Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Systemy Informacyjno-Decyzyjne

Zastosowanie algorytmów upraszczania sztucznych sieci neuronowych w algorytmach regulacji

Damian Koss
Numer albumu 293128

promotor
dr inż. Patryk Chaber

WARSZAWA 2021

Zastosowanie algorytmów upraszczania sztucznych sieci neuronowych w algorytmach regulacji

Streszczenie. To jest streszczenie długiej i ciężkiej pracy nie wiem w sumie co tutaj więcej napisać ale coś trzeba.

Słowa kluczowe: Sieci neuronowe, regulacja predykcyjna, upraszczanie sieci neuronowej

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Politechnika Warszawska

……..
miejscowość i data

……..
imię i nazwisko studenta

……..
numer albumu

…….…….
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1. Wstęp 9
2. Przegląd literatury przedmiotu 11
3. Opis teoretyczny rozwiązania 14
3.1. Regulacja Predykcyjna 14
3.1.1. Algorytm DMC 14
3.1.2. Obiekt regulacji 16
3.2. Sztuczne Sieci Neuronowe 17
3.2.1. Architektura sieci 18
3.2.2. Wsteczna propagacja gradientu 19
3.2.3. Algorytm OBD 21
3.3. Strategia eksperymentów 22
4. Opis wybranych elementów implementacji 25
4.1. Algorytm DMC z obiektem regulacji 25
4.2. Architektura sieci neuronowej 27
4.3. Algorytm OBD 29
5. Wyniki pracy 33
5.1. Generacja danych 33
5.2. Wybór struktury sieci 34
5.3. Zastosowanie algorytmu OBD 36
5.4. Zastosowanie innych obiektów regulacji 36
6. Podsumowanie 37
Bibliografia 39
Wykaz symboli i skrótów 40
Spis rysunków 40
Spis tabel 40
Spis załączników 40

1. Wstęp

Ciągły rozwój nauki i techniki sprawia, że wiele dziedzin życia zmienia się na naszych oczach. Obserwujemy coraz to większe dążenie do automatyzacji i informatyzacji otaczającego nas świata. Zmiany, które obserwujemy nie dotyczą jedynie przemysłu czy wąskiego grona wysoko-wyspecjalizowanych profesji ale często wkraczają do naszego życia codziennego. Komputery i roboty są w stanie przeprowadzić za nas nie tylko skomplikowaną reakcję chemiczną ale również wymieszać składniki na ciasto w odpowiednich proporcjach.

Wspomniane proporcje są tutaj kluczowe, stąd też z automatyzacją nieodzownie łączą się takie zagadnienia jak sterowanie i regulacja. Praktycznie każde mniej lub bardziej skomplikowane zadanie możemy zapisać w postaci konkretnego algorytmu, a następnie zrealizować go poprzez kontrolę pewnych ustalonych parametrów. Doskonałym przykładem jest tutaj dostosowanie temperatury wody w wannie lub prowadzenie samochodu. W obu tych przypadkach dążymy do osiągnięcia pewnego pożądanego rezultatu, prawidłowej temperatury wody lub odpowiedniego toru jazdy.

Automatyka rozwija się już od średniowiecza i w tym czasie można wskazać wiele przełomowych momentów. Za jedne z głównych możemy uznać: prace J. C. Maxwella z zakresu stabilności regulacji (1863 r.), zapoczątkowanie metod częstotliwościowych analizy i syntezy układów przez H. Nyquista (1932 r.) czy w końcu zastosowanie regulacji w strukturze zamkniętej ze sprzężeniem zwrotnym poprzez opracowanie regulatora PID. Obecnie jedną z powszechnie uznanych i stosowanych metod jest regulacja predykcyjna. Jednak potrzeba automatyzacji coraz to nowych dziedzin naszego życia wymusza ciągły rozwój w obszarze automatyki i konieczność udoskonalania istniejących rozwiązań lub tworzenia zupełnie nowych. Stosowane algorytmy mogą okazać się niewystarczające gdy zadania, które przed nimi stawiamy staną się bardziej złożone i wymagać będą uwzględnienia wielu niezależnych czynników. Z pomocą mogą przyjść tutaj niewątpliwie liczne osiągnięcia w dziedzinie przetwarzania informacji, a za jedno z największych w tej dziedzinie należy uznać sztuczne sieci neuronowe.

Rosnąca popularność i uznanie sztucznych sieci neuronowych wiąże się z ich zdolnością łatwego adaptowania się do rozwiązywania różnorodnych problemów obliczeniowych. Możliwość odwzorowania nauczonych wzorców i generalizacji nabytej wiedzy sprawia, że stały się szczytowym osiągnięciem w obszarze sztucznej inteligencji. Z tego względu oczywistym wydaje się chęć wykorzystania ich zalet i sprawdzenia zdolności adaptacyjnych także w obszarze regulacji.

Głównym celem pracy jest zastosowanie jednej z metod sztucznej inteligencji w obszarze regulacji predykcyjnej. Dzięki zastosowaniu możliwie prostych i podstawowych struktur, a za taką możemy uznać jednokierunkową sztuczną sieć neuronową, możliwe będzie przedstawienie ogólnego potencjału wykorzystania sztucznych sieci neuronowych do różnorodnych zadań stawianych przed regulatorami. Ważnym aspektem będzie także

wykorzystanie algorytmu upraszczania sztucznej sieci neuronowej, który ma za zadanie poprawić możliwość generalizacji danego problemu.

W pierwszym rozdziale pracy przybliżona zostanie literatura przedmiotu, przegląd aktualnych rozwiązań i omówienie możliwości ich uzupełnienia. Następnie zaprezentowane zostaną wszelkie założenia teoretyczne, szczegółowy opis stosowanych struktur i sposób ich wykorzystania. W kolejnym rozdziale znajdą się właściwe wyniki przeprowadzonych eksperymentów. Na końcu uzyskane rezultaty zostaną podsumowane co pozwoli na wyciągnięcie ogólnych wniosków z niniejszej pracy.

2. Przegląd literatury przedmiotu

Obszerność dostępnych opracowań z zakresu automatyki jak i sztucznych sieci neuronowych sprawia, że omówienie istniejących rozwiązań zostanie podzielone na dwa etapy. Pierwszym z nich jest przegląd prac z zakresu sztucznych sieci neuronowych. Omówione zostaną zarówno pozycje teoretycznych wprowadzające w dziedzinę sztucznej inteligencji i pozwalające poczynić niezbędne założenia jak i przykłady praktycznego zastosowania wybranych struktur ze szczególnym uwzględnieniem tych, w których wykorzystane zostały algorytmy upraszczania sieci sieci neuronowych. Następnie przeanalizuję dostępną literaturę z zakresu regulacji predykcyjnej i wskaże na przykłady zastosowania sieci neuronowych w tej dziedzinie.

Rozpoczynając pracę nad zagadnieniem sztucznych sieci neuronowych warto zapoznać się z dwoma pozycjami [1] oraz [2]. Oba opracowania stanowią kompleksowy przegląd istniejących metod i mogą stanowić punkt wyjścia do każdej analizy zajmującej się tematyką sztucznych sieci neuronowych. Lektura dwóch pozycji pozwoliła mi zdecydować się na wybór jednokierunkowej sieci typu sigmoidalnego z jedną warstwą ukrytą. Jest to możliwie prosta struktura, która pozwoli zarówno na samodzielną implementację jak i będzie stanowić dobry punkt wyjścia do możliwiej dalszej analizy z wykorzystaniem bardziej skomplikowanych struktur. Moją decyzję opieram także na [3] gdzie autor wykazuje, że podstawowa struktura jaką jest jednokierunkowa sieć z jedną warstwą ukrytą może stanowić uniwersalny aproksymator przy założeniu dostatecznej liczby neuronów w warstwie ukrytej oraz prawidłowego doboru funkcji aktywacji. Szczegółowe umówienie wybranej struktury przedstawione zostanie w kolejnym rozdziale pracy.

Kolejnym zagadnieniem jest wybór metody redukcji sieci i jak wskazuje [4] za jedno z lepszych rozwiązań możemy uznać metody wrażliwościowe, a w szczególności metodę Optimal Brain Damage (OBD) zaproponowaną w pracy [5]. Implementacja i sprawdzenie wskazanej metody w kontekście regulacji predykcyjnej będzie ciekawym zagadnieniem i jednym z głównych celów niniejszej pracy. Możemy wskazać przykłady prac, które pozwalają nam przypuszczać, że zastosowanie OBD rzeczywiście poprawi działanie proponowanej architektury i zwiększy zdolność generalizacji. W pracy [6] autorzy badają efektywność zastosowania metody OBD do redukcji rekurencyjnej sieci neuronowej z jedną warstwą ukrytą. Dzięki zastosowaniu opisywanej techniki dokumentują redukcję 60 procent wag co przyczynia się do 30-to procentowego spadku błędu popełnianego przez model. Dodatkowo warto zwrócić uwagę, że analiza została przeprowadzona na podstawie modelowania chemicznej reakcji neutralizacji, a zatem na przykładzie procesu wysoce dynamicznego. Niniejsza praca zajmować się będzie ogólnym porównaniem dwóch metod regulacji z wykorzystaniem teoretycznego obiektu, jednak mimo to osiągnięte przez autorów rezultaty możemy z pewnym przybliżeniem traktować jako punkt odniesienia w trakcie dalszej analizy.

Następnie warto wskazać prace porównujące różne metody redukcji sieci neurono-

wych, a zwłaszcza na porównanie OBD z inna metodą wrażliwością Optimal Brain Surgeon (OBS) szczegółowo opisaną w [4]. W pracy [7] porównana została efektywność trzech różnych metod redukcji sieci: Magnitude Based Pruning (MP), OBD i OBS. Analizę przeprowadzono z wykorzystaniem jednokierunkowej sieci neuronowe użytej do klasyfikacji obiektów z zdjęć terenu. Przedmiot analizy jest co prawda zupełnie odmienny jednak zbliżona architektura sieci stanowi tutaj podstawę do dokładniejszego przyjrzenia się wynikom uzyskanym przez autorów. Podobnie jak w przypadku poprzedniej pracy strukturę sieci udało zredukować się o 60 procent jednak tym razem wiązało się to jedynie z niewielką choć zauważalną poprawą klasyfikacji. Wartym zauważenia natomiast jest fakt, że OBS pozwolił na osiągnięcie najlepszych rezultatów spośród trzech metod. Co więcej nie jest to jedyna praca, w której wykazana zostaje taka zależność. Autorzy w pracy [8] porównując dwie wspomniane wcześniej metody redukcji sieci wskazują na istotny problem, a mianowicie, że OBD nie zawsze redukuje prawidłowe wagi sieci, a także w większości przypadków prowadzi do mniejszej redukcji sieci niż OBS. Analiza przeprowadzona została na podstawie przykładowego zbioru danych używanego do weryfikacji różnorodnych zagadnień uczenia maszynowego MONK's Problems Data Set. Na podstawie przedstawionych opracowań warto rozważyć użycie nie tylko metody OBD ale poszerzenie analizy o OBS, jednak decyzja o dalszym rozwoju pracy uwarunkowana będzie jakością wyników uzyskanych za pomocą algorytmu OBD.

Po zapoznaniu się z pracami kluczowymi ze względu na aspekt doboru architektury sieci należy spojrzeć na przykłady zastosowania sieci neuronowych w regulacji predykcyjnej. W pracy [9] autorzy badają wpływ wykorzystanie wielowarstwowej jednokierunkowej sieci neuronowej do kontroli nieliniowego wieloczynnikowego procesu chemicznego jakim jest wytrawianie metalu (Steel Pickling). Wykazane zostaje, że w przypadku procesu o podanej charakterystyce tradycyjne metody sterowania okazują się dawać gorsze rezultaty niż podejście oparte o sztuczne sieci neuronowe, które to przykładowo zdecydowanie lepiej radziły sobie z eliminacją oscylacji.

Kolejnym przykładem zastosowania sieci neuronowych do regulacji procesu chemicznego jest praca [10] W tym przypadku regulacja ponownie dotyczy złożonego nieliniowego procesu chemicznego, a w celu jego regulacji użyto rekurencyjnej sieci neuronowej z jedną warstwą ukrytą. Warto zauważyć tutaj fakt, że regulacja oparta o sieci neuronowe porównana zostaje z klasycznym regulatorem PID. Klasyczne podejście prowadzi do wyraźnego prze-regulowania układu oraz zmniejsza stabilność układu. W artykule jednoznacznie wykazana zostaje wyższość metody regulacji opartej o sztuczne sieci neuronowe jednak autorzy wskazują, że jest to głównie związane z dużą złożonością regulowanego procesu. Na tej podstawie możemy przypuszczać, że obserwować będziemy coraz więcej problemów regulacji gdzie tradycyjne metody mogą okazać się niewystarczające i szukanie rozwiązań opartych o sieci neuronowe stanie się niezbędne.

Ostatnim z przykładów, który chce omówić jest praca [11], w której zaprezentowano

przykład wykorzystania regulacji opartej o sieci neuronowe do sterowania system inżynierii sanitarnej w budynkach mieszkalnych. Autorzy analizują wyniki uzyskane za pomocą kilku różnych architektur sieci, a jako najbardziej typowy wybór podają jednokierunkową sieć o jednej lub więcej warstwach ukrytych. Przeprowadzona analiza wskazuje, że zastosowanie regulatora opartego o sztuczne sieci neuronowe pozwoliło zredukować konsumpcję energii w zakresie nawet do 50% w zależności od analizowanego przypadku.

Wyniki wszystkich trzech prac jednoznacznie wskazują na korzyści wynikające z zastosowania sztucznych sieci neuronowych w systemach regulacji. Architektura sieci w przypadku żadnej z prac nie odpowiadała tej wybranej do mojej pracy, co więcej w żadnym z podejść nie został wykorzystany algorytm OBD lub OBS. Pokazuje to, że przeprowadzenie analizy w takim zakresie może nieść za sobą duża wartość zarówno naukową jak i poznawczą.

3. Opis teoretyczny rozwiązania

W rozdziale przedstawione zostaną wszystkie założenia teoretyczne niezbędne do pełnego zrozumienia przedmiotu pracy i późniejszej interpretacji uzyskanych rezultatów. Na początku przedstawiona zostanie idea regulacji predykcyjnej wraz z algorytmem DMC oraz obiektem regulacji, który zostanie użyty w trakcie eksperymentów. W tej części pracy przydatne były opracowania [12] oraz [13]. Następnie omówię architekturę sztucznej sieci neuronowej wraz z opisem funkcji aktywacji i metodą uczenia sieci. Tutaj szczególnie pomocne okazały się pozycje [14], [2] oraz [15]. Uwagę należy zwrócić tutaj na opis algorytmu OBD czyli użytą w tej pracy metodę upraszczania sieci neuronowej. Na końcu omówiona zostanie strategia, która została wykorzystana w trakcie wykonanych eksperymentów. Lektura rozdziału powinna zaznajomić czytelnika z wszystkimi kluczowymi zagadnieniami z punktu widzenia niniejszej pracy.

3.1. Regulacja Predykcyjna

Regulacja predykcyjna uznawana jest za jedną z zaawansowanych technik regulacji, które to zastąpiły uprzednio powszechnie stosowane regulatory PID. Dla wielowymiarowych i skomplikowanych procesów, regulacja w oparciu o identyfikacje jednego punktu charakterystyki obiektu, jak to wygląda w regulatorze PID, może okazać się nieefektywna. Rozwiązaniem jest tutaj wykorzystanie zasady przesuwanego horyzontu i wyznaczanie w każdej chwili kT, gdzie T oznacza okres próbkowania, sekwencji przyszłych wartości sygnału sterującego. Idea każdego z algorytmów regulacji predykcyjnej polega na wyznaczeniu w każdej iteracji wektora przyrostów sygnału sterującego.

$$\Delta U(k) = [\Delta u(k|k) \quad \Delta u(k+1|k) \quad \Delta u(k+2|k) \quad \dots \quad \Delta u(k+N_u-1|k)]^T \tag{1}$$

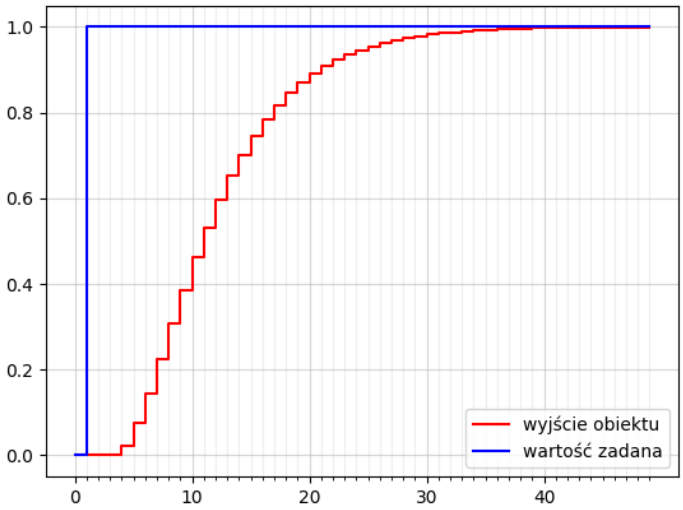
gdzie przez N_u oznaczamy horyzont sterowania, a notacja $\Delta u(k+p|k)$ oznacza przyrost sygnału sterującego obliczony w chwili k, który ma być wykorzystany do sterowania w chwili $k+p$. W istocie jednak do sterowania wykorzystuje się tylko pierwszą wartość wyznaczanego wektora i prawo regulacji w kolejnych iteracjach przyjmuje postać

$$u(k) = u(k|k) = \Delta u(k|k) + u(k-1) \tag{2}$$

3.1.1. Algorytm DMC

Główna idea algorytmu DMC (Dynamic Matrix Control), przedstawionego w [16], opiera się na wykorzystaniu modelu odpowiedzi skokowej do predykcji. Algorytm DMC identyfikuje dynamikę obiektu regulacji za pomocą dyskretnych odpowiedzi skokowych, które są reakcją wyjścia w kolejnych okresach próbkowania na jednostkowy skok sygnału sterującego. Na rysunku 3.1. przedstawiono przykładową odpowiedź skokową obiektu regulacji, który to opisany zostanie w kolejnym podrozdziale. Następnie tak wyznaczona

odpowiedzi skokowa obiektu $\{s_1, s_2, s_3, \ldots s_D\}$ wykorzystywana jest w algorytmie DMC do wyznaczenia najbardziej optymalnych wartości sterowania. Wielkość D oznacza horyzont dynamiki obiektu i powinna zostać dobrana eksperymentalnie do takiej wartości po której wyjście obiektu ulega stabilizacji.



Rysunek 3.1. Przykładowa odpowiedź wyjścia obiektu na jednostkowy skok sterowania

W każdej iteracji wektor (1) wyznaczany jest w wyniku minimalizacji wskaźnika jakości, który zapisany w formie wektorowo-macierzowej przyjmuje postać

$$J(k) = \left\| Y^{zad}(k) - \hat{Y}(k) \right\|_{\mathbf{\Psi}}^2 + \|\Delta U(k)\|_{\mathbf{\Lambda}}^2 \tag{3}$$

gdzie wektory wartości zadanej $Y^{zad}(k)$ oraz prognozowanej trajektorii wyjścia $\hat{Y}(k)$ o długości N będącej horyzontem predykcji definiowane są w następujący sposób

$$Y^{zad}(k) = \begin{bmatrix} y^{zad}(k+1|k) \\ \vdots \\ y^{zad}(k+N|k) \end{bmatrix}, \quad \hat{Y}(k) = \begin{bmatrix} \hat{y}(k+1|k) \\ \vdots \\ \hat{y}(k+N|k) \end{bmatrix} \tag{4}$$

macierze Λ oraz Ψ są macierzami diagonalnymi, a w większości przypadków przyjmują postać macierzy identyczności i takie również założenie przyjęte zostało w tej pracy.

Następnie korzystając z przekształceń szczegółowo omówionych w [12] zapisujemy funkcję kryterialną w postaci

$$J(k) = \left\| Y^{zad}(k) - Y(k) - \boldsymbol{M}^P \Delta U^P(k) - \boldsymbol{M} \Delta U(k) \right\|_{\mathbf{\Psi}}^2 + \|\Delta U(k)\|_{\mathbf{\Lambda}}^2 \tag{5}$$

gdzie macierz M o wymiarowości $N \times N_u$ ma strukturę

$$\boldsymbol{M} = \begin{bmatrix} s_1 & 0 & \cdots & 0 \\ s_2 & s_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ s_N & s_{N-1} & \cdots & s_{N-N_u+1} \end{bmatrix} \tag{6}$$

, macierz M^P o wymiarowości $N \times (D-1)$

$$\boldsymbol{M^P} = \begin{bmatrix} s_2 - s_1 & s_3 - s_2 & \cdots & s_D - s_{D-1} \\ s_3 - s_1 & s_4 - s_2 & \cdots & s_{D+1} - S_{D-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{N+1} - s_1 & s_{N+2} - s_2 & \cdots & s_{N+D-1} - s_{D-1} \end{bmatrix} \tag{7}$$

, natomiast wektor $\Delta U^P(k)$ zawiera przeszłe $D-1$ wartości zamian sygnału sterującego $\Delta u(k-i)$.

Zauważając, że funkcja jest ściśle wypukła, przyrównujemy do zera wektor gradientu funkcji i otrzymujemy wektor optymalnych przyrostów sterowania

$$\Delta U(k) = (\boldsymbol{M}^T \boldsymbol{\Psi} \boldsymbol{M} + \boldsymbol{\Lambda})^{-1} \boldsymbol{M}^T \boldsymbol{\Psi} (Y^{zad}(k) - Y(k) - \boldsymbol{M}^P \Delta U^P(k)) \tag{8}$$

, gdzie początkową część możemy zapisać w formie

$$\boldsymbol{K} = (\boldsymbol{M}^T \boldsymbol{\Psi} \boldsymbol{M} + \boldsymbol{\Lambda})^{-1} \boldsymbol{M}^T \boldsymbol{\Psi} \tag{9}$$

gdzie macierz $\boldsymbol{K}$ o wymiarowości $N_u \times N$ wyznacza jest jednokrotnie w trakcie projektowania algorytmu. Równanie (8) stanowi podstawą działania algorytmu i posłuży do dalszej bezpośredniej implementacji.

3.1.2. Obiekt regulacji

Aby przeprowadzić niezbędne eksperymenty i dokonać porównania różnych regulatorów niezbędne jest wybranie i zaimplementowanie obiektu regulacji. W niniejszej pracy w roli obiektu wykorzystany zostanie człon inercyjny drugiego rzędu z opóźnieniem. Wybrany układ najłatwiej przedstawić w postaci równania różnicowego

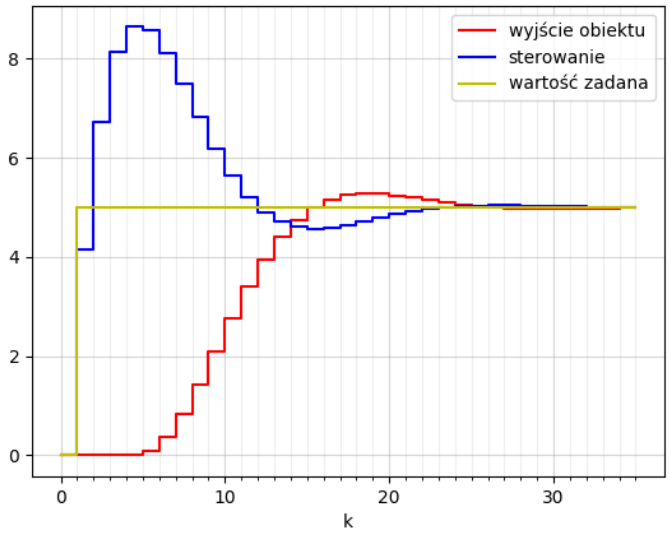
$$y(k) = b_1 u(k - T_d - 1) + b_2 u(k - T_d - 2) - a_1 y(k-1) - a_2 y(k-2) \tag{10}$$

gdzie

$$
\begin{aligned}
a_1 &= -\alpha_1 - \alpha_2 \\
a_2 &= \alpha_1 \alpha_2 \\
\alpha_1 &= e^{-\frac{1}{T_1}} \\
\alpha_2 &= e^{-\frac{1}{T_2}} \\
b_1 &= \frac{K}{T_1 - T_2}[T_1(1-\alpha_1) - T_2(1-\alpha_2)] \\
b_2 &= \frac{K}{T_1 - T_2}[\alpha_1 T_2(1-\alpha_2) - \alpha_2 T_1(1-\alpha_1)]
\end{aligned}
$$

Dzięki takiej prezentacji i implementacji ogólnej klasy układu regulacji mamy możliwość identyfikacji poszczególnych układów poprzez cztery parametry T_1, T_2, K, T_d. Jest to istotna własność dzięki, której w łatwy sposób można dokonywać modyfikacji obiektu i sprawdzać zachowanie regulatora w poszczególnych przypadkach.

Dla lepszego zrozumienia zagadnienia warto przyjrzeć się przebiegowi regulacji predykcyjnej DMC dla jednego wybranego układu regulacji. Eksperyment przedstawiony na rysunku 3.2 przeprowadzony został dla następujących parametrów $T_1 = 5$, $T_2 = 4$, $K = 1$, $T_d = 3$. Warto zaznaczyć, że dla tego samego układ wcześniej pokazana została odpowiedź skokowa z rysunku 3.1.



Rysunek 3.2. Regulacja DMC dla wybranego obiektu

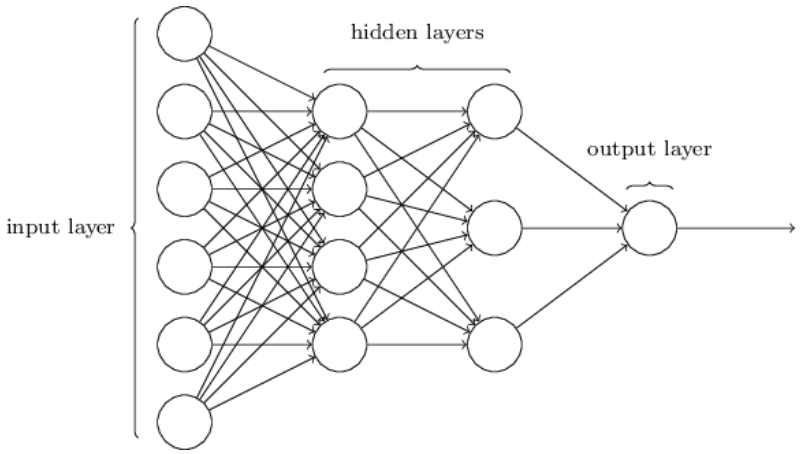
3.2. Sztuczne Sieci Neuronowe

Sztuczne sieci neuronowe stanowią jedną z najczęściej omawianych metod przetwarzania informacji w ostatnich lat. Wielką popularność zyskały niewątpliwie dzięki swojemu

interdyscyplinarnemu charakterowy, łącza one osiągnięcia biocybernetyki, elektroniki, matematyki stosowanej, statystyki oraz automatyki. Istotą działania sieci neuronowych jest próba odtworzenia zjawisk zachodzących w systemach nerwowych istot żywych poprzez zastosowanie nowoczesnych rozwiązań technologicznych.

3.2.1. Architektura sieci

Bazując na literaturze omówionej w poprzednim rozdziale w niniejszej pracy wykorzystana zostanie jednokierunkowa sieć typu sigmoidalnego o jednej warstwie ukrytej. Jest to jedna z najprostszych możliwych struktur, w której występują trzy warstwy neuronów wejściowa, ukryta i wyjściowa. Liczba neuronów w warstwie wejściowej i wyjściowej ściśle zależy od wymiarowości danych, natomiast największy wpływ na efektywność działania sieci ma właściwy dobór liczby neuronów w warstwie ukrytej. Poglądowy schemat jednokierunkowej sieci neuronowej przedstawiony został na rysunku 3.3. Na schemacie pokazano dwie warstwy ukryte dla lepszego zrozumienia połączeń między kolejnymi warstwami. W wyjściowej strukturze pojedynczy neuron danej warstwy połączony jest ze wszystkimi neuronami kolejnej warstwy. Liczba połączeń może być później redukowana z wykorzystywaniem algorytmu OBD, który opisany zostanie w dalszej części.

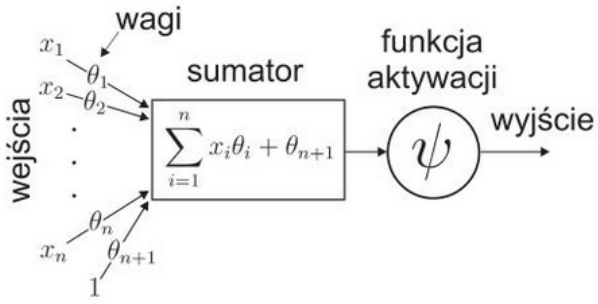


Rysunek 3.3. Schemat poglądowy struktury sieci. Źródło: [14]

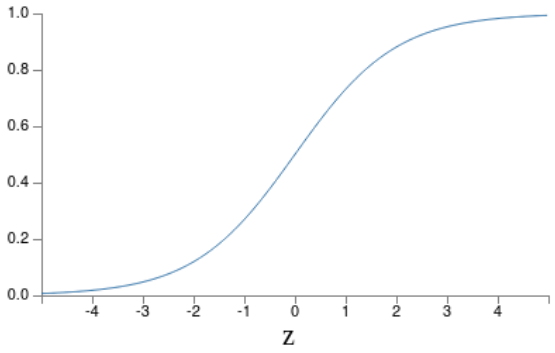
Kluczowym dla zrozumienia działania całej sieci neuronowej jest przekształcenie zachodzące w obrębie pojedynczego neuronu. Model neuronu przedstawiony został na rysunku 3.4. Wejścia do danego neuronu zostają przemnożone przez odpowiednie wagi, a następnie zsumowane. Następnie obliczoną sumę wykorzystujemy jako argument funkcji aktywacji przez co otrzymujemy wyjście z danego neuronu. Jedną z najczęściej stosowanych funkcji aktywacji jest funkcja sigmoidalna unipolarna o postaci

$$f(z) = \frac{1}{1+e^{-z}}, \tag{11}$$

której przebieg widzimy na rysunku 3.5. Warto zauważyć, że w obrębie dostatecznie małych i dużych wartości wejścia do funkcji sigmoidalnej następuje jej stabilizacja, a co za tym idzie tak zwana stabilizacja całego neuronu.



Rysunek 3.4. Model neuronu. Źródło: [15]



Rysunek 3.5. Przebieg funkcji sigmoidalnej unipolarnej. Źródło: [14]

3.2.2. Wsteczna propagacja gradientu

Uczenie sieci neuronowej możemy opisać w postaci następującego problemu. Mamy dany zbiór przykładów uczących postaci $< x, y^d >$, wektor wyjść sieci $\bar{f}(x;\theta)$ oraz pewną funkcje straty mierzącą rozbieżność między y^d a wektorem wyjść obliczonym przez sieć, która w naszym przypadku przyjmuje postać funkcji kwadratowej,

$$q(\bar{f}(x;\theta)) = \left\| \bar{f}(x;\theta) - y^d \right\|^2. \tag{12}$$

Zadaniem algorytmu uczenia jest tak dobrać wartości parametrów θ czyli kolejnych wag sieci, aby osiągnąć minimum funkcji straty co z kolei opiera się na wyznaczeniu gradientu funkcji kosztu po danych parametrach, czyli wektora

$$\frac{dq(\bar{f}(x;\theta))}{d\theta^T}. \tag{13}$$

W praktyce wyprowadzenie konkretnych formuł na gradient funkcji za pomocą funkcji wejścia i wag może być zbyt skomplikowane. Aby uniknąć tej niedogodności stosuje się metodę wstecznej propagacji. Celem tej metody jest obliczenie pochodnych cząstkowych $\frac{\partial q}{\partial w^l_{jk}}$ oraz $\frac{\partial q}{\partial b^l_j}$, gdzie w^l_{jk} oznaczą wagę łączącą k-ty neuron warstwy l-1 z j-tym neuronem warstwy l, natomiast b^l_j oznacza wyraz wolny j-tego neuronu warstwy l. Dla łatwiejszej prezentacji algorytmu wprowadzamy wartość $\delta^l_j = \frac{\partial q}{\partial z^l_j}$, którą możemy interpretować jako błąd j-tego neuronu l-tej warstwy czyli pochodną cząstkową po wartości zwracanej przez sumator z rysunku 3.4.

Teraz możemy przedstawić algorytm wstecznej propagacji za pomocą czterech równań. Pierwszym z nich jest wyrażenie na wartości δ^l_j dla warstwy wyjściowej L o postaci

$$\delta^L_j = \frac{\partial q}{\partial a^L_j} f'(z^L_j), \tag{14}$$

które dla kwadratowej funkcji kosztu w wersji macierzowej zapisujemy jako

$$\delta^L_j = (a^L - y^d) \odot f'(z^L_j), \tag{15}$$

gdzie symbol $\odot$ oznacza iloczyn Hadamarda, f jest zdefiniowaną wcześniej funkcją aktywacji, natomiast $a^L_j = f(z^l_j)$. Drugim równaniem jest wyrażenie błędów neuronów l-tej warstwy poprzez wartości błędów l+1-tej warstwy. Zapis macierzowy wygląda następująco

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot f'(z^l). \tag{16}$$

Za pomocą równań (15) i (16) możemy wyznaczyć wektory δ^l_j dla każdej z warstw. Dwa kolejne równania opisują powiązanie kolejnych pochodnych cząstkowych funkcji kosztu z wyznaczonymi za pomocą dwóch pierwszych równań wektorami. Kolejno dla wyrazów wolnych

$$\frac{\partial q}{\partial b^l_j} = \delta^l_j, \tag{17}$$

oraz wag przypisanych do poszczególnych połączeń

$$\frac{\partial q}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{18}$$

Równanie 15 - 18 stanowią bezpośrednią podstawę do późniejszej implementacji. Wyprowadzenie tych równań zostało pominięte w pracy, z uwagi że algorytm wstecznej propagacji nie jest głównym aspektem niniejszej pracy, jednak uważny czytelnik może znaleźć pełny opis w pozycjach [2] oraz [14].

3.2.3. Algorytm OBD

Dzięki opisanej wcześniej metodzie uczenia sieci neuronowej mamy możliwość minimalizacji błędu popełnianego przez sieć na danych uczących i weryfikujących. Niestety nie daje to pewności, że ostateczny dobór parametrów jest optymalny ze względu na wrażliwość trenowanej sieci na początkowy dobór parametrów uczenia oraz poszczególnych wartości wag. Zgodnie z [2] rozwiązaniem może być tutaj zastosowanie metod redukcji sieci, a za jedną z lepszych możemy uznać algorytm OBD (Optimal Brain Damage) zaproponowany przez LeCun [5]. Wskazany algorytm należy do klasy wrażliwościowych metod redukcji sieci i jego podstawowym zadaniem jest zmniejszenie liczby powiązań między-neuronowych a co za tym idzie uproszczenie struktury sieci.

Idea algorytmu OBD polega na przypisaniu każdej z wag miary, która określi wrażliwość sieci na zmianę tej konkretnej wagi. Następnie wagi o najmniejszej wrażliwości mogą zostać usunięte co nie powinno wpłynąć w istotny sposób na działanie sieci, a za to pozwolić na uproszczenie jej struktury. Twórca algorytmu za miarę ważności danej wagi proponuje współczynnik asymetrii (ang. saliency), który zdefiniowany zostaje w następujący sposób

$$S_i = \frac{1}{2}\frac{\partial^2 q}{\partial \theta_i^2} * \theta_i^2. \tag{19}$$

Podstawą do zdefiniowania miary było rozwinięcie funkcji celu q w szereg Taylora i na tej podstawie znalezienie parametrów, których usunięcie będzie miało najmniejszy wpływ na q. Wyznaczenie pełnej macierzy Hessego H w większości przypadków staje się niemożliwie ze względu na złożoność obliczeniową. Z tego powodu LeCun wykorzystał fakt, że wobec dodatniej określoności hesjanu macierz H jest diagonalnie dominująca. Zasadnym jest zatem uwzględnienie jedynie składników diagonalnych h_{ii}, które to możemy wyznaczyć dostosowując metodę wstecznej propagacji poprzednio omówioną dla pierwszych pochodnych.

Podtrzymując założenie o kwadratowej funkcji kosztu dla każdego przykładu uczącego wyliczamy miarę

$$h_{ii} = \frac{\partial^2 q}{\partial (w_{jk}^l)^2}, \tag{20}$$

gdzie w_{jk}^l jest połączeniem pomiędzy neuronami zdefiniowanym przy prezentacji algorytmu wstecznej propagacji. Formułę możemy rozwinąć do postaci

$$\frac{\partial^2 q}{\partial (w_{jk}^l)^2} = \frac{\partial^2 q}{\partial (z_j^l)^2}(a_k^{l-1})^2, \tag{21}$$

i zastosować wsteczną propagację w postaci

$$\frac{\partial^2 q}{\partial (z_k^l)^2} = f'(z_k^l)^2 \sum_j (w_{jk}^{l+1})^2 \frac{\partial^2 q}{\partial (z_j^{l+1})^2} + f''(z_k^l)\frac{\partial q}{\partial a_k^l}. \tag{22}$$

Dodatkowo poprzez zastosowanie aproksymacji Levenberga-Marquardta możemy uprościć wzór do postaci

$$\frac{\partial^2 q}{\partial (z_k^l)^2} = f'(z_k^l)^2 \sum_j (w_{jk}^{l+1})^2 \frac{\partial^2 q}{\partial (z_j^{l+1})^2} \tag{23}$$

co dodatkowo daje gwarancję, że wyznaczane aproksymacje drugich pochodnych nie będą ujemne. Przed zastosowaniem powyższego wzoru niezbędne jest wyznaczenie wartości początkowej czyli drugich pochodnych dla ostatniej warstwy sieci neuronowej. Ponownie stosując wspomnianą wcześniej aproksymację otrzymujemy wzór postaci

$$\frac{\partial^2 q}{\partial (z_k^L)^2} = 2 f'(z_k^L)^2. \tag{24}$$

Po wyznaczeniu wartości diagonalnych h_{ii} należy uśrednić je po wszystkich przykładach uczących i na tej podstawie według wzoru (19) wyznaczyć współczynniki asymetrii, zgodnie z którymi przycinane będą kolejne wagi sieci. Ogólną procedurę algorytmu OBD zapisać możemy w następujących krokach:

1. Wybór właściwej architektury sieci
2. Pełne wytrenowanie danej struktury przy użyciu jednej z metod uczenia, na przykład wstecznej propagacji
3. Obliczenie wartości diagonalnych macierzy Hessego czyli współczynników h_{ii}
4. Wyznaczenie współczynników asymetrii S_i
5. Posortowanie parametrów rosnąco według współczynników S_i
6. Usunięcie najmniej znaczących wag
7. Powtórzenie procedury od punktu 2, aż do osiągnięcia pożądanego kryterium wyjścia na danych weryfikujących

Warto zauważyć, że usunięcie wagi odbywa się poprzez ustawienie jej wartości na zero i przypisanie odpowiedniej maski, która uniemożliwia metodzie wstecznej propagacji ponowną modyfikację parametru.

3.3. Strategia eksperymentów

Opisane założenia teoretyczne stanowią podstawę do implementacji rozwiązania i przeprowadzenia eksperymentów praktycznych, których wyniki przedstawione zostaną w kolejnym rozdziale. Przed przystąpieniem do prezentacji wyników warto jednak jeszcze opisać schemat według, którego zostaną one przeprowadzone. Pozwoli to czytelnikowi na ewentualną samodzielną interpretację wyników i będzie prowadziło do lepszego zrozumienia wniosków płynących z pracy. Poniższy schemat stanowi jedynie zwięzły opis strategi, a wszystkie szczegóły wskazane zostaną we właściwym rozdziale.

Pierwszym etap jest wygenerowanie danych, które następnie stanowić będą przykłady uczące i weryfikujące zarówno dla algorytmu wstecznej propagacji jak i OBD. Generacja danych polega na przeprowadzeniu symulacji z wykorzystaniem regulatora DMC i zdefi-

niowanego obiektu regulacji. Warto zwrócić uwagę na fakt, że dane uczące i weryfikujące generowane będą oddzielnie, dzięki czemu działanie sieci neuronowej weryfikowane będzie z wykorzystaniem przykładów symulacji niewystępujących w danych uczących. Jednak przed zastosowaniem otrzymanych danych w procedurze uczenia sieci niezbędne jest ich przeskalowanie do wartości z zakresu od -1 do 1. Jest to niezbędny krok wymuszony przez rodzaj zastosowanej funkcji aktywacji. Wartym zauważenia jest fakt, że koniecznym było zastosowanie modułu skalującego pozwalającego na transformację danych wejściowych w dwie strony. Sieć neuronowa wykorzystana zostanie jako regulator, a zatem procedura odwrotnego skalowania jest tutaj niezbędna.

Następnie należy wytrenować sieć neuronową z wykorzystaniem opisanej metody wstecznej propagacji. Z uwagi na fakt, że kolejnym krokiem będzie wykorzystanie algorytmu OBD do redukcji sieci, należy zastosować lekko zmodyfikowaną strategię względem klasycznego podejścia do uczenia sieci neuronowej. Algorytm OBD wymaga aby funkcja celu była w swoim minimum, a zatem konieczne jest pełne wytrenowanie sieci na danych uczących bez stosowania kryterium wyjścia na danych weryfikujących. Algorytm uczenia sieci zakończy swoje działanie gdy zmiany funkcji kosztu w kolejnych iteracjach będą dostatecznie małe. Eksperymenty powtórzone zostaną dla różnej ilości neuronów w warstwie ukrytej oraz różnych wartości współczynnika uczącego sieci. Pozwoli to na wybór najbardziej optymalnej architektury przed przystąpieniem do procedury redukcji sieci.

Kolejno zastosujemy procedurę OBD dla optymalnej struktury sieci. Redukcja sieci zgodnie z opisaną wcześniej procedurą będzie kontynuowana aż zmiana funkcji kosztu na danych weryfikujących osiągnie dostatecznie mała wartość. Dzięki takiemu podejściu otrzymamy sieć neuronową, która powinna posiadać wysokie zdolności generalizacji postawionego przed nią zadania. W celu weryfikacji otrzymanego rezultatu przeprowadzona zostanie symulacja danego układu regulacji dla różnych wartości zadanych z wykorzystaniem sieci neuronowej oraz regulatora DMC. W obu przypadkach wyznaczony zostanie błąd średnio-kwadratowy i na tej podstawie możliwe będzie porównanie dwóch zastosowanych metod i weryfikacja czy sieć neuronowa jest w stanie zastąpić klasyczny regulator.

Ostatnim etapem prowadzonych badań będzie przetestowanie działania sieci neuronowej jako regulatora zmodyfikowanych obiektów regulacji. Wykorzystana zostanie tutaj wcześniej opisana własność łatwej zmiany obiektu regulacji poprzez dobór odpowiednich parametrów. Przeanalizowane zostanie również działanie sieci dla bardziej skomplikowanych przebiegów regulacji, przykładowo wielokrotnego skoku wartości zadanej.

W rozdziale przedstawione zostały wszystkie niezbędne opisy i założenia teoretyczne pozwalające na pełne zrozumienie omawianego zagadnienia. Zaprezentowane wzory i struktury stanowią bezpośrednią podstawę do praktycznej implementacji rozwiązanie,

której najważniejsze części przedstawione zostaną w kolejnym rozdziale. Dodatkowo lektura tego rozdziału powinna umożliwić czytelnikowi samodzielną interpretację wyników i dokładniejsze zrozumienie charakteru pracy.

4. Opis wybranych elementów implementacji

Rozdział poświęcony będzie najważniejszym elementom implementacji, przedstawione w nim zostaną kody źródłowe struktur, których opis teoretyczny czytelnik mógł znaleźć w poprzednim rozdziale. Przedstawienie wszystkich składników rozwiązania można uznać za niecelowe i zbędnie wydłużające pracę dlatego uwaga poświęcona zostanie jedynie krytycznym elementom z punktu widzenia zamierzonej funkcjonalności i realizacji postawionego zadania.

Prezentacje można podzielić na trzy oddzielne fragmenty, w pierwszej pokazana zostanie implementacja algorytmu DMC wraz z używanym w trakcie eksperymentów obiektem regulacji. Następnie czytelnik pozna najważniejsze elementy dotyczące implementacji sieci neuronowej ze szczególnym uwzględnieniem algorytmu OBD, któremu poświęcony jest ostatni podrozdział. Całość rozwiązania zaimplementowana została z wykorzystaniem języka Python 3.6 i korzysta z kilku podstawowych zewnętrznych bibliotek. Za kluczową ze względu na charakter pracy należy uznać bibliotekę NumPy pozwalająca na obsługę wielowymiarowych tablic i macierzy, a także udostępniającą zbiór funkcji matematycznych wysokiego poziomu do obsługi tych tablic. W trakcie implementacji nieocenione okazały się opracowania [14] oraz [15].

4.1. Algorytm DMC z obiektem regulacji

Algorytm regulacji predykcyjnej zaimplementowany został jako klasa DMC, której strukturę widzimy w poniższym kodzie:

Listing 1. Klasa DMC

```
1  class DMC(object):
2   def __init__(self, n_u, n, S_list, psi_const, lambda_const):
3    #horyzont sterwania
4    self.n_u = n_u
5    #horyzont predykcji
6    self.n = n
7    #horyzont dynamiki obiektu - ile s wyzaczamy (10, 20, 30)
8    self.d = len(S_list)
9
10   self.Y_k = np.zeros(n)
11   #wektor wartosci zadanej - stala na calym horyzoncie N
12   self.Y_zad = np.zeros(n)
13   #wektor opisujacy trajektorie sygnalu wyjsciowego
14   self.Y_hat = np.zeros(n)
15
16   #wektor wyznaczanych przyrostow wartosci sterowania
```

```
17      self.U_delta = np.zeros(n_u)
18
19      #wektor przeszlych przyrostow sterowania
20      self.U_delta_P = np.zeros(len(S_list)-1)
21
22      #macierze wyznaczane w trakcie inicjalizacji klasy
23      self.Psi = np.diag([psi_const for i in range(n)])
24      self.Lambda = np.diag([lambda_const for i in range(n_u)])
25      self.M_p = self.init_M_p(S_list, n)
26      self.M = self.init_M(S_list, n, n_u)
27      self.K = inv(self.M.T @ self.Psi @ self.M + self.Lambda) \
28                                              @ self.M.T @ self.Psi
```

Klasa inicjalizowana jest zgodnie z opisem teoretycznym w sekcji 3.1.1. z wykorzystaniem wektora odpowiedzi skokowej układu reprezentowanego przez argument *S_list*. Warto zwrócić tutaj uwagę na linię 27-28 Listingu 1, która stanowi bezpośrednią implementację macierzy K przedstawionej za pomocą równania (9).

W dalszej części występują funkcje pomocnicze wykorzystywane do tworzenia macierzy M oraz M^P, które ze względu na ich trywialność zostaną pominięte. Kluczowym elementem całej klasy ze względu na działanie algorytmu jest funkcja wyznaczająca wektor optymalnych przyrostów sterowania w każdej iteracji algorytmu. Implementację funkcji prezentuje Listing 2, gdzie w liniach 4-5 zaimplementowano równanie (8). W każdej iteracji algorytmu zwracaną jest tylko wartość $\Delta u(k|k)$, której odpowiada zwracana przez funkcję zmienna *delta_u*.

Listing 2. Funcja wyznaczająca wektor przyrostów sterowania

```
1  def calculate_U_delta(self, Y_current):
2
3          self.Y_k.fill(Y_current)
4          self.U_delta = self.K @ (self.Y_zad - self.Y_k - \
5                  (self.M_p @ self.U_delta_P))
6          delta_u = self.U_delta[0]
7          self.update_U_delta_P(delta_u)
8
9          return delta_u
```

Obiekt regulacji zaimplementowany został w postaci klasy *SimObject* gdzie istotę działania obiektu prezentuje funkcja zwracająca wyjście obiektu na podstawie wartości sterowania, w każdej kolejnej iteracji. Listing 3 prezentuje funkcję *make_simulation_step*, w której należy zwrócić uwagę na linie 8-9, w których widzimy zaimplementowane równanie (10).

Listing 3. Funcja wyznaczająca wektor przyrostów sterowania

```
1  def make_simulation_step(self, u, y_zad):
2
3   u_lag1=self.get_lag_u(self.TD+1)
4   u_lag2=self.get_lag_u(self.TD+2)
5   y_lag1=self.get_lag_y(1)
6   y_lag2=self.get_lag_y(2)
7
8   y_current=self.b_1*u_lag1 + self.b_2*u_lag2 - \
9    self.a_1*y_lag1 - self.a_2*y_lag2
10  self.y_list = np.append(self.y_list, y_current)
11  self.u_list = np.append(self.u_list, u)
12  self.y_zad_list = np.append(self.y_zad_list, y_zad)
13  return y_current
```

4.2. Architektura sieci neuronowej

Implementacja sieci neuronowej przedstawiona została w postaci klasy *Network*, której konstruktor widzimy w Listingu 4. Struktura sieci składa się z listy macierzy *weights* (linia 7) oraz *biases* (linia 6) odpowiadających kolejnym wagom oraz wyrazom wolnym dla połączeń między neuronami sąsiadujących warstw. Natomiast lista macierzy *saliencies* o tej samej wymiarowości co *weights* zawierają współczynniki asymetrii wyznaczane w trakcie działania algorytmu OBD. Zmienna *mask* wykorzystywana jest przez algorytm OBD do zamrożenia danych wag w trakcie ponownego uczenia sieci. Na końcu inicjalizacji ustalany jest parametr *cost_delta_epsilon* stanowiący graniczną wartość gradientu funkcji celu poniżej, której proces uczenia sieci zostaje zakończony.

Listing 4. Klasa Network

```
1  class Network(object):
2   def __init__(self, sizes):
3
4    self.num_layers = len(sizes)
5    self.sizes = sizes
6    self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
7    self.weights = [np.random.randn(y, x)
8                    for x, y in zip(sizes[:-1], sizes[1:])]
9    self.saliencies = [np.zeros((y, x))
10                   for x, y in zip(sizes[:-1], sizes[1:])]
11   self.mask = [np.ones((y, x))
12                   for x, y in zip(sizes[:-1], sizes[1:])]
```

```
13
14      self.cost_delta_epsilon = 0.000005
```

Główną część implementacji sieci neuronowej stanowią funkcje wykorzystywane w trakcie procesu trenowania sieci. Jest to odpowiednio funkcja *SGD* implementująca metodę stochastycznego najszybszego spadku (SGD), jedną z najczęściej stosowanych metod iteracyjnej optymalizacji funkcji celu. Metoda to korzysta z funkcji *update_mini_batch*, która to odpowiada za zastosowanie metody wstecznej propagacji dla każdego z przykładów uczących występujących w pojedynczej próbce wyznaczanej przez SGD. Kluczowym dla działania wymienionych funkcji jest implementacja metody wstecznej propagacji dla pojedynczego przykładu uczącego. Teoretyczne założenia używanej metody przedstawione zostały w sekcji 3.2.2, a jej implementacje widzimy na Listingu 5. Poprzednie funkcje nie zostały tutaj zaprezentowane gdyż nie prezentują głównej idei wstecznej propagacji, a jedynie z niej korzystają, co mogłoby niepotrzebnie wydłużyć całość wywodu. W początkowej części (linie 2-3) tworzone są wektory odpowiadające pochodnych cząstkowych $\frac{\partial q}{\partial w^l_{jk}}$ oraz $\frac{\partial q}{\partial b^l_j}$. Następnie (linie 5-12) następuje jednokrotne przejście sieci w przód (*ang. feedforward*) czyli wyliczenie wyjścia sieci na podstawie wektora wejściowego x. Główna zasada działania wstecznej propagacji znajduje odzwierciedlenie w kolejnych liniach kodu. Należy zwrócić tutaj szczególną uwagę na linie 15-16, gdzie wykorzystane zostało równanie (15) stanowiące warunek początkowy dla algorytmu. Następnie w pętli następuje wsteczna propagacja między warstwami opisania równaniem (16), które znajduje bezpośrednie odzwierciedlenie w linijce 23.

Listing 5. Funkcja wstecznej propagacji

```
1  def backprop(self, x, y):
2   nabla_b = [np.zeros(b.shape) for b in self.biases]
3   nabla_w = [np.zeros(w.shape) for w in self.weights]
4   # jednokrotne wyznaczenie wyjscia sieci dla wjescia x
5   activation = x
6   activations = [x]
7   zs = []
8   for b, w in zip(self.biases, self.weights):
9     z = np.dot(w, activation)+b
10    zs.append(z)
11    activation = sigmoid(z)
12    activations.append(activation)
13  # wsteczna propagacja
14  # warunek poczatkowy
15  delta = self.cost_derivative(activations[-1], y) * \
16            sigmoid_prime(zs[-1])
```

```
17    nabla_b[-1] = delta
18    nabla_w[-1] = np.dot(delta, activations[-2].transpose())
19    # iteracja od konca po warstwach
20    for l in range(2, self.num_layers):
21      z = zs[-l]
22      sp = sigmoid_prime(z)
23      delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
24      nabla_b[-l] = delta
25      nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
26    return (nabla_b, nabla_w)
```

Pozostała część implementacji obejmuje funkcje aktywacji, funkcje wyliczająca wyjście sieci na podstawie zadanego wektora wejściowego, a także wszelkie niezbędne funkcje pomocnicze. Sposób ich implementacji nie jest kluczowy dla zrozumienia całości pracy, a może być w łatwy sposób odtworzony na podstawie opisu teoretycznego. Z punktu widzenia czytelnika warto jednak przyjrzeć się ostatniej części klasy *Network*, która zawiera implementację algorytmu redukcji sieci OBD.

4.3. Algorytm OBD

Algorytm OBD zaimplementowany został jako część klasy *Network* na podstawie opisu teoretycznego zawartego w sekcji 3.2.3. i składa się z trzech głównych funkcji *OBD*, *backpropOBD* oraz *cut_weights*. Pierwsza z nich przedstawiona została na Listingu 6 i odpowiada za implementację procedury algorytmu OBD przedstawionej w opisie teoretycznym. Kluczowa dla działania algorytmu jest pętla rozpoczynająca się w linii 8, która kolejno wywołuje metodę wstecznej propagacji dla drugich pochodnych. Po iteracji dla wszystkich przykładów uczących obliczana jest wartość współczynników asymetrii i następnie w linii 16 wywoływana jest funkcja *cut_weights* odpowiedzialna za przycinanie sieci. Po redukcji sieć jest ponownie uczona z wykorzystaniem metody *SGD*. Procedura zostaje powtórzona aż do osiągnięcia kryterium wyjścia na danych testujących.

Listing 6. Algorytm OBD

```
1  def OBD(self, train_data, test_data):
2    nabla_h = [np.zeros(w.shape) for w in self.weights]
3    par_number = sum([w.shape[0]*w.shape[1] for w in self.weights])
4    test_cost = []
5    prev_cost = np.inf
6    prev_delta_cost_list = np.full((3,),np.inf)
7
8    for limit in range(int(0.9*par_number)):
9      for x, y in train_data:
```

```
10        delta_nabla_h = self.backpropOBD(x,y)
11        nabla_h = [nh + dnh
12          for nh, dnh in zip(nabla_h, delta_nabla_h)]
13
14      self.saliencies = [(h * w**2)/(2 * len(train_data))
15                                  for w, h in zip(self.weights, nabla_h)]
16      self.cut_weights(limit+1)
17      self.SGD(train_data, 200, 10, 3.0)
18      test_cost.append(self.total_cost(test_data))
19      current_cost = self.total_cost(test_data)
20      prev_delta_cost_list = np.delete(np.insert(
21        prev_delta_cost_list, 0, prev_cost - current_cost),-1)
22      prev_cost = current_cost
23      #stopping rule
24      if all(prev_delta_cost_list < self.cost_delta_epsilon/100):
25        return train_cost, test_cost
26
27    return train_cost, test_cost
```

Niezbędnym dla zrozumienia całego algorytmu OBD jest proces wstecznej propagacji zastosowany dla drugich pochodnych. Jego implementacja zaprezentowana została na Listingu 7 i zawiera wiele podobieństw do wcześniej omawianej funkcji *backprop*. Należy zwrócić tutaj szczególną uwagę na linie 23-24, które odpowiadają równaniu (23). W linii 15 widzimy natomiast wykorzystanie funkcji pomocniczej do wyliczenia warunku granicznego z równania (24), zabieg ten sprzyja czytelności kodu, a z uwagi na mało skomplikowaną formę równania nie powinien stanowić utrudnienia w zrozumieniu całej metody.

Listing 7. Funkcja wstecznej propagacji drugich pochodnych

```
1  def backpropOBD(self, x, y):
2
3    h_vector = [np.zeros(w.shape) for w in self.weights]
4    # feedforward
5    activation = x
6    activations = [x]
7    zs = []
8    for b, w in zip(self.biases, self.weights):
9      z = np.dot(w, activation)+b
10     zs.append(z)
11     activation = sigmoid(z)
```

```
12      activations.append(activation)
13
14    #wsteczna propagacja drugich pochodnych
15    delta2_z = self.boundry_OBD_derivative(zs[-1], y)
16    h_vector[-1] = np.dot(delta2_z,
17      activations[-2].transpose()**2)
18    #iteracja po kolejnych warstwach
19    for l in range(2, self.num_layers):
20      z = zs[-l]
21      sp = sigmoid_prime(z)
22      sp2 = sigmoid_second_prime(z)
23      delta2_z =  sp**2 * np.dot(self.weights[-l+1].transpose()**2,
24        delta2_z)
25      h_vector[-l] = np.dot(delta2_z,
26        activations[-l-1].transpose()**2)
27
28    return h_vector
```

Ostatnia z wykorzystywanych funkcji w procedurze OBD jest funkcja *cut_weights*. Odpowiada ona za przycięcie odpowiedniej ilości wag, ustalanej przez parametr *limit* według posortowanych wartości współczynników skośności *ang. saliencies*. W liniach 2-8 współczynniki są sortowane i w zmiennej *to_cut* zapisywane są indeksy odpowiednich wag, które to następnie zostają przycięte i zamrożone z wykorzystaniem opisywanej wcześniej maski.

Listing 8. Funkcja redukcji wag

```
1  def cut_weights(self, limit):
2   saliencies_list = []
3    for i_layer , saliencies in enumerate(self.saliencies):
4     for i_row, row in enumerate(saliencies.tolist()):
5      for i_column, value in enumerate(row):
6       saliencies_list.append(( [i_layer, i_row, i_column], value))
7   saliencies_list.sort(key = lambda x: x[1])
8   to_cut = [element[0] for element in saliencies_list[:limit]]
9
10  self.restore_mask()
11  for wt_index in to_cut:
12   self.weights[wt_index[0]][wt_index[1]][wt_index[2]] = 0.0
13   self.mask[wt_index[0]][wt_index[1]][wt_index[2]] = 0.0
```

Lektura powyższego rozdziału stanowiąca uzupełnienie do opisu teoretycznego powinna dać czytelnikowi pełny obraz wykorzystywanych w trakcie eksperymentów struktur. Po zapoznaniu się z elementami implementacji można przejść do rozdziału opisującego wyniki przeprowadzonych eksperymentów. Warto zauważyć, że zrozumienie wszystkich przedstawionych w tym rozdziale szczegółów nie jest warunkiem koniecznym do wyciągnięcia ogólnych wniosków z kolejnego rozdziału lecz na pewno stanowi dużą wartość dodaną dla czytelnika niezaznajomionego z prezentowaną w tej pracy tematyką.

5. Wyniki pracy

Niniejszy rozdział prezentuje główne wyniki uzyskane w trakcie badania i zawiera w sobie podsumowanie przeprowadzonych eksperymentów. Omówione zostaną w nim wszystkie uzyskane rezultaty, które pozwalają na ocenę czy sztuczne sieci neuronowe mogą być stosowane z powodzeniem w obszarze regulacji. Na początku zaprezentowany zostanie sposób generacji danych, poczynione założenia i sposób działania regulacji opartej o sieć neuronową. Kolejno wybrana zostanie optymalna liczba neuronów warstwy ukrytej pozwalająca na minimalizację funkcji celu z zachowaniem ogólności rozwiązania. W kolejnej części zbadany zostanie wpływ zastosowania redukcji sieci na osiągane przez nią rezultaty. Po wybraniu optymalnej struktury i pełnym wytrenowaniu sieci zbadamy jak poradzi sobie z regulacją obiektów, które nie znalazły odzwierciedlenia w przykładach uczących. Ciekawym eksperymentem będzie też sprawdzenie działania sieci w przypadku wielokrotnych skoków wartości zadanej. Pod koniec rozdziału sformułowane zostaną ogólne wnioski i uwagi płynące z całości eksperymentów, które znajdą również swoje odzwierciedlenie w późniejszym podsumowaniu pracy.

5.1. Generacja danych

Niezbędnym krokiem przed przystąpieniem do trenowania sieci neuronowej jest generacja danych, na podstawie których sieć następnie zostanie nauczona. Celem pracy jest zweryfikowanie zdolności adaptacji sieci do działania jako regulator DMC. Kierując się tym założeniem oczywistym wydaje się wygenerowanie danych uczących na podstawie symulacji przeprowadzonych z wykorzystaniem rzeczywistego regulatora DMC. Praca ma jedynie charakter porównawczy, a więc za najogólniejszy przykład regulacji możemy wybrać dostosowanie wyjścia obiektu regulacji do jednokrotnego skoku wartości zadanej. Jako obiekt regulacji wybrany został w tej części układ opisany we wcześniejszej części pracy, który identyfikujemy poprzez następujące parametry: $T_1 = 5$, $T_2 = 2$, $K = 1$, $T_d = 0$. Wybór dokonany został w sposób arbitralny, gdyż przeprowadzenie eksperymentów z wykorzystaniem dowolnego innego układu pozwala osiągnąć zbliżone rezultaty i wyciągnąć analogiczne wnioski.

Na tym etapie należy podjąć decyzję odnośnie wartości na podstawie, których sieć neuronowa dokonywać będzie regulacji. Kierując się analogią do klasycznych metod za dobrą metodę wybrano sterowanie na podstawie wartości uchybu regulacji oraz aktualnej wartości zadanej. W trakcie przeprowadzanych eksperymentów długość symulacji wynosi 40 okresów jako okres, po którym układ regulowany za pomocą regulatora DMC osiąga pełną stabilność. Na tej podstawie do sterowania za pomocą sieci wybrane zostało 30 wartości uchyby regulacji oraz aktualna wartość zadana. Wyjściem sieci jest oczywiście pojedynczy sygnał sterujący co stanowi pewną modyfikację względem algorytmu DMC

gdzie, w każdej iteracji wyznaczana jest zmiana sterowania. Opisana modyfikacja stanowi jedynie szczegół implementacyjny, który nie wpływa na wyniki osiągane przez sieć.

Jak zostało już to opisane w części teoretycznej dane uczące i weryfikujące generowane są w sposób niezależny na podstawie oddzielnych przebiegów regulacji. Dzięki takiej strategi mamy pewność, że sieć zostanie przetestowana pod kątem ogólnej aproksymacji algorytmu OBD, a nie jedynie wybranych przebiegów regulacji. Tak więc kolejno przeprowadzone zostały eksperymenty polegające na wyznaczeniu przebiegu regulacji predykcyjnej dla jednostkowych skoków wartości zadanej z zakresu od 1 do 10 z krokiem co 0,1. Przebiegi regulacji podzielone zostały na dane uczące i weryfikujące w stosunku 75% - 25% . Po próbkowaniu dla każdej iteracji zbiór uczący ostatecznie składa się z 3015 przykładów, natomiast weryfikujący z 1035.

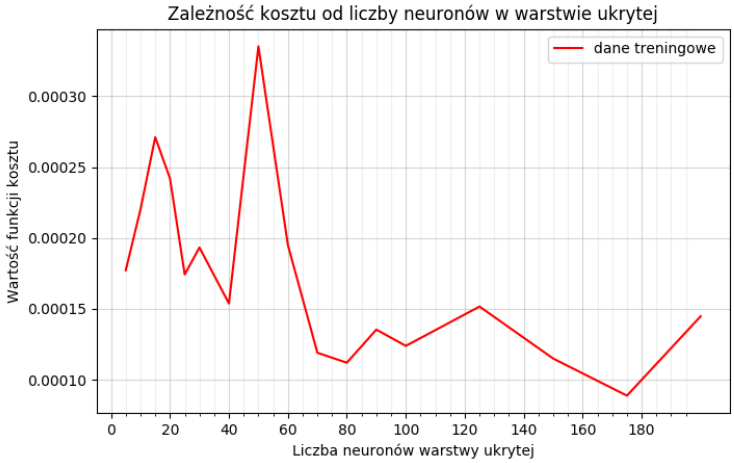
Ostatnim krokiem przed przystąpieniem do procedury uczenia sieci neuronowej i doboru optymalnej architektury. Jak już zostało wskazane z uwagi na sigmoidalną funkcję aktywacji koniecznym było napisanie modułu skalującego zarówno dane wejściowe jak i wyjściowe sieci do zakresu $(-1, 1$. Warto zauważyć, że skalowanie wartości wejściowych i wyjściowych przebiega niezależnie oraz moduł skalujący umożliwia odwrotne skalowanie wartości sterowania, które następnie wykorzystywana jest w trakcie regulacji. Stanowi to istotne ograniczenie w działaniu sieci, a mianowicie sieć jest w stanie regulować tylko układy dla których pożądane wartości sterowania zawierają się w zakresie reprezentowanym przez dane wykorzystane do nauki sieci.

5.2. Wybór struktury sieci

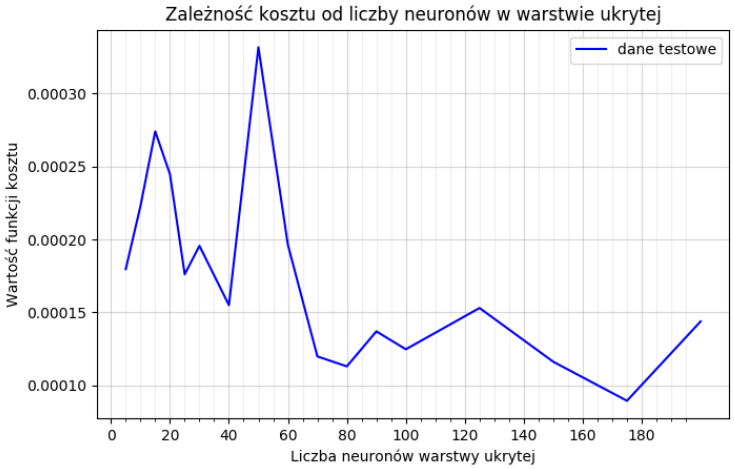
Pierwszym krokiem prowadzącym do wyselekcjonowania optymalnej architektury sieci jest wybór jej struktury. Liczba neuronów w warstwie wejściowej i wyjściowej ściśle zależy do charakterystyki problemu, a więc wygenerowanych uprzednio danych. Szczególną uwagę należy poświęcić za to prawidłowemu doborowi liczby neuronów w warstwie ukrytej. W tym celu przeprowadzono eksperyment pokazujący wartość kwadratowej funkcji kosztu w zależności od liczby neuronów w warstwie ukrytej. Przetestowano wartości z zakresu od 5 do 200 neuronów, a wyniki symulacji zaprezentowane zostały na Rysunku 5.1.

Na tym etapie trenowania sieci neuronowej głównym celem jest minimalizacja funkcji kosztu na danych uczących gdyż później stosowany algorytm OBD wymaga możliwie bliskiego osiągnięcia minimum tej funkcji. W ramach przedstawienie pełniejszego obrazu na Rysunku 5.2. przedstawiona została analogiczna symulacja z wykorzystaniem danych weryfikujących (testowych). Widzimy, że wyniki obu eksperymentów są ze sobą niezwykle spójne. Obserwacja ta potwierdza intuicję, gdyż zarówno zbiór danych uczących jak i weryfikujących wygenerowany został na podstawie analogicznych symulacji, w czasie których modyfikowana była jedynie wartość zadana.

Analiza wykresów dostarcza kilka istotnych obserwacji. Pierwszą z nich jest to, że nawet



Rysunek 5.1. Zależność kosztu od liczby neuronów w warstwie ukrytej - dane treningowe

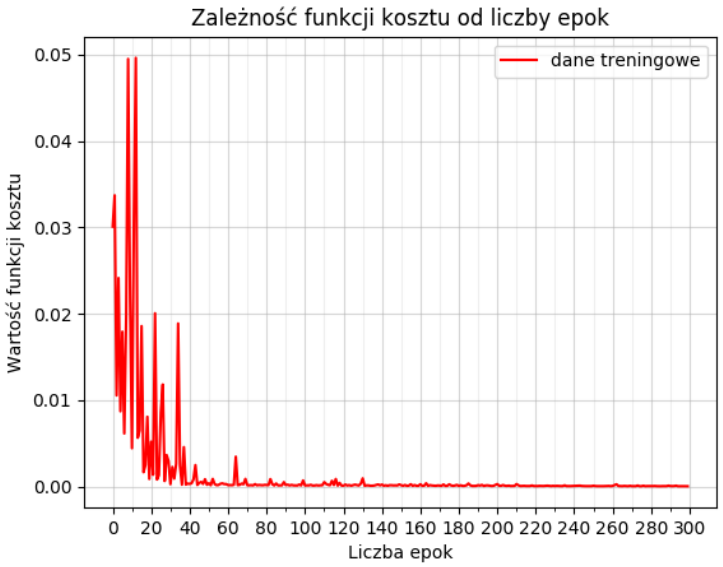


Rysunek 5.2. Zależność kosztu od liczby neuronów w warstwie ukrytej - dane testowe

niezwykle prosta strukturą jaką jest sieć neuronowa o jedynie 5 neuronach w warstwie ukrytej jest w stanie zadowalająco dobrze nauczyć się postawionego przed nią zadania regulacji. Warto jednak zwrócić uwagę, że sieci o małej liczbie neuronów wykazują się wysoką niestabilnością przez co rezultaty przez nie osiągane mogą różnić się pomiędzy kolejnymi próbami. Po kilkukrotnym powtórzeniu eksperymentu zaobserwowano wyeliminowanie problemu dla sieci neuronowych z liczbą neuronów w warstwie ukrytych przekraczającą 100. Na tej podstawie do dalszych eksperymentów wybrana została sieć o 150 neuronach.

Po wybraniu optymalnej struktury sieci należy zwrócić uwagę na zależność kosztu od liczby epok, przez które jest trenowana sieć. Pozwoli to na weryfikację wartości granicznej gradientu funkcji celu, która stanowi warunek wyjścia dla metody uczenia sieci. Początkowa kryterium wyjścia ustalone zostało na poziomie 10^{-5} co oznacza, że jeżeli gradient funkcji kosztu przez 3 kolejne iteracje jest mniejszy od zadanej wartości algorytm

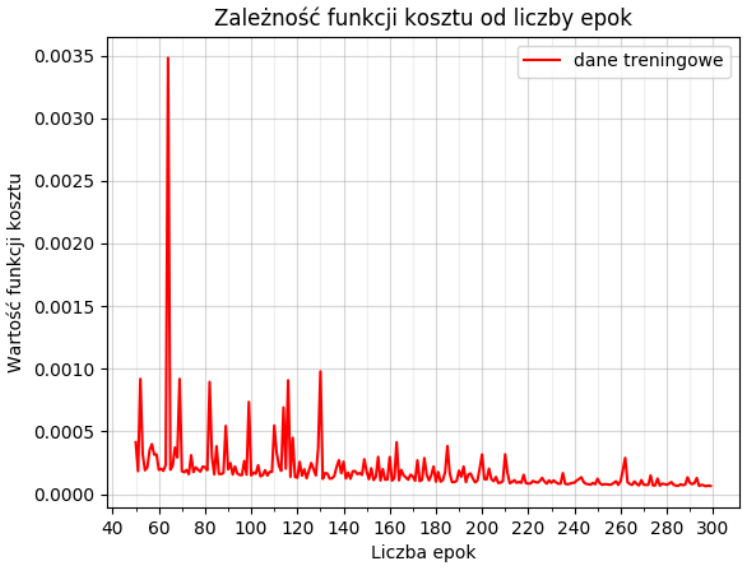
przerywa uczenie sieci. Eksperyment ten możemy przeprowadzić zdejmując wcześniejsze ograniczenie i sprawdzając jak zmienia się funkcja kosztu dla 300 iteracji algorytmu (epok). Wykres przedstawiający wspomnianą zależność zaprezentowany został na Rysunku 5.3 natomiast na Rysunku 5.4 przedstawione zostały te same dane ale z uciętymi 50 początkowymi wartościami w celu dokładniejszej prezentacji wyników czytelnikowi.



Rysunek 5.3. Zależność kosztu od liczby epok dla sieci z 150 neuronami

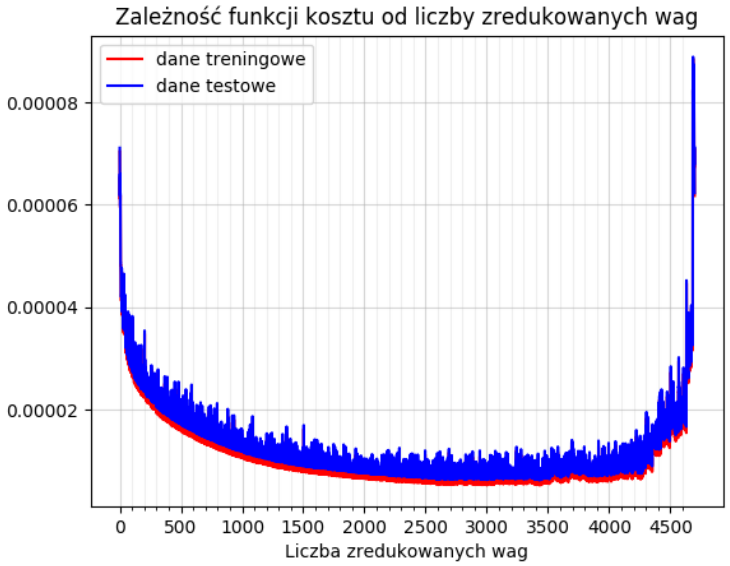
5.3. Zastosowanie algorytmu OBD

5.4. Zastosowanie innych obiektów regulacji



Rysunek 5.4. Zależność kosztu od liczby epok dla sieci z 150 neuronami (ucięte wartości początkowe)

6. Podsumowanie



Rysunek 5.5. Zależność kosztu od liczby zredukowanych wag - dane testowe i treningowe

Bibliografia

[1] S. O. Haykin, *Neural Networks: A Comprehensive Foundation, 2nd Edition*. Ontario Canada: Pearson Education, 1999.

[2] S. Osowski, *Sieci neuronowe do przetwarzania informacji*. Warszawa: Oficyna Wydawnicza Politechniki Warszawskiej, 2013.

[3] H. K., "Approximation capabilities of multilayer feedforward networks", *Neural Networks*, t. 4, nr. 2, s. 251–257, 1991.

[4] S. Osowski, "Sieci neuronowe do przetwarzania informacji", w. Oficyna Wydawnicza Politechniki Warszawskiej, 2013, s. 101–107.

[5] S. A. S. Yann Le Cun John S. Denker, "Optimal Brain Damage", w *Advances in Neural Information Processing Systems 2*, 1989.

[6] Ł. M. Chaber P., "Pruning of recurrent neural models: an optimal brain damage approach", *Nonlinear Dynamics*, t. 92, s. 763–780, 2018.

[7] M. M. P. Kavzoglu T., "Assessing Artificial Neural Network Pruning Algorithms", w *24th Annual Conference and Exhibition of the Remote Sensing Society*, 1998.

[8] W. G. Hassibi B. Stork D.G., "Optimal Brain Surgeon and general network pruning", w *IEEE International Conference on Neural Networks*, 1993.

[9] T. P. Kittisupakorn P., "Neural network based model predictive control for a steel pickling process", *Journal of Process Control*, t. 19, s. 579–590, 2009.

[10] M. F. Hosen M.A. Hussain M.A., "Control of polystyrene batch reactors using neural network based model predictive control (NNMPC): An experimental investigation", *Control Engineering Practice*, t. 19, s. 454–467, 2011.

[11] J.-S. F. Afram A., "Artificial Neural Network (ANN) based Model Predictive Control (MPC) and Optimization of HVAC Systems: A State of the Art Review and Case Study of a Residential HVAC System", *Energy and Buildings*, t. 141, 2017.

[12] M. Ławryńczuk, *Sterowanie procesów ciągłych, preskrypt*. Politechnika Warszawska, 2009.

[13] P. Tatjewski, *Sterowanie zaawansowane obiektów przemysłowych. Struktury i algorytmy*. Warszawa: Akademicka Oficyna Wydawnicza EXIT, 2016.

[14] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[15] P. Wawrzyński, *Podstawy sztucznej inteligencji*. Warszawa: Oficyna Wydawnicza Politechniki Warszawskiej, 2019.

[16] R. B. Culter C.R., "Dynamic matrix control – a computer control algorithm.", w *Proceedings of the AIChE National Meeting, Houston, USA*, 1979.

Wykaz symboli i skrótów

DMC – Dynamic Matrix Control
OBD – Optimal Brain Damage
SGD – Stochastic gradient descent
PW – Politechnika Warszawska

Spis rysunków

3.1. Przykładowa odpowiedź wyjścia obiektu na jednostkowy skok sterowania . . . 15
3.2. Regulacja DMC dla wybranego obiektu 17
3.3. Schemat poglądowy struktury sieci. Źródło: [14] 18
3.4. Model neuronu. Źródło: [15] 19
3.5. Przebieg funkcji sigmoidalnej unipolarnej. Źródło: [14] 19
5.1. Zależność kosztu od liczby neuronów w warstwie ukrytej - dane treningowe . 35
5.2. Zależność kosztu od liczby neuronów w warstwie ukrytej - dane testowe 35
5.3. Zależność kosztu od liczby epok dla sieci z 150 neuronami 36
5.4. Zależność kosztu od liczby epok dla sieci z 150 neuronami (ucięte wartości początkowe) 37
5.5. Zależność kosztu od liczby zredukowanych wag - dane testowe i treningowe . 38

Spis tabel

Spis załączników

1. Nazwa załącznika 1 41
2. Nazwa załącznika 2 43

Załącznik 1. Nazwa załącznika 1

Załącznik 2. Nazwa załącznika 2