

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA AUTOMATYKI

Praca dyplomowa magisterska

*Algorytm sterowania wykorzystujący sztuczne sieci neuronowe dla
bezzałogowego statku latającego typu TRICOPTER*

Autor:

Rafał Włodarz

Kierunek studiów:

Automatyka i robotyka

Opiekun pracy:

dr hab. Adam Piłat

Kraków, 2015

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczanie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję . . . tu ciąg dalszych podziękowań np. dla promotora, żony, sąsiada itp.

Spis treści

1. Wstęp.....	7
1.1. Cele pracy	7
1.2. Zawartość pracy.....	7
2. Sieci neuronowe.....	9
2.1. Początki sztucznych sieci neuronowych.....	9
2.2. Zastosowanie sieci neuronowych	9
2.3. Sztuczne sieci neuronowe.....	11
2.4. Biologiczne neurony.....	12
2.5. Sztuczny neuron	12
2.6. Nauka sieci	13
2.7. Funkcja aktywacji.....	14
2.7.1. Progowa funkcja aktywacji	14
2.7.2. Liniowa funkcja aktywacji.....	14
2.7.3. Sigmoidalna funkcja aktywacji.....	16
2.8. Perceptron.....	16
2.9. Backpropagation.....	18
3. Architektura statku latającego typu tricopter	21
3.1. Konstrukcja i sterowanie tricopterem.....	21
3.1.1. Przemieszczanie względem osi X	22
3.1.2. Przemieszczanie względem osi Y	22
3.1.3. Przemieszczanie względem osi Z	23
3.2. Budowa modelu	24
4. Aplikacja sterująca	25
4.1. System czasu rzeczywistego.....	25
4.2. Konfiguracja beaglebone black.....	26
4.3. Przygotowanie systemu operacyjnego.....	26
4.4. Architektura systemu sterującego.....	27

4.5. Niezależność aplikacji od platformy sprzętowej	28
4.5.1. Testy	30
5. Testy systemu sterującego	35
5.1. Testy poprawności działania sztucznej sieci neuronowej.....	35
6. Podsumowanie.....	39

1. Wstęp

test

1.1. Cele pracy

1.2. Zawartość pracy

2. Sieci neuronowe

Rozdział ten zawiera informacje na temat historii rozwoju sieci neuronowych, ich architektury, zasady działania oraz algorytmów uczenia.

2.1. Początki sztucznych sieci neuronowych

Początki prac nad poznaniem procesów zachodzących w ludzkim mózgu datuje się na rok 1943. W pracy McCulloch'a oraz Pitts'a przedstawiono matematyczny model neuronu, który zapoczątkował badania związane z tym tematem. W 1949 roku Donald Hebb odkrył, iż informacje przechowywane w sieci neuronowej są reprezentowane jako wartości wag pomiędzy poszczególnymi neuronami. Na podstawie tych informacji zaproponował pierwszy algorytm uczenia sieci neuronowej, który został nazwany regułą Hebba. Już wtedy odkryto, iż bardzo dużą zaletą sieci jest równoległy sposób przetwarzania informacji oraz metodologia uczenia, która zastępuje tradycyjny proces programowania. Pierwszym przykładem realizacji sztucznej sieci neuronowej jest Perceptron. Został on zbudowany w 1957 roku przez Franka Rosenblatta i Charlesa Wightmana. Sieć ta została nauczona rozpoznawania znaków alfanumerycznych. Niestety projekt ten nie okazał się w pełni sukcesem, ponieważ sieć była wrażliwa na zmianę wielkości podawanych znaków oraz nie radziła sobie z bardziej skomplikowanymi. Dzieło to wzbudziło zainteresowanie na całym świecie, spowodowało popularności sieci neuronowych wśród naukowców. Powstało wiele ciekawych prac, które wpływały na rozwój tej dziedziny. Badania te zostały spowolnione przez opublikowanie w 1969 roku książki Minskiego i Paperta noszącej tytuł Perceptrons. Dowiodła ona, że jednowarstwowe sieci mają bardzo ograniczone możliwości. Dopiero od połowy lat 80-tych, zagadniением tym oprócz naukowców zaczęły się interesować firmy komercyjne. Było to spowodowane dwoma głównymi czynnikami: wydaniem książki w 1988 roku przez Jamesa Andresona oraz rozwojem technologii układów mogących modelować coraz bardziej złożone sztuczne sieci neuronowe. Dynamiczny rozwój tej dziedziny trwa do dnia dzisiejszego.

2.2. Zastosowanie sieci neuronowych

Rozwój nauki poświęconej sztuczny siecią neuronowym oraz znaczny postęp techniki umożliwia stosowanie sieci neuronowych w prawie każdej dziedzinie naszego życia. Do najważniejszych z nich można zaliczyć:

- prognozy giełdowe,
- analizę badań medycznych,
- sterowanie systemem zasilania oraz chłodzenia serwerowni,
- optymalizację działalności handlowej,
- kryminalistykę,
- sterowanie procesów przemysłowych,
- przemysł lotniczy.

Ze względu na wszechstronność zastosowania sieci neuronowych oraz ich odpowiedników, służących do rozwiązywania bardziej złożonych zagadnień głębokich sieci neuronowych (ang. Deep Neural Network), stosuje się ich podział oparty na możliwościach, dedykowanych rodzinach problemu, do których zostały przystosowane zamiast dziedzin nauki do jakich są wykorzystywane. W swojej książce prof. Ryszard Tadeusiewicz opisuje główne zastosowania sieci neuronowych [Tad93]:

- Predykcji - ich głównym zadaniem jest przewidywanie określonych sytuacji na podstawie danych wejściowych. Zagadnienie to jest wykorzystywane do prognozy giełdowej, prognozy ekonomicznego rozwoju, prognozy rynkowej, oceny zdolności kredytowej oraz wiele innych. Wprowadzanie sieci neuronowych w takich przypadkach pozwala na przewidywanie poszczególnych wyników bez głębszego rozumienia problemu. Bardzo ważną zaletą jest brak potrzeby określania dokładnych hipotez świadczących o powiązaniu wejścia z wyjściem. Podczas procesu nauczania sieć sama dostosowuje się do zagadnienia na podstawie danych uczących. Często skutkuje to większą efektywnością w rozwiązywaniu problemów takich jak prognozy giełdowe w stosunku do tradycyjnych aplikacji. Powodem tego zjawiska jest to, iż aplikacje do gry na giełdzie bazują na wprowadzonych przez człowieka zależnościach, natomiast sieci neuronowe same dobierają sobie kryterium, według którego reagują na poszczególne wejścia.
- Klasyfikacji i rozpoznawania przedmiotów - zagadnienie to jest bardzo często rozwiązywane przy sztucznych sieci neuronowych. Wykorzystywane są one również w ekonomii, gdzie na podstawie wcześniej określonych etapów przyporządkuje jedne z nich do przedsiębiorstwa na podstawie wejść sieci świadczących o jego stanie. Podczas uczenia sieci wychwytuje ona charakterystyczne cechy poszczególnego wejścia, a następnie przypisuje im konkretne wyjście. Rozwiązywanie tego typu problemów za pomocą sztucznych sieci neuronowych okazuje się bardzo wygodne, ponieważ w przypadku zmiany zagadnienia wymagane jest jedynie powtórzenie procesu nauki bez intererencji w implementacji. Jest to zdecydowanie bardzo duża zaleta w stosunku do tradycyjnych aplikacji, ponieważ nie są one uniwersalne, co powoduje ogromne koszty w przypadku wszelkich modyfikacji. Często kosz poprawek jest zbliżony do kosztów tworzenia nowej aplikacji.

- Kojarzenia danych - bardzo często rozwiązaniem wielu problemu jest szybkie kojarzenie różnych faktów. W porównaniu do tradycyjnych systemów sieci neuronowe, dzięki zdolności uczenia oraz adaptacji, pozwalają na kojarzenie różnorodnych danych. Skutkuje to uzyskaniem przejrzystych informacji na wyjściu sieci bez potrzeby analizy szczegółowych, nadmiarowych danych, z których wyciągnięcie wniosków wymaga żmudnej i długotrwałej analizy.
- Analizy danych - zagadnienie to opiera się na wykryciu związków przyczynowo skutkowych w zbiorze wejściowym. Jest to bardzo istotny problem ze względu na częste fałszywe wnioski wyciągane przez ludzi. Sztuczne sieci neuronowe pozwalają na rzeczywiste wykrycie przyczyny sukcesu jak i niepowodzenia poszczególnych operacji. W oparciu o taki mechanizm, na podstawie przeszłości podejmowanie decyzji staje się zdecydowanie łatwiejsze, jak i również minimalizuje ryzyko popełnienia wcześniejszych błędów.
- Filtracji sygnałów - technika ta jest głównie wykorzystywana w telekomunikacji oraz automatycznej diagnostyce medycznej. Odgrywa ona znaczącą rolę dla rozwoju tych dziedzin. Całe zagadnienie opiera się na usuwaniu zakłóceń pochodzących z różnych źródeł. Są one wykorzystywane do wstępnej obróbki danych, z których eliminowane są nie tylko zakłócenia o charakterze losowym jak również celowe przekłamania. Dodatkowo tradycyjne metody nie mają możliwości uzupełniania niekompletnych danych, co w przypadku sztucznych sieci neuronowych odbywa się ze szczególną łatwością.
- Optymalizacji - jest to prawdopodobnie jedno z najważniejszych zagadnień dla każdej dziedziny nauk technicznych. Problemy te są rozwiązywane przez wykorzystywanie tzw. sieci Hopfielda, która rozwiązuje zagadnienia poszukiwania optymalnych decyzji. Sieci te są stosowane do rozwiązywania zagadnień optymalizacji kombinatorycznej, które są znane z ogromnej złożoności obliczeniowej - niektóre z nich są problemami NP-zupełnymi. Ich największym atutem jest czas, w którym znajdują rozwiążanie. Jest on nieporównywalnie mniejszy dzięki współbieżności obliczeniowej.

Warto zaznaczyć, iż również w przypadku sterowania na bardzo dużą skalę wprowadza się algorytmy oparte na sztucznych sieciach neuronowych. Zagadnienie to jest głównym tematem niniejszej pracy, stąd zastosowanie sieci neuronowej do tych zagadnień zostało opisane w późniejszych rozdziałach.

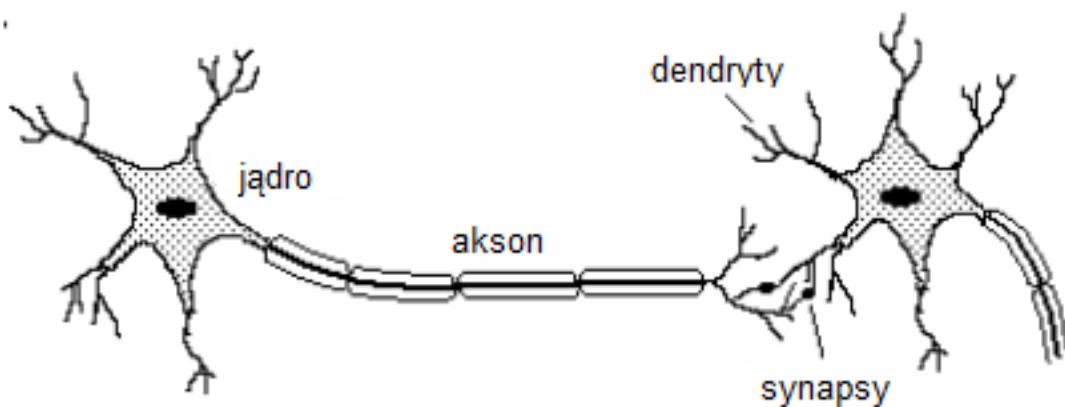
2.3. Sztuczne sieci neuronowe

Ze względu na sposób działania, który przypomina działanie ludzkiego mózgu, sztuczne sieci neuronowe często traktowane są jako początek sztucznej inteligencji. Korzystanie z sieci neuronowych wymaga przejścia przez dwa podstawowe etapy. Pierwszy z nich określany jest jako uczenie sieci, a drugi jako testowanie, które jest używane do oceny poprawności działania sieci po jej wytrenowaniu. Istnieją różne algorytmy uczenia sieci neuronowej, najpopularniejszy z nich nazywany jest perceptronem wielowarstwowy (ang. multilayer perceptron). Polega on na propagacji danych w przód, a następnie przy

użyciu algorytmu wstecznej propagacji, modyfikacji odpowiednich wag neuronów. Sztuczne sieci neuronowe są uznawane jako skuteczna metoda do rozpoznawania wzorców. W ich skład wchodzą połączenia między neuronami jak również same neurony, które równolegle przetwarzają dane wejściowe. Podejście to zostało zaczerpnięte z biologicznego systemu nerwowego.

2.4. Biologiczne neurony

Ludzki mózg składa się z milionów neuronów, które są połączone między sobą przez prawie 10 miliardów synaps. Architektura ta pozwala na równoległe przetwarzanie informacji. Ogólny schemat biologicznego neuronu został przedstawiony na rysunku 2.1.



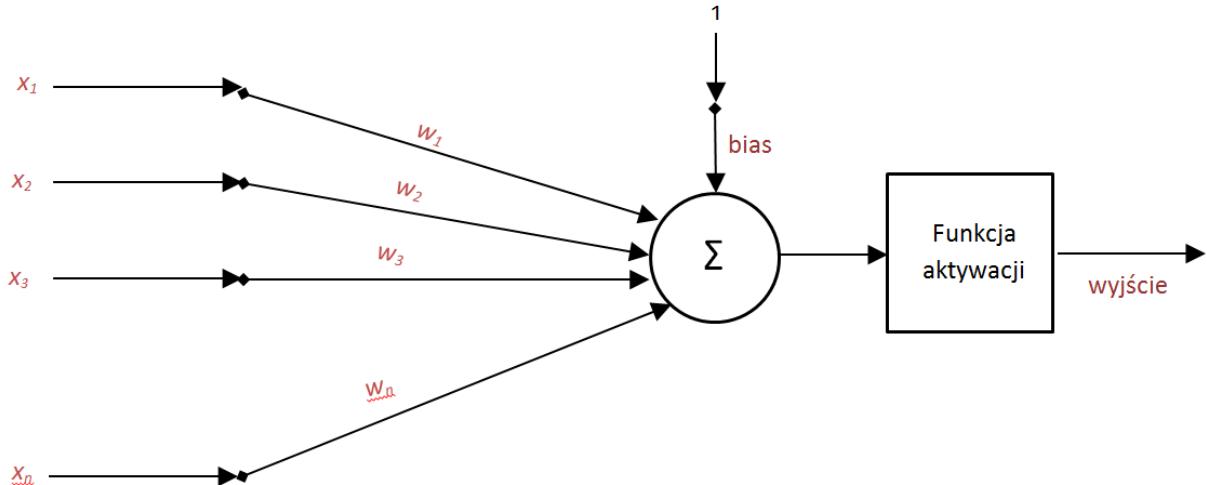
Rysunek 2.1: Biologiczny neuron.

Podstawową funkcją neuronu jest transportowanie przetworzonej informacji w postaci impulsu nerwowego, który jest reprezentowany przez krótkotrwałą zmianę potencjału. Jest on przewodzony od aksonu do synapsy, która znajduje się na jego zakończeniach.

2.5. Sztuczny neuron

Sposób działania sztucznego neuronu jest ściśle oparty na działaniu biologicznego neuronu. Posiada on wiele wejść oraz jedno wyjście. Dla lepszego zrozumienia funkcjonowania takiego neuronu wskażmy jego różnice w stosunku do biologicznego. Współczesne komputery posiadają bardzo dużą moc obliczeniową skupiającą się w pojedynczych procesorach taktowanych wysoką częstotliwością. W przeciwieństwie do nich ludzki mózg posiada miliardy neuronów, które przetwarzają dane wolniej niż współczesne procesory. Informacje przenoszone przez bodźce w ludzkim mózgu są reprezentowane przez logikę binarną o określonym progu aktywacji. W przypadku sztucznych neuronów funkcjonalność ta jest realizowana przez funkcje aktywacji, które na podstawie dobranego progu przypisują wartość logiczną zbliżoną do "1" dla wartości powyżej progu oraz "0" dla pozostałych. W sztucznych sieciach neuronowych po wykonaniu obliczeń w konkretnym neuronie jego wartość wyjściowa przekazywana jest wzdłuż łańcucha

sieci do kolejnych neuronów. W celu lepszego zobrazowania działania sztucznych neuronów często są one porównywane do przekaźników. Synapsy występujące w ludzkim mózgu są odpowiednikami wag dla poszczególnych wejść neuronów.



Rysunek 2.2: Sztuczny neuron.

Jak pokazano na rysunku 2.2 wartości na każdym z wejść są przemnażane przez odpowiadające temu wejściu wagi. Jedno wejście posiada statyczną wartość 1. Zabieg ten umożliwia sieci neuronowej lepiej przystosować się do wzorców. Neuron odpowiada za sumowanie wszystkich wartości. Następnie po obliczeniu sumy ilorazów wartości sygnałów wejściowych z poszczególnymi wagami, podawana jest ona na wejście funkcji aktywacji, której wyjście jest jednoznaczne z końcową wartością uzyskaną po przetworzeniu danego wektora wartości wejściowych.

2.6. Nauka sieci

Nawet przy dotychczasowych osiągnięciach techniki nie możliwe jest w pełni zamodelowanie pracy ludzkiego mózgu. Nie mniej jednak sztuczne sieci neuronowe pozwalają na rozwiązywanie wiele skomplikowanych zagadnień takich jak rozpoznawanie wszelkiego rodzaju wzorców oraz wiele innych. Bardzo ciekawą właściwością sieci neuronowych jest to, iż nawet przy tej samej architekturze po ówczesnym przygotowaniu są w stanie one rozwiązywać całkowicie różne zagadnienia. Takie przystosowanie określone jest jako nauka sieci (ang. learning). Proces ten można porównać do okresu rozwoju noworodka, który na podstawie zdobytych doświadczeń zdobywa nowe umiejętności. Naukę dzieli się głównie na dwie podstawowe kategorie:

- nauczanie z nauczycielem (nadzorowane) (ang. supervised learning) – podejście to wymaga nadzoru, w większości jest to zbiór oczekiwanych wartości odpowiedzi dla konkretnych wejść. Dokonuje się w nim próby przewidzenia wyników dla znanych danych wejściowych. Najbardziej znany algorytm w tej kategorii jest wsteczna propagacja błędów (ang. backpropagation).

lega ona na uczeniu sieci bazując na błędach. Początkowo wagi połączeń między neuronami są wybierane w sposób losowy, następnie na wejście sieci podawany jest wektor wejść ze znanyimi poszczególnymi wartościami, znana jest również wartość oczekiwana dla wyjścia. Jest ona porównywana z aktualnym wyjściem sieci, a następnie na podstawie wielkości błędu obliczana jest wartość korekcji poszczególnych wag każdego z połączeń, tak aby błąd ten został zminimalizowany. Największą wadą algorytmów tego typu jest znajomość dokładnej postaci wektora wyjściowego, która jest ciężka do spełnienia.

- nauczanie bez nauczyciela (nienadzorowane) (ang. unsupervised learning) - podejście to zyskało popularność ze względu na brak konieczności znajomość oczekiwanej wektora danych wyjściowych podczas uczenia sieci. W tym przypadku wyjście sieci nie jest weryfikowane. Do najbardziej znanych algorytmów reprezentujących to podejście zaliczamy sieci Kohonen - samoograniczące się odwzorowania (ang. self-organizing map). Podczas podawania kolejnych danych uczących, to na sieci spoczywa odpowiedzialność za wytworzenie odpowiednich wzorców w zależności od danych wejściowych.

Istnieje możliwość wykorzystania obydwu metod uczenia odpowiednio ze sobą złączonych, które są stosowane oraz dają najlepsze rezultaty dla bardzo złożonych problemów.

2.7. Funkcja aktywacji

W sieciach neuronowych funkcja aktywacji odpowiedzialna jest za przekształcenie sumy powstałej przez dodanie iloczynów poszczególnych wag z odpowiadającymi im sygnałami wejściowymi. Wyróżnia się trzy główne typy takich funkcji: progowa, liniowa, sigmoidalna.

2.7.1. Progowa funkcja aktywacji

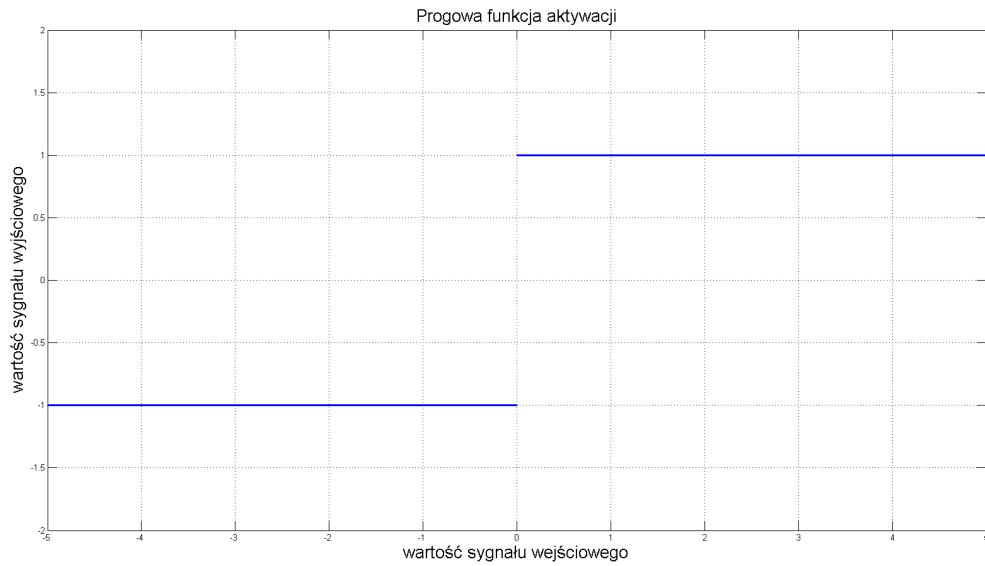
Progowa funkcja aktywacji została przedstawiona przez McCullon'a oraz Pitts'a w 1943 roku. Prosta funkcja tego typu może być określona wzorem:

$$f(x) = \begin{cases} +1 & \text{gdy } x > 0 \\ -1 & \text{gdy } x \leq 0 \end{cases}$$

Funkcja ogranicza się do przypisania zadanej wartości dla wejścia powyżej progu aktywacji. W pozostałych przypadkach neuron otrzymuje stan świadczący o braku jego aktywności. Na rysunku 2.3 przedstawiono przykładowy wykres funkcji progowej.

2.7.2. Liniowa funkcja aktywacji

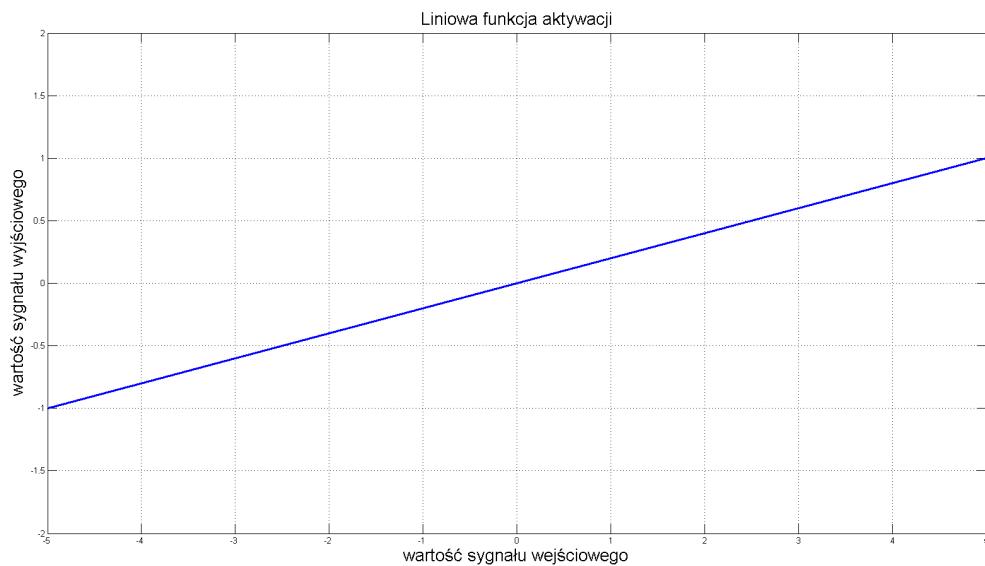
Funkcja liniowa odpowiada za liniowe przekazanie wartości wejściowej na wyjściową po przemnożeniu jej przez odpowiedni współczynnik. Dana jest ona wzorem:



Rysunek 2.3: Przykładowa funkcja progowa.

$$f(x) = a * x$$

Na rysunku 2.4 przedstawiono przykładową funkcję liniową.

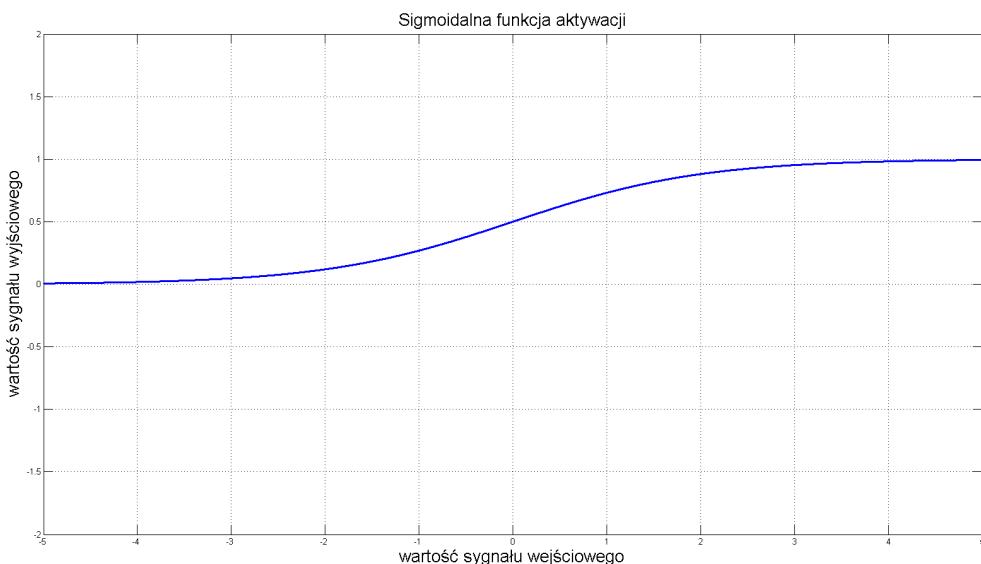


Rysunek 2.4: Przykładowa funkcja liniowa.

2.7.3. Sigmoidalna funkcja aktywacji

Funkcja sigmoidalna jest najczęściej używana w sztucznych sieciach neuronowych ze względu na jej różniczkowalność oraz zachowanie zgodne z głównymi właściwościami sieci neuronowych. Sigmoidalna funkcja unipolarna charakteryzuje się nieliniowym narastaniem w zakresie od 0 do 1. Opisana jest wzorem:

$$f(x) = \frac{1}{1 + e^{-x}}$$



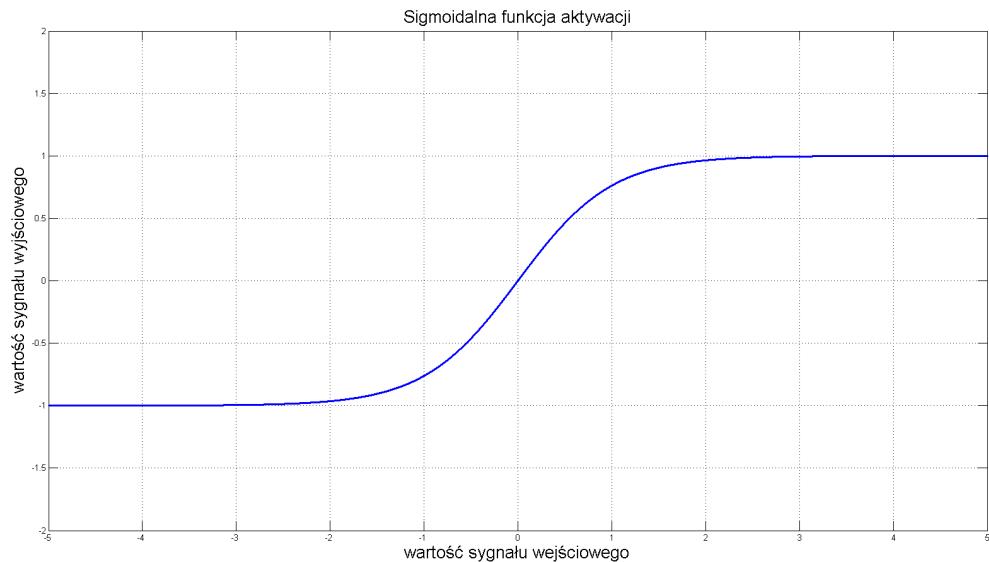
Rysunek 2.5: Wykres funkcji sigmoidalnej.

Bardzo często spotyka się również jej rozszerzoną wersję - bipolarną, przedstawioną na rysunku 2.6, która opisana jest wzorem tangensa hiperbolicznego:

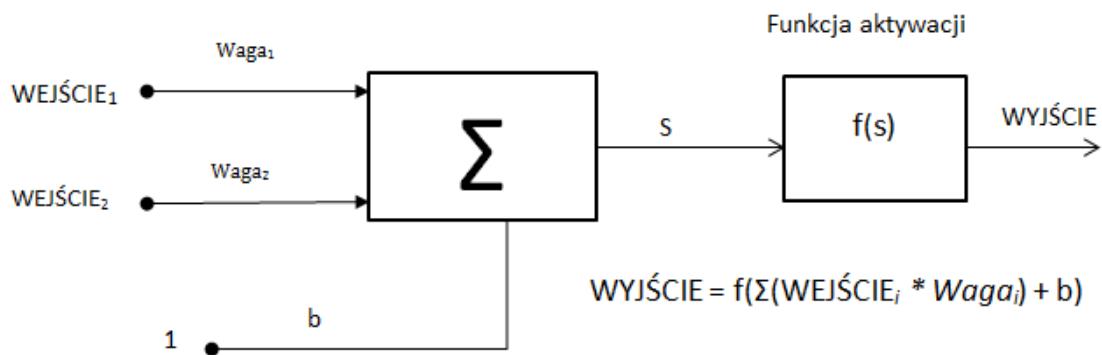
$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

2.8. Perceptron

Perceptronem określa się sieć neuronową najbliższego typu, składającą się z neuronów McCullocha-Pittsa. Ich działanie oparte jest na klasyfikacji sygnału wejściowego przez ustawienie odpowiadającego poziomu wartości wyjściowej. Odbywa się to poprzez równoległe przemnożenie sygnałów wejściowych przez wagi połączeń na których się znajdują. Otrzymane ilorazy są sumowane, następnie stanowią one wejście funkcji aktywacji, której wyjście reprezentuje sygnał wyjściowy całego neuronu. Powyższe operacje zostały przedstawione na rysunku 2.7



Rysunek 2.6: Wykres funkcji tangensa hiperbolicznego.



Rysunek 2.7: Schemat działania perceptronu.

Perceptrony posiadają zdolność klasyfikacji danych na charakterystyczne zbiory, które są liniowo separowalne. Własność ta pozwala tworzyć sieci, w których aktywność poszczególnego neuronu oznacza przynależność do danego zbioru. Cechą ta uniemożliwia również wytrenowanie sieci jednowarstwowej, która rozwiązywałaby problemy nieliniowe. Na przykład, sieć wykonująca operację logiczną xor, która wymaga architektury posiadającej więcej niż jedną warstwę. Ze względu na tą własność pojedyncza warstwa neuronów nie jest szczególnie użyteczna. Dopiero łączenie kilku warstw pozwala na rozwiązywanie złożonych problemów nieliniowych co zdecydowanie zwiększa możliwości sieci neuronowej.

2.9. Backpropagation

Do najbardziej popularnych algorytmów uczenia sieci zaliczany jest algorytm wstecznej propagacji błędu (ang. backpropagation). Głównym założeniem tego procesu jest korekcja wag wszystkich połączzeń w sieci, które wpłyńły na zwiększenie błędu wynikającego z różnicy wartości oczekiwanej i rzeczywistej. W bardzo jasny oraz szczegółowy sposób algorytm został przedstawiony w książce [Tad93].

Wyróżnia się dwie najważniejsze fazy. Pierwszą z nich jest faza propagacji w przód, która podaje na wejście sieci wybrany wektor wejściowy ze zbioru danych uczących oraz odczytuje sygnał wyjściowy. Następną fazę określa się jako propagację wsteczną. Jej działanie rozpoczyna się od ostatniej warstwy sieci, do której podawana jest wartość błędu. Wykonywanie odwrotnych obliczeń wzduż sieci pozwala na wykrycie, które wagi wpłyńły na zwiększenie błędu, a następnie korekcję ich wartości tak, aby zminimalizować błąd. Proces ten został zobrazowany na rysunku 2.8.

Ze względu na istotne znaczenie tego procesu poniżej przedstawiono dokładny opis jego działania przedstawiony na pojedynczym neuronie. Aby przystąpić do fazy nauki sieci należy odpowiednio przygotować zbiór danych uczących. Dane uczące dobierane są w ciąg o budowie opisanej wzorem 2.1.

$$U = \langle\langle X^1, z^1 \rangle, \langle X^2, z^2 \rangle, \dots, \langle X^n, z^n \rangle\rangle \quad (2.1)$$

Jak łatwo zauważyc składa się on z par w postaci $\langle X^j, z^j \rangle$, w których wyróżniamy wektor X z j -tym indeksem odpowiadającym krokowi w procesie uczenia oraz przypisaną do niego informacją o wartości oczekiwanej na wyjściu neuronu. Uwzględniając ówcześnie przedstawioną indeksację zbioru uczącego, regułę uczenia można opisać wzorem:

$$W^{j+1} = W^j + \eta^j \delta^j X^j \quad (2.2)$$

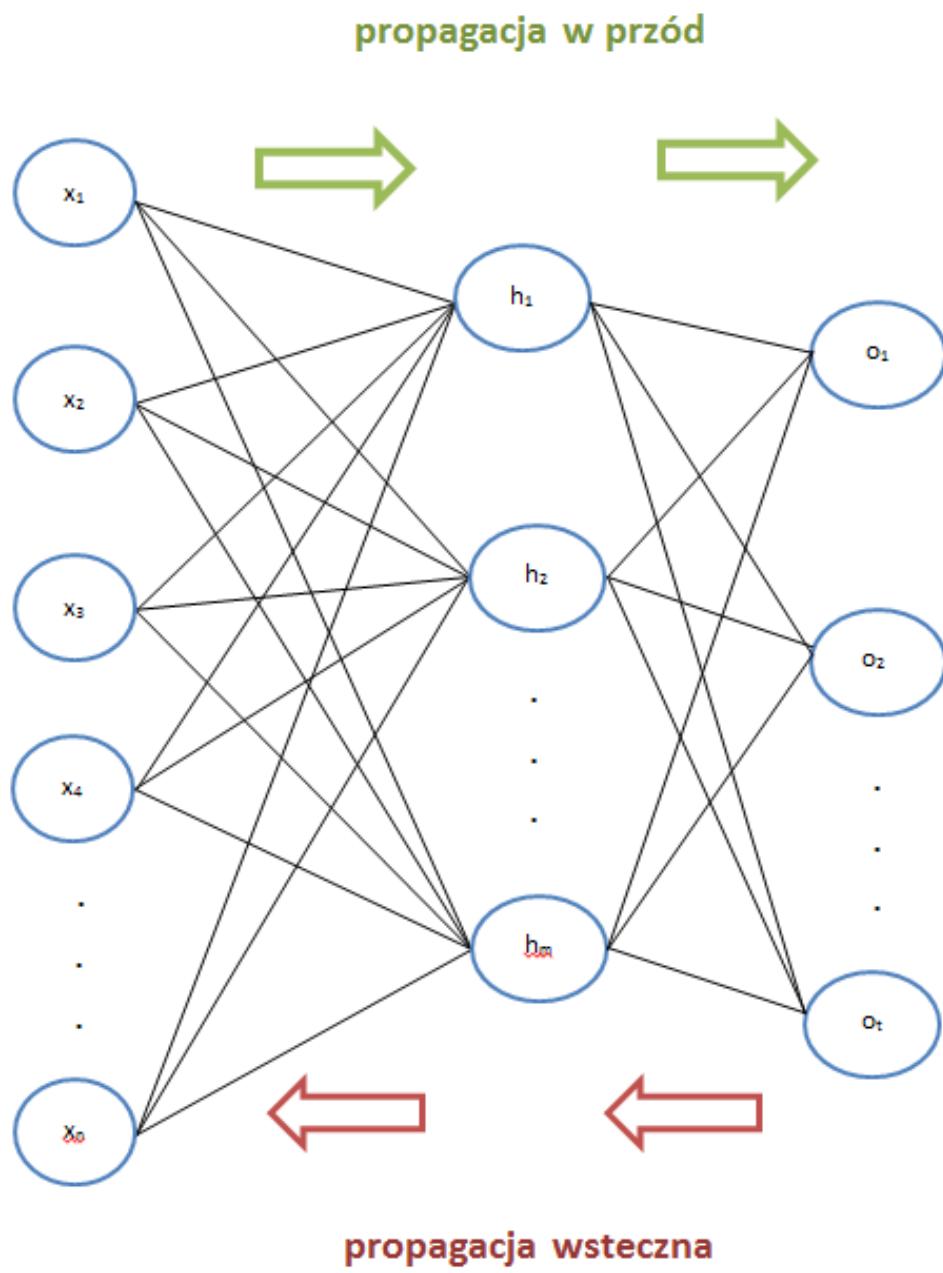
We wzorze 2.2 przyjęto:

$$\delta^j = z^j - y^j \quad (2.3)$$

gdzie:

$$y^j = W^j * X^j \quad (2.4)$$

Reguła 2.2 w pełni opisuje wszystkie konieczne do wykonania obliczenia. Warto jednak mieć na uwadze, iż wektor W^1 musi być określony. Bardzo często jest on inicjowany wartościami wybranymi



x – neurony w warstwie wejściowej

h – neurony w warstwie ukrytej

o – neurony w warstwie wyjściowej

Rysunek 2.8: Przebieg procesu propagacji w przód oraz propagacji wstecznej.

losowo lub wartości te są przenoszone z poprzedniego procesu uczenia nawet jeśli jego zadaniem była realizacja zupełnie innej funkcji. Bez względu na wybrany sposób inicjacji należy mieć na uwadze, aby dobrane wartości były różne $\forall_{\eta,\nu} w_\eta^1 \neq w_\nu^1$. Niespełnienie tego warunku może doprowadzić do braku postępów w początkowych procesie uczenia co wpływa na jego wydłużenie, ewentualnie efekt końcowy.

Bazując na powyższych oznaczenia można zapisać cel procesu uczenia. Celem tym jest doprowadzenie do zgodności wyjścia uczonego neurona y^j z wartością oczekiwana z^j , co jest jednoznaczne ze spełnieniem kryterium 2.5.

$$Q = \frac{1}{2} \sum_{j=1}^N (z^j - y^j)^2 \quad (2.5)$$

Dobrana funkcja Q reprezentuje znaną metodę najmniejszych kwadratów. Wzór ten można ogólnie przedstawić jako

$$Q = \sum_{j=1}^N Q^j \quad (2.6)$$

gdzie:

$$Q^j = \frac{1}{2} (z^j - y^j)^2 \quad (2.7)$$

Warto zauważyć, iż $Q = Q(W)$, zatem poszukiwanie minimum może odbywać się przy użyciu metody gradientowej, i-tą składową wektora W można przedstawić jako:

$$w_i' - w_i = \Delta w_i = -\eta \frac{\delta Q}{\delta w_i} \quad (2.8)$$

Reasumując, proces nauczania sieci opiera się na korekcji wag poszczególnych połączeń między neuronami co powoduje minimalizowanie błędu na wyjściu sieci. Zostaje on zakończony, gdy zostanie wykonana odpowiednia liczba iteracji na zbiorze uczącym lub błęd na wyjściu będzie poniżej zadanego progu. Bardzo często spotyka się również algorytmy uczące, które zaprzestają działania gdy wprowadzane modyfikacje w następnych iteracjach są poniżej określonego progu.

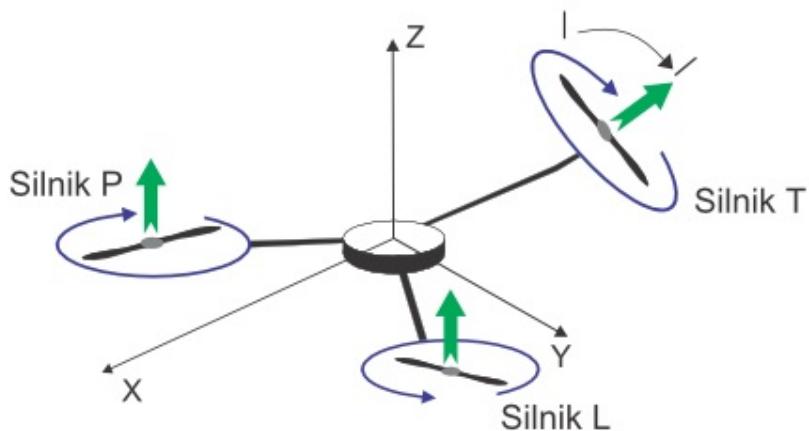
3. Architektura statku latającego typu tricopter

Poniższy rozdział przedstawia zbiór podstawowych zagadnień związanych z konstrukcją wirnikowca typu tricoper oraz zawiera informacje na temat zasad sterowania układem.

3.1. Konstrukcja i sterowanie tricopterem

Ze względu na skomplikowany sposób sterowania tricopterem w przestrzeni oraz złożoną konstrukcję, w poniższym podrozdziale zostaną przedstawione podstawowe zagadnienia związane z tym tematem. Szczegółowy opis mechaniki sterowania wielowirnikowcem o 3 napędach został opisany w [YOWT10].

Na rysunku 3.1 przedstawiono przyjętą konwencję nazewnictwa dla poszczególnych elementów tricoptera odpowiedzialnych za przemieszczanie się układu w przestrzeni. Pokazano również położenie modelu w układzie współrzędnych, które jest niezbędne do przejrzystego opisu zasad sterowania.



Rysunek 3.1: Przyjęta orientacja tricoptera w układzie współrzędnych.

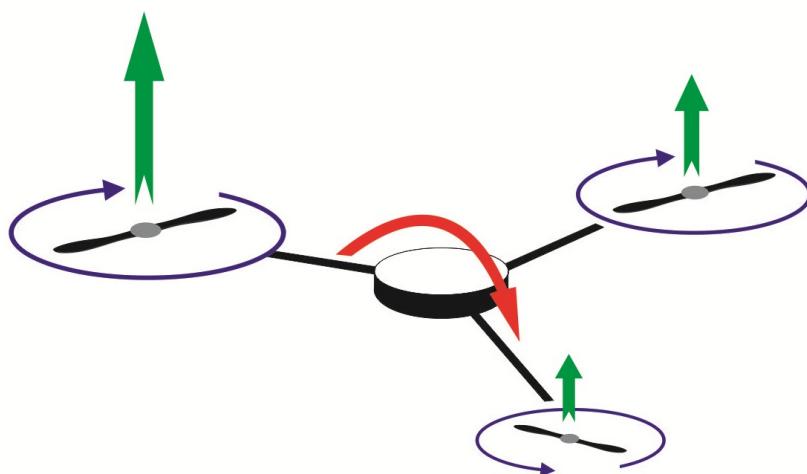
Tricopter należy grupy rzadko spotykanych wielokomórkowców wyposażonych w nieparzystą liczbę silników odpowiadających za napędy w układzie. Konstrukcja ta zdecydowanie utrudnia sterowanie ze względu na brak stabilności tego układu. Spowodowane jest to brakiem kompensacji siły bocznej pochodzącej od napędu niesparowanego. Skutkuje to autorotacją obiektu wokół osi Z przy założeniu, iż

wszystkie wirniki są w pozycji pionowej, tak jak w przypadku większości obiektów latających pionowo startu. Aby wyeliminować powyższy efekt zastosowano odpowiedni moduł składający się z seromechanizmu, który reguluje kąt nachylenia wirnika T.

Przemieszczanie obiektu latającego w przestrzeni wymaga opracowania metodyki sterowania poszczególnymi wirnikami podczas przemieszczania się względem każdej z osi. Bazując na poprzednim algorytmie sterującym poniżej zaprezentowano sposoby przemieszczania względem konkretnych osi tricoptera. Szczegółowe informacje związane z powyższymi zagadnieniami zostały opisane w pracy inżynierskiej autora w rozdziale 5 [ZW13].

3.1.1. Przemieszczanie względem osi X

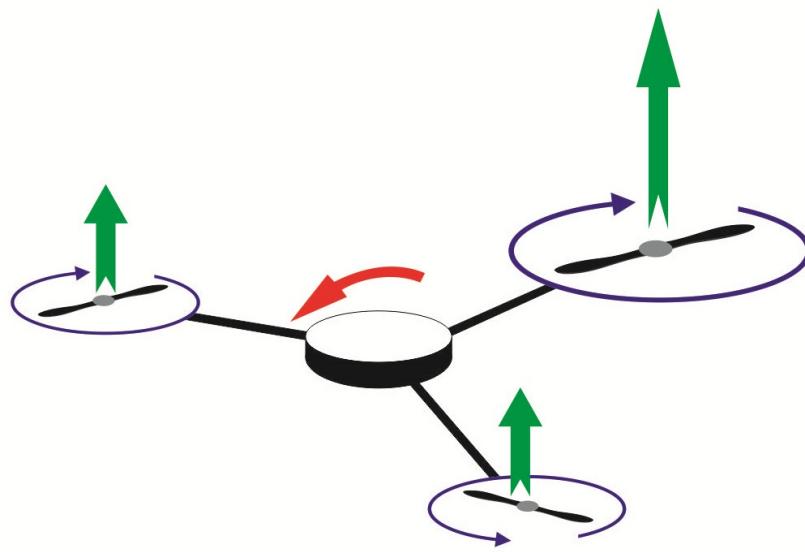
Aby dokonać obrotu względem osi X konieczna jest ingerencja w ciąg wirników L oraz P. Aby uzyskać efekt przechylenia w lewą stronę konieczna jest redukcja siły ciągu na silniku L. Opcjonalnie w celu przyśpieszenia manewru można proporcjonalnie zwiększyć ciąg silnika P. Na rysunku 3.2 przedstawiono opisany manewr.



Rysunek 3.2: Sterowanie oraz stabilizacja w osi X.

3.1.2. Przemieszczanie względem osi Y

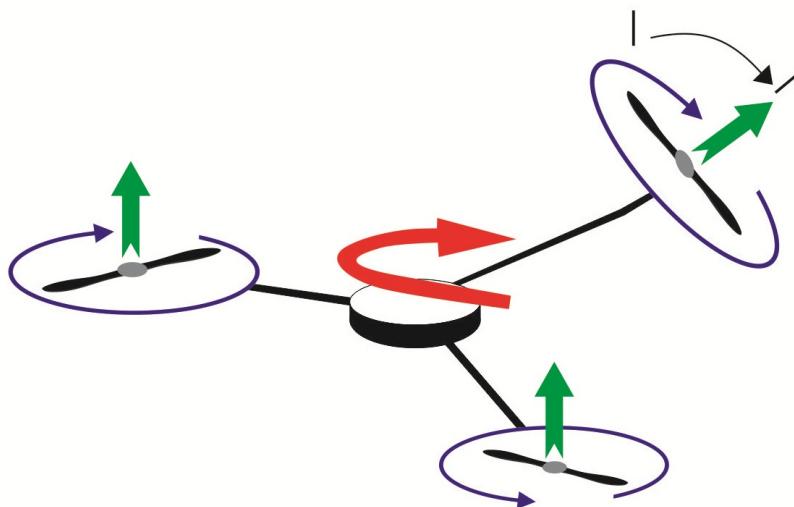
Obrót względem osi Y można zdecydowanie określić jako najważniejszy dla każdego obiektu latającego. Odpowiada on bowiem za przemieszczenie się w przód. Odchylenie tricoptera zapewnia się przez regulację ciągu na wirniku T. Tak jak w przypadku osi X można zwiększyć prędkość obrotu obiektu przez symetryczną modyfikacją mocy silników L oraz P wartość przeciwną co do znaku względem modyfikacji wirnika Z. Manewr ten został przedstawiony na rysunku 3.3.



Rysunek 3.3: Sterowanie oraz stabilizacja w osi Y.

3.1.3. Przemieszczanie względem osi Z

Przemieszczanie względem osi Z jest głównie używane do zmiany kierunku lotu tricoptera. Manewr ten można wykonywać bez ingerencji w moc każdego z wirników. Zmiana kąta nachylenia silnika T powoduje pojawienie się efektu rotacji obiektu. Warto zwrócić uwagę, iż zbyt duże odchylenie wirnika od punktu stabilnego spowoduje również ruch tricoptera względem osi Y, ponieważ zmiana kąta nachylenia silnika Z powoduje zmianę pionowej siły ciągu, która odpowiada również za stabilizację tego obiektu.



Rysunek 3.4: Sterowanie oraz stabilizacja w osi Z.

3.2. Budowa modelu

Ze względu na to, iż temat tej pracy bazuje na wykorzystaniu do testów gotowego obiektu latającego typu tricopter, autor zdecydował się pominąć szczegółowy opis budowy modelu. Na obrazku 3.5 przedstawiono model wykorzystanego wielowirnikowca.



Rysunek 3.5: Model tricoptera.

Szczegółowe informacje dotyczące konstrukcji oraz parametrów konkretnego modelu, wykorzystanego do realizacji tej pracy zostały przedstawione w pracy inżynierskiej autora w rozdziale 2 [ZW13].

4. Aplikacja sterująca

W niniejszym rozdziale przedstawiono informacje na temat aplikacji sterującej. Opisano zagadnienia teoretyczne konieczne do zrozumienia problemu, poszczególne etapy przygotowania platformy sprzętowej oraz proces tworzenia systemu czasu rzeczywistego.

4.1. System czasu rzeczywistego

System czasu rzeczywistego (ang. real-time system) to system, który przetwarza każdy rodzaj informacji i który musi reagować na sygnały wejściowe - bodźce generowane z zewnątrz w skończonym i określonym czasie. Jego poprawne działanie zależy zarówno od prawidłowych rezultatów logicznych, jak również od czasu reakcji. Na podstawie tych kryteriów są one dzielone na:

- Systemy o ostrzych wymaganiach czasowych (ang. hard real-time) - wymagania czasowe muszą być skrupulatnie przestrzegane, naruszenie ram czasowych może wpływać na życie ludzkie, środowisko czy też sam system;
- Systemy o słabych wymaganiach czasowych (ang. soft real-time) – głównym kryterium oceny tych systemów jest średni czas odpowiedzi. Sporadyczne opóźnienie nie powoduje zagrożenia, lecz jedynie wpływa negatywnie na ocenę całego systemu;
- Systemy o solidnych wymaganiach czasowych (ang. firm real-time) - są one kombinacją systemów o wymaganiach ostrych oraz słabych. Naruszenie kryterium czasowych może pojawiać się okazjonalnie. Często dla lepszej oceny systemu stosuje się ograniczenia czasowe o charakterze słabym- krótsze, których przekroczenie nie powoduje katastrofy oraz óstrym- dłuższe, których naruszenie oznacza nieprawidłowe działanie systemu.

Bez względu na to do której grupy systemów zalicza się aplikację musi ona charakteryzować się następującymi cechami:

- Ciągłość działania - powinna działać nieprzerwanie w okresie od uruchomienia systemu do jego wycofania;
- Zależność od otoczenia - zachowanie opiera i rozpatruje w kontekście otoczenia. Prowadzone obliczenia zależą od zdarzeń oraz danych pochodzących z zewnątrz układu;

- Współbieżność - struktura systemu narzuca, aby jednocześnie zdarzenia były obsługiwane równocześnie przez szereg procesów;
- Przewidywalność - zdarzenia i dane generowane przez otoczenie pojawiają się przypadkowo co nie narusza deterministycznego zachowania systemu;
- Punktualność - odpowiedź systemu na bodźce zewnętrzne powinna być dostarczona w odpowiednich momentach - wymaganych ramach czasowych.

Z pewnością aplikacja sterująca obiektami latającymi powinna spełniać wszystkie powyższe założenia. Gdyby któreś z nich nie zostało spełnione, jakiekolwiek próby sterowania zakończyły by się porażką.

Dynamika wielokomórkowców wymaga od aplikacji bardzo szybkiego czasu reakcji na zewnętrzne impulsy. Opóźnienie sterowania w takim przypadku powoduje bardzo negatywne skutki, do których zaliczamy: brak kontroli nad obiektem, utrata stabilności w powietrzu oraz niekontrolowane zetknięcie się z przeszkodą. Wszystkie wymienione zachowania mogą powodować bardzo duże zniszczenia dla modelu, jego otoczenia oraz zagrażać zdrowiu operatora oraz innych osób znajdujących się w zasięgu działania modelu.

Na podstawie definicji oraz powyższej analizy skutków określono, iż aplikacja sterującą zalicza się do systemu hard-real time.

4.2. Konfiguracja beaglebone black

Ze względu na kontynuację prac nad modelem, bazowa konfiguracja beaglebone black wraz z dodatkowymi czujnikami oraz urządzeniami peryferyjnymi została opisana w pracy [ZW13].

W trakcie powstawania niniejszej pracy na rynku dostępne były nowsze czujniki, które mogłyby podnieść jakość orazczęstość pomiarów, jednak ingerencja w konfigurację uniemożliwiłaby porównanie algorytmów tradycyjnych z sieciami neuronowymi stąd wszystkie testy zostały przeprowadzone na tej samej konfiguracji.

4.3. Przygotowanie systemu operacyjnego

Ze względu na wcześniej wspominaną specyfikę systemu sterującego oraz zastosowanie platformy sprzętowej typu mini PC wraz z systemem operacyjnym typu UNIX, ważne jest, aby wyeliminować wszelkie możliwe przerwania procesora oraz inne aspekty, które wpływają na płynność oraz czas wykonywania się aplikacji sterującej.

Wprowadzono usprawnienie w postaci instalacji systemu operacyjnego wyposażonego w jądro czasu rzeczywistego (ang. real-time kernel). Jądro to określone jest również jako w pełni wywłaszczone. Oznacza to, iż dopuszczane jest wywłaszczenie procesu działającego w trybie jądra. Cechą ta wraz z wcześniejszymi narzuconymi priorytetami dla poszczególnych procesów gwarantuje, iż aplikacja sterująca, uruchomiona z wysokim priorytetem nie zostanie wywłaszczena na zbyt długi czas przez inne procesy systemowe.

Zabieg ten pozwala nam spełnić podstawową cechę systemów czasu rzeczywistego, którą jest punktualność.

4.4. Architektura systemu sterującego

Aplikacja sterująca ze względu na zastosowanie algorytmu opartego na sieciach neuronowych nie posiada dedykowanych rozwiązań pod konkretne zagadnienie. Jej budowa opiera się na tworzeniu sztucznej sieci neuronowej oraz dostarczeniu niezbędnych do jej działania funkcji - wczytywania wag dla poszczególnych połączeń oraz algorytmu uczącego. Dodatkowo została ona rozbudowana o system logowania danych, który okazał się bardzo pomocny w wykrywaniu nieprawidłowości w działaniu aplikacji oraz dostarczaniu danych niezbędnych do głębszej analizy zagadnienia.

W celu dokładnego zrozumienia architektury oraz szczegółów związanych z implementacją poniżej przedstawiono szczegółowy opis poszczególnych elementów aplikacji:

- klasa Neuron - jest to najważniejszy element w aplikacji. Jest on odpowiedzialny za przekazywanie odpowiednio przetworzonego sygnału z jego wejściem na wyjście. Dodatkowo znajdują się w nim konieczne do realizacji algorytmu propagacji wstecznej informacje oraz funkcje. Posiada dostęp do wartości współczynników η oraz α , które są niezbędne do parametryzowania procesu uczenia. Został on również wyposażony w indeks, który odpowiada jego miejscu w warstwie. Nie jest to informacja konieczna do poprawnego działania sieci jednak zdecydowanie ułatwia podgląd przepływu informacji przez sieć neuronową.
- klasa Net - odpowiada za pełną reprezentację sieci. Przechowuje w wektorze wszystkie utworzone neurony w uszeregowanych warstwach. W swoim konstruktorze przyjmuje wektor liczb, które odpowiadają za liczbę tworzonych warstw oraz ilości neuronów w każdej z nich. W zależności od konfiguracji inicjowane są wagi wszystkich połączeń. Są one dobierane losowo lub za pomocą wcześniejszych zdefiniowanych wartości. Wywołanie funkcji feedForward z argumentem w postaci wektora danych wejściowych powoduje przepływ i przetworzenie informacji przez sieć. Wektor wartości wyjściowej odczytujemy za pomocą funkcji getResults. Dodatkowo klasa ta została wyposażona w funkcję backProp, która odpowiada za uczenie sieci metodą propagacji wstecznej.
- kontener Layer - dla lepszego zobrazowania struktury aplikacji neurony reprezentujące poszczególną warstwę są przechowywane w wektorze.
- struktura Connection - odpowiada za przejrzystą reprezentację synaps w układzie biologicznym. Przetrzymuje ona informacje na temat wagi połączenia, które reprezentuje oraz wartość modyfikacji z procesu uczenia.
- klasa Logger - ze względu na ostre wymagania czasowe utworzono asynchroniczny logger z konfigurowalnymi poziomami logowania. Jego użycie wymaga wywołania funkcji addMsg, do której podajemy argument świadczący o priorytecie informacji ERROR, WARNING, INFORMATION,

DEBUG oraz jej treść jako następny argument. Wiadomość ta jest dodawana do kolejki, która jest cyklicznie wypisywana do pliku w niezależnym wątku. Zabieg ten pozwala na zabezpieczenie przed ewentualnym przyblokowaniem głównego wątku sterującego w przypadku problemów z dostępem do pliku lub innych nieprawidłowości. Istnieje również możliwość konfiguracji poziomu logowania. Jest to znacząca cecha w przypadku systemów czasu rzeczywistego. Podczas testów warto posiadać wszystkie informacje na temat działania systemu jednak po ich ukończeniu są one zbędne w przypadku normalnego użytkowania, kiedy istotne są jedynie te informacje, które świadczą o błędach w systemie. Nadmiar logowanych danych wpływa negatywnie na czas wykonywania się pętli sterowania a przechowywane dane zajmują sporo pamięci w przypadku systemów wbudowanych. Całość została obudowana we wzorzec projektowy Singleton, aby ułatwić dostęp do klasy oraz zabezpieczyć kod przed utworzeniem wielu instancji tego obiektu.

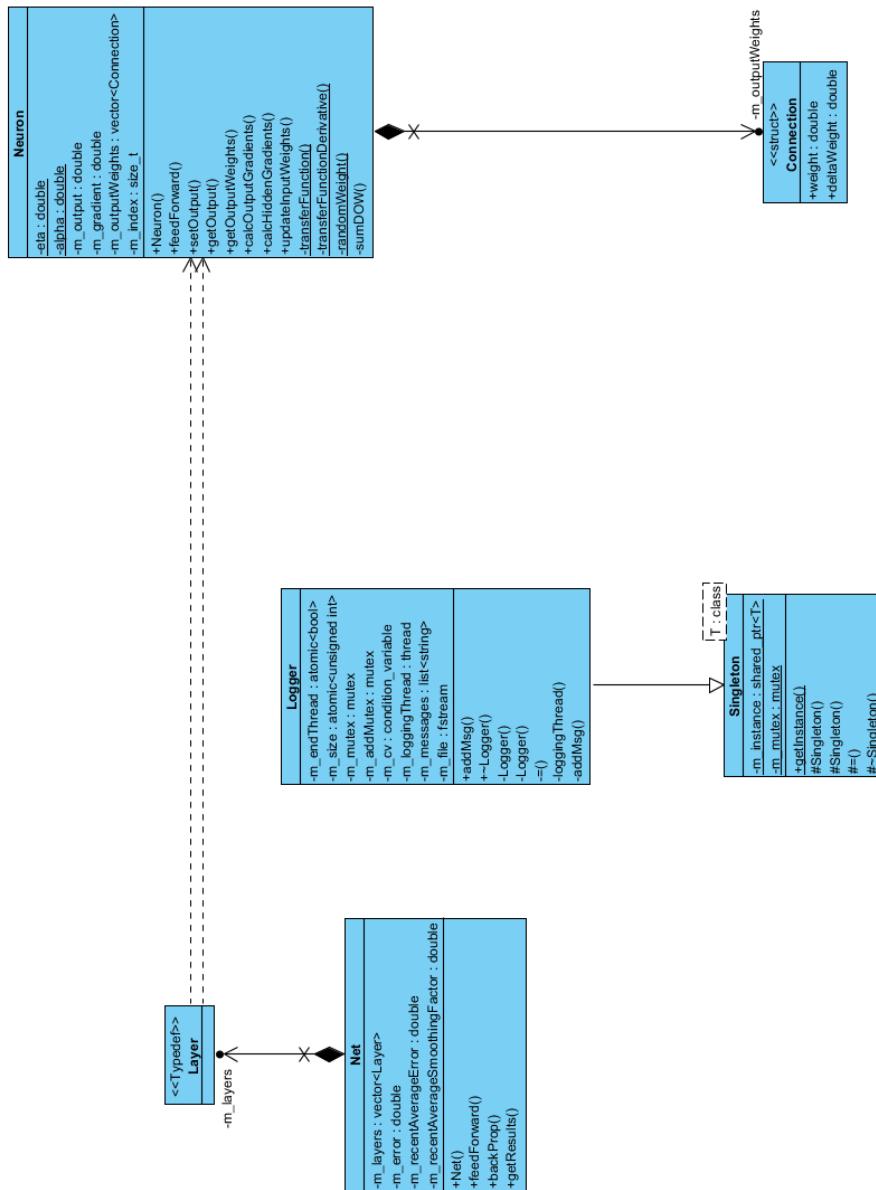
Szczegółowy diagram klas został przedstawiony na rysunku 4.1.

4.5. Niezależność aplikacji od platformy sprzętowej

Dobrze zaprojektowana aplikacja sterująca obiekttami latającymi powinna mieć możliwość uruchomienia na różnorodnych platformach sprzętowych, bez względu na architekturę wykorzystanych procesorów. Ponieważ zagadnienie to jest obszerne, autor zdecydował się wyszczególnić jedynie najważniejsze aspekty związane z przenoszeniem kodu napisanego w języku C++ pomiędzy platformami.

Na podstawie analizy dokonanej przez autora wybrano najważniejsze zagadnienia, które powodują różne działania aplikacji w zależności od systemu:

- przechowywanie wartości typu integer pod typem double - w systemach 32 bitowych nie wystąpił problem reprezentacji liczb całkowitych przez typ double. Standard języka C++ definiuje, iż zmienna ta posiada do 52 bitów zarezerwowanych na reprezentację takiej liczby. Warto jednak mieć na uwadze, iż w przypadku systemów 64 bitowych liczby całkowite przetrzymywane w typie size_t posiadają aż 64 bity przeznaczone do reprezentacji jej zawartości natomiast liczba bitów w przypadku typu double się nie zmienia. Problem ten prowadzi do niepoprawnej reprezentacji wartości całkowitych zajmujących ponad 52 bity przypisywane do typu double. Wartość ta zostanie obcięta o początkowe 12 bitów. Zaleca się unikanie takich przypisań.
- operacje przesunięć bitowych - aplikacje dedykowane na systemy wbudowane często wykorzystują niskopoziomowe operacje takie jak przesunięcia bitowe. Przesunięcie bitowe liczby 0x00000001 o 32 miejsca w lewą stronę powoduje różne wyniki z zależnością od architektury procesora. W przypadku systemu x86 otrzymamy wartość 0x00000000, a system x64 wróci 0x0000000010000000. W przypadku odczytu takich wartości może to zdecydowanie wpływać na działanie aplikacji. Problem ten często rozwiązywany jest za pomocą definiowania maksymalnego przesunięcia w pliku nagłówkowym - o 32 lub 64 bity w zależności od systemu.



Rysunek 4.1: Diagram klas aplikacji sterującej.

- operacje mnożenia liczb zmiennoprzecinkowych - operacja ta jest bardzo znanym zagadnieniem dającym odmienne wyniki w zależności od ilości bitów przewidzianych na poszczególną z liczb oraz wynik, do którego przypisywany jest iloczyn. Ma to związek z uzyskaniem błędu odcięcia w przypadku nie wystarczającej liczby bitów potrzebnych do reprezentacji części ułamkowej.

Na podstawie powyższych przykładów można zauważać, iż możliwe jest stworzenie aplikacji, której kod można przenosić pomiędzy różnymi architekturami. Nie mniej jednak warto zauważać, że wymaga to sporo wiedzy na temat wykorzystywanych narzędzi oraz umiejętności wykonywania wszelakich operacji na liczbach. Podejście to pozwala z wyeliminować błędy związane z dwoma pierwszymi zagadnieniami. Trzeci problem wymaga jednak sprzętowego lub programowego zabezpieczenia przed tego typu zagrożeniami. Ze względu na ostatnie zagadnienie do implementacji sieci neuronowej wybrano język C++. Posiada on zdefiniowany standard, który gwarantuje, iż bez względu na system operacje zmiennoprzecinkowe na typach double gwarantują uzyskanie tego samego wyniku. Jest to bardzo ważne cecha, która zaważyła na tym, iż omawiana aplikacja została stworzona właśnie w tym języku. Warto mieć jednak na uwadze, iż operacje pomiędzy różnymi typami np. iloczyn float oraz double mogą powodować komplikacje stąd należy ich unikać.

Podstawowa wersja aplikacji została poddana testom, które przez porównanie wyników działania sieci neuronowej na 2 całkowicie odmiennych platformach stanowią informację o możliwości stosowania implementacji na różnych systemach.

4.5.1. Testy

Do testów wykorzystano komputer stacjonarny z 64 bitowym procesorem Intel i5-4670 3.40 Ghz oraz układ beaglebone black, który został wyposażony w procesor 32 bitowy ARM Cortex-A8 o częstotliwości taktowania 1GHz.

W celu uwydatnienia problemu stworzono sieć neuronową posiadającą jedną warstwę wejściową, ukrytą, wyjściową. Wagi poszczególnych neuronów zostały dobrane w sposób losowy. Konieczny był jednak dobór liczb, które maksymalnie wykorzystują precyzje liczby zmiennoprzecinkowej typu double. Następnie na wejście sieci podawana jest stała wartość oraz odczytywane jest wyjście sieci. Kroki te powtarzane są na dostępnych platformach sprzętowych. Wyniki poszczególnych operacji są logowane, a następnie porównywane z ich odpowiednikami na innej platformie sprzętowej w celu zbadania rozbieżności.

Poniżej przedstawiono ogólny szkic testowanej topologii sieci z uwzględnieniem wag połączeń.

W tabeli 4.1 zestawiono wszystkie wagi połączeń ustalone dla sieci neuronowej, na której przeprowadzono testy. Warto zauważać, iż odbiegają one od wartości przypisanych ze względu na niewystarczającą ilość bitów potrzebnych do reprezentacji wymaganej precyzji. Zachowanie to jest klasycznym przykładem błędu odcięcia.

Tabela 4.2 przedstawia wartości wyjściowe poszczególnych neuronów wchodzących w skład analizowanej sieci. Zawiera ona zestawienie odpowiedzi każdego z nich w zależności od architektury procesora

Tablica 4.1: Wartości poszczególnych wag podłączeń między neuronami.

	Przypisana wartość	Wartość rzeczywista w aplikacji
$w_{1,2}$	0.39438292681909303816212286619702354073524475097656432	0.3943829268190930381621228661970235407352447509765625
$w_{1,3}$	0.79844003347607328535673322259970009326934814453175654	0.79844003347607328535673322259970009326934814453125
$w_{1,4}$	0.1975513692933839604570778192282887175679206848144531654654	0.197551369293383960457077819228288717567920684814453125
$w_{1,5}$	0.76822959481190400410355323401745408773422241210937999	0.768229594811904004103553234017454087734222412109375
$w_{B,2}$	0.5539699557954305131346472990117035806179046630859371325	0.5539699557954305131346472990117035806179046630859375
$w_{B,3}$	0.628870924761924410262281526229344308376312255859374132	0.628870924761924410262281526229344308376312255859375
$w_{B,4}$	0.513400910195615511888433957210509106516838073730468878238	0.51340091019561551188843395721050910651683807373046875
$w_{B,5}$	0.916195068003700652248255664744647219771072387695312111	0.9161950680037006522482556647446472197710723876953125
$w_{2,6}$	0.71729692943268308358284457426634617149829864501953142135	0.71729692943268308358284457426634617149829864501953125
$w_{3,6}$	0.606968876257058642664787839748896658420562744140628786	0.606968876257058642664787839748896658420562744140625
$w_{4,6}$	0.242886770629736958859723472414771094918251037597698768	0.24288677062973695885972347241477109491825103759765625
$w_{5,6}$	0.80417675422699042009355707705253735184669494628906243432	0.8041767542269904200935570770525373518466949462890625
$w_{B,6}$	0.4009443942461834997637026845040963962674140930175781541	0.400944394246183499763702684504096396267414093017578125

na którym została wykonana iteracja algorytmu.

Na podstawie powyższych badań udało się udowodnić, iż przestrzegając wszystkich omawianych w tym rozdziale aspektów można stworzyć aplikację implementującą sieć neuronową, której działanie nie jest uzależnione od architektury procesora oraz systemu operacyjnego.

Tablica 4.2: Wyjścia poszczególnych neuronów.

	Wartość na wyjściu w systemie x32	Wartość na wyjściu w systemie x64
N_1	0.84018771715470952354820610707974992692470550537109375	0.84018771715470952354820610707974992692470550537109375
w_2	0.70907729414584264038268202057224698364734649658203125	0.70907729414584264038268202057224698364734649658203125
w_3	0.86164859675065408328720195640926249325275421142578125	0.86164859675065408328720195640926249325275421142578125
w_4	0.59111692701338591771076380609883926808834075927734375	0.59111692701338591771076380609883926808834075927734375
w_5	0.915687707062736944152447904343716800212860107421875	0.915687707062736944152447904343716800212860107421875
w_6	0.98058329662773291435229339185752905905246734619140625	0.98058329662773291435229339185752905905246734619140625

5. Testy systemu sterującego

W tym rozdziale zostały omówione badania sieci neuronowej. Ich głównym celem jest sprawdzenie poprawności jej działania, określenie optymalnej topologii sieci oraz układu połączeń między neuronami.

5.1. Testy poprawności działania sztucznej sieci neuronowej

Ze względu na nietrywialną implementację zagadnień związanych z sieciami neuronowymi oraz algorytmów uczących, sieć została przetestowana pod kątem poprawności działania na zdecydowanie prostszym zagadnieniu jakim jest operacja logiczna typu XOR.

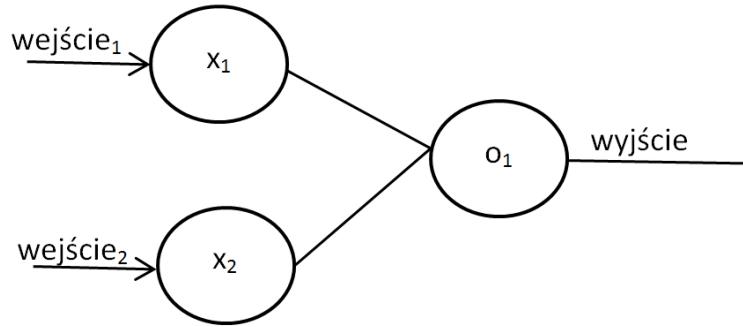
Została ona wybrana ze względu na łatwość zrozumienia problemu, bezproblemowe określenie oczekiwanej wyniku oraz brak wymaganego zbioru uczącego - wektor wejściowy może on zostać wygenerowany podczas samego procesu uczenia, natomiast wartość oczekiwana zostaje obliczona przy pomocy prostej operacji logicznej.

Na podstawie 5.1 łatwo zauważyc, iż operacja te jest nieliniowa. Cechą ta powoduje, iż pojedyńczy do realizacji tego problemu wymagana jest sieć wielowarstwowa - z przynajmniej jedną warstwą ukrytą. Do rozwiązania tego zagadnienia zastosowano dipolarną funkcję sigmoidalną - tangens hiperboliczny. Działanie sieci neuronowej zostało sprawdzone w dwóch przypadkach. Pierwszy z nich dla niewystarczającej ilości neuronów potrzebnych do realizacji tego zagadnienia. Następnie drugi przypadek wykorzystuje odpowiednią architekturę sieci potrzebną do realizacji zagadnienia, aby wykazać poprawność działania implementacji.

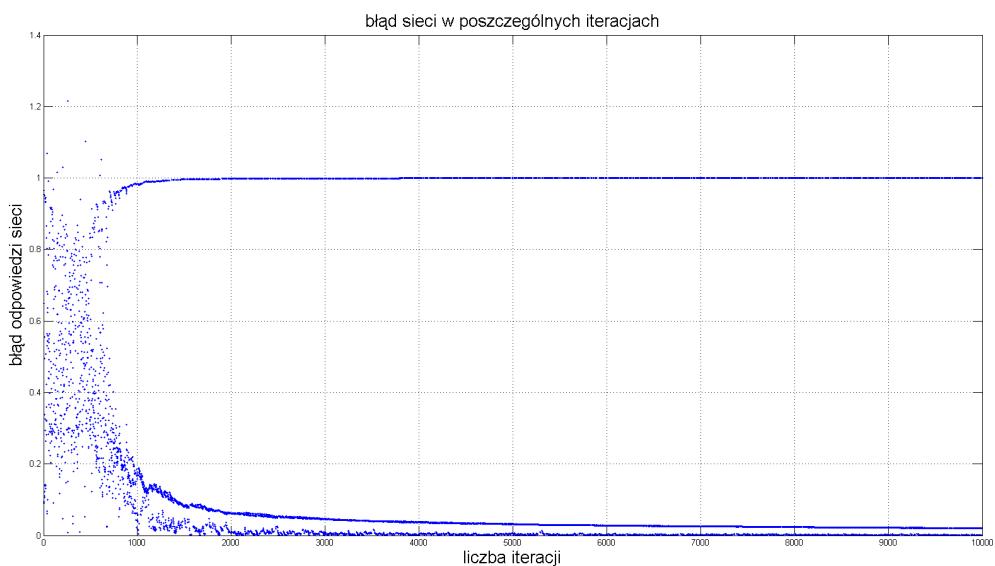
Na wykresie 5.2 przedstawiono proces uczenia sieci. Zawiera on informacje o błędach odpowiedzi sieci w poszczególnych iteracjach algorytmu uczenia.

Wejście 1	Wejście 2	Wyjście
0	0	0
0	1	1
1	0	1
1	1	0

Tablica 5.1: Oczekiwane wartości odpowiedzi sieci realizującej xor.



Rysunek 5.1: Przepływ informacji w topologii 2-1.



Rysunek 5.2: Wykres błędu w kolejnych iteracjach dla sieci o topologii 2-1.

Na podstawie powyższego zestawienia: wykresu 5.2, oraz tabeli 5.1 można uznać, iż weryfikacja potwierdziła wcześniejszą tezę teoretyczną, która zakłada, iż sieć neuronowa nie posiadająca ukrytej warstwy nie jest w stanie rozwiązać zagadnień nieliczniowych. Jednocześnie potwierdza to, zgodność implementacji z teorią dotyczącą sieci neuronowych.

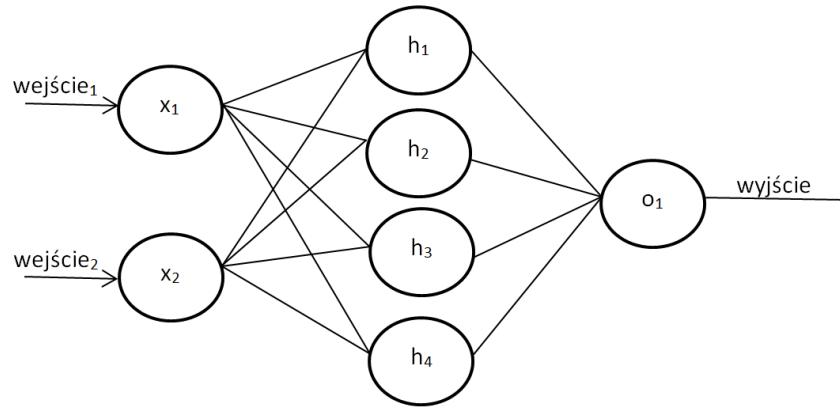
Następnie przeprowadzono test sieci w topologii przedstawionej na rysunku 5.4.

W stosunku do poprzedniego testu, architektura sztucznej sieci neuronowej została rozbudowana o dodatkową warstwę ukrytą. Pozostałe cechy sieci takie jak inicjalizacja wag neuronów oraz funkcja aktywacji pozostały bez zmian.

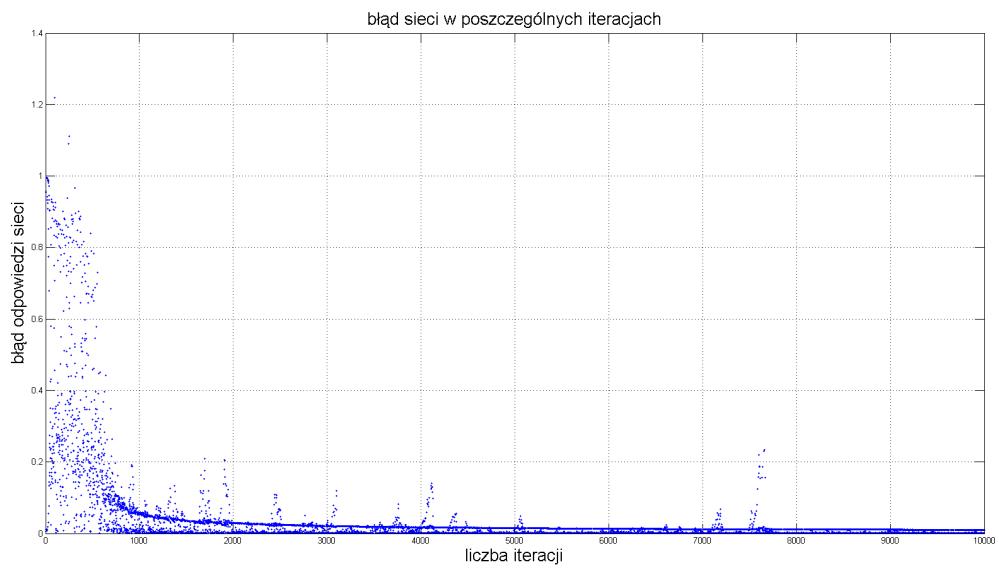
5.2. Testy działania algorytmu sterowania opartego na sztucznej sieci neuronowej

Wejście 1	Wejście 2	Wyjście
0	0	0
0	1	1
1	0	1
1	1	0

Tablica 5.2: Odpowiedzi sieci neuronowej o topologii 2-1.



Rysunek 5.3: Przepływ informacji w topologii 2-4-1.



Rysunek 5.4: Wykres błędu w kolejnych iteracjach dla sieci o topologii 2-4-1.

Wejście 1	Wejście 2	Wyjście
0	0	0
0	1	1
1	0	1
1	1	0

Tablica 5.3: Odpowiedzi sieci neuronowej o topologii 2-4-1.

6. Podsumowanie

Bibliografia

- [Tad93] R. Tadeusiewicz. *Sieci Neuronowe*. Akademia Oficyna Wydawnicza, Warszawa, 1993.
- [YOWT10] Dong-Wan Yoo, Hyon-Dong Oh, Dae-Yeon Won, and Min-Jea Tahk. Dynamic Modeling and Control System Design for Tri-rotor UAV. *Department of Aerospace Engineering, Korea Advanced Institute of Science and Technology*, pages 762–767, 2010.
- [ZW13] J. Źrebiec and R. Włodarz. Laboratoryjny model tricoptera, 2013.