Podstawowe funkcje biblioteki narzędziowej *„Neural Network Toolbox. Version 5”* pakietu MATLAB v. 7.1

Uwaga:

Nazwy funkcji i ich argumentów pisane są w niniejszych materiałach wielkimi literami wyłącznie w celu polepszenia czytelności opisu. W pakiecie MATLAB wywołania funkcji mogą być pisane zarówno wielkimi, jak i małymi literami, pisownia nazw argumentów (małe/wielkie litery) zależy zaś wyłącznie od wyboru użytkownika.

Część I. Sieci jednokierunkowe

I. Funkcje, używane do tworzenia jednokierunkowej sieci neuronowej

newp - Tworzenie jednowarstwowej sieci złożonej z „twardych" perceptronów.

NEWP Funkcja tworzy jednowarstwową sieć neuronową, złożoną z zadanej liczby neuronów o funkcjach aktywacji „twardego" perceptronu (ang. *hardlimit perceptron*).

Wywołanie funkcji: **NET = NEWP(PR, S, TF, LF)**

WEJŚCIE:

PR - macierz o wymiarach *R*x*2*, gdzie *R* jest liczbą wejść sieci (liczbą współrzędnych wektorów wejściowych); pierwsza kolumna macierzy *R* zawiera minimalne wartości kolejnych współrzędnych wektorów wejściowych, druga kolumna – maksymalne wartości tych współrzędnych

S - liczba neuronów sieci

TF - nazwa funkcji aktywacji neuronów (zmienna tekstowa); nazwa domyślna = 'hardlim'; dopuszczalne wartości parametru *TF* to: 'hardlim' i 'hardlims'

LF - nazwa funkcji trenowania sieci perceptronowej (zmienna tekstowa); nazwa domyślna = 'learnp'; dopuszczalne wartości parametru *LF* to: 'learnp' i 'learnpn'

WYJŚCIE:

NET - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości liczbowe wag i progów oraz inne parametry sieci perceptronowej.

Przykłady:

Podane niżej wywołanie funkcji tworzy sieć zawierającą pojedynczy neuron o dwóch wejściach. Zakres wartości pierwszego wejścia to [0, 1] drugiego [-2, 2]. Wywołanie funkcji zawiera tylko dwa argumenty – pozostałe dwa przyjmą wartości domyślne: 'hardlim' dla funkcji aktywacji i 'learnp' dla funkcji treningu sieci:

```
net = newp([0 1; -2 2], 1);
```

Wywołanie funkcji obliczającej wyjście sieci dla zadanego wektora wejściowego *P1* (tzw. symulacja sieci):

```
P1 = {[0; 0] [0; 1] [1; 0] [1; 1]};
Y = sim(net, P1)
```

Zdefiniujmy wektor wyjść sieci, *T* (*P1* i *T* są wartościami binarnymi, które opisują działanie bramki logicznej AND). Wywołajmy funkcję adaptacji sieci (adaptacyjnego doboru wag) poprzez 10-krotną prezentację sekwencji wejściowo-wyjściowej, a następnie wykonajmy symulację sieci:

```
T1 = {0 0 0 1};
net.adaptParam.passes = 10;
net = adapt(net, P1, T1);
Y = sim(net,P1)
```

Kolejny zestaw wejść i wyjść (*P2, T2*) odpowiada opisowi działania bramki OR:

```
P2 = [0 0 1 1; 0 1 0 1];
T2 = [0 1 1 1];
```

Pierwszą instrukcją jest inicjalizacja sieci perceptronowej (w tej metodzie inicjalizacji wartości wag i progów są wyznaczane w sposób losowy); drugą instrukcją – symulacja sieci. Następnie określamy liczbę epok, wywołujemy funkcję 'train' (dokonującą treningu sieci), a następnie ponownie symulujemy działanie sieci dla wartości parametrów wyznaczonych w procesie treningu:

```
net = init(net);
Y = sim(net, P2)
net.trainParam.epochs = 20;
net = train(net, P2, T2);
Y = sim(net, P2)
```

----- *** -----

newlin - Tworzenie jednowarstwowej sieci złożonej z neuronów liniowych.

NEWLIN Funkcja tworzy jednowarstwową sieć neuronową, złożoną z zadanej liczby neuronów o liniowych funkcjach aktywacji. Tego typu sieć jest zwykle wykorzystywana jako filtr adaptacyjny do przetwarzania sygnałów lub predykcji szeregów czasowych.

Wywołanie funkcji: **NET = NEWLIN(PR, S, ID, LF)**

WEJŚCIE:

- *PR* - macierz o wymiarach *R*x*2*, gdzie *R* jest liczbą wejść sieci (tj. liczbą współrzędnych wektorów wejściowych); pierwsza kolumna zawiera minimalne wartości kolejnych współrzędnych wektorów wejściowych, druga kolumna – maksymalne wartości tych współrzędnych
- *S* - liczba neuronów sieci, tj. wyjść sieci (neurony tworzą automatycznie warstwę wyjściową)
- *ID* - wektor opóźnień poszczególnych elementów wektora wejść sieci; domyślnie *ID* = [0]
- *LR* - stała szybkości uczenia (treningu) sieci liniowej; domyślnie *LR* = 0.01

WYJŚCIE:

- *NET* - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry sieci perceptronowej.

Alternatywne wywołanie funkcji: **NET = NEWLIN(PR, S, 0, P)**

WEJŚCIE:

- *PR* - jak wyżej
- *S* - jak wyżej
- *P* - macierz zbudowana z wektorów wejściowych sieci (każdy wektor wejściowy zajmuje jedną kolumnę w tej macierzy).

Struktura *NET*, zwracana przez tę funkcję zawiera maksymalną wartość stałej szybkości uczenia, zapewniającą stabilny trening sieci liniowej dla wektorów wejściowych zawartych w macierzy *P*.

Przykłady:

Podane niżej wywołanie funkcji tworzy sieć liniową zawierającą pojedynczy neuron o dwóch wejściach. Zakres wartości wejść to [-1, 1], opóźnienie sygnału dla pierwszego wejścia wynosi 0, dla drugiego 1 (elementy wektora P1 są w tym przypadku traktowane jako kolejne próbki sygnału). Założona wartość stałej szybkości uczenia wynosi 0.01. W trzeciej instrukcji dokonujemy symulacji sieci liniowej:

```
net = newlin([-1 1], 1, [0 1], 0.01);
P1 = {0 -1 1 1 0 -1 1 0 0 1};
Y = sim(net, P1)
```

Następnie zdefiniujmy wartości elementów wektora wyjść, *T1*, i wywołajmy funkcję adaptacji wag sieci (w tym przypadku procedura przyjmuje domyślne wartości opóźnień dla kolejnych wejść)

```
T1 = {0 -1 0 2 1 -1 0 1 0 1};
[net, Y, E, Pf] = adapt(net, P1, T1); Y
```

Kontynuujemy procedurę adaptacji sieci do nowej sekwencji danych, zdefiniowanych w wektorach *P2* i *T2* (procedura adaptacji startuje z warunku *Pf*):

```
P2 = {1 0 -1 -1 1 1 1 0 -1};
T2 = {2 1 -1 -2 0 2 2 1 0};
[net, Y, E, Pf] = adapt(net, P2, T2); Y
```

Inicjalizacja wag i progów sieci liniowej:

```
net = init(net);
```

Następnie przeprowadzamy trening sieci, wykorzystując obydwie sekwencje danych. Długość treningu określono na 200 epok, zaś kryterium stopu (sumę kwadratów błędów wyjść sieci) na wartość 0.1. Adaptacja i trening są realizowane za pomocą funkcji ADAPTWB I TRAINWB, które wywołują funkcję korekcji wag LEARNWH. Po zakończeniu treningu symulujemy działanie sieci:

```
P3 = [P1 P2]; T3 = [T1 T2];
net.trainParam.epochs = 200;
net.trainParam.goal = 0.1;
net = train(net, P3, T3);
Y = sim(net, [P1 P2])
```

----- *** -----

newlind - Projektowanie jednowarstwowej sieci złożonej z neuronów liniowych.

NEWLIND Funkcja tworzy jednowarstwową sieć neuronową, złożoną z zadanej liczby neuronów o liniowych funkcjach aktywacji. Ponadto funkcja oblicza wagi *W* i progi neuronów sieci (*B*) w wyniku rozwiązania metodą najmniejszych kwadratów poniższego równania liniowego (w/g notacji pakietu MATLAB):

[W b] * [P; ones] = T

Tego typu sieć jest zwykle wykorzystywana jako filtr adaptacyjny do przetwarzania sygnałów lub predykcji szeregów czasowych.

Wywołanie funkcji: **NET = NEWLIND(P, T, Pi)**

WEJŚCIE:

P - macierz o wymiarach *R*x*Q*, gdzie *R* jest liczbą wejść sieci (współrzędnych wektorów wejściowych); zaś *Q* – liczbą wektorów wejściowych, użytych do treningu sieci

T - macierz o wymiarach *S*x*Q*, gdzie *S* jest liczbą wyjść sieci; macierz zawiera *Q* pożądanych wektorów wyjściowych, odpowiadających wektorom zawartym w macierzy *P*

Pi - macierz o wymiarach *1*x*ID*, zawierająca początkowe wartości opóźnionych wejść sieci (*ID* jest liczbą opóźnień); jest to parametr opcjonalny, domyślna wartość *Pi* = []

WYJŚCIE:

NET - struktura (obiekt) zawierająca jednowarstwową liniową sieć neuronową, zaprojektowaną tak, aby minimalizować błąd liniowego odwzorowania macierzy *P* w macierz *T*.

Przykład:

Zdefiniujmy wektorów wejść sieci (*P*) i odpowiadających mu pożądanych wyjść (*T*):

```
P = [1 2 3];
T = [2.0 4.1 5.9];
```

Podane niżej instrukcje tworzą i trenują sieć liniową dla danych wektorów *P* i *T*:

```
net = newlind(P, T);
Y = sim(net, P)
```

Instrukcje poniżej tworzą i wyznaczają wagi sieci liniowej, zawierającej dwa opóźnione wejścia, o wartościach początkowych zawartych w macierzy *Pi*:

```
P = {1 2 1 3 3 2};
Pi = {1 3};
T = {5.0 6.1 4.0 6.0 6.9 8.0};
net = newlind(P, T, Pi);
Y = sim(net, P, Pi)
```

----- *** -----

newff - Tworzenie wielowarstwowej jednokierunkowej sieci neuronowej, złożonej z neuronów o nieliniowych funkcjach aktywacji.

NEWFF Funkcja tworzy wielowarstwową sieć neuronową; każda warstwa składa się z zadanej liczby neuronów o nieliniowych funkcjach aktywacji (jakkolwiek funkcje aktywacji w poszczególnych warstwach mogą mieć również postać liniową).

Wywołanie funkcji: **NET = NEWFF(PR, [S1 S2...SNL], ...**
{TF1 TF2...TFNL}, BTF, BLF, PF)

WEJŚCIE:

PR - macierz o wymiarach *R*x*2*, gdzie *R* jest liczbą wejść sieci (współrzędnych wektorów wejściowych); pierwsza kolumna zawiera minimalne wartości kolejnych współrzędnych wektorów wejściowych, druga kolumna – maksymalne wartości tych współrzędnych

Si - liczba neuronów w *i*-tej warstwie sieci; liczba warstw wynosi *N1*

TFi - nazwa funkcji aktywacji neuronów w *i*-tej warstwie sieci (zmienna tekstowa); domyślna = 'tansig' (tangens hiperboliczny); dopuszczalne wartości parametru *TF* to: ‘tansig’ i ‘logsig’ i ‘purelin’

BTF -nazwa funkcji, wykorzystywanej do treningu sieci (zmienna tekstowa); domyślnie *BTF* = ‘trainlm’ (metoda Levenberga-Marquardta)

BLF -nazwa funkcji, wykorzystywanej do wyznaczania korekcji wag sieci podczas treningu (zmienna tekstowa); domyślnie *BLF* = ‘learngd’; dopuszczalne wartości parametru *BLF* to: ‘learngd’ (gradient prosty) i ‘learngdm’ (gradient prosty z momentum)

PF - funkcja wyznaczająca wartość wskaźnika jakości treningu sieci jednokierunkowej (zmienna tekstowa); domyślnie *PF* = ‘mse’ (błąd średniokwadratowy); parametr ten może oznaczać dowolną różniczkowalną funkcję błędu, np. ‘msereg’ (suma błędu średniokwadratowego i kwadratów wag sieci – metoda regularyzacji wag) lub ‘sse’ (suma kwadratów błędów)

WYJŚCIE:

NET - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry wielowarstwowej sieci jednokierunkowej.

Uwaga: zestaw funkcji treningu sieci jednokierunkowych (tj. zbiór dopuszczalnych wartości zmiennej *BTF*) obejmuje:

trainbfg	metoda gradientowa BFGS, Broydena-Flrtchera-Goldfarba-Shano (quasi-Newtonowska),
trainbr	odmiana metody Levenberga-Marquardta z regularyzacją Bayes’owską,
traincgb	metoda gradientu sprzężonego Powella-Beale’go,
traincgf	metoda gradientu sprzężonego Fletchera-Powella,
traincgp	metoda gradientu sprzężonego Polaka-Ribiere,
traingd	metoda gradientu prostego (wstecznej propagacji błędu),
traingda	metoda propagacji wstecznej błędu z adaptacyjną zmianą stałej szybkości uczenia,
traingdm	metoda propagacji wstecznej błędu z momentum,
traingdx	metoda propagacji wstecznej błędu z momentum i adaptacyjną zmianą stałej szybkości uczenia,
trainlm	metoda Levenberga-Marquardta,
trainscg	metoda skalowanego gradientu sprzężonego.

Przykłady:

Zdefiniujmy wektor wejściowy *P* i odpowiadające jego elementom wartości wyjścia sieci, *T*:

```
P = [0 1 2 3 4 5 6 7 8 9 10];
T = [0 1 2 3 4 3 2 1 2 3 4];
```

Następnie tworzymy dwuwarstwową jednokierunkową sieć neuronową, której wejścia (z założenia) będą przyjmować wartości z zakresu [0, 1]. Pierwsza warstwa składa się z pięciu neuronów o tangensoidalnych funkcjach aktywacji (TANSIG), druga warstwa zawiera pojedynczy neuron o liniowej funkcji aktywacji (PURELIN). Zauważmy, że sieć ma przetwarzać skalarne wejście w skalarne wyjście (liczba współrzędnych wektorów wejściowych, tj. wymiarowość przestrzeni wejść, wynosi jeden; podobnie jak wymiarowość przestrzeni wyjść). Do treningu sieci będzie wykorzystana metoda Levenberga-Marquardta (TRAINLM). Domyślnie, wagi nowo utworzonej sieci są inicjalizowane za pomocą funkcji INITNW.

```
net = newff([0 10], [5 1], {'tansig' 'purelin'});
```

Wyznaczamy wartości wyjść sieci (wykonujemy symulację) oraz prezentujemy aktualne i pożądane wartości wyjść:

```
Y = sim(net, P);
plot(P, T, P, Y, 'o')
```

Przeprowadzamy trening sieci (długość treningu wynosi 50 epok), wykonujemy symulację, a następnie ponownie prezentujemy aktualne i pożądane wartości wyjść:

```
net.trainParam.epochs = 50;
net = train(net, P, T);
Y = sim(net, P);
plot(P, T, P, Y, 'o')
```

----- *** -----

II. Funkcje, używane do podstawowych operacji na strukturach sieci neuronowych

init - Inicjalizacja struktury sieci neuronowej.

INIT Funkcja inicjalizuje wartości wag i progów neuronów sieci, wykorzystując funkcję inicjalizującą, zdefiniowaną w polu NET.initFcn i wartości parametrów, podane w polu NET.initParam wejściowej struktury *NET*. Zazwyczaj pole NET.initFcn ma wartość 'initlay', co powoduje użycie do inicjalizacji każdej warstwy funkcji zdefiniowanej w polu NET.layers{i}.initFcn (indeks *i* oznacza numer warstwy). Dla sieci jednokierunkowych (utworzonych np. za pomocą funkcji ‘newff’) domyślna wartość pola NET.layers{i}.initFcn = 'initnw'; tak więc inicjalizacja wag jest dokonywana za pomocą algorytmu Nguyena-Widrowa.

Wywołanie funkcji: **NET = INIT(NET)**

WEJŚCIE:

NET - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry wielowarstwowej sieci jednokierunkowej, utworzonej np. za pomocą funkcji NEWFF

WYJŚCIE:

NET - struktura wielowarstwowej sieci jednokierunkowej, z nowymi wartościami progów i wag synaptycznych, obliczonych za pomocą funkcji inicjalizacyjnej

Przykład:

Utwórzmy prostą sieć perceptronową, o dwuelementowych wektorach wejściowych; zakres pierwszego wejścia to [0, 1], drugiego – [-2, 2]. Kolejne instrukcję służą do wyświetlenia wartości wag i progów neuronów sieci:

```
net = newp([0 1;-2 2], 1);
net.iw{1, 1}
net.b{1}
```

Trening perceptronu zmienia wartości wag synaptycznych sieci:

```
P = [0 1 0 1; 0 0 1 1]; T = [0 0 0 1];
net = train(net, P, T);
net.iw{1, 1}
net.b{1}
```

Wywołanie funkcji INIT powoduje powtórną inicjalizację wag sieci; w przypadku perceptronu są one ustawiane na zera, zgodnie z algorytmem działania funkcji NEWP:

```
net = init(net);
net.iw{1, 1}
net.b{1}
```

----- *** -----

disp - Wyświetlenie (wydruk na ekranie) informacji o sieci neuronowej.

DISP Funkcja służy do wyświetlenia wartości poszczególnych pól struktury *NET*, opisującej sieć neuronową (architekturę, wartości parametrów, nazwy funkcji inicjalizacji i treningu, itp.).

Wywołanie funkcji: **DISP(NET)**

WEJŚCIE:

NET - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry wielowarstwowej sieci jednokierunkowej, utworzonej np. za pomocą funkcji NEWFF

Przykład:

Poniższe instrukcje tworzą prostą sieć perceptronową oraz wyświetlają informacje o tej sieci:

```
net = newp([-1 1;0 2], 3);
disp(net)
```

----- *** -----

train - Trening sieci neuronowej.

TRAIN Funkcja realizuje trening sieci neuronowej, wykorzystując funkcję treningu, której nazwa została podana w polu NET.trainFcn, zaś wartości niezbędnych parametrów – w polu NET.trainParam. Funkcja treningu jest uniwersalna, wywoływana w jednolity sposób dla wszystkich typów sieci neuronowych, stąd też niektóre argumenty wejściowe nie mają znaczenia dla sieci jednokierunkowych (bez sprzężeń zwrotnych i opóźnień w torach sygnałów wejściowych)

Wywołanie funkcji: **[NET, TR, Y, E, Pf, Af] = TRAIN(NET, P, T, Pi, Ai)**
lub: **[NET, TR, Y, E, Pf, Af] = TRAIN(NET, P, T, Pi, Ai, VV, TV)**

WEJŚCIE:

NET - struktura (obiekt) zawierająca opis wielowarstwowej sieci jednokierunkowej

P - macierz wejść sieci, utworzona z wektorów treningowych (każdy wektor stanowi jedną kolumnę tej macierzy)

T - macierz pożądanych wyjść sieci; dla sieci uczonych bez nauczyciela ten argument przyjmuje domyślnie wartość zerową

Pi - macierz utworzona z wektorów początkowych opóźnień próbek sygnału wejściowego sieci; jeśli użytkownik nie podaje wartości tego argumentu w wywołaniu funkcji – przyjmuje on domyślnie wartość zerową

Ai - macierz utworzona z wektorów początkowych opóźnień próbek sygnału w kolejnych warstwach sieci; jeśli użytkownik nie podaje wartości tego argumentu w wywołaniu funkcji – przyjmuje on domyślnie wartość zerową

VV - opcjonalne dane służące do przeprowadzenia procesu walidacji (oceny) treningu; macierze struktury walidacyjnej: VV.P, VV.T, VV.Pi, VV.Ai odpowiadają argumentom w podstawowej postaci wywołania funkcji treningu, tj.: P, T, Pi, Ai. Argument ten może przyjmować wartość pustą, [].

TV - opcjonalne dane służące do przeprowadzenia procesu testowania sieci po zakończeniu treningu; macierze struktury testowej: TV.P, TV.T, TV.Pi, TV.Ai odpowiadają argumentom w podstawowej postaci wywołania funkcji treningu, tj.: P, T, Pi, Ai. Argument ten może przyjmować wartość pustą, [].

WYJŚCIE:

NET - struktura wielowarstwowej sieci jednokierunkowej, z nowymi wartościami progów i wag synaptycznych, wyznaczonych w wyniku treningu

TR - informacja o przebiegu treningu (liczba epok, przebieg funkcji błędu treningu, itp.)

Y - wartości wyjść wytrenowanej sieci

E - błędy na wyjściu sieci

Pf - wyjściowa macierz utworzona z wektorów opóźnień próbek sygnału wejściowego sieci (również uzyskana w procedurze adaptacji)

Af - wyjściowa macierz utworzona z wektorów opóźnień próbek sygnału w poszczególnych warstwach sieci (również uzyskana w procedurze adaptacji)

Przykład:

Przeprowadzimy trening sieci w zastosowaniu do aproksymacji funkcji jednej zmiennej, *t(p)*:

```
p = [0 1 2 3 4 5 6 7 8];
t = [0 0.84 0.91 0.14 -0.77 -0.96 -0.28 0.66 0.99];
plot(p, t, 'o')
```

Do utworzenia struktury sieci zastosujemy funkcję NEWFF; sieć ma się składać z 10 neuronów o tangensoidalnych funkcjach aktywacji (TANSIG) w warstwie wejściowej i jednego neuronu o liniowej funkcji aktywacji (PURELIN) w warstwie wyjściowej. Zakres zmian wartości wejść określamy na [0, 8], zaś do treningu sieci zastosujemy metodę Levenberga-Marquardta (funkcja TRAINLM).

```
net = newff([0 8], [10 1], {'tansig' 'purelin'}, 'trainlm');
y1 = sim(net, p)
figure(gcf+1); plot(p, t, 'o', p, y1, 'x')
```

Wytrenujmy sieć w ciągu 50 epok, zakładając wartość kryterium stopu (błąd średniokwadratowy) równą 0.01.

```
net.trainParam.epochs = 50;
net.trainParam.goal = 0.01;
net = train(net, p, t);
y2 = sim(net, p)
figure(gcf+1); plot(p, t, 'o', p, y1, 'x', p, y2, '*')
```

----- *** -----

adapt - Adaptacja wag sieci neuronowej.

ADAPT Funkcja realizuje w swej podstawowej postaci jeden krok (tzw. epokę) treningu sieci neuronowej, dokonując adaptacji wag synaptycznych sieci. Do adaptacji jest wykorzystywana funkcja, której nazwa została podana w polu NET.adaptFcn, zaś wartości niezbędnych parametrów tej funkcji – w polu NET.adaptParam. Funkcja adaptacji jest uniwersalna, wywoływana w jednolity sposób dla wszystkich typów sieci neuronowych, stąd też niektóre argumenty wejściowe nie mają znaczenia dla sieci jednokierunkowych (bez sprzężeń zwrotnych i opóźnień w torach sygnałów wejściowych)

Wywołanie funkcji: **[NET, Y, E, Pf, Af, TR] = ADAPT(NET, P, T, Pi, Ai)**

WEJŚCIE:

- *NET* - struktura (obiekt) zawierająca opis wielowarstwowej sieci jednokierunkowej
- *P* - macierz wejść sieci, utworzona z wektorów treningowych (każdy wektor stanowi jedną kolumnę tej macierzy)
- *T* - macierz pożądanych wyjść sieci, dla sieci uczonych bez nauczyciela ten argument przyjmuje domyślnie wartość zerową
- *Pi* - macierz utworzona z wektorów początkowych opóźnień próbek sygnału wejściowego sieci; jeśli użytkownik nie podaje wartości tego argumentu w wywołaniu funkcji – przyjmuje on domyślnie wartość zerową
- *Ai* - macierz utworzona z wektorów początkowych opóźnień próbek sygnału w kolejnych warstwach sieci; jeśli użytkownik nie podaje wartości tego argumentu w wywołaniu funkcji – przyjmuje on domyślnie wartość zerową

WYJŚCIE:

- *NET* - struktura wielowarstwowej sieci jednokierunkowej, z nowymi wartościami progów i wag synaptycznych, wyznaczonych w wyniku adaptacji
- *Y* - wartości wyjść sieci
- *E* - błędy na wyjściu sieci
- *Pf* - wyjściowa macierz utworzona z wektorów opóźnień próbek sygnału wejściowego sieci (również uzyskana w procedurze adaptacji)
- *Af* - wyjściowa macierz utworzona z wektorów opóźnień próbek sygnału w poszczególnych warstwach sieci (również uzyskana w procedurze adaptacji)
- *TR* - informacja o przebiegu treningu (liczba epok, przebieg funkcji błędu treningu, itp.)

Przykłady:

Przeprowadzimy adaptację wag sieci, prezentując jej 12 wzorców (zakładamy, że *T1* zależy od *P1*) – próbek sygnału wejściowego i wyjściowego filtru liniowego:

```
p1 = {-1  0 1 0 1 1 -1  0 -1 1 0 1};
t1 = {-1 -1 1 1 1 2  0 -1 -1 0 1 1};
```

Tworzymy strukturę sieci za pomocą funkcji NEWLIN; zakres zmian wejścia wybieramy jako [-1, 1], opóźnienia na każdym z wejść sieci wynoszą odpowiednio 0 i 1 (tzn. sieć realizuje filtr liniowy I-go rzędu: *t1(k) = a1*p1(k) + a2*p1(k-1)*). Wartość stałej szybkości uczenia jest równa 0.5.

```
net = newlin([-1 1], 1, [0 1], 0.5);
```

Kolejna instrukcja wywołuje procedurę adaptacji sieci, tj. modyfikacji jej parametrów po przedstawieniu całej sekwencji sygnałów: wejściowego i wyjściowego, których próbki znajdują się w wektorach *p1* i *t1*. Zauważmy, że funkcja oblicza również nowy

wektor opóźnień; w kolejnej instrukcji drukowana jest wartość błędu średniokwadratowego na wyjściu filtru:

```
[net, y, e, pf] = adapt(net, p1, t1);
mse(e)
```

Ze względu na dużą wartość błędu, wywołujemy powtórnie procedurę adaptacji wag do nowej sekwencji próbek sygnału, z wyznaczonymi poprzednio wartościami wag synaptycznych i opóźnień *pf* (jako argumentem wejściowym funkcji):

```
p2 = {1 -1 -1 1 1 -1  0 0 0 1 -1 -1};
t2 = {2  0 -2 0 2  0 -1 0 0 1  0 -1};
[net, y, e, pf] = adapt(net, p2, t2, pf);
mse(e)
```

Następnie dokonujemy adaptacji wag sieci w 100 krokach czasowych (takie wywołanie funkcji ADAPT jest równoważne treningowi sieci w ciągu 100 epok treningowych).

```
p3 = [p1 p2]; t3 = [t1 t2];
net.adaptParam.passes = 100;
[net, y, e] = adapt(net, p3, t3);
mse(e)
```

----- *** -----

sim - Symulacja sieci neuronowej dla zadanych danych wejściowych.

SIM

Funkcja służy do wyznaczenia wyjść sieci neuronowej dla zadanej macierzy danych wejściowych. Argumenty *Pi*, *Ai*, *Pf*, *Af* są opcjonalne i nie będą używane przez sieci nieliniowe, wykorzystywane w ćwiczeniach

Wywołanie funkcji: **[Y, Pf, Af, E, Perf] = SIM(NET, P, Pi, Ai, T)**
lub: **[Y, Pf, Af, E, Perf] = SIM(NET, {Q, TS}, Pi, Ai, T)**
lub: **[Y, Pf, Af, E, Perf] = SIM(NET, Q, Pi, Ai, T)**

WEJŚCIE:

- *NET* - struktura (obiekt) zawierająca opis sieci neuronowej
- *P* - macierz wejść sieci, utworzona z wektorów wejściowych (każdy wektor stanowi jedną kolumnę tej macierzy)
- *Pi* - macierz utworzona z wektorów początkowych opóźnień próbek sygnału wejściowego sieci; jeżeli użytkownik nie podaje wartości tego argumentu w wywołaniu funkcji – przyjmuje on domyślnie wartość zerową
- *Ai* - macierz utworzona z wektorów początkowych opóźnień próbek sygnału w kolejnych warstwach sieci; jeżeli użytkownik nie podaje wartości tego argumentu w wywołaniu funkcji – przyjmuje on domyślnie wartość zerową
- *T* - macierz pożądanych wyjść sieci, dla sieci uczonych bez nauczyciela ten argument przyjmuje domyślnie wartość zerową

WYJŚCIE:

- *Y* - macierz, utworzona z wektorów wyjść sieci, wyznaczonych dla wektorów zapisanych w macierzy *P*
- *Pf* - wyjściowa macierz utworzona z wektorów opóźnień próbek sygnału wejściowego sieci
- *Af* - wyjściowa macierz utworzona z wektorów opóźnień próbek sygnału w poszczególnych warstwach sieci
- *E* - błędy na wyjściu sieci
- *Perf* - wartość funkcji oceny błędu odwzorowania sieci, której nazwę zawiera pole ‘net.performFcn’ (domyślnie ‘mse’ – błąd średniokwadratowy)

Uwaga: Dwa alternatywne wywołania funkcji SIM są wykorzystywane przez sieci, które nie mają wejść, jak np. sieci Hopfielda. W przypadku sieci jednokierunkowych, prawidłowy jest pierwszy sposób wywołania funkcji SIM.

Przykład:

Poniższa instrukcja tworzy prostą sieć perceptronową (zbudowaną z jednego neuronu) o dwóch wejściach zmieniających się w zakresie [0, 1]:

```
net = newp[0 1; 0 1], 1);
disp(net)
```

Działanie perceptronu dla trzech różnych sekwencji wejść jest zilustrowane poniżej:

```
p1 = [.2; .9]; a1 = sim(net, p1)
p2 = [.2 .5 .1; .9 .3 .7]; a2 = sim(net, p2)
p3 = {[.2; .9] [.5; .3] [.1; .7]}; a3 = sim(net, p3)
```

Poniżej, sieć liniowa o trzech wejściach, złożona z dwóch neuronów, jest tworzona za pomocą funkcji NEWLIN:

```
net = newlin([0 2; 0 2; 0 2], 2, [0 1]);
```

Wykorzystajmy funkcję SIM również do obliczenia wyjść tej sieci dla dwóch zadanych wektorów wejściowych (przy założeniu zerowych opóźnień na wejściach):

```
p1 = {[2; 0.5; 1] [1; 1.2; 0.1]};
[y1, pf] = sim(net, p1)
```

Wywołanie poniżej wykorzystuje wartości opóźnień, wyznaczone w poprzednim wywołaniu funkcji:

```
p2 = {[0.5; 0.6; 1.8] [1.3; 1.6; 1.1] [0.2; 0.1; 0]};
[y2, pf] = sim(net, p2, pf)
```

----- *** -----

III. Funkcje, używane do inicjalizacji wag i progów jednokierunkowej sieci neuronowej

initnw - Inicjalizacja wag sieci metodą Nguyena-Widrowa.

INITNW Funkcja inicjalizuje wartości wag i progów neuronów *i*-tej warstwy w jednokierunkowej sieci neuronowej. Algorytm inicjalizacji zapewnia jak najbardziej równomierny podział przestrzeni wejściowej warstwy na „obszary wpływu" każdego z neuronów. W ten sposób zapobiega się „utracie" niektórych neuronów (tj. wszystkie neurony sieci biorą wówczas udział w przetwarzaniu danych) oraz przyspiesza trening sieci.

Wywołanie funkcji: **NET = INITNW(NET, I)**

WEJŚCIE:

NET - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry wielowarstwowej sieci jednokierunkowej, utworzonej np. za pomocą funkcji NEWFF

I - numer warstwy sieci (warstwie wejściowej przypisany jest numer 1)

WYJŚCIE:

NET - struktura wielowarstwowej sieci jednokierunkowej, z nowymi wartościami progów i wag synaptycznych

----- *** -----

initwb - Inicjalizacja wag metodą indywidualną dla każdej z warstw sieci.

INITWB Funkcja inicjalizuje wartości wag i progów neuronów *i*-tej warstwy w jednokierunkowej sieci neuronowej. Funkcja umożliwia inicjalizację wag w poszczególnych warstwach za pomocą różnych funkcji inicjalizujących, określonych indywidualnie dla każdej warstwy.

Wywołanie funkcji: **NET = INITWB(NET, I)**

WEJŚCIE:

NET - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry wielowarstwowej sieci jednokierunkowej, utworzonej np. za pomocą funkcji NEWFF

I - numer warstwy sieci (warstwie wejściowej przypisany jest numer 1)

WYJŚCIE:

NET - struktura wielowarstwowej sieci jednokierunkowej, z nowymi wartościami progów i wag synaptycznych

Przykład:

Aby przygotować sieć do inicjalizacji wag za pomocą funkcji INITWB należy wykonać następującą sekwencję instrukcji:

```
NET.initFcn = 'initlay';
NET.layers{i}.initFcn = 'initwb';  % Dla każdego numeru warstwy, i
NET.inputWeights{i,j}.learnFcn = ‘nazwa_funkcji_inicjalizacji_wag’;
NET.layerWeights{i,j}.learnFcn = ‘nazwa_funkcji_inicjalizacji_wag’;
NET.biases{i}.learnFcn = ‘nazwa_funkcji_inicjalizacji_progów’;
```

Instrukcję definiującą funkcję inicjalizacji wag należy wykonać nie tylko dla każdego numeru warstwy, *i*, ale również dla każdego numeru neuronu w warstwie, *j*. Przykładowo, funkcjami inicjalizacji wag i progów mogą być: RANDS, RANDNC, RANDNR, MIDPOINT lub INITZERO (opis tych funkcji można uzyskać za pomocą odpowiednich instrukcji pomocy, np. ‘help rands’). Następnie dokonujemy inicjalizacji wag synaptycznych za pomocą instrukcji:

```
net = init(net);
```

----- *** -----

IV. Funkcje pomocnicze

filter - Jednowymiarowy filtr cyfrowy.

FILTER Funkcja realizuje filtrację sygnału wejściowego *x* za pomocą filtru cyfrowego, opisanego za pomocą poniższego równania różnicowego (*y* jest sygnałem wyjściowym filtru):

a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb) +
- a(2)*y(n-1) - ... - a(na+1)*y(n-na)

Jeśli a(1) nie jest równe 1, operacje wewnątrz funkcji normalizują współczynniki filtru.

Wywołanie funkcji: **Y = FILTER(B, A, X)**
lub: **[Y, Zf] = FILTER(B, A, X, Zi)**

WEJŚCIE:

A, B – wektory współczynników równania różnicowego filtru

X - wektor zawierający próbki sygnału wejściowego; jeśli *X* jest macierzą, funkcja działa na wektorach tej macierzy

Zi - wektor wartości początkowych sygnałów występujących po prawej stronie równania różnicowego, o długości (MAX(LENGTH(A), LENGTH(B))-1); jest to parametr opcjonalny, domyślna wartość *Zi* = []

WYJŚCIE:

Y - sygnał wyjściowy (odpowiedź) filtru

Zf - wektor wartości końcowych sygnałów odpowiadających wektorowi wejściowemu *Zi*.

----- *** -----

delaysig - Opóźnianie spróbkowanego sygnału.

DELAYSIG Funkcja tworzy macierz zawierającą opóźnione próbki sygnału wejściowego. W aktualnej wersji pakietu MATLAB (7.1) funkcja ta jest nadal dostępna, jakkolwiek zaleca się zastępowanie jej działania przez tzw. macierze komórkowe (ang. *cell arrays*)

Wywołanie funkcji: **Y = DELAYSIG(X, D)**
lub: **Y = DELAYSIG(X, D1, D2)**

WEJŚCIE:

X - macierz o wymiarach *S*x*T*, zawierająca *S* wektorów wierszowych, każdy po *T* próbek sygnału wejściowego

D - maksymalne opóźnienie próbek; sygnał wyjściowy zawiera w wierszach próbki opóźnione odpowiednio o *0, 1, ..., D* kroków czasowych

D1, D2 - minimalne i maksymalne opóźnienie próbek; sygnał wyjściowy zawiera w wierszach próbki opóźnione odpowiednio o *D1*, *D1+1*, ..., *D2* kroków czasowych

WYJŚCIE:

Y - macierz, zawierająca w wierszach: sygnał wejściowy i sygnał opóźniony odpowiednio do wartości parametrów *D1* i *D2*

----- *** -----

hintonwb - Wykres Hintona wartości wag i progów sieci neuronowej.

HINTONWB Funkcja prezentuje wartości wag i progów sieci na specjalnym wykresie, tzw. wykresie Hintona – jako siatkę kolorowych kwadratów. Długość boku każdego kwadratu jest proporcjonalna do modułu wartości odpowiadającej mu wagi. W kolorze czerwonym są rysowane wagi o wartościach ujemnych, w kolorze zielonym – wagi o wartościach dodatnich.

Wywołanie funkcji: **HINTONWB(W, b, maxw, minw)**

WEJŚCIE:

W - macierz wag synaptycznych wybranej warstwy sieci (o wymiarach *S*x*R*, gdzie *S* jest liczbą neuronów w warstwie, zaś *R* – liczbą neuronów w warstwie poprzedniej)

B - wektor progów neuronów danej warstwy (wymiar *S*x*1*)

MAXW - maksymalna wartość wag synaptycznych; domyślnie max(max(abs(*W*)))

MINW - minimalna wartość wag synaptycznych; domyślnie MAXW/100

Przykład:

```
W = rands(4, 5);
b = rands(4, 1);
hintonwb(W, b)
```

----- *** -----

plotpc - Wykreśla granicę decyzyjną dla sieci preceptronowej

PLOTPC Funkcja wykreśla granicę decyzyjną w przestrzeni wag sieci neuronowej, złożonej z perceptronów. Wywołanie funkcji ma sens, gdy wymiarowość wektora wejściowego sieci jest nie większa niż trzy.

Wywołanie funkcji: **PLOTPC(W, b)**

WEJŚCIE:

W - macierz wag synaptycznych perceptronu (o wymiarach *S*x*R*, gdzie *S* jest liczbą neuronów w warstwie, zaś *R* – wymiarowością wektora wejść sieci, $R \leq 3$)

b - wektor wartości progów perceptronu (o wymiarach *S*x*1*)

Przykład:

Instrukcje poniżej definiują wejścia i wyjścia perceptronu i ukazują je na płaszczyźnie (za pomocą funkcji ‘plotpv’)

```
p = [0 0 1 1; 0 1 0 1];
t = [0 0 0 1];
plotpv(p, t)
```

Poniżej tworzymy perceptron, definiujemy wartości wag (bez treningu sieci) i ukazujemy granicę decyzyjną nałożoną na poprzedni wykres:

```
net = newp(minmax(p), 1);
net.iw{1,1} = [-1.2 -0.5];
net.b{1} = 1;
plotpc(net.iw{1,1}, net.b{1})
```

----- *** -----

plotperf - Wykreśla przebieg treningu sieci neuronowej

PLOPERF Funkcja wykreśla przebieg procesu treningu sieci neuronowej. Jeśli dane te są zawarte w strukturze *TR*, funkcja wykreśla również błędy dla zbiorów: testowego i walidacyjnego.

Wywołanie funkcji: **PLOTPERF(TR, GOAL, NAME, EPOCH)**

WEJŚCIE:

TR - wektor wartości funkcji błędu w poszczególnych krokach treningu, uzyskany w wyniku wywołania funkcji 'train'

GOAL - wartość kryterium stopu; domyślnie GOAL = NaN

NAME - nazwa funkcji trenującej sieć; domyślnie NAME = ''

EPOCH - liczba epok treningu; domyślnie długość wektora TR

Przykład:

Instrukcje poniżej definiują wejścia i wyjścia sieci, jak również zbiór walidacyjny sieci:

```
P = 1:8; T = sin(P);
VV.P = P; VV.T = T+rand(1,8)*0.1;
```

Tworzenie i trening sieci jest wykonywane w kolejnych instrukcjach:

```
net = newff(minmax(P), [4 1], {'tansig', 'tansig'});
[net, tr] = train(net, P, T, [], [], VV);
```

Funkcja PLOTPERF jest wywoływana podczas treningu, jakkolwiek można ją również wywołać po zakończeniu treningu:

```
plotperf(tr)
```

----- *** -----

Część II. Sieci współzawodniczące i samoorganizujące się

I. Funkcje, używane do tworzenia sieci neuronowych współzawodniczących i samoorganizujących się

newc - Tworzenie jednowarstwowej sieci neuronowej, złożonej z neuronów współzawodniczących.

NEWC Funkcja tworzy jednowarstwową sieć neuronową, złożoną z zadanej liczby neuronów współzawodniczących (ang. *competitive layer*)

Wywołanie funkcji: **NET = NEWC(PR, S, KLR, CLR)**

WEJŚCIE:

PR - jest macierzą o wymiarach *R*x*2*, gdzie *R* jest liczbą wejść sieci (współrzędnych wektorów wejściowych); pierwsza kolumna zawiera minimalne wartości kolejnych współrzędnych wektorów wejściowych, druga – maksymalne wartości tych współrzędnych

S - liczba neuronów sieci

KLR - stała szybkości uczenia sieci (tzw. stała Kohonena, ang. *Kohonen learning rate*), wykorzystywana do korekty wag synaptycznych neuronów-zwycięzców; domyślna wartość KLR = 0.01

CLR - stała (tzw. stała sumienia, ang. *conscience learning rate*), która zapobiega sytuacji, w której niektóre neurony w sieci są całkowicie pomijane w procesie

treningu (gdy np. początkowe położenie tych neuronów jest odległe od skupisk prezentowanych wzorców, co powoduje, że te neurony nigdy nie staną się zwycięzcami); domyślna wartość KLR = 0.001

WYJŚCIE:

NET - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry sieci współzawodniczącej.

Przykład:

Podane niżej wywołanie funkcji tworzy sieć złożoną z pojedynczej warstwy neuronów współzawodniczących o dwóch wejściach. Wartości współrzędnych wektorów podawanych na każde z wejść są zawarte w przedziale [0, 1]. Do treningu sieci zostaną użyte cztery wektory, tworzące macierz *P*:

```
P = [.1 .8  .1 .9; .2 .9 .1 .8];
```

Sieć współzawodnicząca będzie w tym przykładzie użyta do podziału zbioru wektorów na dwie klasy (liczba neuronów w wywołaniu funkcji tworzącej sieć wynosi 2). Funkcja 'newc' tworzy sieć współzawodniczącą, zaś wywołana kolejno funkcja 'train' wyznacza optymalne wagi synaptyczne tej sieci.

```
net = newc([0 1; 0 1], 2);
net = train(net, P);
```

Następnie dokonujemy symulacji działania sieci współzawodniczącej oraz przetwarzamy wartości wyjść sieci na indeksy (numery) klas:

```
Y = sim(net, P)
Yc = vec2ind(Y)
```

----- *** -----

newsom - Tworzenie samoorganizującej się mapy topologicznej.

NEWSOM Funkcja tworzy jednowarstwową sieć neuronową o postaci samoorganizującej się mapy topologicznej (ang. *Self-Organizing Map*, *SOM*). Neurony sieci są połączone z wektorem wejściowym za pomocą połączeń synaptycznych (z wagami synaptycznymi), jednak nie mają progów wzbudzenia. Sieć typu *SOM* jest wykorzystywana w zadaniach klasyfikacyjnych.

Wywołanie funkcji: **NET = NEWSOM(PR, [D1, D2, ...], TFCN, DFCN, OLR, OSTEPS, TLR, TNS**)

WEJŚCIE:

PR - jest macierzą o wymiarach *Rx2*, gdzie *R* jest liczbą wejść sieci (współrzędnych wektorów wejściowych); pierwsza kolumna zawiera minimalne wartości kolejnych współrzędnych wektorów wejściowych, druga – maksymalne wartości tych współrzędnych

Di - liczba neuronów w *i*-tym wymiarze sieci, dla sieci liniowych jest to pojedyncza wartość; domyślna wartość tego parametru to [5 8], co odpowiada sieci dwuwymiarowej (siatce rozpiętej na płaszczyźnie) zawierającej 40 neuronów

TFCN - nazwa funkcji określającej topologię sieci (zmienna tekstowa); domyślnie *TFCN* = 'hextop'; dopuszczalne wartości tego parametru to: 'hextop', 'gridtop' i 'randtop'

DFCN - nazwa funkcji obliczającej odległość neuronu sieci od wektora wejściowego (zmienna tekstowa); domyślnie *DTFCN* = ‘linkdist’; dopuszczalne wartości tego parametru to: ‘linkdist’, ‘dist’ i ‘mandist’

OLR - stała szybkości treningu zgrubnego (fazy porządkowania) mapy topologicznej; domyślnie *OLR* = 0.9

OSTEPS - liczba cykli (kroków) treningowych podczas porządkowania mapy; domyślnie *OSTEPS* = 1000

TLR - stała szybkości drugiej fazy uczenia sieci (tzw. „tuningu”); domyślnie *TLR* = 0.02

TND - promień sąsiedztwa neuronów mapy (obliczany przez funkcję odległości, *DFCN*), wykorzystywany w fazie tzw. „tuningu” sieci; wartość domyślna *TND* = 1

WYJŚCIE:

NET - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry samoorganizującej się mapy topologicznej.

Przykład:

Zdefiniowane poniżej wektory wejściowe są położone w przestrzeni dwuwymiarowej wewnątrz prostokąta, wyznaczonego granicami: [0, 2] (w osi *x*) i [0, 1] (w osi *y*). Wektory te zostaną użyte do treningu dwuwymiarowej mapy topologicznej o wymiarach 3x5 neuronów.

```
P = [rand(1,400)*2; rand(1,400)];
net = newsom([0 2; 0 1], [3 5]);
plotsom(net.layers{1}.positions)
```

W kolejnej sekwencji instrukcji realizujemy trening sieci neuronowej i wykreślamy punkty użyte do treningu sieci na tle siatki wyznaczonej przez neurony sieci SOM.

```
net = train(net, P);
plot(P(1,:), P(2,:), '.g', 'markersize', 20)
hold on
plotsom(net.iw{1,1}, net.layers{1}.distances)
hold off
```

----- *** -----

newlvq - Tworzenie dwuwarstwowej sieci neuronowej, uczącej się kwantyzacji wektorowej (sieci typu *LVQ*).

NEWLVQ Funkcja tworzy dwuwarstwową sieć neuronową, realizującą kwantyzację wektorową danych wejściowych (ang. *Learning Vector Quantization, LVQ*). Pierwsza warstwa składa się z neuronów współzawodniczących, druga (wyjściowa) – z neuronów o liniowych funkcjach aktywacji. Sieci typu *LVQ* są wykorzystywane w problemach klasyfikacyjnych.

Wywołanie funkcji: **NET = NEWLVQ(PR, S1, PC, LR, LF)**

WEJŚCIE:

PR - jest macierzą o wymiarach *R*x*2*, gdzie *R* jest liczbą wejść sieci (współrzędnych wektorów wejściowych); pierwsza kolumna zawiera minimalne wartości kolejnych współrzędnych wektorów wejściowych, druga – maksymalne wartości tych współrzędnych

S1 - liczba neuronów w warstwie ukrytej sieci

PC - wektor zawierający *S2* elementów (*S2* jest liczbą klas, do których należą wektory treningowe), wartości kolejnych elementów oznaczają częstość występowania wektorów danej klasy w zbiorze treningowym; suma elementów wektora *PC* powinna być równa jedności

LR - stałą szybkości uczenia; domyślnie *LR* = 0.01

LF - nazwa funkcji, wykorzystywanej do treningu sieci (zmienna tekstowa); domyślnie *LF* = 'learnlv2'; dopuszczalne wartości parametru *LF* to: 'learnlv1' i 'learnlv2'

WYJŚCIE:

NET - struktura (obiekt) zawierająca opis architektury, metod treningu, wartości liczbowe wag oraz inne parametry dwuwarstwowej sieci typu *LVQ*.

Przykład:

Zdefiniujmy macierz *P* dwuelementowych wektorów wejściowych i wektor *Tc* indeksów (kodów) klas, do których należą wektory wejściowe:

```
P = [-3 -2 -2  0  0  0  0 +2 +2 +3; 0 +1 -1 +2 +1 -1 -2 +1 -1  0];
Tc = [1 1 1 2 2 2 2 1 1 1];
```

Dokonujemy konwersji kodów klas do wektora pożądanych wyjść sieci, *T*. Następnie tworzymy sieć typu *LVQ*, zawierającą 4 wektory ukryte (w warstwie wejściowej). Zakres wejść sieci wyznaczamy bezpośrednio z macierzy *P* (funkcja 'minmax'), zaś wektor częstości występowania wektorów poszczególnych klas to [.6, .4] (w wektorze *Tc* sześć elementów oznacza kod klasy *1*, cztery elementy – kod klasy *2*). W dalszej kolejności trenujemy sieć *LVQ*:

```
T = ind2vec(Tc);
net = newlvq(minmax(P), 4, [.6 .4]);
net = train(net, P, T);
```

Wyniki treningu można poznać po wykonaniu symulacji sieci i konwersji jej wyjść do wektora indeksów klas:

```
Y = sim(net, P)
Yc = vec2ind(Y)
```

----- *** -----

II. Funkcje obliczania odległości, używane przez sieci współzawodniczące

boxdist - Wyznaczanie odległości pomiędzy neuronami sieci neuronowej.

BOXDIST Funkcja oblicza wzajemne odległości pomiędzy wektorami, stanowiącymi kolumny macierzy *POS*. W szczególności jest ona wykorzystywana do wyznaczenia odległości pomiędzy neuronami sieci współzawodniczącej, których położenie jest zadane w macierzy *POS*. Odległości pomiędzy punktami (wektorami) P_i i P_j są określone wzorem:

$$D_{ij} = \max(\text{abs}(P_i - P_j))$$

Wywołanie funkcji: **D = BOXDIST(POS)**

WEJŚCIE:

POS - macierz o wymiarach *N*x*S*, której kolumny zawierają położenia wektorów w *N*-wymiarowej przestrzeni

WYJŚCIE:

D - symetryczna macierz o wymiarach *S*x*S*, zawierająca wzajemne odległości pomiędzy wektorami

Przykład:

W trójwymiarowej przestrzeni zdefiniujmy zbiór 10 wektorów, o losowych wartościach współrzędnych oraz obliczmy ich wzajemne odległości:

```
pos = rand(3, 10);
d = boxdist(pos)
```

----- *** -----

dist - Euklidesowa odległość pomiędzy wektorami.

DIST Funkcja wyznacza wzajemne odległości pomiędzy wektorami w *N*-wymiarowej przestrzeni. Do wyznaczania odległości pomiędzy dwoma wektorami (*x* i *y*) używana jest zależność, zapisana w notacji pakietu Matlab jako:

```
d = sum((x-y).^2).^0.5;
```

Funkcję obliczającą odległość pomiędzy neuronami w sieci współzawodniczącej można zmienić na ‘dist’, ustawiając odpowiednie pole struktury ‚NET’, tj. NET.layers{i}.distanceFcn = 'dist'. Funkcję obliczającą odległość pomiędzy wektorami wejściowymi a neuronami w sieci współzawodniczącej można zmienić, ustawiając w strukturze ‚NET’: NET.inputWeight{i,j}.weightFcn = 'dist'.

Wywołanie funkcji: **D = DIST(POS)**
lub: **D = DIST(W, P)**

WEJŚCIE:

POS - macierz o wymiarach *N*x*S*, której kolumny zawierają położenia wektorów w *N*-wymiarowej przestrzeni

W - macierz wag sieci o wymiarach *S*x*R*, którą utożsamiamy z położeniem neuronów sieci w *R*-wymiarowej przestrzeni

P - macierz wejść (wektorów wejściowych) sieci, o wymiarach *R*x*Q*; każdy z wektorów zajmuje jedną kolumnę macierzy

WYJŚCIE:

D - symetryczna macierz o wymiarach *S*x*S*, zawierająca wzajemne odległości pomiędzy wektorami; w przypadku wywołania funkcji z dwoma argumentami – macierz o wymiarach *S*x*Q*, zawierająca odległości pomiędzy wektorami wejściowymi a wektorami (wagami) sieci neuronowej

Przykład:

Poniższe instrukcje definiują losową macierz wagową (odpowiadającą położeniu czterech neuronów sieci) oraz wektor wejściowy *P* w trójwymiarowej przestrzeni wejść, a następnie wyznaczają odległości tego wektora od neuronów sieci:

```
W = rand(4, 3);
P = rand(3, 1);
Z = dist(W, P)
```

W trójwymiarowej przestrzeni zdefiniujmy zbiór 10 wektorów, o losowych wartościach współrzędnych oraz obliczmy ich wzajemne odległości:

```
pos = rand(3, 10);
d = dist(pos)
```

----- *** -----

mandist - Odległość pomiędzy wektorami według normy ‘Manhattan’.

MANDIST Funkcja wyznacza wzajemne odległości w/g normy ‘Manhattan’ pomiędzy wektorami w *N*-wymiarowej przestrzeni. Do wyznaczania odległości pomiędzy dwoma wektorami (*x* i *y*) używana jest zależność, zapisana w notacji pakietu Matlab jako:

```
d = sum(abs(x-y));
```

Funkcję obliczającą odległość pomiędzy neuronami w sieci współzawodniczącej można zmienić na ‘mandist’, ustawiając odpowiednie pole struktury ‚NET’, tj. NET.layers{i}.distanceFcn = 'mandist'. Funkcję obliczającą odległość pomiędzy wektorami wejściowymi a neuronami w sieci współzawodniczącej można zmienić, ustawiając w strukturze ‚NET’: NET.inputWeight{i,j}.weightFcn = 'mandist'.

Wywołanie funkcji: **D = MANDIST(POS)**
lub: **D = MANDIST(W, P)**

WEJŚCIE:

- *POS* - macierz o wymiarach *N*x*S*, której kolumny zawierają położenia wektorów w *N*-wymiarowej przestrzeni
- *W* - macierz wag sieci o wymiarach *S*x*R*, którą utożsamiamy z położeniem neuronów sieci w *R*-wymiarowej przestrzeni
- *P* - macierz wejść (wektorów wejściowych) sieci, o wymiarach *R*x*Q*; każdy z wektorów zajmuje jedną kolumnę macierzy

WYJŚCIE:

- *D* - symetryczna macierz o wymiarach *S*x*S*, zawierająca wzajemne odległości pomiędzy wektorami; w przypadku wywołania funkcji z dwoma argumentami – macierz o wymiarach *S*x*Q*, zawierająca odległości pomiędzy wektorami wejściowymi a wektorami (wagami) sieci neuronowej

Przykład:

Poniższe instrukcje definiują losową macierz wagową (odpowiadającą położeniu czterech neuronów sieci) oraz wektor wejściowy *P* w trójwymiarowej przestrzeni wejść, a następnie wyznaczają odległości tego wektora od neuronów sieci:

```
W = rand(4, 3);
P = rand(3, 1);
Z = mandist(W, P)
```

W trójwymiarowej przestrzeni zdefiniujmy zbiór 10 wektorów, o losowych wartościach współrzędnych oraz obliczmy ich wzajemne odległości:

```
pos = rand(3, 10);
d = mandist(pos)
```

----- *** -----

linkdist - Wyznaczanie odległości pomiędzy neuronami sieci.

LINKDIST Funkcja wyznacza wzajemne odległości pomiędzy neuronami sieci; jest przede wszystkim stosowana w odniesieniu do samoorganizujących się map topologicznych (sieci *SOM*). Odległości pomiędzy punktami (wektorami) P_i i P_j są określone wzorem:

d_{ij} = 0, gdy i == j,
= 1, gdy sum((P_i-P_j).^2).^0.5 jest <= 1
= 2, gdy istnieje indeks *k*, taki, że: $d_{ik} = d_{kj} = 1$
= 3, gdy istnieją indeksy *k1* i *k2*, takie, że: $d_{ik1} = d_{k1k2} = d_{k2j} = 1$
= N, gdy istnieją indeksy *k1* ... *kN*, takie, że: $d_{ik1} = d_{k1k2} = ... = d_{kNj} = 1$
= S, gdy nie zachodzi żadna z powyższych zależności

Funkcję obliczającą odległość pomiędzy neuronami w sieci współzawodniczącej można zmienić na ‘mandist’, ustawiając odpowiednie pole struktury ‚NET’, tj. NET.layers{i}.distanceFcn = 'linkdist'.

Wywołanie funkcji: **D = LINKDIST(POS)**

WEJŚCIE:
POS - macierz o wymiarach *N*x*S*, której kolumny zawierają położenia wektorów w *N*-wymiarowej przestrzeni

WYJŚCIE:
D - symetryczna macierz o wymiarach *S*x*S*, zawierająca wzajemne odległości pomiędzy wektorami (neuronami sieci neuronowej)

Przykład:

W trójwymiarowej przestrzeni zdefiniujmy zbiór 10 wektorów, o losowych wartościach współrzędnych oraz obliczmy ich wzajemne odległości:

```
pos = rand(3, 10);
d = linkdist(pos)
```

----- *** -----

III. Funkcje pomocnicze

vec2ind - Funkcja konwersji wektorów zapisanych w kodzie 1-z-N na indeksy.

VEC2IND Funkcja zwraca indeksy tych współrzędnych wektorów kolumnowych (zawartych w macierzy *VEC*), które mają jednostkową wartość. Pozostałe współrzędne każdego z wektorów muszą być zerowe – wektory można więc interpretować jako liczby w kodzie *1-z-N*.

Wywołanie funkcji: **IND = VEC2IND(VEC)**

WEJŚCIE:
VEC - macierz złożona z wektorów kolumnowych; każdy z wektorów zawiera dokładnie jedną jedynkę, zaś pozostałe jego współrzędne są zerowe (wektor może być interpretowany jako zapis liczby w kodzie *1-z-N*)

WYJŚCIE:

IND - wektor wierszowy, który zawiera numery (indeksy) wierszy macierzy *VEC* o wartościach równych 1

Przykład:

Macierz ‘vec’ zawiera cztery wektory (kolumnowe); tylko jedna współrzędna każdego wektora jest równa 1, pozostałe są zerowe. Funkcja ‘vec2ind’ zwraca numery współrzędnych wektorów o jednostkowych wartościach:

```
vec = [1 0 0 0; 0 0 1 0; 0 1 0 1]
ind = vec2ind(vec)
```

----- *** -----

ind2vec - Funkcja konwersji indeksów na wektory zapisane w kodzie 1-z-N.

IND2VEC Funkcja zwraca macierz rzadką, zawierająca dokładnie jedną jedynkę w każdej kolumnie. Położenie jedynki (tzn. wiersz, w którym występuje ta wartość) jest określone przez numer podany w odpowiedniej kolumnie wektora wejściowego ‘*IND*’. Można więc interpretować każdą kolumnę macierzy ‘*VEC*’ jako liczbę, zapisaną w kodzie *1-z-N*.

Wywołanie funkcji: **VEC = IND2VEC(IND)**

WEJŚCIE:

IND - wektor wierszowy, który zawiera liczby naturalne (indeksy)

WYJŚCIE:

VEC - macierz złożona z wektorów kolumnowych; każdy z wektorów zawiera dokładnie jedną jedynkę, zaś pozostałe jego współrzędne są zerowe (wektor może być interpretowany jako zapis liczby w kodzie *1-z-N*)

Przykład:

Wektor ‘ind’ zawiera cztery indeksy; funkcja ‘ind2vec’ zwraca macierz o wymiarach *3*x*4* (zapisaną w notacji macierzy rzadkiej), zawierającą reprezentację indeksów w kodzie 1-z-3:

```
ind = [1 3 2 3]
vec = ind2vec(ind)
```

----- *** -----

nngenc - Funkcja generacji zadanej liczby skupisk punktów w przestrzeni.

NNGENC Funkcja generuje skupiska (klastry) o zadanej wariancji, złożone z zadanej liczby wektorów. Skupiska są położone w granicach obszarów, zdefiniowanych przez użytkownika. W aktualnej wersji pakietu MATLAB (7.1) funkcja ta jest nadal dostępna, jakkolwiek nie jest udokumentowana i może nie wystąpić w kolejnej wersji biblioteki.

Wywołanie funkcji: **V = NNGENC(X, C, N, D)**

WEJŚCIE:

X - macierz o wymiarach *R*x*2*, gdzie *R* jest liczbą współrzędnych generowanych wektorów; każdy wiersz pierwszej kolumny zawiera

minimalną wartość odpowiedniej współrzędnej obszaru, w którym mieszczą się generowane wektory; każdy wiersz drugiej kolumny – maksymalną wartość odpowiedniej współrzędnej centrum skupiska

C - liczba skupisk

N - liczba punktów (wektorów) w każdym skupisku

D - odchylenie standardowe wygenerowanych punktów wokół centrum skupiska (argument opcjonalny, wartość domyślna $d = 1.0$)

WYJŚCIE:

V - macierz zawierająca *C*x*N* *R*-elementowych wektorów, położonych w *C* skupiskach, zawartych w obszarze określonym przez współrzędne macierzy *X*; każde skupisko zawiera *N* punktów, rozmieszczonych losowo wokół centrum, z odchyleniem standardowym *D*

Przykład:

Instrukcje poniżej generują osiem skupisk punktów na płaszczyźnie (liczba współrzędnych generowanych wektorów wynosi dwa). Każde skupisko zawiera po sześć punktów, rozmieszczonych wokół centrum skupiska z odchyleniem standardowym równym 0.5. Centra skupisk położone są wewnątrz prostokąta, którego współrzędna $x \in [-10, 10]$, zaś współrzędna $y \in [-5, 5]$.

```
X = [-10 10; -5 5];
V = nngenc(X, 8, 6, 0.5);
plot(V(1,:), V(2,:), '+')
```

----- *** -----

plotsom - Wykreśla dwuwymiarową mapę topologiczną (sieć typu SOM)

PLOTSOM Funkcja wykreśla położenie neuronów dwu- lub trójwymiarowej mapy topologicznej. W alternatywnej wersji funkcja wykreśla neurony oraz połączenia pomiędzy tymi neuronami, których wzajemne odległości są mniejsze od zadanej.

Wywołanie funkcji: **PLOTSOM(POS)**
lub: **PLOTSOM(W, D, ND)**

WEJŚCIE:

POS - macierz o wymiarach *N*x*S*, której kolumny zawierają położenia neuronów sieci w *N*-wymiarowej przestrzeni

W - macierz wag neuronów sieci o wymiarach *S*x*R*, którą utożsamiamy z położeniem neuronów sieci w *R*-wymiarowej przestrzeni

D - symetryczna macierz o wymiarach *S*x*S*, zawierająca wzajemne odległości pomiędzy wektorami z macierzy *W*

ND - zadana odległość sąsiedztwa; funkcja łączy te neurony, których wzajemna odległość jest nie większa od *ND*; jest to parametr opcjonalny o domyślnej wartości = 1

Przykład:

Zaleca się sprawdzenie działania funkcji ‘plotsom’ dla sieci o różnych topologiach:

```
pos = hextop(5, 6); plotsom(pos)
pos = gridtop(4, 5); plotsom(pos)
pos = randtop(18, 12); plotsom(pos)
```

```
pos = gridtop(4, 5, 2); plotsom(pos)
pos = hextop(4, 4, 3); plotsom(pos)
```

Uwaga!!!

Pełny opis funkcji biblioteki narzędziowej pt. „*Neural Network Toolbox. Version 5*” (w angielskiej wersji językowej) można znaleźć w podkatalogu dokumentacji, utworzonym w katalogu roboczym pakietu *Matlab 7.1*. Możliwe jest również uzyskanie pliku dokumentacji bezpośrednio ze strony internetowej pakietu:

http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/nnet.shtml.