Кабінет Міністрів України
Національний університет біоресурсів і природокористування України
Кафедра автоматики і робототехнічних систем
ім. акад. І.І. Мартиненка

Лисенко В.П., Штепа В.М., Заєць Н.А.

СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ
НЕЙРОННІ МЕРЕЖІ
(ПАКЕТ NEURAL NETWORKS TOOLBOX)

(навчальний посібник)

Для студентів напряму підготовки

6.050101 – «Комп'ютерні науки»

6.050202 – «Автоматизація та комп'ютерно-інтегровані технології»

6.100101 – «Енергетика та електротехнічні системи в агропромисловому комплексі»

(денна та заочна форма навчання)

Київ – 2011

УДК 604.94(075.8)

У навчальному посібнику викладено лекційний матеріал занять із дисципліни «Системи штучного інтелекту». Основну увагу приділено представленню, із використанням характерних прикладів, практичній реалізації положень теорії нейромережевих систем.

Навчальний посібник призначено для використання при підготовці фахівців за напрямами підготовки: 6.050101 «Комп'ютерні науки», 6.050202 – «Автоматизація та комп'ютерно-інтегровані технології», 6.100101 – «Енергетика та електротехнічні системи в агропромисловому комплексі».

Рекомендовано до видання навчально-методичною радою факультету енергетики і автоматики Національного університету біоресурсів і природокористування України.

***Укладачі*:** кандидат технічних наук, професор Лисенко В.П., кандидат технічних наук, старший викладач Штепа В.М., кандидат технічних наук, старший викладач Заєць Н.А.
***Рецензенти*:** доктор технічних наук, професор Донченко М.І. (Національний технічний університет "КПІ"), доктор технічних наук, професор Котов Б.І. (Національний університет біоресурсів і природокористування України)

Навчальний посібник

СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ НЕЙРОННІ МЕРЕЖІ
(ПАКЕТ NEURAL NETWORKS TOOLBOX)

ДЛЯ СТУДЕНТІВ НАПРЯМОМ ПІДГОТОВКИ:
6.050101 «Комп'ютерні науки», 6.050202 – «Автоматизація та комп'ютерно-інтегровані технології», 6.100101 – «Енергетика та електротехнічні системи в агропромисловому комплексі»

Укладачі: ЛИСЕНКО Віталій Пилипович
ШТЕПА Володимир Миколайович
ЗАЄЦЬ Наталія Анатоліївна

Зав. видавничим центром НУБіП України А.П. Колесніков
Редактор В.М. Штепа (в авторській редакції)

Підписано до друку Формат 60×84 *1/16*
Ум. друк. арк. 2,8 Обл.-вид. арк. 2,25
Наклад 100 прим. Зам. № 3302
Видавничий центр НУБіП України
03041, Київ, вул. Героїв Оборони, 15

ЗМІСТ

Вступ

Пакет *Neural Networks Toolbox* містить засоби для проектування, моделювання, навчання і використання відомих парадигм апарату штучних нейронних мереж (ШНМ), від базових моделей персептрона до сучасних асоціативних мереж, що самоорганізуються. Пакет можна використовувати для вирішення різноманітних завдання, таких як обробка сигналів, нелінійне керування, фінансове моделювання, екологічно безпечне прогнозування тощо.

Для кожного типу архітектури і навчального алгоритму ШНМ є функції ініціалізації, навчання, адаптації, створення і моделювання, демонстрації і приклади застосування.

До складу пакету *Neural Networks* входять більше 150 різних функцій, утворюючи собою своєрідну макромову програмування і дозволяючи користувачеві створювати, навчати і використовувати різні типи НМ.

1 . Основні компоненти пакету Neural Networks Toolbox

» help nnet

введення команди дозволить отримати перелік весь пакет функцій. Для візуалізації довідки стосовно будь-якої функції можна використовувати команду

» help им’я функції

1.1. Функції активації (передаточні функції)

compet(X) – функція конкуренції – як аргумент використовує матрицю X, стовпці якої асоціюються з векторами входів. Повертає розріджену матрицю з одиничними елементами, індекси яких відповідають індексам найбільших елементів кожного стовпця.

Приклад

```
» X = [0.9 -0.6;0.1 0.4;0.2 -0.5;0 0.5]; » compet(X)
ans =
  (1.1)   1
  (4.1)   1
```

У формі **compet(code)**, де змінна **code** може набувати значень '**deriv**' (ім'я похідної функції), '**name**' (повне ім'я), '**output**' (діапазон виходу), '**active**' (можливий діапазон входів).

Приклад

```
» compet ('deriv')
  ans = //
» compet ('name')
ans =
Competitive
» compet ('output')
ans =
   01
```

» compet ('active') ans = -Inf Inf

Така функція використовується при створенні НМ з шаром нейронів, що «змагаються» (наприклад, у мережах зустрічного розповсюдження).

hardlim(X) – порогова функція активації; аргумент має той же сенс, що і для попередньої команди. Повертає матрицю, розмір якої дорівнює розміру матриці X, а елементи мають значення 0 або 1 – залежно від знаку відповідного елементу X.

Приклад

```
» X = [0.9 -0.6;0.1 0.4;0.2 -0.5;0 0.5];
 » hardlim(X) ans =
  1     0
  1     1
  1     0
  1     1
```

У формі **hardlim(code)** повертає інформацію, аналогічну розглянутій для команди **compet**.

hardlims(X) – знакова або сигнатурна функція активації; діє так само, як функція **hardlim(X),** але повертає значення - 1 або +1.

logsig(X) – сигмоїдальна логістична функція. Повертає матрицю, елементи якої є значеннями логістичної функції від аргументів – елементів матриці X.

poslin(X) – повертає матрицю значень напівлінійної функції.

purelin(X) – повертає матрицю значень лінійної функції активації.

radbas(X) – повертає матрицю значень радіальної базисної функції.

satlin(X) – повертає матрицю значень напівлінійної функції з насиченням.

satlins(X) – повертає матрицю значень лінійної функції з насиченням.

softmax(X) – повертає матрицю, елементи якої розраховуються за формулою:

$$\frac{e^{x_{ij}}}{\sum_{i=1}^{N} e^{x_{ij}}}, \qquad (1.1)$$

де: **N –** число рядків матриці-аргументу X.

tansig(X) – повертає матрицю значень сигмоїдальної (гіперболічний тангенс) функції.

tribas(X) – повертає матрицю значень трикутної функції належності.

dhardlim(X,Y) – похідна порогової функції активації. Аргументами є матриця входів X і матриця виходів Y; матриці мають однаковий розмір. Повертається матриця того ж розміру з нульовими елементами.

dhardlms(X,Y) – похідна знакової функції активації. Повертається матриця з нульовими елементами.

dlogsig(X,Y) – похідна сигмоїдальної логістичної функції.

Приклад

»X = [0.1; 0.8;-0.7];

```
 » Y = logsig(X)
Y= 0.5250
   0.6900
   0.3318
 »dY_dX = dlogsig(X,Y)
dY_dX =
  0.2494
  0.2139
  0.2217
```

dposlin(X,Y) – похідна напівлінійної функції. Повертається матриця з елементами, рівними одиниці для відповідних додатніх елементів матриці-аргументу Y і рівними нулю в протилежному випадку.

dpurelin(X,Y) – похідна лінійної функції активації. Повертається матриця з одиничними елементами.

dradbas(X,Y) – похідна радіальної базисної функції.

dsatlin(X,Y) – повертає матрицю значень похідної напівлінійної функції з насиченням. Елементи такої матриці – одиниці, якщо відповідні елементи матриці Y належать інтервалу (0,1), і нулі в протилежному випадку.

dsatlins(X,Y) – повертає матрицю значень похідної лінійної функції з насиченням. Елементи такої матриці – одиниці, якщо відповідні елементи матриці Y належать інтервалу (-1; 1), і нулі в протилежному випадку.

dtansig(X,Y) – повертає матрицю значень похідної сигмоїдальної функції гіперболічного тангенса.

dtribas(X,Y) – повертає матрицю значень похідної трикутної функції активації.

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведені у прикладах функції активації.
2. Згідно завдання викладача реалізувати 5 фунцій активації.

1.2. Функції навчання нейронних мереж

Дозволяють встановити алгоритм і параметри навчання НМ заданою конфігурація за бажанням користувача. У групу входять наступні функції.

[net,tr] = trainbfg(net,Pd,Tl,Ai,Q,TS,VV,TV) – функція навчання, що реалізовує різновид квазіньютонівського алгоритму зворотного розповсюдження помилки (BFGS). Аргументи функції:

net – ім'я навчаної НМ;

Pd – найменування масиву входів навчальної вибірки;

TI – масив цільових значень виходів;

Ai – матриця початкових умов вхідних затримок;

Q – кількість навчальних пар в одному циклі навчання;

TS – вектор тимчасових інтервалів;

VV – порожній ([]) масив або масив перевіряльних даних;

TV – порожній ([]) масив або масив тестових даних.

Функція повертає навчену нейронну мережу **net** і набір записів **tr** для кожного циклу навчання (**tr.epoch –** номер циклу, **tr.perf –** поточна помилка навчання, **tr.vperf –** поточна помилка для перевірочної вибірки, **tr.tperf –** поточна помилка для тестувальної вибірки).

Процес навчання відбувається відповідно до значень наступних параметрів (у дужках приведені значення за умовчанням):

net.trainParam.epochs (100) – задана кількість циклів навчання;

net.trainParam.show (25) – кількість циклів для показу проміжних результатів;

net.trainParam.goal (0) – цільова помилка навчання;

net.trainParam.time () – максимальний час навчання в секундах;

net.trainParam.min_grad () – цільове значення градієнта;

net.trainParam.max_fail (5) – максимально допустима кратність перевищення помилки перевірочної вибірки в порівнянні з досягнутим мінімальним значенням;

net.trainParam.searchFcn (**'srchcha'**) **–** ім'я використовуваного одномірного алгоритму оптимізації.

Структури і розміри масивів:

Pd – масив, кожен елемент которого P{i,j,ts} є матрицею;

TI – масив, кожен елемент якого P{i,ts} є матрицею;

Ai – масив, кожен елемент якого Ai{i,k} є матрицею

де: Ni = **net.numInputs** (кількість входів мережі),

NI = **net.numLayers** (кількість її шарів),

LD = **net.numLayerDelays** (кількість шарів затримки),

Ri = **net.inputsi.size** (розмір і-го входу),

Si = **net.layersi.size** (розмір і-го шару),

Vi = **net.targetsi.size** (розмір цільового вектора),

Dij = **Ri * length(net.inputWeightsi,j.delays)**.

Якщо масив **VV –** не порожній, то він повинен мати структуру, визначену наступними компонентами:

VV.PD – затримані значення входів перевірочної вибірки;

VV.TI – цільові значення;

VV.Ai – початкові вхідні умови;

VV.Q – кількість перевірочних пар в одному циклі навчання;

VV.TS – тимчасові інтервали перевірочної вибірки.

Ці параметри використовуються для задання зупинки процесу навчання у випадках, коли помилка для перевірочної вибірки не зменшується або починає зростати.

Структуру, аналогічну структурі масиву **VV**, має масив **TV**.

Така функція, задана у формі **trainbfg(code)**, повертає для значень аргументу, відповідно, '**pnames**' і '**pdefaults**', імена параметрів навчання і їх значення за замовчунням.

Для використання функції в НМ, встановлених користувачем, необхідно:

1) встановити параметр **net.trainFcn = 'trainbfg'** (при цьому параметри алгоритму будуть задані за умовчанням);

2) встановити необхідні значення параметрів (**net.trainParam**).

Процесс навчання зупиняється у разі виконання будь-якого з наступних умов:

- перевищена задана кількість циклів навчання (**net.trainParam.epochs**);
- перевищений заданий час навчання (**net.trainParam.time**);
- помилка навчання стала менша заданою (**net.trainParam.goal**);
- градієнт став менший заданого (**net.trainParam.mingrad**);
- зростання помилки перевірочної вибірки в порівнянні з досягнутим мінімальним перевищило задане значення (**net.trainParam.maxfail**).

Приклад

```
» P = [0 1 2 3 4 5]; % Задання вхідного вектора
» T = [0 0 0 1 1 1]; % Задання цільового вектора
» % Створення і тестування мережі
» net = newff([0 5],[2 1], { 'tansig',logsig'}, ‘traincgf’);
» a = sim(net,P)
a=
0.0586    0.0772    0.0822    0.0870    0.1326    0.5901 »
% Навчання з новими параметрами і повторне тестування
»net.trainParam.searchFcn = 'srchcha';
» net.trainParam.epochs = 50;
» net.trainParam.show = 10;
» net.trainParam.goal = 0.1;
» net = train(net,P,T);
TRAINCGF-srchcha, Epoch 0/50, MSE 0.295008/0.1, Gradient 0.623241/1e-
006
TRAINCGF-srchcha, Epoch 1/50, MSE 0.00824365/0.1, Gradient
0.0173555/1e-006
    TRAINCGF, Performance goal met.
    » a = sim(net,P)
      a= 0.0706    0.1024    0.1474    0.9009    0.9647    0.9655
```

У даному прикладі створена багатошарова НМ, навчена з установками за умовчанням, спочатку показала поганий результат відображення навчальної вибірки, але після зміни параметрів і повторного навчання мережі результат став цілком прийнятним.

[net,tr] = trainbr(net,Pd,Tl,Ai,Q,TS,VV) – функція, що реалізує так званий Баєсовський метод навчання, суть котрого полягає в підстроюванні вагів і

зсувів мережі на основі алгоритму Льовенберга-Марквардта. Даний алгоритм мінімізує комбінацію квадратів помилок і вагів з вибором найкращої (для набуття найкращих узагальнювальних властивостей мережі). Ця процедура відома як *Баєсовська регуляризація*.

Аргументи, параметри, величини, що повертаються, і їх використання такі ж, як у попередньої функції.

[net,tr] = traincgb(net,Pd,TI,Ai,Q,TS,VV) – функція навчання НМ, що реалізовує різновид алгоритму зв'язаних градієнтів (так званий метод Powell-Beale).

[net,tr] = traincgf(net,Pd,TI,Ai,Q,TS,VV) – функція навчання НМ, що реалізовує різновид алгоритму зворотного розповсюдження помилки у поєднанні з методом оптимізації Флетчера-Поуелла.

[net,tr] = traincgp(net,Pd,TI,Ai,Q,TS,VV) – те ж, що в попередньому випадку, але із використанням методу Polak-Ribiere.

[net,tr] = traingd(net,Pd,TI,Ai,Q,TS,VV) – функція, яка реалізує «класичний» алгоритм зворотного розповсюдження помилки.

[net,tr] = traingda(net,Pd,TI,Ai,Q,TS,VV) – те ж, що у попередньому випадку, але з адаптацією коефіцієнта швидкості навчання.

[net,tr] = traingdm(net,Pd,TI,Ai,Q,TS,VV) – функція, яка реалізує модифікований алгоритм зворотного розповсюдження помилки з введеною «інерційністю» корекції вагів і зсуву.

[net,tr] = traingdx(net,Pd,TI,Ai,Q,TS,VV) – функція, яка реалізує комбінований алгоритм навчання, об'єднує особливості двох наведених вище.

[net,tr] = trainlm(net,Pd,TI,Ai,Q,TS,VV) – функція повертає ваги і зсув НМ, використовуючи алгоритм оптимізації Льовенберга-Марквардта.

[net,tr] = trainoss(net,Pd,TI,Ai,Q,TS,VV) – функція, яка реалізує різновид алгоритму зворотного розповсюдження помилки з використанням методу січних.

[net,tr] = trainrp(net,Pd,TI,Ai,Q,TS,VV) – функція, яка реалізує різновид алгоритму зворотного розповсюдження помилки, так званий пружний алгоритм зворотного розповсюдження (resilient backpropagation algorithm, RPROP).

[net,tr] = trainscg(net,Pd,TI,Ai,Q,TS,VV) – дана функція повертає ваги і зсув НМ, використовуючи алгоритм масштабуючих зв'язаних градієнтів.

[net,tr] = trainwb(net,Pd,TI,Ai,Q,TS,VV) – функція коректує ваги і зсув мережі відповідно до заданої функції навчання нейронів.

[net,tr] = trainwb1(net,Pd,TI,Ai,Q,TS,VV) – те ж, що і попередня функція, але одночасно на вхід мережі пред'являється тільки один вектор входу.

[net,Ac,El] = adaptwb(net,Pd,TI,Ai,Q,TS) – функція адаптації вагів і зсувів НМ. Використовується спільно з функціями *newp* і *newlin*. Повертає масив виходів шарів *Ac* і масив помилок шарів *EI*.

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведені у прикладах функції навчання НМ.
2. Згідно завдання викладача реалізувати 3 фунції навчання НМ.

1.3.Функції налаштування шарів нейронів

Функції даної групи є допоміжними при роботі з деякими розглянутими функціями навчання НМ (наприклад, **trainwb**, **trainwb1**, **adaptwb**), а також використовуються при налаштуваннях одношарових НМ (персептронів, шарів Кохонена і т. п.).

[dB,LS] = learncon(B,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функція налаштування вагів з введенням «відчуття справедливості».

Аргументи:

B – S×1 вектор зсувів;

P – 1×Q вхідний вектор;

Z – S×Q матриця зважених входів;

N – S×Q матриця входів;

A – S×Q матриця вихідних векторів;

T – S×Q матриця цільових векторів;

E – S×Q матриця помилок;

gW – S×R градієнт критерію ефективності по відношенню до вектора вагів;

gA – S×Q градієнт критерію ефективності по відношенню до вектора виходу;

D – S×S матриця відстаней між нейронами;

LP – параметр навчання, LP = [];

LS – стан навчання, на початку - [];

Величини, що повертаються:

dB – Sx1 вектор змін вагів (або зсувів);

LS – новий стан навчання.

Функція у формі **learncon(code)** повертає наступну інформацію:

при аргументі **'pnames' –** імена параметрів навчання,

при **'pdefaults' –** значення параметрів за умовчанням,

при **'needg' –** 1, якщо ця функція використовує **gW** або **gA**.

Алгоритм виконання функції спочатку обчислює «відчуття справедливості» нейрона по виразу **c= (1-lr)*c+lr*a**, а вже потім коректує вагу відповідно до формули

b = exp(1-log(c)) - b.

Приклад

```
» a = rand(3,1);
 » b = rand(3,1);
»lp.lr=0.5; % Задання параметру навчання
»dW = lear ncon(b,[],[],[],a,[],[],[],[],[],lp,[])
dW =
  0.3449
  0.7657
  0.5405
```

learngd – функція корекції вагів і зсувів, яка реалізує градієнтний алгоритм оптимізації.

Запис: **[dW,LS] = learngd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) [db,LS]=**
= learngd(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)
info = learngd(code)

Опис. Аргументи функції: **W** – матриця вагів або вектор зсуву, решта аргументів – як у попередньої функції.

Learngdm – функція практично аналогічна попередній, але використано інший алгоритм оптимізації – градієнтний метод з інерційною складовою.

[dW,LS] = learnh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функція корекції вагів, що використовує правило Хебба.

[dW,LS] = learnhd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функція реалізує модифікацію правила Хебба.

[dW,LS] = learnis(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функція підстроювання вагів «вхідної зірки» (нейрона Гроссберга).

[dW,LS] = learnk(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функція підстроювання вагів Кохонена.

[dW,LS] = learnlv1(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) і

[dW,LS] = learnlv2(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функції налаштування мереж зустрічного розповсюдження.

[dW,LS] = learnos(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функція налаштування нейрона типу «вихідна зірка».

[dW,LS] = learnp(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функція, що реалізовує алгоритм навчання персептрона.

[dW,LS] = learnpn(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – те ж, що і попередня функція, але з нормалізацією входів. Ефективніша при великих амплітудних змінах вхідних сигналів.

[dW,LS] = learnsom(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функція навчання карт, що самоорганізовуються.

[dW,LS] = learnwh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS) – функція навчання, що реалізовує так званий алгоритм Уідроу-Хоффа (Widrow-Hoff).

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведені у прикладах функції навчання шарів нейронів НМ.
2. Згідно завдання викладача реалізувати 3 фунції навчання шарів нейронів НМ.

1.3. Функції одновимірної оптимізації та ініціалізації шарів і зсувів

Функції цієї групи можна розглядати як допоміжні для функцій навчання нейронних мереж. Щодо оптимізації, то вони реалізують ряд алгоритмів одномірного пошуку.

srchbac – функція реалізує так званий алгоритм перебору з поверненнями (backtracking).

srchbre – функція реалізує комбінований метод оптимізації, який об'єднує метод золотого перетину і квадратичну інтерполяцію.

srchcha – функція реалізує різновид методу оптимізації із застосуванням кубічної інтерполяції.

srchgol – функція реалізує метод золотого перетину.

srchhyb – функція реалізує комбінований метод оптимізації, який об'єднує метод дихотомії і кубічну інтерполяцію.

Для багатомірних нейронних мереж етапом, що передує процедурі їх навчання, є етап ініціалізації – задання деяких, зазвичай вибраних випадковим чином, вагів і зсувів мережі. Така ініціалізація виконується за допомогою функцій відповідної групи команд.

initcon(S,PR) – функція, що встановлює зсуви нейронів залежно від середнього виходу нейрона. *Аргументи*: s – кількість нейронів, PR = [Pmin Pmax] – матриця (з двома стовпцями) мінімальних і максимальних значень входів, за умовчанням [1 1]. Повертає вектор зсувів. Використовується спільно із командою з командою *learncon*.

Приклад

```
» b = initcon(3)
b=
  8.1548
  8.1548
  8.1548
```

initzero – функція задання нульових початкових значень вагам або зсувам. Аргументи ті ж, що і у попередньої команди.

midpoint(S,PR) – функція ініціалізації, що встановлює ваги відповідно до середніх значень входів.

randnc(S,R) – функція задання матриці вагів. Повертає матрицю розміру S×R з випадковими елементами, нормалізовану за стовпцями (вектори-стовпці мають одиничну довжину).

randnr(S,R) – те ж, що попередня функція, але повертає матрицю вагів, нормалізовану за рядками.

rands – функція ініціалізації вагів/зсувів заданням їх випадкових значень з діапазону [-1, 1].

Запис:

```
W = rands(S,PR)
M = rands(S,R)
v = rands(S)
```

Опис. Аргументи ті ж, що і для функції **initcon**; значення R за умовчанням - 1. Повертається матриця відповідного розміру.

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведений приклад встановлення зсувів.
2. Згідно завдання викладача реалізувати 3 фунції оптимізації/ініціалізації шарів НМ.

1.4. Функції створення нейронних мереж

network – функція створення нейронної мережі користувача.

Запис:

net = network

net = network(numInputs, numLayers, biasConnect, inputConnect layerConnect, outputConnect, targetConnect)

Опис. Функція повертає створену нейронну мережу з ім'ям net і з наступними характеристиками (у дужках представлено значення за замовчанням):

numInputs – кількість входів (0);

numLayers – кількість шарів (0);

biasConnect – булевий вектор з числом елементів, рівним кількості шарів (нулі);

inputConnect – булева матриця з числом рядків, рівним кількості шарів, і числом стовпців, рівним кількості входів (нулі);

layerConnect – булева матриця з числом рядків і стовпців, рівним кількості шарів (нулі);

outputConnect – булевий вектор-рядок з числом елементів, рівним кількості шарів (нулі);

targetConnect – вектор-рядок, такий же, як попередня (нулі).

net = newc(PR,S,KLR,CLR) – функція створення шару Кохонена.

Функція використовує аргументи:

PR – Rx2 матрицю мінімальних і максимальних значень для R вхідних елементів;

S – число нейронів;

KLR – коефіцієнт навчання Кохонена (за умовчанням 0,01);

CLR – коефіцієнт «справедливості» (за умовчанням 0,001).

net = newcf(PR,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF,PF) – функція створення різновиду багатошарової НМ із зворотним розповсюдженням помилки (каскадна НМ). Така мережа містить прихованих шарів NI, використовує вхідні функції типу **dotprod** і **netsum**, ініціалізація мережі здійснюється функцією **initnw**.

Аргументи функції:

PR – Rx2 матриця мінімальних і максимальних значень R вхідних елементів;

Si – розмір і-го прихованого шару, для NI шарів;

TFi – функція активації нейронів і-го шару, за умовчанням '**tansig**';
BTF – функція навчання мережі, за умовчанням '**traingd**';
BLF – функція налаштування вагів і зсувів, за умовчанням '**learngdm**';
PF – функція помилки, за умовчанням '**mse**'.

Приклад

```
»P = [0 1 2 3 4 5 6 7 8 9 10];
»T = [0 1 2 3 4 3 2 1 2 3 4];
» net = newcf([0 10],[5 1],{'tansig' 'purelin'}); % Створення нової мережі
» net.trainParam.epochs = 50; % Задання кількості циклів навчання
» net = train(net,P,T); % Навчання НМ
TRAINLM, Epoch 0/50, MSE 7.77493/0, Gradient 138.282/1e-
010
TRAINLM, Epoch 25/50, MSE 4.01014e-010/0, Gradient
0.00028557/1e-010
TRAINLM, Epoch 50/50, MSE 1.13636e-011/0, Gradient 1.76513e-
006/1e-010
TRAINLM, Maximum epoch reached, performance goal was not met.
» Y = sim(net,P); % Використання НМ
» plot(P,T,'d',P,Y)% Графічна ілюстрація роботи мережі
```

На мал. 1.1 крапками відображено елементи навчальної вибірки, лінією вихід мережі.

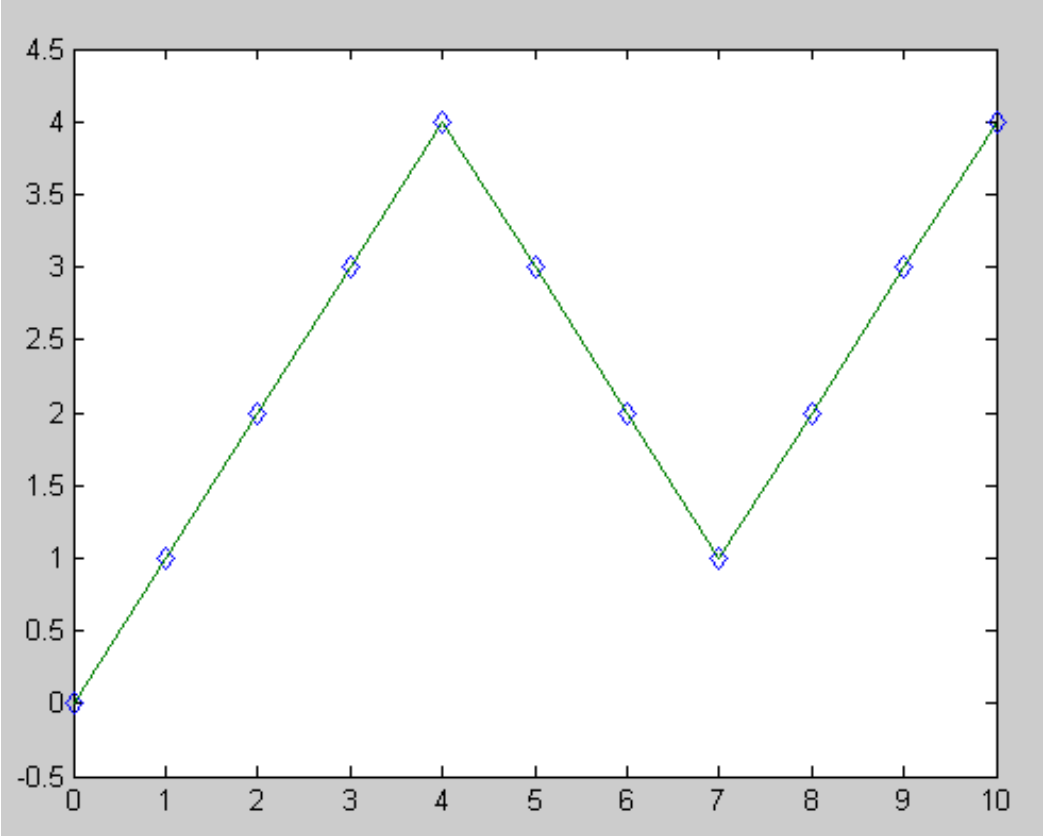


Рис. 1.1. Ілюстрація роботи мережі

net = newelm(PR,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF,PF) – функція створення мережі Елмана. Аргументи такі ж, як і у попередньої функції.

net = newff(PR,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF,PF) – функція створення «класичною» багатошарової НМ з навчанням за методом зворотного розповсюдження помилки.

net = newfftd(PR,ID,[S1 S2...SNI],TF1 TF2...TFNI,BTF,BLF, PF) – те ж, що і попередня функція, але з наявністю затримок за входами. Додатковий аргумент ID вектор вхідних затримок.

net = newgrnn(P,T,spread)- функція створення загальнооегресійної моделі мережі.

Аргументи:

P – R×Q матриця Q вхідних векторів;

T – S×Q матриця Q цільових векторів;

spread – відхилення (за замовчанням 1,0).

net = newhop(T) – функція створення мережі Хопфілда. Використовується тільки один аргумент.

T – R×Q матриця Q цільових векторів (значення елементів мають бути +1 або -1).

net = newlin(PR,S,ID,LR) – функція створення шару лінійних нейронів.

Аргументи:

PR – R×2 матриця мінімальних і максимальних значень для R вхідних елементів;

S – число елементів у вихідному векторі;

ID – вектор вхідної затримки (за замовчанням [0]);

LR – коефіцієнт навчання (за замовчанням 0,01).

Повертається новий лінійний шар. При записі у формі **net = newlin(PR,S,0,P)** використовується аргумент: **P –** матриця вхідних векторів; повертається лінійний шар з максимально можливим коефіцієнтом навчання при заданій P.

net = newlind(P,T) – функція проектування нового лінійного шару. Функція матриць вхідних і вихідних векторів, отриманих методом найменших квадратів, визначає ваги і зсуви лінійної НМ.

net = newlvq(PR,S1,PC,LR,LF) – функція створення мережі зустрічного розповсюдження.

Аргументи:

PR – Rx2 матриця мінімальних і максимальних значень R вхідних елементів;

S1 – число прихованих нейронів;

PC – S2 елементів вектора, які задаюють ступінь належності до різних класів;

LR – коефіцієнт навчання, за умовчанням 0,01;
LF – функція навчання, за умовчанням 'learnlv2'.

net = newp(PR,S,TF,LF) – функція створення персептрона.
Аргументи:
PR – Rx2 матриця мінімальних і максимальних значень R вхідних елементів;
S – число нейронів;
TF – функція активації, за умовчанням '**hardlim**';
LF – функція навчання, за умовчанням '**learnp**'.

net = newpnn(P,T,spread) – створення ймовірнісної НМ. Аргументи **–** як у функції **newgrnn**.

net = newrb(P,T,goal,spread) – функція створення мережі з радиальними базисними елементами. Аргументи: **P, T, spread –** такі ж, як у функції **newgrnn**; аргумент **goal –** задана средньоквадратична похибка.

net = newrbe(P,T,spread) – функція створення мережі з радиальними базисними елементами з нульовою помилкою на навчальній вибірці.

net = newsom(PR,[D1,D2...],TFCN,DFCN,OLR,OSTEPS,TLR,TND) – функція створення самонавчальної карти з аргументами:
PR – Rx2 матриця мінімальних і максимальних значень R вхідних елементів;
I – розмір і-го шару, за умовчанням [5 8];
TFCN – топологічна функція, за замовчанням '**hextop**';
DFCN – функція відстані, за замовчанням '**linkdist**';
OLR – коефіцієнт навчання фази впорядковування, за замовчанням 0,9;
OSTEPS – число кроків фази впорядковування, за замовчанням 1000;
TLR – коефіцієнт навчання фази налаштування, за замовчанням 0,02;
TND – відстань для фази налаштування, за замовчанням 1.

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведений приклад створення НМ.
2. Згідно завдання викладача реалізувати 3 фунції сторення НМ.

1.4. Функції перетворення входів мережі, вагів та відстаней

Функції даної групи перетворять значення входів з використанням операцій множення або додавання.

netprod(Z1,Z2...) – повертає матрицю, елементи якої визначаються як елементи вхідних векторів і зсувів. Аргументи **Z1,Z2... –** матриці, чиї стовпці асоційоровані з входами або зсувами.

Приклади

```
»z1 = [1 2 4;3 4 1];
»z2 = [-1 2 2;-5-6 1];
» n = netprod(z1,z2)
n =
  -14     8
 -15 -24    1
»b = [0;-1];
» n = netprod(z1,z2,concur(b,3))% Функція concur(b,3) створює 3 копії вектора зсуву
n=
      0     0     0
     15    24    -1
```

netsum(Z1,Z2...) – те ж, що у попередньому випадку, але замість множення використовується додавання.

dnetprod(Z,N) – повертає матрицю значень першої похідної входів, перетворених функцією **N = netprod(Z1,Z2...).**

Приклад

```
»Z1 = [0; 1; -1];
»Z2 = [1; 0.5; 1.2];
»N = netprod(Z1,Z2)
 N=
0
0.5000
-1.2000
» dN_dZ2 = dnetprod(Z2,N)
»  dN_dZ2 =
 0
 1
-1
```

dnetsum(Z,N) – те ж, що і у попередньому випадку, але по відношенню до функції **netsum(Z1,Z2...)**.

boxdist(pos) – функція визначення box-відстані між нейронами в шарі. Має один аргумент pos-матрицу розміру N×S, елементи якої визначають координати нейронів, повертає матрицю розміру S×S відстаней.

Відстані (елементи матриці, що повертається) обчислюються за виразом:

$$D_{ij} = max(abs(P_i — P_j))$$

де: P_i і P_j – вектори, що містять координати нейронів *i* і *j*.

Приклад

```
» pos = rand(3,4)% Випадкове розміщення 4-х нейронів в 3-мірному просторі (генерація випадкової матриці 3х4)
pos =
```

```
    0.8600    0.4966    0.6449    0.3420
    0.8537    0.8998    0.8180    0.2897
    0.5936    0.8216    0.6602    0.3412

» d=boxdist(pos)
d=
         0    0.3635    0.2151    0.5639
    0.3635         0    0.1614    0.6100
    0.2151    0.1614         0    0.5282
    0.5639    0.6100    0.5282         0
```

dist – функція обчислення евклідової відстані.

Z = dist(W,P) – повертає матрицю, елементи якої являются евклідовими відстанями між рядками (векторами) матриці **W** і стовпцями матриці **P** (матриці повинні мати відповідні розміри).

D = dist(pos) – у такій формі функція аналогічна функції **boxdist(pos)** за тим виключенням, що повертається матриця евклідових відстаней.

negdist(W,P) – функція ідентична попередній, але елементи матриці, що повертається, є евклідовими відстанями, взятими зі знаком мінус.

mandist(W,P) – функція аналогічна попередній, але елементи матриці, що повертається, є відстанями за Манхеттеном, які для векторів *x* і *y* визначаються співвідношенням: *D = sum(abs(x — y))*.

linkdist(pos) – функція визначення лінійної відстані між нейронами в шарі. Аналогічна функції *boxdist;* відрізняється від останньої алгоритмом визначення відстані:

D ij = 0, якщо i = j;

D ij = 1, якщо евклідова відстань між і і P j менше або рівна 1;

Dij = 2, якщо існує D таке, що D ik = D kj = 1;

D ij = 3, якщо існують k1 і k2 такі, що -Oik1 = D k1k2=... = D k2j =1;

D ij = N, якщо існують k1, k2 :::, kN такі, що -Oik1 = D k1k2 =...= D kNj =1;

D ij = S, якщо не виконана жодна з попередніх умов.

dotprod(W,P) – функція додавання входам **P** деяких вагів **W.** Повертає матрицю **Z=W*P**.

normprod(W,P) – функція аналогічна попередній, але кожний елемент матриці, що повертається, додатково ділиться на суму елементів відповідного стовпця-співмножника матриці **P**.

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведені приклади щодо перетворення входів мережі та її вагів.
2. Згідно завдання викладача реалізувати 5 фунцій перетворення входів, вагів та відстаней НМ.

1.5. Функції розміщення нейронів (топологічні функції) та використання НМ

Функції даної групи використовуються при створенні карт самоорганізації.

gridtop(dim1,dim2...,dimN) – функція розміщення N шарів нейронів у вузлах регулярних прямокутних N-вимірних решіток. **dim1,dim2...,dimN** – число нейронів у шарах. Повертає матрицю із N рядків і **(dim1xdim2x.. .xdimN)** стовпців з координатами нейронів.

Приклад

```
» pos = gridtop(2,3)
 pos =
   0   1   0   1   0   1
   0   0   1   1   2   2
```

hextop(dim1,dim2...,dimN) – функція аналогічна попередній, але розміщення нейронів проводиться у вузлах гексагональних (шестикутних) решіток.

Приклад

» pos = hextop(8,5); plotsom(pos) (рис. 1.2)

randtop(dim1,dim2...,dimN) – аналогічна функції **gridtop(dim1,dim2...,dimN),** але координати нейронів вибираються випадковим чином.

Приклад

pos = randtop(16,12); plotsom(pos) (рис. 1.3)

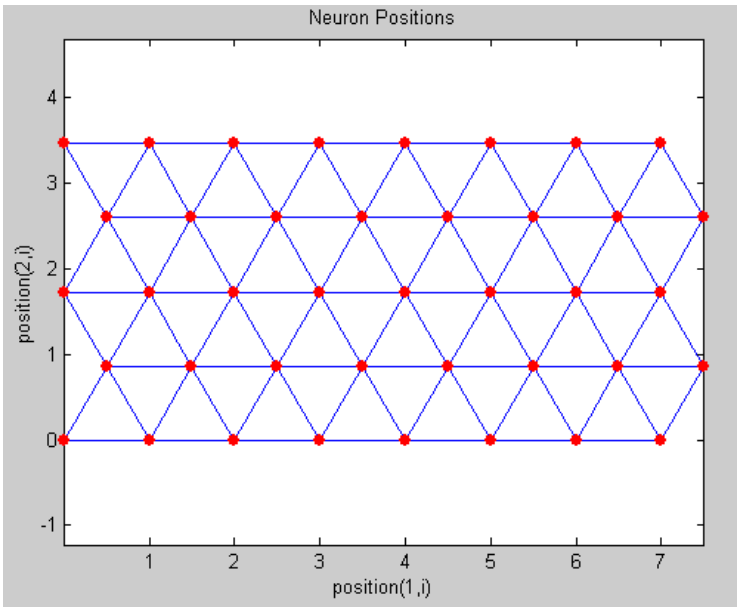


Рис. 1.2. Результат виконання команди hextop(8,5)

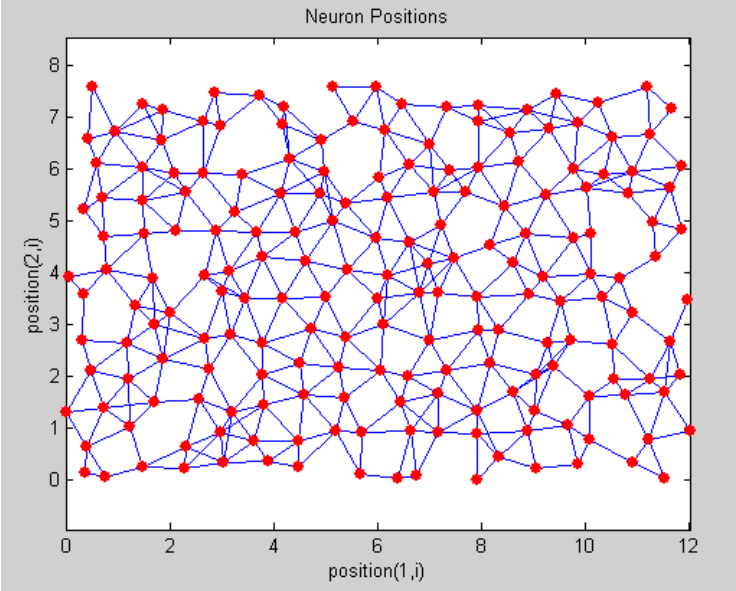


Рис. 1.3. Результат виконання команди randtop(16,12)

[Y,Pf,Af] = sim(net,P,Pi,Ai) – функція, що моделює роботу нейронної мережі.

Аргументи:

net – ім'я мережі, **P** – її входи, **Pi** – масив початкових умов вхідних затримок (за умовчанням вони нульові), **Ai** – масив початкових умов затримок (за умовчанням вони нульові).

Функція повертає значення виходів **Y** і масиви кінцевих умов затримок.

Аргументи **Pi, Ai, Pf, Af** використовуються тільки у випадках, коли мережа має затримки за входами або за шарами нейронів. Структура даних аргументів: **P** – масив розміру **NI**x**TS**, кожен елемент якого **P{i,ts}** є матрицею розміру **RI**×**Q**.

Pi – масив розміру **NI**x**ID**, кожен елемент якого **Pi{i,k}** (і-й вхід у момент ts= k — LD) є матрицею розміру **RI**x**Q** (за умовчанням – нуль).

Ai – масив розміру **NL**x**LD**, кожен елемент якого **Ai{i,k}** (вихід і-го шару у момент ts = k - LD) є матрицею розміру **SI**x**Q** (за умовчанням - нуль).

Y – масив розміру **NO**x**TS**, кожен елемент якого **Y{i,ts}** є матрицею розміру **UI**x**Q**.

Pf – масив розміру **NI**x**ID**, кожен елемент якого **Pf{i,k}** (і-й вхід у момент ts= TS + k - LD) є матрицею розміру **RI**x**Q**.

Af – масив розміру **NI**x**LD**, кожен елемент якого **Af{i,k}** (вихід і-го шару у момент ts = TS + k - LD) є матрицею розміру **SIXQ**, при цьому:

Ni = net.numInputs – кількість входів мережі;

Nl = net.numLayers – кількість її шарів;

No = net.numOutputs – кількість виходів мережі;

ID = net.numInputDelays – вхідні затримки;

LD = net.numLayerDelays – затримки шару;

TS = Number of time steps – число тимчасових інтервалів;

Q = Batch size – розмір набору векторів, що подаються;

Ri= net.inputs{ i }.size – розмір і-го вектора входу;
Si = net.layers{ i }.size – розмір і-го шару;
Ui = net.outputs{ i }.size – розмір і-го вектора виходу.

net = init(net) – функція ініціалізує нейронну мережу з ім'ям **net**, встановлюючи ваги і зсуви мережі, відповідно до установок **net.initFcn** і **net.initParam**.

[net,Y,E,Pf,Af] = adapt(net,P,T,Pi,Ai) – функція адаптації НМ. Виконує адаптацію мережі відповідно до установок **net.adaptFcn** і **net.adaptParam**.

Де: **E** – помилки мережі, **T** – цільові значення виходів (за умовчанням - нуль); решта аргументів – як у команди **sim**.

[net,tr] = train(net,P,T,Pi,Ai) – функція здійснює навчання НМ відповідно до установок **net.trainFcn** і **net.trainParam.**

Де**: tr** – інформація про виконання процесу навчання (кількість циклів і відповідна помилка навчання).

disp(net) – функція повертає розгорнену інформацію про структуру і властивості НМ.

Приклад

```
» net = newp([-1 1; 0 2],3); % Створення НМ типу персептрон
» disp(net) Neural Network object: architecture:
    numInputs: 1
    numLayers: 1
    biasConnect: [1]
    inputConnect: [1]
    layerConnect: [0]
    outputConnect: [1]
    targetConnect: [1]
    numOutputs: 1 (read-only)
    numTargets: 1 (read-only)
    numInputDelays: 0 (read-only)
    numLayerDelays: 0 (read-only) subobject structures:
    inputs: {1x1 cell} of inputs
    layers: {1x1 cell} of layers
    outputs: {1x1 cell} containing 1 output
    targets: {1x1 cell} containing 1 target
    biases: {1x1 cell} containing 1 bias
    inputWeights: {1x1 cell} containing 1 input weight
    layerWeights: {1x1 cell} containing no layer weights
  functions:
    adaptFcn: 'adaptwb' initFcn: 'initlay'
performFcn: 'mae' trainFcn: 'trainwb'
parameters: adaptParam: .passes initParam:
(none) performParam: (none)
```

trainParam: .epochs, .goal, .max_fail, .show, .time
weight and bias values:
IW: {1x1 cell} containing 1 input weight matrix LW: {1x1 cell} containing no layer weight matrices b: {1x1 cell} containing 1 bias vector
other:
userdata: (user stuff)

display(net) – те ж, що попередня команда, але додатково повертає ім'я нейронної мережі.

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведені приклади щодо розміщення нейронів та використання НМ.
2. Згідно завдання викладача реалізувати 5 функцій розміщення нейронів та використання НМ.

1.6. Графічні функції

hintonw(W,maxw,minw) – функція повертає хінтоновський графік матриці вагів, при якому кожен ваговий коефіцієнт відображається квадратом з площею, пропорційній величині даного коефіцієнта. Знак відображається кольором квадрата.

Аргументи:

W – матриця вагів; **maxw** і **minw** – мінімальне і максимальне значення її коефіцієнтів (можуть не задаватися).

Приклад

»W = rands(2,3)
W=
-0.7495 0.8228 0.2340
-0.6677 -0.7275 -0.4620
» hintonw(W) (рис. 1.4).

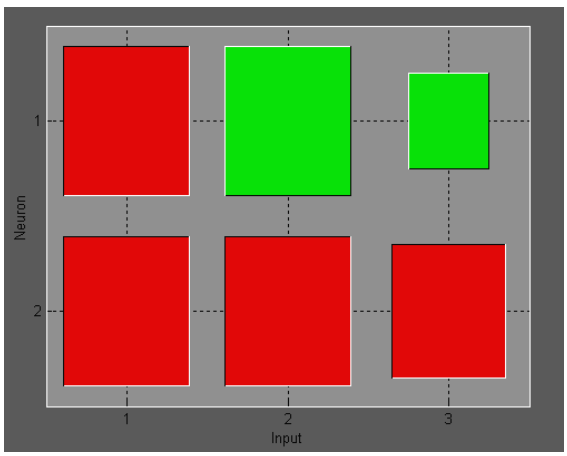


Рис. 1.4. Ілюстрація до виконання функції **hintonw**

hintonwbM(W,b,maxw,minw) – те ж, що і попередня функція, але на графіці відображаються не тільки ваги, але і зсув.

plotbr(tr,name,epoch) – функція повертає графіки зміни критерію якості НМ в процесі навчання при використанні Баєсівського методу.

Аргументи:

tr – запис процесу навчання, **name** – ім'я НМ, **epoch** – кількість циклів навчання (за умовчанням – довжина запису навчання).

Приклад

```
»p = [-1:.05:1];
»t = sin(2*pi*p) +0.1*randn(size(p));
»net = newff([-1 1],[20 1], {''tansig','purelin'},'trainbr'); %Створення нової мережі
»[net,tr] = train(net,p,t); % Навчання мережі
TRAINBR, Epoch 0/100, SSE 228.933/0, SSW 21461.7, Grad 2.33e+002/1.00e-010, #Par 6.10e+001/61
TRAINBR, Epoch 25/100, SSE 0.235423/0, SSW 211.044, Grad 9.43e-002/1.00e-010, #Par 1.35e+001/61
TRAINBR, Epoch 50/100, SSE 0.240881/0, SSW 121.647, Grad 1.87e-001/1.00e-010, #Par 1.23e+001/61
TRAINBR, Epoch 75/100, SSE 0.239867/0, SSW 116.884, Grad 1.62e-002/1.00e-010, #Par 1.22e+001/61
TRAINBR, Epoch 100/100, SSE 0.239762/0, SSW 116.871, Grad 9.60e-003/1.00e-010, #Par 1.22e+001/61
TRAINBR, Maximum epoch reached.
» plotbr(tr)
```

(рис 1.5).

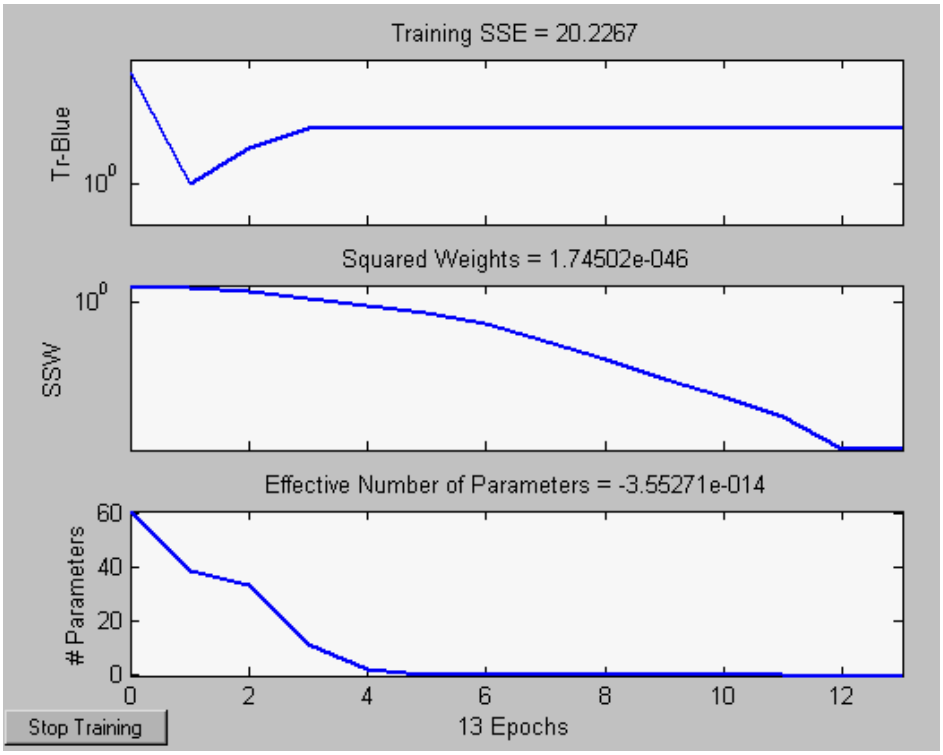


Рис. 1.5. Результат виконання функції **plotbr(tr)**

plotep(w,b,e) – функція відображає позиції вагів і зміщень на поверхні помилки НМ.

Аргументи:

w, b, e – відповідно, матриці вагів, зсувів і помилок. Повертається вектор, використаний для реалізації графіку, створеного функцією **plotes**.

plotes(wv,bv,e,v) – функція повертає графік поверхні помилки нейрону із одним входом.

Аргументи:

wv, bv – відповідно, набори значень ваги і зсуву нейрона, **e** – матриця значень помилки, **v** – опція виду зображения (за умовчанням [-37,5, 30]).

plotpc(W,b) – функція повертає графік лінії розв'зку для персептрона.

Аргументи:

W – матриця вагів, **b** – вектор зсувів. Використовується спільно з функцією **plotpv**.

plotperf(tr,goal,name,epoch) – повертає графік зміни критерію якості НМ в процесі навчання.

Аргументи: **tr** – запис процесу навчання; **goal** – цільове значення критерію; **name** – ім'я НМ; **epoch** – кількість циклів навчання.

plotpv(p,t) – функція повертає графічне відображення вхідних **p** і цільових **t** векторів персептрона.

Приклад

```
»p = [0 0 1 1; 0 1 0 1];
»t=[0 0 0 1];
» plotpv(p,t)
```
(рис. 1.6).

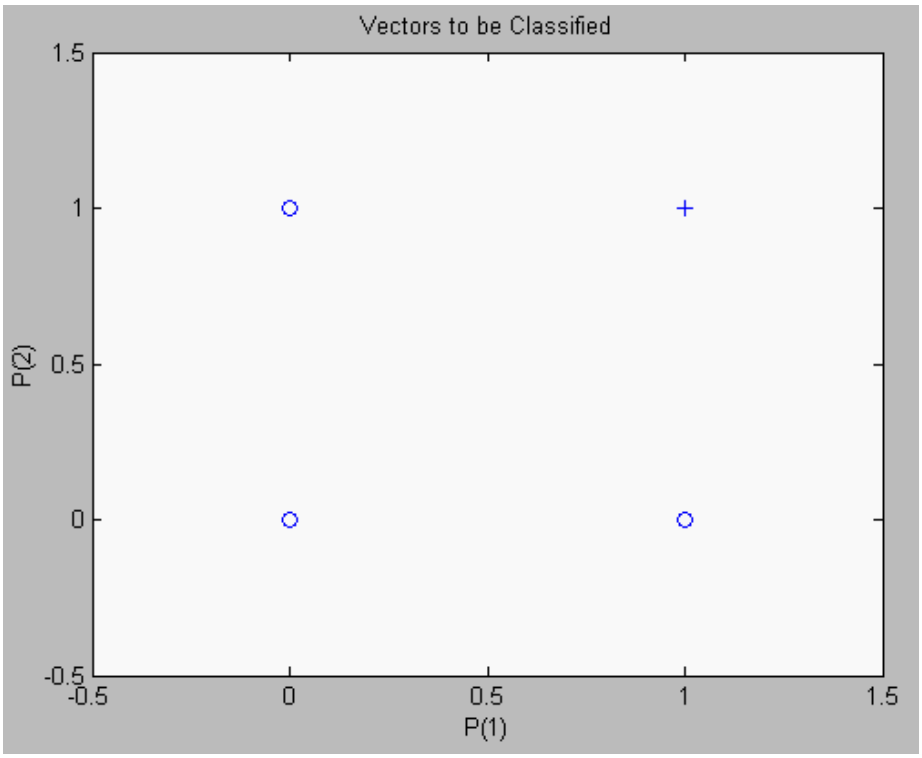


Рис1.6. Результат використання функції **plotpv(p,t)**

plotsom(pos) – функція повертає графічне представлення розташування нейронів у картах, що самоорганизующихся (див. функції розміщення нейронів).

plotv(M,t) – функція графічного зображення векторів.

Аргументи: **M** – матриця з двома рядками, стовпці якої асоційовані із векторами, **t** – опція, що задає тип лінії.

Приклад

»plotv([-.4 0.7 .2; -0.5 .1 0.5],'-') (рис. 1.7).

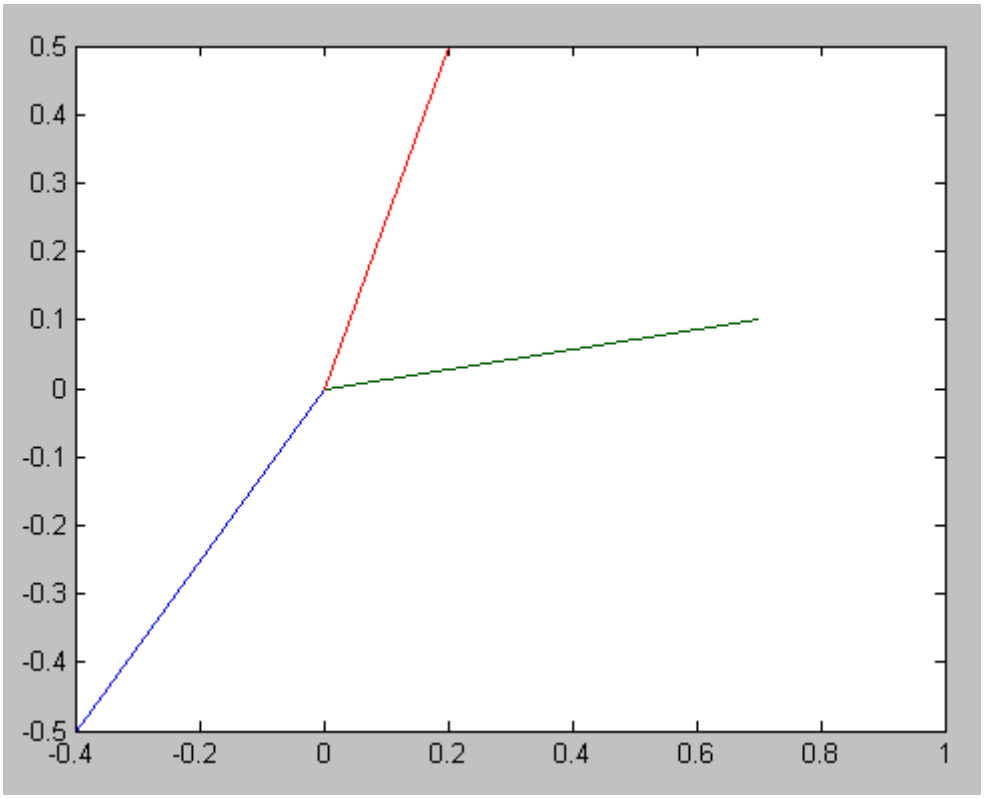


Рис. 1.7. Результат використання функції **plotv(M,t)**

plotvec(M,C,m) – функція графічного зображення векторів різними кольорами.

Аргументи:

M – матриця з двома рядками, стовпці якої асоціюються з векторами; **C** – рядок задання кольорів; **m** – тип точок, які вказують на кінці векторів (за умовчанням +).

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведені приклади графічних функцій.
2. Згідно завдання викладача реалізувати 4 графічні фунції.

1.7. Інші функції

errsurf(p,t,wv,bv,f) – функція, що повертає матрицю значень поверхні помилок нейрона з одним входом і одним виходом залежно від значень ваги і зсуву.

Аргументи:

p – вектор значень входу;

t – вектор значень виходу;

wv – набір значень вагів нейрона;

bv – набір значень зсуву;

f – назва функції активації, що реалізовується (рядок).

Розмір матриці, що повертається = (кількість значень bv) * (кількість значень wv).

Приклад

```
» p = [-6.0 -6.1 -4.1 -4.0 +4.0 +4.1 +6.0 +6.1];
» t = [+0.0 +0.0 +.97 +.99 +.01 +.03 +1.0 +1.0];
»wv = -1:.1:1;
bv = -2.5:.25:2.5;
» es = errsurf(p,t,wv,bv,'logsig');
» plotes(wv,bv,es[60 30])
```
(рис 1.8)

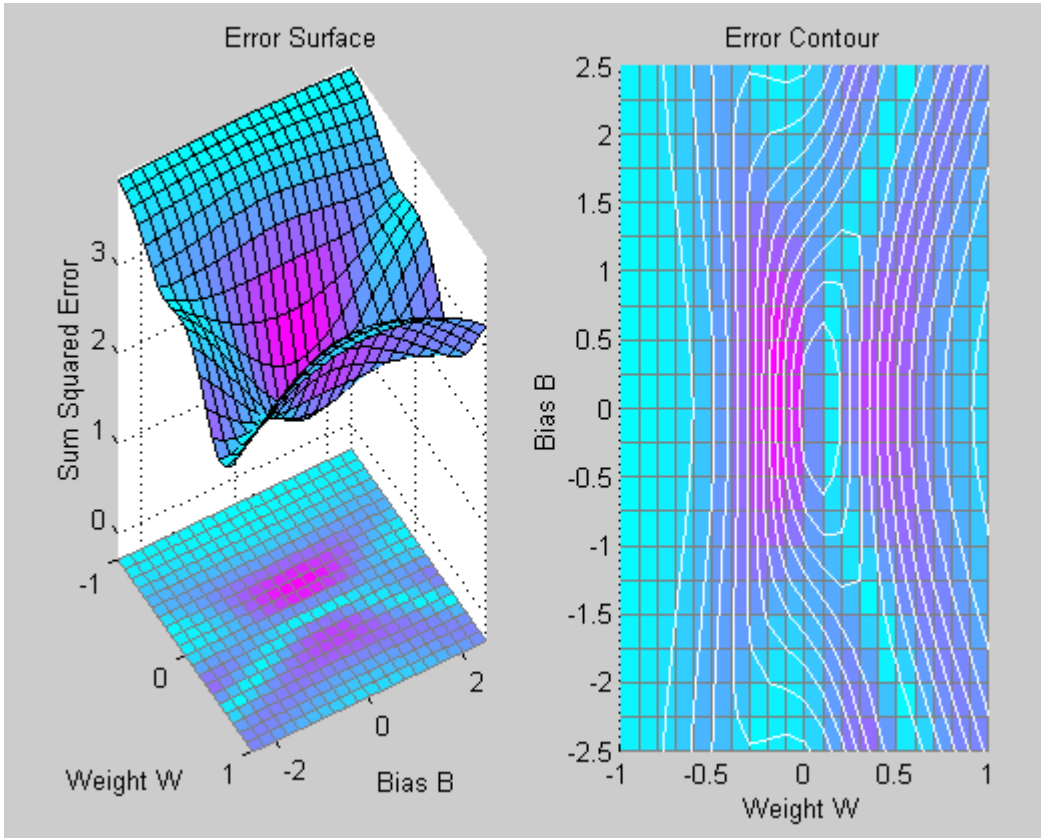


Рис. 1.8. Ілюстрація виконання функції **errsurf**

maxlinlr(P) – повертає максимальну величину коефіцієнта навчання для лінійного шару нейронів. Тут **P** – матриця входів**.**

При записі у формі **maxlinlr(P,'bias')** функція повертає максимальну величину коефіцієнта навчання для лінійного шару нейронів із зсувом.

gensim(net,st) – функція генерує нейромережевий блок **Simulink** (рис. 1.9) для подальшого моделювання НМ засобами цього пакету.

Приклад

```
» net = newff([0 1],[5 1]); % Створення нової НМ
» gensim(net)
```

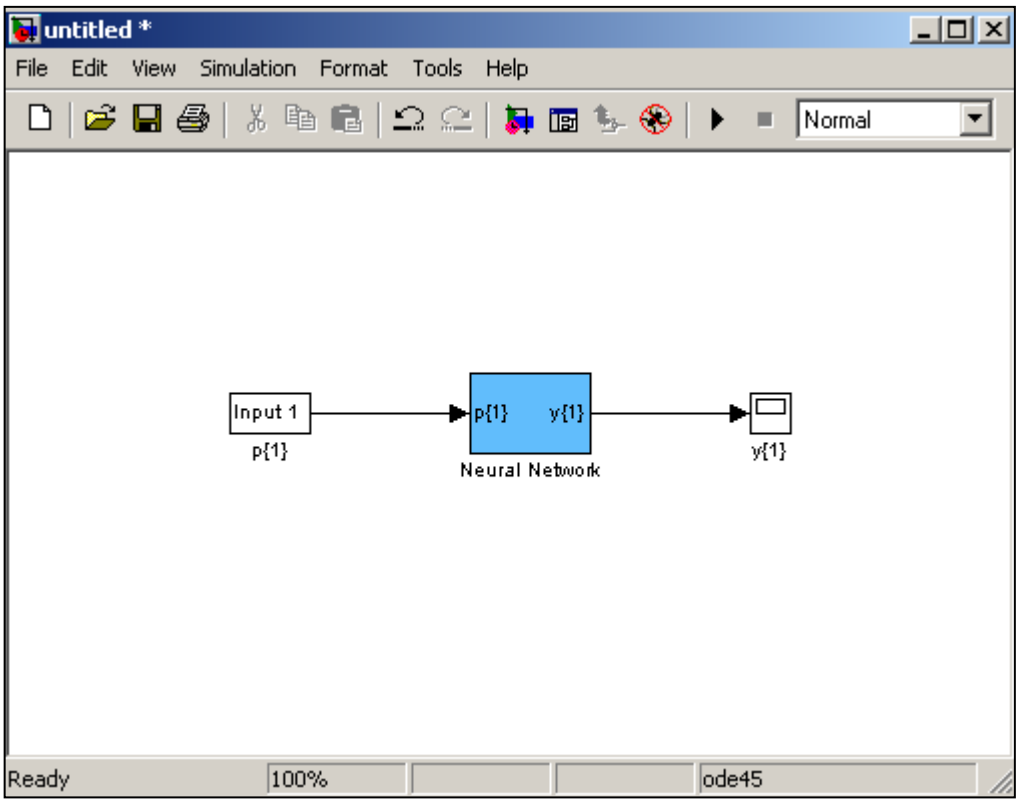


Рис. 1.9. Результат виконання функції **gensim**

initlay(net) – функція ініціалізації шарів нейронної мережі. Як аргумент використовує ім'я (ідентифікатор) **net** НМ. Повертає нейронну мережу, шари нейронів в якій ініціалізовані відповідно до функції **net.layers{i j.initFcn}**.

У формі **initlay(code)**, де змінна **code** може приймати значення **'pnames'** або **'pdefaults'**, функція повертає інформацію про імена або про значення за замовчанням параметрів ініціалізації.

initnw(net,i) – функція ініціалізації шару *p*.

initwb(net,i) – майже те ж, що у попередньому випадку, але ваги і зсув і-го шару ініціалізуються відповідно до їх власних функцій ініціалізації.

ddotprod – функція визначення похідної від результату **Z** множення матриці вагів **W** на матрицю входів **P**.

Запис:

```
dZ_dP = ddotprod('p',W,P,Z)
dZ_dW = ddotprod('w',W,P,Z)
```

П р и к л а д и

```
»W = [0 -1 0.2;-1.1 1 0];
»P = [0.1; 0.6;-0.2];
» Z = dotprod(W,P)% Обчислення Z=W*P
Z= -0.6400
    0.4900
» dZ_dP = ddotprod('p',W,P,Z)
dZ_dP =
0          -1.0000    0.2000
-1.1000    1.0000     0

» dZ_dW = ddotprod(''w',W,P,Z)
dZ_dW = 0.1000
            0.6000
           -0.2000
```

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведені функції.
2. Згідно завдання викладача реалізувати 2 функції.

2. Створення і використання нейронних мереж за допомогою командного рядка

2.1. Нейронні мережі для апроксимації функцій

Створим узагальнено-регресійну НМ з ім'ям *a* для апроксимації функції вигляду $y = x^2$ на відрізку [-1, 1], використовуючи наступні експериментальні дані:

x = [-1 - 0,8 - 0,5 -0,2 0 0,1 0,3 0,6 0,9 1],

y = [1 0,64 0,25 0,04 0 0,01 0,09 0,36 0,81 1].

Процедура створення і використання даної НМ описується таким чином:

```
» P = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1]; % Задання вхідних значень
» T = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1]; % Задання вихідних значень
» a = newgrnn(P,T,0.01); % Створення НМ з відхиленням 0.01
» Y = sim(a,[-0.9-0.7-0.3 0.4 0.8])% Опитування НМ
Y = 0.8200    0.6400    0.0400    0.0900    0.8100
```

Як видно, точність апроксимації в даному випадку отримали не дуже високою.

Можна спробувати поліпшити якість апроксимації за рахунок підбору величини відхилення, але в умовах прикладу необхідний результат легко досягається шляхом застосування мережі з радиальнобазисними елементами:

```
» a = newrbe(P,T);
»Y = sim(a,[-0.9-0.7-0.3 0.4 0.8]) % Опитування НМ
 Y= 0.8100    0.4900    0.0900    0.1600    0.6400
```

Створену мережу можна зберегти для подальшого використання набором в командному рядку **save('a');** при цьому буде створений файл **a.mat**, тобто файл з ім'ям НМ і розширенням **mat**. У наступних сеансах роботи збережену мережу можна завантажити, використовуючи функцію **load('a').** Природно, допустимі всі інші форми запису операторів **save** і **load.**

Розглянемо тепер аналогічне завдання, але з використанням лінійної НМ.

Хай експериментальна інформація задана значеннями:

x = [+1,0 +1,5 +3,0 -1,2],

y = [+0,5 +1,1 +3,0 -1,0].

Процес створення, навчання і використання лінійної НМ з ім'ям *b* ілюструється приведеними функціями (рис. 2.1).

```
» P = [+1.0 +1.5 +3.0-1.2];
» T = [+0.5 +1.1 +3.0-1.0];
» maxlr = maxlinlr(P,'bias'); % Визначення величини коефіцієнта навчання
» b = newlin([-2 2],1,[0],maxlr); % Створення лінійної НМ з ім'ям b
» b.trainParam.epochs = 15; % Задання кількості циклів навчання
»b = train(b,P,T); % Навчання НМ
TRAINWB, Epoch 0/15, MSE 2.865/0.
TRAINWB, Epoch 15/15, MSE 0.0730734/0.144
TRAINWB, Maximum epoch reached.
» p = -1.2;
» y = sim(b,p)% Опитування мережі
y= -0.8161
```

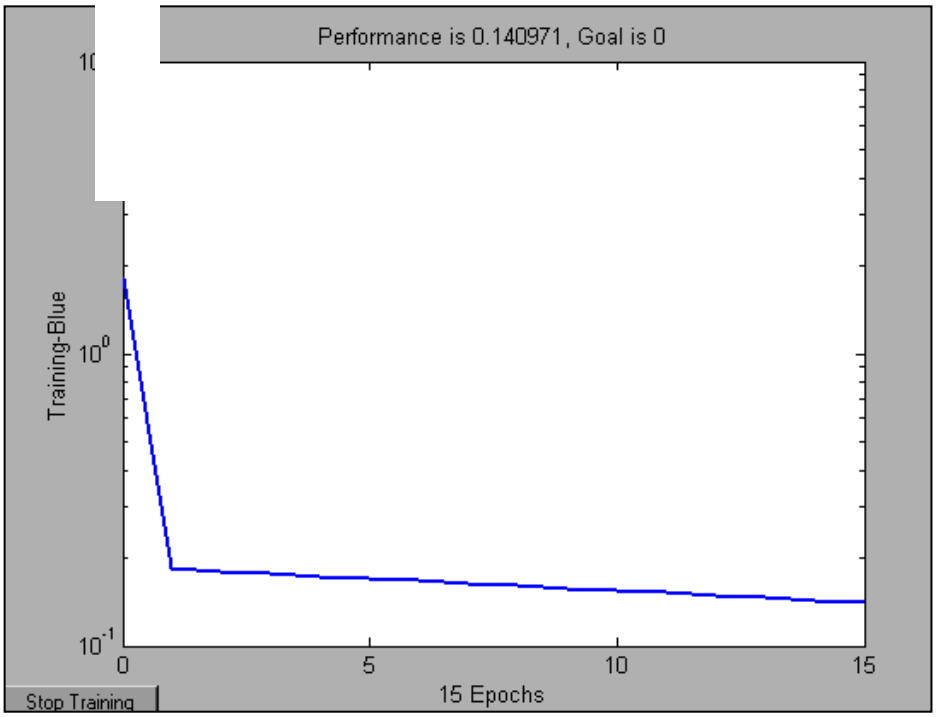


Рис. 2.1. Зміна помилки мережі в процесі її навчання

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведений приклади нейромережевої апроксимації.
2. Згідно завдання викладача провести нейромережеву апроксимацію та нарисувати блок-схему роботи НМ з розшифровкою функцій та їх аргументів.

2.2. Прогнозування значень процесу

Розглянемо тепер такий приклад. Припустимо, що є сигнал (функція часу), що описується співвідношенням **x(t) = sin (47 π t)**, який піддається дискретизації з інтервалом 0,025с.

Побудуємо лінійну нейронну мережу, яка дозволяє прогнозувати майбутнє значення подібного сигналу за 5 попередніми.

```
» t = 0:0.025:5; % Задання діапазону часу від 0 до 5 секунд
» x = sin(t*4*pi); % Сигнал, що передбачається
» Q = length (x); % Створення вхідних векторів

» P = zeros(5,Q); % Створення нульової матриці P
» P(1,2:Q)=x(1,1:(Q-1));
» P(2,3:Q)=x(1,1:(Q-2));
» P(3,4:Q)=x(1,1:(Q-3));
» P(4,5:Q)=x(1,1:(Q-4));
» P(5,6:Q)=x(1,1:(Q-5));
» s = newlind(P,x); % Створення нової НМ з ім'ям s
» y = sim(s,P); % Розрахунок прогнозованих значень

» % Створення графіків початкового сигналу і прогнозу

» plot(t,y,t,x,'+');
» xlabel('Time');
» ylabel('Predict - Signal ‘+’);
» title ( 'Output neural network ');

» % Розрахунок і створення графіка помилки прогнозу

» e = x-y;
» plot(t,e)
» hold on
»plot([min(t) max(t)],[0 0],':r');
» hold off
» xlabel ('Time');
» ylabel('Mistake');
» title('Mistakes signal');
```

У такому випадку мережа створювалася за допомогою функції **newlind,** при якій не вимагається додаткового навчання. Судячи із графічних результатів (рис. 2.2, 2.3) точність прогнозу з використанням лінійної НМ можна вважати достатньою.

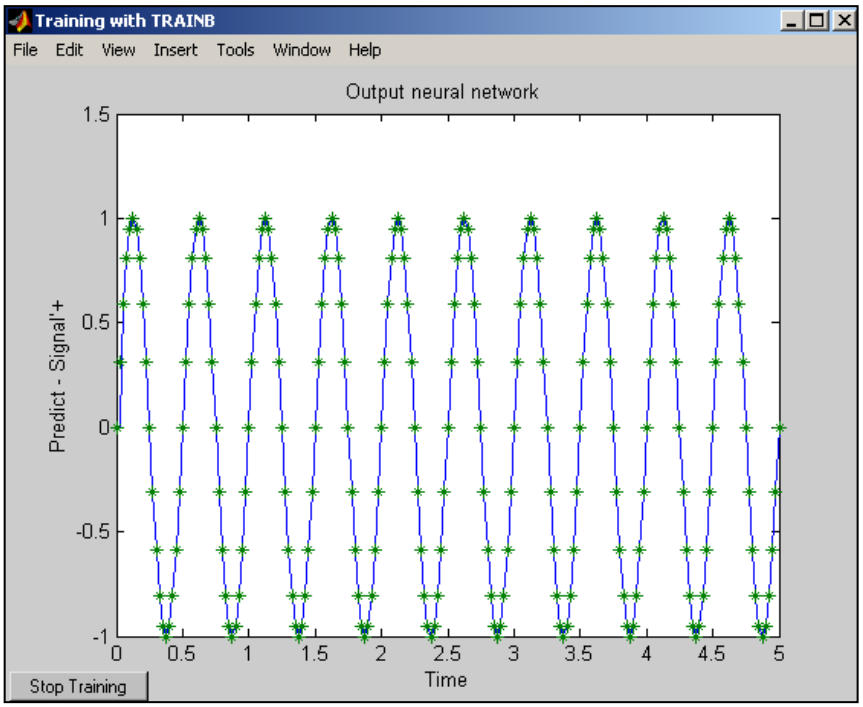


Рис. 2.2. Початковий сигнал і прогноз

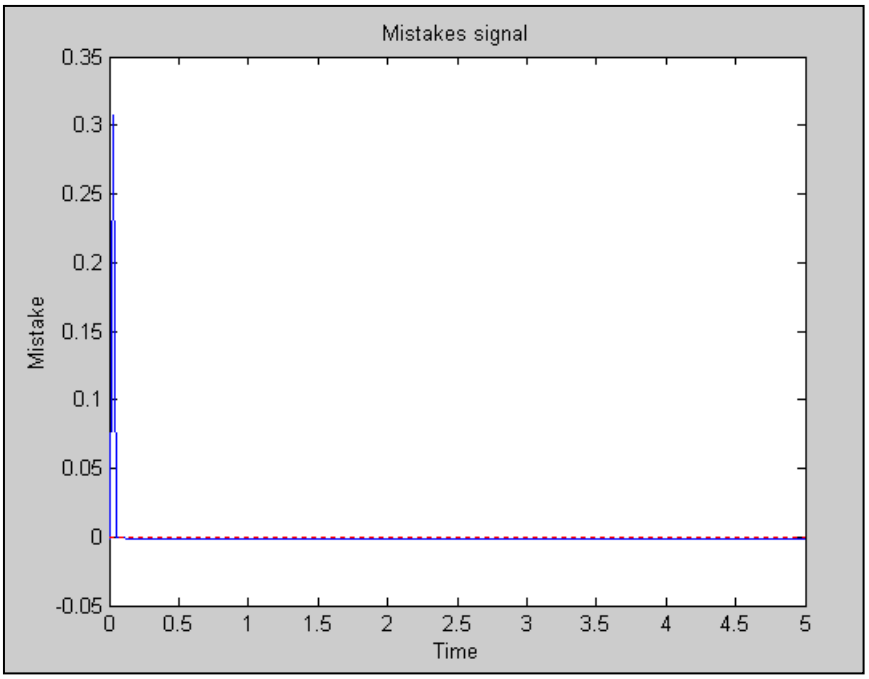


Рис.2.3. Помилка прогнозу

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведений приклади нейромережевого прогнозування.
2. Згідно завдання викладача провести нейромережеве прогнозування та нарисувати блок-схему роботи НМ з розшифровкою функцій та їх аргументів.

2.3. Використання шару Кохонена

Розглянемо завдання автоматичного виявлення (у режимі навчання без вчителя) центрів кластерів входів для двовимірного випадку з використанням шару Кохонена (шару нейронів, що «змагаються»). Вирішення такої задачі приведене нижче (рис. 2.4).

```
» X = [0 1; 0 1]; % Задання діапазонів можливого положення центрів кластерів
» % Задання параметрів для моделювання початкових даних
» % що належать 8 класам (кластерам)

» clusters = 8;

» points = 10;

» std_dev = 0.05;

» P = nngenc(X,clusters,points,std_dev); % Моделювання вхідних даних
» h = newc([0 1;0 1],8,0.1); % Створення шару Кохонена
» h.trainParam.epochs = 500; % Задання кількості циклів навчання

» h = init(h); % Ініціалізація мережі
» h = train(h,P); % Навчання мережі
» w = h.IW{1};

» %Виведення графіка початкових даних і виявлених центрів кластерів

» plot(P(1,:),P(2,:),'+r');
» hold on; plot(w(:,1),w(:,2),'ob');
» xlabel('p(1)');
» ylabel('p(2)');
» p = [0; 0.2]; % Задання нового вхідного вектора

» y = sim(h,p)% Опитування мережі

» y= (8,1)     1
```

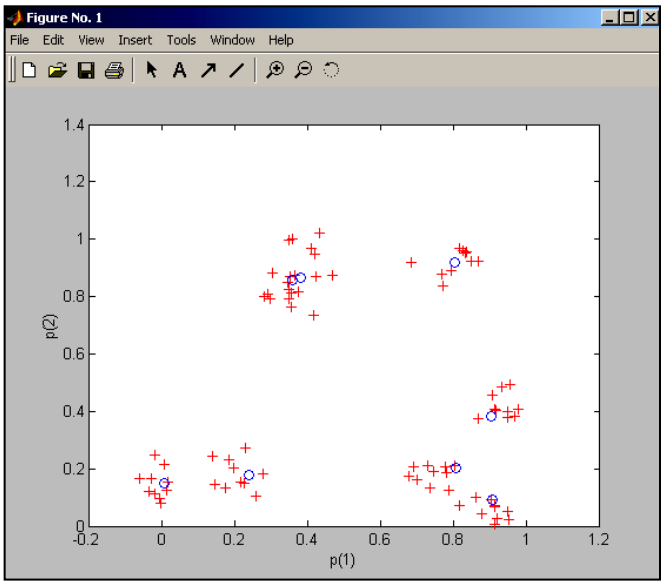


Рис. 2.4. Початкові дані і виявлені центри кластерів

Роботу і результат опитування НМ (видаються у формі розрідженої матриці) ілюструє рисунок 2.4 . В умовах прикладу пред'явлений вектор віднесений до восьмого класу (кластеру).

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведений приклади кластеризації із використанням шару Кохонена.
2. Згідно завдання викладача провести кластеризацію із використанням шару Кохонена та нарисувати блок-схему роботи НМ з розшифровкою функцій та їх аргументів.

2.4. Мережа Хопфілда з двома нейронами

Розглянемо мережу Хопфілда, що має два нейрони і два стійкі стани, які відображаються векторами [1 -1] і [-1 1]. Представимо ці вектори за допомогою рисунку 2.5, що виводиться програмою:

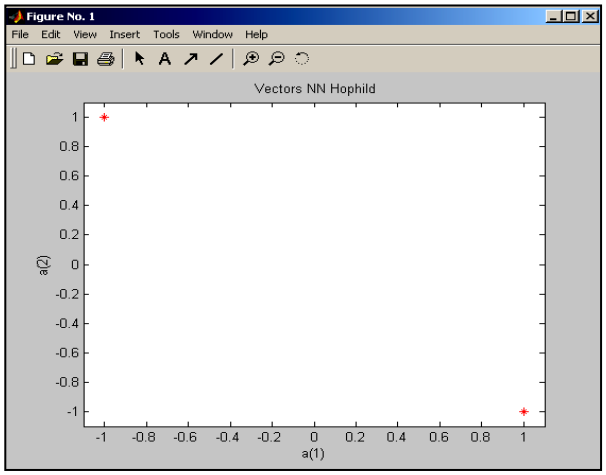


Рис. 2.5. Стійкі точки мережі Хопфілда

```
» T = [+1 -1;-1 +1];
» plot(T(1,:),T(2,:),'r +')
» axis([-1.1 1.1 -1.1 1.1]);
» title(‘ Vectors NN Hophild’);
» xlabel('a(1)'); » ylabel('a(2)');
```

Створимо мережу Хопфілда (з ім'ям H) і перевіримо її роботу, подавши на вхід вектори, відповідні стійким точкам. Якщо мережа працює правильно, вона повинна виробити ці ж вектори без яких-небудь змін.

```
» H = newhop(T); % Створення НМ Хопфілда
» [Y,Pf,Af] = sim(H,2,[ ],T);Y % Опитування мережі Хопфілда
  Y = 1     -1
     -1     1
```

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведений приклади налаштування мережі Хопфілда.
2. Згідно завдання викладача провести налаштування мережі Хопфілдата та нарисувати блок-схему роботи НМ з розшифровкою функцій та їх аргументів.

2.5. Класифікація за допомогою персептрона

Наступний приклад ілюструє рішення задачі класифікації за допомогою персептрона (рис. 2.6). Початкові вхідні вектори (при їх належності до одного з двох класів) і результат налаштування персептрона (з ім'ям My_net) представлені на рисунку.

```
» % Задання вхідних векторів з вказівкою їх належності
» % одному з двох класів
» P = [-0.5 -0.5 +0.3 -0.1;-0.5 +0.5 -0.5 +1.0];
» T=[1 1 0 0];
» plotpv(P,T); % Графічне представлення початкових векторів

» % Створення персептрона з вказівкою меж змін входів і 1 нейроном
» My_net=newp([-1 1;-1 1],1);
» E=1;
» My_net =init(My_net); % Ініціалізація персептрона

» % Організація циклу адаптивного налаштування персептрона
» % з виведенням графіка розділяючої лінії
» while (sse(E))
  [My_net,Y,E] = adapt(My_net,P,T);
  linehandle = plotpc(My_net.IW{1},My_net.b{1});
  drawnow;
  end;
```

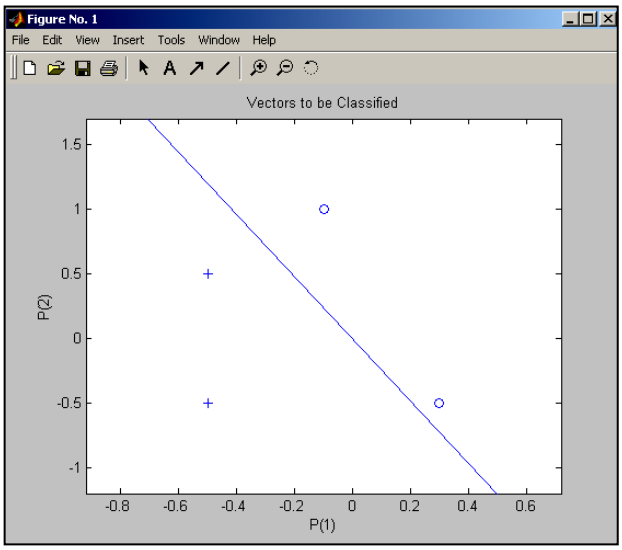


Рис. 2.6. Початкові вхідні вектори і розділяюча лінія

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведений прикладу використання персептрона.
2. Згідно завдання викладача провести налаштування персептрона та нарисувати блок-схему роботи НМ з розшифровкою функцій та їх аргументів.

2.6. Адаптивний лінійний прогноз

У попередньому прикладі налаштування НМ проводилось адаптивно. Відмінність такої настройки від виконаної, наприклад, за допомогою методу зворотного розповсюдження помилки, полягає в тому, що вектори навчальної вибірки поступають на вхід мережі не всі «одночасно», а послідовно, поодинці, при цьому після пред'явлення чергового вектора проводиться коректування вагів і зсувів і може бути проведене опитування мережі, потім все повторюється. Адапація налаштування особливо зручна при роботі НМ у «реальному» режимі часу.

Розглянемо приклад задання з прогнозуванням значень сигналу (за 5-ма попередніми значеннями) з використанням вказаного налаштування.

Припустимо, що початковий сигнал визначений на інтервалі часу від 0 до 6 с., при 0 < t < 4 с. він описується співвідношенням $x(t)= sin(47\ \pi t)$, а при 4 < t < 6 с. співвідношенням $x(t)= sin(8\ \pi\ t)$. Графік такого сигналу приведений на рисунку 2.7:

```
» time1 = 0:0.05:4; % від 0 до 4 секунд
» time2 = 4.05:0.024:6; % від 4 до 6 секунд
» time= [time1 time2];

» % T визначає початковий сигнал:
» T = con2seq([sin(time1*4*pi) sin(time2*8*pi)]);
```

» % Графік початкового сигналу:
» plot(time,cat(2,T{:}))
» xlabel('Time');
» ylabel (' Start signal ');
» title('Predict signal ');

Для прогнозу значень сигналу створимо лінійну НМ.
» % Вхідний і цільовий прогнозований сигнал однакові:
» P=T;
» % Задання коефіцієнта навчання
» lr = 0.1;
» % Для прогнозу використовуються 5 попередніх значень
» delays = [1 2 3 4 5];

» % Створення і налаштування лінійної НМ
» net = newlin(minmax(cat(2,P{:})),1,delays,lr); % Створення НМ
» [net,y,e] = adapt(net,P,T); % Адаптивне налаштування мережі

» % Графіки початкового сигналу і прогнозу
»plot(time,cat(2,y{:}),time,cat(2,T{:}),'--')
» xlabel('Time');
» ylabel (' Predict signal ');
» title ('Start signal and predict '); (Рис 2.8)

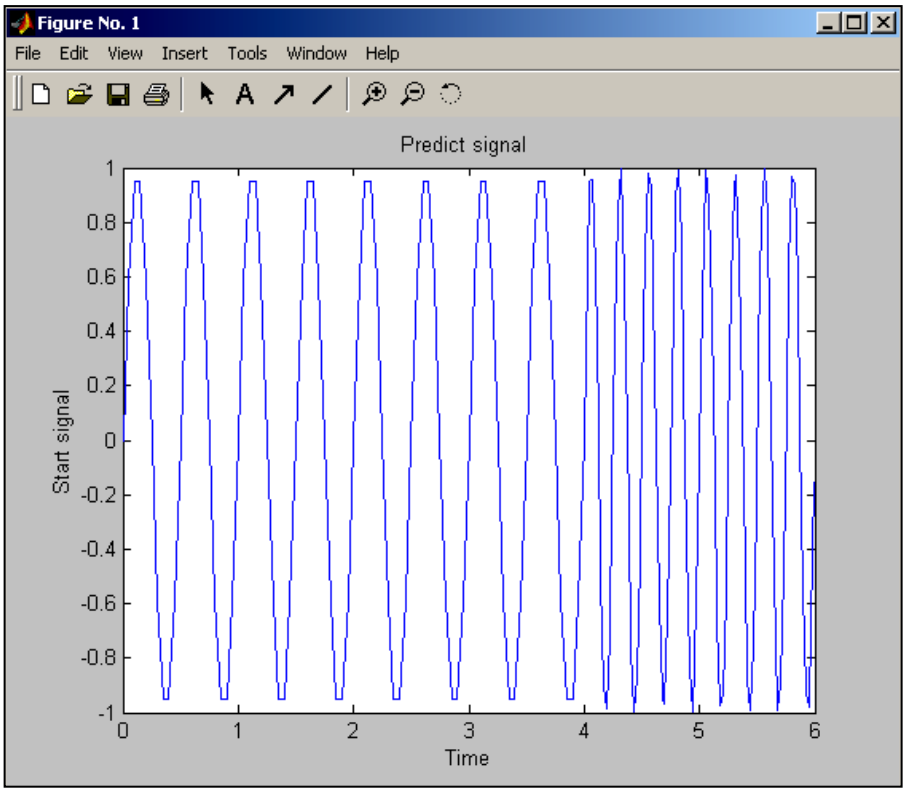
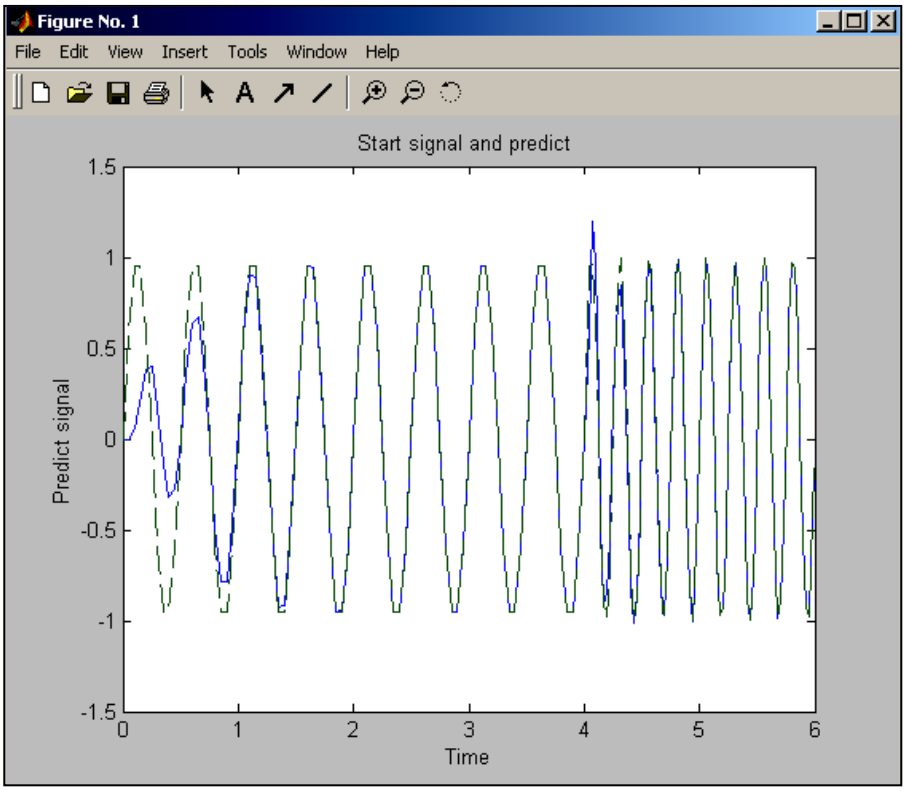


Рис. 2.7. Графік прогнозованого сигналу



Мал. 2.8. Початковий сигнал і прогноз

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведений приклад адаптивного лінійного прогнозу.
2. Згідно завдання викладача провести адаптивний лінійний прогноз та нарисувати блок-схему роботи НМ з розшифровкою функцій та їх аргументів.

2.7. Мережі Елмана

Розглянемо завдання відновлення форми сигналу з використанням рекурентної мережі Елмана. Нехай є два синусоїдальні сигнали – один з одиничною амплітудою, інший – з амплітудою, яка дорівнює двом:

```
p1 = sin(1:20);
p2 = sin(1:20)*2.
```

Нехай цільовим сигналом буде сигнал, складений з їх амплітудних значень:

```
t1 = ones(1,20);
t2 = ones(1,20)*2;
```

при цьому дані амплітуди чергуються, так що вхідні і цільові значення можуть бути представлені у формі:

```
p=[p1p2p1p2];
t = [t1 t2 t1 t2].
```

Перетворимо ці значення в послідовності:

Pseq = con2seq(p);

Tseq = con2seq(t).

Після цього можна безпосередньо перейти до проектуванню НМ. У ній запроектуємо один вхід і один вихід, тобто мережа матиме один вхідний елемент і один вихідний нейрон. Число нейронів у прихованому шарі може бути будь-яким (воно залежить від складності завдання); приймемо, що цей шар містить 10 нейронів. Подальше вирішення завдання ілюструється нижчим, при цьому на рисунку 2.9 приведено графік зміни помилки мережі в процесі її навчання, а на рисунку. 2.10 результати тестування мережі.

```
» % Задання початкових даних
» p1 = sin(1:20);
» t1 = ones(1,20);
» p2 = sin(1:20)*2;
» t2 = ones(1,20)*2;
» p = [p1 p2 p1 p2];
» t = [t1 t2 t1 t2];
» Pseq = con2seq(p);
» Tseq = con2seq(t);
»% Створення мережі Елмана з діапазоном входу [-2, 2] та 10 нейронами
» % прихованого шару, одним вихідним нейроном,
» % функцією активації у вигляді гіперболічного тангенса
» % для нейронів прихованого шару, лінійною функцією активації
» % для вихідного нейрона, функцією навчання з адаптацією
» % коефіцієнта навчання
» net = newelm([-2 2],[10 1],{ 'tansig', 'purelin'},'traingdx');
» % Задання параметрів навчання:
» net.trainParam.epochs = 500; % Число циклів навчання
» net.trainParam.goal= 0.01; % Цільове значення функції помилки
» net.performFcn = 'sse'; % Задання вигляду функції помилки

» % Навчання мережі. За умовчанням проміжні результати навчання
» % виводяться через 25 циклів
» [net,tr] = train(net,Pseq,Tseq);

» % Побудова графіка функції помилки
» semilogy(tr.epoch,tr.perf);
» title( 'The sum of squares of the errors Elman’s network');
» xlabel(' Cicles');
» ylabel( ' The sum of squares of the errors');

» % Тестування мережі
» a = sim(net,Pseq);
» time= 1:length(p);
» time= 1:length(p);
```

```
» plot(time,t,'-',time,cat(2,a{:}))
» title ( 'Testing results');
» xlabel( 'Time');
» ylabel( 'Preset values- - Neural networks ---')
```

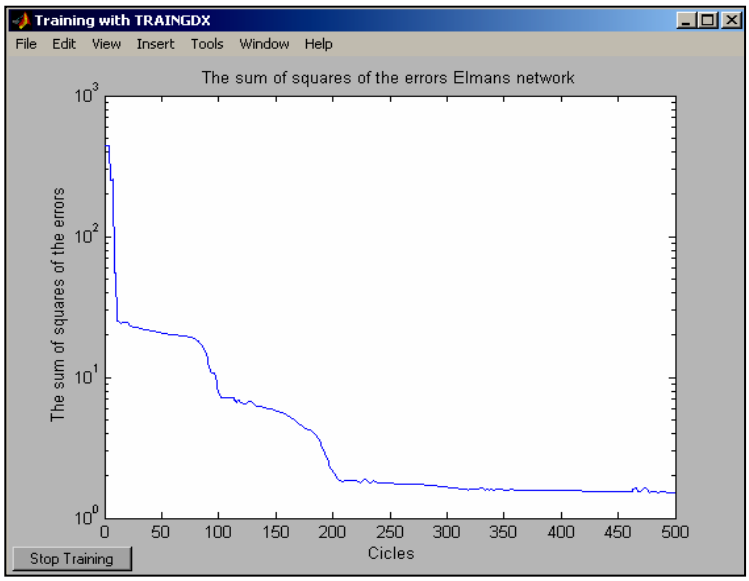


Рис. 2.9. Зміна помилки мережі в процесі навчання

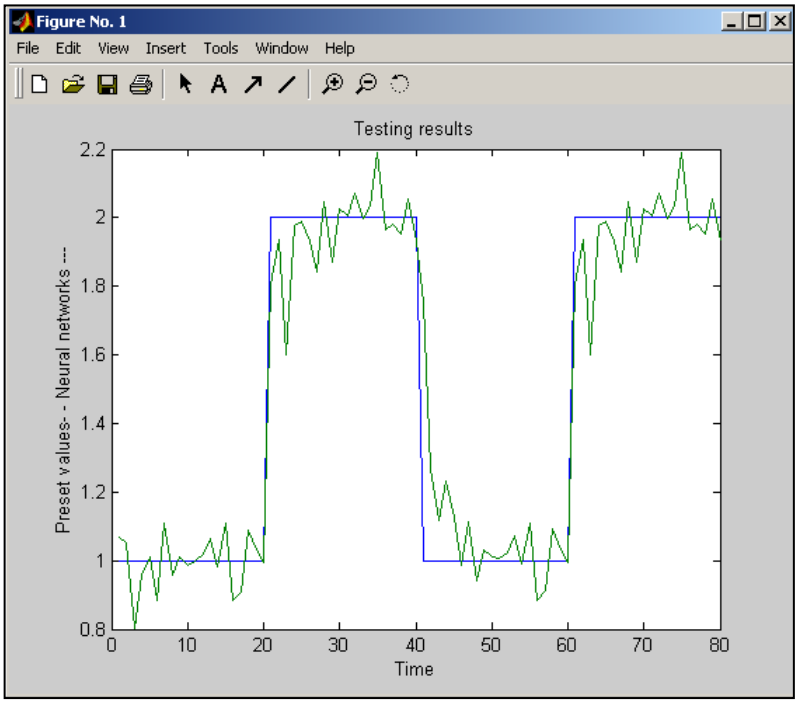


Рис. 2.10. Результати тестування мережі

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведений приклад використання мережі Елмана.
2. Згідно завдання викладача реалізувати НМ Елмана та нарисувати блок-схему роботи НМ з розшифровкою функцій та їх аргументів.

2.8. Мережі зустрічного розповсюдження

Припустимо, поставлено наступне завдання класифікації: задано набір з 10 векторів, представлених у вигляді стовпців-матриць, також задано вектор-рядок, який вказує належність кожного вектора до одного з двох класів:

Tc = [1 1 1 2 2 2 2 1 1 1].

Потрібно: побудувати автоматичний класифікатор подібних векторів, використовуючи наведені дані як навчальну вибірку.

Рішення подібної задачі проведемо із застосуванням мережі зустрічного розповсюдження:

```
» P = [-3 -2 -2 0 0 0 0 +2 +2 +3; 0 +1 -1 +2 +1 -1 -2 +1 -1 0];
» C = [1 1 1 2 2 2 2 1 1 1];
» T = ind2vec(C); % Перетворення вектора C в матрицю T з двома рядками

» % Створення нової мережі зустрічного розповсюдження вимагає 4-х па-раметрів:
» % 1) матриці мінімальних і максимальних значень вхідних елементів
» % 2) числа прихованих нейронів
» % 3) вектор з елементами, які вказують частку кожного з класів
» % 4) величини коефіцієнта навчання

» net = newlvq(minmax(P),4,[.6 .4],0.1); % Створення мережі
» net.trainParam.epochs = 150; % Задання числа циклів навчання
»net.trainParam.show = Inf; % Заборона на видачу проміжних результатів
» net = train(net,P,T); % Навчання НМ

TRAINWB1, Epoch 150/150
TRAINWB1, Maximum epoch reached.

» Y = sim(net,P)% Тестування мережі

Y=
     1   1   1   0   0   0   0   1   1   1
     0   0   0   1   1   1   1   0   0   0
```

Як видно з результатів тестування, класифікація елементів навчальної вибірки проведена точно (три перших і три останні вектори віднесено до першого класу, останні – до другого).

Завдання

1. Реалізувати у *Neural Networks Toolbox* наведений приклад застосування НМ зустрічного розповсюдження.
2. Згідно завдання викладача реалізувати НМ зустрічного розповсюдження та нарисувати блок-схему роботи НМ з розшифровкою функцій та їх аргументів.

3. Створення і використання нейронних мереж у середовищі Simulink

Пакет Neural Network Toolbox містить ряд блоків, які або можуть бути безпосередньо використані для побудови нейронних мереж в середовищі *Simulink*, або застосовуватися разом з розглянутою вище функцією **gensim**.

Для виклику цього набору блоків, в командному рядку необхідно набрати команду **neural**, після виконання який з'являється відповідне робоче вікно (3.1).

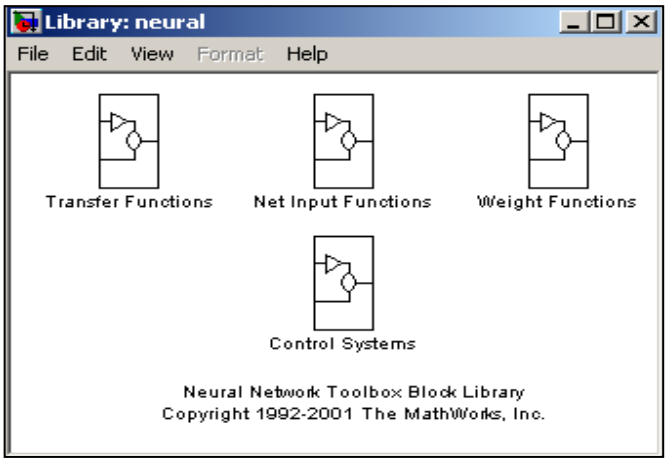


Рис 3.1. Основні нейромережеві блоки Simulink

Кожен з представлених на рисунку 3.1 блоків у свою чергу є набором (бібліотекою) деяких блоків. Розглянемо їх.

Блоки функцій активації (Transfer Functions). Подвійний клік лівої кнопки миші на блоці Transfer Functions призводить до появи бібліотеки функцій активації (рис. 3.2). Кожен з блоків даної бібліотеки перетворить поданий на нього вектор у відповідний вектор тієї ж розмірності.

Блоки перетворення входів мережі. Проводячи аналогічну розглянутій операцію, але з блоком *Net Input Functions*, прийдемо до бібліотеки блоків представлених на рисунку 3.3. Блоки даної бібліотеки реалізують розглянуті вище функції перетворення входів мережі.

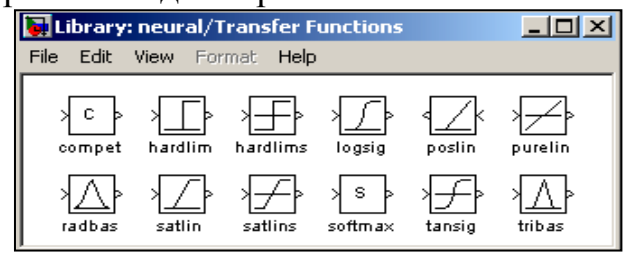


Рис. 3.2. Бібліотека функцій активації

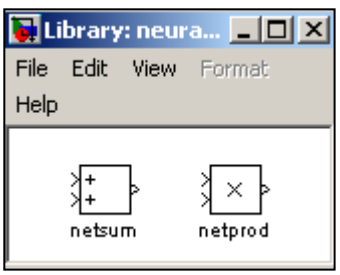


Рис. 3.3. Блоки бібліотеки перетворення входів мережі

Інша бібліотека – бібліотека блоків, що реалізовують функції вагів і зсувів.

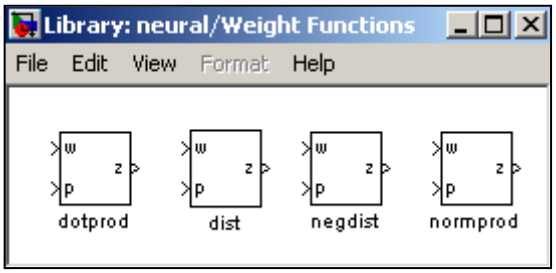


Рис. 3.4. Блоки бібліотеки функцій вагів та зсувів

Відзначимо, що при задані конкретних числових значень, під час операції із всіма приведеними блоками зважаючи на особливості Simulink вектори необхідно представляти як стовпці, а не як рядки (як це було до цих пір).

Основною функцією для формування нейромережевих моделей в *Simulink* є функція **gensim**, яка записується у формі: **gensim(net,st)**, де: **net** – ім'я створеної НМ, **st** – інтервал дискретизації (якщо НМ не має затримок, що асоціюються з її входами або шарами, значення даного аргументу встановлюється рівним -1).

Приклад

Хай вхідний і цільовий вектори мають вигляд:

p=[1 2 3 4 5];
t=[1 3 5 7 9].

Створимо лінійну НМ і протестуємо її:

```
» p = [1 2 3 4 5];
» t = [1 3 5 7 9];
» net = newlind(p,t);
» y = sim(net,p)
   y= 1.0000    3.0000    5.0000    7.0000    9.0000
```

Потім запустимо Simulink командою **» gensim(net, -1)** (рис. 3.5).

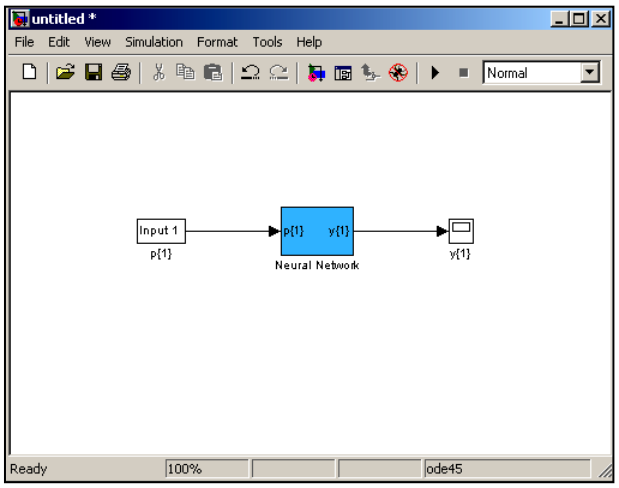


Рис. 3.5. Створена нейромережева модель Simulink

Для проведення тестування моделі клікнемо двічі по лівій іконі (Input 1), що приведе до відкриття діалогового вікна (рис. 3.6).

В даному випадку блок Input 1 є стандартним блоком задання константи (Constant). Змінимо значення за умовчанням на 2 і натиснемо кнопку OK. Потім натиснемо кнопку Start в меню моделювання. Розрахунок нового значення мережею проводиться практично миттєво. Для його виводу необхідно двічі клацнути мишею по правій іконі (блок y(1)). Результат обчислень відображається на інтерфейсному дисплеї, результат дорівнює 3, як і потрібно.

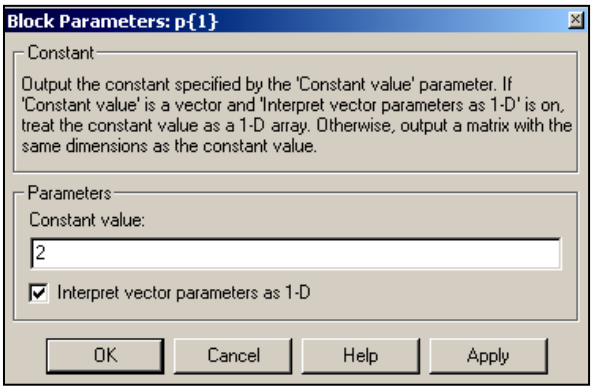


Рис. 3.6. Діалогове вікно задання входу НМ

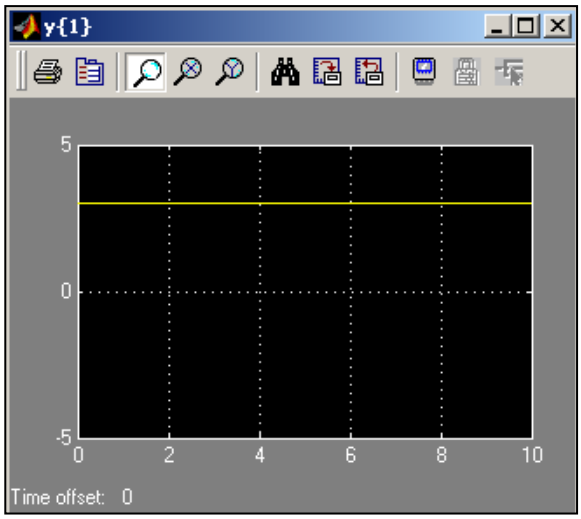


Рис. 3.6. Вікно з виходом НМ

Відзначимо, що, двічі клікаючи лівою кнопкою миші по блоку **Neural Network**, потім – по блоку **Layer 1**, можна отримати детальну графічну інформацію щодо архітектури мережі (рис. 3.7).

Із створеною мережею можна проводити різні операції, можливі в середовищі Simulink; взагалі за допомогою команди **gensim** здійснюється інтеграція створених нейромреж у блок-діаграми цього пакету з використанням інструментів моделювання різних систем, що є при цьому (наприклад, вбудовувати нейромережевий регулятор в систему управління і моделювання).

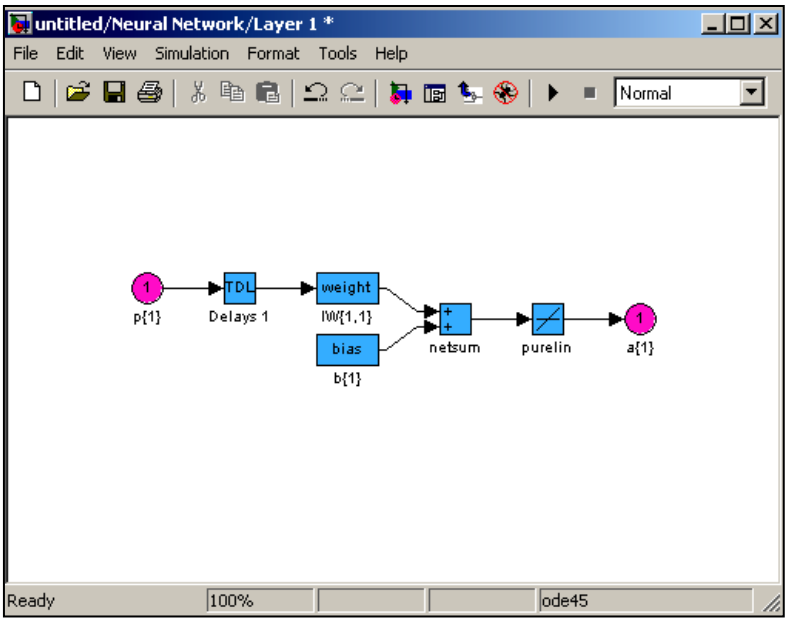


Рис. 3.7. Архітектура створеної НМ

Завдання

1. Реалізувати у *Simulink* наведений приклад застосування лінійної НМ.
2. Згідно завдання викладача реалізувати лінійну НМ у *Simulink*.

ЛІТЕРАТУРА

1. Корчемний М.О. Нейронні мережі / М.О. Корчемний, В.П. Лисенко, М.В. Чапний. – К.: НАУ, 2008. – 156 с.
2. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы: Пер с польск / Д. Рутковская, М. Пилиньский, Л. Рутковский – М.: Горячая линия - Телеком, 2004. – 452 с.
3. Круглов В.В. Искусственные нейронные сети. Теория и практика - М.: Горячая линия – Телеком, 2002. – 382 с.
4. Рідкокаша А.А. Основи систем штучного інтелекту. Навчальний посібник / А.А. Рідко каша, К.К. Голдер. – Черкаси: "ВІДЛУННЯ-ПЛЮС", 2002. – 240 с.
5. Рассел С. Искусственный интеллект: современный подход, 2-е изд.: Пер с англ / С. Рассел, П. Норвиг. – М.: Вильямс, 2006. – 1408 с.
6. Люгер Дж.Ф. Искусственный интеллект: стратегии и методы решения сложных проблем / Пер. с англ. – М.: Вильямс, 2005. – 864 с.
7. Советов Е. Я. Моделирование систем. Учебн. для вузов.- М: Высшая школа, 1985. – 271 с.