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 poświadczenie 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

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 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 on 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.

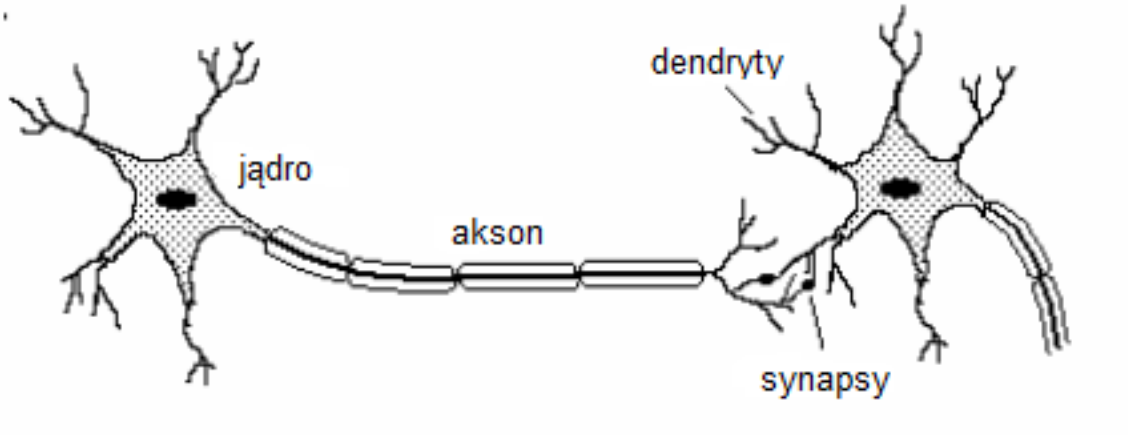
2.2. Sztuczne sieci neuronowe

Sztuczne sieci neuronowe są często określane jako początki sztucznej inteligencji ze względu na ich sposób działania, który przypomina działanie ludzkiego mózgu. Korzystanie z sieci neuronowych wymaga przejścia przez dwa podstawowe etapy, pierwszy z nich określany jest jako uczenie sieci, a drugi testowanie, które jest używane do oceny poprawności działania sieci po jej wytrenowaniu. Istnieją różne algorytmy uczenia sieci neuronowej, najpopularniejszy z nich nazywany jest perceptron 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 jaki same neurony, które równolegle przetwarzają dane wejściowe. Podejście to zostało zainspirowane z biologicznego systemu nerwowego.

2.3. 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.



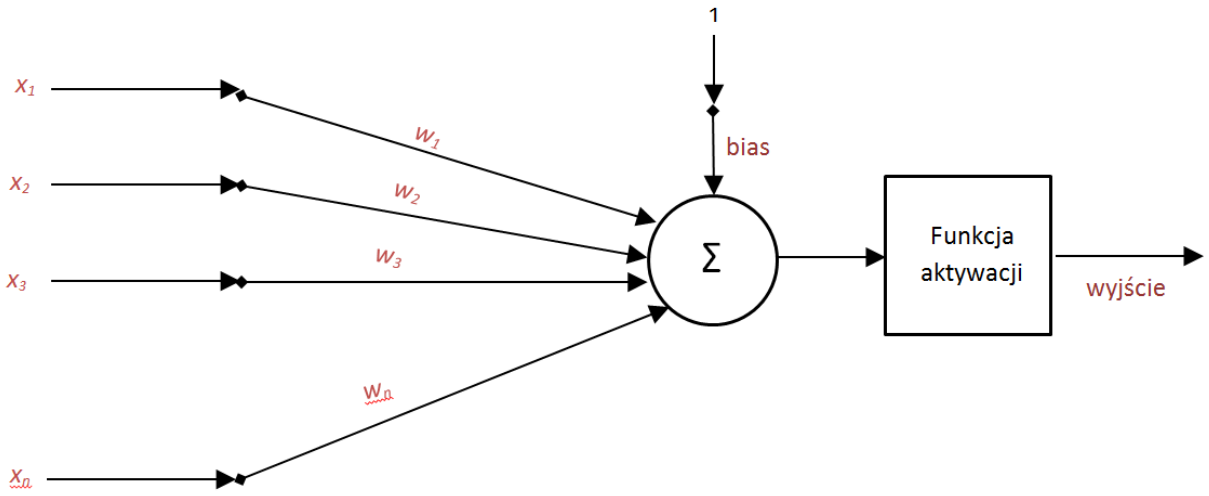
Rysunek 2.1: Biologiczny neuron.

Podstawową funkcję neuronu jest transportowanie przetworzonej informacji w postaci impulsu nerwowego, które są reprezentowane przez krótkotrwałą zmianę potencjału. Są one przewodzone od aksonu do synapsy która znajduje się na jego zakończeniach.

2.4. Sztuczny neuron

Sposób działania sztucznego neuronu jest ściśle oparty na działaniu biologicznego neuronu. Posiada on wiele wejść oraz jedno wyjście. Dla lepsze zrozumienia funkcjonowania takiego neuronu wskażmy jego różnice w stosunku do biologicznego. Współczesne komputery posiadają bardzo dużą moc obliczeniową skupiającej się w pojedynczych procesorach taktowanych wysoką częstotliwością. W odróżnieniu ludzki mózg posiada miliardy neuronów, które przetwarzają informacje wolniej niż współczesne procesory. Informacja przenoszona przez bodźce w ludzkim mózgu są reprezentowane przez logikę binarną o określonym progu aktywacji. W przypadku sztucznych neuronów funkcjonalność ta jest realizowane przez funkcje aktywacji, która na podstawie dobranego progu przypisuje wartość logiczną zbliżoną do "1"dla wartości powyżej 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 porównywane sa one do przekaźników. Synapsy występujące w ludzkim mózgu są odpowiednikami wag dla poszczególnych wejść neuronów.

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 przystosowywać 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.



Rysunek 2.2: Sztuczny neuron.

2.5. Nauka sieci

Nie jest możliwe aby w pełni za modelować pracę ludzkiego mózgu. Nie mniej jednak sztuczne sieci neuronowe pozwalają na rozwiązywanie wiele skomplikowanych zagadnień jak rozpoznawanie wszelkiego rodzaju wzorców oraz wiele innych. Bardzo ciekawą własnością sieci neuronowych jest to, iż nawet przy tej samej architekturze po ówczesnym przygotowaniu są w stanie one rozwiązywać całkowicie różne zagadnienia. Takie przystosowanie określane 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 znanym algorytmem w tej kategorii jest wsteczna propagacja błędów (ang. backpropagation). Polega ono na uczeniu sieci bazując na błędach. Początkowo wagi połączeń między neuronami są wybierane w sposób losowy, następnie na wejście sici podawany jest wektor wejść z znanymi poszczególnymi wartościami oraz znana jest również wartość oczekiwana dla wyjścia. Jest ona porównywana z aktualnym wyjściem sieci, a następnie na podstawie wielkości tego błędu wyliczana 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 swoją popularność ze względu na brak konieczności znajomość oczekiwanego wektora danych wyjściowych podczas uczenia sieci. W tym przypadku wyjście sieci nie jest weryfikowane. Do najbardziej znanych algorytmów reprezentujących to podejście zaliczamy sieci Kohonena - samo-

ograniczające 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.

2.6. Funkcja aktywacji

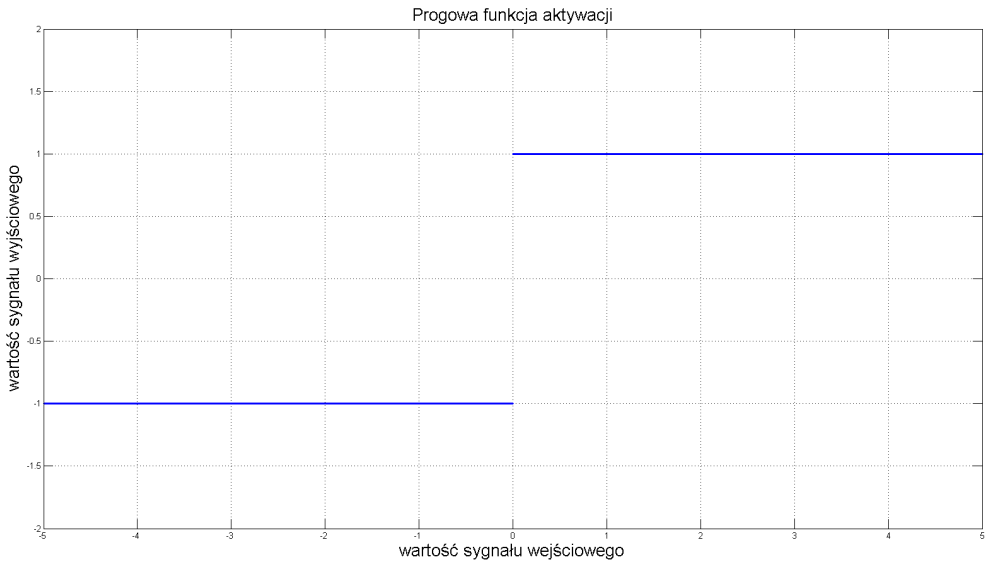
W sieciach neuronowych funkcja aktywacji odpowiedzialna jest za przekształcenie sumy powstałej przez dodanie iloczynów poszczególnych wag z odpowiadającymi im sygnałami wejściowymi. Wyróżnia się trzy główne typy takich funkcji: progowa, liniowa, sigmoidalna.

2.6.1. Progowa funkcja aktywacji

Progowa funkcja aktywacji została przedstawiona przez McCullon'a oraz Pits'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.



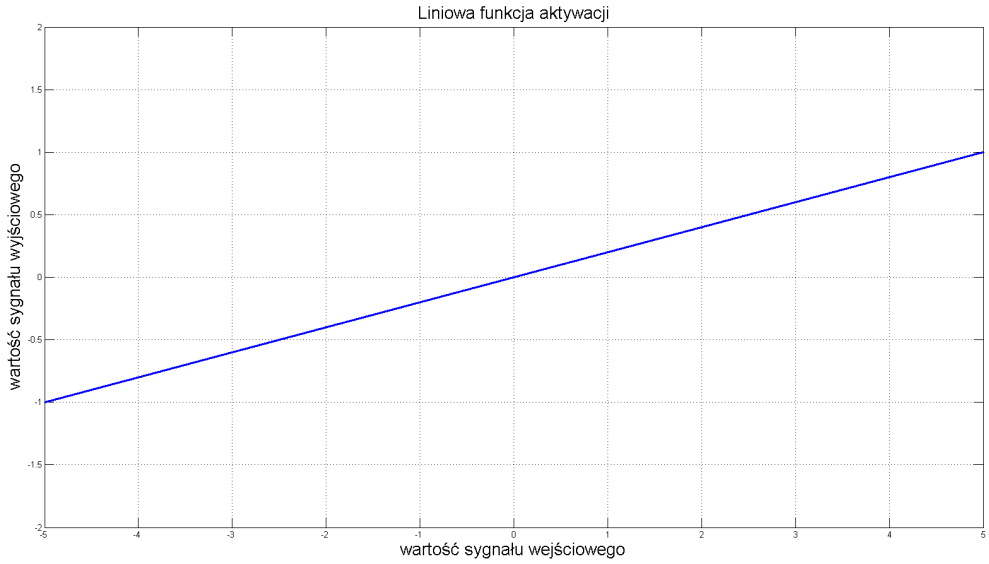
Rysunek 2.3: Przykładowa funkcja progowa.

2.6.2. Liniowa funkcja aktywacji

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

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

Na rysunku



Rysunek 2.4: Przykładowa funkcja liniowa.

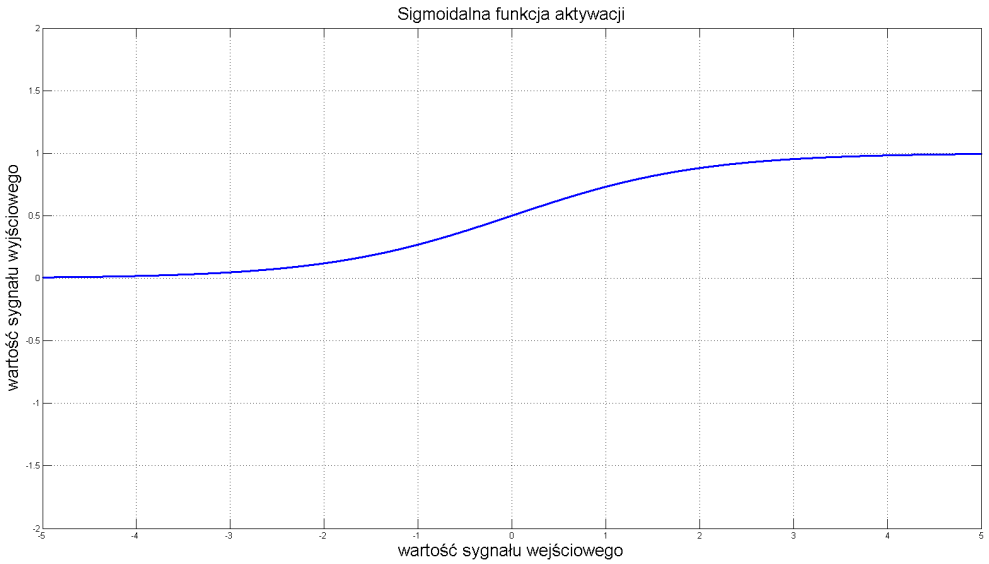
2.6.3. Sigmoidalna funkcja aktywacji

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

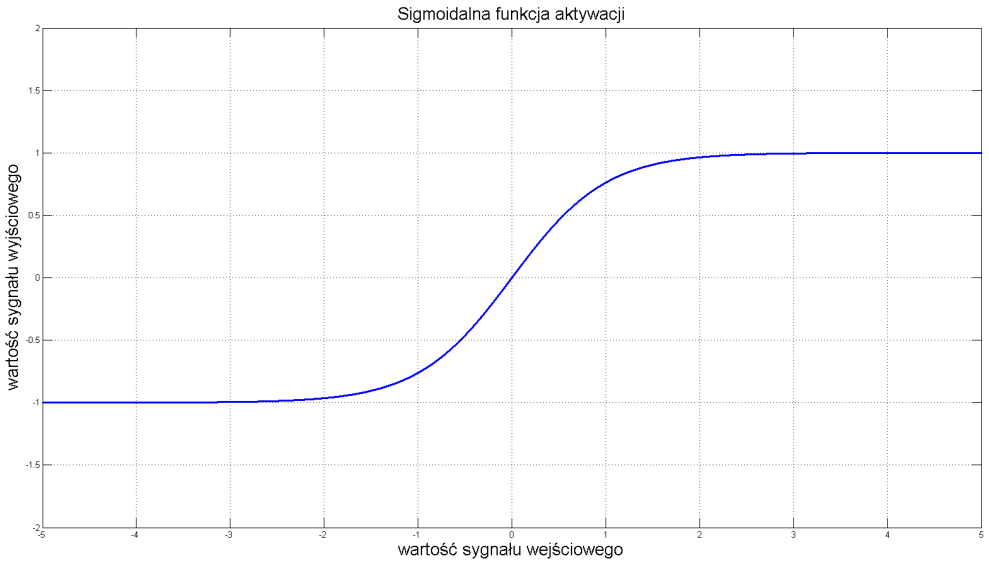
$$f(x) = \frac{1}{1 + e^{-x}}$$

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

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



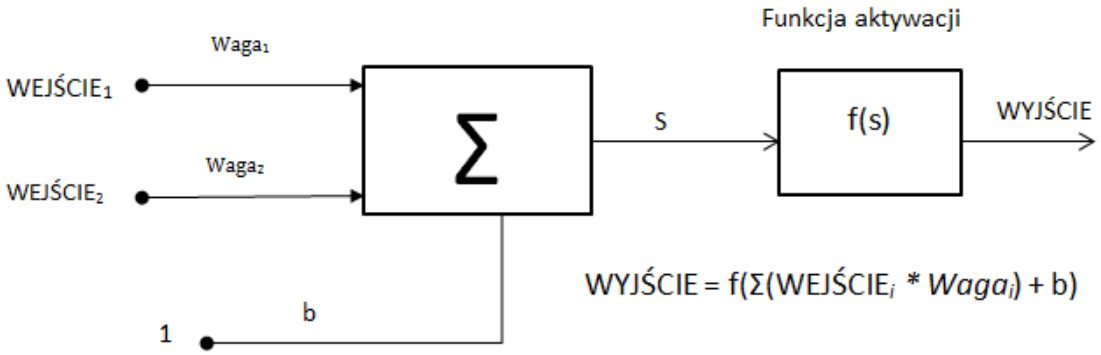
Rysunek 2.5: Wykres funkcji sigmoidalnej.



Rysunek 2.6: Wykres funkcji tangensa hiperbolicznego.

2.7. Perceptron

Perceptronem określa się sieć neuronową najprostszego typu, składającej się z neuronów McCullocha-Pittsa. Ich działania oparte jest na klasyfikacji sygnału wejściowego przez ustawieniu odpowiadający mu poziom wartości wyjściowej. Odbywa się to przez równoległe przemnożenie sygnałów wejściowych przez wagi połączeń na których się znajdują. Otrzymane ilorazy są sumowane,a następnie stanowią one wejście funkcji aktywacji której wyjście stanowi sygnał wyjściowy całego neuronu. Powyższe operacje zostały przedstawione na rysunku 2.7



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. Cecha ta uniemożliwia również wytrenowanie sieci jednowarstwowej, która rozwiązywała by problemy nieliniowe np. sieć wykonująca operację logiczną xor, która wymaga architektury posiadającej więcej niż jednej warstwy. 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.

2.8. Backpropagation

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

Wyróżnia się dwie najważniejsze fazy, 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 działania rozpoczyna się od ostatniej warstwy sieci do której podawana jest wartość błędu. Wykonywanie odwrotnych obliczeń wzdłuż sieci pozwala na wykrycie,

które wagi wpłynęły na zwiększenie błędu a następnie korekcję ich wartości tak aby zminimalizować błąd.

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.

$$U =<< X^1, z^1 >, < X^2, z^2 >, ..., < X^n, z^n >> \tag{2.1}$$

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

$$W^{j+1} = W^j + \eta^j \delta^j X^j \tag{2.2}$$

We wzorze 2.2 przyjęto:

$$\delta^j = z^j - y^j \tag{2.3}$$

gdzie:

$$y^j = W^j * X^j \tag{2.4}$$

Reguła 2.2 w pełni opisuje wszystkie konieczne do wykonania obliczenia. Warto jednak mieć na uwadze, iż wektor W^1 musi być określone. Bardzo często jest on inicjowany wartościami wybranymi losowo lub wartości te są przenoszone z poprzedniego procesu uczenia nawet jeśli jego zadaniem była realizacja zupełnie innej funkcji. Bez względu na dobierane sposób inicjacji należy mieć na uwadze aby dobrane wartości były różne $\forall_{\eta,\nu} w^1_\eta \neq w^1_\nu$. 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 neurono y^j z wartością oczekiwaną z^j, co jest jednoznaczne ze spełnieniem kryterium 2.5.

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

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

$$Q = \sum_{j=1}^{N} Q^j \tag{2.6}$$

gdzie:

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

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

$$w_i^{'} - w_i = \Delta w_i = -\eta \frac{\delta Q}{\delta w_i} \tag{2.8}$$

W algorytmie backpropagation wyróżnia się poszczególne etapy:

1. test
2. test2
3. test3

Podczas efektywnej nauki korekty wag wraz z błędem na wyjściu sieci maleją z upływem czasu nauczania sieci. Zostaje on zakończony, gdy zostanie wykonana odpowiednia liczba iteracji na zbiorze uczącym lub błąd na wyjściu będzie poniżej zadanego progu.

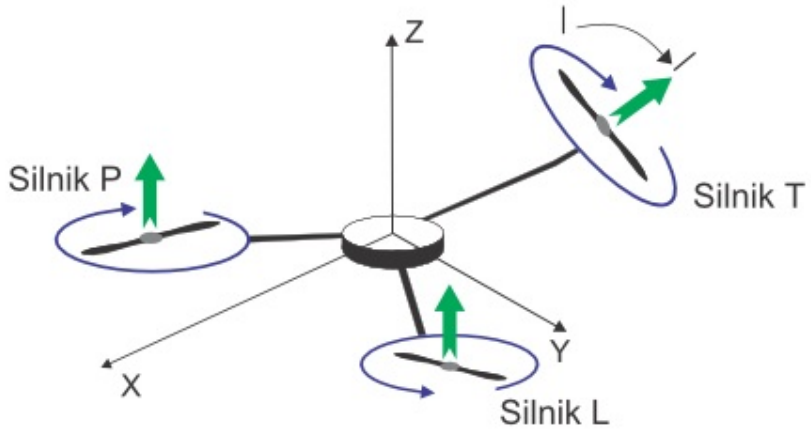
3. Architektura statku latającego typu tricopter

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 sterowanie tricopterem w przestrzeni, oraz złożoną konstrukcję w poniższym podrozdziale zostaną przedstawione podstawowe zagadnienia związane z tymi zagadnieniami.

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.

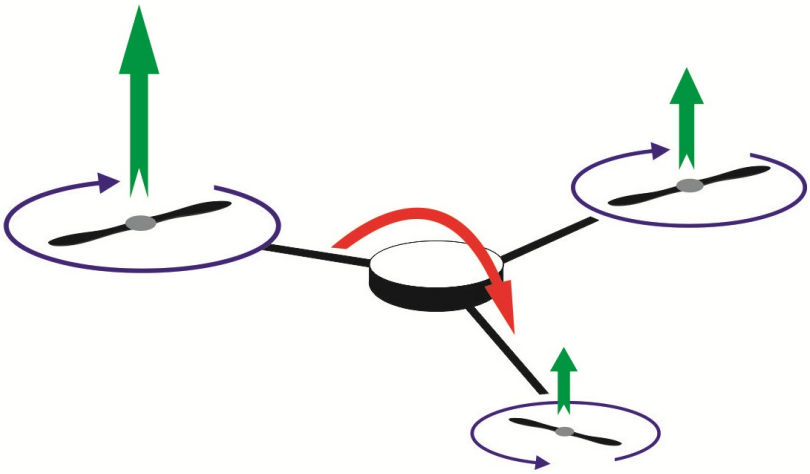
Tricopter należy grupy rzadko spotykanych wielokomórkowców wyposażonych w nieparzystą liczbę silników odpowiadających za napędy w układzie. Konstrukcja ta zdecydowanie utrudnia sterowanie ze względu na brak stabilności tego układu. Spowodowane jest to brakiem kompensacji siły bocznej pochodzącej od napędu niesparowanego. Czego skutkiem jest efekt autorotacji obiektu wokół osi Z przy założeniu, iż wszystkie wirniki są w pozycji pionowej, tak jak w przypadku większości obiektów latają-

cych pionowego startu. Aby wyeliminować powyższy efekt zastosowano odpowiedni moduł składający się z serwomechanizmu, 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 zaprezentowano poniżej sposoby przemieszczania względem konkretnych osi tricoptera. Szczegółowe informacje związane z powyższymi zagadnieniami zostały opisane w pracy inżynierskiej autora w rozdziale 5.

3.1.1. Przemieszczanie względem osi X

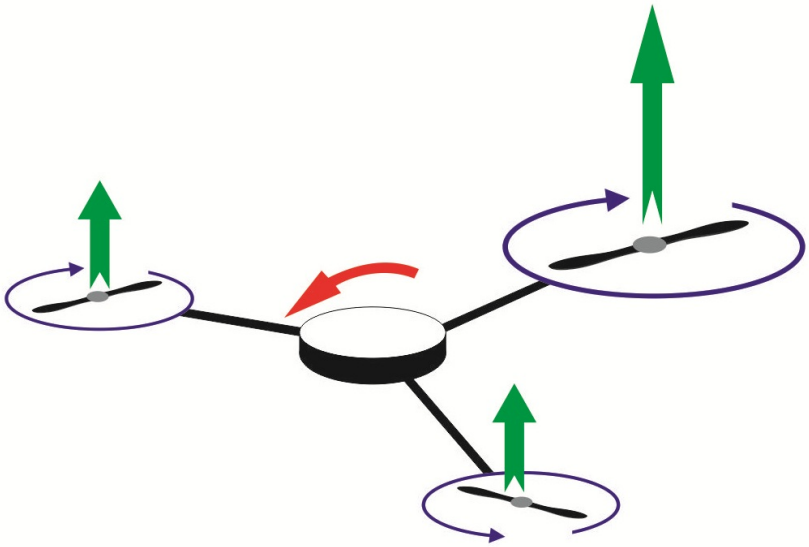
Aby dokonać obrotu względem osi X konieczna jest ingerencja w ciąg wirników L oraz P. Aby uzyskać efekt przechylenia w lewą stronę konieczna jest redukcja siły ciągu na silniku L. Opcjonalnie w celu przyśpieszenia manewru można proporcjonalnie zwiększyć ciąg silnika P. Na rysunku 3.2 przedstawiono opisany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



Rysunek 3.2: Sterowanie oraz stabilizacja w osi X.

3.1.2. Przemieszczanie względem osi Y

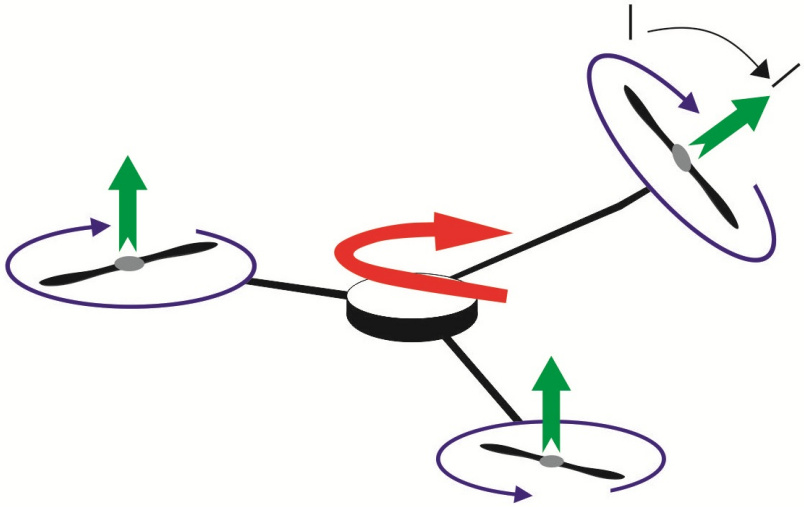
Obrót względem osi Y można zdecydowanie określić jako najważniejszy dla każdego obiektu latającego, odpowiada on za przemieszczenie się w przód. Odchylenie tricoptera zapewnia się przez regulację ciągu na wirniku T. Tak jak w przypadku osi X można zwiększyć prędkość obrotu obiektu przez symetrycznym 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 pominąć szczegółowy opis budowy modelu. Szczegółowe informacje dotyczące konstrukcji oraz parametrów wielowirnikowca zostały przedstawione w pracy inżynierskiej autora w rozdziale 2.

4. Aplikacja sterująca

W niniejszym rozdziale przedstawiono informacje na temat aplikacji sterującej. 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 ostrych wymaganiach czasowych (ang. hard real-time) - wymagania czasowe muszą być skrupulatnie przestrzegane, naruszenie ram czasowych może wpłynąć na życie ludzkie, środowisko czy też sam system,

– Systemy o słabych wymaganiach czasowych (ang. soft real-time) – głównym kryterium oceny tych 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 - powinny działać nieprzerwanie w okresie od uruchomienia systemu do jego wycofania,

– Zależność od otoczenia - zachowanie opiera rozpatruje się w kontekście otoczenia. Prowadzone obliczenia zależą od zdarzeń oraz danych pochodzących z zewnątrz układu,

– Współbieżność - struktura systemu narzuca, aby jednoczesne 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 zetknięciu 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 opisano w pracy

W trakcje powstawania niniejszej pracy na rynku dostępne są nowsze czujniki, które mogłyby podnieść jakość oraz częstość pomiarów, jednak ingerencja w konfigurację uniemożliwiła by porównanie algorytmów tradycyjnych z sieciami neuronowym 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 określane jest również jako w pełni wywłaszczalne. Oznacza to, iż dopuszczane jest wywłaszczenie procesu działającego w trybie jądra. Cecha ta wraz z wcześniej narzuconymi priorytetami dla poszczególnych procesów gwarantuje, iż aplikacja sterująca, uruchomiona z wysokim priorytetem nie zostanie wywłaszczona na zbyt długi czas przez inne procesy systemowe.

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

4.4. Aplikacja sterująca

Ze względu na powyżej omówione aspekty, do tworzenia aplikacji sterującej wykorzystano język C++.

4.5. Niezależność aplikacji od platformy sprzętowej

Dobrze zaprojektowana aplikacja sterująca obiektami 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 w wybrano najważniejsze zagadnienia:

nteger pod typem double - w systemach 32 bitowych nie występował 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 przypisywanych do typu double. Wartość ta zostanie obcięta o początkowe 12 bitów.

Podstawowa wersja aplikacji została poddana testom, które porównały wyniki działania sieci neuronowej na 2 całkowicie odmiennych platformach.

4.5.1. Najważniejsze aspekty

Pierwszym, a zarazem najważniejszym aspektem jest problem, który może powstać podczas mnożenia liczb zmiennoprzecinkowych. W przypadku wykonywania tej operacji dwie liczby reprezentowane w postaci binarnej o z góry określonej liczbie dostępnych bitów o określonej liczbie bitów wynik zapisywany jest do zmiennej o określonej liczbie bitów, co wiąże się z uzyskaniem błędu odcięcia w przypadku nie wystarczającej liczbie bitów potrzebnych do reprezentacji części ułamkowej.

Konfiguracja miała na celu uwydatnienie problemu błędu obcięcia liczb zmiennoprzecinkowych

4.5.2. 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. Konieczne jednak był dobór liczb maksymalnie wykorzystujący precyzje liczby zmiennoprzecinkowej. Następnie na wejście sieci podawana jest stała wartość oraz odczytywane jest wyjście sieci. Kroki te powtarzane są na wszystkich dostępnych platformach sprzętowych,a następnie porównywane są poszczególne wyniki w celu zbadania rozbieżności.

4.6. 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 dostarcza dane niezbędne do dogłębnej 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

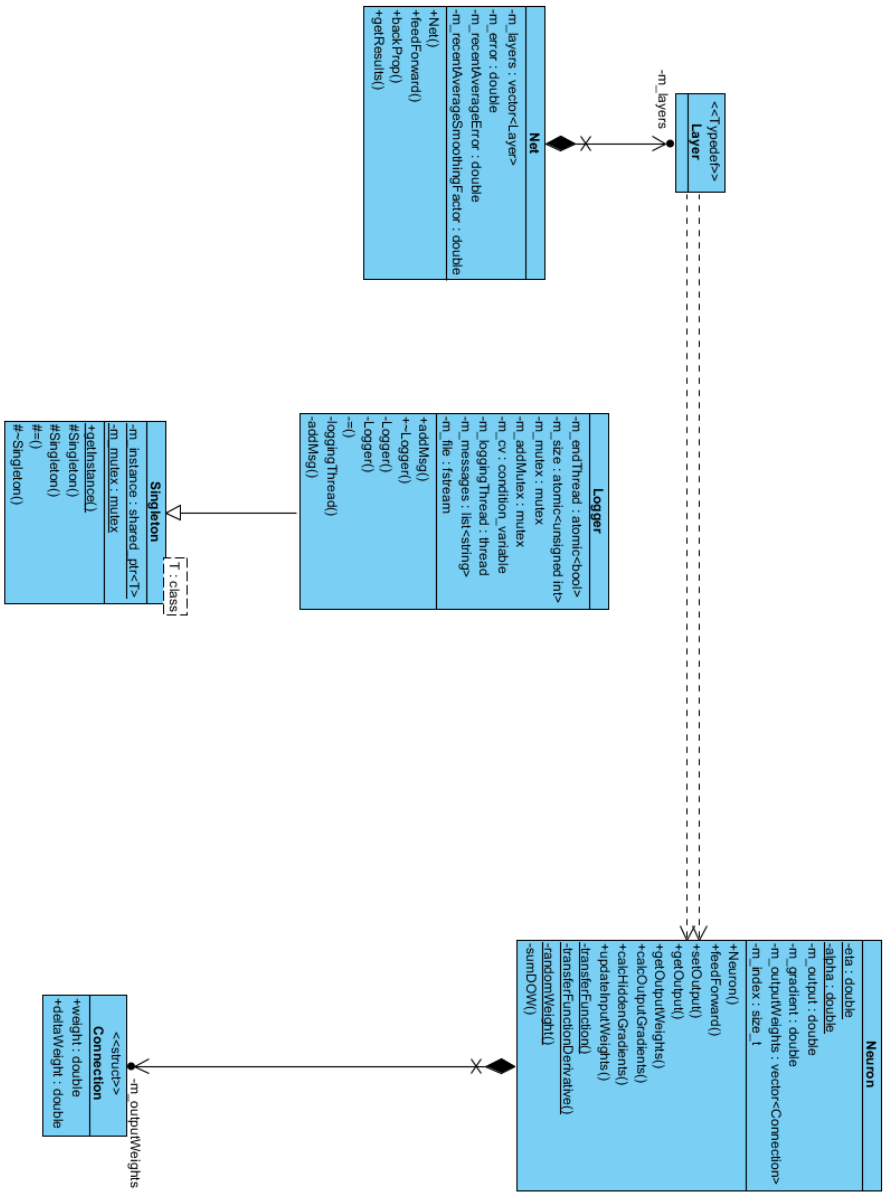
– klasa Net - odpowiada za pełną reprezentację sieci. Przechowuje w wektorze wszystkie utworzone neurony w uszeregowanych warstwach. W swoim konstruktorze przyjmuje wektor liczb, które odpowiadają za liczbę tworzonych warstw oraz ilości neuronów w każdej z nich. W zależności od konfiguracji inicjowane są wagi wszystkich połączeń. Są one dobierane losowo lub za pomocą wcześniej zdefiniowanych wartości. Wywołanie funkcji feedForward z argumentem w postaci wektora danych wejściowych powoduje przepływ i przetworzenie informacji przez sieć. Wektor wartości wyjściową odczytujemy za pomocą funkcji getResults. Dodatkowo klasa ta została wyposażona w funkcje 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 treści 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 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 oraz przechowywane dane zajmują sporo pamięci w przypadku systemów wbudowanych. Całość została obudowana w wzorzec projektowy Singleton, aby ułatwić dostęp do klasy oraz zabezpieczyć kod przed utworzeniem wielu instancji tego obiektu.



Rysunek 4.1: Diagram klass aplikacji sterującej.

5. Testy systemu sterującego

6. Podsumowanie

Bibliografia