili czasowej. To rozwiązanie dobrze sprawdza się przy tłumaczeniu języka – często ostatnie słowa mają kluczowy wpływ na znaczenie słów poprzedzających i w rezultacie całe zdanie. Przetwarzanie sekwencji słów jednocześnie od początku i od końca może pozwolić na trafniejszą predykcję w danej chwili czasowej.

Sieci dwukierunkowe są kosztowniejsze w użyciu – wymagają większych zasobów obliczeniowych do trenowania oraz predykcji.

4.2. Autoenkodery

Autoenkodery są symetrycznymi sieciami neuronowymi trenowanymi w sposób nienadzorowany. Ich zadaniem jest odtworzenie danych wejściowych. Składają się z dwóch części: enkodera i dekodera.

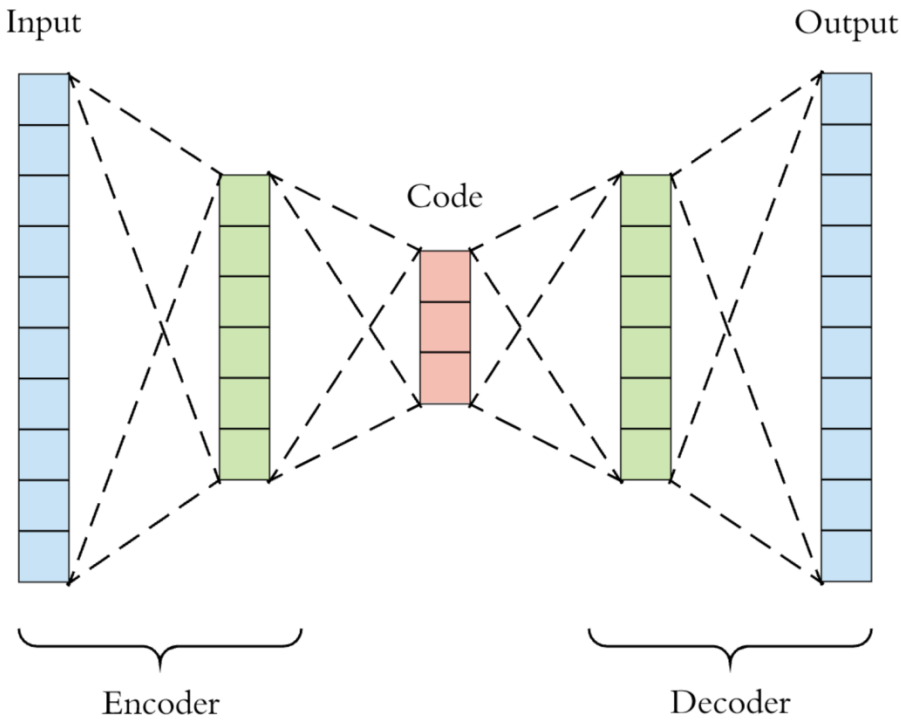
Enkoder jest funkcją, która rzutuje dane wejściowe na ich utajoną reprezentację. Przyjmuje postać:

$$z = f(x) = s(W_x + b_x) \tag{9}$$

Dekoder zaś jest funkcją, która odtwarza z tej utajonej reprezentacji dane wejściowe.

$$x' = g(z) = g\big(f(x)\big) = s(W_z + b_z) \tag{10}$$

, gdzie:
s – nieliniowa funkcja aktywacji



Rys. 4.7. Schemat ideowy autoenkodera
https://blog.paperspace.com/autoencoder-image-compression-keras/

Warstwa wyjściowa ma ten sam rozmiar co wejściowa, ponieważ jej celem jest zrekonstruowane własnych wejść, a nie przewidywanie wartości docelowych. Wyjście autoenkodera jest zatem przybliżeniem jego wejścia. Symbolicznie można opisać w następujący sposób:

$$f(X) \approx X \tag{11}$$

Trenowanie autoenkodera sprowadza się do zminimalizowania błędu odwzorowania wektora wejściowego. Funkcję strat opisuje błąd rekonstrukcji – najczęściej błąd średniokwadratowy (ang. Mean Squared Error – MSE), który definiuje jak dobrze sieć zrekonstruowała podany na wejście wektor danych. Ta reprezentacja zostaje zapisana jako parametry warstwy utajonej „Code", która ma najczęściej mniejszy wymiar od warstwy wejściowej. Taka konfiguracja zmusza sieć do znalezienia najbardziej skutecznej i kompaktowej reprezentacji danych wejściowych z minimalną utratą informacji. Głębsze autoenkodery, w których wykorzystane są nieliniowe funkcje aktywacji bardzo dobrze radzą sobie ze znalezieniem ukrytych lub skompresowanych cech danych wejściowych.

Struktury autoenkoderów wykorzystuje się w wielu dziedzinach uczenia maszynowego. Znajdują one zastosowanie m.in. podczas detekcji anomalii, kompresji wymiarów danych wejściowych czy odszumianiu obrazów. Z powodu zdolności do wyuczenia ukrytych rozkładów danych wejściowych są one także komponentem modeli generatywnych transferujących style malarskie lub generujące utwory muzyczne. Wykorzystuje się je także w kontrowersyjnym projekcie o nazwie DeepFake, który polega na zamianie twarzy osób występujących w filmach bądź na zdjęciach.

4.3. Opis zaprojektowanego algorytmu

Zaproponowane rozwiązanie do wykrycia anomalii w szeregach czasowych reprezentujących próbki napięcia i prądu warystora wykorzystuje architekturę autoenkodera z dwukierunkowymi warstwami rekurencyjnymi LSTM. Schemat działania takiego rozwiązania jest następujący:

Krok 1.
Identyfikacja „prawidłowego" zbioru danych. Chodzi o próbki, co do których istnieje pewność, że reprezentują encje niebędące anomaliami. Identyfikacja bazuje na danych historycznych oraz na założeniu, że oficjalnie nie rozpoznano żadnych anomalii.

Takie podejście jest często określane mianem uczenia seminadzorowanego. Nie jest ono czysto nienadzorowane. Bazuje ono bowiem na założeniu, że większość obserwacji jest pozbawiona anomalii.

Krok 2.
Trening modelu na sekwencjach „prawidłowych" danych. Zadaniem modelu jest wyuczenie się rozkładu danych odpowiadającemu prawidłowej pracy warystora – przed pojawieniem się oznak degradacji struktury wewnętrznej. Autoenkoder powinien z możliwie jak najmniejszym błędem odtwarzać podane na wejście treningowe sekwencje danych.

Krok 3.
Zaprzestanie trenowania modelu i przełączenie w tryb czuwania. Tryb czuwania polega na podawaniu sieci rzeczywistych danych i sprawdzaniu otrzymanego błędu rekonstrukcji autoenkodera. Jeśli błąd mieści się w przedziale, który wcześniej odnotowano na danych uczących – sekwencja danych uznawana jest za prawidłową. Jeśli zaś błąd jest większy od przyjętego, czyli sekwencja zawiera inny rozkład zmiennych niż sieć widziała dotychczas – traktujemy to jako anomalię. Dzięki wytrenowaniu modelu na danych nie zawierających anomalii istnieje duże prawdopodobieństwo poprawnej rekonstrukcji danych reprezentujących dobry stan warystora, ale w przypadku sekwencji odstających błąd rekonstrukcji wzrasta ponad nieodnotowaną wcześniej wartość i istnieje możliwość zasygnalizowania pojawienia się degradacji urządzenia.

5. PRAKTYCZNA IMPLEMENTACJA

Niniejszy rozdział poświęcony jest praktycznemu aspektowi stworzenia modelu do detekcji anomalii w warystorze tlenkowo-cynkowym. Całość została napisana w wysokopoziomowym języku programistycznym Python w wersji 3.8 i jest dostępna pod następującym linkiem:

https://github.com/r-zareba/praca_inzynierska

5.1. Wczytanie i przygotowanie danych uczących

Jak każdy model uczenia maszynowego, sieć neuronowa do treningu potrzebuje danych. Jako zbiór uczący wykorzystano zarejestrowane próbki napięcia i natężenia prądu warystora podczas jego pracy w systemie elektroenergetycznym.

Ze względu na złożoność obliczeniową zaproponowanej architektury sieci oraz ograniczone zasoby komputerowe dostępne podczas tworzenia niniejszej pracy, zdecydowano, że dane uczące składać się będą jedynie z 10 tysięcy próbek. Zatem wektor danych uczących będzie miał wymiar (10000, 2). Składają się na niego dwa pełne okresy przebiegów napięcia i prądu. Praca ma charakter eksperymentalny, więc dla uproszczenia założono, że podane przebiegi występowały przez dłuższy cały czas pracy warystora.

Przed rozpoczęciem pracy ze środowiskiem programistycznym, wczytano niezbędne moduły:

```
# Import used modules
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.preprocessing
```

Rys. 5.1. Wczytanie użytych w projekcie modułów (bibliotek)

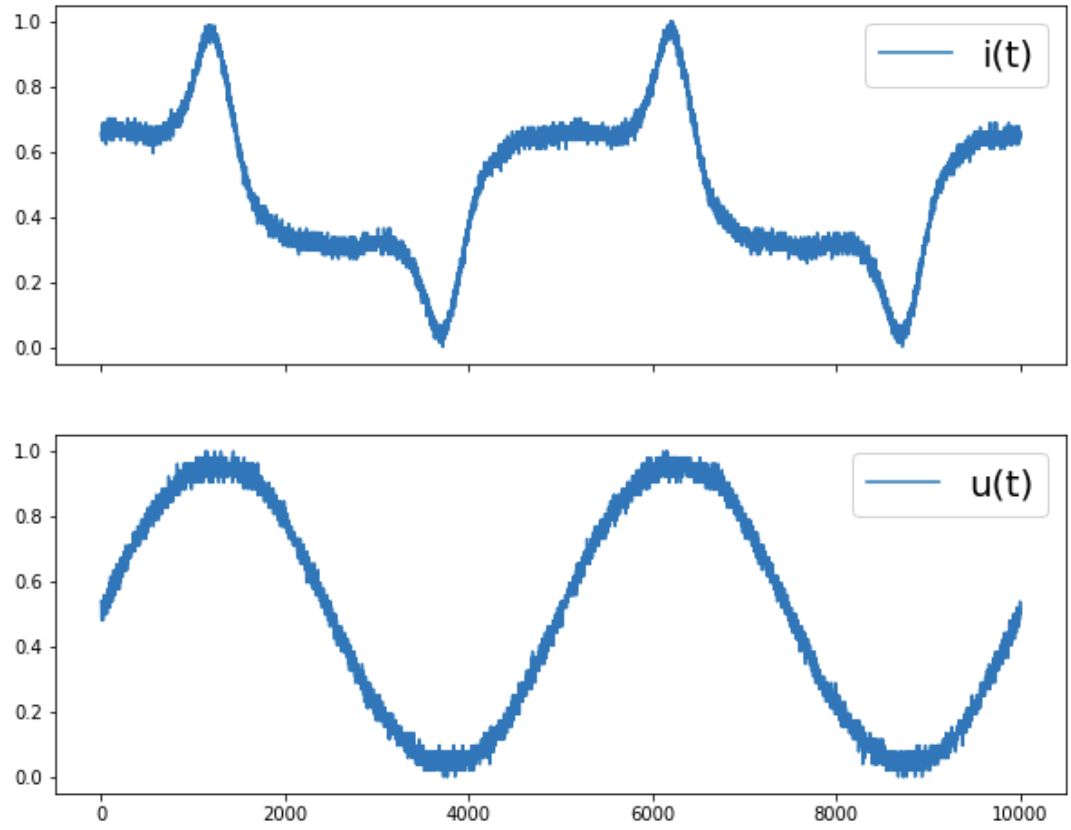
Dane wczytano z pliku o rozszerzeniu csv (ang. comma seperated values), a następnie oczyszczono i dokonano skalowania do przedziału (0, 1) wykorzystując do tego odpowiednie funkcje.

```
# Define data preprocessing functions
CSV_COLUMN_NAMES = ('time', 'u', 'i')

def _normalize_df(df: pd.DataFrame, normalize: bool) -> None:
    """
    Replace ',' with '.', convert columns to float type
    and normalize to range (0, 1)
    """
    scaler = sklearn.preprocessing.MinMaxScaler((0, 1)) if normalize else None

    for col in df.columns:
        df.loc[:, col] = df.loc[:, col].apply(lambda x: x.replace(',', '.'))
        df.loc[:, col] = df.loc[:, col].astype(np.float64)
        if normalize:
            df.loc[:, col] = scaler.fit_transform(df.loc[:, col].values.reshape(-1, 1))

def load_csv(data_path: str, normalize=True) -> pd.DataFrame:
    df = pd.read_csv(data_path, delimiter=';', names=CSV_COLUMN_NAMES, header=0)
    _normalize_df(df, normalize)
    return df

# Read training data
path = 'data/arrester_data.csv'
df = load_csv(path, normalize=True)
```

Rys. 5.2. Wczytanie i przygotowanie danych treningowych

Dane treningowe narysowano na wykresie, korzystając z biblioteki matplotlib.

```
# Plotting training data
fig, (ax1, ax2) = plt.subplots(2, sharex=True, figsize=(10, 8))

ax1.plot(df.loc[:, 'i'], label='i(t)')
ax1.legend(loc='upper right', prop={'size': 20})

ax2.plot(df.loc[:, 'u'], label='u(t)')
ax2.legend(loc='upper right', prop={'size': 20})

plt.show()
```



Rys. 5.3. Wykresy danych uczących – napięcia i natężenia prądu

Następnym krokiem było stworzenie odpowiednich tensorów treningowych. Sieć rekurencyjna działa na sekwencjach danych. Dotychczasowe dane zawierały dwa wymiary (10000, 2). Aby sieć mogła dostawać na wejście całe sekwencje, potrzebne było stworzenie dodatkowego wymiaru.

```
# Creating training array for recurrent neural network
n_timestamps = 32    # Number of points in single sequence

x = df.loc[:, ['i', 'u']].values  # Converting to numpy array
training_array = []

for i in range(len(x) - n_timestamps):
    training_array.append(x[i:i + n_timestamps])

x_train = np.array(training_array)
y_train = x_train

print(f'Training x shape: {x_train.shape}')
print(f'Training y shape: {y_train.shape}')
```

```
Training x shape: (9968, 32, 2)
Training y shape: (9968, 32, 2)
```

Rys. 5.4. Stworzenie tensorów treningowych zawierających sekwencje próbek

Jak widać, stworzone tensory mają wymiar (9968, 32, 2), co można interpretować jako 9968 sekwencji, każda będąca ciągiem 32 występujących po sobie próbek, a dodatkowo każda próbka opisana jest przez 2 zmienne (natężenie prądu i napięcie). Warto zwrócić uwagę na miejsce definicji tensora „y_train". Zawiera on elementy, do których sieć powinna się zbliżyć w swoich predykcjach. Jest on wykorzystywany podczas treningu, do obliczenia funkcji błędu. Z racji tego, że celem autoenkodera jest rekonstrukcja jego własnego wejścia, tensor wyjściowy jest więc tym samym co wejściowy. Programistycznie rozwiązano to za pomocą operacji prostego przypisania.

Końcowym etapem przygotowania danych uczących jest stworzenie finalnych zbiorów za pomocą obiektów klasy Dataset biblioteki tensorflow.

```
# Converting to tensorflow Datasets
batch_size = 64            # Number of sequences in single data batch
shuffle_buffer_size = 1000 # Number of data points in single shuffle batch

train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_dataset = train_dataset.cache().batch(batch_size).shuffle(shuffle_buffer_size).repeat()
```

Rys. 5.5. Stworzenie obiektów zbiorów danych uczących

Ten krok ma na celu zoptymalizowanie operacji wczytywania danych podczas uczenia sieci i podzielenie zbioru na tzw. wsady (ang. batch). Jest to o tyle istotne, że podczas uczenia sieci, dopiero po przejściu przez określony wsad wykona się aktualizacja wag (wsteczna propagacja). Mamy zatem wpływ na to jak często - po ilu sekwencjach danych sieć będzie aktualizowała swoje połączenia. W pracy zdecydowano się na wartość 64. To znaczy, że po iteracyjnym przejściu przez 64 sekwencje (każda po 32 próbki) sieć zaktualizuje swoje połączenia.

Parametr „shuffle_buffer_size” określa, ile próbek ma występować w wybranym zbiorze danych do konstrukcji partii uczącej. Oznacza, to że algorytm losowo weźmie 1000 próbek danych np. z końca okresu przebiegu, utworzy z nich partie uczące i dokona treningu sieci. Następnie skorzysta z 1000 próbek z początku przebiegu itd. Kolejność jest w tym wypadku losowa. Takie podejście przyczynia się do lepszej generalizacji otrzymanych predykcji.

5.2. Definicja modelu sieci neuronowej

Model autoendkodera z warstwami dwukierunkowych LSTM został zdefiniowany używając biblioteki tensorflow w wersji 2.1. Enkoder i dekoder zostały zdefiniowane jako osobne warstwy, a docelowy model używa ich za pomocą prostej kompozycji w swojej strukturze.

```
# Define LSTM Autoencoder model
from tensorflow.keras import layers

class Encoder(layers.Layer):
    def __init__(self, n_neurons: int, n_timestamps: int, n_features: int):
        super().__init__()
        self._lstm1 = layers.Bidirectional(
            layers.LSTM(units=n_neurons, activation='relu',
                        input_shape=(n_timestamps, n_features),
                        return_sequences=True))
        self._lstm2 = layers.Bidirectional(
            layers.LSTM(units=int(n_neurons / 2), activation='relu',
                        return_sequences=False))

    def call(self, x):
        x = self._lstm1(x)
        return self._lstm2(x)


class Decoder(layers.Layer):
    def __init__(self, n_neurons: int, n_timestamps: int, n_features: int):
        super().__init__()
        self._lstm1 = layers.Bidirectional(
            layers.LSTM(units=int(n_neurons / 2), activation='relu',
                        input_shape=(n_timestamps, n_features),
                        return_sequences=True))
        self._lstm2 = layers.Bidirectional(
            layers.LSTM(units=n_neurons, activation='relu',
                        return_sequences=True))

    def call(self, x):
        x = self._lstm1(x)
        return self._lstm2(x)
```

Rys. 5.6. Definicja enkodera i dekodera

```
class Model(tf.keras.Model):
    def __init__(self, n_neurons: int, n_timestamps: int, n_features: int):
        super().__init__()
        self._encoder = Encoder(n_neurons, n_timestamps, n_features)
        self._decoder = Decoder(n_neurons, n_timestamps, n_features)

        self._repeat_vector = layers.RepeatVector(n_timestamps)
        self._time_distributed = layers.TimeDistributed(
            layers.Dense(n_features))

    def call(self, x):
        x = self._encoder(x)
        x = self._repeat_vector(x)
        x = self._decoder(x)
        return self._time_distributed(x)

    def calculate_reconstruction_loss(self, x: np.array) -> np.array:
        y_pred = self.predict(x)
        loss = np.mean((x - y_pred) ** 2, axis=1)
        return np.mean(loss, axis=1)

    def calculate_anomaly_threshold(self, loss: np.array) -> float:
        return loss.max() + (0.25 * loss.std())
```

Rys. 5.7. Definicja modelu autoenkodera z dwukierunkowymi warstwami LSTM

Z racji niedużego skomplikowania danych uczących – zawierają one tylko 2 zmienne i ich przebiegi są okresowe, zdecydowano się na stosunkowo prostą strukturę autoenkodera. Enkoder i dekoder zawierają po 2 warstwy dwukierunkowych LSTM. Pierwsza warstwa enkodera zawiera liczbę neuronów równą „n_neurons" – argumentowi konstruktora. Druga jest o połowę mniejsza. Z kolei dekoder jest lustrzanym odbiciem enkodera – jego pierwsza warstwa zawiera połowę wartości „n_neurons", a druga całą wartość.

Wyjście z enkodera (wąskie gardło sieci) zawiera skompresowaną reprezentację przetworzonej sekwencji. Aby przekazać te dane do warstwy LSTM dekodera potrzebne jest ponowne utworzenie sekwencji. Do tego celu skorzystano z obiektu klasy RepeatVector, który powielił skompresowane wyjście tworząc odpowiednie sekwencje.

Wyjście dekodera zostało zaś przywrócone do wymiarów tensora wejściowego za pomocą obiektu klasy TimeDistributed. Predykcją modelu, zgodnie z ideą autoenkodera jest więc tensor o tych samych wymiarach, co wejściowy.

Model został uzupełniony także o dwie istotne z punktu widzenia detekcji anomalii metody. Pierwsza z nich: „calculate_reconstruction_loss" przyjmuje tensor danych wejściowych i zwraca wektor błędów rekonstrukcji. Błąd jest w tym wypadku błędem średniokwadratowym, dodatkowo uśrednionym do jednej zmiennej. Druga metoda „calculate_anomaly_threshold" oblicza z wektora błędu próg wykrycia anomalii. Próg został ustalony jako maksymalna wartość błędu powiększona o 0,25 razy odchylenie standardowe.

```
# Create model instance
model = Model(n_neurons=64,
              n_timestamps=n_timestamps,
              n_features=x_train.shape[-1])
```

Rys. 5.8. Utworzenie obiektu modelu sieci neuronowej

Utworzony model sieci neuronowej posiada 64 dwukierunkowe neurony LSTM w pierwszej warstwie enkodera i 32 w drugiej. Długość sekwencji, czyli ilość próbek przypadająca na jedna sekwencje wynosi 32 i ilość zmiennych opisujących próbkę (n_features) to 2.

5.3. Uczenie modelu

Model skompilowano z użyciem funkcji błędu średniokwadratowego i optymalizatora „Adam". Optymalizator Adam pozwala na skuteczne trenowanie sieci z dynamicznie ustalanym krokiem uczenia (ang. learning rate). Ogranicza to ryzyko utknięcia gradientu w lokalnym minimum i powoduje szybsze uczenie się modelu.

```
# Compile model with loss function and optimizer
model.compile(loss='mse', optimizer='adam')
```

Rys. 5.9. Kompilacja modelu

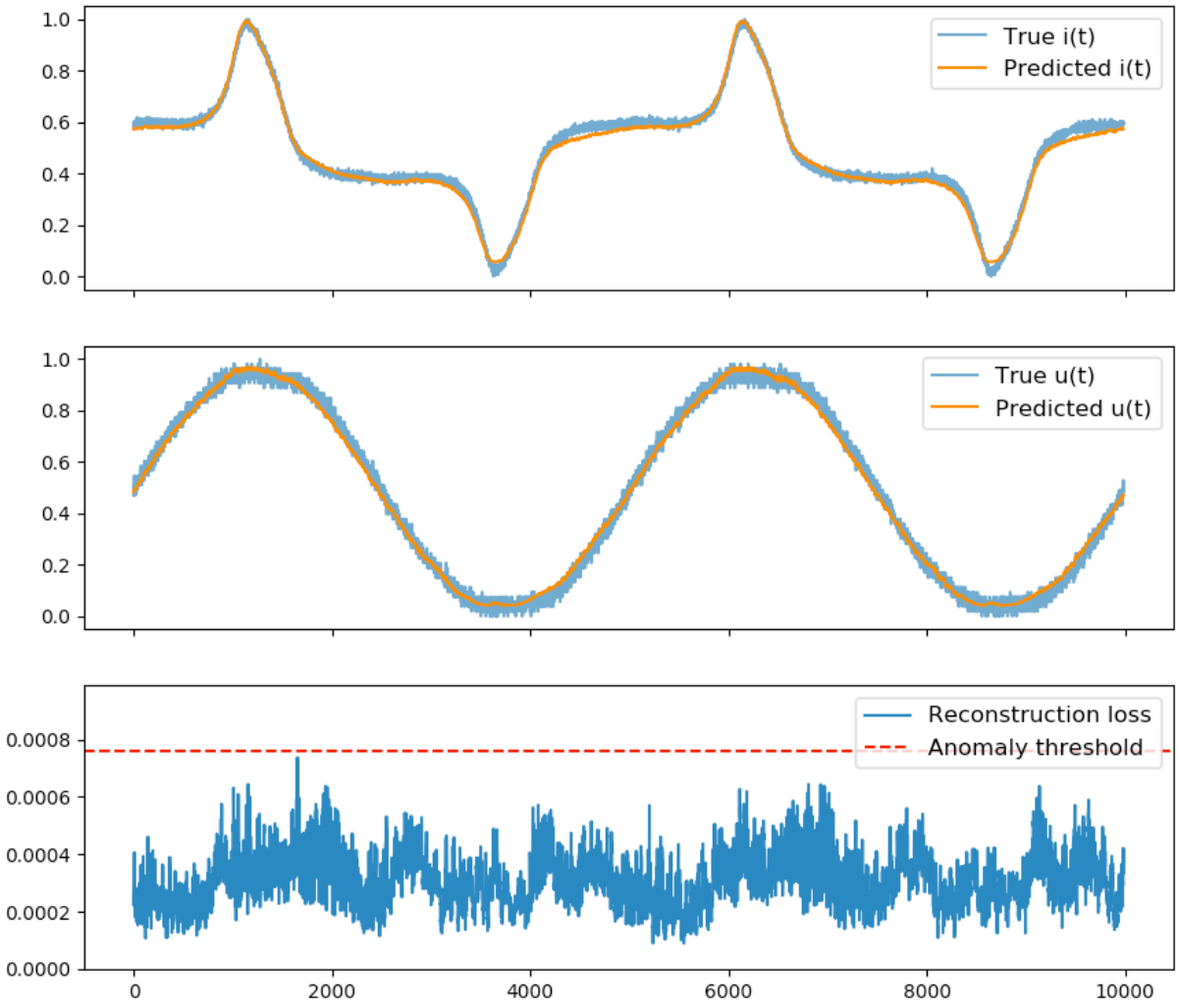
```
# Train model
steps_per_epoch = x_train.shape[0] // batch_size
model.fit(train_dataset, epochs=10, steps_per_epoch=steps_per_epoch)
```

Rys. 5.10. Trening modelu

Po skończonym treningu sprawdzono, jak model jest w stanie odwzorować dane wejściowe i jak wygląda przy tym jego błąd rekonstrukcji.

```
y_pred = model.predict(x_train)

loss = model.calculate_reconstruction_loss(x_train)
anomaly_threshold = model.calculate_anomaly_threshold(loss)

fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, figsize=(12, 12))

ax1.plot(x_train[:, 0, 0], label='True i(t)', alpha=0.7)
ax1.plot(y_pred[:, 0, 0], label='Predicted i(t)')
ax1.legend(loc='upper right')

ax2.plot(x_train[:, 0, 1], label='True u(t)', alpha=0.7)
ax2.plot(y_pred[:, 0, 1], label='Predicted u(t)')
ax2.legend(loc='upper right')

ax3.plot(loss, label='Reconstruction loss')
ax3.axhline(y=anomaly_threshold, label='Anomaly threshold', c='r', linestyle='dashed')
ax3.set_ylim([0, anomaly_threshold + (0.25 * anomaly_threshold)])
ax3.legend(loc='upper right')

plt.show()
```

Rys. 5.11. Predykcja danych treningowych i stworzenie wykresów wyników

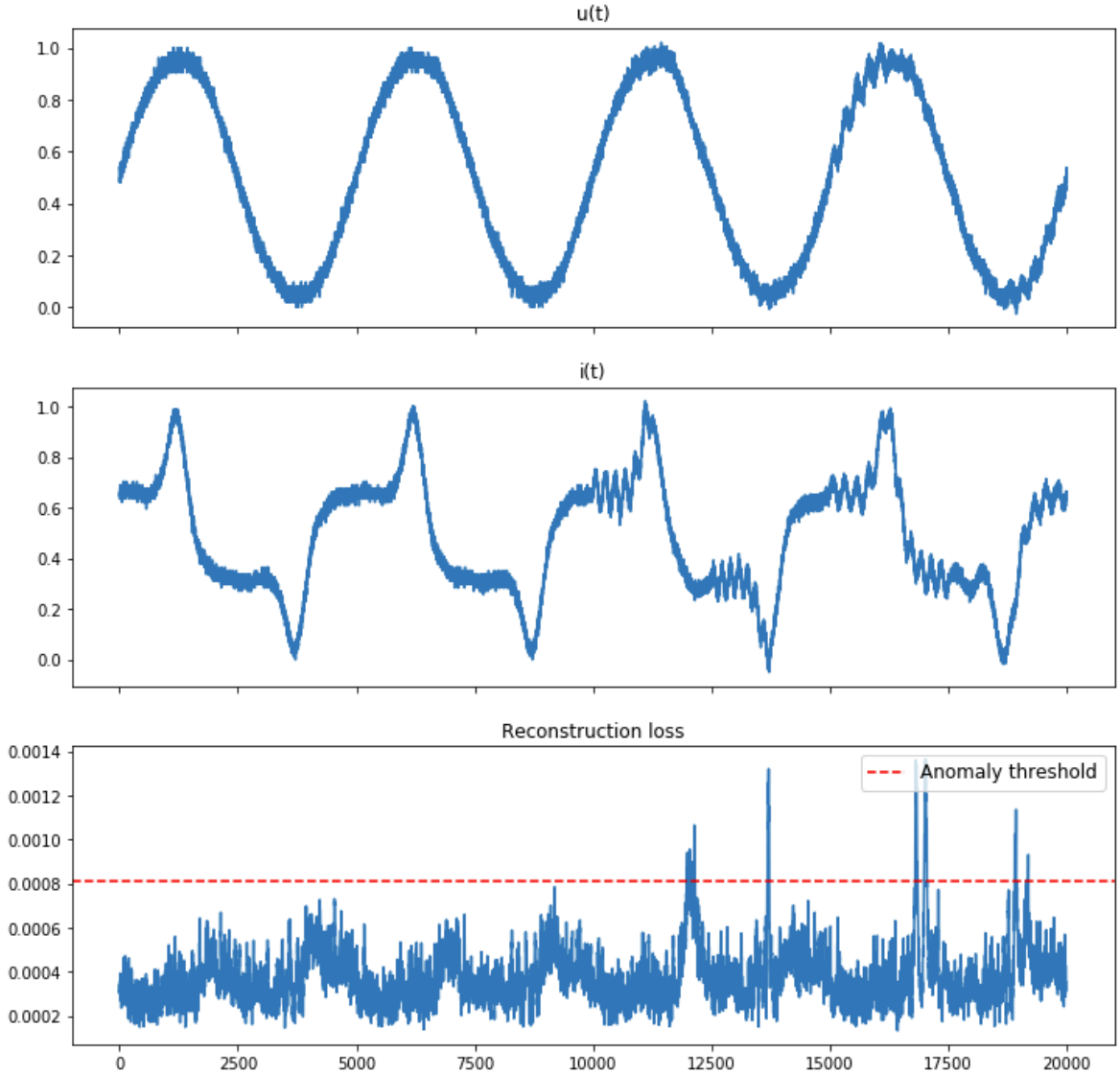


Rys. 5.12. Wyniki predykcji na danych treningowych

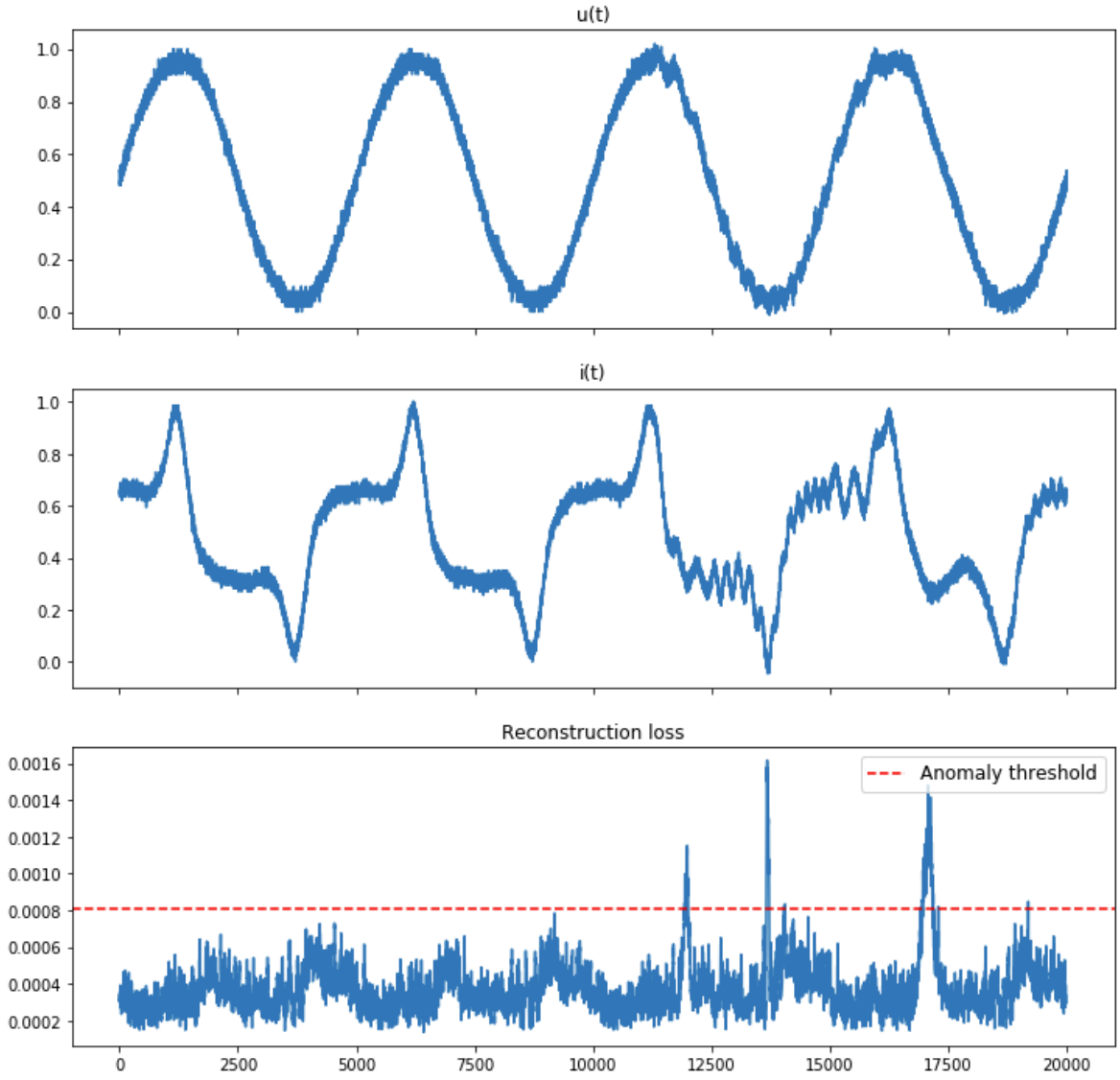
Jak widać model nauczył się dobrze odwzorowywać sekwencje wejściowe (pomarańczowe linie). Linią przerywaną na ostatnim wykresie zaznaczono obliczony za pomocą modelu próg wykrycia anomalii. Po jego przekroczeniu model zarejestruje, że w dostarczonej sekwencji rozkład danych różni się od widzianych podczas uczenia.

5.4. Walidacja

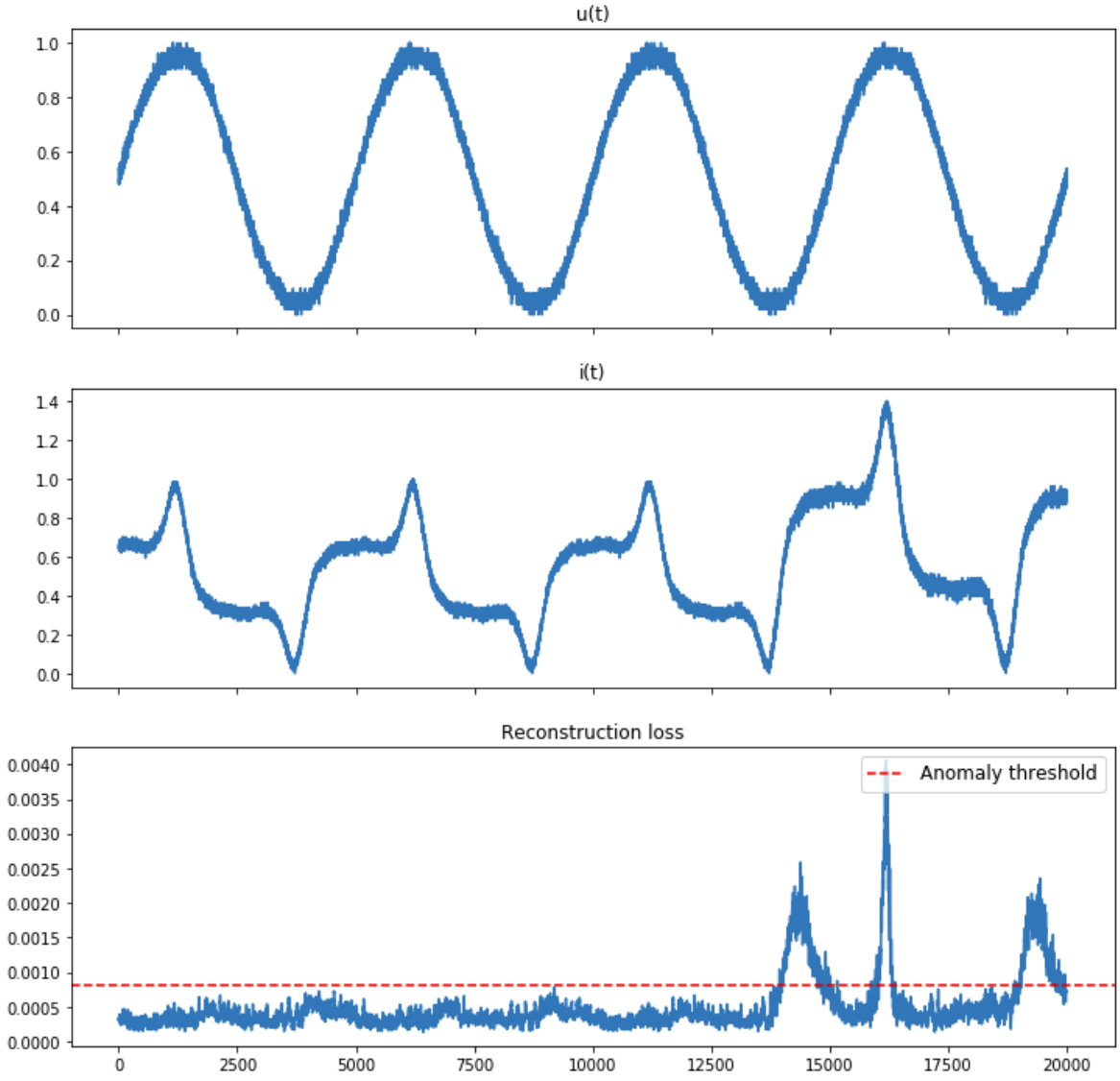
W celu przetestowania modelu zdecydowano się na sztuczne wygenerowanie odpowiednio zanieczyszczonych zbiorów danych. Pierwsze 2 okresy próbek zostały pozostawione w niezmienionej formie.



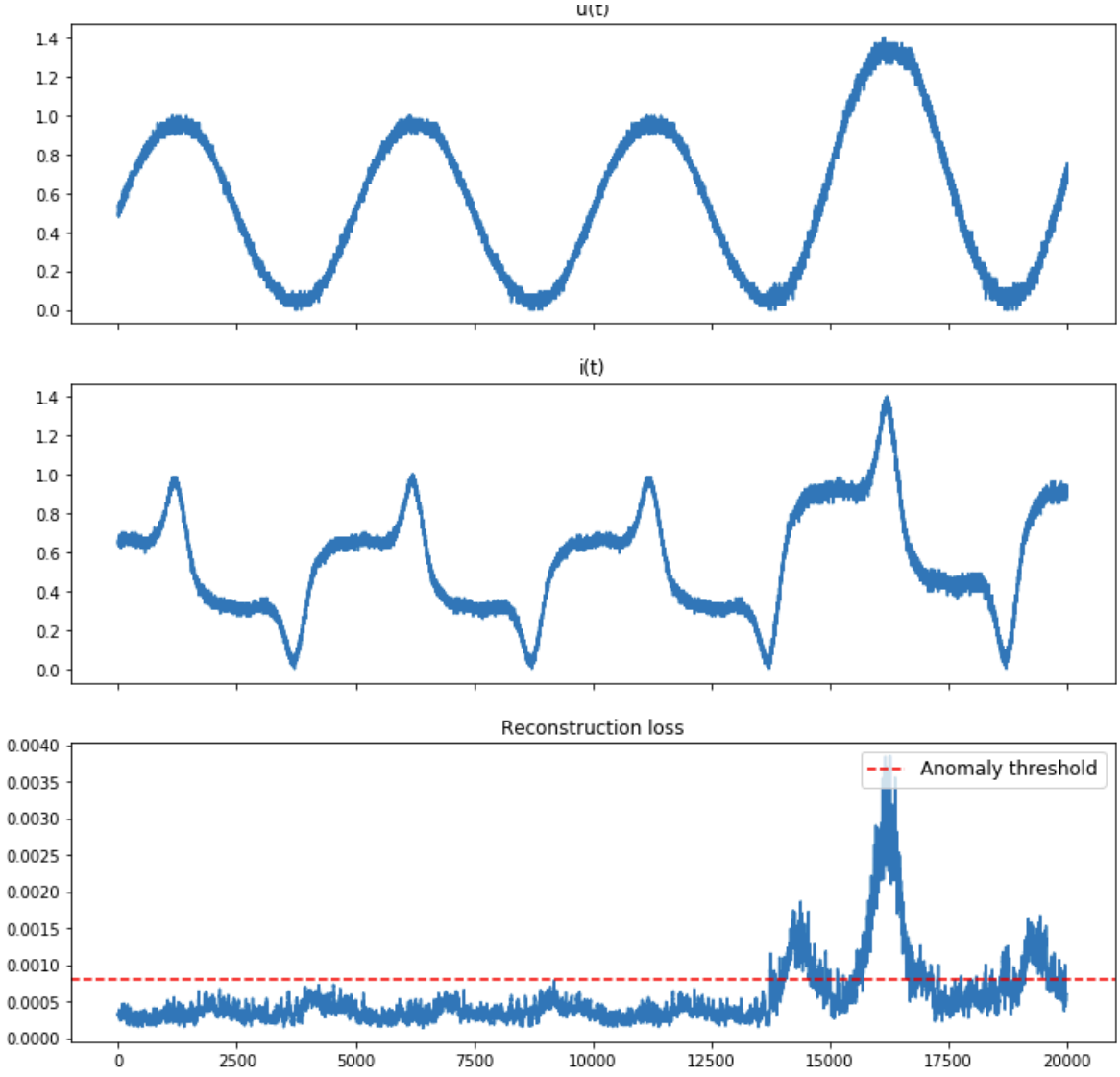
Rys. 5.13. Reakcja modelu na szum w sygnałach – pierwszy przypadek



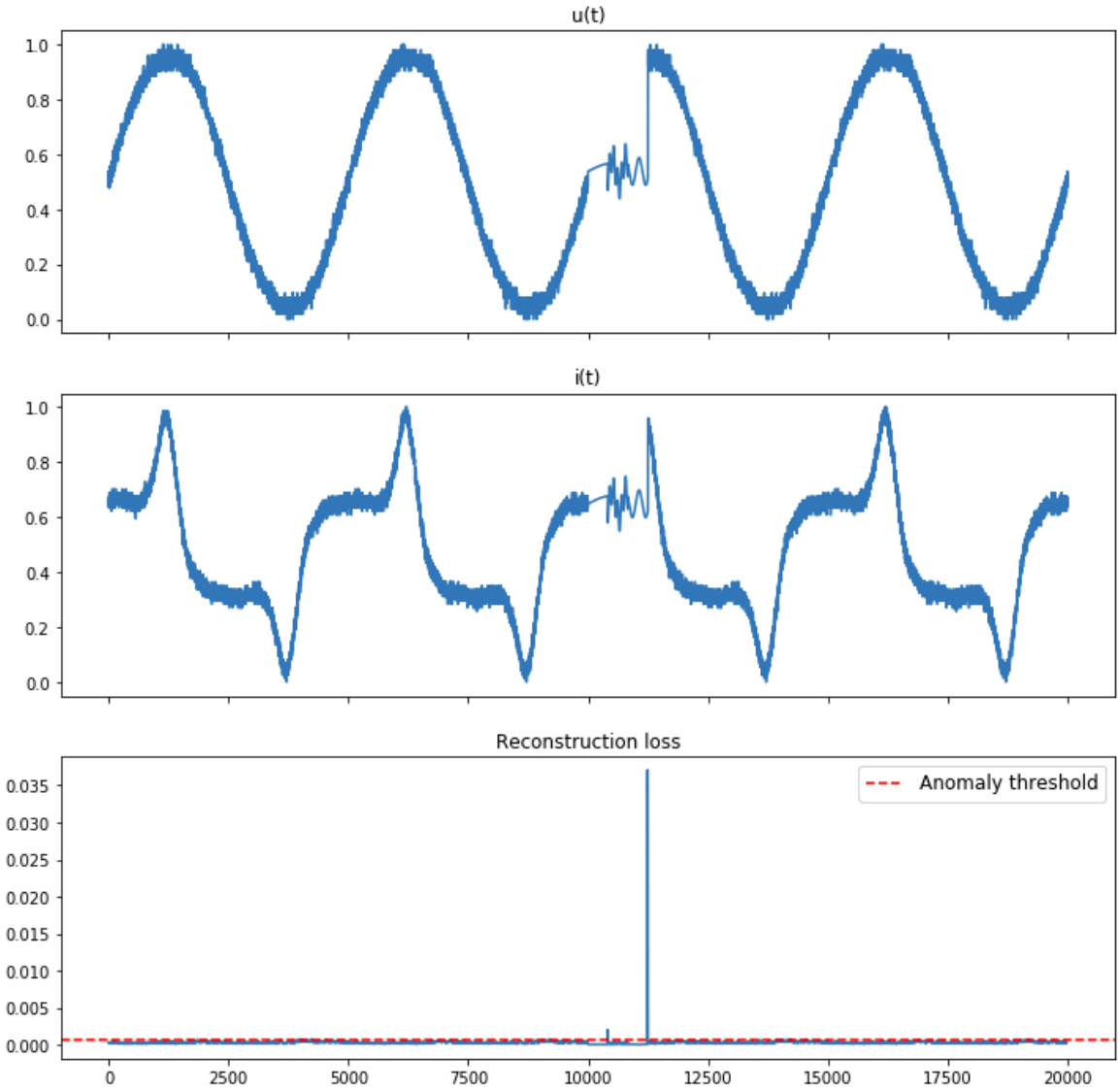
Rys. 5.14. Reakcja modelu na szum w sygnałach – drugi przypadek



Rys. 5.15. Reakcja modelu na wzrost prądu



Rys. 5.16. Reakcja modelu na wzrost napięcia i prądu



Rys. 5.17. Reakcja modelu na zapad napięcia

Jak widać na powyższych wykresach model poradził sobie z wykryciem zmian w rozkładzie danych wejściowych. Obliczony błąd rekonstrukcji w momencie wystąpienia w sygnałach zakłóceń znacząco przekroczył wyznaczoną podczas uczenia wartość progową. W takim wypadku zostaje wygenerowany sygnał o zaistniałej w sygnałach anomalii.

6. WNIOSKI

W pracy przedstawiono przykład możliwego zastosowania modelu uczenia maszynowego do konserwacji predykcyjnej warystora tlenkowo-cynkowego.

W przedstawionym przykładzie model został nauczony na sygnałach składających się z dwóch okresów i 10000 próbek. W rzeczywistych warunkach, takie podejście byłoby niedopuszczalne. Modelując „poprawność" urządzenia w celu późniejszego wykrycia anomalii należy posłużyć się znacznie większymi zbiorami danych. Poprzez wyuczenie modelu zbyt małym zbiorem danych łatwo jest spowodować jego przeuczenie. Podczas pracy w rzeczywistych warunkach algorytm mógłby powodować zbyt częste sygnalizowanie o degradacji struktury warystora. Nie znałby bowiem innych rozkładów sygnałów napięcia i prądu, które mogą pojawić się w trakcie prawidłowej pracy warystora, ale nie świadczą o jego degradacji.

Z wyżej wymienionego powodu podstawowym usprawnieniem, które należałoby zastosować przed wdrożeniem takiego modelu do produkcji jest uzyskanie bardziej wartościowych danych uczących. Dane powinny składać się ze znacznie dłuższego przedziału czasowego, aby model nauczył się generalizacji. Oprócz tego, jeśli istnieje taka możliwość dane można uzupełnić o dodatkowe zmienne jak np. temperatura czy wilgotność powietrza. W tym przypadku uzyskano by dodatkowe informacje, które mogłyby poprawić skuteczność działania modelu. Oprócz tego, aby zwiększyć margines bezpieczeństwa sygnalizacji modelu o zaistniałej anomalii można ustalić wartość progową błędu rekonstrukcji na wyższą wartość lub zaprogramować logikę, która sygnalizowałaby o anomalii dopiero po przekroczeniu wartości progowej konkretną ilość razy w danym oknie czasowym.

Drugim problemem, który należałoby wziąć pod uwagę przed wdrożeniem modelu do produkcji jest jego złożoność obliczeniowa i czas działania. Sieci neuronowe same w sobie są strukturami wymagającymi dużych zasobów obliczeniowych podczas swojej pracy. Ich rekurencyjne odpowiedniki są ponadto jeszcze bardziej złożone, bowiem przetwarzają jednocześnie całe sekwencje danych. Przedstawiony model analizuję pracę warystora badając sekwencję otrzymanych próbek napięcia i prądu. Problemem jest jednak fakt, że przy częstotliwości sieci 50Hz wykorzystane podczas treningu 10 tysięcy próbek zostaje wygenerowane w czasie 40 milisekund (dwa okresy) Aby monitorowanie stanu warystora mogło odbywać się faktycznie w czasie rzeczywistym model musi być w stanie nadążyć za dostarczanymi przez aparaturę pomiarową próbkami.

Z praktycznego punktu widzenia problem można rozwiązać na wiele sposobów. Podstawową operacją, którą należałoby wykonać bez względu na docelowe miejsce pracy modelu (mini-komputer, serwer czy architektura chmurowa) jest jego odchudzenie. Należy stworzyć możliwie jak najmniej skomplikowany model, które spełnia przedstawione oczekiwania. Oznaczałoby to przykładowo zmniejszenie liczby neuronów w warstwach rekurencyjnych lub pozbycie się warstw dwukierunkowych. Oprócz tego można zastosować techniki odchudzające jak kwantyzacja parametrów (ang. quantization) lub tzw. przycinanie (ang. pruning), polegające na wykorzystaniu podczas pracy tylko części neuronów, a więc i znacznej redukcji operacji obliczeniowych.

Jeśli model miałby działać na osprzęcie o ograniczonej mocy obliczeniowej warto rozważyć uruchomienie go w innym środowisku niż Python. Choć jest on niezwykle popularnym językiem, doskonale sprawdzającym się na etapie modelowania, w momencie, gdy należy liczyć się z każdą milisekundą czasu odpowiedzi i megabajtem wykorzystanej pamięci może

okazać się on zbyt ciężki. Docelowo wyuczony model można skompresować do postaci binarnej i wczytać za pomocą bardziej wydajnych języków jak C++, Java. Biblioteka tensorflow umożliwia takie operacje i jest to często poruszane zagadnienie, jeśli modele uczenia maszynowego pracują na komputerach o ograniczonych zasobach jak mini komputery lub smartfony.

Wykorzystanie sztucznych sieci neuronowych do diagnostyki aparatury przeciwprzepięciowej nie jest trywialnym zagadnieniem. Nie istnieje uniwersalna metoda bądź architektura modelu, która sprawdziłaby się w każdym przypadku. Każde urządzenie, warunki pracy, parametry oraz generowane przez aparaturę pomiarową dane cechują się swoją unikalną specyfiką. Dlatego każda próba stworzenia nawet najprostszego systemu konserwacji predykcyjnej powinna być poprzedzona rzetelną analizą wszystkich możliwych zmiennych. Sieci neuronowe są w rzeczywistości przybliżeniami skomplikowanych funkcji matematycznych. Ich zdolności do odpowiednio dokładnych przybliżeń podyktowane są jakością danych uczących. Analiza danych pomiarowych które wykorzystane będą przez algorytm jest zatem kluczowym etapem, któremu należy poświęcić szczególną uwagę.

System konserwacji predykcyjnej oparty o architekturę sztucznej sieci neuronowej nie musi być jedynym systemem monitorującym stan warystorów. Może być to rodzaj uzupełnienia istniejących, bardziej znanych i zaufanych systemów. Z racji praktycznie zerowej interpretowalności modelu sieci neuronowej może być on problematyczny w przypadkach, kiedy istotna jest informacja o przyczynie degradacji warystora. Sieć posiadająca w swojej strukturze na przykład 350 tysięcy parametrów nie będzie w stanie dostarczyć tak szczegółowej informacji.

Konserwacja predykcyjna, w stosunku do klasycznej, prewencyjnej wiąże się z większymi nakładami czasowymi i finansowymi na początku inwestycji. Ponadto w przypadku, gdy dane nie są w ogóle dostępne konieczne jest zbudowanie infrastruktury pomiarowej i magazynującej te dane. Jest to jednak przedsięwzięcie, na które decyduje się coraz więcej przedsiębiorstw z powodu wymiernych korzyści w długim terminie.

7. LITERATURA

[1] Tadeusiewicz R. “Elementarne wprowadzenie do sieci neuronowych z przykładowymi programami (1998)

[2] Kostyła P., Jaroszewski M., „Application of artificial neural networks to determine the parameters of the substitute model of ZnO vartistors based on the measurement of instantaneous current and voltage”

[3] Olesz M. “Diagnostyka niskonapięciowych warystorowych ograniczników przepięć” XXII Seminarium „Zasosowanie komputerów w nauce i technice” Oddział Gdański PTETiS Referat nr 25, 2012

[4] Malek Z., Syafii, Ahmad M., Aulia, Ulfiah S., „Condition based monitoring of gapless surge arrester using electrical and thermal parameters” Novizon et al 2019 IOP Conf. Ser.: Mater. Sci. Eng. 602 01207

[5] Zulkurnain A. “Correlation between Third Harmonic Leakage Current and Thermography Image of Zinc Oxide Surge Arrester for Fault Monitoring Using Artificial Neural Network” Applied Mechanics and Materials Vol. 554 (2014) pp 598-602

[6] Kostyła P., Jaroszewski M., Wacławek Z. „Metoda określania wartości parametrów wewnętrznych modelu warystora na podstawie pomiaru wartości chwilowych prądu i napięcia” XI Sympozjum „Problemy eksploatacji układów izolacyjnych wysokiego napięcia”, Krynica 25-28 września 2007

[7] Pereira J. „Unsupervised Anomaly Detection In Time Series Data Using Deep Learning” (2018)