Skrypt
Nr 289

Krzysztof Bartecki

Sztuczne sieci neuronowe w zastosowaniach

Zbiór ćwiczeń laboratoryjnych z wykorzystaniem przybornika *Neural Network* programu *Matlab*

ISSN 1427-9932
ISBN 978-83-60691-89-2

Opole 2010

POLITECHNIKA OPOLSKA

Recenzent:
prof. dr hab. inż. Dariusz Uciński

Redaktor:
Jan Sadecki

Komitet Redakcyjny Wydawnictw Politechniki Opolskiej
ul. S. Mikołajczyka 5

Skład: Oficyna Wydawnicza Politechniki Opolskiej
Nakład 265+25+10 egz. Ark. wyd. 5,6. Ark. druk. 5,5.
Druk i oprawa: Sekcja Poligrafii Politechniki Opolskiej.

SPIS TREŚCI

WPROWADZENIE

W dziedzinie numerycznego przetwarzania danych współczesne komputery są znacznie szybsze i dokładniejsze od ludzkiego mózgu. Znacznie szybciej wykonują takie operacje, jak np. całkowanie numeryczne, operacje na macierzach czy wyszukiwanie informacji w bazach danych. Jednak istnieje pewna grupa zagadnień, w których mózg nadal wykazuje przewagę nad komputerem. Na przykład ludzie o wiele szybciej i lepiej rozpoznają twarze innych osób, niż czynią to współczesne systemy komputerowe; lepiej także wykonują np. tłumaczenia złożonego tekstu z jednego języka na inny. Wynika to z dwóch zasadniczych przyczyn, charakteryzujących pracę mózgu:

- przetwarzanie danych odbywa się w nim w sposób równoległy – tworzące go komórki nerwowe, zwane neuronami, wykonują swoje „obliczenia" równocześnie, dlatego też ich efektywna prędkość wielokrotnie może przewyższać prędkości współczesnych komputerów,
- mózg jest strukturą elastyczną – nie wymaga programowania, wiedzę nabywa w wyniku procesu uczenia.

Sztuczne sieci neuronowe (ang. *Artificial Neural Networks*) stanowią bardzo uproszczony model systemów nerwowych żywych organizmów, w tym ludzkiego mózgu. Mimo swej uproszczonej budowy wykazują one charakterystyczne dla swych biologicznych odpowiedników cechy, takie jak: zdolność do uogólniania wiedzy, jej aktualizacji na podstawie wcześniej poznanych wzorców, czy też odporność na uszkodzenia pojedynczych elementów przetwarzających – sztucznych neuronów.

Do pierwszych na krajowym rynku wydawniczym pozycji poświęconych problematyce sztucznych sieci neuronowych można zaliczyć książki Tadeusiewicza [12, 13], którego wcześniejsze prace, dotyczące biocybernetyki, stanowiły podbudowę dla rozwoju tej dziedziny w Polsce. W początkowym okresie jej rozwoju wiele pozycji książkowych stanowiły tłumaczenia pozycji angielskojęzycznych, wśród których można wymienić m.in. prace Hertza, Krogha i Palmera [2] czy też Mastersa [6]. Jedną z pierwszych pozycji w języku polskim, w której szczególną uwagę poświęcono możliwości zastosowania

sztucznych sieci neuronowych w zadaniach identyfikacji oraz sterowania obiektami dynamicznymi była książka Korbicza, Obuchowicza i Ucińskiego [3]. Na możliwości zastosowania metod bazujących na algorytmach neuronowych oraz logice rozmytej w automatyce wskazano również w pracy [10]. Autorem serii ciekawych pozycji z dziedziny sieci neuronowych jest Osowski, omawiający w nich różnorakie aspekty dotyczące zarówno podstaw algorytmicznych sieci neuronowych [7], jak i możliwości ich zastosowań w wielu różnych dziedzinach [8, 9]. Szereg różnych zastosowań sieci neuronowych, m.in. w zagadnieniach aproksymacji, modelowania i identyfikacji, przetwarzania i rozpoznawania obrazów oraz w medycynie i diagnostyce medycznej zaprezentowano w monografii [1]. Z kolei w książce Rutkowskiej i współautorów omówiono szereg zagadnień dotyczących nie tylko sieci neuronowych, ale również algorytmów genetycznych oraz systemów rozmytych [11]. Kilka spośród zaprezentowanych w niniejszym skrypcie ćwiczeń zostało zainspirowanych ciekawymi przykładami zawartymi w pracy Żurady, Barskiego i Jędrucha [15]. Atrakcyjną pozycją o charakterze popularnonaukowym jest wydana niedawno książka prof. Tadeusiewicza i współautorów [14]. Najnowszymi, w momencie pisania niniejszego opracowania, pozycjami książkowymi poświęconymi problematyce sztucznych sieci neuronowych, są: obszerna monografia Łęskiego, w której omówiono systemy neuronowo-rozmyte [5] oraz książka Kosińskiego, omawiająca sieci neuronowe w ujęciu metod dynamiki nieliniowej [4].

Stosowane obecnie metody implementacji sztucznych sieci neuronowych podzielić można na dwie zasadnicze grupy:

- metody sprzętowe, czyli dedykowane komputery neuronowe, procesory i układy scalone,
- metody programowe, czyli aplikacje, umożliwiające symulację działania sieci neuronowych na komputerach osobistych.

Wśród dostępnych obecnie aplikacji można wymienić m.in. takie programy, jak: *STATISTICA Automatyczne Sieci Neuronowe* (SANN), symulator sieci neuronowej *Neuronix* wchodzący w skład pakietu sztucznej inteligencji *Aitech Sphinx*, czy też bibliotekę FANN (ang. *Fast Artificial Neural Network*), będącą otwartym projektem programistycznym, implementującym wielowarstwową, jednokierunkową sieć neuronową. Jednak jednym z najbardziej

Oprócz wyżej wymienionych głównych kategorii funkcji, w skład przybornika wchodzą również funkcje pomocnicze. Wśród nich można wymienić m.in.:

- funkcje służące do inicjalizacji współczynników wagowych,
- funkcje wyznaczające wartość błędu sieci,
- funkcje umożliwiające tzw. *preprocessing*, czyli wstępne przetwarzanie danych uczących,
- funkcje użytkowe, wykorzystywane w procesie uczenia sieci, np. obliczające gradient funkcji błędu sieci względem jej współczynników wagowych,
- funkcje do tworzenia wykresów,
- funkcje realizujące graficzny interfejs użytkownika.

Składnia funkcji przybornika, zgodnie z macierzową filozofią *Matlaba*, umożliwia przetwarzanie wsadowe (ang. *batching*), czyli jednoczesną prezentację wielu wzorców, zarówno na etapie uczenia jak i symulacji działania sieci. W znaczący sposób wpływa to na efektywność działania tych funkcji.

Zawartość skryptu

Niniejszy skrypt powstał jako materiał pomocniczy do prowadzonych przez autora na Wydziale Elektrotechniki, Automatyki i Informatyki Politechniki Opolskiej ćwiczeń laboratoryjnych z przedmiotów *narzędzia sztucznej inteligencji* oraz *metody sztucznej inteligencji*. W trakcie ćwiczeń z sztucznych sieci neuronowych wykorzystuje się opisany w poprzednim punkcie przybornik *Neural Network* programu *Matlab*.

W skrypcie zaprezentowano zestaw sześciu ćwiczeń laboratoryjnych, zrealizowanych z wykorzystaniem funkcji przybornika, dotyczących typowych, praktycznych zastosowań sztucznych sieci neuronowych. Należą do nich, zgodnie z kolejnością wykonywanych ćwiczeń, takie zagadnienia jak:

- prognozowanie ciągów czasowych,
- klasyfikacja danych,
- kompresja i dekompresja danych,

- aproksymacja wielowymiarowych funkcji nieliniowych,
- usuwanie zakłóceń z wzorców,
- modelowanie (identyfikacja) obiektów dynamicznych.

W ćwiczeniu pierwszym wykorzystano najprostszą sieć neuronową, zrealizowaną w formie pojedynczego, liniowego neuronu. Realizuje ona klasyczny *autoregresyjny model prognostyczny*, przy czym w roli wzorców uczących wykorzystuje się dane reprezentujące aktywność plam słonecznych w kolejnych latach. Zadaniem sieci jest tu prognoza aktywności plam na podstawie danych z lat poprzednich.

W ćwiczeniu drugim warstwa perceptronów, czyli neuronów o progowej funkcji aktywacji, wykorzystana zostaje w zadaniu klasyfikacji wzorców, reprezentujących mapy bitowe kolejnych liter alfabetu. Działanie sieci perceptronowej, nauczonej rozpoznawania liter wzorcowych, podlega weryfikacji na danych testowych w postaci liter zawierających zakłócenia.

Ćwiczenie trzecie polega na zastosowaniu sieci Kohonena w zadaniu kompresji obrazu. Poszczególne neurony sieci uczą się reprezentować kategorie elementów (ramek) obrazu, w zależności od stopnia szarości wchodzących w ich skład pikseli. W oparciu o informację zawartą we współczynnikach wagowych tych neuronów możliwa jest dekompresja obrazu.

W czwartym ćwiczeniu jednokierunkowa, dwuwarstwowa sieć neuronowa wykorzystana jest jako układ aproksymujący. Sieć realizuje tu odwzorowanie, znane w robotyce pod nazwą *odwrotnego zadania kinematyki*: na podstawie podanych na jej wejścia zadanych współrzędnych chwytaka manipulatora, na wyjściach sieci generowane są odpowiednie kąty przegubów jego ramion. Nauczona sieć testowana jest zarówno na wzorcowej trajektorii chwytaka, jak i na trajektorii innej niż wykorzystana w procesie uczenia sieci.

Piąte ćwiczenie polega na wykorzystaniu rekurencyjnej sieci Hopfielda jako tzw. „pamięci skojarzeniowej" (ang. *autoasociative memory*). Dzięki właściwościom skojarzeniowym, możliwe jest stopniowe odtwarzanie przez sieć oryginalnych wzorców, reprezentujących mapy bitowe kolejnych liter alfabetu, zniekształconych wcześniej przez wprowadzone losowo zakłócenia.

W ostatnim, szóstym ćwiczeniu sieć z neuronami o radialnej funkcji aktywacji wykorzystana zostaje jako nieliniowy model obiektu dynamicznego, przy czym rolę identyfikowanego obiektu pełni wahadło matematyczne. Model neuronowy, w oparciu o spróbkowane, opóźnione wartości wielkości wyjściowej obiektu (kąta wychylenia wahadła) oraz jego wielkości wejściowej (zewnętrznego momentu obrotowego przyłożonego do wahadła) dokonuje predykcji kolejnej wartości wielkości wyjściowej.

Stopień złożoności poszczególnych ćwiczeń, a w związku z tym również czas potrzebny na ich wykonanie, nie jest jednakowy. Najbardziej pracochłonne jest ćwiczenie czwarte, w którym wykorzystuje się jednokierunkową, nieliniową sieć neuronową w zadaniu aproksymacji odwrotnego zadania kinematyki. Z doświadczeń autora wynika, że przy podstawowej znajomości składni języka *Matlaba*, na ćwiczenie to należy przeznaczyć około czterech godzin laboratoryjnych. Z kolei na ćwiczenia: pierwsze (neuronowy model predykcyjny), trzecie (sieć Kohonena w kompresji danych) oraz szóste (identyfikacja nieliniowego obiektu dynamicznego), należy przeznaczyć około trzech godzin. Ćwiczenie drugie, w którym wykorzystywany jest klasyfikator na bazie sieci perceptronowej, powinno zająć studentom około dwóch godzin. Najmniej czasu zajmie wykonanie ćwiczenia piątego, w którym badana jest rekurencyjna sieć Hopfielda. Przy odpowiednim przygotowaniu, ćwiczenie jest możliwe do zrealizowania przez studentów w trakcie jednej godziny laboratoryjnej (45 minut).

Schemat postępowania w przypadku każdego z ćwiczeń, niezależnie od typu wykorzystanej w nim sieci oraz jej praktycznego zastosowania, jest podobny. Można tu wymienić następujące etapy:

- przygotowanie wzorców uczących,
- utworzenie sieci neuronowej,
- przeprowadzenie procedury uczenia sieci,
- weryfikację działania sieci na danych uczących,
- weryfikację działania sieci na danych testowych.

Ze względu na specyfikę poszczególnych typów sieci i metod ich uczenia, powyższy ogólny schemat ulegać może modyfikacjom w niektórych z ćwiczeń. Na przykład w ćwiczeniu trzecim, w którym wykorzystano sieć

Kohonena, tzn. sieć uczoną metodą „bez nauczyciela", na wzorce uczące składa się tylko macierz wzorców wejściowych. Z kolei w przypadku rekurencyjnej sieci Hopfielda (ćwiczenie piąte) brak etapu uczenia sieci. Wynika to z faktu, iż wartości współczynników wagowych są tu wyznaczane w pojedynczym kroku, już w fazie tworzenia sieci.

Zaprezentowana w skrypcie podstawowa struktura ćwiczeń stanowić może punkt wyjścia do ich modyfikacji bądź dalszej rozbudowy. Wskazana jest tu inwencja, zarówno ze strony prowadzącego zajęcia, jak i samych studentów. W ćwiczeniach od drugiego do piątego zaproponowano po sześć wariantów, umożliwiających podział studentów na grupy laboratoryjne. Zasadnicza treść ćwiczenia pozostaje w nich taka sama, różnią się one natomiast zestawem wzorców uczących, np. rozpoznawanych przez sieć liter. W ćwiczeniu pierwszym oraz szóstym decyzję odnośnie ewentualnego podziału na grupy oraz związanej z nim ewentualnej modyfikacji wzorców uczących pozostawiono prowadzącemu. Ponadto w końcowej części każdego z ćwiczeń znajduje się propozycja zestawu zadań do samodzielnego wykonania przez studentów.

Przedstawione w skrypcie ćwiczenia nie wykorzystują wszystkich istniejących struktur i metod uczenia sieci neuronowych, dostępnych w przyborniku *Neural Network*, nie wyczerpują również możliwości ich zastosowań. Spośród innych dostępnych w ramach przybornika struktur sieci neuronowych, których nie wykorzystano w niniejszym zbiorze ćwiczeń, można wymienić m.in. rekurencyjną sieć Elmana oraz sieć z tzw. adaptacyjnym kwantowaniem wektorowym (ang. *Learnig Vector Quantization*, LVQ).

Do wykonania ćwiczeń wymagany jest pakiet *Matlab* wraz z przybornikiem *Neural Network* w wersji 3.0 lub wyższej. Ze względu na zmieniającą się w kolejnych wersjach przybornika składnię niektórych jego funkcji, wskazane jest zapoznanie się z nią na początku ćwiczenia. Ponadto zakłada się, że przed przystąpieniem do kolejnego ćwiczenia studenci zapoznali się z teoretycznymi podstawami wykorzystywanej w nim sieci neuronowej. W skrypcie pominięto teoretyczne aspekty związane z działaniem oraz uczeniem poszczególnych typów sieci, skupiając się raczej na omówieniu ich zastosowania w określonym zadaniu oraz składni poleceń *Matlaba* i funkcji przybornika. Odpowiednie zależności matematyczne, opisujące działanie oraz algorytmy

rozbudowanych pod względem funkcjonalnym narzędzi programistycznych, umożliwiających projektowanie, symulację i analizę działania sztucznych sieci neuronowych, jest wchodząca w skład programu do obliczeń numerycznych *Matlab* biblioteka *Neural Network Toolbox* [17]. Jej popularność wynika m.in. z faktu, iż *Matlab* jest powszechnie znanym i stosowanym przez naukowców, inżynierów i studentów środowiskiem obliczeniowym. Obecnie na rynku istnieje wiele pozycji literaturowych opisujących możliwości *Matlaba*, zarówno polskich [16, 18, 20], jak i tłumaczeń z języków obcych, np. [19].

Neural Network Toolbox – ogólna charakterystyka

Przybornik *Neural Network* jest biblioteką rozszerzającą możliwości programu *Matlab* o funkcjonalności związane z projektowaniem, implementacją, symulacją oraz wizualizacją działania różnych struktur sztucznych sieci neuronowych. Funkcje wchodzące w jej skład można podzielić, ze względu na ich przeznaczenie, na kilka głównych kategorii. Można tu wymienić m.in.:

- funkcje służące do tworzenia różnych struktur sieci, takie jak: `newlind`, `newlin`, `newp`, `newc`, `newff`, `newhop`, i wiele innych,
- funkcje użytkowe, służące m.in. do inicjalizacji, przeprowadzania treningu i symulacji działania sieci, np.: `init`, `train`, `sim`,
- funkcje uczące – służące do realizacji pojedynczego cyklu uczenia, np. `learnwh`, `learnp`, `learngdm`,
- funkcje treningowe, których działanie polega na wielokrotnym wywoływaniu odpowiedniej funkcji uczącej,
- funkcje aktywacji neuronów – m.in. `purelin`, `hardlim`, `hardlims`, `tansig`, `logsig` i inne.

Spośród wyżej wymienionych, zazwyczaj jedynie funkcje należące do dwóch pierwszych kategorii wykorzystywane są w sposób bezpośredni przez użytkownika. Natomiast funkcje uczące, treningowe oraz funkcje aktywacji neuronów wykorzystywane są z reguły jedynie w sposób pośredni, przez funkcje tworzące struktury sieci oraz funkcje użytkowe.

uczenia poszczególnych rodzajów sieci neuronowych, znajdzie czytelnik w pozycjach literaturowych, których wykaz został dołączony do skryptu.

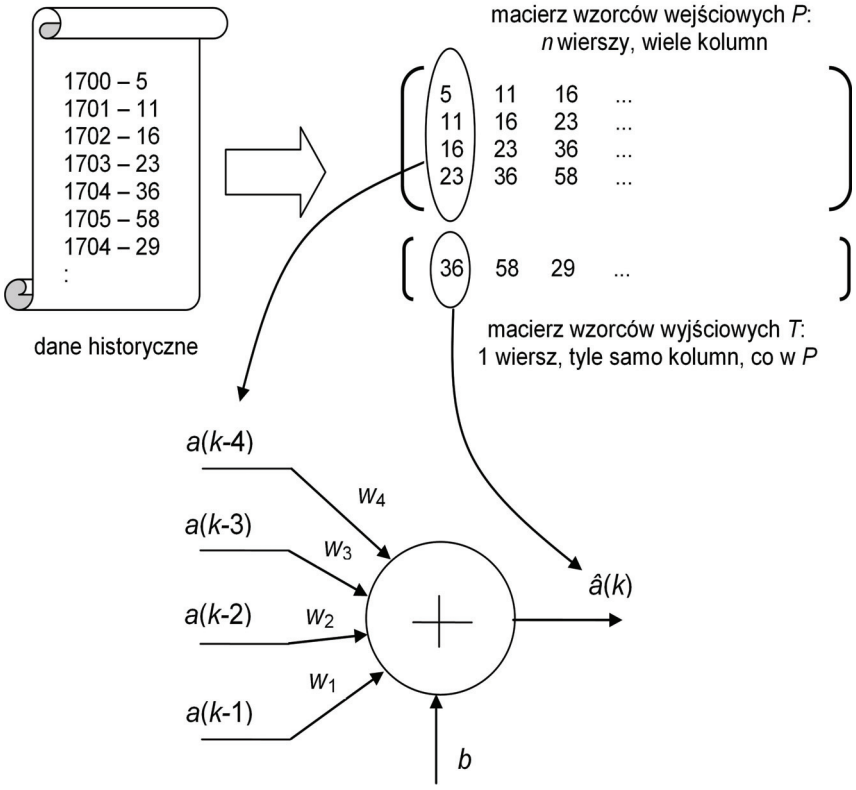
Autor wyraża przekonanie, że skrypt będzie pomocny studentom w poznawaniu zagadnień dotyczących zastosowań sztucznych sieci neuronowych, a także zachęci ich do dalszego zgłębiania ciekawej dziedziny naukowej, jaką są metody sztucznej inteligencji.

Autor

ĆWICZENIE 1

PROGNOZOWANIE CIĄGU CZASOWEGO Z WYKORZYSTANIEM NEURONU LINIOWEGO

W ćwiczeniu wykorzystamy sztuczną sieć neuronową o najprostszej strukturze, zrealizowaną w formie pojedynczego neuronu o liniowej funkcji aktywacji. Jego zadaniem będzie prognozowanie kolejnej, *k*-tej wartości pewnego ciągu czasowego $a(k)$, na podstawie znanych n poprzednich wartości elementów tego ciągu, tzn. wartości $a(k\text{-}1)$, $a(k\text{-}2)$, ..., $a(k\text{-}n)$. Taki model, zrealizowany przy założeniu, że zależność między prognozowaną wartością ciągu a wartościami jego poprzednich elementów może być opisana funkcją liniową, nazywamy *liniowym modelem autoregresyjnym rzędu n.*



Rys. 1.1. Schemat neuronu liniowego realizującego model autoregresyjny rzędu n=4

Ogólne równanie liniowego modelu autoregresyjnego ma zatem następującą postać:

$$\hat{a}(k) = w_1 \cdot a(k-1) + w_2 \cdot a(k-2) + \mathrm{K} + w_n \cdot a(k-n) + b \qquad (1.1)$$

gdzie w_1, w_2, ..., w_n oraz b to *parametry* modelu, zaś $\hat{a}(k)$ oznacza *prognozę* kolejnej wartości ciągu.

W naszym ćwiczeniu rolę modelu autoregresyjnego pełnił będzie sztuczny neuron – na jego wejścia podawać będziemy poprzednie wartości ciągu, rolę parametrów modelu pełnić będą *współczynniki wagowe* tego neuronu, zaś wartość prognozowana będzie pojawiać się na jego wyjściu (rys. 1.1).
Równanie (1.1) można zapisać również w następującej postaci:

$$a(k) = w_1 \cdot a(k-1) + w_2 \cdot a(k-2) + \mathrm{K} + w_n \cdot a(k-n) + b + e(k) \qquad (1.2)$$

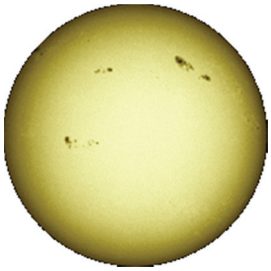
gdzie $e(k) = a(k) - \hat{a}(k)$ jest *błędem prognozy* dla k-tego kroku czasowego.

Dysponując zbiorem danych historycznych, dotyczących procesu podlegającego prognozowaniu, będziemy poszukiwać optymalnych wartości parametrów modelu autoregresyjnego. Oznacza to, że żądamy, aby prognozy $\hat{a}(k)$ generowane przez ten model jak najmniej różniły się od wartości rzeczywistych $a(k)$, tzn., aby suma kwadratów błędów prognozy $e(k)$ była jak najmniejsza dla całego zbioru danych. Tak opracowany model będzie można wykorzystać do prognozowania kolejnych wartości generowanych przez ten proces.

W ćwiczeniu rolę procesu podlegającego prognozowaniu pełnić będzie aktywność plam słonecznych**,** wyrażona liczbą plam obserwowanych na Słońcu w danym roku kalendarzowym (rys. 1.2). Aktywność ta wykazuje 11-letnią cykliczność. Okres największej aktywności w czasie tego cyklu nosi nazwę *maksimum słonecznego*.

Korzystając z historycznych danych odnośnie aktywności plam słonecznych, najpierw zdefiniujemy wzorce wejściowe, podawane na wejścia neuronu,

oraz wzorce wyjściowe („podpowiedzi nauczyciela" – rys. 1.1). Zbiór danych podzielimy również na wzorce uczące oraz wzorce testujące.



Rys. 1.2. Plamy słoneczne

Następnie przeprowadzimy procedurę doboru optymalnych wartości współczynników wagowych modelu, przy czym w przypadku rozpatrywanego neuronu liniowego współczynniki te można wyznaczyć na dwa sposoby: obliczając je bezpośrednio z odpowiedniego układu równań lub stosując metodę iteracyjną, czyli tzw. trening neuronu metodą uczenia nadzorowanego. Po wyznaczeniu wartości współczynników wagowych przeprowadzona zostanie weryfikacja modelu, mająca na celu sprawdzenie jakości prognozowania.

Czynności wstępne

1. Zapoznać się z dołączonymi do biblioteki *Neural Network Toolbox* pakietu *Matlab* skryptami demonstracyjnymi, ilustrującymi działanie pojedynczego neuronu (liniowego oraz nieliniowego). W tym celu należy uruchomić odpowiedni skrypt, wpisując w linii poleceń *Matlaba* nazwę skryptu oraz wciskając klawisz *Enter*:

`nnd2n1` – ilustracja działania pojedynczego neuronu o jednym wejściu,
`nnd2n2` – ilustracja działania pojedynczego neuronu o dwóch wejściach,
`demolin1` – wyznaczanie optymalnych wartości współczynników wagowych liniowego neuronu dla dwóch wzorców uczących (metoda bezpośrednia – obliczanie wag w jednym kroku). Za-

wartość skryptu można edytować poleceniem `edit demolin1`. Zapoznać się z zawartością skryptu.

`demolin2` – wyznaczanie optymalnych wartości współczynników wagowych liniowego neuronu dla dwóch wzorców uczących (metoda iteracyjna – tzw. trening neuronu).

`demolin4` – wyznaczanie optymalnych wartości współczynników wagowych liniowego neuronu dla czterech wzorców uczących (problem nieliniowy, aproksymacja przy pomocy liniowego neuronu).

`demolin7` – jak w skrypcie `demolin2`, dodatkowo zilustrowano efekt zbyt dużej wartości współczynnika prędkości uczenia.

2. Zapoznać się z przeznaczeniem oraz składnią funkcji biblioteki *Neural Network Toolbox*, które będziemy wykorzystywać w ćwiczeniu. W tym celu w linii poleceń *Matlaba* należy wpisać słowo kluczowe `help`, a po nim nazwę funkcji, której opis chcemy uzyskać:

`newlind` – tworzenie warstwy neuronów liniowych (także pojedynczego neuronu) oraz obliczanie wartości jej współczynników wagowych dla podanych wzorców uczących metodą bezpośrednią (poprzez pseudorozwiązanie nadokreślonego układu równań).

`newlin` – tworzenie warstwy neuronów liniowych (także pojedynczego neuronu) oraz inicjalizacja jej współczynników wagowych wartościami losowymi.

`train` – uczenie dowolnej jednokierunkowej sieci neuronowej (także liniowego neuronu) – czyli obliczanie wartości współczynników wagowych metodą iteracyjną.

`sim` – funkcja służąca do symulacji działania dowolnej sieci neuronowej (także liniowego neuronu).

Uwaga: funkcja sim jest funkcją przeciążoną – w celu uzyskania informacji o funkcji dotyczącej sieci neuronowych, w linii poleceń *Matlaba* należy wpisać polecenie: `help network/sim`.

Przebieg ćwiczenia

1. Pobrać ze strony http://www.k.bartecki.po.opole.pl/nsi/sunspot.txt plik z danymi do ćwiczenia oraz zapisać go (bez zmiany nazwy) w folderze roboczym *Matlaba*.

Uwaga: Plik zawiera dane o aktywności plam słonecznych w latach 1700-1950. W przypadku, gdyby plik był niedostępny w podanej lokalizacji, dane dotyczące plam słonecznych można znaleźć w Internecie, np. na stronie http://sidc.oma.be/sunspot-data/.

2. Załadować z pliku do przestrzeni roboczej (pamięci) *Matlaba* zawartość pliku `sunspot.txt`, wpisując w linii komend polecenie:

```
load sunspot.txt
```

3. Sprawdzić przy pomocy polecenia `whos`, czy odpowiednia macierz znajduje się w przestrzeni roboczej. Wyświetlić jej zawartość, wpisując jej nazwę (`sunspot`) w oknie poleceń. Jakie są rozmiary tej macierzy (polecenie `size`)?

Uwaga: pierwsza kolumna macierzy reprezentuje kolejne lata, zaś druga – liczbę plam zaobserwowanych w danym roku.

Usunąć zmienną `sunspot` z pamięci *Matlaba* (`clear sunspot`).

Uwaga: Wszystkie kolejne polecenia *Matlaba* będziemy zapisywać w formie skryptu (*m-pliku* skryptowego) – dzięki temu możliwe będzie wielokrotne ich wykonywanie; łatwiejsze będzie także opracowanie sprawozdania z ćwiczenia.

4. Pierwszym zadaniem, jakie powinien wykonać nasz skrypt, jest narysowanie wykresu aktywności plam słonecznych w latach 1700–1950. Wykres powinien być kompletny, tzn. mieć opisane osie oraz posiadać tytuł.

Otworzyć okno edytora skryptów (*File/New/M-File*) i umieścić w nim następujące polecenia:

```
clear         % usuwa wszystkie zmienne z pamięci
close all     % zamyka wszystkie okna graficzne
```

Zapoznać się ze składnią funkcji służącej do rysowania wykresów (`help plot`) oraz sposobem formatowania koloru, punktów i linii wykresu. Przykładowe wywołanie funkcji plot może wyglądać następująco:

```
figure(1)
plot(sunspot(:,1),sunspot(:,2),'r-*')
```

Dodać polecenia uzupełniające wykres o opisy osi oraz tytuł (`xlabel`, `ylabel`, `title`).

Uruchomić skrypt, przeanalizować wykres.

Uwaga: W przypadku wystąpienia w skrypcie błędu składniowego, w oknie *Matlaba* wyświetli się odpowiedni komunikat – należy wówczas poprawić błąd, ponownie zapisać i uruchomić skrypt ponownie.

5. Przyjmijmy, że rząd modelu autoregresyjnego będzie wynosił n=2 (tzn. zakładamy, że prognoza liczby plam na przyszły rok jest możliwa na podstawie znajomości ich liczby w roku bieżącym i poprzednim). Neuron będzie posiadał zatem 2 wejścia. Uzupełnić skrypt o definicję odpowiednich macierzy `P` oraz `T`, zawierających odpowiednio wzorcowe (uczące) dane wejściowe oraz wyjściowe dla naszego modelu (rys. 1.1).

```
L = length(sunspot);          % liczba danych

P = [sunspot(1:L-2,2)' ;      % macierz wzorców
     sunspot(2:L-1,2)'];      % wejściowych

T =  sunspot(3:L,2)';         % wektor wzorców
                              % wyjściowych
```

Przeanalizować składnię poleceń tworzących macierze `P` i `T`. Uruchomić skrypt, sprawdzić rozmiary macierzy (`size`) oraz ich zawartość. Można

w tym celu wykorzystać narzędzie *Workspace Browser* z paska ikon *Matlaba*.

6. Zapoznać się ze składnią funkcji: `plot3`, `grid`, `zlabel`. Uzupełnić skrypt o rysowanie w nowym oknie graficznym (polecenie `figure (2)`) wykresu trójwymiarowego, będącego przestrzenną wizualizacją wzorców wejściowych `P` oraz wyjściowych `T`.

Pomoc: Wywołanie funkcji `plot3` może wyglądać np. tak:
```
plot3(P(1,:),P(2,:),T,'bo')
```

Uruchomić skrypt, przeanalizować otrzymany wykres. Dołączyć opisy osi oraz tytuł. Jaka jest geometryczna interpretacja doboru optymalnych wartości współczynników wagowych neuronu `w1`, `w2`, `b`?

Uwaga: Wykres można obracać – umożliwia to ikona *Rotate 3D* na pasku ikon okna graficznego.

7. Wydzielmy ze zbioru danych wejściowych *P* oraz wyjściowych *T* podzbiór, obejmujący pierwszych 200 danych – tzw. zbiór wzorców uczących. W oparciu o ten zbiór obliczymy optymalne wartości współczynników wagowych neuronu (parametry modelu autoregresyjnego). Pozostałe dane posłużą nam natomiast do weryfikacji modelu. W tym celu należy, wykorzystując istniejące już macierze `P` i `T`, zdefiniować 2 nowe macierze: `Pu`, `Tu`.

Pomoc: Przykładowa definicja macierzy `Pu`:
```
Pu = P(:,1:200);
```

Uruchomić skrypt, sprawdzić, czy powstały odpowiednie macierze. Sprawdzić ich rozmiary oraz zawartość.

8. Utworzyć sztuczny neuron o odpowiedniej strukturze oraz obliczyć wartości jego współczynników wagowych metodą bezpośrednią (funkcja `newlind` – patrz przykład `demolin1`). Wykorzystać w tym celu macierze wzorców uczących `Pu` oraz `Tu`. Zmiennej reprezentującej sieć nadać nazwę `net`. Odpowiednie polecenie wpisać do skryptu.

9. Wyświetlić otrzymane wartości współczynników wagowych neuronu:

```
disp('współczynniki wagowe neuronu:' )
disp( net.IW{1} )
disp( net.b{1} )
```

Wartości odpowiednich współczynników wagowych przypisać pomocniczym zmiennym `w1`, `w2` oraz `b`:

```
w1 = net.IW{1}(1)
w2 = net.IW{1}(2)
b = net.b{1}
```

10. W kolejnym kroku dokonamy weryfikacji modelu – tzn. sprawdzimy jakość prognozowania, przeprowadzając symulację działania neuronu. Najpierw wykonamy ją na zbiorze danych uczących – tych samych, które posłużyły nam do obliczenia współczynników wagowych.

Załóżmy, że chcemy dokonać prognozy aktywności plam słonecznych na lata 1702–1901. W tym celu na wejście neuronu będziemy podawać kolejno zestawy danych o aktywności z lat: 1700 i 1701; 1701 i 1702; ..., 1899 i 1900.

Dzięki składni biblioteki możliwe jest przeprowadzenie tej czynności w sposób wsadowy – tzn. na wejścia neuronu podajemy jednocześnie wszystkie wzorce wejściowe, zawarte w macierzy `Pu`. W wyniku otrzymamy wektor odpowiedzi neuronu `Tsu` na każdy z wzorców wejściowych zapisanych kolejno w kolumnach macierzy `Pu` – czyli prognozowane wartości aktywności dla lat 1702–1901:

```
Tsu = sim(net,Pu)
```

Ponieważ znamy rzeczywiste wartości aktywności plam dla rozpatrywanego okresu, możemy porównać wartości prognozowane z rzeczywistymi. W tym celu w nowym oknie graficznym narysować (`plot`) nałożone na siebie (`hold on`) wykresy: rzeczywistych wartości aktywności plam słonecznych (`Tu`) oraz wartości z wyjścia modelu (`Tsu`). Na wy-

kresie przebiegi te powinny być zaznaczone różnymi kolorami. Ponadto wykres powinien posiadać tytuł, opisy osi oraz legendę (`legend`).

Odpowiednie polecenia dołączyć do skryptu. Uruchomić skrypt, sprawdzić jakość prognozowania.

11. Zadanie analogiczne jak w punkcie 10. powtórzyć dla wszystkich posiadanych danych (tzn. dokonać prognozy aktywności plam na lata 1702–1950). Narysować wykres porównawczy wartości rzeczywistych T oraz wartości prognozowanych (odpowiedzi neuronu) `Ts`.

Uwaga: nie wyznaczamy w tym celu nowych wartości współczynników wagowych (`newlind`). Chcemy jedynie sprawdzić (korzystając z funkcji `sim`), jak neuron utworzony w oparciu o dane uczące poradzi sobie z prognozą dla danych testowych.

12. Utworzyć wektor błędu prognozy `e` – patrz zależność (1.2). Dla przypomnienia: wartości jego elementów równe są różnicy między rzeczywistymi wartościami aktywności (`T`) a wartościami prognozowanymi, uzyskanymi z modelu neuronowego (`Ts`).

Narysować w nowym oknie graficznym wykres błędu prognozy (`plot`). Opisać jego osie, nadać tytuł.

13. Narysować histogram błędów prognozy (`hist`). Wyznaczyć wartość średnią (`mean`) i maksymalną (`max`) tego błędu oraz jego odchylenie standardowe (`std`).

14. Na podstawie zależności (1.3) obliczyć wartość sumarycznego średniokwadratowego błędu prognozy (ang. *Mean-Square-Error, MSE*):

$$MSE = \frac{1}{N}\sum_{k=1}^{N}\left(a(k)-\hat{a}(k)\right)^2 = \frac{1}{N}\sum_{k=1}^{N}e^2(k) \tag{1.3}$$

Można w tym celu skorzystać np. z funkcji `sumsqr`.

15. Rozpatrywany neuron liniowy o dwóch wejściach realizuje równanie płaszczyzny. W punkcie 6. narysowaliśmy wykres przedstawiający wzorce wejściowe *P* oraz wyjściowe *T* w postaci punktów w przestrzeni. Zgodnie z interpretacją geometryczną naszego zagadnienia, zadaniem neuronu jest aproksymacja tych punktów w taki sposób, aby błąd tej aproksymacji był jak najmniejszy. Na wykresie z punktu 6. będziemy chcieli zatem dorysować wykres płaszczyzny aproksymującej, realizowanej przez nasz neuron.

W tym celu należy:

- utworzyć siatkę, nad którą rozpięty będzie wykres płaszczyzny (funkcja `meshgrid`) – w wyniku powinny powstać 2 macierze `X` oraz `Y`, zawierające odpowiednie współrzędne punktów siatki (np. od 0 z krokiem co 10 do 200),
- utworzyć macierz `Z`, zawierającą wartości funkcji realizowanej przez neuron (równanie płaszczyzny):
  `Z = w1*X + w2*Y + b;`
- w drugim oknie graficznym (`figure(2)`) przy pomocy funkcji `mesh` dorysować, na tle narysowanych wcześniej punktów (`hold on`), wykres realizowanej przez neuron płaszczyzny aproksymującej.

Jeśli mamy do dyspozycji więcej czasu, możemy poeksperymentować z opcjami wykresu (np. `colormap`), ewentualnie wykorzystać inną funkcję do tworzenia wykresów – np. `surf`.

W ostatnich punktach ćwiczenia będziemy chcieli zmodyfikować zawartość skryptu w taki sposób, aby współczynniki wagowe neuronu wyznaczane były tym razem metodą iteracyjną – przeprowadzimy zatem procedurę uczenia (treningu) neuronu.

16. Zapisać skrypt pod nową nazwą. Opatrzyć komentarzem linię skryptu, w której wagi neuronu wyznaczane były metodą bezpośrednią (punkt 8). Tuż pod tą linią zdefiniować zmienne `S` oraz `lr` reprezentujące odpowiednio: liczbę neuronów w warstwie oraz współczynnik prędkości uczenia.

Pomoc: na początku wartość współczynnika prędkości uczenia `lr` przyjąć dowolną, mniejszą od jedności.

17. Korzystając z funkcji `newlin` utworzyć neuron liniowy. Sprawdzić, jakie początkowe wartości zostały przypisane jego współczynnikom wagowym (patrz punkt 9).

Uwaga: W różnych wersjach biblioteki składnia funkcji `newlin` może być różna – przed jej wywołaniem należy skorzystać z pomocy (`help newlin`).

18. Ustalić docelową wartość błędu uczenia (ang. *error goal*) oraz maksymalną liczbę kroków uczących (ang. *epochs*), np.:

```
net.trainParam.goal = 100;
net.trainParam.epochs = 1000;
```

19. Dopisać wywołanie funkcji realizującej procedurę uczenia neuronu (`train`). Przekazać 3 wymagane argumenty, wynik przypisać ponownie zmiennej `net`. Przed wywołaniem funkcji `train` można umieścić polecenie `figure` – wykres ilustrujący proces uczenia sieci będzie rysował się wówczas w oddzielnym oknie.

20. Zapisać i uruchomić skrypt.
- Co przedstawia wykres wyświetlany w trakcie uczenia neuronu?
- Czy proces uczenia jest zbieżny? Jeśli nie, to zastanowić się, jaka może być tego przyczyna – zmodyfikować odpowiedni parametr uczenia.
- Jakie są nowe wartości współczynników wagowych?
- Jaka jest uzyskana wartość błędu średniokwadratowego?

21. Powtórzyć procedurę uczenia dla innych wartości parametrów z punktu 18. Zbadać wpływ ich wartości na przebieg i jakość procesu uczenia oraz na jakość prognozowania. Ile wynosi maksymalna dopuszczalna wartość współczynnika prędkości uczenia `lr`, zapewniająca zbieżność tej procedury? Zapoznać się z przeznaczeniem funkcji `maxlinlr`.

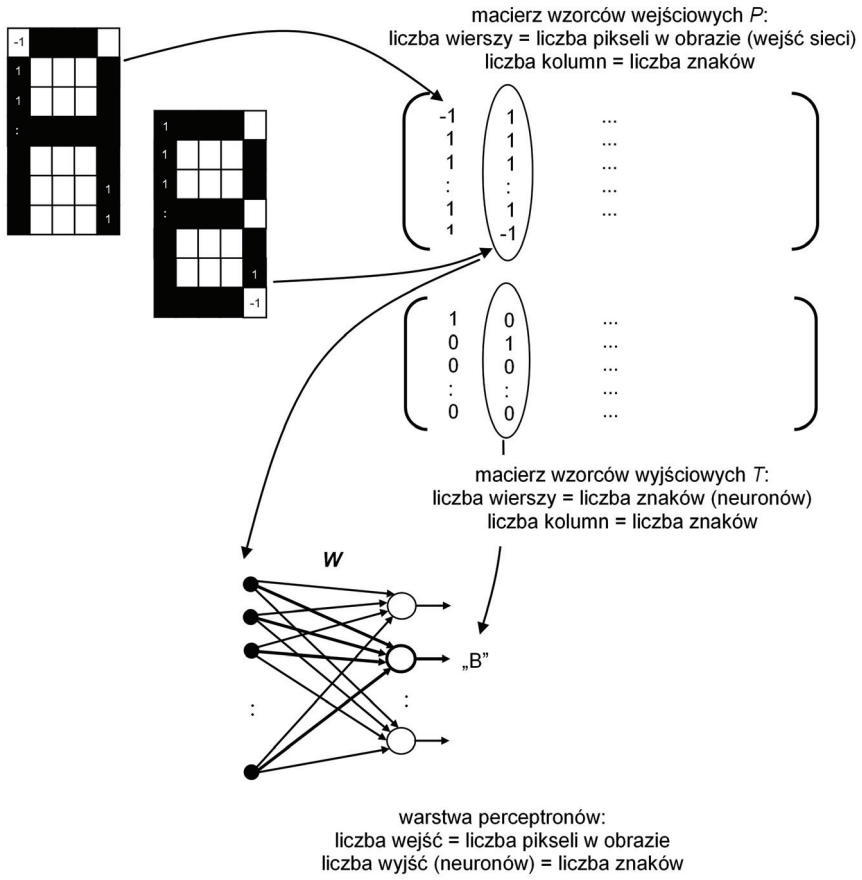
Zadanie dodatkowe:

Ćwiczenie wykonaliśmy, przyjmując narzuconą na wstępie strukturę naszego modelu. Prognoza kolejnej wartości aktywności odbywała się w oparciu o dane z dwóch poprzednich lat (tzn. rząd modelu n=2). Zmodyfikować skrypt w taki sposób, aby prognozowanie odbywało się w oparciu o większą niż poprzednio liczbę danych wejściowych (np. dla n=6, n=12). W tym celu należy m.in. odpowiednio zmodyfikować definicje macierzy `P` oraz `T`. Zbadać wpływ zmiany struktury modelu na jakość prognozowania.

ĆWICZENIE 2

ZASTOSOWANIE WARSTWY PERCEPTRONOWEJ W ZADANIU KLASYFIKACJI – ROZPOZNAWANIE ZNAKÓW

Celem ćwiczenia jest zastosowanie sieci neuronowej, złożonej z pojedynczej warstwy *perceptronów*, czyli neuronów o progowej funkcji aktywacji, w zadaniu klasyfikacji (rozpoznawania) wzorców w postaci map bitowych, zawierających obrazy wybranych liter alfabetu. Przed wykonaniem ćwiczenia należy przypomnieć sobie wiadomości dotyczące sieci perceptronowych.



Rys. 2.1. Struktura jednowarstwowego klasyfikatora perceptronowego

W pierwszej części ćwiczenia utworzymy kilka niewielkich map bitowych (o rozdzielczości 7×5 pikseli), zawierających obrazy liter, przydzielonych poszczególnym grupom laboratoryjnym:

a) A, B, C, D;
b) E, F, G, H;
c) I, J, K, L;
d) N, O, P, R;
e) S, T, U, V;
f) W, X, Y, Z.

Przygotujemy także wzorce wyjściowe, reprezentujące poprawne odpowiedzi sieci na kolejne wzorce wejściowe (rys. 2.1). Następnie utworzymy sieć neuronową złożoną z pojedynczej warstwy perceptronów. Liczba wejść tej sieci będzie odpowiadać liczbie pikseli w obrazie (35), zaś liczba jej wyjść (czyli neuronów w warstwie) – liczbie liter, które sieć ma rozpoznawać (4).

Kolejnym etapem będzie przeprowadzenie procesu uczenia sieci *metodą perceptronową* z wykorzystaniem przygotowanych wzorców uczących. Po nauczeniu sieci zweryfikujemy jej działanie, podając na jej wejścia obrazy wzorcowe – poprawnie nauczona sieć powinna rozpoznawać je jako odpowiednie litery. W ostatnim etapie sprawdzimy, czy sieć nauczona na obrazach wzorcowych będzie w stanie poprawnie rozpoznawać również obrazy zniekształcone, tzn. takie, w których zmienione zostaną wartości losowo wybranych pikseli.

Czynności wstępne

1. Zapoznać się z działaniem dołączonych do biblioteki *Neural Network Toolbox* skryptów przykładowych, ilustrujących działanie pojedynczego perceptronu:

`nnd4db` – ilustracja działania pojedynczego perceptronu o dwóch wejściach – granica decyzyjna ma tu postać linii prostej; współczynniki wagowe neuronu dobieramy ręcznie.

`nnd4pr` – ilustracja działania pojedynczego perceptronu o dwóch wejściach – współczynniki wagowe wyznaczane są tym razem metodą uczenia, z zastosowaniem tzw. *reguły perceptronowej*.

Dokonać edycji (poleceniem `edit nazwa_skryptu`) oraz zapoznać się z zawartością następujących skryptów demonstracyjnych:

`demop1` – klasyfikacja pięciu wzorców (cztery z nich to wzorce uczące) z zastosowaniem dwuwejściowego perceptronu.

`demop6` – zadanie jak wyżej, ale wzorce wejściowe są nieseparowalne liniowo.

Uruchomić skrypty, przeanalizować ich działanie.

2. Zapoznać się z przeznaczeniem oraz składnią funkcji biblioteki *Neural Network Toolbox*, które będziemy wykorzystywać w tym ćwiczeniu. W tym celu w linii poleceń *Matlaba* należy wpisać słowo kluczowe `help`, a po nim nazwę funkcji, której opis chcemy uzyskać:

`newp` – tworzenie warstwy perceptronów, czyli neuronów o progowej funkcji aktywacji, oraz inicjalizacja jej współczynników wagowych wartościami zerowymi.

`train` – uczenie dowolnej jednokierunkowej sieci neuronowej, (także sieci perceptronowej) – czyli obliczanie wartości współczynników wagowych metodą wsadową. Wagi są modyfikowane po prezentacji wszystkich wzorców.

`adapt` – uczenie dowolnej jednokierunkowej sieci neuronowej (także sieci perceptronowej) – czyli obliczanie wartości współczynników wagowych metodą adaptacyjną. Wagi są modyfikowane każdorazowo po prezentacji pojedynczego wzorca.

`sim` – funkcja służąca do symulacji działania dowolnej sieci neuronowej (także sieci perceptronowej).

Uwaga: funkcja `sim` jest funkcją przeciążoną – w celu uzyskania informacji o funkcji dotyczącej sieci neurono-

wych, w linii poleceń *Matlaba* należy wpisać polecenie: `help network/sim`.

Przebieg ćwiczenia

Uwaga: Wszystkie kolejne polecenia będziemy zapisywać, podobnie jak w poprzednim ćwiczeniu, w pliku skryptowym *Matlaba*.

1. Otworzyć nowe okno edytora *m-plików* (*File/New/M-File*), zapisać w nim polecenia `close all` oraz `clear`. Dzięki nim, w momencie uruchomienia skryptu zamknięte zostaną wszystkie otwarte poprzednio okna graficzne oraz usunięte zostaną z przestrzeni roboczej *Matlaba* wszystkie zmienne.

2. Zdefiniować cztery macierze o wymiarach 7×5, reprezentujące mapy bitowe określonych liter. Zgaszone piksele należy zakodować jako -1, zaś pikselom zapalonym przypisać wartość 1. Zapisać i uruchomić skrypt. Sprawdzić rozmiary macierzy i wyświetlić ich elementy (`whos`, `disp`).

3. Korzystając z funkcji `hintonw` zrealizować wizualizację liter. W tym celu pod definicjami macierzy dopisać dla każdej litery następujące polecenia:

```
figure(nr_okna)
hintonw(nazwa_macierzy)
```

Zapisać i uruchomić skrypt. Jeśli nie jesteśmy zadowoleni z wyglądu liter, możemy wprowadzić odpowiednie poprawki do skryptu.

4. Wszystkie wzorce wejściowe (obrazy liter) powinny przed rozpoczęciem procesu uczenia sieci zostać zgromadzone w jednej macierzy `P`. Poszczególne wzorce wejściowe powinny znaleźć się w kolejnych kolumnach tej macierzy (rys. 2.1).

Uwaga: Najwygodniej, bez konieczności żmudnego wpisywania po raz kolejny wartości wszystkich pikseli, skorzystać można w tym celu z funkcji `reshape`. Funkcja ta z elementów podanej macierzy tworzy nową

macierz o zmienionych rozmiarach. Zapoznać się odpowiednim opisem w pomocy *Matlaba* (np. klawisz *F1 / index*).

Z każdego wzorca o wymiarach 7×5 należy zatem utworzyć wektor kolumnowy złożony z 35 elementów. Wektory te, ułożone obok siebie, utworzą macierz `P`. Korzystając z funkcji `reshape` zdefiniować tę macierz. Zapisać i uruchomić skrypt, sprawdzić czy macierz znajduje się w pamięci *Matlaba* i jakie są jej rozmiary. Wyświetlić jej zawartość:

```
whos
disp(P)
```

Jeśli wymiary (lub zawartość) macierzy P nie są właściwe, poprawić w skrypcie jej definicję.

5. Zdefiniować odpowiednią macierz wzorców wyjściowych `T`, zawierającą poprawne odpowiedzi sieci na kolejne wzorce wejściowe (rys. 2.1).

W tym celu zakładamy, że każdy neuron będzie odpowiadać za jeden znak. Wartość 1 na wyjściu tego neuronu będzie oznaczała, że rozpoznaje on „swoją" literę na wejściu. Pozostałe neurony powinny mieć w tym czasie na swoich wyjściach wartość 0.

Uwaga: każdemu wzorcowi wejściowemu (tzn. znakowi zakodowanemu w danej kolumnie macierzy `P`) powinien odpowiadać jeden wzorzec wyjściowy (tzn. wektor odpowiedzi sieci zakodowany w odpowiedniej kolumnie macierzy `T`). Elementy macierzy `T` możemy zdefiniować ręcznie lub wykorzystać w tym celu funkcję `eye`, generującą macierz jednostkową (`help eye`).

Uruchomić skrypt, sprawdzić czy powstała macierz `T` i czy jej rozmiary oraz wartości elementów są poprawne.

6. Utworzyć warstwę perceptronów o strukturze z rys. 2.1, wykorzystując w tym celu funkcję `newp` biblioteki *NNT* (`help newp`). Wywołać funkcję, pamiętając o przekazaniu odpowiednich argumentów oraz

o przypisaniu wartości zwracanej przez funkcję zmiennej `net`, reprezentującej utworzoną sieć.

7. Dopisać do skryptu instrukcje, które po inicjalizacji wyświetlą rozmiary i zawartość macierzy współczynników wagowych oraz wektora współczynników progowych:

```
disp('Rozmiary macierzy wag: ')
disp(net.IW)
disp('Zawartość macierzy wag: ')
disp(net.IW{1})
disp('Rozmiar wektora wsp. progowych: ')
disp(net.b)
disp('Zawartość wektora wsp. progowych: ')
disp(net.b{1})
```

Zapisać i uruchomić skrypt. Sprawdzić rozmiar oraz zawartość macierzy wag oraz wektora współczynników progowych. Oczywiście zerowe wartości wag nie powinny nas niepokoić – nasza sieć została zainicjowana, ale jeszcze nie jest nauczona. Istotne jest natomiast to, aby poprawne były rozmiary macierzy wag i wektora współczynników progowych.

8. Wykorzystując funkcję `train` oraz przygotowane wcześniej wzorce uczące przeprowadzić procedurę uczenia sieci.

Uwaga: Domyślna wartość docelowego błędu (ang. *error goal*) w przypadku sieci perceptronowej wynosi 0 – zazwyczaj żądamy, aby poprawnie klasyfikowała ona wszystkie bez wyjątku wzorce. Natomiast na wypadek, gdyby problem okazał się być liniowo nieseparowalny, przed wywołaniem funkcji `train` ustalmy maksymalną dopuszczalną liczbę cykli treningowych, np.:

```
net.trainParam.epochs = 50;
```

Należy pamiętać, aby funkcja `train` zwracała nowy obiekt reprezentujący sieć neuronową, ze zmodyfikowanymi już współczynnikami wagowymi.

Zapisać i uruchomić skrypt. Jaką wartość błędu osiągnęliśmy? Ile cykli zajęła procedura uczenia sieci? Wyświetlić w oknie *Matlaba* nowe wartości współczynników wagowych oraz progowych (patrz punkt 7).

9. Korzystając z funkcji `sim` przeprowadzić weryfikację działania sieci, podając na jej wejścia w sposób wsadowy wzorce wejściowe (tzn. wszystkie znaki naraz). Wynik działania sieci przypisać zmiennej `Y`. Porównać odpowiedzi sieci na kolejne wzorce wejściowe z odpowiedziami wzorcowymi.

10. Przeprowadzić wizualizację działania sieci. Chcemy, aby po lewej stronie okna graficznego wyświetlał się obraz litery podawanej na wejście sieci, a po jego prawej stronie – odpowiedź sieci w formie wykresu graficznego.

W tym celu należy:
- otworzyć nowe okno graficzne (`figure`),
- umieścić w skrypcie instrukcję pętli `for` (powinna wykonywać się tyle razy, ile mamy obrazów do podania na wejście sieci), a wewnątrz niej:
- umieścić wywołanie funkcji `subplot`, dzielącej okno graficzne na 2 pionowe części; aktywnym uczynić lewe (pierwsze) okno,
- umieścić wywołanie funkcji `reshape`, przekształcające `i`-tą kolumnę macierzy wejściowej `P` z powrotem do postaci macierzy o wymiarach 7×5,
- wyświetlić otrzymaną macierz przy pomocy funkcji `hintonw`,
- przy pomocy funkcji `subplot` aktywnym uczynić prawe (drugie) okno,
- przy pomocy funkcji `hintonw` wyświetlić `i`-tą kolumnę macierzy `Y`, zawierającą odpowiedzi sieci na `i`-ty obraz wejściowy,
- umieścić instrukcję `pause`, zatrzymującą wykonywanie pętli do momentu wciśnięcia klawisza.

11. W kolejnym kroku będziemy chcieli sprawdzić, jak nasza sieć radzi sobie z rozpoznawaniem zniekształconych (zakłóconych) obrazów. W tym celu:

- skopiować definicje macierzy z obrazami wzorcowymi i wkleić je w dolnej części skryptu,
- wprowadzić ręcznie do każdego obrazu po jednym zakłóceniu, zmieniając wartość wybranego piksela z -1 na 1 lub odwrotnie,
- zmienić nazwy macierzy z zakłóconymi obrazami,
- wszystkie obrazy zakłócone umieścić w macierzy `PT` (w taki sam sposób, w jaki obrazy wzorcowe umieszczaliśmy w macierzy `P`),
- sprawdzić, czy sieć poprawnie rozpoznaje zakłócone obrazy, wykonując polecenia analogiczne jak w punkcie 9. oraz 10. (symulacja oraz wizualizacja działania sieci, tym razem dla obrazów zakłóconych).

Uwaga: nie należy ponownie uczyć sieci na obrazach zakłóconych. Naszym zadaniem jest jedynie sprawdzenie, czy sieć nauczona na obrazach wzorcowych radzi sobie z rozpoznawaniem obrazów zakłóconych (generalizacja).

12. Zwiększać stopniowo (do 2, 3, itd.) liczbę przekłamań w obrazach podawanych na wejścia sieci. Obserwować odpowiedzi sieci.

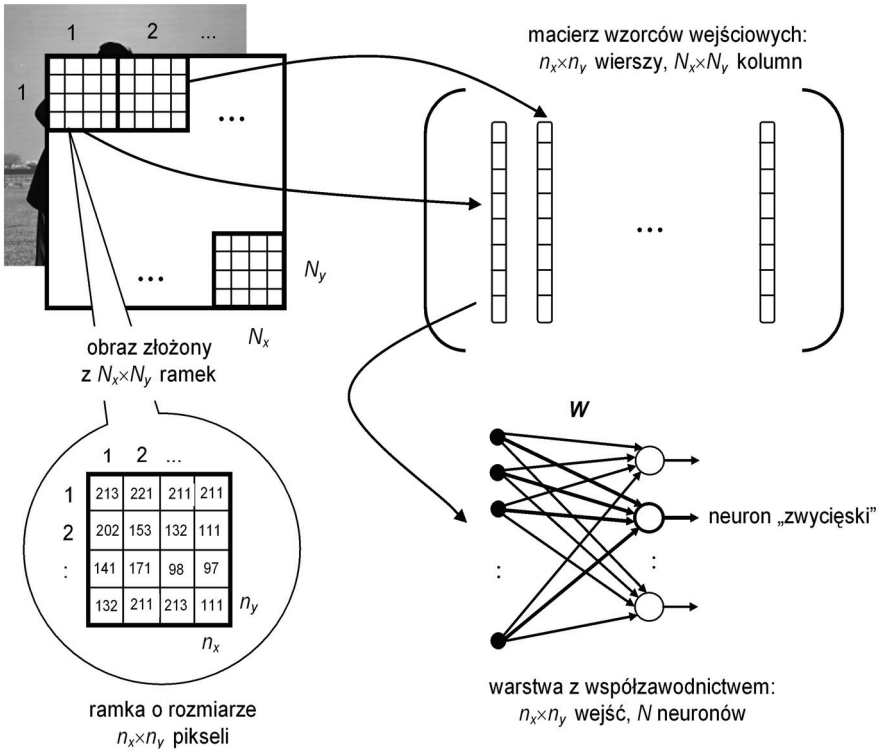
Zadanie dodatkowe:

Przyjęty przez nas sposób kodowania odpowiedzi sieci wymaga, aby warstwa składała się z tylu neuronów, ile liter sieć ma rozpoznawać. Zaproponować inny, wymagający mniejszej liczby neuronów sposób kodowania odpowiedzi sieci. Zmodyfikować skrypt w taki sposób, aby rozpoznawanie odbywało się w oparciu o zaproponowaną metodę kodowania.

ĆWICZENIE 3

ZASTOSOWANIE SIECI NEURONOWEJ KOHONENA DO KOMPRESJI OBRAZÓW

Celem ćwiczenia jest wykorzystanie warstwy neuronów z tzw. *współzawodnictwem*, uczonych metodą Kohonena (która jest metodą *uczenia nienadzorowanego*), w zadaniu kompresji obrazu. Sieć taka jest w stanie realizować *kompresją stratną*, polegającą na zmniejszeniu ilości informacji reprezentującej dany obraz przy zachowaniu błędu odwzorowania na określonym poziomie. Przed wykonaniem ćwiczenia należy przypomnieć sobie wiadomości dotyczące sieci neuronowych z współzawodnictwem, w szczególności zaś sieci Kohonena.



Rys. 3.1. Ilustracja zastosowania sieci z współzawodnictwem w zadaniu kompresji obrazu

W pierwszej części ćwiczenia wczytamy do przestrzeni roboczej *Matlaba* obraz testowy o rozdzielczości 512×512 pikseli, zakodowanych w jednobajtowej skali szarości. Następnie podzielimy go na $N_x \times N_y$ ramek, każda o rozmiarach $n_x \times n_y$ pikseli, oraz utworzymy macierz wzorców wejściowych, złożoną z $N_x \times N_y$ kolumn zawierających wszystkie ramki wchodzące w skład obrazu (rys. 3.1).

W kolejnym kroku utworzymy sieć neuronową Kohonena, złożoną z pojedynczej warstwy N neuronów. Liczba wejść tej sieci odpowiadać będzie liczbie pikseli w ramce ($n_x \times n_y$), zaś liczba jej neuronów – liczbie kategorii, na które sieć ma podzielić wzorce (ramki) wejściowe (np. N=8). Stopień kompresji obrazu zależał będzie przede wszystkim od liczby ramek w obrazie – im mniej ramek, tym większa kompresja, w mniejszym stopniu również od liczby neuronów (im mniej neuronów, tym większy stopień kompresji).

Kolejnym etapem będzie procedura uczenia sieci z wykorzystaniem ramek obrazu wybranych losowo z macierzy wzorców wejściowych. W odróżnieniu od zastosowanej w ćwiczeniu 2. sieci perceptronowej, warstwa neuronów uczonych metodą Kohonena sama (tzn. bez zewnętrznego „nauczyciela", czyli bez wzorców wyjściowych) nauczy się grupować kolejne ramki obrazu w określone kategorie, przy czym każda kategoria reprezentowana będzie przez jeden neuron.

Po nauczeniu sieci zweryfikujemy jej działanie, podając na jej wejścia wszystkie ramki obrazu – także te, które nie były wykorzystane w procesie uczenia. Uzyskamy w ten sposób tzw. *książkę kodową*, określającą dla poszczególnych ramek neurony zwyciężające. Współczynniki wagowe neuronu zwyciężającego dla danej grupy ramek reprezentować będą uśrednioną wartość odpowiednich pikseli, wchodzących w skład tej grupy ramek. Korzystając z książki kodowej oraz z informacji zawartej we współczynnikach wagowych nauczonej sieci, możliwe będzie odtworzenie (dekompresja) obrazu oryginalnego.

W ostatnim etapie sprawdzimy wizualnie jakość obrazu poddanego kompresji oraz obliczymy pewne parametry określające stopień oraz jakość kompresji.

Czynności wstępne

1. Zapoznać się z działaniem wybranych skryptów demonstracyjnych, ilustrujących działanie warstwy neuronów z współzawodnictwem:

`nnd14cl` – ilustracja działania warstwy trzech neuronów z współzawodnictwem, realizujących klasyfikację (klasteryzację) wzorców wejściowych do jednej z trzech kategorii.

`democ1` – klasyfikacja (klasteryzacja) grupy wzorców reprezentowanych przez punkty na płaszczyźnie z zastosowaniem warstwy 8 neuronów z współzawodnictwem. Wyedytować skrypt (`edit democ1`), zapoznać się z jego zawartością. Uruchomić skrypt, przeanalizować jego działanie.

2. Zapoznać się z przeznaczeniem oraz składnią funkcji biblioteki *Neural Network Toolbox* wykorzystywanych w tym ćwiczeniu. W tym celu w linii poleceń *Matlaba* należy wpisać słowo kluczowe `help`, a po nim nazwę odpowiedniej funkcji:

`newc` – tworzenie warstwy neuronów z współzawodnictwem uczonych metodą Kohonena oraz inicjalizacja jej współczynników wagowych.

`train` – uczenie dowolnej jednokierunkowej sieci neuronowej (także warstwy neuronów z współzawodnictwem).

`sim` – funkcja służąca do symulacji działania dowolnej sieci neuronowej – także warstwy neuronów z współzawodnictwem.

Uwaga: funkcja `sim` jest funkcją przeciążoną – w celu uzyskania informacji o funkcji dotyczącej sieci neuronowych, w linii poleceń *Matlaba* należy wpisać polecenie: `help network/sim`.

Przebieg ćwiczenia

1. Z podanej poniżej lokalizacji pobrać przeznaczony dla danej grupy plik graficzny i umieścić go w folderze roboczym *Matlaba*:
   a) http://www.k.bartecki.po.opole.pl/nsi/baboon.tif
   b) http://www.k.bartecki.po.opole.pl/nsi/barbara.tif
   c) http://www.k.bartecki.po.opole.pl/nsi/boat.tif
   d) http://www.k.bartecki.po.opole.pl/nsi/goldhill.tif
   e) http://www.k.bartecki.po.opole.pl/nsi/lena.tif
   f) http://www.k.bartecki.po.opole.pl/nsi/peppers.tif

   Uwaga: W przypadku, gdyby pliki z obrazami były niedostępne pod podanym adresem, można je znaleźć w Internecie – są to klasyczne obrazy testowe wykorzystywane m.in. w badaniach różnych algorytmów kompresji.

2. Otworzyć nowe okno edytora *m-plików* (*File/New/M-File*) i zapisać w nim polecenia: `close all` oraz `clear`.

3. Umieścić w skrypcie wywołanie funkcji `imfinfo`, wyświetlającej informacje o danym pliku graficznym (nie umieszczać średnika na końcu linii).

4. Korzystając z funkcji `imread` napisać w skrypcie polecenie wczytujące zawartość pliku graficznego do zmiennej (macierzy) o nazwie `obraz`.

5. W kolejnym kroku będziemy chcieli wyświetlić wczytany obraz w oknie graficznym. W tym celu umieścić w skrypcie następującą sekwencję poleceń:

```
figure(1)          % otwiera nowe okno graficzne
colormap gray      % obraz w skali szarości
imagesc(obraz,[0 255]) % wyświetla obraz
```

6. Zapisać i uruchomić skrypt.
   - Przeczytać dokładnie informacje o obrazie, wyświetlone przez funkcję `imfinfo`. Jaka jest rozdzielczość obrazu? Ile bitów wykorzysta-

no do zakodowania stopnia szarości pojedynczego piksela? Jaka wartość odpowiada pikselowi w kolorze czarnym?
- Korzystając z polecenia `whos` sprawdzić rozmiar oraz typ elementów macierzy `obraz`. Wyświetlić elementy macierzy w oknie *Matlaba* lub korzystając z narzędzia *Workspace Browser*.
- Korzystając z odpowiedniej ikony na palecie narzędzi okna graficznego powiększyć wybrany fragment obrazu i przyjrzeć się poszczególnym pikselom wchodzącym w jego skład.

7. W kolejnym kroku będziemy chcieli utworzyć tzw. tablicę blokową (`cell`), odzwierciedlającą podział obrazu na ramki o określonym rozmiarze, zgodnie z rys. 3.1. W tym celu należy:
- zdefiniować zmienne `nx` oraz `ny`, reprezentujące rozmiary ramki w pikselach i przypisać im żądane wartości (np. `nx=4;ny=4`),
- korzystając z funkcji size przypisać zmiennym `X` oraz `Y` wartości reprezentujące rozmiary (szerokość i wysokość) obrazu w pikselach,
- w oparciu o znane wartości `X`, `Y`, `nx`, `ny` przypisać zmiennym `Nx`, `Ny` wartości reprezentujące liczbę ramek w obrazie (rys. 3.1),
- korzystając z funkcji mat2cell utworzyć tablicę blokową `A`, złożoną z $N_x \times N_y$ macierzy reprezentujących poszczególne ramki obrazu:

```
A=mat2cell(obraz,repmat(ny,1,Ny),repmat(nx,1,Nx));
```

8. Zapisać i uruchomić skrypt. Korzystając z polecenia whos sprawdzić typ i rozmiar zmiennej `A`. Wyświetlić zawartość zmiennej `A` oraz jej wybranego bloku (ramki), np.:

```
disp(A)
disp(A{1,1})    % lewa górna ramka obrazu
```

9. Korzystając z danych umieszczonych w tablicy `A` utworzyć macierz wzorców wejściowych `P`, zgodnie ze schematem zaprezentowanym na rys. 3.1:

```
P=zeros(nx*ny,Nx*Ny); % tworzymy macierz zer

for i=1:Ny
```

```
  for j=1:Nx
  :             % instrukcja wstawiająca kolejne
                % ramki z tablicy A do kolejnych
                % kolumn macierzy P
  end
end
```

Podpowiedź: w celu zamiany kolejnych ramek tablicy `A` na kolumny macierzy `P` można skorzystać z funkcji `reshape`.

Zapisać i uruchomić skrypt, sprawdzić rozmiar i zawartość macierzy wzorców `P`.

10. W kolejnym kroku utworzymy sieć Kohonena, czyli warstwę neuronów z współzawodnictwem, uczonych bez nauczyciela, o odpowiedniej strukturze. W tym celu należy:

- zdefiniować zmienną `N` określającą liczbę neuronów w warstwie (np. 8) – na tyle kategorii podzielone zostaną wzorce wejściowe, czyli ramki obrazu,
- zdefiniować zmienną `eta`, określającą współczynnik prędkości uczenia sieci, o wartości równej np. 0.1,
- wywołać funkcję `newc`, pamiętając o przekazaniu jej odpowiednich argumentów oraz o przypisaniu wartości zwracanej przez funkcję zmiennej `net` reprezentującej sieć (`help newc`).

11. Dopisać do skryptu instrukcje, które po inicjalizacji warstwy wyświetlą rozmiary i początkowe wartości elementów macierzy współczynników wagowych:

```
disp('Rozmiary macierzy wag: ')
disp(net.IW)
disp('Zawartość macierzy wag: ')
disp(net.IW{1})
```

Zapisać i uruchomić skrypt. Sprawdzić rozmiar macierzy współczynników wagowych. Jakie początkowe wartości zostały przypisane tym współczynnikom?

12. Ponieważ macierz wzorców wejściowych `P` ma stosunkowo duże rozmiary (zawiera $N_x \times N_y$ kolumn reprezentujących wszystkie ramki obrazu), wybierzemy z niej losowo pewną liczbę kolumn (ramek), które pełnić będą rolę wzorców uczących. Wzorce te utworzą macierz wzorców uczących `Pu`.

W tym celu należy:

- utworzyć zmienną `nf`, określającą ile ramek obrazu będzie prezentowanych sieci w trakcie uczenia, np.:

```
nf = Nx*Ny/4;
```

- korzystając z funkcji `randperm` (`help`) utworzyć wektor `Fi`, zawierający `nf` losowych numerów ramek obrazu (kolumn macierzy `P`):

```
r=randperm(Nx*Ny);
Fi=r(1:nf);
```

- sprawdzić rozmiar i wyświetlić zawartość utworzonego wektora `Fi`,
- utworzyć macierz `Pu`, zawierającą wylosowane kolumny macierzy `P`.

  Uwaga: Nie jest konieczne użycie w tym celu instrukcji `for` – wystarczy pojedyncza instrukcja przypisania, wykorzystująca składnię *Matlaba*.

- sprawdzić rozmiar i zawartość macierzy `Pu` – powinna składać się z $n_x \times n_y$ wierszy oraz `nf` kolumn, wybranych losowo z macierzy `P`.

13. Korzystając z funkcji `train` przeprowadzić uczenie sieci w oparciu o przygotowaną w poprzednim punkcie macierz wzorców uczących `Pu`. W odróżnieniu od poprzednich ćwiczeń, tym razem funkcja `train` wymaga podania tylko dwóch argumentów (dlaczego?):

```
net = train(net,Pu)
```

Uwaga: Ponieważ w przypadku sieci uczonej bez nauczyciela nie korzystamy z wzorców wyjściowych, nie określa się również docelowego błę-

du sieci (*error goal*). Jedynym parametrem, którego wartość należy ustalić przed rozpoczęciem uczenia, jest liczba cykli treningowych przeznaczonych na uczenie sieci.

Wykorzystywana przez funkcję `train` funkcja ucząca (`trainr`) w trakcie jednego kroku prezentuje sieci wszystkie wzorce uczące. Zakładając, że w celu nauczenia sieci wystarczy jednokrotna prezentacja każdego z `nf` wzorców uczących, przed wywołaniem funkcji `train` należy dokonać następującego przypisania:

```
net.trainParam.epochs = 1;
```

Zapisać i uruchomić skrypt.

Po nauczeniu sieci wyświetlić w oknie *Matlaba* nowe wartości współczynników wagowych (patrz punkt 11). Przypomnieć sobie, co reprezentują wartości współczynników wagowych poszczególnych neuronów. Któremu z neuronów w trakcie uczenia przyporządkowane zostały np. najciemniejsze ramki obrazu?

14. Przypisać macierz współczynników wagowych (`net.IW{1}`) nauczonej warstwy zmiennej `W`.

15. Przeprowadzić symulację działania sieci (`sim`), podając na jej wejścia w sposób wsadowy wszystkie wzorce wejściowe, czyli ramki składające się na obraz zapisane w macierzy `P`. Wynik działania sieci przypisać zmiennej `a`. Zapisać i uruchomić skrypt. Zaobserwować, które neurony zwyciężają przy prezentacji poszczególnych ramek obrazu.

16. Odpowiedzi sieci zapisane zostały w zmiennej a, reprezentowanej przez macierz o rozmiarach N wierszy na $N_x \times N_y$ kolumn. Zawiera ona tylko po jednej jedynce w każdej kolumnie, określającej odpowiedź zwycięskiego neuronu – pozostałe wartości macierzy to zera. Z tego względu zmienna `a` została zapisana w pamięci *Matlaba* w postaci tzw. macierzy rzadkiej (ang. *sparse array*). Elementy zerowe takiej macierzy nie są zapamiętywane w pamięci, co daje znaczną oszczędność jej wykorzystania.

Korzystając z funkcji `full` dokonać konwersji macierzy `a` do postaci macierzy pełnej. Wynik konwersji zapisać w zmiennej `af`. Zapisać i uruchomić skrypt, porównać ilość pamięci zajmowanej przez obydwie macierze, sprawdzić wartość elementów macierzy `af`.

17. Zapoznać się z działaniem funkcji `vec2ind`. Dokonać konwersji macierzy `a` (lub `af`) do wektora `ac` o długości $N_x \times N_y$, zawierającego numery neuronów zwyciężających przy prezentacji poszczególnych ramek. Wektor ten reprezentować będzie tzw. *książkę kodową*, w oparciu o którą możliwe będzie odtworzenie uśrednionych wartości pikseli wchodzących w skład poszczególnych ramek, zakodowanych w ich współczynnikach wagowych.

18. Sprawdzić (polecenie `whos`) jakiego są typu i ile bajtów zajmują elementy macierzy współczynników wagowych `W`. Korzystając z funkcji `uint8` dokonać ich konwersji do jednobajtowego typu całkowitego bez znaku; wynik zapisać ponownie w zmiennej `W`. W podobny sposób dokonać konwersji typu zmiennej `ac`, reprezentującej książkę kodową.

19. Korzystając z polecenia `save` zapisać w pliku binarnym `obraz.mat` zmienne przechowujące informacje o skompresowanym obrazie, tzn. zmienne: `W`, `ac`, `Nx`, `Ny`, `nx`, `ny`. Po zapisaniu sprawdzić rozmiar pliku `obraz.mat` i porównać go z rozmiarem oryginalnego pliku obrazu.

20. W oparciu o zależność (3.1) obliczyć wartość *współczynnika kompresji* K_r, wyrażonego jako stosunek liczby bitów oryginalnego obrazu do sumy bitów wymaganych do zapamiętania współczynników wagowych sieci oraz książki kodowej:

$$K_r = \frac{N_x N_y n_x n_y p}{N_x N_y k + N n_x n_y m} \tag{3.1}$$

gdzie:

p – liczba bitów przyjętych do reprezentacji stopnia szarości piksela,
k – liczba bitów przyjętych do zapisu elementu książki kodowej,
m – liczba bitów przyjętych do zapisu współczynnika wagowego.

W kolejnych punktach ćwiczenia będziemy chcieli odtworzyć (zdekompresować) skompresowany obraz. W tym celu należy:

21. Otworzyć nowe okno edytora *m-plików*, zapisać w nim polecenia `clear` oraz `close all`. Napisać polecenie `load` wczytujące z pliku `obraz.mat` dane potrzebne do odtworzenia obrazu. Korzystając z odpowiedniej funkcji *Matlaba* przeprowadzić konwersję zmiennych `W` i `ac` do typu rzeczywistego (`double`), zachowując te same nazwy zmiennych.

22. Zapisać i uruchomić skrypt. Korzystając z polecenia `whos` sprawdzić, czy w pamięci *Matlaba* znajdują się wszystkie potrzebne do odtworzenia obrazu zmienne: `W`, `ac`, `Nx`, `Ny`, `nx`, `ny`.

23. Korzystając z funkcji `cell` utworzyć pustą tablicę blokową `Ad` o rozmiarach $N_x \times N_y$ – będziemy do niej wpisywać poszczególne ramki obrazu podlegającego dekompresji.

24. Napisać podwójnie zagnieżdżoną instrukcję pętli `for`, a w niej operację wpisywania odpowiednich wartości pikseli do odpowiedniej składowej (ramki) tablicy `Ad`. Pamiętamy przy tym o następujących założeniach:
- numer neuronu zwyciężającego dla danej ramki jest zapisany w wektorze `ac`,
- współczynniki wagowe tego neuronu reprezentują wartości pikseli tej ramki,
- ponieważ współczynniki wagowe mają postać wektora wierszowego o długości $n_x \times n_y$, przed wpisaniem ich do kolejnego bloku (ramki) tablicy `Ad` należy, korzystając z funkcji `reshape`, przekształcić ten wektor do macierzy o rozmiarach $n_x \times n_y$.

25. Po wpisaniu do tablicy `Ad` wszystkich odtworzonych ramek, przekształcić tę tablicę w „zwykłą” macierz o nazwie `obraz_d`, korzystając w tym celu z funkcji `cell2mat`. Macierz ta reprezentować będzie cały odtworzony obraz, już bez podziału na ramki. Zapisać i uruchomić skrypt, sprawdzić rozmiary i zawartość macierzy `Ad`.

W ostatnich punktach ćwiczenia będziemy chcieli porównać wizualnie obraz odtworzony z obrazem oryginalnym oraz zbadać jakość rekonstrukcji obrazu. W tym celu dodać do skryptu następujące polecenia:

26. Analogicznie jak w punkcie 4., korzystając z funkcji `imread` wczytać zawartość pliku z oryginalnym obrazem do zmiennej `obraz`.

27. Korzystając z funkcji `subplot` wyświetlić obraz oryginalny (`obraz`) oraz odtworzony (`obraz_d`) we wspólnym oknie graficznym, po jego lewej i prawej stronie. Porównać wizualnie jakość obrazu przed kompresją i po dekompresji.

28. Dokonać konwersji zmiennej `obraz` do typu rzeczywistego podwójnej precyzji (`double`). Korzystając z funkcji `sumsqr`, obliczyć w oparciu o zależność (3.2) wartość średniokwadratowego błędu kompresji (ang. *Mean-Square-Error, MSE*):

$$MSE = \frac{1}{X \cdot Y} \sum_{i=1}^{X} \sum_{j=1}^{Y} \left(p(i,j) - p_d(i,j) \right)^2 \tag{3.2}$$

gdzie:

X, Y – rozmiary obrazu (w pikselach),
$p(i,j)$ – wartość piksela w obrazie oryginalnym,
$p_d(i,j)$ – wartość piksela w obrazie odtworzonym,

29. W oparciu o zależność (3.3) obliczyć wartość współczynnika *PSNR* (ang. *Peak Signal-to-Noise Ratio*) reprezentującego szczytowy stosunek sygnału do szumu, określającego w naszym przypadku poziom podobieństwa obrazu oryginalnego z obrazem skompresowanym:

$$PSNR = 10 \cdot \log_{10} \left(\frac{\left(2^p - 1\right)^2}{MSE} \right) \quad [dB] \tag{3.3}$$

gdzie:

p – liczba bitów przyjętych do reprezentacji stopnia szarości piksela.

Czy otrzymana wartość współczynnika *PSNR* jest akceptowalna z punktu widzenia jakości odtworzonego obrazu – jakie są typowe wartości te-

go współczynnika, uzyskiwane dla kompresji obrazów i filmów wideo? Jaką wartość *PSNR* uzyskamy dla dwóch identycznych obrazów? Czy i kiedy *PSNR* może przyjąć wartość zerową?

Zadania dodatkowe:

1. Powtórzyć kompresję dla kilku różnych wartości zmiennej `nf`, określającej liczbę ramek obrazu prezentowanych sieci w trakcie uczenia.

2. Powtórzyć kompresję obrazu dla kilku różnych wartości zmiennej `N`, reprezentującej liczbę neuronów sieci. W razie potrzeby zwiększyć również liczbę cykli treningowych.

3. Powtórzyć kompresję obrazów dla kilku różnych rozmiarów ramek n_x i n_y, np. 2×2, 8×8, 2×8. Dla każdego z przypadków obliczyć wartość współczynnika kompresji (3.1), średniokwadratowego błędu kompresji (3.2) oraz współczynnika *PSNR* (3.3).

4. Zaproponować i zaimplementować metodę neuronowej kompresji obrazów kolorowych, np. zapisanych w formacie mapy bitowej RGB. Odpowiednie kolorowe obrazy testowe: *baboon.bmp*, *lena.bmp* oraz *peppers.bmp* znajdują się w lokalizacji podanej w punkcie 1.

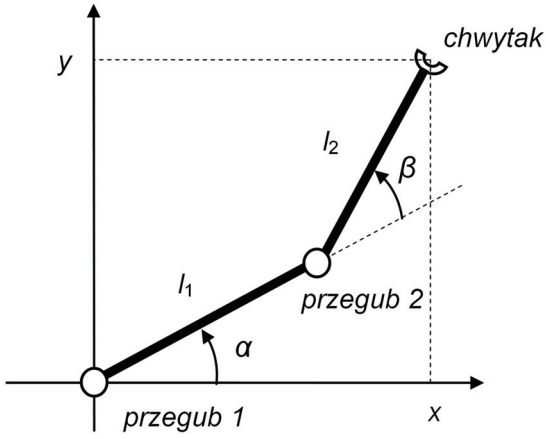
ĆWICZENIE 4

JEDNOKIERUNKOWA SIEĆ NEURONOWA JAKO APROKSYMATOR FUNKCJI – ODWROTNE ZADANIE KINEMATYKI

Celem ćwiczenia jest zastosowanie jednokierunkowej, dwuwarstwowej sieci neuronowej do rozwiązania pewnego zagadnienia z dziedziny robotyki – tzw. *odwrotnego zadania kinematyki*. Zadanie polegać będzie na zastosowaniu sieci neuronowej jako układu sterowania dwuczłonowym *manipulatorem planarnym*, tzn. prostym robotem, którego dwa ramiona operują w obrębie płaszczyzny (rys. 4.1).

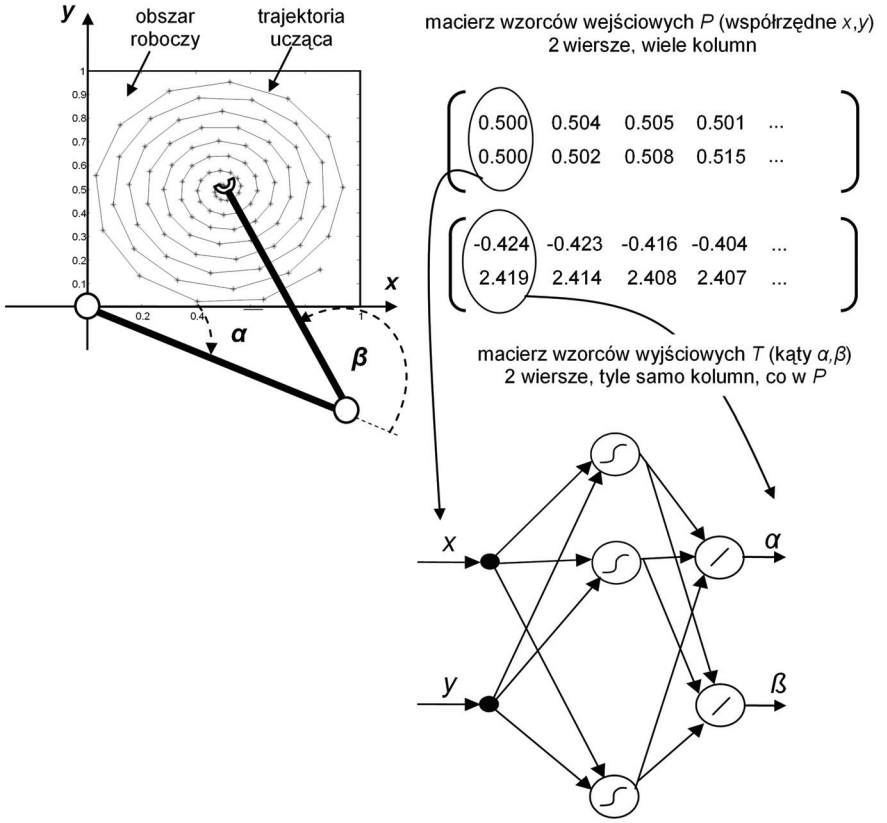
W oparciu o podane przez operatora docelowe współrzędne chwytaka (x,y), neuronowy układ sterujący powinien wyznaczyć takie wartości kątów przegubów α oraz β manipulatora, aby jego chwytak znalazł się dokładnie w punkcie zadanym przez operatora. Operator może podać również całą sekwencję współrzędnych (x,y), odpowiadającą określonej trajektorii chwytaka – układ sterujący powinien wygenerować wówczas odpowiednią sekwencję kątów przegubów (α,β).

Przed wykonaniem ćwiczenia należy przypomnieć sobie wiadomości dotyczące jednokierunkowych, wielowarstwowych sieci neuronowych, zastosowania tych sieci do aproksymacji funkcji, a także metod ich uczenia.



Rys. 4.1. Schemat poglądowy dwuczłonowego manipulatora planarnego

W pierwszej części ćwiczenia zdefiniujemy samodzielnie dwie funkcje *Matlaba*, rozwiązujące proste oraz odwrotne zadanie kinematyki. Następnie przygotujemy dane uczące, reprezentujące współrzędne punktów (*x,y*) na płaszczyźnie (wzorce wejściowe) oraz odpowiadające im wartości kątów przegubów (*α,β*) manipulatora (wzorce wyjściowe). Wzorcowe wartości współrzędnych oraz kątów wyznaczone zostaną dla pewnej zadanej trajektorii chwytaka, mieszczącej się wewnątrz obszaru roboczego manipulatora. Kolejnym etapem będzie utworzenie jednokierunkowej sieci neuronowej o dwóch wejściach i dwóch wyjściach, złożonej z dwóch warstw neuronów. Pierwsza warstwa składać będzie się z neuronów o nieliniowej funkcji aktywacji, zaś warstwa wyjściowa – z neuronów liniowych (rys. 4.2).



Rys. 4.2. Struktura jednokierunkowej nieliniowej sieci neuronowej do aproksymacji odwrotnego zadania kinematyki

Następnie przeprowadzimy proces uczenia sieci z wykorzystaniem przygotowanych wcześniej wzorców uczących. Po nauczeniu sieci zweryfikujemy jej działanie, podając na jej wejścia zadane wartości współrzędnych (x,y) chwytaka, odpowiadające trajektorii uczącej. Wygenerowane przez sieć wartości kątów (α,β) przegubów skonfrontujemy z wartościami wzorcowymi. Sprawdzimy także, jak wyglądałaby trajektoria chwytaka, gdyby do sterowania przegubami manipulatora wykorzystane zostały wartości kątów wygenerowane przez sieć neuronową; trajektorię tę również porównamy z trajektorią wzorcową.

W ostatnim etapie zbadamy, czy sieć nauczona na podstawie wzorców wchodzących w skład trajektorii uczącej będzie w stanie poprawnie wyznaczać wartości kątów przegubów dla punktów rozmieszczonych na trajektorii innej niż wzorcowa, leżącej w granicach obszaru roboczego.

Czynności wstępne

1. Zapoznać się z działaniem dołączonych do biblioteki *Neural Network Toolbox* skryptów przykładowych, ilustrujących działanie jednokierunkowej nieliniowej sieci neuronowej. W tym celu należy:
- uruchomić program *Matlab*,
- uruchomić odpowiedni skrypt demonstracyjny, wpisując w linii poleceń jego nazwę oraz wciskając klawisz *Enter*:

`nnd11nf` – ilustracja działania dwuwarstwowej sieci neuronowej z pojedynczym wejściem i wyjściem oraz dwoma nieliniowymi neuronami w warstwie pierwszej (ukrytej).
`nnd11gn` – neuronowa aproksymacja funkcji; możliwość zmiany postaci aproksymowanej funkcji oraz liczby nieliniowych neuronów w pierwszej (ukrytej) warstwie sieci.
`nnd11bc` – ilustracja uczenia sieci jednokierunkowej z zastosowaniem metody wstecznej propagacji błędu.

2. Zapoznać się z przeznaczeniem oraz składnią funkcji biblioteki *Neural Network Toolbox*, które będziemy wykorzystywać w tym ćwiczeniu.

W tym celu w linii poleceń *Matlaba* należy wpisać słowo kluczowe `help`, a po nim nazwę odpowiedniej funkcji:

`newff` – inicjalizacja wielowarstwowej sieci jednokierunkowej (początkowe wartości współczynników wagowych – losowe).

`train` – uczenie dowolnej jednokierunkowej sieci neuronowej (także jednokierunkowej sieci wielowarstwowej).

`sim` – symulacja działania dowolnej sieci neuronowej (także sieci jednokierunkowej).

Uwaga: funkcja ta jest funkcją przeciążoną – w celu uzyskania informacji o funkcji dotyczącej sieci neuronowych, w linii poleceń *Matlaba* należy wpisać polecenie: `help network/sim`.

Przebieg ćwiczenia

W pierwszym kroku zdefiniujemy dwie nowe funkcje *Matlaba*, służące do rozwiązywania tzw. prostego i odwrotnego zadania kinematyki. Proste zadanie kinematyki polega na wyznaczeniu współrzędnych (x,y) chwytaka, gdy znane są wartości kątów (α,β) przegubów. Odpowiednie zależności dla prostego zadania kinematyki mają następującą postać (oznaczenia – jak na rys. 4.1):

$$\begin{aligned} x &= l_1 \cos\alpha + l_2 \cos(\alpha+\beta) \\ y &= l_1 \sin\alpha + l_2 \sin(\alpha+\beta) \end{aligned} \tag{4.1}$$

zaś dla zadania odwrotnego:

$$\begin{aligned} \alpha &= \operatorname{atan2}(y,x) - \operatorname{atan2}(k_2,k_1) \\ \beta &= \operatorname{atan2}(s_2,c_2) \end{aligned} \tag{4.2}$$

gdzie:

$$
\begin{aligned}
c_2 &= \frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2} \\
s_2 &= \sqrt{1 - c_2^2} \\
k_1 &= l_1 + l_2 c_2 \\
k_2 &= l_2 s_2
\end{aligned}
\tag{4.3}
$$

zaś *atan2* jest dwuargumentową funkcją *arcus tangens*, w której znaki argumentów są wykorzystywane do wyznaczenia ćwiartki układu współrzędnych, do której należy wynik.

W tym celu należy:

1. Otworzyć nowe okno edytora *m-plików* (*File/New/M-File*) i zapisać w nim nagłówek funkcji *Matlaba*, która będzie rozwiązywać proste zadanie kinematyki (4.1):

```
function [x,y] = prostkin(alfa,beta,l1,l2)
```

2. Zdefiniować ciało funkcji `prostkin`, składające się z dwóch instrukcji przypisania, wyznaczających wartości współrzędnych *x* oraz *y* chwytaka zgodnie z zależnością (4.1). Należy pamiętać o umieszczaniu średników na końcu znajdujących się w ciele funkcji instrukcji przypisania – tak, aby w trakcie ich wykonywania nie wyświetlały się w oknie *Matlaba* wyniki tych przypisań.

3. Bezpośrednio pod nagłówkiem funkcji (a przed instrukcjami przypisania) umieścić kilka wierszy komentarza – może być to np. informacja na temat przeznaczenia oraz składni funkcji. Linie te wyświetlane będą jako pomoc funkcji (po wpisaniu w oknie *Matlaba* polecenia `help prostkin`).

4. Zapisać skrypt funkcyjny w katalogu roboczym *Matlaba*, w pliku o nazwie takiej samej jak nazwa funkcji (`prostkin.m`) .

5. W analogiczny sposób utworzyć nowy skrypt z funkcją `odwrkin`, realizującą odwrotne zadanie kinematyki, zgodnie z zależnościami (4.2) i (4.3).

Uwaga:

- Argumentami (parametrami) funkcji `odwrkin` powinny być współrzędne chwytaka (x,y) oraz długości ramion l_1 i l_2, zaś wartościami zwracanymi przez funkcję – odpowiednie kąty przegubów (α,$ß$).
- Zwrócić uwagę na kolejność instrukcji przypisania w ciele funkcji `odwrkin` – najpierw wyznaczamy wartość parametru c_2, później s_2, k_1 i k_2, na końcu zaś wartości kątów α oraz $ß$. Zwrócić uwagę również na priorytet operatorów – konieczne może być użycie nawiasów.
- Operator potęgowania w *Matlabie* to ^ (*daszek*).
- Jako argumenty funkcji `odwrkin` będziemy chcieli przekazywać jednocześnie współrzędne wielu punktów, np. leżących na pewnej trajektorii. Oznacza to, że funkcja `odwrkin` dla argumentów wejściowych podanych w postaci wektorów współrzędnych `x` oraz `y` będzie wyznaczała odpowiednie wektory kątów `alfa` oraz `beta`. Dlatego przed operatorami potęgowania wektorów: `x`, `y` oraz `c2` należy umieścić znak kropki (czyli zastosować operator tzw. potęgowania tablicowego: .^).

6. Po zapisaniu skryptu `odwrkin.m` przejść do okna poleceń *Matlaba* i sprawdzić, czy działa pomoc funkcji (`help prostkin`, `help odwrkin`).

7. Przetestować w oknie *Matlaba* działanie funkcji `prostkin`, wyznaczając współrzędne położenia chwytaka x, y dla określonych długości ramion oraz kątów przegubów, np. dla l_1=l_2=1 m, α=0 rad, $ß$=π/2 rad. W tym celu najpierw naszkicować na kartce położenie ramion manipulatora dla podanych kątów i zastanowić się, w jakim punkcie znajdzie się chwytak. Następnie wywołać funkcję `prostkin` dla podanych wartości argumentów (stała `pi` jest w *Matlabie* predefiniowana) i sprawdzić poprawność otrzymanego wyniku:

```
[x,y]=prostkin(0,pi/2,1,1)
```

Sprawdzić także, czy funkcja `prostkin` działa dla argumentów wektorowych (tzn. dla wielu kątów jednocześnie). W tym celu obliczyć współrzędne chwytaka np. dla : (α_1=0 rad, β_1=0 rad) oraz (α_2=π/2 rad, β_2=0 rad):

```
alfa = [0 pi/2]
beta = [0 0]
[x,y]=prostkin(alfa,beta,1,1)
```

8. W analogiczny sposób przetestować działanie funkcji `odwrkin`. W wypadku błędów poprawić zawartość odpowiednich *m-plików* funkcyjnych.

Uwaga: Sprawdzenie poprawności działania obydwu funkcji jest **konieczne** przed przystąpieniem do dalszych punktów ćwiczenia.

W kolejnych krokach utworzymy wzorce uczące, w oparciu o które sieć nauczy się obliczać wartości kątów (α,β) przegubów dla określonych współrzędnych (x,y) chwytaka. Przedstawiona na rys. 4.2 trajektoria ucząca chwytaka będzie składała się ze 100 punktów rozmieszczonych na spirali Archimedesa o równaniu $r = 0.01\varphi$ (w układzie biegunowym: r – promień wodzący spirali, φ – kąt, jaki tworzy wektor wodzący spirali z osią X).

W ćwiczeniu przyjmiemy jednakowe długości ramion manipulatora: l_1=l_2=1 m. Zadania dla poszczególnych grup będą różniły się wartościami współrzędnych (x_{sr},y_{sr}) punktu środkowego spirali:

a) x_{sr} = 0.5, y_{sr} = 0.5;
b) x_{sr} = 0, y_{sr} = 0.7;
c) x_{sr} = -0.5, y_{sr} = 0.5;
d) x_{sr} = -0.5, y_{sr} = -0.5;
e) x_{sr} = 0, y_{sr} = -0.7;
f) x_{sr} = 0.5, y_{sr} = -0.5.

9. Otworzyć nowe okno edytora *m-plików* (*File/New/M-File*) i umieścić w skrypcie polecenia `close all` oraz `clear`.

10. Utworzyć zmienne `l1` oraz `l2` reprezentujące długości ramion manipulatora, przypisując im wartość 1.

11. Utworzyć zmienną `fi` reprezentującą 100-elementowy wektor kątów, odpowiadający położeniu kolejnych punktów na spirali: od φ=0 rad do φ=49.5 rad z krokiem co 0.5 rad.

12. Zgodnie z podanym równaniem trajektorii spiralnej (r=0.01φ) utworzyć 100-elementowy wektor `r`, reprezentujący długości promieni kolejnych jej punktów. Zapisać i uruchomić skrypt, sprawdzić rozmiar i wartości elementów wektorów `fi` oraz `r`.

13. Dokonać konwersji współrzędnych biegunowych (r,φ) na kartezjańskie (x,y). Wykorzystać w tym celu funkcję *Matlaba* `pol2cart` (`help`). W wyniku powinniśmy otrzymać dwa 100-elementowe wektory: `x` oraz `y`, zawierające współrzędne kolejnych punktów spirali.

14. Narysować wykres punktów położonych na spirali. Przybliżony kształt spirali można uzyskać, łącząc w wywołaniu funkcji plot kolejne punkty odcinkami:

```
plot(x,y,'r*-')
```

Dołączyć do wykresu opisy osi oraz tytuł (`xlabel`, `ylabel`, `title`).

15. W jakim punkcie znajduje się środek spirali? Przed wywołaniem funkcji `plot` umieścić dwie instrukcje modyfikujące wartości współrzędnych x oraz y w taki sposób, aby środek spirali znalazł się w punkcie o współrzędnych (x_{sr},y_{sr}) przydzielonych przez prowadzącego.

16. Współrzędne (x,y) kolejnych punktów spirali stanowić będą w procedurze uczenia sieci wzorce wejściowe. Jednak sieć uczona metodą „z nauczycielem" wymaga określenia również wzorców wyjściowych – w naszym przypadku rolę tę pełnią odpowiednie wartości kątów przegubów α oraz β.

W tym celu zdefiniować wektory kątów `alfa` oraz `beta` (jaką funkcję wykorzystamy w tym celu?). Następnie sprawdzić rozmiary tych wektorów. Ile wynoszą wartości kątów (α,β) przegubów np. dla pierwszego punktu trajektorii, odpowiadającego środkowi spirali?

17. W nowym oknie graficznym (`figure`) narysować wykres kątów przegubów manipulatora dla kolejnych punktów trajektorii, np:

```
plot(alfa,'r*-')
hold on
plot(beta,'b*-')
```

Dołączyć do wykresu opisy osi, tytuł, legendę, siatkę.

18. Ze strony http://www.k.bartecki.po.opole.pl/nsi/robotanim.m pobrać skrypt funkcyjny i umieścić go w katalogu roboczym *Matlaba*. Zapoznać się z jego zawartością (`edit robotanim`) oraz składnią wywołania funkcji `robotanim`. Wywołać funkcję z linii poleceń *Matlaba*, przekazując odpowiednie argumenty wejściowe. W miarę potrzeb spowolnić lub przyspieszyć animację.

19. Utworzyć macierz wzorców wejściowych sieci `P`, zawierającą kolejne zestawy współrzędnych punktów wchodzących w skład trajektorii uczącej. Tak jak w poprzednich ćwiczeniach, liczba wierszy tej macierzy powinna być równa liczbie wejść sieci, zaś liczba jej kolumn – liczbie wzorców uczących. W każdej kolumnie macierzy `P` znajdą się współrzędne (*x,y*) jednego punktu spirali. Po utworzeniu macierzy sprawdzić jej rozmiary oraz zawartość (rys. 4.2).

20. W analogiczny sposób utworzyć macierz wzorców wyjściowych `T`, zawierającą wzorcowe odpowiedzi sieci (tzn. wartości kątów przegubów dla kolejnych punktów trajektorii). Sprawdzić jej rozmiary oraz wartości jej elementów.

21. Korzystając z funkcji `newff` utworzyć jednokierunkową, dwuwarstwową sieć neuronową o strukturze przedstawionej na rys. 4.2. Na początek pierwsza warstwa sieci niech zawiera tylko jeden neuron o nieliniowej funkcji aktywacji – *tangens hiperboliczny* (`'tansig'`). Później będziemy zwiększać liczbę neuronów w tej warstwie. Druga warstwa będzie natomiast składać się z odpowiedniej liczby neuronów liniowych (`'purelin'`). Wynik działania funkcji przypisać zmiennej `net`.

22. Zapisać i uruchomić skrypt. Sprawdzić rozmiary oraz początkowe wartości elementów macierzy współczynników wagowych:
- pierwszej warstwy (`net.IW{1}`),
- drugiej warstwy (`net.LW{2}`),
- współczynników progowych (`net.b{1}, net.b{2}`).

Przeanalizować rozmiary macierzy wagowych.

23. Przed rozpoczęciem procedury uczenia sieci ustalić odpowiednie jej parametry:
- Algorytm uczenia – w pierwszej kolejności przetestujemy jedną z najprostszych metod uczenia – gradientową metodę największego spadku z adaptacyjną zmianą wartości współczynnika prędkości uczenia oraz z tzw. członem *momentum*:

```
net.trainFcn='traingdx';
```

- docelową wartość funkcji błędu – równą np. 0.01:

```
net.trainParam.goal = 0.01;
```

- Maksymalną liczbę kroków (cykli) treningowych – np. 100:

```
net.trainParam.epochs = 100;
```

Nowsze wersje biblioteki *Neural Network Toolbox* automatycznie przerywają procedurę uczenia sieci również w sytuacji, gdy przez określoną liczbę cykli treningowych wzrasta wartość błędu dla tzw. zbioru danych walidacyjnych, wydzielonych ze zbioru danych uczących. Ma to na celu polepszenie właściwości uogólniania sieci neuronowej. Ponieważ nie będziemy chcieli na razie korzystać z tej funkcji, dodatkowo należy dokonać następującego przypisania:

```
net.trainParam.max_fail = net.trainParam.epochs;
```

24. Wywołać procedurę uczenia sieci (`train`), pamiętając o przypisaniu wartości zwracanej przez funkcję zmiennej `net`.

25. Zapisać i uruchomić skrypt. Przeanalizować wykres zmian wartości funkcji błędu sieci w trakcie uczenia. Odczytać z wykresu uzyskaną wartość funkcji błędu (*'Performance is ... '*). Zastanowić się, czy i w jaki sposób można uzyskać mniejszą wartość tej funkcji. Zmodyfikować odpowiedni parametr (lub parametry) treningu. Metodę uczenia na razie pozostawić bez zmian. Uruchomić skrypt ponownie.

26. Jeśli w trakcie uczenia wartość funkcji błędu ustabilizowała się na pewnym poziomie i maleje już nieznacznie, uczenie można uznać za wystarczające (np. po 2000 cykli). Zanotować uzyskaną wartość funkcji błędu.

27. Umieścić w skrypcie polecenie `save`, zapisujące w pliku `dane_ucz.mat` zmienną net, zmienne `l1`, `l2`, a także macierze z wzorcami wejściowymi (`P`) oraz wyjściowymi (`T`):

```
save dane_ucz net l1 l2 P T
```

Zapisać i uruchomić ponownie skrypt.

W kolejnym kroku przeprowadzimy weryfikację działania nauczonej sieci, porównując generowane przez nią wartości kątów przegubów z wartościami wzorcowymi, zawartymi w danych uczących. Odpowiednie polecenia zapisywać będziemy w nowym (drugim) skrypcie.

28. Otworzyć nowe okno edytora *m-plików* (*File/New/M-File*) i umieścić w skrypcie polecenia `close all` oraz `clear`.

29. Umieścić w skrypcie wywołanie polecenia `load`, wczytujące z pliku dane_ucz.mat zapisane w nim dane (`help load`). Po uruchomieniu skryptu sprawdzić, czy odpowiednie dane zostały wczytane (`whos`).

30. Zasymulować działanie sieci (`sim`), podając na jej wejścia w sposób wsadowy wszystkie współrzędne kolejnych punktów trajektorii wzorcowej (spirali). Odpowiedzi sieci przypisać zmiennej `Y`. Po uruchomieniu skryptu sprawdzić, czy zmienna `Y` znajduje się w przestrzeni roboczej *Matlaba* oraz jakie są jej rozmiary.

31. Macierz `Y` zawierać będzie odpowiedzi sieci, reprezentujące wartości kątów przegubów α oraz β dla kolejnych punktów spirali. W idealnym przypadku (tzn. dla zerowej wartości funkcji błędu sieci) wartości tych kątów byłyby równe ich wartościom wzorcowym, zawartym w macierzy `T`.

Z elementów macierzy `T` wydzielić ponownie zmienne `alfa` i `beta` zawierające wzorcowe wartości kątów przegubów, zaś z elementów macierzy `Y` – zmienne `alfas` i `betas`, reprezentujące wartości tych kątów wygenerowane przez sieć.

32. Utworzyć macierz błędów `E` jako różnicę między wzorcowymi wartościami kątów a wartościami kątów uzyskanych na wyjściach sieci w wyniku symulacji. Następnie wyznaczyć wartość sumarycznego błędu średniokwadratowego sieci *SSE* (czyli sumy kwadratów błędów dla wszystkich *Q* wzorców i dla wszystkich *M* wyjść sieci, podzielonej przez *Q* i *M*):

$$SSE = \frac{1}{Q \cdot M} \sum_{k=1}^{Q} \sum_{m=1}^{M} \left(e_m^{(k)} \right)^2 = \frac{1}{Q \cdot M} \sum_{k=1}^{Q} \sum_{m=1}^{M} \left(t_m^{(k)} - y_m^{(k)} \right)^2 \qquad (4.4)$$

Wykorzystać w tym celu funkcję `sumsqr`, liczącą sumę kwadratów elementów macierzy. Wartość błędu otrzymaną na podstawie zależności (4.4) porównać z wartością uzyskaną na etapie uczenia sieci (punkt 26).

33. Narysować w nowym oknie graficznym wykres porównawczy wzorcowych wartości kątów α i β oraz wartości kątów uzyskanych w wyniku symulacji nauczonej sieci.

Kąty wzorcowe oznaczyć na wykresie kółkami ('o'), kąty generowane przez sieć – plusami ('+'). Wartości kątów α zaznaczyć kolorem czerwonym, zaś wartości kątów β – kolorem niebieskim:

```
figure
plot(alfa,'ro-')    % wzorcowe wartości kątów alfa
hold on
:                % kolejne 3 wywołania funkcji plot
```

Dołączyć do wykresu opisy osi, tytuł oraz legendę.

34. W nowym oknie graficznym narysować wykres błędów aproksymacji neuronowej, osobno dla kąta α (kolorem czerwonym) oraz β (kolorem niebieskim). Skorzystać w tym celu z macierzy błędów `E`, utworzonej w punkcie 32. Dołączyć do wykresu opisy osi, tytuł oraz legendę.

35. W nowym oknie graficznym narysować wykres porównawczy trajektorii wzorcowej (spirali) oraz trajektorii, jaką zakreśliłby chwytak manipulatora, gdyby kąty przegubów równe były kolejnym kątom generowanym przez sieć (tzn. gdybyśmy wykorzystali nauczoną sieć jako układ sterujący kątami przegubów). Z jakiej funkcji skorzystamy w celu obliczenia współrzędnych `xs`, `ys` chwytaka?

```
[xs,ys]= ...  % tu wywołanie odpowiedniej funkcji
figure
x=P(1,:);           % wzorcowe współrzędne x
y=P(2,:);           % wzorcowe współrzędne y
plot(x,y,'r*-')     % trajektoria wzorcowa
hold on
plot(xs,ys,'b*-') % trajektoria wygenerowana
                    % przez sieć neuronową
```

Dołączyć do wykresu opisy osi, tytuł oraz legendę. Czy uzyskana trajektoria jest zbliżona do trajektorii wzorcowej?

36. Z linii poleceń *Matlaba* wywołać funkcję `robotanim`, przekazując jako argumenty wejściowe wartości kątów generowane przez sieć.

W kolejnym kroku będziemy chcieli poprawić jakość działania sieci. Przypomnieć sobie, co może być przyczyną słabej aproksymacji. Oczywiście w naszym przypadku jest nią zbyt uboga struktura sieci, czyli zbyt mała liczba neuronów nieliniowych w pierwszej warstwie.

37. Przejść do pierwszego skryptu, realizującego uczenie sieci, i zwiększyć liczbę nieliniowych neuronów w pierwszej warstwie sieci do dwóch. W razie potrzeby zmodyfikować także docelową wartość błędu (*goal*) oraz maksymalną liczbę cykli treningowych (*epochs*), w taki sposób, aby możliwe było uzyskanie możliwie małej wartości funkcji błędu.

Uwaga: Jeśli wartość funkcji celu w trakcie uczenia nie spada, uczenie można przeprowadzić wielokrotnie. W sprawozdaniu wyjaśnić, dlaczego ponowne uczenie tej samej sieci może dawać za każdym razem inne wyniki.

38. Przeprowadzić weryfikację działania nauczonej sieci o dwóch neuronach nieliniowych, uruchamiając drugi skrypt. Sprawdzić wyniki, porównać jakość aproksymacji.

39. Przejść do skryptu uczącego i zwiększyć liczbę neuronów nieliniowych do czterech, później do ośmiu. W razie potrzeby zmodyfikować także docelową wartość błędu oraz maksymalną liczbę cykli treningowych.

Zmienić algorytm uczenia sieci z gradientowej metody największego spadku (`'traingdx'`) na algorytm Levenberga-Marquardta (`'trainlm'`). W przypadku którego z algorytmów poszukiwanie minimum funkcji błędu daje lepsze rezultaty? Ponownie zweryfikować jakość aproksymacji dla trajektorii wzorcowej (skrypt drugi).

Na koniec sprawdzimy jakość działania sieci dla trajektorii testowej, innej niż trajektoria wykorzystana w procesie jej uczenia. Nasza trajektoria testowa będzie na początek łamaną, złożoną z dwóch odcinków: AB oraz BC. Współrzędne punktów A, B oraz C dla poszczególnych grup są następujące:

a) A(0,1), B(1,0), C(1,1);
b) A(-0.5,1.2), B(0.5,0.2), C(0.5,1.2);
c) A(-1,1), B(0,0), C(0,1);
d) A(-1,0), B(0,-1), C(0,0);
e) A(-0.5,-0.2), B(0.5,-1.2), C(0.5,-0.2);
f) A(0,0), B(1,-1), C(1,0).

Naszkicować na kartce położenie trajektorii testowej w układzie współrzędnych.

40. Otworzyć nowe okno edytora *m-plików* (*File/New/M-File*) i umieścić w nowym (trzecim z kolei) skrypcie polecenia: `close all`, `clear` oraz `load dane_ucz.mat`.

41. Zdefiniować w skrypcie wektory: `xab`, `yab`, `xbc`, `ybc`, reprezentujące współrzędne punktów trajektorii testowej. Zakładamy, że na jeden odcinek trajektorii przypadać będzie 21 punktów.

Przykład definicji wektorów `xab`, `yab` (dla grupy a):

```
xab=[0:0.05:1];
yab=[1:-0.05:0];
:                          % dwa pozostałe wektory
```

Po uruchomieniu skryptu sprawdzić rozmiary wektorów.

42. Zdefiniować wektory współrzędnych łamanej: `xt` oraz `yt`, korzystając ze zdefiniowanych w poprzednim punkcie wektorów jej poszczególnych odcinków:

```
xt = [xab xbc];
yt = [yab ybc];
```

Sprawdzić rozmiary otrzymanych wektorów.

43. W nowym oknie graficznym narysować wykres trajektorii testowej:

```
figure(1)
plot(xt,yt,'ro-')
```

44. Korzystając z sieci neuronowej wyznaczyć wartości kątów przegubów manipulatora dla trajektorii testowej. W tym celu współrzędne punktów trajektorii testowej umieścić w macierzy `PT`, a następnie zasymulować działanie sieci. Odpowiedzi sieci umieścić w macierzy `YT`.

W oparciu o elementy macierzy `YT` utworzyć zmienne `alfast` i `betast`, reprezentujące wygenerowane przez sieć wartości kątów trajektorii testowej.

45. Porównać na wykresie dokładne (wzorcowe) wartości kątów, odpowiadające trajektorii testowej z wartościami kątów uzyskanymi na wyjściu sieci. W tym celu najpierw należy obliczyć wzorcowe wartości kątów przegubów dla tej trajektorii w oparciu o współrzędne xt, yt. Z jakiej funkcji skorzystamy w tym celu?

```
[alfat,betat]=...   % wywołanie odpowiedniej funkcji
```

Po wyznaczeniu dokładnych wartości kątów dla trajektorii testowej, narysować w nowym oknie graficznym wykres porównawczy tych kątów oraz kątów generowanych przez sieć. Przyjąć oznaczenia jak w punkcie 33:

```
figure(2)
plot(alfat,'ro-')  % wzorcowe kąty alfa
                   % trajektorii testowej
hold on
:             % kolejne 3 wywołania funkcji plot
```

Dołączyć do wykresu opisy osi, tytuł oraz legendę.

46. W oknie graficznym (1) narysowaliśmy wykres trajektorii testowej (łamanej). Nanieść na tym samym wykresie trajektorię, po jakiej poruszałby się chwytak, gdyby kąty przegubów manipulatora równe były kątom generowanym przez sieć. Jaką funkcję należy wykorzystać w celu wyznaczenia tych współrzędnych?

```
[xts,yts] = ...     % wywołanie odpowiedniej funkcji
figure(1)
hold on
plot(xts,yts, 'bo-')
```

Dołączyć do wykresu opisy osi, tytuł oraz legendę.

Wyjaśnić ewentualne rozbieżności między trajektorią wzorcową a trajektorią generowaną przez sieć.

47. Rozszerzyć trajektorię testową o dwa kolejne odcinki: CD oraz DA w taki sposób, aby utworzyła ona zamkniętą „klepsydrę". Przetestować działanie sieci dla takiej trajektorii. Przeprowadzić wizualizację ruchu manipulatora na trajektorii testowej, korzystając z funkcji `robotanim`.

Podpowiedź: Współrzędne punktu D dla poszczególnych grup powinny być następujące:
a) D(0,0);
b) D(-0.5,0.2);
c) D(-1,0);
d) D(-1,-1);
e) D(-0.5,-1.2);
f) D(0,-1).

Zadania dodatkowe:

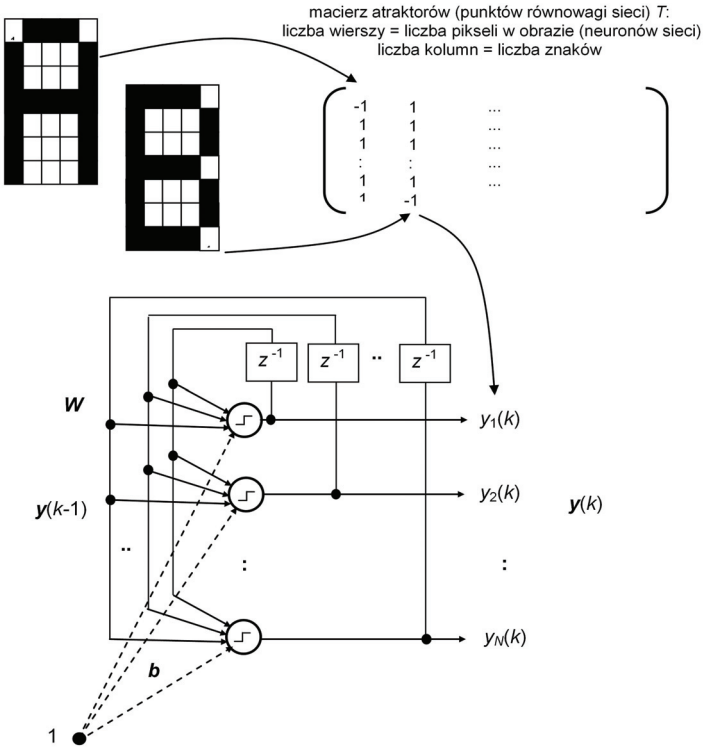
1. Zademonstrować zjawisko nadparametryzacji (ang. *overfitting*) w przypadku sieci wykorzystywanej w ćwiczeniu. Pokazać jego niekorzystny wpływ na właściwości uogólniania sieci na przykładzie trajektorii testowej (klepsydra).

2. Wykorzystując funkcję `surf` utworzyć:
   - dwa wykresy trójwymiarowe ilustrujące proste zadanie kinematyki (4.1): $x=f(\alpha,ß)$ oraz $y=f(\alpha,ß)$ dla $l_1=1$, $l_2=1$ oraz $\alpha\in[0,\pi]$ i $ß\in[0,\pi]$,
   - dwa wykresy trójwymiarowe ilustrujące odwrotne zadanie kinematyki (4.2): $\alpha=f(x,y)$ oraz $ß=f(x,y)$ dla $l_1=1$, $l_2=1$ oraz $x\in[-1,1]$, $y\in[-1,1]$.

ĆWICZENIE 5

REKURENCYJNA SIEĆ HOPFIELDA JAKO PAMIĘĆ SKOJARZENIOWA – REKONSTRUKCJA WZORCÓW ZNAKOWYCH

Celem ćwiczenia jest zastosowanie rekurencyjnej sieci Hopfielda do usuwania zakłóceń z map bitowych, zawierających obrazy wybranych liter alfabetu. Działanie sieci umożliwia rekonstrukcję zniekształconego obrazu, poprzez skojarzenie go z oryginalnym wzorcem, pełniącym rolę tzw. *atraktora* (punktu stabilnego) sieci. Przed wykonaniem ćwiczenia należy przypomnieć sobie wiadomości dotyczące sieci rekurencyjnych, w szczególności zaś – sieci Hopfielda.



Rys. 5.1. Rekurencyjna sieć Hopfielda jako pamięć skojarzeniowa
Oznaczenia: $\boldsymbol{y}(k)$ – wektor sygnałów wyjściowych w k-tym kroku działania sieci; $\boldsymbol{W}$ – macierz współczynników wagowych sieci; $\boldsymbol{b}$ – wektor współczynników progowych; z^{-1} – opóźnienie sygnału o 1 krok

W ćwiczeniu wykorzystane zostaną mapy bitowe, zawierające obrazy liter przydzielonych przez prowadzącego w ćwiczeniu 2. Utworzona zostanie neuronowa sieć Hopfielda, złożona z pojedynczej warstwy neuronów o liniowej, symetrycznej funkcji aktywacji z nasyceniem (`'satlins'`). Liczba neuronów sieci (i jednocześnie jej wyjść) odpowiadać będzie liczbie pikseli w obrazie (N=35). Rolę atraktorów, czyli stanów stabilnych sieci, będą pełniły wektory reprezentujące wzorcowe obrazy liter (rys. 5.1).

W roli stanów początkowych sieci wykorzystane zostaną zniekształcone (zaszumione) obrazy poszczególnych liter. W kolejnych krokach działania sieci powinniśmy zaobserwować zmiany jej stanu w kierunku określonego atraktora, widoczne w oknie graficznym jako stopniowe odtwarzanie oryginalnego wzorca znakowego.

Czynności wstępne

1. Zapoznać się z działaniem dołączonych do biblioteki *Neural Network Toolbox* skryptów przykładowych, ilustrujących działanie sieci Hopfielda. Dokonać ich edycji (poleceniem `edit nazwa_skryptu`) oraz zapoznać się z ich zawartością i działaniem:

`demohop1` – ilustracja działania sieci Hopfielda złożonej z dwóch neuronów, dla której zdefiniowano dwa atraktory.

`demohop2` – przykład identyczny jak powyżej, dodatkowo zademonstrowano istnienie tzw. „fałszywego atraktora", czyli niepożądanego punktu równowagi.

Uruchomić również skrypt przykładowy:

`nnd18db` – wizualizacja tzw. *funkcji energetycznej* (Lapunowa) oraz atraktorów sieci Hopfielda złożonej z dwóch neuronów. Możliwa jest zmiana wartości współczynników wagowych sieci i obserwacja wpływu tych zmian na kształt jej funkcji energetycznej.

2. Zapoznać się z przeznaczeniem oraz składnią funkcji biblioteki *Neural Network Toolbox*, które będziemy wykorzystywać w tym ćwiczeniu:

`newhop` – funkcja tworząca sieć Hopfielda i obliczająca wartości jej współczynników wagowych w oparciu o podaną macierz atraktorów.

`sim` – funkcja służąca do symulacji działania sieci Hopfielda; składnia jest tu nieco inna niż w przypadku innych sieci neuronowych i wyjaśniona zostanie w dalszej części ćwiczenia.

Uwaga: funkcja `sim` jest funkcją przeciążoną – w celu uzyskania informacji o funkcji dotyczącej sieci neuronowych, w linii poleceń *Matlaba* należy wpisać polecenie: `help network/sim`.

Przebieg ćwiczenia

W pierwszej części ćwiczenia utworzymy sieć Hopfielda, w której rolę atraktorów będą pełniły cztery 35-elementowe wektory, reprezentujące wzorcowe obrazy liter w postaci prostych map bitowych.

1. Otworzyć nowe okno edytora *m-plików Matlaba* (*File/New/M-File*), zapisać w nim polecenia `close all` oraz `clear`.

2. Zdefiniować cztery macierze o wymiarach 7×5, reprezentujące mapy bitowe wzorcowych liter (identycznych jak w ćwiczeniu 2).

3. Dokonać wizualizacji obrazów liter. Tym razem wszystkie cztery litery wyświetlić w jednym oknie graficznym, w dwóch wierszach po dwa znaki. Wykorzystać w tym celu funkcje `subplot` oraz `hintonw`.

4. Każdy z obrazów będzie stanowił jeden z atraktorów (punktów przyciągania, stanów stabilnych) sieci Hopfielda. Przed utworzeniem sieci należy zdefiniować macierz atraktorów `T`, przy czym poszczególne atraktory powinny znaleźć się w kolejnych kolumnach tej macierzy. W celu zmiany rozmiarów macierzy wzorcowych można wykorzystać funkcję `reshape`.

5. Zapisać i uruchomić skrypt. Sprawdzić, czy powstała macierz `T` i jakie są jej rozmiary. Wyświetlić jej zawartość. Jeśli wymiary (lub zawartość) macierzy `T` nie są właściwe, poprawić w skrypcie jej definicję.

6. Wykorzystując funkcję `newhop` utworzyć sieć Hopfielda złożoną z 35 neuronów, dla której atraktorami będą nasze wzorce znakowe. Sprawdzić, czy w pamięci *Matlaba* znajduje się zmienna `net`, reprezentująca utworzoną sieć.

7. Sprawdzić rozmiary macierzy współczynników wagowych oraz progowych sieci. Wyświetlić ich zawartość:

```
disp('Rozmiary macierzy wag: ')
disp(net.LW)
disp('Zawartość macierzy wag: ')
disp(net.LW{1})
disp('Rozmiar wektora wsp. progowych: ')
disp(net.b)
disp('Zawartość wektora wsp. progowych: ')
disp(net.b{1})
```

W drugiej części ćwiczenia zbadamy skojarzeniowe właściwości utworzonej sieci Hopfielda. W tym celu sprawdzimy, czy sieć będzie potrafiła odtwarzać oryginalne wzorce liter na podstawie ich zniekształconych (zaszumionych) obrazów.

8. Skopiować definicje macierzy z obrazami wzorcowymi i wkleić je w dolnej części skryptu. Wprowadzić ręcznie do każdego obrazu po jednym zakłóceniu, zmieniając wartość wybranego piksela z -1 na 1 lub odwrotnie. Zmienić nazwy macierzy z zakłóconymi literami. Wszystkie zakłócone obrazy umieścić w macierzy `TZ`, w taki sam sposób, w jaki z obrazów wzorcowych utworzyliśmy macierz `T`.

9. Zasymulować (`sim`) działanie sieci Hopfielda, podając jako warunki początkowe stany reprezentujące zakłócone obrazy. Symulację można przeprowadzić w sposób wsadowy, tzn. przekazując do funkcji całą macierz `TZ`. Składnia funkcji `sim` w rozpatrywanym przypadku jest nieco inna niż w poprzednich ćwiczeniach:

```
Y = sim(net,{LS KS},{},{TZ})
```

gdzie:

- `net` jest zmienną reprezentującą utworzoną w punkcie 6. sieć,
- zmienna `LS` reprezentuje liczbę różnych stanów początkowych, dla których chcemy przeprowadzić symulację (w naszym przypadku mamy po jednym zakłóconym obrazie dla każdego wzorca, zatem `LS`=4),
- zmienna `KS` reprezentuje liczbę kroków czasowych symulacji – na początek przyjmiemy `KS`=5, zakładając, że sieć zdoła w pięciu krokach odtworzyć obrazy wzorcowe, czyli odszukać odpowiedni atraktor.

10. W wyniku wywołania funkcji `sim` otrzymujemy zmienną `Y`, która jest tablicą macierzy (*cell array*). W naszym przypadku będzie się ona składała z 5 macierzy – gdyż tyle kroków działania sieci symulujemy (`KS`=5). Każda z tych macierzy będzie miała rozmiary 35×4 i zawierać będzie informacje o stanach kolejnych wyjść sieci (wiersze) dla poszczególnych stanów początkowych (kolumny).

Po wywołaniu funkcji `sim` sprawdzić, czy powstała w przestrzeni roboczej *Matlaba* zmienna `Y`. Sprawdzić jej typ oraz rozmiary (`whos`). Wyświetlić na ekranie zawartość tablicy oraz kolejne macierze wchodzące w jej skład:

```
disp(Y)
disp(Y{1})
:
disp(Y{5})
```

11. Przeprowadzić wizualizację stanów sieci w kolejnych krokach jej działania. Najpierw powinny wyświetlić się, analogicznie jak w punkcie 3., wszystkie obrazy zakłócone przed podaniem ich do sieci. Po kolejnych naciśnięciach klawisza, w tym samym oknie będziemy chcieli zobaczyć obrazy odpowiadające stanom na wyjściach sieci w kolejnych krokach jej działania.

W tym celu należy:

- dodać nowe okno graficzne (`figure`),
- umieścić w skrypcie instrukcję pętli `for .. end`, wyświetlającą w dwóch wierszach okna graficznego (`subplot(2,2,i)`) cztery zakłócone obrazy przed podaniem ich sieci, czyli kolejne kolumny macierzy `TZ`, przekształcone przy pomocy funkcji `reshape` do macierzy o rozmiarach 7×5,
- pod instrukcją `end` kończącą pętlę umieścić instrukcję `pause`,
- pod spodem umieścić dwie zagnieżdżone instrukcje `for`, wyświetlające w tym samym oknie po cztery obrazy odpowiadające stanom wyjść sieci w bieżącym kroku.

Uwaga: Zewnętrzna instrukcja `for` powinna zliczać kolejne kroki działania sieci, zaś instrukcja wewnętrzna – kolejne wzorce (stany początkowe). Odwołanie do odpowiedniego wektora wyjściowego sieci powinno wyglądać następująco: `Y{k}(:,j)`, gdzie `k` – numer kroku, `j` – numer stanu początkowego). Instrukcję `pause` umieścić tylko raz, w pętli zewnętrznej.

12. Zapisać i uruchomić skrypt. Prześledzić zmiany stanów (odpowiedzi) sieci w kolejnych krokach jej działania. Zastanowić się nad interpretacją rozmiarów poszczególnych kwadratów na wykresach graficznych. Jaki jest związek tych rozmiarów z funkcją aktywacji (`satlins`) rozpatrywanej sieci Hopfielda ?

13. Zwiększać stopniowo (np. do 3, 5, 7, itd.) liczbę zakłóceń w obrazach liter. Obserwować odpowiedzi sieci w kolejnych krokach jej działania – w razie potrzeby zwiększyć ich liczbę (np. `KS=10`). Czy zaobserwowano istnienie tzw. „fałszywych atraktorów" ?

Zadanie dodatkowe:

Obliczyć, zgodnie z poniższą zależnością:

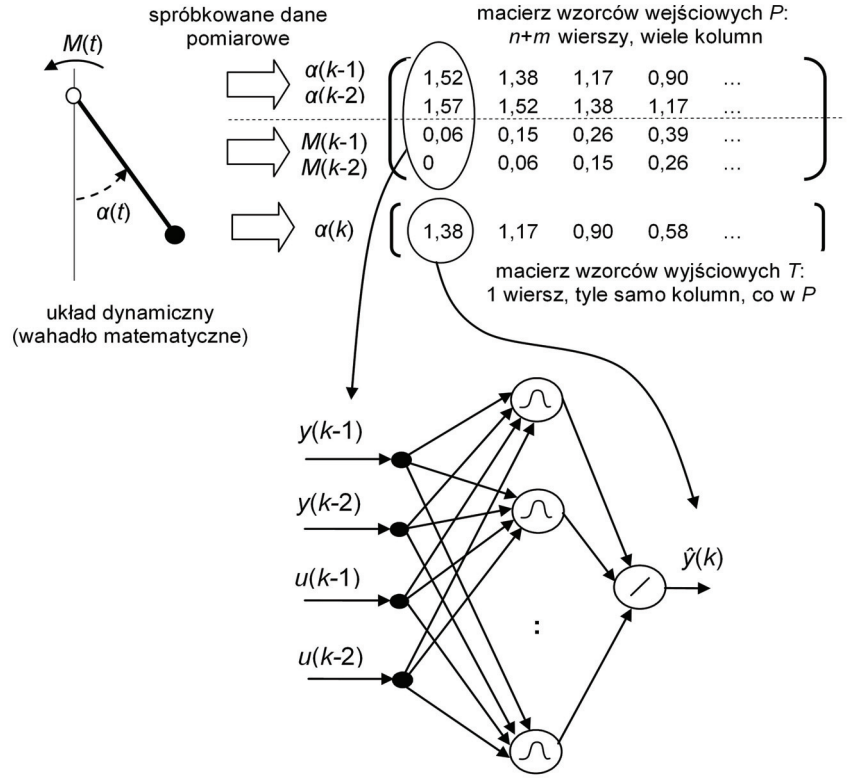
$$E(k) = -\frac{1}{2}\mathbf{y}^T(k)\,\mathbf{W}\mathbf{y}(k) - \mathbf{b}^T\mathbf{y}(k) \tag{5.1}$$

wartość funkcji energetycznej $E(k)$ sieci Hopfielda w każdym kolejnym, k-tym kroku jej działania, dla każdego ze stanów (obrazów) początkowych. Jak zmieniają się wartości tej funkcji w kolejnych krokach? Ile wynoszą jej wartości dla poszczególnych atraktorów? Obliczanie wartości funkcji $E(k)$ można zrealizować wewnątrz instrukcji pętli `for` z punktu 11.

ĆWICZENIE 6

ZASTOSOWANIE SIECI RADIALNEJ W IDENTYFIKACJI NIELINIOWEGO OBIEKTU DYNAMICZNEGO

W ćwiczeniu pierwszym wykorzystaliśmy liniowy neuron w roli *modelu autoregresyjnego*, umożliwiającego prognozę kolejnej wartości ciągu czasowego na podstawie jego poprzednich wartości. W przypadku typowych układów dynamicznych, oprócz poprzednich wartości sygnału wyjściowego, w ich modelach należy uwzględnić również poprzednie wartości sygnału wejściowego (wymuszenia zewnętrznego). W niniejszym ćwiczeniu, w roli nieliniowego, dyskretnego modelu układu dynamicznego wykorzystana zostanie dwuwarstwowa sieć neuronowa z jednostkami o radialnej funkcji aktywacji w pierwszej warstwie i liniowym neuronem wyjściowym (rys. 6.1).



Rys. 6.1. Schemat sieci radialnej realizującej nieliniowy model autoregresyjny z zewnętrznym wymuszeniem dla n=2 i m=2

Zadaniem sieci będzie tu predykcja (prognoza) wartości kolejnej, *k*-tej próbki wielkości wyjściowej *y* układu, na podstawie *n* poprzednich spróbkowanych jej wartości *y*(*k*-1), *y*(*k*-2), ..., *y*(*k*-*n*), oraz na podstawie *m* poprzednich próbek wartości wielkości wejściowej, *u*(*k*-1), *u*(*k*-2), ..., *u*(*k*-*m*). Taki model nazywany jest *modelem autoregresyjnym z zewnętrznym wymuszeniem* (ang. *AutoregRessive with eXogenous input*, ARX). Ogólne równanie liniowego modelu ARX ma następującą postać:

$$\begin{aligned}\hat{y}(k) = &-a_1 \cdot y(k-1) - a_2 \cdot y(k-2) - \mathrm{K} - a_n \cdot y(k-n) + \\ &+ b_1 \cdot u(k-1) + b_2 \cdot u(k-2) + \mathrm{K} + b_m \cdot u(k-m)\end{aligned} \tag{6.1}$$

gdzie *y*(.) oraz *u*(.) reprezentują opóźnione sygnały wyjściowe i wejściowe, a_1, a_2, ..., a_n, b_1, b_2, ..., b_m to parametry modelu, zaś $\hat{y}(k)$ oznacza prognozę kolejnej wartości na wyjściu układu.

Równanie (6.1) można zapisać również w następującej postaci:

$$\begin{aligned}y(k) = &-a_1 \cdot y(k-1) - a_2 \cdot y(k-2) - \mathrm{K} - a_n \cdot y(k-n) + \\ &+ b_1 \cdot u(k-1) + b_2 \cdot u(k-2) + \mathrm{K} + b_m \cdot u(k-m) + e(k)\end{aligned} \tag{6.2}$$

gdzie $e(k) = y(k) - \hat{y}(k)$ jest błędem predykcji dla *k*-tej chwili czasowej.

Opisany zależnością (6.1) liniowy model ARX może zostać zrealizowany, analogicznie jak model autoregresyjny (AR) rozpatrywany w ćwiczeniu 1., w formie pojedynczego neuronu liniowego.

Jednak w przypadku, gdy modelowany układ dynamiczny charakteryzuje się nieliniowością, zastosowanie liniowego modelu ARX może dać niezadowalające rezultaty. Wówczas bardziej uzasadnione będzie wykorzystanie jego nieliniowego odpowiednika (NARX), opisanego zależnością:

$$\hat{y}(k) = f\left(y(k-1), y(k-2), \mathrm{K}, y(k-n), u(k-1), u(k-2), \mathrm{K}, u(k-m)\right) \tag{6.3}$$

gdzie *f*(.) jest pewną funkcją nieliniową.

W naszym ćwiczeniu rolę nieliniowego modelu autoregresyjnego z zewnętrznym wymuszeniem (NARX) pełniła będzie sieć neuronowa radialna. Na jej wejścia podawać będziemy spróbkowane, opóźnione wartości wielkości wyjściowej i wyjściowej układu, zaś prognozowana wartość wielkości wyjściowej układu będzie pojawiać się na wyjściu sieci (rys. 6.1).

Obiektem podlegającym identyfikacji, czyli modelowaniu w oparciu o dane pomiarowe, będzie w ćwiczeniu układ wahadła matematycznego. Rolę sygnału wejściowego pełni zewnętrzny moment obrotowy $M(t)$ przyłożony do wahadła, zaś wielkością wyjściową jest kąt jego wychylenia $\alpha(t)$. Równanie różniczkowe opisujące dynamikę wahadła ma następującą postać:

$$\frac{d^2\alpha(t)}{dt^2} + \frac{k}{ml}\frac{d\alpha(t)}{dt} + \frac{g}{l}\sin\alpha(t) = \frac{1}{ml^2}M(t) \tag{6.4}$$

gdzie:

l – długość wahadła [m],
m – masa wahadła [kg],
g – przyspieszenie grawitacyjne [m/s^2],
k – współczynnik tarcia.

W pierwszej części ćwiczenia, na podstawie równania różniczkowego (6.4), zbudowany zostanie z wykorzystaniem bloków biblioteki *Matlab/Simulink* model symulacyjny obiektu. Model ten posłuży do przeprowadzenia eksperymentu identyfikacyjnego, w trakcie którego zebrane zostaną dane pomiarowe. W oparciu o te dane utworzone zostaną dwa modele, reprezentujące odpowiednio: opisany zależnością (6.1) liniowy model autoregresyjny, zrealizowany w formie liniowego neuronu, oraz nieliniowy model NARX (6.3), zbudowany w oparciu o radialną sieć neuronową. Następnie obydwa te modele poddane zostaną weryfikacji – najpierw przy użyciu sygnału wejściowego wykorzystanego w trakcie eksperymentu identyfikacyjnego, a następnie również dla innych sygnałów wejściowych.

Czynności wstępne

1. Zapoznać się z dołączonymi do biblioteki *Neural Network Toolbox* skryptami demonstracyjnymi, ilustrującymi działanie sieci radialnych. W tym celu należy uruchomić odpowiedni skrypt, wpisując w linii poleceń *Matlaba* nazwę skryptu oraz wciskając klawisz *Enter*. Zawartość skryptów można edytować poleceniem `edit nazwa_skryptu`.

`demorb1` – dwuwarstwowa, nieliniowa sieć radialna (pierwsza warstwa radialna, druga – liniowa) realizuje aproksymację funkcji na podstawie zbioru złożonego z dwudziestu jeden wzorców uczących.

`demorb3` – jak w `demorb1`, jednak ze względu na zbyt małą wartość parametru funkcji bazowej, konieczne jest użycie dużej liczby neuronów, co skutkuje nadparametryzacją i słabymi właściwościami uogólniania sieci.

`demorb4` – zadanie jak w `demorb1`, jednak ze względu na zbyt dużą wartość parametru funkcji bazowej, wszystkie neurony, niezależnie od wartości sygnału wejściowego, mają jednakową wartość na wyjściu – aproksymacja nie jest poprawna.

2. Zapoznać się z przeznaczeniem oraz składnią funkcji biblioteki *Neural Network Toolbox*, które będziemy wykorzystywać w tym ćwiczeniu. W tym celu w linii poleceń *Matlaba* należy wpisać słowo kluczowe `help`, a po nim nazwę funkcji, której opis chcemy uzyskać:

`newrb` – funkcja tworząca sieć radialną i obliczająca wartości jej współczynników wagowych w taki sposób, aby nieliniowe odwzorowanie zbioru wzorców wejściowych P w zbiór wzorców wyjściowych T realizowane było z jak najmniejszym (zadanym) błędem średniokwadratowym.

`sim` – funkcja służąca do symulacji działania dowolnej sieci neuronowej, także sieci radialnej.

Uwaga: funkcja `sim` jest funkcją przeciążoną – w celu uzyskania informacji o funkcji dotyczącej sieci neuronowych, w linii poleceń *Matlaba* należy wpisać polecenie: `help network/sim`.

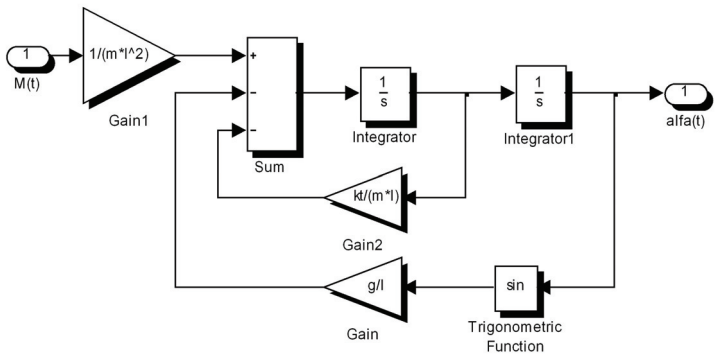
`gensim` – funkcja generuje blok biblioteki *Matlab*/*Simulink*, reprezentujący podaną jako argument wywołania sieć neuronową.

Przebieg ćwiczenia

1. Wpisując w linii komend *Matlaba* polecenie `simulink`, otworzyć okno z biblioteką bloków graficznych. Wybierając z górnego menu *Simulinka* polecenie *New*/*Model*, otworzyć nowe okno graficzne, w którym będziemy tworzyć model symulacyjny wahadła.

2. Korzystając z odpowiednich bloków *Simulinka*, na podstawie równania różniczkowego (6.4) zbudować model symulacyjny wahadła jak na rys. 6.2.

3. Do bloków wzmacniaczy (*Gain*) wpisać odpowiednie wyrażenia, zaś do bloków układów całkujących – wprowadzić zmienne reprezentujące warunki początkowe: `alfa0` (kąt początkowy α_0 – w bloku *Integrator1*) oraz `omega0` (początkowa prędkość kątowa ω_0 – w bloku *Integrator*). Zmienić podpis pod portem wejściowym (*Inport*) na $M(t)$ oraz pod portem wyjściowym (*Outport*) na *alfa*(t).



Rys. 6.2. Model symulacyjny wahadła

4. Zaznaczyć myszką prostokątny obszar wokół utworzonego modelu, a następnie wybrać z menu polecenie *Edit/Create Subsystem*. W ten sposób utworzymy pojedynczy blok podsystemu, reprezentującego model wahadła. Kliknąć podwójnie w symbol podsystemu – w nowym oknie otworzy się model z rys. 6.2. Zamknąć okno z wnętrzem podsystemu, zapisać schemat z pojedynczym blokiem podsystemu wahadła.

5. W kolejnym kroku utworzymy dla podsystemu wahadła okno dialogowe, przez które będziemy mogli przypisywać odpowiednie wartości zmiennym, reprezentującym parametry modelu (`l`, `m`, `k`, `g`) oraz warunki początkowe (`alfa0`, `omega0`).

W tym celu należy:

- Kliknąć prawym przyciskiem myszy na podsystemie wahadła, z menu kontekstowego wybrać polecenie *Mask Subsystem,* a następnie zakładkę *Parameters*.
- Klikając ikonę *Add* w lewym górnym rogu okna dodawać kolejno nagłówki tekstów (*Prompt*) oraz odpowiadające im nazwy zmiennych (*Variable*):

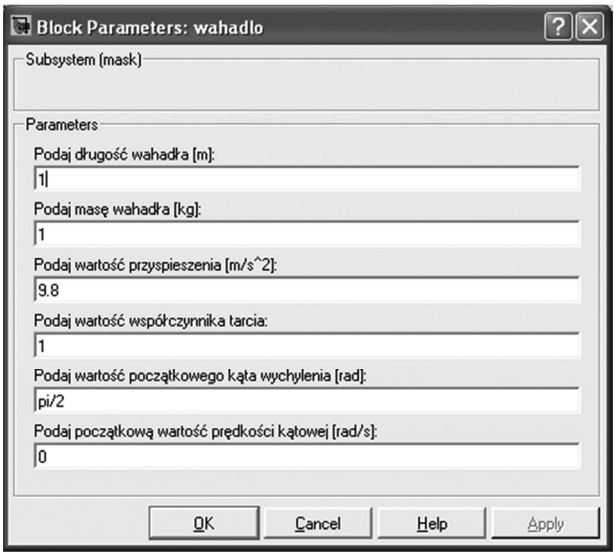
Podaj długość wahadła [m]:	`l`
Podaj masę wahadła [kg]:	`m`
Podaj wartość przyspieszenia grawitacyjnego [m/s^2]:	`g`
Podaj wartość współczynnika tarcia:	`k`
Podaj początkowy kąt wychylenia wahadła [rad]:	`alfa0`
Podaj początkową prędkość kątową wahadła [rad/s]:	`omega0`

- Po wprowadzeniu wszystkich pozycji kliknąć przycisk OK w oknie edytora maski.

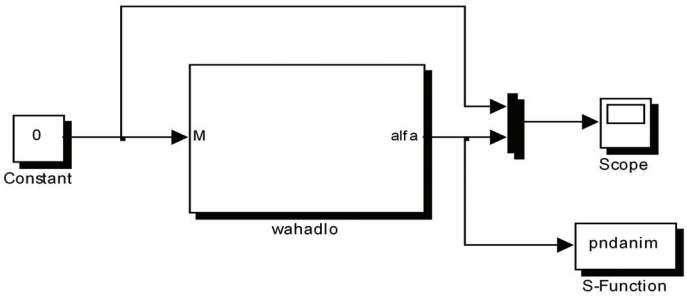
6. Kliknąć dwukrotnie w blok podsystemu wahadła. W oknie edycyjnym wprowadzić następujące wartości parametrów oraz warunków początkowych: l=1 m, m=1 kg, g=9.8 m/s^2, k=1, $\alpha_0=\pi/2$ rad, ω_0=0 rad/s (rys. 6.3).

7. Ze strony http://www.k.bartecki.po.opole.pl/nsi/pndanim.m pobrać plik funkcji realizującej animację układu wahadła i zapisać go w folderze roboczym *Matlaba*.

8. Zbudować układ do badania odpowiedzi czasowych wahadła jak na rys. 6.4. W polu *S-Function Name* bloku *S-Function* wpisać nazwę funkcji `pndanim`, w polu *S-Function Parameters* – wartość 0.01.



Rys. 6.3. Okno edycyjne podsystemu wahadła

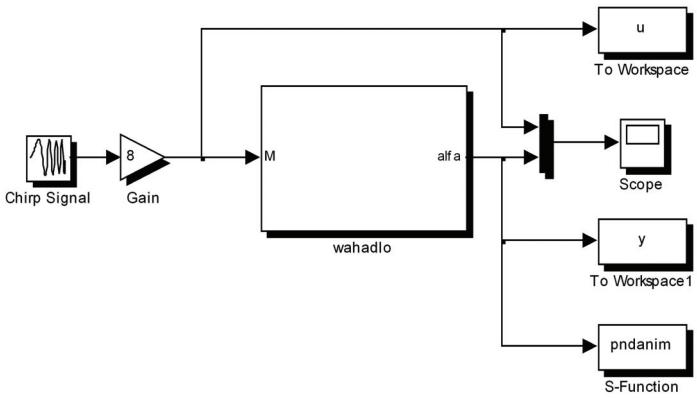


Rys. 6.4. Układ do badania odpowiedzi czasowych wahadła

9. Zbadać odpowiedź swobodną wahadła (tzn. dla M=0), przyjmując następujące warunki początkowe: $\alpha_0=\pi/2$ rad, ω_0=0 rad/s. Czas symulacji ustalić na 20. Po zakończeniu symulacji w oknie wyświetlacza (*Scope*) zaobserwować odpowiedź czasową wahadła w postaci tłumionych oscylacji. W trakcie symulacji powinna uruchomić się również animacja, ilustrująca zachowanie się wahadła.
W analogiczny sposób zarejestrować odpowiedzi czasowe wahadła dla stałych, niezerowych wartości momentu zewnętrznego (np. M=8 $N{\cdot}m$, M=–8 $N{\cdot}m$), przy zerowych warunkach początkowych, α_0=0 rad, ω_0=0 rad/s.

10. Układ z rys. 6.4. zmodyfikować w taki sposób, aby mógł posłużyć do zebrania danych identyfikacyjnych do uczenia neuronowych modeli wahadła. W tym celu należy:
- w miejsce bloku *Constant* użyć bloku *Chirp Signal*, generującego sygnał sinusoidalnie zmienny o narastającej częstotliwości,
- w polu *Initial frequency* (częstotliwość początkowa) generatora wpisać wartość 0.01, w polu *Target time* (czas końcowy) wartość 20, zaś w polu *Frequency at target time* (częstotliwość końcowa) – wartość 1,
- amplitudę sygnału z generatora zwiększyć ośmiokrotnie, wykorzystując blok *Gain*,
- aby umożliwić zapis sygnału wejściowego i wyjściowego do pamięci *Matlaka,* użyć bloków *To Workspace*,
- jako format zapisu danych (*Save format*) w obydwu blokach *To Workspace* wybrać *Array*, czas próbkowania (*Sample time*) ustawić równy 0.1, zaś zmiennym (*Variable name*) przypisać odpowiednio nazwę u (sygnał wejściowy M) oraz y (sygnał wyjściowy *alfa*).

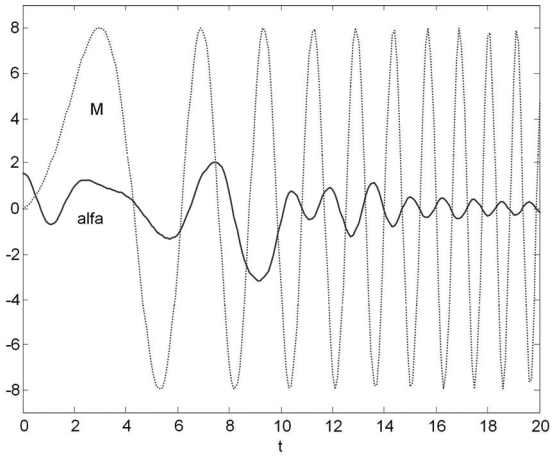
Po modyfikacjach układ powinien mieć postać jak na rys. 6.5.



Rys. 6.5. Schemat do badań identyfikacyjnych układu wahadła

11. Przeprowadzić symulację działania układu dla warunków początkowych $\alpha_0=\pi/2$ rad, $\omega_0=0$ rad/s. Klikając na bloku wyświetlacza (*Scope*) sprawdzić przebiegi czasowe sygnału wejściowego $M(t)$ oraz wyjściowego $\alpha(t)$ – powinny mieć one postać jak na rys. 6.6.

Przejść do okna poleceń *Matlaba*. Sprawdzić, czy w przestrzeni roboczej znajdują się zmienne `u` i `y` oraz jakie mają rozmiary.



Rys. 6.6. Przebiegi czasowe $M(t)$ oraz $\alpha(t)$ zarejestrowane w trakcie eksperymentu identyfikacyjnego układu wahadła

Po przeprowadzeniu eksperymentu identyfikacyjnego przystąpimy do tworzenia neuronowych modeli obiektu. W tym celu należy wykonać następujące czynności:

12. Otworzyć nowe okno edytora skryptów (*File/New/M-File*). Umieścić w nim definicję zmiennej N, reprezentującej liczbę zarejestrowanych w trakcie symulacji próbek oraz macierzy P, zawierającej odpowiednio opóźnione sygnały wyjściowe oraz wejściowe obiektu:

```
N = length(u);
P = [y(2:N-1)';    % wyjścia opóźnione o 1 próbkę
     y(1:N-2)';    % wyjścia opóźnione o 2 próbki
     u(2:N-1)';    % wejścia opóźnione o 1 próbkę
     u(1:N-2)'];   % wejścia opóźnione o 2 próbki
```

Wektor `T`, zawierający prognozowane wartości na wyjściu obiektu, zdefiniować samodzielnie – patrz rys. 6.1:

```
T = ...
```

Zapisać i uruchomić skrypt, sprawdzić, czy w pamięci *Matlaba* znajdują się macierze `P` oraz `T`. Sprawdzić ich rozmiary oraz wartości elementów.

13. Wykorzystując funkcję `newlind` utworzyć i obliczyć wartości współczynników wagowych liniowego neuronu, realizującego model ARX opisany zależnością (6.1). Zmiennej reprezentującej sieć nadać nazwę `net1`.

14. Wyświetlić wartości współczynników wagowych neuronu:

```
disp('współczynniki wagowe neuronu:')
disp( net1.IW{1} )
```

W oparciu o wyznaczone wartości współczynników wagowych zapisać równanie modelu ARX, zgodnie z zależnością (6.1).

15. Korzystając z funkcji `newrb` utworzyć nieliniową sieć radialną, realizującą opisany zależnością (6.3) model NARX. Wartość błędu średniokwadratowego przyjąć równą 0.0001, zaś wartość parametru funkcji ba-

zowej (*spread*) – równą 10. Zmiennej reprezentującej sieć nadać nazwę `netn`.

16. Umieścić w skrypcie dwa wywołania funkcji `gensim`, służące do wygenerowania bloków *Simulinka*, reprezentujących utworzone w poprzednich punktach modele neuronowe (`help gensim`). Czas próbkowania w naszym przypadku wynosi 0.1. Najpierw wygenerować w skrypcie blok modelu liniowego, następnie – nieliniowego.

17. Korzystając z funkcji `set_param` zmienić domyślne nazwy wygenerowanych bloków `'Neural Network'` odpowiednio na: `'model liniowy'` oraz `'model radialny'`. Wywołanie funkcji może wyglądać następująco:

```
set_param('untitled/Neural  Network','Name','model
liniowy')

set_param('untitled1/Neural  Network','Name','model
radialny')
```

W ostatnich punktach ćwiczenia dokonamy weryfikacji obydwu modeli neuronowych, poprzez wizualne porównanie ich odpowiedzi czasowych z odpowiedzią modelu wyjściowego wahadła na ten sam sygnał wejściowy. Najpierw wykorzystamy w tym celu sygnał wejściowy identyczny jak wykorzystywany w procesie uczenia, później również sygnały wejściowe innej postaci.

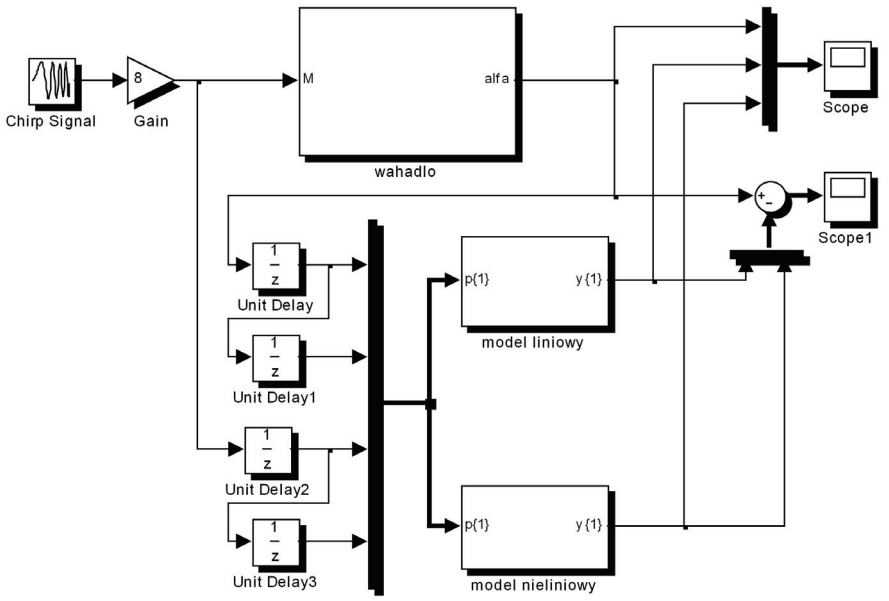
18. Zapisać pod nową nazwą schemat do badań identyfikacyjnych z rys. 6.6. Zmodyfikować jego strukturę w taki sposób, aby mógł posłużyć do weryfikacji neuronowych modeli wahadła. Wykorzystać w tym celu wygenerowane w punkcie 16. bloki reprezentujące liniową oraz radialną sieć neuronową. Sygnały wejściowe oraz wyjściowe obiektu doprowadzić do wejść modeli neuronowych poprzez bloki *Unit Delay*, reprezentujące opóźnienia jednostkowe oraz blok multipleksera (*Mux*). Model powinien mieć postać jak na rys. 6.7.

19. W blokach *Unit Delay* wartość czasu próbkowania (*Sample Time*) ustawić równą 0.1. Parametry bloków *Chirp Signal*, *Gain*, modelu wahadła

oraz parametry symulacji przyjąć identyczne jak w układzie do badań identyfikacyjnych z rys. 6.5.

20. Uruchomić symulację. Po jej zakończeniu sprawdzić jakość działania modeli neuronowych: liniowego oraz radialnego. W tym celu porównać wizualnie odpowiedzi czasowe modeli oraz obiektu (*Scope*) oraz wartości błędów identyfikacji $e(k)$ każdego z modeli (*Scope1*).

21. Sprawdzić jakość działania modeli neuronowych dla sygnału wejściowego innej postaci niż wykorzystany podczas eksperymentu identyfikacyjnego. W tym celu zmodyfikować schemat z rys. 6.7, usuwając bloki *Chirp Signal* oraz *Gain*. Na wejście obiektu oraz modeli neuronowych podać sygnał z generatora tzw. pasmowo ograniczonego białego szumu (ang. *Band-Limited White Noise*). Parametry tego bloku ustawić następująco: *Noise Power* = 10, *Sample time* = 1. Czas symulacji wydłużyć do 50.



Rys. 6.7. Schemat do badań weryfikacyjnych neuronowych modeli wahadła

22. Uruchomić symulację. Po jej zakończeniu sprawdzić jakość modelu liniowego oraz radialnego w sposób analogiczny, jak w punkcie 20.

Zadania dodatkowe:

W ostatnim ćwiczeniu dodatkowe zadania do wykonania zaproponować samodzielnie.

LITERATURA

Sztuczne sieci neuronowe

[1] DUCH W., KORBICZ J., RUTKOWSKI L., TADEUSIEWICZ R. (red. Nałęcz M.): *Biocybernetyka i inżynieria biomedyczna 2000. Tom 6 – Sieci neuronowe.* Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2000.

[2] HERTZ J., KROGH A., PALMER R.G.: *Wstęp do teorii obliczeń neuronowych.* WNT, Warszawa, 1995.

[3] KORBICZ J., OBUCHOWICZ A., UCIŃSKI D.: *Sztuczne sieci neuronowe. Podstawy i zastosowania.* AOW PLJ, Warszawa, 1994.

[4] KOSIŃSKI R.: *Sztuczne sieci neuronowe. Dynamika nieliniowa i chaos.* WNT, Warszawa, 2007.

[5] ŁĘSKI J.: *Systemy neuronowo-rozmyte.* WNT, Warszawa, 2008.

[6] MASTERS T.: *Sieci neuronowe w praktyce. Programowanie w języku C++.* WNT, Warszawa, 1996.

[7] OSOWSKI S.: *Sieci neuronowe w ujęciu algorytmicznym.* WNT, Warszawa, 1996.

[8] OSOWSKI S.: *Sieci neuronowe.* Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 1996.

[9] OSOWSKI S.: *Sieci neuronowe do przetwarzania informacji.* Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2006.

[10] ROJEK R., BARTECKI K., KORNIAK J.: *Zastosowanie sztucznych sieci neuronowych i logiki rozmytej w automatyce.* Skrypt Politechniki Opolskiej nr 234, Opole, 2000.

[11] RUTKOWSKA D., PILIŃSKI M., RUTKOWSKI L.: *Sieci neuronowe, algorytmy genetyczne i systemy rozmyte.* Wydawnictwo Naukowe PWN, Warszawa-Łódź, 1997.

[12] TADEUSIEWICZ R.: *Sieci neuronowe.* Akademicka Oficyna Wydawnicza RM, Warszawa, 1993.

[13] TADEUSIEWICZ R.: *Problemy biocybernetyki.* Wydawnictwo Naukowe PWN, Warszawa, 1994.

[14] TADEUSIEWICZ R., GĄCIARZ T., BOROWIK B., LEPER B.: *Odkrywanie właściwości sieci neuronowych przy użyciu programów w języku C#.* PAU, Kraków, 2007.

[15] ŻURADA J., BARSKI M., JĘDRUCH W.: *Sztuczne sieci neuronowe. Podstawy teorii i zastosowania.* Wydawnictwo Naukowe PWN, Warszawa, 1996.

Matlab* i *Neural Network Toolbox

[16] BRZÓZKA J., DOROBCZYŃSKI L.: *Matlab – środowisko obliczeń naukowo-technicznych.* Wydawnictwo Naukowe PWN, Warszawa, 2008.

[17] DEMUTH H., BEALE M., HAGAN M.: *Neural Network Toolbox 6. User's Guide.* The MathWorks, Inc., Natick MA, 2009.

[18] MROZEK B., MROZEK Z.: *Matlab i Simulink. Poradnik użytkownika.* Wydawnictwo Helion, Warszawa, 2004.

[19] PRATAP R.: *Matlab 7 dla naukowców i inżynierów.* Wydawnictwo Naukowe PWN, Warszawa, 2007.

[20] ZALEWSKI A., CEGIEŁA R.: *Matlab – obliczenia numeryczne i ich zastosowania.* Wydawnictwo NAKOM, Poznań, 2000.