

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.** TODO

**Słowa kluczowe:** Sieci neuronowe, regulacja predykcyjna, upraszczanie sieci neuronowej, algorytm OBD, algorytm DMC





.....  
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

<b>1. Wstęp</b>	9
<b>2. Przegląd literatury przedmiotu</b>	11
<b>3. Opis teoretyczny rozwiązania</b>	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	23
<b>4. Opis wybranych elementów implementacji</b>	25
4.1. Algorytm DMC z obiektem regulacji	25
4.2. Architektura sieci neuronowej	27
4.3. Algorytm OBD	29
<b>5. Wyniki pracy</b>	32
5.1. Generacja danych	32
5.2. Wybór struktury sieci	33
5.3. Zastosowanie algorytmu OBD	38
5.4. Zastosowanie innych obiektów regulacji	42
<b>6. Podsumowanie</b>	48
<b>Bibliografia</b>	51
<b>Wykaz symboli i skrótów</b>	52
<b>Spis rysunków</b>	52
<b>Spis tabel</b>	53
<b>Spis załączników</b>	53





# 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 teoretyczne 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 neuronowych. Następnie przeanalizuję dostępną literaturę z zakresu regulacji predykcyjnej i wskaże 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 możliwie 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 inną 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 neuronowej 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. Warty zauważyć 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 przeregulowania 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 wskazanych powyżej prac [9], [10] oraz [11] 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 dużą 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 identyfikację 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

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

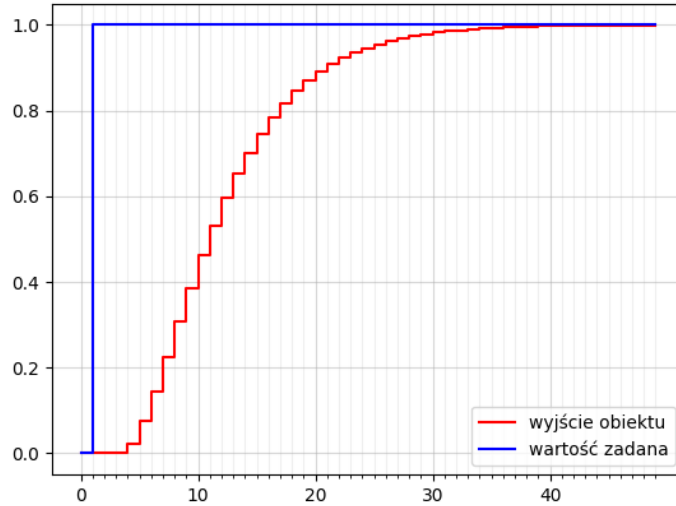
gdzie przez  $N_u$  oznaczamy horyzont sterowania, a notacja  $\Delta u(k+p|k)$  oznacza przyrost sygnału sterującego obliczony w chwili  $k$ , który ma być wykorzystany do sterowania w chwili  $k+p$ . W istocie jednak do sterowania wykorzystuje się tylko pierwszą wartość wyznaczanego wektora i prawo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

$$u(k) = u(k|k) = \Delta u(k|k) + u(k-1) \quad (2)$$

##### 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

odpowieź skokowa obiektu  $\{s_1, s_2, s_3, \dots, s_D\}$  wykorzystywana jest w algorytmie DMC do wyznaczenia 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 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

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

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

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

Macierze  $\Lambda$  oraz  $\Psi$  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

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

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

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

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

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

wektor  $\Delta U^P(k)$  zawiera przeszłe  $D-1$  wartości zamian sygnału sterującego  $\Delta u(k-i)$ , natomiast wektor  $Y(k)$  o długości  $N$  wypełniony jest wyjściem obiektu w chwili  $k$ .

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

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

gdzie początkową część możemy zapisać w formie

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

gdzie macierz  $K$  o wymiarowości  $N_u \times N$  wyznacza jest jednokrotnie w trakcie projektowania algorytmu. Równanie (8) stanowi podstawę 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

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



gdzie

$$a_1 = -\alpha_1 - \alpha_2$$

$$a_2 = \alpha_1 \alpha_2$$

$$\alpha_1 = e^{-\frac{1}{T_1}}$$

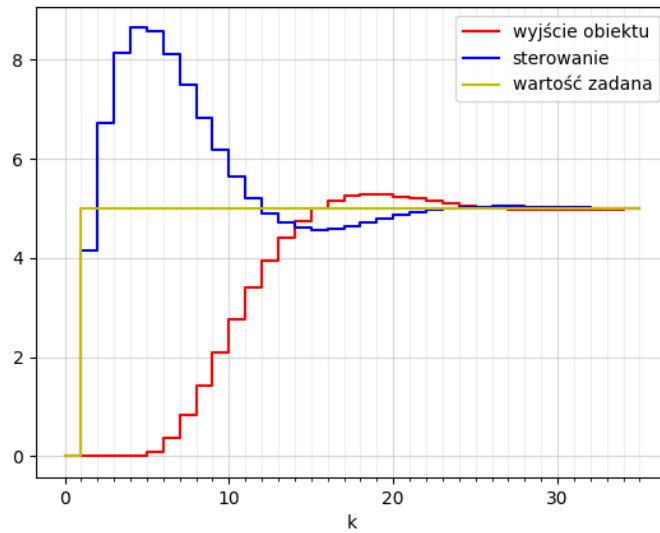
$$\alpha_2 = e^{-\frac{1}{T_2}}$$

$$b_1 = \frac{K}{T_1 - T_2} [T_1(1 - \alpha_1) - T_2(1 - \alpha_2)]$$

$$b_2 = \frac{K}{T_1 - T_2} [\alpha_1 T_2(1 - \alpha_2) - \alpha_2 T_1(1 - \alpha_1)]$$

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 przyrzeć 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

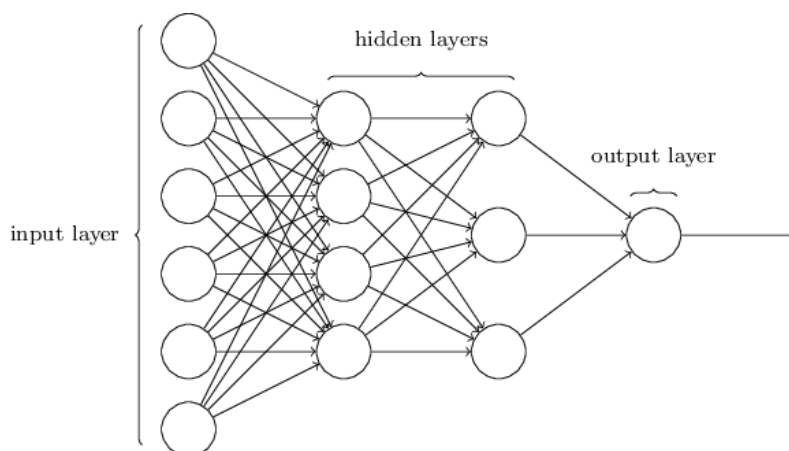
### 3.2. Sztuczne Sieci Neuronowe

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

interdyscyplinarnemu charakterowi, łączą 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 zredukowana z wykorzystywaniem algorytmu OBD, który opisany zostanie w dalszej części.

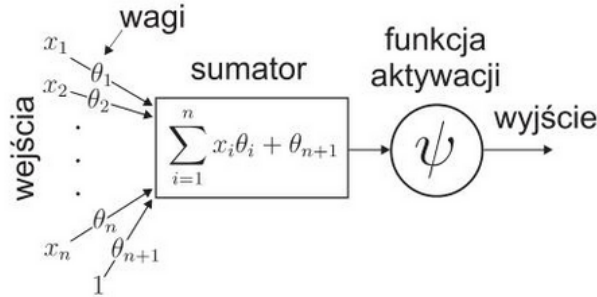


**Rysunek 3.3.** Schemat poglądowy struktury sieci. Źródło: [14]

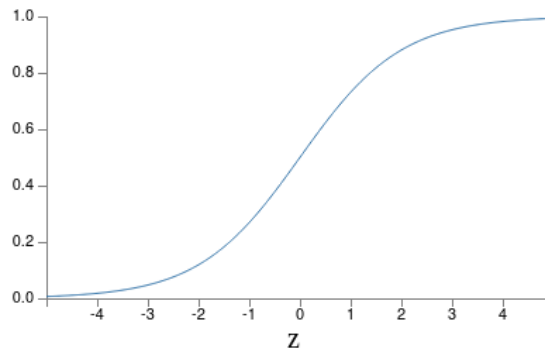
Kluczowym dla zrozumienia działania całej sieci neuronowej jest przekształcenie zachodzące w obrębie pojedynczego neuronu. Model neuronu przedstawiony został na rysunku 3.4. Wejścia do danego neuronu zostają przemnożone przez odpowiednie wagi, a następnie zsumowane. Następnie obliczoną sumę wykorzystujemy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

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

której przebieg widzimy na rysunku 3.5. Warto zauważyć, że w obrębie dostatecznie małych i dużych wartości wejścia do funkcji sigmoidalnej następuje jej nasycenie, 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  $\langle x, y^d \rangle$ , wektor wyjść sieci  $\bar{f}(x; \theta)$  oraz pewną funkcję straty mierzącą rozbieżność między  $y^d$  a wektorem wyjść obliczonym przez sieć, która w naszym przypadku przyjmuje postać funkcji kwadratowej,

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

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

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

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  $\frac{\partial q}{\partial w_{jk}^l}$  oraz  $\frac{\partial q}{\partial b_j^l}$ , gdzie  $w_{jk}^l$  oznaczają wagę łączącą  $k$ -ty neuron warstwy  $l-1$  z  $j$ -tym neuronem warstwy  $l$ , natomiast  $b_j^l$  oznacza wyraz wolny  $j$ -tego neuronu warstwy  $l$ . Dla łatwiejszej prezentacji algorytmu wprowadzamy wartość  $\delta_j^l = \frac{\partial q}{\partial z_j^l}$ , 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  $\delta_j^l$  dla warstwy wyjściowej  $L$  o postaci

$$\delta_j^L = \frac{\partial q}{\partial a_j^L} f'(z_j^L), \quad (14)$$

które dla kwadratowej funkcji kosztu w wersji macierzowej zapisujemy jako

$$\delta_j^L = (a^L - y^d) \odot f'(z_j^L), \quad (15)$$

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

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

Za pomocą równań (15) i (16) możemy wyznaczyć wektory  $\delta_j^l$  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

$$\frac{\partial q}{\partial b_j^l} = \delta_j^l, \quad (17)$$

oraz wag przypisanych do poszczególnych połączeń

$$\frac{\partial q}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad (18)$$

Równania 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 pełny opis można znaleźć w pozycjach [2] oraz [14].

Metoda wstecznej propagacji jest jednym z najczęściej wykorzystywanych algorytmów uczenia w zagadnieniach dotyczących uczenia maszynowego. Ogólne stosowane

podejście w tej dziedzinie polega na trenowaniu wybranej struktury za pomocą wcześniej wyselekcjonowanych danych. W celu ograniczenia ryzyka przeuczenia czyli zbytniego dopasowania się sieci do danych stosuje się strategię polegającą na podziale dostępnej liczby przykładów uczących na dwa zbiory uczący oraz weryfikujący, nazywane również treningowym i testowym. W ogólności metoda wstecznej propagacji powinna być przeprowadzona z wykorzystaniem zbioru uczącego, a kryterium wyjścia kończące procedurę uczenia zdefiniowane za pomocą zbioru testującego. Takie rozwiązanie powinno zapobiec niepożądanego zjawiska przeuczenia.

### 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 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

$$S_i = \frac{1}{2} \frac{\partial^2 q}{\partial \theta_i^2} \theta_i^2. \quad (19)$$

Podstawą do zdefiniowania miary było rozwinięcie funkcji celu  $q$  (12) 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

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

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

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

i zastosować wsteczną propagację w postaci

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

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

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

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

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

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 ponownej modyfikacji 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 strategii, a wszystkie szczegóły wskazane zostaną we właściwym rozdziale.

Pierwszym etapem jest wygenerowanie danych, które następnie stanowią będą przykłady uczące i weryfikujące zarówno dla algorytmu wstecznej propagacji jak i OBD. Generacja danych polega na przeprowadzeniu symulacji z wykorzystaniem regulatora DMC i zdefiniowanego 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. Ten krok wymuszony został przez rodzaj zastosowanej funkcji aktywacji. Warto zauważyć, że koniecznym było zastosowanie modułu skalującego pozwalającego na transformację danych wejściowych w dwie strony. Wyjście sieci neuronowej wykorzystane zostanie jako wartość sterująca, 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, koniecznym staje się zastosowanie lekko zmodyfikowanej strategii względem klasycznego podejścia w uczeniu maszynowym. Algorytm OBD wymaga aby funkcja celu była w swoim minimum, a zatem niezbędne 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 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 małą 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 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. Pozwoli to określić wrażliwość obu wykorzystywanych metod na modyfikacje obiektu regulacji względem tego, do którego zostały skalibrowane.

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, której najważniejsze części przedstawione zostaną w kolejnym rozdziale. Dodatkowo lektura tego rozdziału powinna umożliwić czytelnikowi samodzielnią 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.

Prezentację 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 pozwalającą 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:

---

```
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 wyznaczamy (10, 20, 30)
8         self.d = len(S_list)
9         self.Y_k = np.zeros(n)
10        #wektor wartosci zadanej - stala na calym horyzoncie N
11        self.Y_zad = np.zeros(n)
12        #wektor opisujacy trajektorie sygnalu wyjsciowego
13        self.Y_hat = np.zeros(n)
14        #wektor wyznaczanych przyrostow wartosci sterowania
15        self.U_delta = np.zeros(n_u)
16        #wektor przeszlych przyrostow sterowania
17        self.U_delta_P = np.zeros(len(S_list)-1)
18        #macierze wyznaczane w trakcie inicjalizacji klasy
19        self.Psi = np.diag([psi_const for i in range(n)])
20        self.Lambda = np.diag([lambda_const for i in range(n_u)])
21        self.M_p = self.init_M_p(S_list, n)
22        self.M = self.init_M(S_list, n, n_u)
23        self.K = inv(self.M.T @ self.Psi @ self.M + self.Lambda) @ self.M.T
24        @ self.Psi
```

---

Listing 1: Klasa DMC

#### 4. Opis wybranych elementów implementacji

---

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 linie 23 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  $M^P$ , których implementacja przedstawiona została na Listingu 2.

---

```
1 def init_M_p(self, S_list, N):
2     arr=np.zeros((N,len(S_list)-1))
3     for i in range(N):
4         for j in range(len(S_list)-1):
5             arr[i][j] = (S_list[i+j+1] if i+j+1+1<=len(S_list) else
6                          S_list[-1]) - S_list[j]
7     return arr
8
9 def init_M(self, S_list, n, n_u):
10    arr=np.zeros((n,n_u))
11    for i in range(n):
12        for j in range(i+1 if i+1<=n_u else n_u):
13            arr[i][j]= S_list[i-j] if i-j+1<=len(S_list) else S_list[-1]
14    return arr
```

---

Listing 2: Funkcje wyznaczające macierze  $M$  oraz  $M^P$

Kluczowym elementem całej klasy ze względu na działanie algorytmu jest funkcja wyznaczająca wektor optymalnych przyrostów sterowania w każdej iteracji algorytmu. Implementację funkcji prezentuje Listing 3, gdzie w linii 3 zaimplementowano równanie (8). W każdej iteracji algorytmu zwracana jest tylko wartość  $\Delta u(k|k)$ , której odpowiada zwracana przez funkcję zmienna `delta_u`.

---

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

---

Listing 3: Funcja wyznaczająca wektor przyrostów sterowania

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 4 prezentuje funkcję `make_simulation_step`, w której należy zwrócić uwagę na linie 6, w której widzimy zaimplementowane równanie (10).

---

```

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

```

---

Listing 4: Funkcja zwracająca wyjście obiektu sterowania

## 4.2. Architektura sieci neuronowej

Implementacja sieci neuronowej przedstawiona została w postaci klasy *Network*, której konstruktor widzimy w Listingu 5. Struktura sieci składa się z listy macierzy *weights* (linia 6) oraz *biases* (linia 5) 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.

---

```

1 class Network(object):
2     def __init__(self, sizes):
3         self.num_layers = len(sizes)
4         self.sizes = sizes
5         self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
6         self.weights = [np.random.randn(y, x) for x, y in zip(sizes[:-1],
            ↪ sizes[1:])]
7         self.saliencies = [np.zeros((y, x)) for x, y in zip(sizes[:-1],
            ↪ sizes[1:])]
8         self.mask = [np.ones((y, x)) for x, y in zip(sizes[:-1],
            ↪ sizes[1:])]
9         self.cost_delta_epsilon = 0.000005

```

---

Listing 5: Klasa *Network*

Główną część implementacji sieci neuronowej stanowią funkcje wykorzystywane w trakcie procesu trenowania sieci. Jest to odpowiednio funkcja SGD implementująca metodę stochastycznego najszybszego spadku SGD (*Stochastic Gradient Descent*), jedną z najczęściej stosowanych metod iteracyjnej optymalizacji funkcji celu. Metoda ta korzysta z funkcji *update\_mini\_batch*, która to odpowiada za zastosowanie metody wstecznej propagacji dla każdego z przykładów uczących występujących w pojedynczej próbkę wyznaczonej przez SGD. Kluczowym dla działania wymienionych funkcji jest implementacja

#### 4. Opis wybranych elementów implementacji

---

metody wstecznej propagacji dla pojedynczego przykładu uczącego. Teoretyczne założenia używanej metody przedstawione zostały w sekcji 3.2.2, a jej implementację widzimy na Listingu 6. W początkowej części (linie 2-3) tworzone są wektory odpowiadające pochodnym cząstkowych  $\frac{\partial q}{\partial w_{jk}^l}$  oraz  $\frac{\partial q}{\partial b_j^l}$ . Następnie w liniach 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, gdzie wykorzystane zostało równanie (15) stanowiące warunek początkowy dla algorytmu. Następnie w pętli następuje wsteczna propagacja między warstwami opisana równaniem (16), które znajduje bezpośrednie odzwierciedlenie w linii 22.

---

```
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 wyjścia sieci dla wejścia 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 początkowy
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 końca 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)
```

---

Listing 6: Funkcja wstecznej propagacji

Pozostała część implementacji obejmuje funkcje aktywacji, funkcje wyliczającą 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 przyrzeć 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 7 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 15 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.

---

```

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)) for w, h in
15                               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(prev_delta_cost_list,
21                                                       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

```

---

Listing 7: Algorytm OBD

Niezbędnym dla zrozumienia całego algorytmu OBD jest proces wstecznej propagacji zastosowany dla drugich pochodnych. Jego implementacja zaprezentowana została na Listingu 8 i zawiera wiele podobieństw do wcześniej omawianej funkcji `backprop`. Należy zwrócić tutaj szczególną uwagę na linie 21, która odpowiada równaniu (23). W linii 14 widzimy natomiast wykorzystanie funkcji pomocniczej do wyliczenia warunku granicznego z równania (24), zabieg ten sprzyja czytelności kodu, a z uwagi na mało

#### 4. Opis wybranych elementów implementacji

---

skomplikowaną formę równania nie powinien stanowić utrudnienia w zrozumieniu całej metody.

---

```
1 def backpropOBD(self, x, y):
2     h_vector = [np.zeros(w.shape) for w in self.weights]
3     # feedforward
4     activation = x
5     activations = [x]
6     zs = []
7     for b, w in zip(self.biases, self.weights):
8         z = np.dot(w, activation)+b
9         zs.append(z)
10        activation = sigmoid(z)
11        activations.append(activation)
12
13    #wsteczna propagacja drugich pochodnych
14    delta2_z = self.boundary_OBD_derivative(zs[-1], y)
15    h_vector[-1] = np.dot(delta2_z, activations[-2].transpose()**2)
16    #iteracja po kolejnych warstwach
17    for l in range(2, self.num_layers):
18        z = zs[-l]
19        sp = sigmoid_prime(z)
20        sp2 = sigmoid_second_prime(z)
21        delta2_z = sp**2 * np.dot(self.weights[-l+1].transpose()**2,
22                                   ↪ delta2_z)
23        h_vector[-l] = np.dot(delta2_z, activations[-l-1].transpose()**2)
24
25    return h_vector
```

---

Listing 8: Funkcja wstecznej propagacji drugich pochodnych

---

```
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
```

---

Listing 9: Funkcja redukcji wag

Ostatnią z wykorzystywanych funkcji w procedurze OBD jest funkcja `cut_weights`, która pokazana została na Listingu 9. Odpowiada ona za przycięcie odpowiedniej ilości wag, ustalonej przez parametr `limit` według posortowanych wartości współczynników skośności. W liniach 2-7 współczynniki są sortowane aby następnie w zmiennej `to_cut`

zapisać indeksy odpowiednich wag, które to następnie zostają przycięte i zamrożone z wykorzystaniem opisywanej wcześniej maski.

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 osiągnięte 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. 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 praktykę 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 uchybu 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 osiągnięte przez sieć.



Jak zostało już opisane w części teoretycznej dane uczące i weryfikujące generowane są w sposób niezależny na podstawie oddzielnych przebiegów regulacji. Dzięki takiej strategii mamy pewność, że sieć zostanie przetestowana pod kątem ogólnej aproksymacji algorytmu OBD, a nie jedynie wybranych przebiegów regulacji. 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% do 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 było wymagane przeskalowanie zbioru danych. Jak już zostało wskazane z uwagi na sigmoidalną funkcję aktywacji koniecznym było napisanie modułu transformującego 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 wyjścia sieci, które następnie wykorzystywane 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

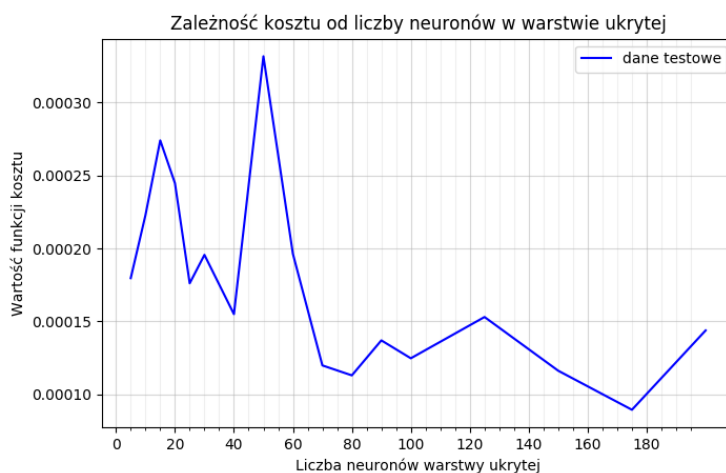
Pierwszym krokiem prowadzącym do wyselekcjonowania optymalnej architektury sieci jest wybór jej struktury. Liczba neuronów w warstwie wejściowej i wyjściowej ściśle zależy do charakterystyki problemu, a więc wygenerowanych uprzednio danych. Szczególną uwagę należy poświęcić za to prawidłowemu doborowi 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 z uwagi na stosowany w kolejnym kroku algorytm OBD. W ramach zaprezentowania 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 kilku istotnych obserwacji. Pierwszą z nich jest to, że nawet niezwykle prosta struktura 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



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



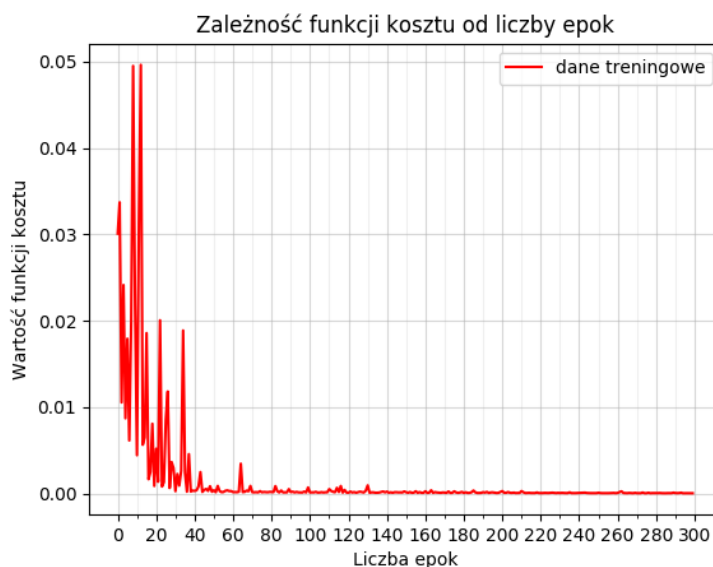
**Rysunek 5.2.** Zależność kosztu od liczby neuronów w warstwie ukrytej - dane testowe

wysoką niestabilnością przez co rezultaty przez nie osiągnięte mogą różnić się pomiędzy kolejnymi próbami. Po kilukrotnym 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 sieć jest trenowana. Pozwoli to na weryfikację wartości granicznej gradientu funkcji celu, która stanowi warunek wyjścia dla metody uczenia sieci. Początkowe 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

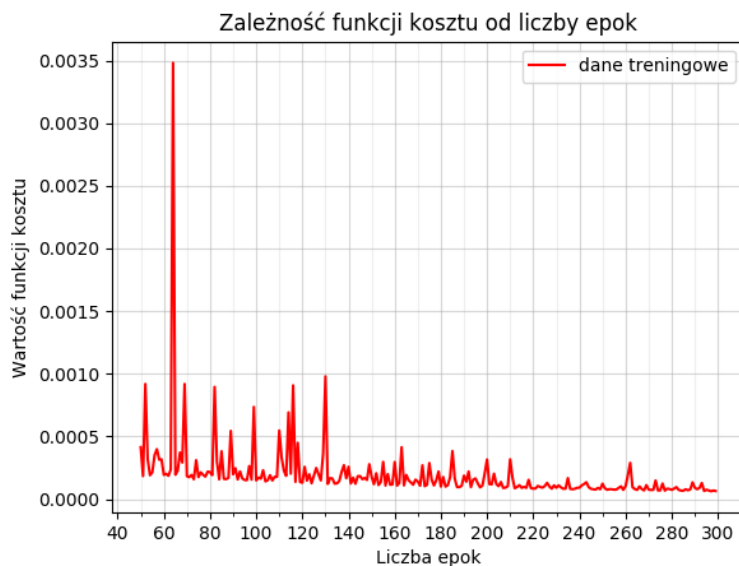
natomiast na Rysunku 5.4 przedstawione zostały te same dane ale z uciętymi 50 początkowymi wartościami w celu dokładniejszej prezentacji wyników czytelnikowi.

Należy zwrócić uwagę, że już po 40 epokach wyniki osiągnięte przez sieć można uznać za satysfakcjonujące, jednak w dalszym ciągu obserwujemy stosunkowo duże wahania wartości funkcji kosztu i nie osiągamy pełnej stabilności. Okresowe wahania ustają po około 150 iteracjach i po takiej liczbie epok możemy uznać, że redukcja funkcji kosztu jest nieznaczna. Dzięki wielokrotnemu powtórzeniu eksperymentu oraz przetestowaniu różnych wartości kryterium wyjścia za zadowalającą wartość przyjęto  $5 \cdot 10^{-6}$ , co stanowi redukcję wcześniej przyjętego poziomu o połowę.



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

Na tym etapie dokonano wstępnego porównania klasycznej metody DMC z regulatorem opartym o w pełni wytrenowaną sieć neuronową, jeszcze przed zastosowaniem algorytmu OBD. W tym celu przeprowadzono następujący eksperyment, sprawdzono zdolność dostosowania wyjścia obiektu do jednokrotnego skoku wartości zadanej. Za miarę jakości regulacji przyjęto powszechnie stosowany błąd średniokwadratowy (*ang. mean squared error, MSE*). Sprawdzono zdolność dostosowania wyjścia obiektu do 5 różnych wartości zadanych: 2,5 ; 4 ; 5,21 ; 6,1 ; 7. Wartości dobrane zostały w taki sposób aby sprawdzić zachowanie regulatora na całym zakresie reprezentowanym przez dane uczące. Eksperyment został powtórzony 10-krotnie aby wyeliminować wpływ pewnej losowości, jaką charakteryzuje się proces uczenia sieci neuronowej. Uśrednione wartości błędu MSE dla każdej z wartości zadanej zaprezentowane zostały w tabeli 5.1. Analizując przedstawioną tabelę możemy stwierdzić, że sieć neuronowa poradziła sobie z zadaniem regulacji, które zostało przed nią postawione. Co więcej dla czterech z pięciu przypadków obserwujemy mniejszą wartość błędu MSE niż dla regulatora DMC. Przypadek dla którego



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

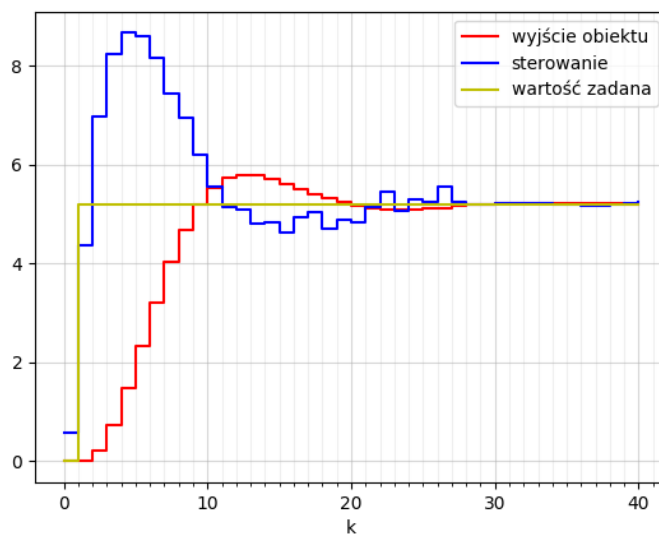
**Tabela 5.1.** Porównanie MSE dla w pełni wytrenowanej sieci

Wartość zadana	MSE sieć	MSE DMC
2,5	0,606	0,592
4	1,499	1,517
5,21	2,518	2,573
6,1	3,457	3,527
7	4,579	4,645
Średnia	2,532	2,571

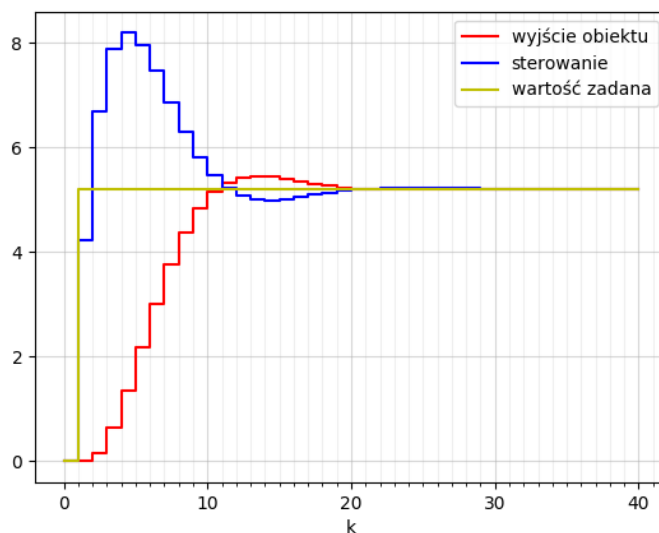
klasyczny regulator poradził sobie nieznacznie lepiej dotyczył najmniejszej wartości zadanej co może świadczyć o tym, że w obrębie względnie małych zmian wartości zadanych regulacja oparta o sieć neuronową charakteryzuje się mniejszą precyzją. Warto również zwrócić uwagę na to, że dla wartości zadanej wynoszącej 5,21 sieć w pełni poradziła sobie z zadaniem regulacji przed nią postawionym. Jest to o tyle ważne, że dana wartość nie była reprezentowana w danych uczących co udowadnia, że sieć neuronowa nauczona została ogólnego zadania regulacji, a nie jedynie przebiegów dla pojedynczych wartości zadanych, które znalazły odzwierciedlenie w przykładach trenujących.

Warto pokazać również przebieg jednej z przykładowych regulacji, na podstawie których wyznaczona została Tabela 5.1. Zaprezentowane zostaną przebiegi dla wartości zadanej 5,21 z przyczyn opisanych w poprzednim paragrafie. Na Rysunku 5.5 widzimy regulację z wykorzystaniem sieci neuronowej dla wybranej wartości zadanej, natomiast Rysunek 5.6 pokazuje referencyjny przebieg uzyskany z wykorzystaniem regulatora DMC. Dokonując porównania dwóch wykresów możemy stwierdzić, że przebiegi wyjścia obiektu

dla dwóch różnych regulatorów są do siebie zbliżone. Warto zauważyć, że regulacja z wykorzystaniem sieci neuronowej charakteryzuje się mniejszą stabilnością sygnału sterującego. Z całą pewnością jest to istotne ograniczenie, które jednak może potwierdzać zauważoną zależność w trakcie przeglądu literatury, przykłady praktycznego zastosowania sieci neuronowej w regulacji dotyczą jedynie wysoce dynamicznych obiektów. Pełnej weryfikacji postawionej tezy będzie można jednak dokonać po przetestowaniu uproszczonej sieci neuronowej.



**Rysunek 5.5.** Regulacja do wartości zadanej 5,21 - wytrenowana sieć neuronowa



**Rysunek 5.6.** Regulacja do wartości zadanej 5,21 - DMC

Przeprowadzony eksperyment wskazuje również na zasadność ustalenia kryterium

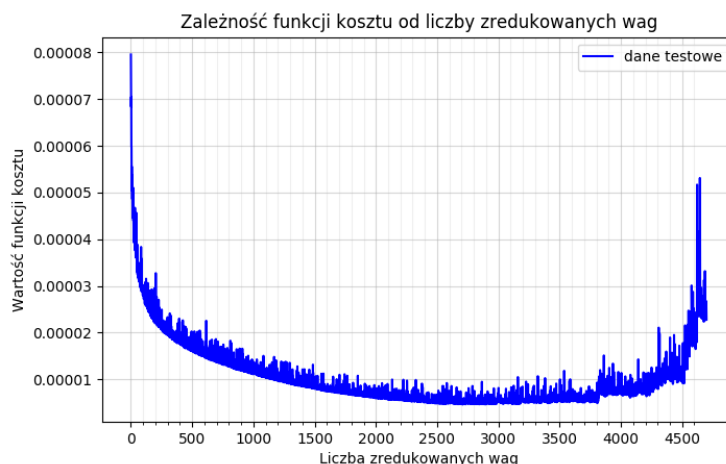
wyjścia na wartość  $5 \times 10^{-6}$ . W trakcie 10 powtórzeń uczenie sieci kończyło się w zakresie od 126 do 258 iteracji, po których uśredniony koszt wynosił 0,00012 z odchyleniem standardowym wynoszącym  $2,9 \times 10^{-5}$ . Są to rezultaty, które możemy uznać za satysfakcjonujące przed przystąpieniem do procedury redukcji sieci.

### 5.3. Zastosowanie algorytmu OBD

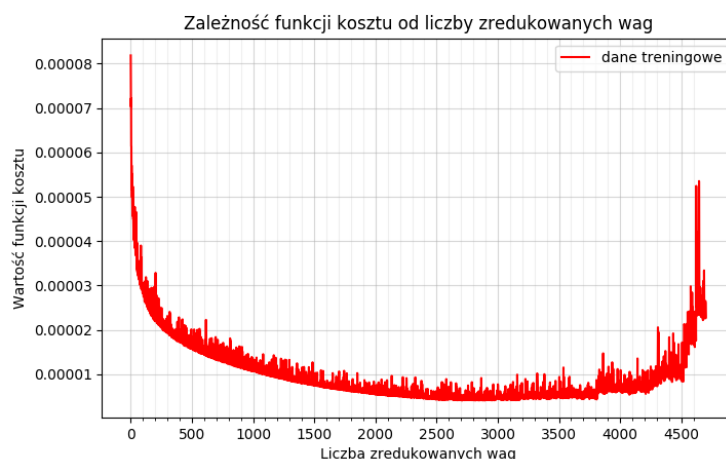
Jak już zostało wskazane w rozdziale z opisem teoretycznym zastosowanie algorytmu OBD wymaga wcześniejszego osiągnięcia minimum funkcji kosztu na danych uczących. Z tego względu algorytm OBD może zostać zastosowany do sieci neuronowej stanowiącej wynik prac opisanych w poprzednim podrozdziale. Zgodnie z procedurą opisaną w sekcji 3.2.3 upraszczanie sieci neuronowej następuje w wersji iteracyjnej co oznacza, że w każdej kolejnej iteracji algorytmu redukowana będzie o jedna waga więcej. Jest to stosunkowo długa procedura gdyż wykorzystywana w pracy architektura składa się z 4800 połączeń między-neuronowych i jest to jednocześnie długość pętli algorytmu OBD. W każdym kroku należy również ponownie nauczyć sieć aby spełniony był warunek o osiągnięciu minimum funkcji celu. Z tego względu czas obliczeniowy algorytmu wykonywanego na komputerze osobistym może wynieść nawet do kilku godzin, a wielokrotne powtarzanie procedury może okazać się niedogodne. Warto zauważyć również, że przycinanie stosowane jest dla wszystkich wag jednocześnie bez rozróżnienia na to pomiędzy, którymi warstwami sieci występują.

Kluczowym dla ostatecznej architektury jest prawidłowy wybór kryterium wyjścia z pętli algorytmu OBD. Istotnym jest wybór liczby upraszczanych wag, która pozwoli na zwiększenie zdolności generalizacji danego zadania. Jednocześnie należy pamiętać o tym, że redukcja zbyt dużej ilości połączeń może znacząco zaburzyć działanie sieci i drastycznie zwiększyć popełniany przez nią błąd. Optymalnym wyborem jest przeprowadzenie procedury OBD z wykorzystaniem danych uczących natomiast uzależnienie kryterium wyjścia od wartości funkcji kosztu dla danych weryfikujących. Zależy nam na minimalizacji wskazanego kosztu przy jednoczesnym zapewnieniu stabilnego działania sieci. Zdefiniowanie odpowiedniej miary a priori może okazać się niezwykle trudnym zadaniem dlatego pomocnym będzie eksperyment zastosowania procedury upraszczania sieci na całym dostępnym zakresie parametrów. W ten sposób wyznaczymy zależność funkcji kosztu od liczby zredukowanych wag sieci neuronowej. Wyniki eksperymentu zaprezentowane zostały na rysunkach 5.7 oraz 5.8, które prezentują wykres funkcji kosztu dla kolejno danych testowych i treningowych. Na obu wykresach zakres prezentowanych wyników ograniczony został do 4700 początkowych pętli algorytmu, powyżej tej wartości następuje gwałtowny wzrost funkcji kosztu co przekłada się na wypłaszczenie pozostałej części przebiegu. W konsekwencji uniemożliwiłoby to prawidłową interpretację uzyskanych rezultatów.

Główną uwagę należy zwrócić na wykres dla danych testowych gdyż to minimaliza-



**Rysunek 5.7.** Zależność funkcji kosztu od liczby zredukowanych wag - dane testowe



**Rysunek 5.8.** Zależność funkcji kosztu od liczby zredukowanych wag - dane treningowe

cja funkcji kosztu popełnianego na tym zestawie danych jest podstawowym zadaniem algorytmu OBD. Warto jednak zauważyć, że podobnie jak to miało miejsce w przypadku badania wpływu liczby neuronów warstwy ukrytej na funkcję kosztu, tak i tutaj przebiegi funkcji dla danych treningowych i testowych są praktycznie identyczne. Jest to jednak zależność, której należało się spodziewać na podstawie wcześniejszych obserwacji. Istotnym z punktu widzenia pracy jest fakt, że zastosowanie procedury przycinania sieci w istotny sposób wpływa na redukcję wartości kosztu. W przypadku naszego eksperymentu obserwowaliśmy prawie ośmiokrotny spadek analizowanej statystyki. Wprawdzie możliwe było by osiągnięcie mniejszego bazowego kosztu dzięki wydłużeniu początkowej liczby epok, przez które trenowana była sieć jednak jak wykazaliśmy wcześniej nie można oczekiwać, aż tak znacznej redukcji jaką obserwujemy w trakcie procedury przycinania. Istotnym jest również fakt, że w czasie douczania sieci neuronowej po każdej kolejnej redukcji stosowane było takie samo kryterium wyjścia jak w początkowym etapie. W

trakcie trwania całego eksperymentu liczba epok, po której dotrenowanie było zakańczane nie przekroczyła 50. Na tej podstawie należy stwierdzić, że algorytm OBD w istotny sposób przyczynia się do redukcji funkcji kosztu zarówno dla danych testowych jak i treningowych, a jego zastosowanie powinno przyczynić się do większej stabilności regulacji.

Analizując Rysunki 5.7 oraz 5.8 należy stwierdzić, że pomimo osiągnięcia zadowalających rezultatów, na całym przebiegu funkcji występują zauważalne oscylacje pomiędzy kolejnymi iteracjami. Powoduje to, że zdefiniowanie kryterium wyjścia opartego na gradiencie funkcji staje się niestabilne i nie pozwala na osiągnięcie zadowalającej liczby zredukowanych wag. Z tego względu zastosowana została inna strategia. Na podstawie zaprezentowanych przebiegów widzimy, że funkcja kosztu maleje aż zredukowane zostanie około 2500 parametrów czyli nieco ponad połowa wag sieci. Dalsze upraszczanie prowadzi do utrzymania kosztu na prawie stałym poziomie aż przycięte zostanie zbyt dużo parametrów co w rezultacie uniemożliwia sieci generalizację problemu przed nią postawionego. Dla danych testowych minimum wynoszące  $4,614 * 10^{-6}$  osiągnięte zostało po przycięciu 2708 wag. Kierując się powyższymi obserwacjami zasadnym jest ustalenie kryterium zakończenia procedury przycinania sieci na redukcję przynajmniej 2600 wag i osiągnięcie historycznego minimum funkcji kosztu. Liczba 2600 wag odpowiada za 54,16% wszystkich połączeń i jest to znaczna redukcja. Znajduje ona jednak swoje potwierdzenie w wynikach prac zaprezentowanych w przeglądzie literatury gdzie autorzy wskazują na konieczność redukcji od 40% do nawet 60% połączeń.

Zgodnie z zaproponowaną procedurą wyznaczona została optymalna struktura sieci, której zdolności regulacji porównane zostaną z klasycznym regulatorem DMC. Procedura upraszczania sieci zakończyła się po przycięciu 2621 wag z 4800 wszystkich połączeń. Warto zwrócić uwagę ile wag zostało przyciętych pomiędzy kolejnymi warstwami. Kolejno między warstwą wejściową a ukrytą zredukowanych zostało 2603 z 4650 połączeń, natomiast między warstwą ukrytą a wyjściem sieci było to 18 ze 150 połączeń. Dzięki zastosowaniu algorytmu OBD udało się zredukować wartość funkcji kosztu na danych testowych z wartości  $8,586 * 10^{-5}$  do  $5,121 * 10^{-6}$ . Bezwzględna wartość zmiany kosztu może nieznacznie różnić się pomiędzy kolejnymi powtórzeniami eksperymentu jednak każdorazowo obserwowano około 90% redukcję na danych testowych co jednoznacznie świadczy o zasadności wykorzystania procedury OBD w procesie trenowania sieci neuronowej.

Po wyselekcjonowaniu ostatecznej struktury sieci możliwym jest porównanie jakości regulacji dokonywanej przez sieć neuronową z algorytmem OBD. W celu oceny dwóch metod wykorzystana zostanie analogiczna procedura jak w przypadku porównania dokonanego po pełnym wytrenowaniu sieci. Zastosowano jednak dwie modyfikacje stosowanej metody. Pierwsza z nich dotyczy redukcji liczby powtórzeń eksperymentu z 10 do 5, podyktowane zostało to czasem niezbędnym do przeprowadzenia iteracyjnej procedury OBD. Druga modyfikacja sprzyjać ma natomiast dokonaniu kompleksowego porównania



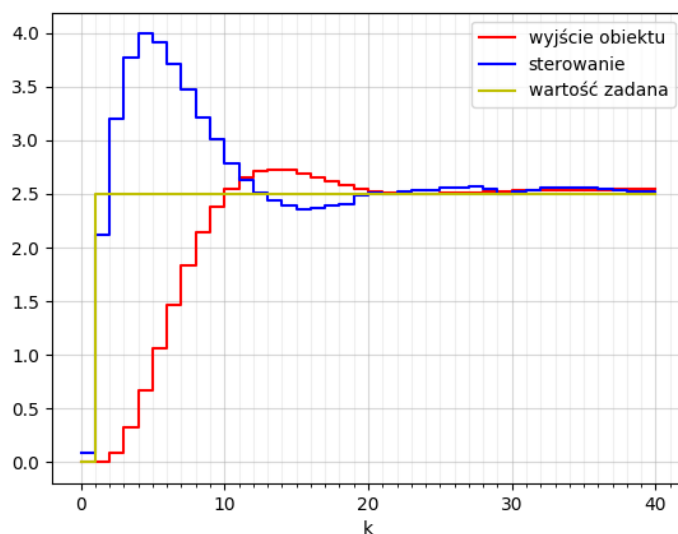
dla całego zakresu wartości zadanych reprezentowanych w danych uczących. Sprawdzona zostanie błąd MSE nie tylko dla 5 wybranych wartości zadanych (2,5 ; 4 ; 5,21 ; 6,1 ; 7) ale dla pełnego zakresu od 2 do 9 z krokiem co jedną dziesiątą. Wyniki zaprezentowane zostały w tabeli 5.2, w której jedyna modyfikacja względem poprzednio prezentowanej polega na zastąpieniu wiersza "Średnia" poprzez wiersz pełny zakres, który w istocie stanowi średnią wyznaczoną na opisanym wcześniej zakresie wartości zadanych.

**Tabela 5.2.** Porównanie MSE dla w pełni wytrenowanej sieci

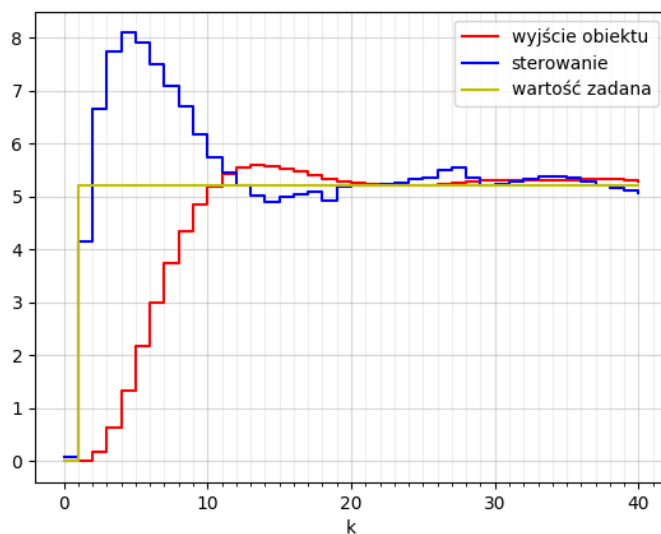
Wartość zadana	MSE sieć	MSE DMC
2,5	0,588	0,592
4	1,519	1,517
5,21	2,563	2,573
6,1	3,553	3,527
7	4,579	4,645
Pełny zakres	3,206	3,207

Analizując tabele należy zauważyć, że zarówno MSE na pełnym zakresie jak i dla 5 poszczególnych przypadków nie różni się znacząco pomiędzy obiema metodami. Biorąc pod uwagę pewną losowość procedury uczenia sieci neuronowej oraz zmniejszenie liczby powtórzeń eksperymentu należy uznać powyższe wartości za identyczne. Istotnym jest fakt, że dzięki algorytmowi OBD udało wyeliminować się problem niższej jakości regulacji dla małych wartości zadanych. Jak widzimy w wierszu 2,5 (Tabela 5.2) wartość błędu MSE dla sieci neuronowej jest nieznacznie mniejsza od tej dla algorytmu DMC, a w ogólności należy uznać te wartości za równe. Pokazuje to, że sieć neuronowa po zastosowaniu algorytmu OBD jest w stanie odwzorować zadanie regulacji na pełnym zakresie wartości zadanych, których została nauczona. Potwierdzenie postawionego wniosku czytelnik może znaleźć również w wizualizacji przykładowego przebiegu regulacji dla omawianej wartości zadanej, który zaprezentowany został na Rysunku 5.9. Widzimy na nim prawie bezbłędne odwzorowanie przebiegu regulacji DMC.

Warto zauważyć również, że regulator oparty o sieć neuronową radzi sobie prawie bezbłędnie z utrzymaniem stałej wartości wyjścia obiektu po osiągnięciu wartości zadanej. Dla pełniejszego porównania wpływu redukcji sieci na działanie regulatora na Rysunku 5.10 przedstawiony został przebieg regulacji dla wartości zadanej 5.21. Stanowi to analogiczny przypadek do regulacji z wykorzystaniem niezredukowanej sieci neuronowej, którą widzimy na wcześniejszym rysunku 5.5. Porównując przebiegi obu symulacji należy zwrócić uwagę na dwie istotne przewagi zredukowanej sieci. Algorytm OBD pozwolił na redukcję wartości przeregulowania oraz przyczynił się do wyraźnie większej stabilności przebiegu kolejnych wartości sterujących pokazanych na wykresach niebieską linią. Wskazane korzyści jednoznacznie potwierdzają zasadność wykorzystania algorytmu OBD w trakcie nauki przez sieć neuronową zadania regulacji.



**Rysunek 5.9.** Regulacja do wartości zadanej 2,5 - zredukowana sieć neuronowa



**Rysunek 5.10.** Regulacja do wartości zadanej 5,21 - zredukowana sieć neuronowa

#### 5.4. Zastosowanie innych obiektów regulacji

Całość analizy przedstawionej w poprzednich podrozdziałach pozwoliła na wyselekcjonowanie i pełne wytrenowanie struktury sieci neuronowej, która w najlepszy sposób generalizuje postawione przed nią zadanie regulacji obiektu identyfikowanego przez następujące parametry  $T_1 = 5$ ,  $T_2 = 2$ ,  $K = 1$ ,  $T_d = 0$ . Na podstawie dokonanych porównań należy stwierdzić, że regulator oparty o sieć neuronową nie posiada istotnych ograniczeń względem swojego klasycznego odpowiednika wykorzystującego metodę DMC. Jednak w realnym świecie zadanie regulacji, a co za tym idzie obiekt regulacji może charakteryzować

się pewną zmiennością, która to przyczynić się może do istotnego spadku efektywności działania wykorzystywanego regulatora. Ostatnim etapem badań zaprezentowanych w tej pracy będzie zbadanie odporności danych regulatorów na modyfikacje pierwotnego obiektu regulacji. W praktyce do modyfikacji danego obiektu regulacji posłużą zmiany wcześniej ustalonych parametrów identyfikujących obiekt. Warto zauważyć, że w trakcie porównania zarówno dane, na podstawie których uczona będzie sieć jak i wektor odpowiedzi skokowej dla algorytmu DMC wyznaczany był na podstawie pierwotnego obiektu regulacji. Strategia porównania opiera się na przeprowadzeniu analogicznych symulacji jak w poprzedniej części pracy i porównaniu kolejnych błędów MSE.

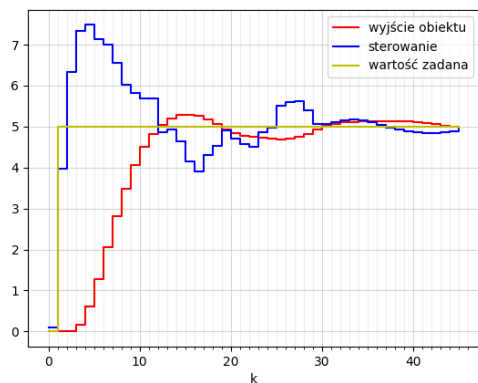
Pierwszy z przeprowadzonych eksperymentów dotyczył modyfikacji parametru  $T_d$ . Porównano jakość regulacji dla 5 kolejnych wartości parametru  $T_d$ : 1, 2, 3, 4, 5. W przypadku każdej z tych wartości wyznaczono uśredniony błąd MSE po kolejnych symulacji dla wartości zadanych z zakresu od 2 do 9. Wyniki eksperymentu zaprezentowane zostały w tabeli 5.3. Kolumna z referencyjną wartością błędu zawiera statystykę dla regulacji DMC wyznaczonej na podstawie odpowiedzi skokowej obecnego układu regulacji. Analizując kolejne wartości należy zauważyć wyraźny wzrost błędu dla obu regulatorów względem wartości referencyjnej, zwłaszcza dla wartości opóźnienia obiektu większych od dwóch. Dla pierwszego i drugiego przypadku można jednak uznać, że jakość regulacji jest akceptowalna. Kluczowym jest natomiast fakt, że dla każdego z przypadków błąd MSE popełniany przez regulator oparty o sieć neuronową jest zauważalnie mniejszy, a wraz z zwiększaniem opóźnienia różnica narasta. Jednoznacznie wskazuje to, że sieć neuronowa jest bardziej odporna na modyfikacje opóźnienia układu zwłaszcza w obrębie mniejszych zmian.

**Tabela 5.3.** Porównanie MSE dla różnych wartości  $T_d$  zmodyfikowanego obiektu

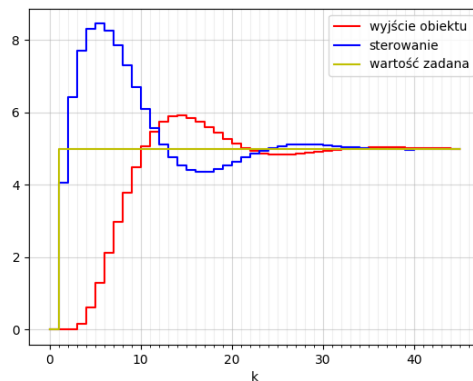
$T_d$	MSE sieć	MSE DMC	MSE referencyjne
1	3,639	3,668	3,588
2	4,46	4,827	4,323
3	5,352	6,593	5,057
4	6,281	9,277	5,792
5	7,221	13,490	6,526

Dla pełniejszego obrazu postawionego twierdzenia warto przyjrzeć się przebiegom regulacji dla pierwszych dwóch przypadków. Przebiegi z wykorzystaniem dwóch metod dla kolejno parametru  $T_d$  równego 1 oraz 2 przedstawione zostały na Rysunkach 5.11-5.14. Wszystkie wykresy wyznaczone zostały dla przykładowej wartości zadanej wynoszącej 5. Na podstawie ogólnej analizy należy stwierdzić, że sieć neuronowa poradziła sobie lepiej niż regulator DMC, kluczowa jest tutaj wartość przeregulowania, która jest wyraźnie niższa zarówno dla pierwszego jak i drugiego przypadku. Jednoznacznie wskazuje to na większą odporność sieci neuronowej na zmiany opóźnienia układu względem tradycyjnego regulatora DMC. Powodów takiej zależności szukać możemy w wykorzystanej procedurze

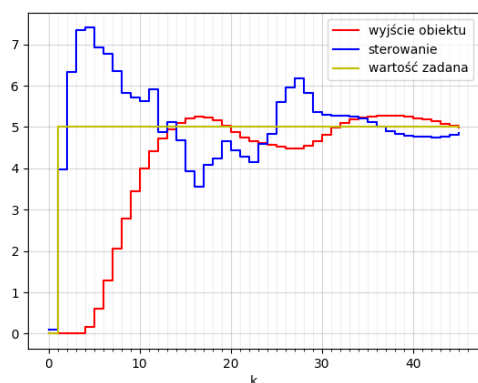
upraszczania sieci, która w założeniu sprzyjać ma generalizacji danego problemu. Ostateczna weryfikacja postawionej tezy wymagałaby jednak przeprowadzenia dalszych prac nad omawianym zagadnieniem.



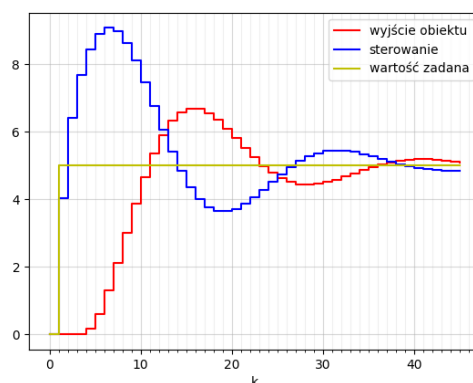
**Rysunek 5.11.** Regulacja Sieć obiekt ( $T_d = 1$ )



**Rysunek 5.12.** Regulacja DMC obiekt ( $T_d = 1$ )



**Rysunek 5.13.** Regulacja Sieć obiekt ( $T_d = 2$ )



**Rysunek 5.14.** Regulacja DMC obiekt ( $T_d = 2$ )

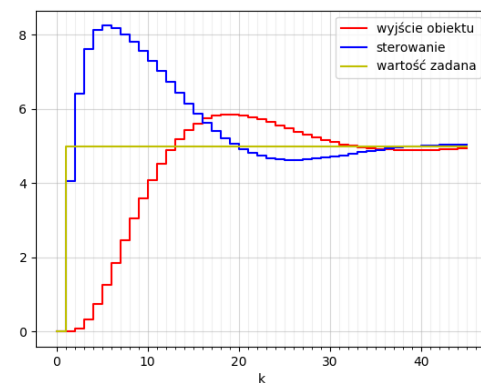
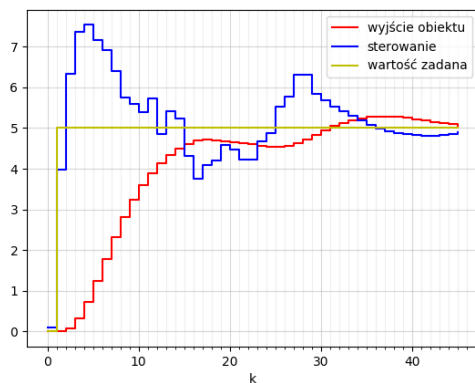
Kolejnym i zarazem ostatnim eksperymentem przeprowadzonym w ramach niniejszej pracy było sprawdzenie jakości regulacji dla zmodyfikowanych układów regulacji z wykorzystaniem parametrów  $T_1$  oraz  $T_2$ , które początkowo wynosiły odpowiednio 5 i 2. Analizę przeprowadzono dla kilku modyfikacji, które zostały dobrane w arbitralny sposób tak aby najpełniej przedstawić możliwe warianty różnych zmian tych parametrów. Wyniki analogicznego eksperymentu do tego wykonanego wcześniej dla wpływu parametru  $T_d$  przedstawiono w Tabeli 4. Analizując przedstawione wartości błędów MSE możemy jednoznacznie stwierdzić, że obie metody regulacji są odporniejsze na zmiany parametrów  $T_1$  oraz  $T_2$  w stosunku do wcześniej prezentowanego parametru  $T_d$ . Wzrost błędu MSE dla obu metod względem wartości referencyjnej jest w większości przypadków nieznaczny,

a w pojedynczych przypadkach widzimy też mniejsze wartości. Jedynie w przypadku zwiększenia parametrów do wartości 7 i 6 (wiersz 5 Tabeli) rejestrujemy zauważalny wzrost wartości błędu MSE w przypadku sieci jest to wzrost o 18% natomiast dla DMC o 20%.

**Tabela 5.4.** Porównanie MSE dla różnych wartości  $T_1$  oraz  $T_2$  zmodyfikowanego obiektu

$T_1$	$T_2$	MSE sieć	MSE DMC	MSE referencyjne
6	2	3,114	3,110	3,039
5	3	3,284	3,289	3,199
7	3	3,881	3,903	3,580
4	1	2,296	2,205	2,229
7	6	5,203	5,291	4,412
3	2	2,558	2,361	2,411
8	1	3,087	3,074	2,888

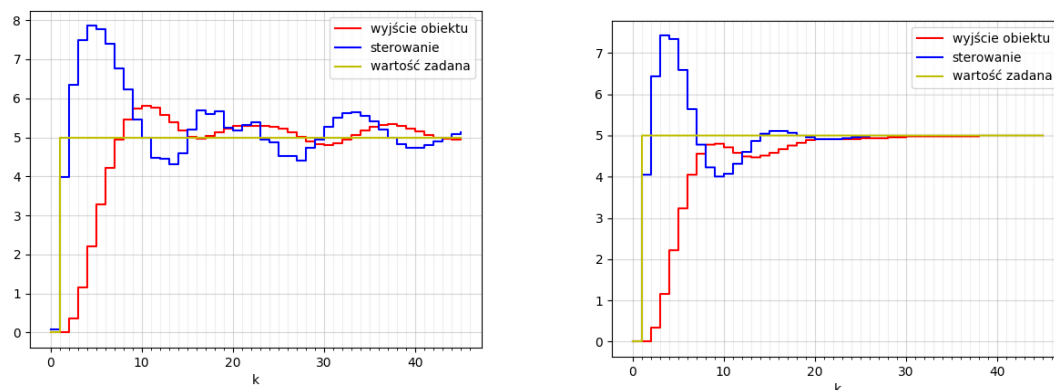
Dla lepszego zobrazowania przedstawionych różnic warto przyjrzeć się przebiegom regulacji dla dwóch przypadków. Pierwszy z nich dotyczy parametrów  $T_1 = 7$ ,  $T_2 = 3$  gdy to sieć poradziła sobie lepiej ze zmianą obiektu. Rysunki 5.15 oraz 5.16 prezentują kolejno regulację z wykorzystaniem sieci neuronowej i algorytmu DMC. Główną wadą regulatora DMC w tym przypadku jest stosunkowo duża wartość przeregulowania, która nie wystąpiła przy wykorzystaniu sieci. Z drugiej strony sieć neuronowa także miała zauważalne problemy z osiągnięciem zadanej wartości.



**Rysunek 5.15.** Regulacja Sieć obiekt ( $T_1 = 7$ ,  $T_2 = 3$ ) **Rysunek 5.16.** Regulacja DMC obiekt ( $T_1 = 7$ ,  $T_2 = 3$ )

Drugim z prezentowanych przypadków dotyczy niższej wartości MSE dla algorytmu DMC, przykładem jest tutaj modyfikacja parametrów do wartości  $T_1 = 4$ ,  $T_2 = 1$ . Analogiczne przebiegi regulacji względem poprzedniego punktu znajdują się na Rysunkach 5.17 i 5.18. Na wykresach odnajdujemy potwierdzenie wartości przedstawionych w tabeli, sieć

neuronowa poradziła sobie gorzej z powodu występujących oscylacji wokół wartości zadanej. Regulacja oparta o algorytm DMC w tym przypadku przebiegła również z pewnymi zakłóceniami jednak należy uznać ją za bliższą porządnemu efektowi.



**Rysunek 5.17.** Regulacja Sieć obiekt ( $T_1 = 4$ ,  $T_2 = 1$ ) **Rysunek 5.18.** Regulacja DMC obiekt ( $T_1 = 4$ ,  $T_2 = 1$ )

Przedstawione w tym podrozdziale analizy pozwalają wyciągnąć kilka istotnych wniosków. Zarówno regulatory oparte o sieci neuronowe jak i klasyczny regulator OBD są wrażliwe na zmiany obiektu regulacji. Należy jednak zauważyć, że wpływ zmian parametru opóźnienia prowadzi do wyraźnie większego zaburzenia działania obu metod w stosunku do zmian parametrów  $T_1$  oraz  $T_2$ . Pierwsza z analiz pokazała, że w obrębie małych zmian parametru  $T_d$  sieć neuronowa wykazała się większą odpornością na modyfikację układu, a uzasadnienia możemy szukać w wykorzystanej w pracy procedurze OBD. Wprawdzie jest to jedynie hipoteza, która wymaga dokładniejszej weryfikacji w czasie dalszego rozwoju zagadnienia. Nie mniej jednak wskazaną zależność należy uznać, za jedną z podstawowych przewag badanego w pracy alternatywnego podejścia względem klasycznej metody regulacji predykcyjnej. W przypadku analizy dla obiektów zmodyfikowanych poprzez parametry  $T_1$  oraz  $T_2$  wykazane zostało, że obie metody porównywalnie dobrze radzą sobie z postawionym zagadnieniem. Nie można wskazać tutaj wyższości żadnej z metod należy jednak pamiętać, że obie z nich poradziły sobie umiarkowanie dobrze z tą klasą zmian obiektów regulacji.

Lektura niniejszego rozdziału dostarczyła czytelnikowi pełnego obrazu dokonanych eksperymentów. Szczegółowo omówiony został proces uczenia oraz upraszczania sieci. Po każdym z tych etapów porównano efektywność sieci neuronowej z klasycznym regulatorem DMC. Pod koniec przeanalizowano odporność dwóch regulatorów na zmiany obiektu regulacji. Wykazano istotne ograniczenia prezentowanych metod, a także wskazano na konkretne przypadki, w których regulacja oparta o sieć neuronową okazała się efektywniejsza. W czasie badań skupiono się na możliwie uniwersalnym podejściu dlatego wybrane zadanie regulacji może wydawać się trywialne, jednak zaprezentowane wyniki

stanowić powinny wstęp do dalszej pracy nad omawianym zagadnieniem. Szczególnie interesującym może być porównanie efektywności obu metod dla wysoce dynamicznych procesów regulacji.

## 6. Podsumowanie

Praca w swoim założeniu miała na celu dokonanie możliwie uniwersalnej weryfikacji zdolności sieci neuronowej do generalizacji zadania regulacji. Jak zostało wskazane w przeglądzie literatury istnieją przykłady udanego wykorzystania sztucznych sieci neuronowych w obszarze automatyki jakim jest sterowanie. Dostępne prace skupiają się na opisanu praktycznych zastosowań różnych rodzajów sieci neuronowych do regulacji wysoce dynamicznych procesów. Niniejsza praca natomiast wpisuje się w pewną niszę polegającą na braku prac przedstawiających możliwie ogólny charakter porównania.

Kierując się wskazanymi założeniami, w pracy zdecydowano się na dokonanie porównania pomiędzy dwoma możliwie prostymi algorytmami z dziedziny sztucznych sieci neuronowych oraz regulacji predykcyjnej. Za dobrego reprezentanta pierwszej grupy wybrano jednokierunkową sztuczną sieć neuronową z jedną warstwą ukrytą. Natomiast oczywistym wyborem w przypadku tradycyjnego podejścia do zagadnienia regulacji predykcyjnej była jedna z najczęściej stosowanych metod tego typu czyli algorytm DMC. Dodatkowo mając na celu otrzymanie możliwie uniwersalnego aproksymatora zadania regulacji zdecydowano się wykorzystać w pracy metodę redukcji sieci neuronowej. Przycinanie wag w swoim założeniu ma na celu zwiększyć zdolność generalizacji danego zagadnienia poprzez wyeliminowanie niepotrzebnych połączeń międzyneuronowych. W tej dziedzinie powszechne uznanie zyskał algorytm OBD i to właśnie on został wykorzystany w trakcie badań.

Niewątpliwą wartością dodaną niniejszej pracy jest samodzielna implementacja wykorzystywanych struktur. Pozwoliło to na pełniejsze zrozumienie tematyki badania ale za to niewątpliwie wiąże się z utratą wydajności w stosunku do metod implementowanych w zewnętrznych bibliotekach. Biorąc jednak pod uwagę podstawowe zadanie regulacji, które zostało wykorzystane do porównania dwóch metod, nie stanowi to istotnego ograniczenia i nie powinno być uznane za wadę. Zaimplementowane struktury pozwoliły w pierwszej kolejności na wygenerowanie i odpowiednie przeskalowanie danych uczących i weryfikujących. Warto podkreślić, że oba zbiory generowane były w sposób niezależny co pozwala na weryfikację działania sieci pod kątem ogólnego zadania regulacji, a nie jedynie odwzorowania konkretnych przebiegów.

Dalsze prace podzielone zostały na trzy etapy. W pierwszej części skupiono się na doborze odpowiedniej architektury sieci pozwalającej na optymalną generalizację zadania regulacji. Efektem tych prac był wybór struktury opartej o 150 neuronów warstwy ukrytej, a za zmienne na podstawie których odbywa się sterowanie przyjęto 30 wartości uchybu regulacji oraz aktualną wartość zadaną. Następnie zbadano zależność funkcji kosztu od liczby iteracji algorytmu uczenia sieci, pozwoliło to na zdefiniowanie spójnego kryterium zakończenia uczenia, które skutkowało kończeniem procedury po około 200 epokach. Warto zauważyć, że kryterium wyjścia dotyczy danych uczących, a nie weryfikujących jak w przypadku większości rozwiązań z dziedziny uczenia maszynowego. Modyfikacja wynika



wprost z założenia konieczności osiągnięciu minimum funkcji kosztu przed rozpoczęciem procedury OBD. Po zakończonym etapie uczenia dokonano wstępnego porównania wyselekcjonowanej struktury sieci z regulatorem DMC. Wyniki wskazują na zbliżoną efektywność obu metod, z pewnymi niedoskonałościami działania sieci neuronowej w obszarze małych wartości zadanych.

Drugi etap prac dotyczył zastosowania procedury redukcji sieci neuronowej. W trakcie eksperymentów wykazano, że redukcja blisko 55% połączeń międzyneuronowych przyczynia się do około 90% procentowej redukcji funkcji kosztu na danych weryfikujących. Niewątpliwie jest to jednoznaczne udowodnienie korzyści wynikających z zastosowania algorytmu OBD. Po skutecznym uproszczeniu sieci dokonano pełnego porównania dwóch podejść do zagadnienia regulacji. Wyniki pokazują, że sieć neuronową w pełni poradziła sobie z zadaniem regulacji, a co ważne udało się wyeliminować początkowe problemy występujące w obszarze małych wartości zadanych. Za jedyną niedoskonałość regulacji opartej o sieć neuronową należy uznać problem z długotrwałym utrzymaniem wartości sterowania na stałej wartości. Analizując wartości błędów MSE należy przyjąć, że problem ma charakter marginalny nie mniej jednak wskazuje to na zasadność wykorzystywania regulacji opartej o sieci neuronowe głównie w przypadku wysoce dynamicznych procesów, co w ogólności znajduje swoje potwierdzenie w dokonanym przeglądzie literatury.

Zakończeniem prac było zweryfikowanie odporności dwóch porównywanych regulatorów na zmiany obiektu regulacji. W pracy za przykład obiektu posłużył człon inercyjny drugiego rzędu z opóźnieniem. Wykazano, że zmiana opóźnienia w znaczący sposób przyczynia się do wzrostu błędu regulacji dla obu metod. Jednak należy stwierdzić, że w obrębie małych modyfikacji sieć neuronowa poradziła sobie wyraźnie lepiej od algorytmu DMC i należy uznać to za jedną z głównych przewag badanej struktury. Przyczynę takiej własności można wiązać z wykorzystaną procedurą upraszczania sieci, jest to jednak jedynie hipoteza, która wymaga weryfikacji w czasie przyszłych prac nad zagadnieniem. Zmiany stałych czasowych obiektów regulacji wiązały się z mniejszymi zakłóceniami działania algorytmów i należy uznać, że obie metody poradziły sobie w tym przypadku porównywalnie dobrze.

W ogólności niniejsza praca udowadnia, że zdolność sztucznych sieci neuronowych do generalizacji różnorodnych zagadnień obliczeniowych może z powodzeniem być wykorzystywana w obszarze regulacji. Niewątpliwie ważnym aspektem było wskazanie na korzyści wynikające z zastosowania procedury upraszczania połączeń międzyneuronowych. Praca miała jednak charakter ogólnego i uniwersalnego porównania, z tego względu należałoby w ramach przyszłej pracy skupić się nad uwzględnieniem w analizie wysoce dynamicznych procesów oraz dokładniej zbadać wpływ algorytmu OBD na redukcję wrażliwości sieci pod kątem modyfikacji układów regulacji.



## 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  
**MSE** – Mean Squared Error  
**PW** – Politechnika Warszawska

## Spis rysunków

3.1. Przykładowa odpowiedź 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 .	34
5.2. Zależność kosztu od liczby neuronów w warstwie ukrytej - dane testowe . . .	34
5.3. Zależność kosztu od liczby epok dla sieci z 150 neuronami . . . . .	35
5.4. Zależność kosztu od liczby epok dla sieci z 150 neuronami (ucięte wartości początkowe) . . . . .	36
5.5. Regulacja do wartości zadanej 5,21 - wytrenowana sieć neuronowa . . . . .	37
5.6. Regulacja do wartości zadanej 5,21 - DMC . . . . .	37
5.7. Zależność funkcji kosztu od liczby zredukowanych wag - dane testowe . . . .	39
5.8. Zależność funkcji kosztu od liczby zredukowanych wag - dane treningowe . .	39
5.9. Regulacja do wartości zadanej 2,5 - zredukowana sieć neuronowa . . . . .	42
5.10. Regulacja do wartości zadanej 5,21 - zredukowana sieć neuronowa . . . . .	42
5.11. Regulacja Sieć obiekt ( $T_d = 1$ ) . . . . .	44
5.12. Regulacja DMC obiekt ( $T_d = 1$ ) . . . . .	44
5.13. Regulacja Sieć obiekt ( $T_d = 2$ ) . . . . .	44
5.14. Regulacja DMC obiekt ( $T_d = 2$ ) . . . . .	44
5.15. Regulacja Sieć obiekt ( $T_1 = 7, T_2 = 3$ ) . . . . .	45
5.16. Regulacja DMC obiekt ( $T_1 = 7, T_2 = 3$ ) . . . . .	45
5.17. Regulacja Sieć obiekt ( $T_1 = 4, T_2 = 1$ ) . . . . .	46
5.18. Regulacja DMC obiekt ( $T_1 = 4, T_2 = 1$ ) . . . . .	46

## Spis tabel

5.1. Porównanie MSE dla w pełni wytrenowanej sieci . . . . .	36
5.2. Porównanie MSE dla w pełni wytrenowanej sieci . . . . .	41

5.3. Porównanie MSE dla różnych wartości $T_d$ zmodyfikowanego obiektu . . . . .	43
5.4. Porównanie MSE dla różnych wartości $T_1$ oraz $T_2$ zmodyfikowanego obiektu .	45

## **Spis załączników**