# 2. МОДЕЛЬ НЕЙРОНА ТА АРХІТЕКТУРА МЕРЕЖІ

## 2.1. Модель нейрона

### 2.1.1. Простий нейрон

Елементарною складовою нейронної мережі є нейрон. Структура нейрона з єдиним скалярним входом показана на рис. 1 а.

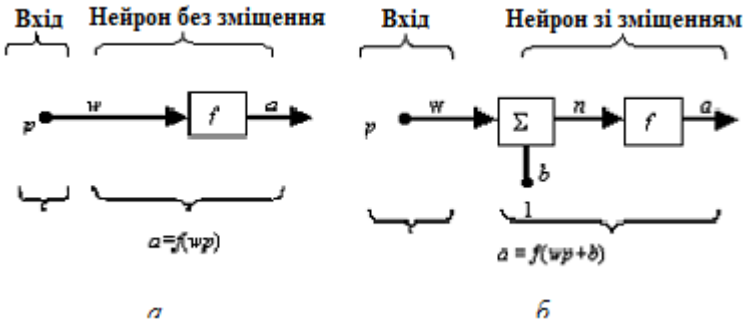



Рис. 1 (а, б)

Скалярний вхідний сигнал *p* множиться на скалярний ваговий коефіцієнт *w* і результуючий зважений вхід *w*p* є аргументом функції активації *f* нейрона, яка породжує скалярний вихід *a*.

Нейрон показаний на рис. 1 б, доповнений скалярним зміщенням *b*. Зміщення додається із зваженим входом *w*p* і приводить до зсуву аргументу функції *f* на величину *b*. Дію зміщення можна звести до схеми зважування, якщо уявити, що нейрон має другий вхідний сигнал зі значенням, рівним 1. Вхід *n* функції активації нейрона, як і раніше, залишається скалярним і рівним сумі зваженого входу та зміщення *b*. Ця сума є аргументом функції активації *f*; Виходом функції активації є сигнал *a*. Константи *w* та *b* є скалярними параметрами нейрона. Основний принцип роботи нейронної мережі полягає в налаштуванні параметрів нейрона таким чином, щоб поведінка мережі відповідала деякій бажаній поведінці. Регулюючи ваги чи параметри зміщення, можна навчити мережу виконувати конкретну роботу.

Рівняння нейрона зі зміщенням має вигляд

$$a = f(w * p + b * 1).$$

Як зазначалося, зміщення b – скалярний параметр нейрона, що налаштовується, який не є входом, а константа 1, яка керує зсувом, розглядається, як вхід і може бути врахована наступним чином

$$a = \begin{bmatrix} w & b \end{bmatrix} \begin{bmatrix} p \\ 1 \end{bmatrix}.$$

### 2.1.2. Функція активації

Функції активації (передавальні функції) нейрона можуть мати різний вигляд. Функція активації *f*, як правило, належить до класу сигмоїдальних[1] функцій з аргументом *n* та виходом *a*.

Розглянемо три найпоширеніші форми функції активації.

*Одинична функція активації* hardlim порогового типу. Ця функція описується співвідношенням *a* = hardlim(*n*) = 1(*n*) і на рис. 2. Вона дорівнює 0, якщо *n* < 0, і 1, якщо *n*≥0.

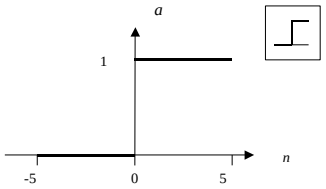



Рис. 2.

До складу ППП Neural Network Toolbox входить М-функція hardlim, що реалізує цю функцію активації. Тепер можна побудувати графік цієї функції, використовуючи оператори мови MATLAB:

```
n = -5: 0.1: 5;
plot(n,hardlim(n),'c+:');
```

В результаті отримаємо графік функції hardlim у діапазоні значень входу від –5 до +5 (Рис. 2).

*Лінійна функція* активації purelin. Ця функція описується співвідношенням *a*= purelin(*n*) = *n* і показана на рис. 3.

[1]Сигмоїдальною (S-подібною) функцією називається неперервна функція, що має дві горизонтальні асимптоти та одну точку перегину.

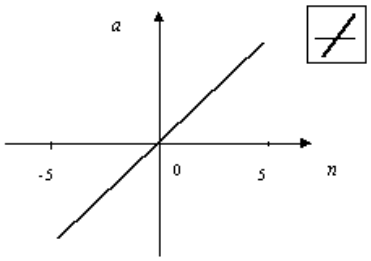



Рис. 3.

*Логістична функція*активації logsig. Ця функція описується співвідношенням $a$= logsig($n$) = 1/(1 + exp(-$n$)) і показана на рис. 4. Вона належить до класу сигмоїдальних функцій, і її аргумент може набувати будь-якого значення в діапазоні від –∞ до +∞, А вихід змінюється в діапазоні від 0 до 1. У ППП Neural Network Toolbox вона представлена М-функцією logsig. Завдяки властивості диференційованості ця функція часто використовується в мережах з навчанням на основі методу зворотного розповсюдження помилки.

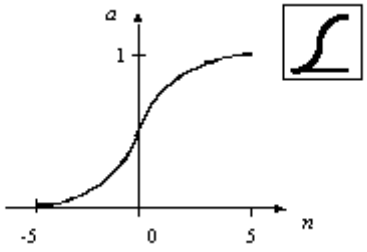



Рис. 4.

Символ у квадраті у правому верхньому кутку графіка характеризує функцію активації. Це зображення використовується на структурних схемах нейронних мереж.

У ППП Neural Network Toolbox включені інші функції активації. Використовуючи мову MATLAB, користувач може створювати свої власні унікальні функції.

### 2.1.3. Нейрон із векторним входом

Нейрон з одним вектором входу $p$ з $R$ елементами $p_1$, $p_2$, …, $p_R$ показаний на рис. 5. Тут кожен елемент входу множиться на ваги $w_{11}$, $w_{12}$, …, $w_{1R}$ відповідно і зважені значення передаються суматор. Їхня сума дорівнює скалярному добутку вектор-рядка $W$ на вектор входу $p$.

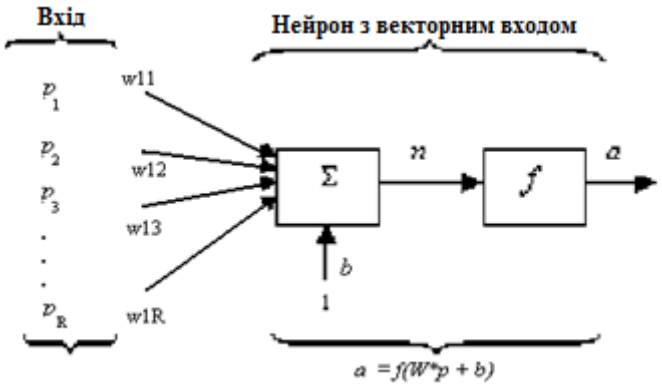



Рис. 5

Нейрон має зміщення $b$, яке додається із зваженою сумою входів. Результуюча сума $n$ дорівнює

$$n = w_{11} p_1 + w_{12} p_2 + \ldots + w_{1R} p_R + b$$

і є аргументом функції активації $f$. У нотатках мови MATLAB цей вираз записується так:

$$n = W * p + b.$$

Структура нейрона, наведена вище, містить багато зайвих деталей. Під час розгляду мереж з великої кількості нейронів, використовуватиметься укрупнена структурна схема нейрона (рис. 6).

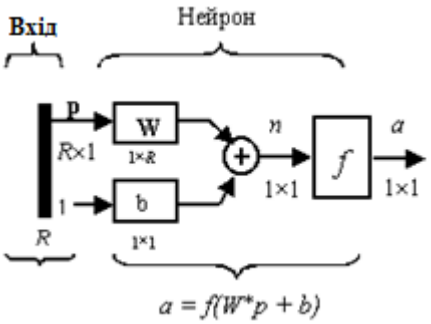



Рис. 6

Вхід нейрона зображується у вигляді темної вертикальної риски, під якою вказується кількість елементів входу *R*. Розмір вектора входу *p* вказується нижче за символ *p* і дорівнює R×1. Вектор входу множиться на вектор-рядок *W* довжини *R*. Як і раніше, константа 1 розглядається як вхід, який множиться на скалярне зміщення *b*. Входом функції активації нейрона служить сума зміщення *b* і добутку *W* * *p*. Ця сума перетворюється функцією активації *f*, на виході якої отримуємо вихід нейрона *a*, який у цьому випадку є скалярною величиною. Структурна схема, наведена на рис. 6 називається прошарком мережі. Прошарок характеризується матрицею ваг *W*, зміщенням *b*, операціями множення *W*p*, додавання та функцією активації *f*. Вектор входів *p* зазвичай не включається до характеристики прошарку.

Щоразу, коли використовується скорочене позначення мережі, розмірність матриць вказується під іменами векторно-матричних змінних. Ця система позначень пояснює будову мережі та пов'язану з нею матричну математику.

На укрупненій структурній схемі для позначення типу функції активації використовуються спеціальні графічні символи; деякі з них наведені на рис. 7 де. *а* – порогова, *б* – лінійна, *в* – логістична функція.

hardlim *а*  purelin *б*  logsig *в*

Рис. 7

## 2.2. Архітектура нейронних мереж

Реальна нейронна мережа може містити один або більше прошарків і відповідно характеризуватись як одношарова або як багатошарова.

### 2.2.1. Одношарові мережі

Розгорнута схема мережі з одного прошарку з *R* вхідними елементами та *S* нейронами показана на рис. 8.

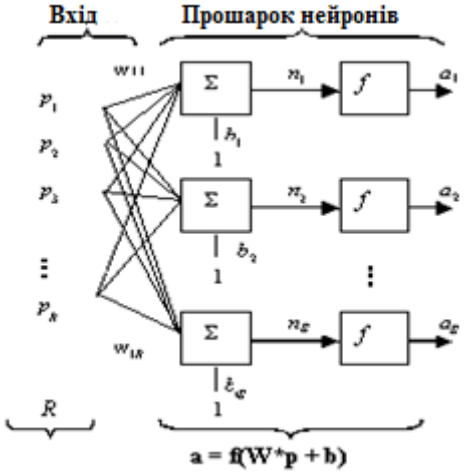



Рис. 8

У цій мережі кожен елемент вектора входу з'єднаний з усіма входами нейрона і це з'єднання задається матрицею ваг *W*; при цьому кожен *i*-й нейрон включає підсумовуючий елемент, який формує скалярний вихід n(*i*). Сукупність скалярних функцій n(*i*) поєднується в *S*-елементний вектор входу *n* функції активації прошарку. Виходи прошарку нейронів формують вектор-стовпець *a*, і, таким чином, опис прошарку нейронів має вигляд:

$$\mathbf{a} = \mathbf{f}(\mathbf{W} * \mathbf{p} + \mathbf{b}) \bullet$$

Кількість входів $R$ у прошарку може не збігатися з кількістю нейронів $S$. У кожному прошарку, як правило, використовується та сама функція активації. Однак можна створювати складені прошарки нейронів з використанням різних функцій активації, з'єднуючи мережі, подібні до зображеної на рис. 8, паралельно. Обидві мережі матимуть ті самі входи, і кожна мережа генеруватиме певну частину виходів. Елементи вектора входу передаються в мережу через матрицю ваг $W$, що має вигляд:

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1R} \\ w_{21} & w_{22} & \dots & w_{2R} \\ \dots & \dots & \dots & \dots \\ w_{S1} & w_{S2} & \dots & w_{SR} \end{bmatrix}.$$

Зауважимо, що індекси рядків матриці W вказують адресатів (пункти призначення) ваг нейронів, а індекси стовпців – яке джерело є входом для цієї ваги. Таким чином, елемент матриці ваг $w_{12}$ = W(1, 2) визначає коефіцієнт, на який множиться другий елемент входу при передачі його на перший нейрон.

Для одношарової мережі з нейронами $S$ укрупнена структурна схема показана на рис. 9.

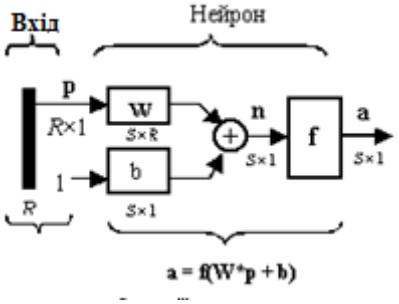



Рис. 9

Тут $p$ – вектор входу розміру R×1, $W$ – вагова матриця розміру S×R, $a$, $b$, $n$ – вектори розміру S×1.

## 2.2.2. Багатошарові мережі

Розглянемо мережі, які мають кілька прошарків. Будемо називати вагові матриці, з'єднані з входами, вагами входу прошарку, а вагові матриці для сигналів, що виходять із прошарку, назвемо вагами виходу прошарку. Далі будемо використовувати верхні індекси, щоб вказати джерело та адресат для різних ваг та інших елементів нейронної мережі. Щоб пояснити це, розглянемо спочатку лише один, перший прошарок багатошарової мережі (рис. 10).

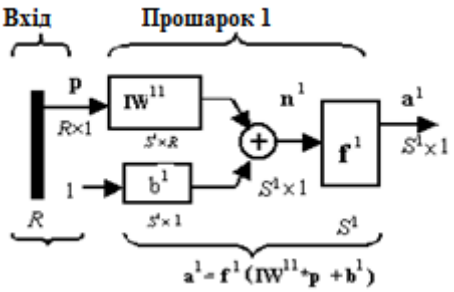



Рис. 10

Позначимо вагову матрицю, пов'язану з входами через $IW^{11}$, верхні індекси якої вказують, що джерелом входів є перший прошарок (другий індекс) і адресатом є перший прошарок (перший індекс). Елементи цього прошарку, такі як зміщення $b^1$, вхід функції активації $n^1$ і вихід прошарку $a^1$, мають верхній індекс 1, щоб позначити, що вони пов'язані з першим прошарком. Надалі для матриць ваг входу та виходу прошарку будуть використані позначення IW(Input Weight) та LW(Layer Weight) відповідно.

Коли мережа має кілька прошарків, кожен прошарок має свою матрицю ваг $W$, вектор зміщення $b$ і вектор виходу $a$. Щоб розрізняти вагові матриці, вектори виходу і т. д. для кожного з цих прошарків, введемо номер прошарку як верхній індекс для змінної, що представляє інтерес. Використання цієї системи позначень для мережі із трьох прошарків можна бачити на наведеній нижче структурній схемі та з рівнянь, наведених у нижній частині рис. 11.

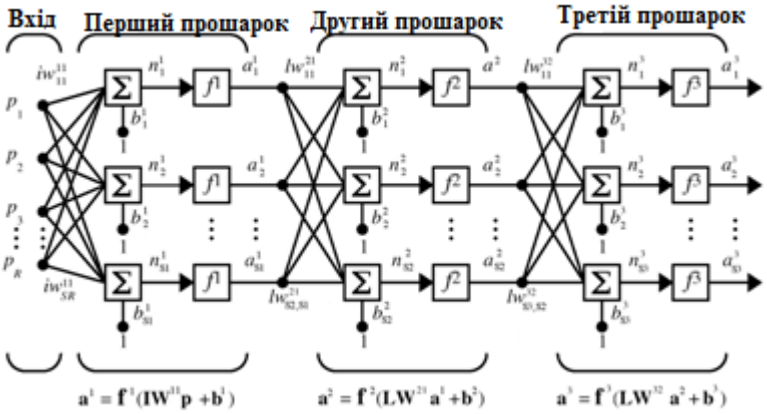



Рис. 11

Мережа, показана вище, має R входів, $S^1$ нейронів у першому прошарку, $S^2$ нейронів у другому прошарку і т. д. Для загальності вважатимемо, що різні прошарки мають різну кількість нейронів. На зміщення кожного нейрона подано постійний вхідний сигнал 1. Зауважимо, що виходи кожного проміжного прошарку служать входами для наступного прошарку. Таким чином, прошарок 2 може бути розглянутий як один прошарок мережі з входами $S^1$, нейронами $S^2$ і $S^1 \times S^2$ матрицею ваг $W^2$. Вхід до прошарку 2 є 1, а вихід - 2. Тепер, коли позначені всі вектори та матриці 2 прошарку, можна трактувати його як самостійну одношарову мережу. Такий підхід може бути використаний до будь-якого прошарку мережі.

Прошарки багатошарової мережі мають різні призначення. Прошарок, який утворює вихід мережі, називається прошарком виходу. Всі інші прошарки називаються прихованими прошарками. Тришарова мережа, показана вище, має вихідний прошарок (прошарок 3) і прихованих два прошарки (прошарок 1 і прошарок 2). Ця ж тришарова мережа може бути представлена у вигляді укрупненої структурної схеми (рис. 12).

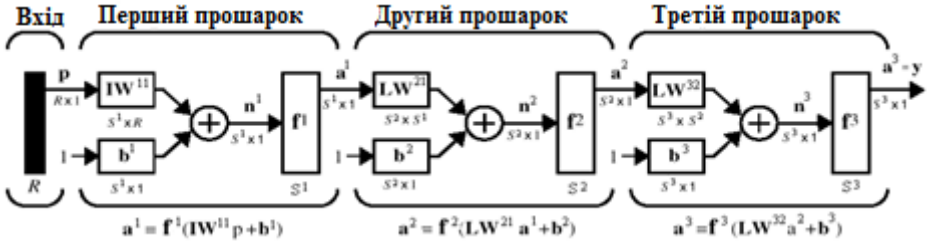



Рис.12

Зауважимо, що вихід третього прошарку $a^3$ позначено через $y$. Це зроблено для того, щоб наголосити, що вихід останнього прошарку є виходом мережі.

Багатошарові мережі мають дуже потужні можливості. Наприклад, двошарова мережа, у якій перший прошарок містить сигмоїдальну, а другий прошарок – лінійну функцію активації, може бути навчена апроксимувати з довільною точністю будь-яку функцію з кінцевим числом точок розриву.

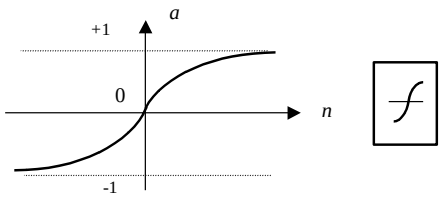
На закінчення можна сформулювати такі висновки. Вхід функції активації нейрона визначається зміщенням та сумою зважених входів. Вихід нейрона залежить як від входів нейрона, і від виду функції активації. Один нейрон не може вирішувати складні завдання, проте кілька нейронів, об'єднаних в один або кілька прошарків, мають великі можливості.

Архітектура мережі складається з опису того, скільки прошарків має мережа, кількості нейронів у кожному прошарку, виду функції активації кожного прошарку та інформації про з'єднання прошарків. Архітектура мережі залежить від того конкретного завдання, яке має вирішувати мережа.

Робота мережі полягає у обчисленні виходів мережі на основі відомих входів з метою формування бажаного відображення вхід/вихід. Конкретне завдання визначає кількість входів та кількість виходів мережі. Крім числа нейронів у вихідному прошарку мережі, для проектувальника важлива кількість нейронів у кожному прошарку. Більша кількість нейронів у прихованих шарах забезпечує більш потужну мережу. Якщо має бути реалізовано лінійне відображення, слід використовувати нейрони з лінійними функціями активації. При цьому треба пам'ятати, що лінійні нейронні мережі неспроможні формувати нелінійні відображення. Використання нелінійних функцій активації дозволяє налаштувати нейронну мережу на реалізацію нелінійних зв'язків між входом та виходом.

Мережі зі зміщенням дозволяють формувати складніші зв'язки між входами та виходами, ніж мережі без зміщення. Наприклад, нейрон без зміщення, коли всі входи нульові, завжди задаватиме вхід функції активації рівним нулю, проте нейрон зі зміщенням може бути навчений так, щоб за тих же умов задати вхід функції активації довільної форми.

У багатошарових мережах часто використовуються нелінійні сигмоїдні функції активації типу логістичної (див. рис. 7, в) або гіперболічного тангенсу (рис. 13).




$a$ = tansig($n$)

Рис. 13

Якщо останній прошарок багатошарової мережі використовує такі функції активації, виходи мережі будуть обмежені. Коли у вихідному прошарку використовуються лінійні нейрони, виходи мережі можуть приймати довільні значення. У ППП NNT передбачені М-функції, що дозволяють обчислювати похідні функції активації. Щоб отримати інформацію про ім'я потрібної функції, слід скористатися наступним оператором:

<ім'я_функції_активації>('deriv')

Наприклад, звернення виду

```
tansig('deriv')
ans = dtansig
```

дає можливість дізнатися, що ім'я М-функції, що дозволяє обчислити похідну гіперболічного тангенсу, dtansig.

### 2.2.3. Мережі із прямою передачею сигналу

Одношарова мережа з S нейронами з функціями активації logsig, що має R входів, показана на рис. 14.

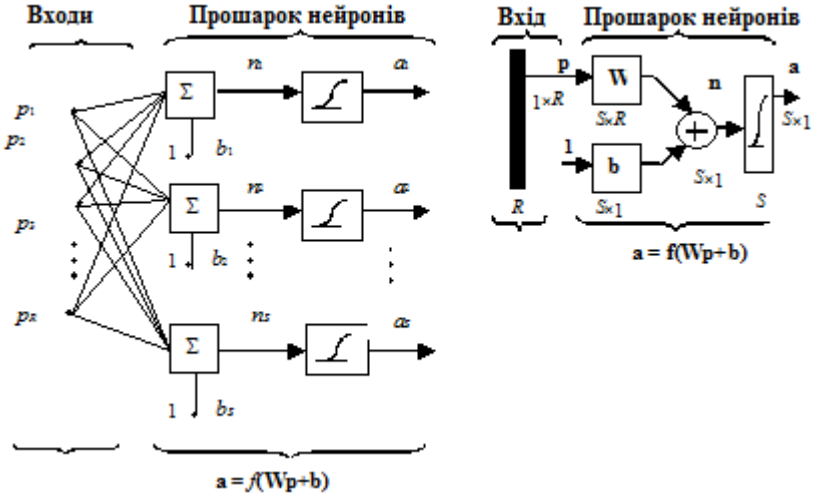



Рис. 14

Ця мережа, яка не має зворотних зв'язків, називається *мережею із прямою передачею сигналу*.Такі мережі часто мають один або більше прихованих прошарків нейронів із сигмоїдальними функціями активації, у той час як вихідний прошарок містить нейрони з лінійними функціями активації. Мережі з такою архітектурою можуть відтворювати складні нелінійні залежності між входом і виходом мережі.

Усі позначення в багато шаровій нейронній мережі з рисунку 15 були уже означені вище.

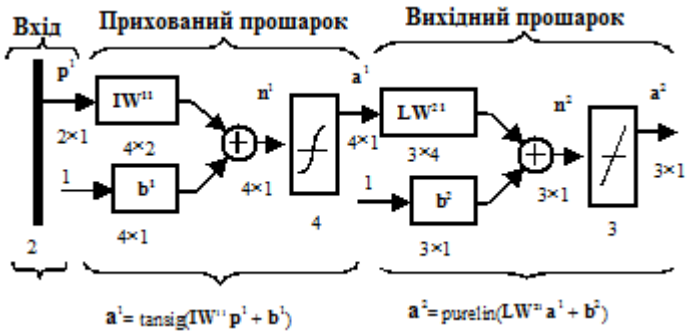



Рис. 15

Ця мережа може бути використана для апроксимації функцій. Вона може точно відтворити будь-яку функцію із скінченним числом точок розриву, якщо задати достатню кількість нейронів прихованого прошарку.

## 2.3. Створення, ініціалізація та моделювання мережі

### *Формування архітектури мережі*

Перший крок під час роботи з нейронними мережами – це створення моделі мережі. Для створення мереж із прямою передачею сигналу в ППП NNT призначена функція **newff**. Вона має 4 вхідні аргументи та 1 вихідний аргумент – об'єкт класу network. Перший вхідний аргумент – це масив розміру R×2, що містить допустимі межі значень (мінімальне та максимальне) для кожного з R елементів вектора входу; другий - масив для задання кількості нейронів кожного прошарку; третій - масив комірок, що містить імена функцій активації для кожного прошарку; четвертий – ім'я функції навчання.

Наприклад, наступний оператор створює мережу із прямою передачею сигналу:

```
net = newff([–1 2; 0 5],[3,1],{'tansig','purelin'},'traingd');
```

Ця мережа використовує 1 вектор входу з двома елементами, що мають допустимі межі значень [-12] і [05]; має 2 прошарки з трьома нейронами у першому прошарку та одним нейроном у другому прошарку; використовувані функції активації: tansig – у першому прошарку, purelin – у другому прошарку; використовується функція навчання– traingd.

М-функція newff не тільки створює архітектуру мережі, а й ініціалізує її ваги та зміщення, готуючи нейронну мережу до навчання. Проте існують ситуації, коли потрібна спеціальна процедура ініціалізації мережі.

### *Ініціалізація мережі*

Після того, як сформована архітектура мережі, повинні бути задані початкові значення ваг та зміщень, або, іншими словами, мережа повинна бути ініціалізована. Така процедура виконується за допомогою методу init для об'єктів класу мережі. Оператор виклику цього методу має вигляд:

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

Спосіб ініціалізації залежить від вибору параметрів мережі net.initFcn та net.layers{i}.initFcn,які встановлюють ту чи іншу функцію ініціалізації. Параметр net.initFcn визначає функцію ініціалізації для всієї мережі. Для мереж із прямою передачею сигналу за замовчуванням використовується функція ініціалізації initlay, яка дозволяє кожному прошарку мережі використовувати власну функцію ініціалізації, що визначається властивістю net.layers{i}.initFcn.

Для мереж із прямою передачею сигналу зазвичай застосовується одна з двох функцій ініціалізації прошарку: initwb або initnw.

Функція initwb дозволяє використовувати власні функції ініціалізації для кожної матриці ваги входу та вектора зсувів, задаючи параметри net.inputWeights{i, j}.initFcn та net.biases{i}.initFcn. Для мереж без зворотніх зв'язків з лінійними функціями активації ваги зазвичай ініціалізуються випадковими значеннями з інтервалу [–1 1].

Функція initnw застосовується для прошарків, які використовують сигмоїдні функції активації. Вона генерує початкові ваги та зміщення для прошарку так, щоб активні області нейронів були рівномірно розподілені щодо області значень входу. Це має кілька переваг у порівнянні з випадковим розподілом ваг і зміщень: по-перше, надлишок нейронів мінімальний, оскільки активні області всіх нейронів відповідають області значень входу, по-друге, навчання виконується швидше, тому що для кожної області значень входу знайдуться нейрони з тією ж областю визначення аргументу.

У розглянутому вище прикладі створення мережі із прямою передачею сигналу метод init викликається автоматично при зверненні до М-функції newff. Тому ініціалізація мережі виконується за замовчуванням. Якщо користувач хоче застосувати спеціальний метод ініціалізації або примусово встановити значення ваг і зміщень, то він може безпосередньо звернутися до функції init.

Наприклад, якщо ми хочемо знову ініціалізувати ваги і зміщення в першому прошарку, використовуючи функцію rands, то треба ввести наступну послідовність операторів:

```
net.layers{1}.initFcn = 'initwb';
net.inputWeights{1,1}.initFcn = 'rands';
net.biases{1,1}.initFcn = 'rands';
net.biases{2,1}.initFcn = 'rands';
net = init (net);
```

### *Моделювання мережі*

**Статичні мережі.** Статична нейронна мережа характеризується тим, що у її складі немає елементів затримки та зворотних зв'язків. Її поведінка не залежить від типу вектора входу, оскільки вектори, що послідовно подаються, можна розглядати як діючі одночасно або як один об'єднаний вектор. Тому як модель статичної мережі розглянемо мережу, показану на рис. 16.

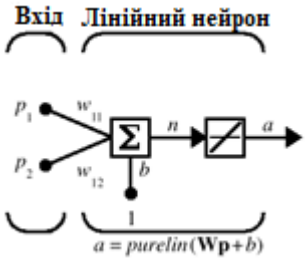



Рис. 16

Це одношарова мережа з двоелементним вектором входу та лінійною функцією активації. Для задання такої мережі призначено М-функцію newlin з ППП Neural Network Toolbox, яка вимагає вказати мінімальне та максимальне значення для кожного з елементів входу; в даному випадку вони рівні –1 і 1 відповідно, і навіть кількість прошарків, в даному випадку - 1.

```
% Формування одношарової лінійної мережі net із двоелементним
% вхідним сигналом із значеннями від -1 до 1
net = newlin([-1 1;-1 1],1);
```

Визначимо вагову матрицю і зміщення рівними W = [1 2], b = 0, і задаємо ці значення, використовуючи опис структури мережі

```
net.IW{1,1} = [1 2]; % Присвоєння значень ваг
net.b{1} = 0; % Присвоєння значення зміщення
```

Припустимо, що на мережу подається така послідовність із чотирьох векторів входу:

$$\left\{\begin{bmatrix}-1\\0\end{bmatrix},\begin{bmatrix}0\\-1\end{bmatrix},\begin{bmatrix}0\\1\end{bmatrix},\begin{bmatrix}1\\-1\end{bmatrix}\right\}.$$

Оскільки мережа статична, можна перегрупувати цю послідовність у наступний числовий масив:

```
P = [-1 0 0 1; 0 -1 1 -1];
```

Тепер можна моделювати мережу:

```
A = sim(net,P) % Моделювання мережі net із вектором входу P та виходом A
A = -1 -2 2 -1
```

Результат слід інтерпретувати так. На вхід мережі подається послідовність чотирьох вхідних сигналів, і мережа генерує вектор виходу з чотирьох елементів. Результат був би той самий, якби було 4 однакових мережі, що функціонують паралельно, і на кожну мережу був би поданий один із векторів входу та генерувався один із виходів.

**Динамічні мережі.** Коли мережа містить лінії затримки, вхід мережі слід розглядати як послідовність векторів, що подаються на мережу у певні моменти часу. Щоб пояснити цей випадок, розглянемо просту лінійну мережу, що містить один елемент лінії затримки (рис. 17).

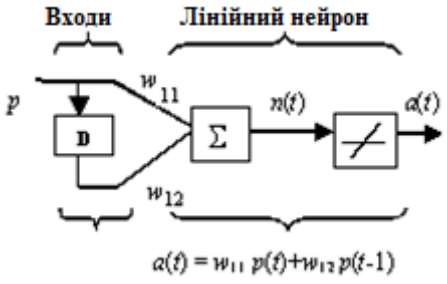



Рис. 17

Побудуємо таку мережу:

```
% Створення одношарової лінійної мережі з лінією затримки [0 1]
net = newlin([-1 1], 1, [0 1]);
```

Задамо наступну матрицю ваг W = [1 2] і нульове зміщення:

```
net.IW{1,1} = [1 2]; % Присвоєння значень ваг
net.biasConnect = 0; % Присвоєння значень зміщень
```

Припустимо, що вхідна послідовність має вигляд {–1, –1/2, 1/2, 1}, і задамо її у вигляді масиву комірок

```
P = {-1 -1/2 1/2 1};
```

Тепер можна моделювати мережу, використовуючи метод sim:

```
A = sim(net,P) % Моделювання мережі net із вхідним сигналом P та виходом A
A = [-1] [-5/2] [-1/2] [2]
```

Справді,

$$[1\ \ 2]\begin{bmatrix}-1\\0\end{bmatrix}=[-1]\ \ [1\ \ 2]\begin{bmatrix}-1/2\\-1\end{bmatrix}=[-5/2]\ \ [1\ \ 2]\begin{bmatrix}1/2\\-1/2\end{bmatrix}=[-1/2]\ \ [1\ \ 2]\begin{bmatrix}1\\1/2\end{bmatrix}=[2].$$

Ввівши масив комірок, що містить послідовність входів, мережа згенерувала масив комірок, що містить послідовність виходів. У цьому випадку кожен вихід формується відповідно до співвідношення

$$a(t) = p(t) + 2p(t-1).$$

При зміні порядку проходження елементів у вхідній послідовності будуть змінюватися значення на виході.

Якщо ті самі входи подати на мережу одночасно, то отримаємо зовсім іншу реакцію. Для цього сформуємо наступний вектор входу:

```
P = [-1 -1/2 1/2 1];
```

Після моделювання отримуємо:

```
A = sim(net,P) % Моделювання мережі
A = -1 -1/2 1/2 1
```

Результат такий самий, як якщо б ми застосували кожен вхід до окремої мережі та обчислити її вихід. Оскільки початкові умови для елементів затримки не вказані, за замовчуванням вони прийняті нульовими. У цьому випадку вихід мережі дорівнює

$$[1\ \ 2]\begin{bmatrix}-1 & -1/2 & 1/2 & 1\\ 0 & 0 & 0 & 0\end{bmatrix}=[-1\ \ -1/2\ \ 1/2\ \ 1].$$

Якщо потрібно моделювати реакцію мережі для кількох послідовностей сигналів на вході, треба сформувати масив комірок, розмір кожної з яких збігається з числом таких послідовностей. Нехай, наприклад, потрібно додати до мережі 2 послідовності:

$$p_1(1)=[-1],\ p_1(2)=[-1/2],\ p_1(3)=[1/2],\quad p_1(4)=[1];$$
$$p_2(1)=[1],\ \ p_2(2)=[1/2],\quad p_2(3)=[-1/2],\ \ p_2(4)=[-1].$$

Вхід P у цьому випадку має бути масивом комірок, кожна з яких містить два елементи за кількістю послідовностей

```
P = {[-1 1] [-1/2 1/2] [1/2 -1/2] [1 -1]};
```

Тепер можна моделювати мережу:

```
A = sim (net, P); % Моделювання мережі net з вхідним сигналом P та виходом A
```

Результуючий вихід мережі дорівнює

```
A = {[-1 1] [-5/2 5/2] [-1/2 1/2] [2 -2]}
```

Справді,

$$\{[1\ \ 2]\begin{bmatrix}-1 & 1\\ 0 & 0\end{bmatrix}=[-1\ \ 1],\qquad [1\ \ 2]\begin{bmatrix}-1/2 & 1/2\\ -1 & 1\end{bmatrix}=[-5/2\ \ 5/2],$$
$$[1\ \ 2]\begin{bmatrix}1/2 & -1/2\\ -1/2 & 1/2\end{bmatrix}=[-1/2\ \ 1/2],\qquad [1\ \ 2]\begin{bmatrix}1 & -1\\ 1/2 & -1/2\end{bmatrix}=[2\ \ -2]\}$$

На рис. 18 показаний формат масиву P, представленого Q вибірками (реалізаціями), при моделюванні мережі.

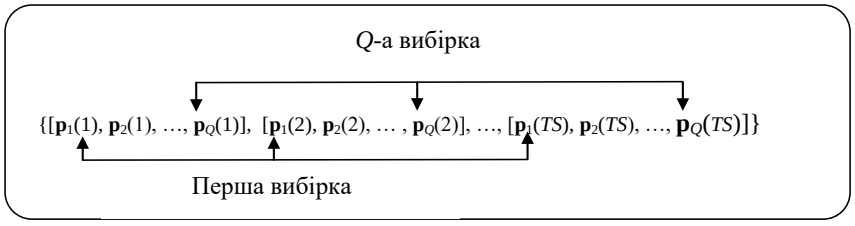



Рис. 18

В даному випадку це масив комірок з одним рядком, кожен елемент якого поєднує Q реалізацій вектора p – [$p_1$(TS), $p_2$(TS), …, $p_Q$(TS)] – для деякого моменту часу TS. Якщо на вхід мережі подається кілька векторів входу, кожному входу буде відповідати 1 рядок масиву комірок. Подання входів як масиву комірок відповідає послідовному представленню спостережуваних даних у часі.

Подання входів як числового масиву вибірок у форматі double відповідає груповому представленню даних, коли реалізації вектора входу для всіх значень часу на інтервалі вибірки обробляють потоком.

# 3. НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ

При вирішенні прикладних завдань за допомогою нейронних мереж необхідно зібрати достатній та представницький обсяг даних для того, щоб навчити нейронну мережу вирішенню таких завдань. Навчальний набір даних – це набір спостережень, що містять ознаки об'єкта, що вивчається. Перше питання, які ознаки використовувати і скільки та які спостереження треба провести?

Вибір ознак принаймні початковий здійснюється евристично на основі наявного досвіду, який може підказати, які ознаки є найважливішими. Спочатку слід включити всі ознаки, які, на думку аналітиків чи експертів, є суттєвими, на наступних етапах цю множину буде зменшено.

Нейронні мережі працюють із числовими даними, взятими, зазвичай, з деякого обмеженого діапазону. Це може створити проблеми, якщо значення спостережень виходять за межі цього діапазону або пропущені.

Питання про те, скільки потрібно мати спостережень для навчання мережі, часто виявляється непростим. Відомий ряд евристичних правил, які встановлюють зв'язок між кількістю необхідних спостережень та розмірами мережі. Найпростіше з них свідчить, що кількість спостережень має бути в 10 разів більше числа зв'язків у мережі. Насправді це число залежить від складності відображення, яке повинна відтворювати нейронна мережа. Зі зростанням кількості використовуваних ознак кількість спостережень зростає за нелінійним законом, отже при досить невеликій кількості ознак, скажімо 50, може знадобитися велике число спостережень. Ця проблема носить назву "прокляття розмірності".

Більшості реальних завдань буває достатньо кількох сотень чи тисяч спостережень. Для складних завдань може знадобитися більше, проте дуже рідко зустрічаються завдання, де потрібно менше 100 спостережень. Якщо даних мало, то мережа не має достатньої інформації для навчання, і найкраще, що можна в цьому випадку зробити, - це спробувати підігнати до даних деяку лінійну модель.

## 3.1. Процедури адаптації та навчання

Після того як визначено кількість прошарків мережі та число нейронів у кожному з них, потрібно призначити значення ваги та зсувів, які мінімізують похибку розв'язку. Це досягається за допомогою *процедур навчання*. Шляхом аналізу наявних у розпорядженні аналітика вхідних та вихідних даних, ваги та зміщення мережі автоматично налаштовуються так, щоб мінімізувати різницю між бажаним сигналом та отриманим на виході в результаті моделювання. Ця різниця називається *помилкою навчання*. Таким чином, процес навчання - це процес налаштування (підгонки) параметрів тієї моделі процесу або явища, яка реалізується нейронною мережею. Помилка навчання конкретної конфігурації нейронної мережі визначається шляхом прогону через мережу всіх наявних спостережень і порівняння вихідних значень з бажаними, цільовими значеннями. Ці різниці дозволяють сформувати так звану функцію помилок (критерій якості навчання). У якості такої функції найчастіше береться сума квадратів помилок. При моделюванні нейронних мереж із лінійними функціями активації нейронів можна побудувати алгоритм, що гарантує досягнення абсолютного мінімуму помилки навчання. Для нейронних мереж з нелінійними функціями активації загалом не можна гарантувати досягнення глобального мінімуму функції помилки.

При такому підході до процедури навчання може бути корисним геометричний аналіз поверхні помилок. Визначимо ваги та зміщення як вільні параметри моделі та їх загальне число позначимо через $N$; кожному набору таких параметрів поставимо у відповідність один вимір у функції помилки мережі. Тоді для всіляких поєднань ваг та зсувів відповідну помилку мережі можна зобразити точкою в $N$+1-мірному просторі, проте такі точки утворюють деяку поверхню, яку називають *поверхнею функції помилок*. При такому підході мета навчання нейронної мережі полягає в тому, щоб знайти на цій багатовимірній поверхні глобальний мінімум.

У разі лінійної моделі мережі та функції помилок у вигляді суми квадратів, така поверхня буде параболоїдом, який має єдиний мінімум, і це дозволяє відшукати такий мінімум досить просто.

У разі нелінійної моделі поверхня помилок має набагато складнішу будову і має ряд несприятливих властивостей, зокрема може мати локальні мінімуми, плоскі ділянки, сідлові точки і довгі вузькі яри.

Визначити глобальний мінімум багатовимірної функції аналітично неможливо, тому навчання нейронної мережі, по суті, є процедурою вивчення поверхні функції помилок. Відштовхуючись від випадково обраної точки поверхні функції помилок, алгоритм навчання поступово шукає глобальний мінімум. Як правило, для цього обчислюється градієнт (нахил) функції помилок у цій точці, а потім ця інформація використовується для просування вниз схилом. Зрештою, алгоритм зупиняється в деякому мінімумі, який може виявитися лише локальним мінімумом, а якщо пощастить, то і глобальним.

Таким чином, по суті, алгоритми навчання нейронних мереж аналогічні алгоритмам пошуку глобального екстремуму функції багатьох змінних.

Проте з урахуванням специфіки нейронних мереж їм розроблено спеціальні алгоритми навчання, серед яких слід виділити *алгоритм зворотного поширення помилки*.

При використанні алгоритму зворотного поширення помилки мережа розраховує помилку, що виникає у вихідному прошарку, і обчислює вектор градієнта як функцію ваг і зсувів. Цей вектор вказує напрямок найкоротшого спуску по поверхні цієї точки, тому якщо просунутися в цьому напрямку, то помилка

зменшиться. Послідовність таких кроків, зрештою, приведе до мінімуму того чи іншого типу. Певну трудність тут викликає вибір величини кроку.

При великій довжині кроку збіжність буде швидше, але є небезпека перестрибнути через розв'язок або піти в неправильному напрямку. Класичним прикладом такого явища під час навчання нейронної мережі є ситуація, коли алгоритм дуже повільно просувається вузьким яром з крутими схилами, перестрибуючи з одного схилу на інший. Навпаки, при малому кроці, ймовірно, буде обрано правильний напрямок, проте при цьому буде потрібно багато ітерацій. Насправді величина кроку вибирається пропорційною крутості схилу (градієнту функції помилок); такий коефіцієнт пропорційності називається параметром *швидкості налаштування* (*навчання)*. Правильний вибір параметра швидкості налаштування залежить від конкретної задачі та зазвичай здійснюється дослідним шляхом; цей параметр може залежати від часу, зменшуючись у міру виконання алгоритму.

Алгоритм діє ітеративно, і його кроки прийнято називати *епохами* або *циклами*. На кожному циклі на вхід мережі послідовно подаються всі навчальні спостереження, вихідні значення порівнюються з цільовими значеннями та обчислюється функція помилки. Значення функції помилки та її градієнта використовуються для коригування ваг і зсувів, після чого всі дії повторюються. Початкові значення ваги і зміщень мережі вибираються випадковим чином, і процес навчання припиняється або коли реалізовано певну кількість циклів, або коли помилка досягне деякого малого значення або перестане зменшуватися.

### *Явище перенавчання*

Одна з найбільш серйозних труднощів під час навчання мережі полягає в тому, що в ряді випадків ми мінімізуємо не ту помилку, яку насправді потрібно мінімізувати; потрібно мінімізувати помилку, яка з'являється в мережі, коли на неї подаються нові спостереження. Дуже важливо, щоб нейронна мережа мала здатність пристосовуватися до цих нових спостережень. Що ж відбувається насправді? Мережа навчається мінімізувати помилку на деякій обмеженій навчальній множині. Це не відповідає вимогам теорії про наявність ідеальної та нескінченно великої навчальної множини. І це не відповідає тій реальній ситуації, коли треба мінімізувати конкретну функцію помилок для наперед невідомої моделі.

Це породжує проблему, яка відома як явище перенавчання. Звернемося до завдання апроксимації деякої функції многочленом. Графіки многочленів часто мають досить хитромудрі форми, і що вищий степінь многочлена, то складніша їх форма. Якщо є деякий набір даних, то можна поставити за мету підібрати для нього апроксимуючий многочлен і, таким чином, отримати відповідну математичну модель для цього набору даних. Оскільки вихідні дані зазвичай задані з похибками, то не можна вважати, що найкраща модель задається кривою, яка проходить точно через задані точки. Многочлен низького порядку може виявитися досить грубим для апроксимації даних, в той час як многочлен високого порядку може точно дотримуватись даних, приймаючи при цьому досить хитромудру форму, яка не має жодного відношення до форми істинної залежності. Остання ситуація демонструє те, що називається явищем перенавчання.

При роботі з нейронними мережами користувач стикається з проблемою. Мережі з великою кількістю ваг дозволяють відтворювати дуже складні функції, і вони можуть бути схильні до перенавчання. Мережа з невеликою кількістю ваг може виявитися недостатньо гнучкою, щоб змоделювати наявну залежність. Наприклад, одношарова лінійна мережа здатна відтворювати лише лінійні функції. Якщо використовувати багатошарові лінійні мережі, то помилка завжди буде меншою, але це може свідчити не про хорошу якість моделі, а про те, що проявляється явище перенавчання.

Для того, щоб виявити ефект перенавчання, використовується механізм контрольної перевірки. Частина навчальних спостережень резервується як контрольні спостереження і використовується під час навчання мережі. Натомість у міру роботи алгоритму ці спостереження застосовуються для незалежного контролю результату. Спочатку помилка мережі на навчальній та контрольній множинах буде однаковою; якщо вони суттєво відрізняються, то, ймовірно, це означає, що розбиття спостережень на 2 множини не забезпечило їх однорідність. У міру навчання мережі помилка зменшується, і, поки навчання зменшує функцію помилок, помилка на контрольній множині також зменшуватиметься. Якщо ж контрольна помилка перестала зменшуватися або почала зростати, це вказує на те, що мережа почала надто близько наслідувати вихідні дані і навчання слід зупинити. У цьому випадку слід зменшити кількість нейронів або прошарків, бо мережа є надто потужною для вирішення цього завдання. Якщо ж, навпаки, мережа має недостатню потужність, щоб відтворити наявну залежність, то явище перенавчання швидше за все не спостерігатиметься і обидві помилки – навчання та контролю – не досягнуть достатньо малого рівня.

Проблеми пошуку глобального мінімуму чи вибору розміру мережі, що виникають під час роботи з нейронними мережами, приводять до того, що в практичній роботі доводиться експериментувати з великою кількістю мереж різних конфігурацій, часом навчаючи кожну з них кілька разів і порівнюючи отримані результати. Головним критерієм вибору в цих випадках є контрольна похибка. При цьому застосовується правило, згідно з яким із двох нейронних мереж з приблизно рівними контрольними похибками слід вибирати ту, що простіша.

Необхідність багаторазових експериментів веде до того, що контрольна множина починає грати ключову роль у виборі моделі нейронної мережі, тобто, стає частиною процесу навчання. Тим самим її роль як незалежного критерію якості моделі послаблюється, оскільки за великої кількості експериментів виникає ризик перенавчання нейронної мережі на контрольній множині. Для того щоб гарантувати надійність обраної моделі мережі, резервують ще одну *тестову множину спостережень*. Підсумкова модель

тестується на даних з цієї множини, щоб переконатися, що результати, досягнуті на навчальній та контрольній множинах, є реальними. Зрозуміло, що для того, щоб добре грати свою роль, тестова множина має бути використана лише 1 раз: якщо її використовувати повторно для коригування процесу навчання, то вона фактично перетвориться у контрольну множину.

Отже, процедура побудови нейронної мережі складається з наступних кроків:

- вибору початкової конфігурації мережі; наприклад, у вигляді одного прошарку з числом нейронів, рівним 1/2 загальної кількості входів та виходів;
- моделювання та навчання мережі з оцінкою контрольної помилки та використанням додаткових нейронів або проміжних прошарків;
- виявлення ефекту перенавчання та коригування конфігурації мережі.

### *Властивість узагальнення*

При описі процедури навчання нейронних мереж неявно використовувалося припущення, що навчальна, контрольна та тестова множини є репрезентативними для задачі. Зазвичай в якості навчальних беруться дані, випробувані на ряді прикладів. Якщо обставини змінилися, то закономірності, що мали місце у минулому, можуть не діяти.

Крім того, нейронна мережа може навчатися тільки на тих даних, які вона має. Припустимо, що відома навчальна множина системи стабілізації літака при польоті в спокійній атмосфері, а потрібно спроектувати систему стабілізації на основі нейронної мережі для умов польоту при сильних збуреннях. Тоді навряд чи можна очікувати від мережі правильного рішення в новій для неї ситуації.

Класичним прикладом непредставницької моделі нейронної мережі є така ситуація. При проектуванні системи машинного зору, призначеної для автоматичного розпізнавання цілей, мережа навчалася на 100 картинках, що містять зображення танків, і на 100 інших картинках, де танків не було. Після навчання мережі було досягнуто стовідсотково "правильний" результат. Але коли на вхід мережі було подано нові дані, вона безнадійно провалилася. У чому була причина? З'ясувалося, що фотографії з танками були зроблені у похмурий, дощовий день, а фотографії без танків – у сонячний день. Мережа навчилася вловлювати різницю у загальній освітленості. Щоб мережа могла результативно працювати, її слід було навчати на даних, де були б присутні всі погодні умови та типи освітлення, при яких мережа буде використовуватись, вже не говорячи про рельєф місцевості, кут та відстань зйомки, та ін.

Якщо мережа мінімізує загальну похибку, велике значення набувають пропорції, в яких представлені дані різних типів. Мережа, навчена на 900 "хороших" та 100 "поганих" спостереженнях, спотворюватиме результат на користь хороших спостережень, оскільки це дозволить алгоритму зменшити загальну похибку. Якщо в реальній ситуації "хороші" та "погані" об'єкти представлені в іншій пропорції, то результати, що видаються мережею, можуть виявитися невірними. Прикладом цього може бути завдання виявлення захворювань. Нехай, наприклад, при звичайних обстеженнях у середньому 90 % людей виявляються здоровими та мережа, таким чином, навчається на даних, у яких пропорція здорові/хворі дорівнює 90/10. Потім ця мережа застосовується для діагностики пацієнтів з певними скаргами, серед яких співвідношення здорові/хворі вже 50/50. У цьому випадку мережа ставитиме діагноз надто обережно і не розпізнаватиме захворювання у деяких хворих. Якщо ж, навпаки, мережу навчити на даних зі скаргами, а потім протестувати на звичайних даних, то вона видаватиме підвищену кількість неправильних діагнозів про наявність захворювання. У таких ситуаціях навчальні дані потрібно скоригувати так, щоб були враховані відмінності в розподілі даних (наприклад, можна повторити рідкісні спостереження або видалити найпоширеніші). Зазвичай, потрібно постаратися зробити так, щоб спостереження різних типів були представлені рівномірно, і відповідно до цього інтерпретувати результати, які видає мережа.

Здатність мережі, навченої на деякій множині даних, видавати правильні результати для досить широкого класу нових даних, зокрема і не представлених під час навчання, називається *властивістю узагальнення* нейронної мережі.

Інший підхід до процедури навчання мережі можна сформулювати, якщо розглядати її як процедуру, обернену до моделювання. У цьому випадку потрібно підібрати такі значення ваг і зсувів, які б забезпечували потрібну відповідність між входами та бажаними значеннями на виході. Така процедура навчання зветься процедурою адаптації і досить широко застосовується для налаштування параметрів нейронних мереж.

## 3.1.1. Способи адаптації та навчання

У ППП Neural Network Toolbox реалізовано 2 способи адаптації та навчання: послідовний та груповий, залежно від того, чи застосовується послідовне чи групове представлення входів.

### *Адаптація нейронних мереж*

**Статичні мережі.** Скористаємося наступною моделлю одношарової лінійної мережі з двоелементним вектором входу, значення якого знаходяться в інтервалі [–1 1], та нульовим параметром швидкості налаштування:

```
%Формування одношарової статичної лінійної мережі з двома входами
%і нульовим параметром швидкості налаштування
net = newlin([-1 1;-1 1],1, 0, 0);
```

Потрібно адаптувати параметри мережі так, щоб вона формувала лінійну залежність виду

$$t = 2p_1 + p_2.$$

*Послідовний метод.*Розглянемо випадок послідовного подання навчальної послідовності. У цьому випадку входи та цільовий вектор формуються у вигляді масиву формату cell:

```
% Масив комірок векторів входу
P = {[-1; 1] [-1/3; 1/4] [1/2; 0] [1/6; 2/3]};
T = {-1 -5/12 1 1}; % Масив комірок цільових векторів
P1 = [P{:}], T1=[T{:}] % Перехід від масиву комірок до масиву double
P1 =
-1 -0.33333 0.5 0.16667
10.250 0.66667
T1 = -1 -0.41667 1 1
```

Спочатку задамо мережу з нульовими значеннями початкових ваг і зміщень:

```
net.IW{1} = [0 0]; % Присвоєння початкових ваг
net.b{1} = 0; % Присвоєння початкового зміщення
```

У ППП NNT процедури адаптації реалізуються на основі методу adapt. Для керування процедурою адаптації використовується властивість net.adaptFcn, яка задає метод адаптації; Для статичних мереж за замовчуванням застосовується метод adaptwb, який дозволяє вибирати довільні функції для налаштування ваги та зсувів. Функції налаштування ваги та зміщень задаються властивостями net.inputWeights{i, j}.learnFcn, net.layerWeights{i, j}.learnFcn та net.biases{i, j}.learnFcn.

***Зауваження. У багатьох версіях MATLAB метод adapt уже не працює, а користовачу пропонують використовувати лише метод train. Це буде видно по тому, чи вдалося описаній мережі адаптуватись під виконання поставленої задачі. Якщо у вас метод adapt не працює, то переробіть цю ж задачу із функцією train.***

Виконаємо 1 цикл адаптації мережі з нульовим параметром швидкості налаштування:

```
% Послідовна адаптація мережі з входами P та цілями T
[net1, a, e] = adapt (net, P, T);
% net1-нова мережа, a-вихід, e-помилка навчання
```

У цьому випадку ваги не модифікуються, виходи мережі залишаються нульовими, оскільки параметр швидкості налаштування дорівнює нулю та адаптації мережі не відбувається. Похибки збігаються зі значеннями цільової послідовності

```
net1.IW{1, 1}, a, e
ans = 0 0
a = [0] [0] [0] [0]
e = [-1] [-0.41667] [1] [1]
```

Задамо значення параметрів швидкості налаштування та ваги входу та зміщення:

```
net.IW{1} = [0 0]; % Присвоєння початкових ваг
net.b{1} = 0; % Присвоєння початкового зміщення
net.inputWeights{1,1}.learnParam.lr = 0.2;
net.biases{1,1}.learnParam.lr = 0;
```

Нульове значення параметра швидкості налаштування для зміщення обумовлено тим, що залежність, що виявляється, не має постійної складової.

Виконаємо 1 цикл налаштування:

```
[net1, a, e] = adapt (net, P, T);
net1.IW{1, 1}, a, e
ans = 0.34539 -0.069422
a = [0] [-0.11667] [0.11] [-0.091833]
e = [-1] [-0.3] [0.89] [1.0918]
```

Тепер виконаємо послідовну адаптацію мережі протягом 30 циклів:

```
% Послідовна адаптація мережі з входами P та цілями T за 30 циклів
net = newlin([-1 1;-1 1],1, 0, 0);
net.IW{1} = [0 0]; % Присвоєння початкових ваг
net.b{1} = 0; % Присвоєння початкового зміщення
```

Задамо значення параметрів швидкості налаштування для ваг входу та зміщення:

```
net.inputWeights{1,1}.learnParam.lr = 0.2;
net.biases{1,1}.learnParam.lr = 0;
P = {[-1; 1] [-1/3; 1/4] [1/2; 0] [1/6; 2/3]}; % Масив векторів входу
T = {-1 -5/12 1 1}; % Масив цільових векторів
for i=1:30,
[net, a {i}, e {i}] = adapt (net, P, T);
W(i,:)=net.IW{1,1};
```

```
end
mse(cell2mat(e{30})) % Середньоквадратична помилка адаптації
ans = 0.0017176
W(30,:) % Ваги після 30 циклів
ans = 1.9199 0.925
cell2mat(a{30})
ans = -0.9944 -0.40855 0.95663 0.93005
cell2mat(e{30})
ans = -0.0055975 -0.0081125 0.043367 0.069947
```

Побудуємо графіки залежності значень виходів мережі та вагових коефіцієнтів залежно від кількості ітерацій (рис. 19):

```
subplot(3,1,1)
for i=1:30,
a{i}=(cell2mat(a{i}))';
end
a=cell2mat(a);
a=cat(2,[0;0;0;0],a);
plot(0:30,a,'k') % Рис 19,а
xlabel(''), ylabel('Виходи a(i)'),grid
subplot(3,1,2)
plot(0:30,[[0 0]; W],'k') % Рис 19,б
xlabel(''), ylabel('Вага входів w(i)'),grid
subplot(3,1,3)
for i=1:30, E(i) = mse(e{i}); end
semilogy(1:30, E,'+k') % Мал. 19,в
xlabel('Цикли'), ylabel('Помилка'),grid
```

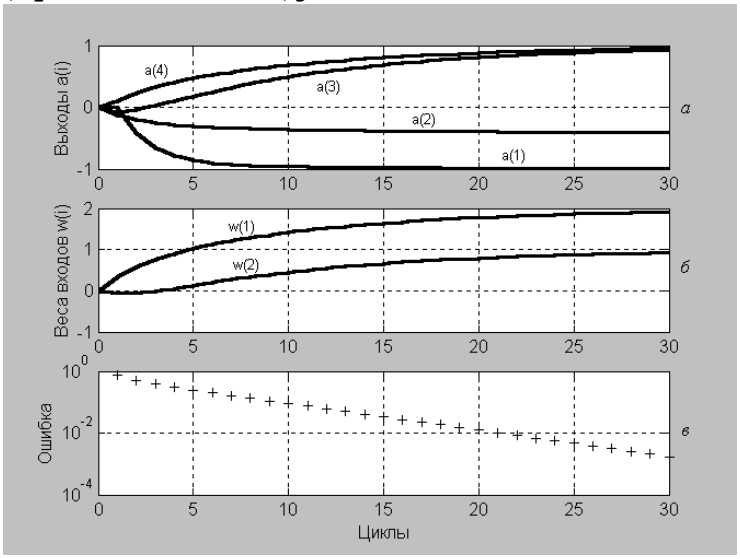



Рис. 19

Перший вихід такий же, як і при нульовому значенні параметра швидкості налаштування, оскільки до подання мережі першого входу ніяких модифікацій не відбувається. Другий вихід відрізняється, оскільки параметри мережі модифіковані. Ваги продовжують змінюватись при подачі нового входу. Якщо мережа відповідає завданню, коректно задані навчальні послідовності, початкові умови та параметр швидкості налаштування, то зрештою похибка може бути зведена до нуля.

У цьому всьому можна переконатися, вивчаючи процес адаптації, показаний на рис. 19. Умова закінчення адаптації визначається похибкою наближення до цільового вектора; в даному випадку мірою такої похибки є середньоквадратична помилка mse(e{i}), яка має бути меншою за 0.015.

На рис. 19 *а* показані виходи нейронів у процесі адаптації мережі, на рис. 19 *б* – коефіцієнти відновлюваної залежності, які відповідають елементам вектора ваг входу, а на рис. 19, *в* – похибка навчання. Як випливає з аналізу графіків, за 12 кроків отримано похибку навчання $1.489e^{-3}$.

Пропоную студентові самостійно переконатися, що для досліджуваної залежності навчальні послідовності виду

```
P = {[-1; 1] [-1/2; 1/2] [1/2; -1/2] [1; -1]}; % Масив векторів входу
T = {-1 -1/2 1/2 1}; % Масив векторів мети
```

є репрезентативними.

*Груповий метод.* Розглянемо випадок групового подання навчальної послідовності. У цьому випадку входи та цільовий вектор формуються у вигляді масиву формату double.

```
P = [-1 -1/3 1/2 1/6; 1 1/4 0 2/3];
```

```
T = [-1 -5/12 1 1];
```

Використовується та сама модель статичної мережі з тими самими вимогами до похибки адаптації (навчання). При зверненні до М-функції adapt за замовчуванням викликаються функції adaptwb та learnwh; остання виконує налаштування параметрів мережі на основі алгоритму WH, що реалізує правило Уідроу – Хоффа (Widrow – Hoff).

Основний цикл адаптації мережі із заданою похибкою виглядає так:

```
% Груповий спосіб адаптації мережі з входами P та цілями T
net3 = newlin([-1 1;-1 1],1, 0, 0.2);
net3.IW {1} = [0 0]; % Присвоєння початкових ваг
net3.b{1} = 0; % Присвоєння початкового зміщення
net3.inputWeights{1,1}.learnParam.lr = 0.2;
P = [-1 -1/3 1/2 1/6; 1 1/4 0 2/3];
T = [-1 -5/12 1 1];
EE = 10; i=1;
while EE > 0.0017176
[net3, a {i}, e {i}, pf] = adapt (net3, P, T);
W(i,:) = net3.IW{1,1};
EE = mse (e {i});
ee (i) = EE;
i = i +1;
end
```

Результатом адаптації при заданій похибці є наступні значення коефіцієнтів лінійної залежності, значення виходів нейронної мережі (що наближаються до бажаного значення виходу), а також середньоквадратична похибка адаптації:

```
W(63,:)
ans = 1.9114 0.84766
cell2mat(a(63))
ans = -1.003 -0.36242 1.0172 0.94256
EE = mse (e {63})
EE = 0.0016368
mse(e{1})
ans = 0.7934
```

Процедура адаптації виходів та параметрів нейронної мережі проілюстрована на рис. 20.

```
subplot(3,1,1)
plot(0:63,[zeros(1,4); cell2mat(a')],'k') % Рис.20,а
xlabel(''), ylabel('Виходи a(i)'),grid
subplot(3,1,2)
plot(0:63,[[0 0]; W],'k') % Рис.20,б
xlabel(''), ylabel('Вага входів w(i)'),grid
subplot(3,1,3)
semilogy(1:63, ee,'+k') % Рис.20,в
xlabel('Цикли'), ylabel('Помилка'),grid
```

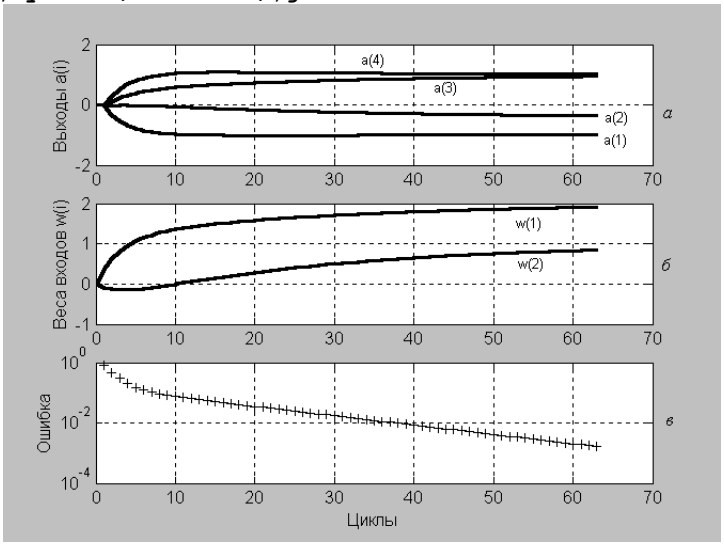



Рис. 20

Як випливає з аналізу графіків, для досягнення необхідної точності адаптації потрібно 12 кроків. Порівнюючи рис. 20 та 19, можна переконатися, що існує відмінність у динаміці процедур адаптації при послідовному та груповому поданні даних.

**Динамічні мережі.** Ці мережі характеризуються наявністю ліній затримки, і для них послідовне подання входів є найбільш природним.

*Послідовний метод.* Звернемося до лінійної моделі нейронної мережі з одним входом та одним елементом затримки. Встановимо початкові умови на лінії затримки, а також ваги та зміщення рівними 0, а параметр швидкості налаштування рівним 0.5:

```
net = newlin([-1 1], 1, [0 1], 0.5);
Pi = {0}; % Початкова умова для елемента затримки
net.IW{1} = [0 0]; % Значення ваг
net.biasConnect = 0; % Значення зміщення
```

Щоб застосувати послідовний спосіб адаптації, представимо входи та цілі як масиви комірок:

```
P = {-1/2 1/3 1/5 1/4}; % Вектор входу
T = {-1 1/6 11/15 7/10}; % Вектор цілі
```

Спробуємо пристосувати мережу на формування потрібного виходу з урахуванням наступного співвідношення:

$$y(t) = 2p(t) + p(t-1).$$

Використовуємо для цієї мети М-функцію adapt та основний цикл адаптації мережі із заданою похибкою, як це вже було описано вище:

```
EE = 10; i = 1;
while EE > 0.0001
[net, a {i}, e {i}, pf] = adapt (net, P, T);
W(i,:)=net.IW{1,1};
EE = mse (e {i});
ee(i) = EE;
i = i +1;
end
```

Мережа адаптувалася за 22 цикли. Результатом адаптації при заданій похибці є наступні значення коефіцієнтів лінійної залежності, значень виходів нейронної мережі (що наближаються до бажаного значення виходу), а також середньоквадратична похибка адаптації:

```
W(22,:)
ans = 1.983 0.98219
a{22}
ans = [-0.98955] [0.17136] [0.72272] [0.69177]
EE
EE = 7.7874e-005
```

Побудуємо графіки залежності виходів системи та вагових коефіцієнтів від числа циклів навчання (рис. 21):

```
subplot(3,1,1)
plot(0:22,[zeros(1,4); cell2mat(cell2mat(a'))],'k') % Рис.21,а
xlabel(''), ylabel('Виходи a(i)'),grid
subplot(3,1,2)
plot(0:22,[[0 0]; W],'k') % Рис.21,б
xlabel(''), ylabel('Вага входів w(i)'),grid
subplot(3,1,3)
semilogy(1:22,ee,'+k') % Рис.21,в
xlabel('Цикли'), ylabel('Помилка'),grid
```

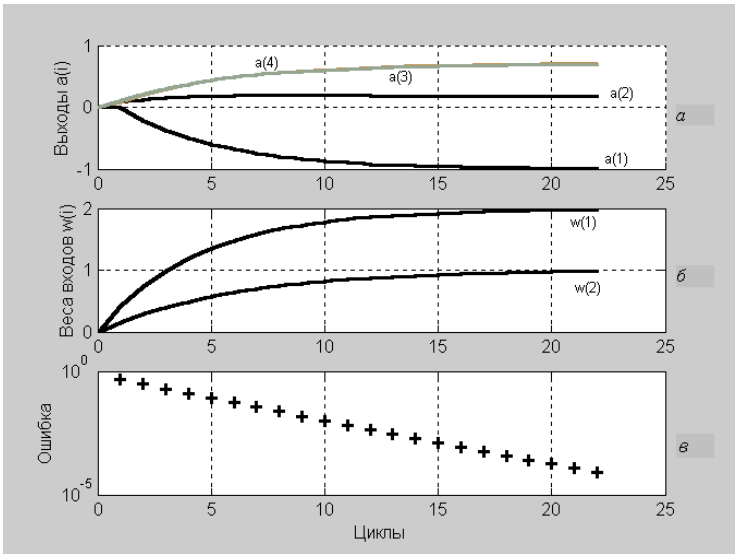



Рис. 21

На рис. 21 *а* показані виходи нейронів у процесі адаптації мережі, а на рис. 21 *б* – коефіцієнти відновлюваної залежності, які відповідають елементам вектора ваг входу.

*Груповий спосіб* подання навчальної множини для адаптації динамічних мереж не застосовується.

### *Навчання нейронних мереж*

**Статичні мережі.**Скористайтеся розглянутою вище моделлю одношарової лінійної мережі з двоелементним вектором входу, значення якого знаходяться в інтервалі [–1 1], та нульовим параметром швидкості налаштування, як це було для випадку адаптації:

% Формування одношарової статичної лінійної мережі з двома входами

```
%і нульовим параметром швидкості налаштування
net = newlin([-1 1;-1 1],1, 0, 0);
net.IW{1} = [0 0]; % Значення ваг
net.b{1} = 0; % Значення зміщень
```

Потрібно навчити параметри мережі так, щоб вона формувала лінійну залежність виду

$$t = 2p_1 + p_2.$$

*Послідовний спосіб*. Для цього представимо навчальну послідовність у вигляді масивів комірок

```
P = {[-1; 1] [-1/3; 1/4] [1/2; 0] [1/6; 2/3]}; % Масив векторів входу
T = {-1 -5/12 1 1}; % Масив векторів мети
```

Тепер все готове до навчання мережі. Навчатимемо її за допомогою функції train протягом 30 циклів.

У цьому випадку для навчання та налаштування параметрів мережі використовуються функції trainwb і learnwhy відповідно.

```
% Параметр швидкості налаштування ваг
net.inputWeights{1,1}.learnParam.lr = 0.2;
net.biases{1}.learnParam.lr = 0; % Параметр швидкості налаштування зсувів
net.trainParam.epochs = 30; % Число циклів навчання
net1 = train (net, P, T);
```

Параметри мережі після навчання дорівнюють наступним значенням:

```
W = net1.IW {1}
W = 1.9214 0.92599
y = sim(net1, P)
y = [-0.99537] [-0.40896] [0.96068] [0.93755]
EE = mse([y{:}]-[T{:}])
EE = 1.3817e-003
```

Залежність величини помилки навчання від числа циклів навчання наведено на рис. 22.

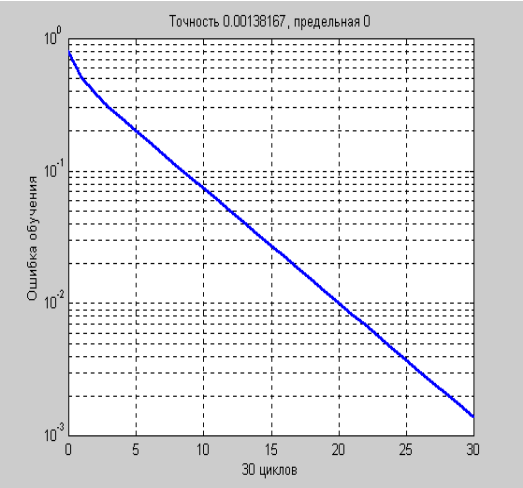



Рис. 22

Це той же результат, що був отриманий для групового способу адаптації з використанням функції adapt.

*Груповий спосіб.* Для цього представимо навчальну послідовність у вигляді масивів формату double array:

```
P = [-1 -1/3 1/2 1/6; 1 1/4 0 2/3];
T = [-1 -5/12 1 1];
net1 = train (net, P, T);
TRAINWB, Epoch 0/10, MSE 0.793403/0.
TRAINWB, Epoch 10/10, MSE 0.00243342/0.
TRAINWB, Maximum epoch reached.
```

Параметри мережі після навчання дорівнюють наступним значенням:

```
W = net1.IW {1}
W = 1.9214 0.92599
y = sim(net1, P)
y = -0.99537 -0.40896 0.96068 0.93755
EE = mse(y-T)
EE = 1.3817e-003
```

Цей результат повністю збігається з результатом послідовного навчання цієї мережі.

**Динамічні мережі.** Навчання динамічних мереж виконується аналогічно з використанням методу train.

*Послідовний метод.* Звернемося до лінійної моделі нейронної мережі з одним входом та одним елементом затримки.

Встановимо початкові умови для елемента затримки, ваги та зміщення рівними 0, а параметр швидкості налаштування рівним 0.5:

```
net = newlin([-1 1], 1, [0 1], 0.5);
Pi = {0}; % Початкова умова для елемента затримки
net.IW{1} = [0 0]; % Значення ваг
net.biasConnect = 0; % Значення зміщення
net.trainParam.epochs = 22;
```

Щоб застосувати послідовний спосіб навчання, представимо входи та цілі як масиви комірок:

```
P = {-1/2 1/3 1/5 1/4}; % Вектор входу
```

Навчимо мережу формувати потрібний вихід на основі співвідношення $y(t) = 2p(t) + p(t–1)$, тоді

```
T = {-1 1/6 11/15 7/10}; % Вектор цілі
```

Використовуємо для цієї мети М-функцію train:

```
net1 = train(net, P, T, Pi);
```

Параметри мережі після навчання дорівнюють наступним значенням:

```
W = net1.IW {1}
W = 1.9883 0.98414
y = sim(net1, P)
y = [-0.99414] [0.17069] [0.7257] [0.6939]
EE = mse([y{:}]-[T{:}])
EE = 3.6514e-005
```

Графік залежності помилки навчання від числа циклів наведено на рис. 23.

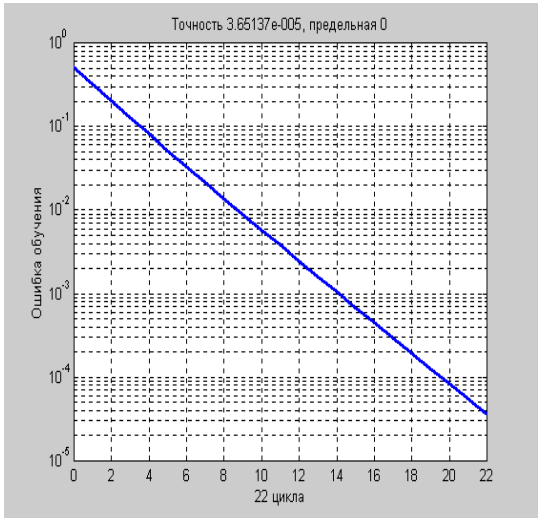



Рис. 23

Пропоную студентові самостійно виконати порівняння результатів навчання (train) з результатами адаптації (adapt) цієї мережі.

*Груповий спосіб* подання навчальної послідовності на навчання динамічних мереж не застосовується.

# 4. ПЕРСЕПТРОНИ

З цього розділу починається систематичний аналіз конкретних нейронних мереж, і першою є нейронна мережа, яку називають *персептроном*. Винайдена і названа так Розенблатом. Це одношарова нейронна мережа.

За командою help percept можна отримати наступну інформацію про М-функції, що входять до складу ППП Neural Network Toolbox і які стосуються побудови нейронних мереж на основі персептронів:

| *Perceptrons* | *Персептрони* |
|---|---|
| **New networks** | **Формування нейронної мережі** |
| newp | Створення персептрону |
| **Using networks** | **Робота з нейронною мережею** |
| sim | Моделювання мережі |
| init | Ініціалізація мережі |
| adapt | Адаптація мережі |
| train | Навчання мережі |
| **Weight functions** | **Функції зважування** |
| dotprod | Скалярний добуток |
| **Net input functions** | **Функції накопичення** |
| netsum | Сума зважених входів |
| **Transfer функцій** | **Функції активації** |
| hardlim | Порогова функція з жорсткими обмеженнями |
| hardlims | Симетрична порогова функція з жорсткими обмеженнями |
| **Initialization функцій** | **Функції ініціалізації** |
| initlay | Ініціалізація прошарків |
| initwb | Ініціалізація ваг та зміщень |
| initzero | Ініціалізація нульових ваг та зміщень |
| **Performance functions** | **Функції оцінки якості мережі** |
| mae | Середня абсолютна похибка |
| **Learning functions** | **Функції налаштування параметрів персептрону** |
| learnp | Абсолютна функція налаштування |
| learnpn | Нормована функція налаштування |
| **Adapt functions** | **Функції адаптації** |
| adaptwb | Адаптація ваг та зміщень |
| **Training functions** | **Функції навчання** |
| trainwb | Правило навчання ваг та зміщень |
| **Demonstrations** | **Демонстраційні приклади** |
| demop1 | Класифікація з використанням персептрону з двома входами |
| demop2 | Класифікація з використанням персептрону з трьома входами |
| demop3 | Класифікація з використанням персептрону з двома нейронами |
| demop4 | Формування вхідних векторів зовнішнього прошарку |
| demop5 | Навчання з використанням нормованої функції налаштування |
| demop6 | Приклад лінійно нероздільних векторів |
| demop7 | Класифікація з використанням двошарового персептрону |

Слід звернути увагу, що у версії ППП Neural Network Toolbox Version 3.0.1 (R11) представлені лише демонстраційні приклади demop1, demop4, demop5, demop6.

## 4.1. Архітектура персептрону

**Нейрон персептрону.** Нейрон, що використовується в моделі персептрона, має порогову функцію активації hardlim із жорсткими обмеженнями (рис. 24).

Кожен елемент вектора входу персептрона множиться на відповідну вагу $w_{1j}$, а сума цих добутків є входом функції активації. Нейрон персептрона повертає число 1, якщо вхід функції активації $n \geq 0$ і число 0, якщо $n < 0$.

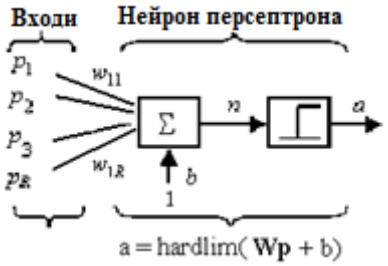
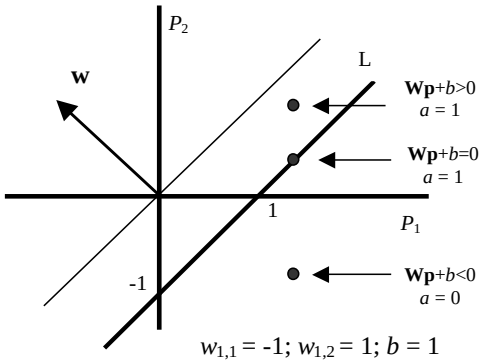



Рис. 24

Функція активації з жорсткими обмеженнями дає персептрону здатність класифікувати вектори входу, розділяючи простір входів на 2 області, як це показано на рис. 25 для персептрона з двома входами та зміщенням.




$w_{1,1} = -1$; $w_{1,2} = 1$; $b = 1$

Рис. 25

Простір входів ділиться на 2 області розділяючою (відокремлюючою) прямою L, яка для двовимірного випадку задається рівнянням

$$\mathbf{w}^{\mathrm{T}}\mathbf{p} + b = 0.$$

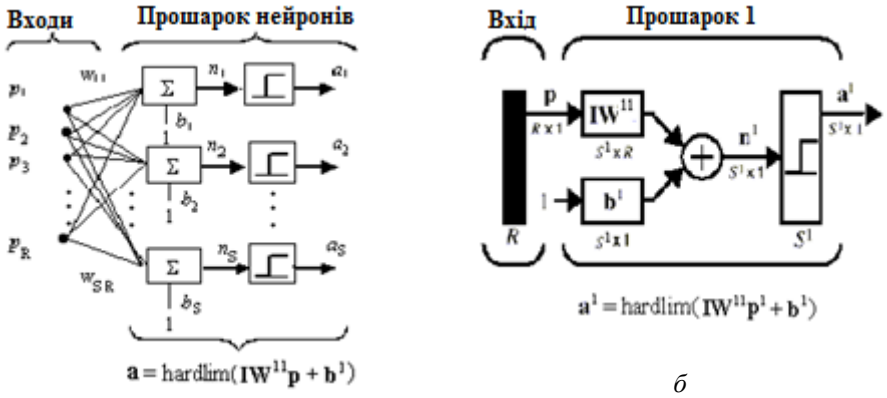
Ця пряма перпендикулярна до вектора ваг $w$ і зміщена на величину $b$. Вектори входу вище прямої L відповідають позитивному потенціалу нейрона, і, отже, вихід персептрону для цих векторів дорівнюватиме 1; вектори входу нижче лінії L відповідають виходу персептрона, що дорівнює 0. При зміні значень зміщення та ваги межа прямої L змінює своє положення. Персептрон без зміщення завжди формує пряму, що проходить через початок координат; додавання зміщення формує пряму, яка не проходить через початок координат, як це показано на рис. 25. У разі коли розмірність вектора входу перевищує 2, розділяючим елементом буде гіперплощина.

Демонстраційна програма nnd4db наочно ілюструє переміщення лінії розділення при вирішенні задачі класифікації векторів входу.

**Архітектура мережі.** Персептрон складається з єдиного прошарку, що включає S нейронів, як це показано на рис. 26, *а* і *б* у вигляді відповідно розгорнутої та укрупненої структурних схем; ваги $w_{ij}$ – це коефіцієнти передачі від j-го входу до i-го нейрону.

Рівняння одношарового персептрону має вигляд:

$$\mathbf{a} = \mathbf{f}\,(\mathbf{Wp} + \mathbf{b})\ \mathbf{.}$$




*а* *б*

Рис. 26

## 4.2. Модель персептрону

Для формування моделі одношарового персептрону призначено функцію **newp**

```
net = newp(PR, S)
```

з наступними вхідними аргументами: PR – масив мінімальних та максимальних значень для R елементів входу розміру R×2; S – число нейронів у прошарку.

Як функція активації персептрона за замовчуванням використовується функція hardlim.

*Приклад:*

Функція

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

створює персептрон з одноелементним входом та одним нейроном; діапазон значень входу – [0 2].

Визначимо деякі параметри персептрону, які ініціалізуються за замовчуванням.

*Ваги входів:*

```
inputweights = net.inputweights{1,1}
inputweights =
delays: 0
initFcn: 'initzero'
learn: 1
learnFcn: 'learnp'
learnParam: []
size: [1 1]
userdata: [1x1 struct]
weightFcn: 'dotprod'
```

Зауважимо, що функція налаштування персептрона за замовчуванням learnp; вхід функції активації обчислюється за допомогою функції скалярного добутку dotprod; Функція ініціалізації initzero використовується для встановлення початкових нульових ваг.

*Зміщення:*

```
biases = net.biases {1}
biases =
initFcn: 'initzero'
learn: 1
learnFcn: 'learnp'
learnParam: []
size: 1
userdata: [1×1 struct]
```

Неважко побачити, що початкове зміщення також встановлено 0.

## *Моделювання персептрону*

Розглянемо одношаровий персептрон з одним двоелементним вектором входу, значення елементів якого змінюються в діапазоні від -2 до 2:

```
net = newp([-2 2; -2 2],1); % Створення персептрона net
```

За замовчуванням ваги та зміщення дорівнюють 0, і для того, щоб встановити бажані значення, необхідно застосувати наступні оператори:

```
net.IW {1,1} = [-1 1];
net.b{1} = [1];
```

У цьому випадку розділяюча пряма має вигляд:

$$L: -p_1 + p_2 + 1 = 0.$$

Це відповідає рис. 24.

Структурна схема моделі персептрону показана на рис. 27

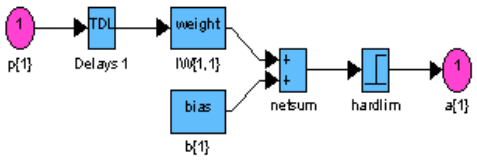



Рис. 27

Тепер визначимо, що видає мережа на вхідні вектори $p_1$ і $p_2$, розташовані по різні боки від розділяючої прямої:

```
p1 = [1; 1];
a1 = sim(net,p1) % Моделювання мережі net із вхідним вектором p1
a1 = 1
p2 = [1; -1];
a2 = sim(net,p2) % Моделювання мережі net із вхідним вектором p2
a2 = 0
```

Персептрон правильно класифікував ці 2 вектори.

Зауважимо, що можна було б ввести послідовність двох векторів у вигляді масиву комірок і отримати результат також у вигляді масиву комірок:

```
% Послідовність двох векторів у вигляді масиву комірок
p3 = {[1; 1] [1; -1]};
a3 = sim(net,p3) % Моделювання мережі net при вхідному сигналі p3
a3 = [1] [0]
```

### *Ініціалізація параметрів*

Для одношарового персептрона як параметри нейронної мережі в загальному випадку виступають ваги входів і зсуви. Припустимо, що створюється персептрон із двоелементним вектором входу та одним нейроном:

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

Запросимо характеристики ваг входу:

```
net.inputweights{1, 1}
ans =
delays: 0
initFcn: 'initzero'
learn: 1
learnFcn: 'learnp'
learnParam: [ ]
size: [1 2]
userdata: [1×1 struct]
weightFcn: 'dotprod'
```

З цього переліку випливає, що як функція ініціалізації за замовчуванням використовується функція initzero, яка присвоює вагам входу нульові значення.

У цьому можна переконатися, якщо отримати значення елементів матриці ваг і зміщення:

```
wts = net.IW {1,1}
wts = 0 0
bias = net.b {1}
bias = 0
```

Тепер перевстановимо значення елементів матриці ваги та зміщення:

```
net.IW{1,1} = [3, 4]
net.b{1} = 5
wts = 3 4
bias = 5
```

Для того, щоб повернутися до початкових параметрів персептрона, скористаємось функцією init:

```
net = init (net);
wts
wts = 0 0
bias
bias = 0
```

Можна змінити спосіб, яким ініціалізується персептрон за допомогою функції init. Для цього достатньо змінити тип функцій ініціалізації, які застосовуються для встановлення початкових значень ваг входів і зсувів. Наприклад, скористаємось функцією ініціалізації rands, яка встановлює випадкові значення параметрів персептрона.

```
% Задати функції ініціалізації ваг та зміщень
net.inputweights{1,1}.initFcn = 'rands';
net.biases{1}.initFcn = 'rands';
% Виконати ініціалізацію раніше створеної мережі з новими функціями
net = init (net);
wts = net.IW {1,1}
wts = -0.96299 0.64281
bias = net.b {1}
bias = -0.087065
```

Видно, що ваги та зміщення обрані випадковим чином.

## 4.3. Процедури налаштування параметрів

Визначимо процес навчання персептрона як процедуру налаштування ваг та зміщень з метою зменшити різницю між бажаним (цільовим) та істинним сигналами на його виході, використовуючи деяке правило налаштування (навчання). Процедури навчання діляться на 2 класи: навчання з учителем та навчання без учителя.

При навчанні з учителем задається множина прикладів для необхідної поведінки мережі, яка називається навчальною множиною

$$\{p_1, t_1\}, \{p_2, t_2\}, ..., \{p_Q, t_Q\}.$$

Тут $p_1$, $p_2$, …, $p_Q$ – входи персептрону, а $t_1$, $t_2$, …, $t_Q$ – необхідні (цільові) виходи.

При подачі входів виходи персептрону порівнюються із цілями. Правило навчання використовується для налаштування ваг та зміщень персептрона так, щоб наблизити значення виходу до цільового значення. Алгоритми, які використовують такі правила навчання, називають алгоритмами навчання з учителем. Для їх успішної реалізації необхідні експерти, які повинні попередньо сформувати навчальну множину. Розробка таких алгоритмів сприймається як перший крок у створенні систем штучного інтелекту.

У зв'язку з цим вчені не припиняють суперечки на тему, чи можна вважати алгоритми навчання з учителем природними чи властивими природі, або вони створені штучні. Наприклад, навчання людського мозку на перший погляд відбувається без вчителя: на зорові, слухові, тактильні та інші рецептори надходить інформація ззовні і всередині мозку відбувається самоорганізація. Однак не можна заперечувати і того, що в житті людини чимало вчителів – і в буквальному, і в переносному значенні – які координують реакції на зовнішні впливи. Проте, як би не розвивалася суперечка прибічників цих двох концепцій навчання, здається, що вони мають право на існування. І розглянуте нами правило навчання персептрона належить до правила навчання з учителем.

Під час навчання без вчителя ваги та зміщення змінюються лише у зв'язку із змінами входів мережі. У цьому випадку цільові виходи у явному вигляді не задаються. Головна риса, яка робить навчання без вчителя привабливим, – це його самоорганізація, яка зумовлена, як правило, використанням зворотних зв'язків. Що стосується процесу налаштування параметрів мережі, то він організується з використанням тих самих процедур. Більшість алгоритмів навчання без вчителя застосовується під час вирішення завдань кластеризації даних, коли необхідно розділити входи на скінченне число класів.

Що стосується персептронів, які розглядаються в цьому розділі, то хотілося б сподіватися, що в результаті навчання може бути побудована така мережа, яка забезпечить правильне рішення, коли на вхід буде поданий сигнал, який відрізняється від тих, що використовувалися в процесі навчання.

### *Правила налаштування*

Налаштування параметрів (навчання) персептрона здійснюється з використанням навчальної множини. Позначимо через *p* вектор входів персептрону, а через *t* вектор відповідних бажаних виходів. Мета навчання – зменшити похибку $e = a - t$, яка дорівнює різниці між реакцією нейрона *a* та цільовим вектором *t*.

Правило налаштування (навчання) персептрона повинне залежати від величини похибки *e*. Цільовий вектор t може включати лише значення 0 і 1, оскільки персептрон з функцією активації hardlim може генерувати тільки такі значення.

При налаштуванні параметрів персептрона без зміщення та з єдиним нейроном можливі лише 3 ситуації:

1. Для даного вектора входу вихід персептрона правильний (a = t і e = t - a = 0) і тоді вектор ваг w не змінюється.
2. Вихід персептрону дорівнює 0, а має дорівнювати 1 (a = 0, t = 1 і e = t – 0 = 1). У цьому випадку вхід функції активації $w^T p$ є від'мним і його необхідно скоригувати. Додамо до вектора ваг *w* вектор входу *p*, і тоді добуток $(w^T + p^T)\, p = = w^T p + p^T p$ зміниться на додаттну величину, а після кількох таких кроків вхід функції активації стане додатним і вектор входу класифікується правильно. При цьому зміняться налаштування ваг.
3. Вихід нейрона дорівнює 1, а повинен дорівнювати 0 (a = 0, t = 1 і e = t - a = -1). У цьому випадку вхід функції активації $w^T p$ є додатним і його необхідно скоригувати. Віднімемо від вектора ваг *w* вектор входу *p*, і тоді добуток $(w^T - p^T)\, p = = w^T p - p^T p$ зміниться на від''vуе величину, а після кількох кроків вхід функції активації стане від'ємним і вектор входу класифікується правильно. При цьому зміняться налаштування ваг.

Тепер правило налаштування (навчання) персептрона можна записати, зв'язавши зміну вектора ваг$\Delta\mathbf{w}$ з похибкою e = t – a:

$$\Delta\mathbf{w} = \begin{cases} 0, & якщо\ e = 0; \\ \mathbf{p}, & якщо\ e = 1; \\ -\mathbf{p}, & якщо\ e = -1; \end{cases}$$

Усі 3 випадки можна описати одним співвідношенням:

$$\Delta\mathbf{w} = (t - a)\mathbf{p} = e\mathbf{p}\,.$$

Можна отримати аналогічний вираз зміни зміщення, враховуючи, що зміщення можна розглядати як вагу для одиничного входу:

$$\Delta b = (t - a)\,1 = e.$$

У разі кількох нейронів ці співвідношення узагальнюються так:

$$\begin{cases} \Delta\mathbf{W} = (\mathbf{t} - \mathbf{a})\mathbf{p}^{\mathbf{T}} = \mathbf{e}\mathbf{p}^{\mathbf{T}}; \\ \Delta\mathbf{b} = (\mathbf{t} - \mathbf{a}) = \mathbf{e}. \end{cases}$$

Тоді правило налаштування (навчання) персептрона можна записати в такій формі:

$$\begin{cases} \mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^{\mathbf{T}}; \\ \mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e}. \end{cases}$$

Описані співвідношення покладено в основу алгоритму налаштування параметрів персептрона, який реалізований у ППП Neural Network Toolbox у вигляді М-функції **learnp**. Щоразу під час виконання функції learnp відбуватиметься переналаштування параметрів персептрона. Доведено, що якщо розв'язок існує, то процес навчання персептрона збігається за скінченне число ітерацій. Якщо зміщення не використовується, функція learnp шукає розв'язок, змінюючи лише вектор ваг w. Це приводить до знаходження розділяючої прямої, перпендикулярної вектору w, яка належним чином розділяє вектори входу.

Розглянемо простий приклад персептрона з єдиним нейроном та двоелементним вектором входу:

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

Визначимо зміщення b рівним 0, а вектор ваг w рівним [1 –0.8]:

```
net.b{1} = 0;
w = [1 -0.8];
net.IW{1,1} = w;
```

Навчальну множину задамо так:

```
p = [1; 2];
t = [1];
```

Моделюючи персептрон, розрахуємо вихід та помилку на першому кроці налаштування (навчання):

```
a = sim (net, p)
a = 0
e = t-a
e = 1
```

Нарешті, використовуючи М-функцію налаштування параметрів learnp, знайдемо потрібну зміну ваг:

```
dw = learnp(w,p,[ ],[ ],[ ],[ ],e,[ ],[ ],[ ])
dw = 1 2
```

Тоді новий вектор ваг набуде вигляду:

```
w = w + dw
w = 2.0000 1.2000
```

Зауважимо, що описані вище правило та алгоритм налаштування (навчання) персептрону гарантують збіжність за скінченне число кроків для всіх завдань, які можуть бути вирішені з використанням персептрону. Це насамперед завдання класифікації векторів, які відносяться до класу лінійно відокремлених, коли весь простір входів можна розділити на 2 області деякою прямою лінією, у багатовимірному випадку – гіперплощиною.

Демонстраційний приклад nnd4pr дозволяє виконати численні експерименти з налаштування (навчання) персептрона для вирішення задач класифікації вхідних векторів.

### *Процедура адаптації*

Багаторазово використовуючи М-функції sim і learnp для зміни ваг і зміщення персептрона, можна зрештою побудувати розділяючу лінію, яка вирішить задачу класифікації, за умови, що персептрон узагалі може її вирішувати. Кожна реалізація процесу налаштування з використанням всієї навчальної множини називається проходом або циклом. Такий цикл може бути виконаний за допомогою спеціальної адаптаційної функції adapt. При кожному проході функція adapt використовує навчальну множину, обчислює вихід, похибку та виконує підстроювання параметрів персептрона.

Зауважимо, що процедура адаптації не гарантує, що синтезована мережа виконає класифікацію нового вектора входу. Можливо, знадобиться нове налаштування матриці ваг W та вектора зсувів b з використанням функції adapt.

Щоб пояснити процедуру адаптації, розглянемо найпростіший приклад. Виберемо персептрон з одним нейроном та двоелементним вектором входу (рис. 4.5).

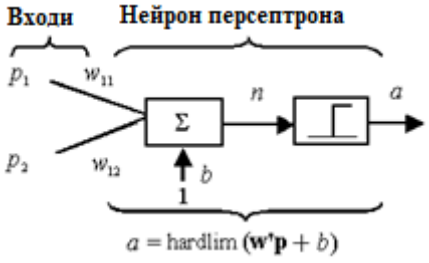



Рис. 28

Ця мережа та завдання, яке ми збираємося розглядати, досить прості, тому можна всі розрахунки виконати вручну.

Припустимо, що потрібно за допомогою персептрона розв'язати задачу класифікації векторів, якщо задано таку навчальну множину:

$$\left\{\mathbf{p}_1=\begin{bmatrix}2\\2\end{bmatrix},t_1=0\right\}\quad\left\{\mathbf{p}_2=\begin{bmatrix}1\\-2\end{bmatrix},t_2=1\right\}\quad\left\{\mathbf{p}_3=\begin{bmatrix}-2\\2\end{bmatrix},t_3=0\right\}\quad\left\{\mathbf{p}_4=\begin{bmatrix}-1\\1\end{bmatrix},t_4=1\right\}.$$

Використовуємо нульові початкові ваги та зміщення. Для позначення змінних на кожному кроці використовуємо індекс у круглих дужках. Таким чином, початкові значення вектора ваг $w^T(0)$ та зміщення b(0) рівні відповідно $w^T(0) = [0\ 0]$ та b(0) = 0.

Обчислимо вихід персептрона для першого вектора входу $p_1$, використовуючи початкові ваги та зсув:

$$a=\text{hardlim}\left(\mathbf{w}^{\mathrm{T}}(0)\mathbf{p}_1+b(0)\right)=\text{hardlim}\left(\begin{bmatrix}0&0\end{bmatrix}\begin{bmatrix}2\\2\end{bmatrix}+0\right)=\text{hardlim}(0)=1.$$

Вихід не збігається з цільовим значенням t1 і необхідно застосувати правило налаштування (навчання) персептрона, щоб обчислити необхідні зміни ваг і зсувів:

$$\begin{cases}e=t_1-a=0-1=-1;\\ \Delta\mathbf{w}^{\mathrm{T}}=e\mathbf{p}_1^{\mathrm{T}}=(-1)[2\ \ 2]=[-2\ -2];\\ \Delta b=e=(-1)=-1.\end{cases}$$

Обчислимо нові ваги та зміщення, використовуючи введені раніше правила навчання персептрона.

$$\begin{cases}\mathbf{w}^{\mathrm{T}\,new}=\mathbf{w}^{\mathrm{T}\,old}+\Delta\mathbf{w}^{\mathrm{T}}=\begin{bmatrix}0&0\end{bmatrix}+\begin{bmatrix}-2&-2\end{bmatrix}=\begin{bmatrix}-2&-2\end{bmatrix}=\mathbf{w}^{\mathrm{T}}(1);\\ b^{new}=b^{old}+\Delta b=0+(-1)=-1=b(1).\end{cases}$$

Використаємо новий вектор входу p2, тоді

$$a=\text{hardlim}\left(\mathbf{w}^{\mathrm{T}}(1)\mathbf{p}_2+b(1)\right)=\text{hardlim}\left(\begin{bmatrix}-2&-2\end{bmatrix}\begin{bmatrix}1\\-2\end{bmatrix}+(-1)\right)=\text{hardlim}(1)=1.$$

У цьому випадку вихід персептрона збігається з цільовим виходом, так що похибка дорівнює 0 і не потрібні зміни у вагах або зміщенні. Таким чином,

$$\begin{cases}\mathbf{w}^{\mathrm{T}}(2)=\mathbf{w}^{\mathrm{T}}(1)=\begin{bmatrix}-2&-2\end{bmatrix};\\ b(2)=b(1)=-1.\end{cases}$$

Продовжимо цей процес і переконаємось, що після третього кроку налаштування не змінилися:

$$\begin{cases}\mathbf{w}^{\mathrm{T}}(3)=\mathbf{w}^{\mathrm{T}}(2)=\begin{bmatrix}-2&-2\end{bmatrix};\\ b(3)=b(2)=-1,\end{cases}$$

а після четвертого набули значення

$$\begin{cases}\mathbf{w}^{\mathrm{T}}(4)=\begin{bmatrix}-3&-1\end{bmatrix};\\ b(4)=0.\end{cases}$$

Щоб визначити, чи отримано задовільне рішення, потрібно зробити один прохід через всі вектори входу, щоб перевірити, чи відповідає розв'язок навчальній множині. Знову використаємо перший член навчальної послідовності та отримуємо:

$$\begin{cases}\mathbf{w}^{\mathrm{T}}(5)=\mathbf{w}^{\mathrm{T}}(4)=\begin{bmatrix}-3&-1\end{bmatrix};\\ b(5)=b(4)=0.\end{cases}$$

Переходячи до другого члена, отримаємо наступний результат:

$$\begin{cases}\mathbf{w}^{\mathrm{T}}(6)=\begin{bmatrix}-2&-3\end{bmatrix},\\ b(6)=1.\end{cases}$$

Цим закінчуються ручні обчислення.

Тепер виконаємо аналогічні розрахунки, використовуючи М-функцію adapt. Знову сформуємо модель персептрона, зображеного на рис. 28:

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

Введемо перший елемент навчальної множини

p = {[2; 2]};

```
t = {0};
```

Встановимо параметр passes (число проходів) рівним 1 і виконаємо 1 крок налаштування:

```
net.adaptParam.passes = 1;
[net, a, e] = adapt (net, p, t);
a
```

```
a = [1]
e
e = [-1]
```

Скориговані векторні ваги і зміщення рівні

```
twts = net.IW {1,1}
twts = -2 -2
tbiase = net.b {1}
tbiase = -1
```

Це збігається з результатами, отриманими за ручного розрахунку. Тепер можна ввести другий елемент навчальної множини і т. д., тобто повторити всю процедуру ручного обрахунку і отримати ті ж результати.

Але можна цю роботу виконати автоматично, задавши відразу всю навчальну множинута виконавши 1 прохід:

```
net = newp([-2 2; -2 2],1);
net.trainParam.passes = 1;
p = {[2;2] [1;-2] [-2;2] [-1;1]};
t = {0 1 0 1};
```

Тепер навчимо мережу:

```
[net, a, e] = adapt (net, p, t);
```

Повертаються вихід та помилка:

```
a
a = [1] [1] [0] [0]
e
e = [-1] [0] [0] [1]
```

Скориговані векторні ваги і зміщення рівні

```
twts = net.IW {1,1}
twts = -3 -1
tbiase = net.b {1}
tbiase = 0
```

Моделюючи отриману мережу по кожному входу, отримаємо:

```
a1 = sim (net, p)
a1 = [0] [0] [1] [1]
```

Можна переконатися, що не всі виходи дорівнюють цільовим значенням навчальної множини. Це означає, що слід продовжити налаштування персептрону.

Виконаємо ще 1 цикл налаштування:

```
[net, a, e] = adapt (net, p, t);
a
a = [0] [0] [0] [1]
e
e = [0] [1] [0] [0]
twts = net.IW {1,1}
twts = 2 -3
tbiase = net.b {1}
tbiase = 1
a1 = sim (net, p)
a1 = [0] [1] [0] [1]
```

Тепер рішення збігається з цільовими виходами навчальної множини і всі входи класифіковані правильно.

Якби розраховані виходи персептрона не збіглися з цільовими значеннями, необхідно було б виконати ще кілька циклів налаштування, застосовуючи М-функцію adapt і перевіряючи правильність отриманих результатів.

Для засвоєння викладеного матеріалу можна звернутися до демонстраційних програм, зокрема до програми demop1, яка вирішує завдання класифікації за допомогою простого персептрона.

Як випливає зі сказаного вище, для налаштування (навчання) персептрону застосовується процедура адаптації, яка коригує параметри персептрона за результатами обробки кожного вектора. Застосування М-функції adapt гарантує, що будь-яке завдання класифікації з лінійно відокремленими векторами буде вирішено за скінченне число налаштувань.

Для налаштування (навчання) персептрона можна було б скористатися також М-функцією train. У цьому випадку використовується вся навчальна множина і налаштування параметрів мережі виконується не після кожного проходу, а в результаті всіх проходів навчальної множини. На жаль, не існує доказів того, що такий алгоритм навчання персептрона є схожим. Тому використання М-функції train для навчання персептрону не рекомендується.

Нейронні мережі на основі персептрону мають низку обмежень. По-перше, вихід персептрона може набувати лише одне з двох значень (0 або 1); по-друге, персептрони можуть вирішувати завдання класифікації тільки для лінійно відокремлених наборів векторів. Якщо за допомогою прямої лінії або гіперплощини в багатовимірному випадку можна розділити простір входів на 2 області, в яких будуть розташовані вектори входу, що відносяться до різних класів, вектори входу вважаються лінійно відокремленими. Якщо вектори входу лінійно відокремлені, то доведено, що з використанням процедури адаптації завдання класифікації буде вирішене за скінченний час. Якщо вектори входу лінійно невіддільні, то процедура адаптації не може класифікувати всі вектори належним чином. Демонстраційна програма demop6 ілюструє марність спроби класифікувати вектори входу,

Для вирішення складніших завдань можна використовувати мережі з кількома персептронами. Наприклад, для класифікації чотирьох векторів на 4 групи можна побудувати мережу з двома персептронами, щоб сформувати 2 лінії, що розділяють, і таким чином приписати кожному вектору свою область.

Зазначимо ще одну особливість процесу навчання персептрону. Якщо довжина (модуль) деякого вектора входу набагато більша або менша за довжину інших векторів, то для навчання може знадобитися значний час. Це пов'язано з тим, що алгоритм налаштування пов'язаний із додаванням чи відніманням вхідного вектора від поточного вектора ваг. Таким чином, присутність вектора входу з дуже великими або малими елементами може привести до тривалого часу для налаштування параметрів. Демонстраційна програма demop4 пояснює, як впливає викид довжини вектора на тривалість навчання.

Можна зробити час навчання нечутливим до великих або малих викидів векторів входу, якщо дещо змінити вихідне правило навчання персептрона:

$$\Delta \mathbf{w}^{\mathrm{T}} = (t - a)\mathbf{p}^{\mathrm{T}} = e\mathbf{p}^{\mathrm{T}}.$$

Дійсно, з цього співвідношення випливає, що чим більше компоненти вектора входу p, тим більше він впливає на зміну елементів вектора w. Можна врівноважити вплив великих або малих компонентів, якщо ввести масштабування вектора входу.

Рішення полягає в тому, щоб нормувати вхідні дані так, щоб вплив будь-якого вектора входу мав приблизно рівний внесок:

$$\Delta \mathbf{w}^{\mathrm{T}} = (t - a)\frac{\mathbf{p}^{\mathrm{T}}}{\|\mathbf{p}\|} = e\frac{\mathbf{p}^{\mathrm{T}}}{\|\mathbf{p}\|} \bullet$$

Нормоване правило навчання персептрона реалізується М-функцією learnpn. Цей алгоритм вимагає більшого часу, але значно скорочує кількість циклів навчання, коли зустрічаються викиди векторів входу. Демонстраційна програма demop5 ілюструє це правило навчання.

На закінчення слід зазначити, що основне призначення персептронів – вирішувати завдання класифікації. Вони чудово справляються із завданням класифікації лінійно відокремлених векторів; збіжність гарантується за скінченне число кроків. Тривалість навчання чутлива до викидів довжини (модуля) окремих векторів, але й у такому випадку рішення може бути побудоване.

Одношаровий персептрон може класифікувати лише лінійно відокремлені вектори. Можливі способи подолати ці труднощі передбачають або попередню обробку з метою сформувати лінійно відокремлену множину вхідних векторів або використання багатошарових персептронів. Можна також застосувати інші типи нейронних мереж, наприклад, лінійні мережі або мережі зі зворотним розповсюдженням сигналу, які можуть виконувати класифікацію лінійно нероздільних векторів входу.

# 5. ЛІНІЙНІ МЕРЕЖІ

Лінійні нейронні мережі, що обговорюються в цьому розділі, за своєю структурою аналогічні персептрону і відрізняються лише функцією активації, яка є лінійною. Вихід лінійної мережі може приймати будь-яке значення, у той час як вихід персептрону обмежений значеннями 0 або 1. Лінійні мережі, як і персептрони, здатні вирішувати лише лінійно відокремлені завдання класифікації, однак у них використовується інше правило навчання, засноване на методі найменших квадратів, яке є потужнішим, ніж правило навчання персептрона. Налаштування параметрів виконується таким чином, щоб забезпечити мінімум помилки. Поверхня помилки як функція входів має єдиний мінімум, і визначення цього мінімуму не викликає труднощів. На відміну від персептрона, налаштування лінійної мережі може бути виконане за допомогою як процедури адаптації (adapt), так і процедури навчання (train);

Крім того, у розділі розглядаються адаптовані лінійні нейронні мережі ADALINE (ADAptive Linear Neuron networks), які дозволяють коригувати ваги та зміщення при вступі на вхід кожного нового елемента навчальної множини. Такі мережі широко застосовуються при вирішенні завдань обробки сигналів та в системах управління.

За командою help linnet можна отримати наступну інформацію про М-функції, що входять до складу ППП Neural Network Toolbox і які стосуються побудови лінійних нейронних мереж:

| *Linear networks* | *Лінійні мережі* |
|---|---|
| **New networks** | **Формування нейронної мережі** |
| newlind | Формування лінійного прошарку |
| newlin | Формування лінійного прошарку, що адаптується |
| **Using networks** | **Робота з нейронною мережею** |
| sim | Моделювання мережі |
| init | Ініціалізація мережі |
| adapt | Процедура адаптації |
| train | Процедура навчання |
| **Weight functions** | **Функції зважування** |
| dotprod | Скалярний добуток |
| **Net input functions** | **Функції накопичення** |
| netsum | Сума зважених входів |
| **Transfer функцій** | **Функції активації** |
| purelin | Лінійна |
| **Initialization функцій** | **Функції ініціалізації** |
| initlay | Пошарова ініціалізація |
| initwb | Ініціалізація ваг та зміщень |
| initzero | Ініціалізація нульових ваг та зміщень |
| **Performance** | **Функції оцінки якості мережі** |
| mse | Середньоквадратична похибка |
| **Learning** | **Функції налаштування параметрів персептрону** |
| learnwh | Правило налаштування WH |
| **Adaption** | **Функції адаптації** |
| adaptwb | Функція адаптації ваг та зміщень |
| **Training** | **Функції навчання** |
| trainwb | Функція навчання ваг та зміщень |
| **Analysis функцій** | **Функції аналізу** |
| maxlinlr | Оцінка максимального значення параметра налаштування |
| **Demonstrations and applications** | **Демонстраційні приклади** |
| demolin1 | Приклад функціонування лінійної мережі |
| demolin2 | Навчання лінійного нейрона |
| demolin3 | Навчання лінійного прошарку |
| demolin4 | Завдання лінійної апроксимації |
| demolin5 | Завдання з неповними даними |
| demolin6 | Завдання з лінійно залежними даними |
| demolin7 | Оцінка впливу параметра швидкості налаштування |
| demolin8 | Адаптований лінійний прошарок |
| applin1 | Завдання передбачення |
| applin2 | Завдання адаптивного передбачення |
| applin3 | Ідентифікація лінійної системи |
| applin4 | Адаптивна ідентифікація лінійної системи |

Слід звернути увагу на те, що у версії ППП Neural Network Version 3.0.1 (R11) не представлений демонстраційний приклад demolin3.

## 5.1. Архітектура лінійної мережі

**Модель нейрона.** На рис. 29 показаний лінійний нейрон із двома входами. Він має структуру, подібну до структури персептрону; єдина відмінність полягає в тому, що використовується лінійна функція активації purelin.

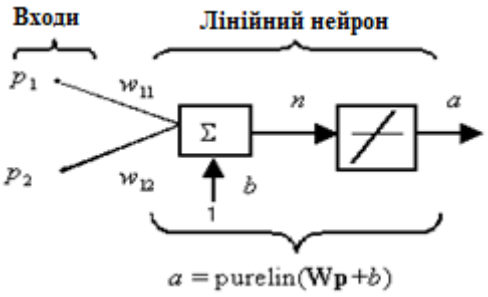



Рис. 29

Вагова матриця W у цьому випадку має лише один рядок і вихід мережі визначається виразом

$$a = \text{purelin}(n) = \text{purelin}(\mathbf{Wp}+b) = \mathbf{Wp}+b = w_{11}p_1 + w_{12}p_2 + b\,.$$

Подібно до персептрону, лінійна мережа задає в просторі входів лінію, на якій функція активації f($n$)=$n$ дорівнює 0 (рис. 30).

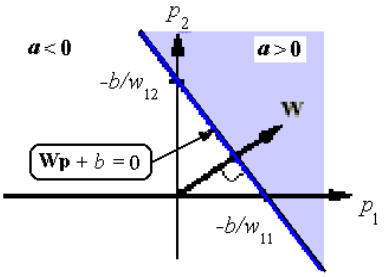



Рис. 30

Вектори входу, розташовані вище цієї лінії, відповідають додатнім значенням виходу, а розташовані нижче - від'мним. Це означає, що лінійна мережа може бути використана для вирішення завдань класифікації. Однак така класифікація може бути виконана тільки для класу об'єктів, що лінійно відокремлюються. Таким чином, лінійні мережі мають те саме обмеження, що і персептрон.

**Архітектура мережі.** Лінійна мережа показана на рис. 31 *а*, включає S нейронів, розміщених в одному прошарку і пов'язаних з R входами через матрицю ваг W.

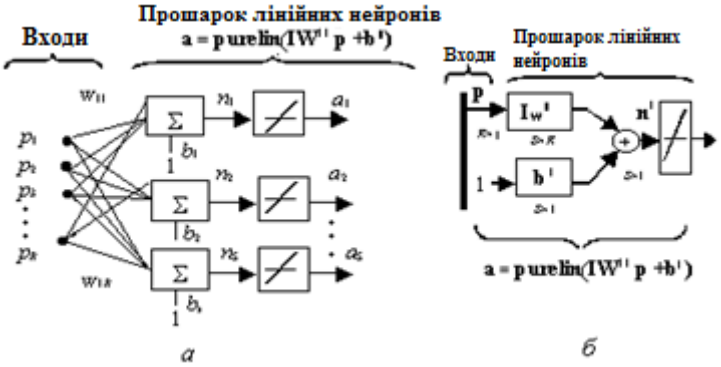



Рис. 31

На рис. 31 б показана укрупнена структурна схема цієї мережі, вектор виходу а якої має розмір S×1.

## 5.2. Створення моделі лінійної мережі

Лінійну мережу з одним нейроном, показану на рис. 29, можна створити так:

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

Перший вхідний аргумент визначає діапазон зміни елементів вектора входу; другий аргумент показує, що мережа має єдиний вихід. Початкові ваги та зміщення за замовчуванням дорівнюють 0.

Надамо вагам і зміщенню наступні значення:

```
net.IW{1,1} = [2 3];
net.b{1} =[-4];
```

Тепер можна промоделювати лінійну мережу для наступного пред'явленого вектора входу:

```
p = [5; 6];
a = sim (net, p)
a = 24
```

## 5.3. Навчання лінійної мережі

Для заданої лінійної мережі та відповідної множини векторів входу та цілей можна обчислити вектор виходу мережі та сформувати різницю між вектором виходу та цільовим вектором, яка визначить деяку похибку. У процесі навчання мережі потрібно знайти такі значення ваг та зсувів, щоб сума квадратів відповідних похибок була мінімальною. Це завдання можна розв'язати, тому що для лінійних систем функція квадратичної помилки є унімодальною.

Як і для персептрона, застосовується процедура навчання з учителем, яка використовує навчальну множину виду:

$$\{\mathbf{p}_1\ \mathbf{t}_1\},\{\mathbf{p}_2\ \mathbf{t}_2\},\ldots,\{\mathbf{p}_Q\ \mathbf{t}_Q\},$$

де $\mathbf{p}_1,\mathbf{p}_2,\ldots,\mathbf{p}_Q$ - входи мережі; $\mathbf{t}_1,\mathbf{t}_2,\ldots,\mathbf{t}_Q$ - відповідні цільові виходи.

Потрібно мінімізувати наступну функцію середньої квадратичної помилки:

$$mse=\frac{1}{Q}\sum_{k=1}^{Q}e(k)^2=\frac{1}{Q}\sum_{k=1}^{Q}(t(k)-a(k))^2\ .$$

### *Процедура налаштування*

На відміну від багатьох інших мереж налаштування лінійної мережі для заданої навчальної множини може бути виконане за допомогою прямого розрахунку з використанням М-функції newlind.

Припустимо, що задані такі вектори з навчальної множини:

```
P = [1 -1.2];
T = [0.5 1];
```

Побудуємо лінійну мережу та промоделюємо її:

```
net = newlind(P,T);
Y = sim(net, P);
Y = 0.5 1
net.IW {1,1}
ans = -0.22727
net.b
ns = [0.72727]
```

Вихід мережі відповідає цільовому вектору.

Задамо наступний діапазон ваг та зміщень, розрахуємо критерій якості навчання та побудуємо його лінії рівня (рис. 32):

```
w_range=-1:0.1: 0; b_range=0.5:0.1:1;
ES = errsurf (P, T, w_range, b_range, 'purelin');
contour(w_range, b_range,ES,20)
hold on
plot(-2.2727e-001,7.2727e-001, 'x') % Рис.32.
hold off
```

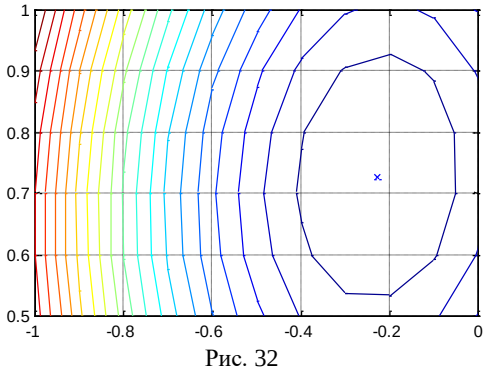



Рис. 32

На графіку знаком **x** відзначені оптимальні значення ваги та зміщення для цієї мережі.

Демонстраційний приклад demolin1 пояснює структуру лінійної мережі, побудову поверхні помилок та вибір оптимальних налаштувань.

### *Процедура навчання*

Для навчання лінійної нейронної мережі може бути використана типова процедура навчання за допомогою М-функції train. Ця функція для кожного вектора входу виконує налаштування ваг та зсуву, використовуючи М-функцію learnp. В результаті мережа налаштовуватиметься за сумою всіх корекцій.

Назвемо кожен перерахунок для набору вхідних векторів епохою. Це відрізняє процедуру навчання від процедури адаптації adapt, коли налаштування параметрів реалізовується при поданні кожного окремого вектора входу. Потім процедура train моделює налаштовану мережу для набору векторів, порівнює результати з набором цільових векторів і обчислює середньоквадратичну помилку. Як тільки значення помилки стає менше від заданого або вичерпано граничну кількість епох, навчання припиняється.

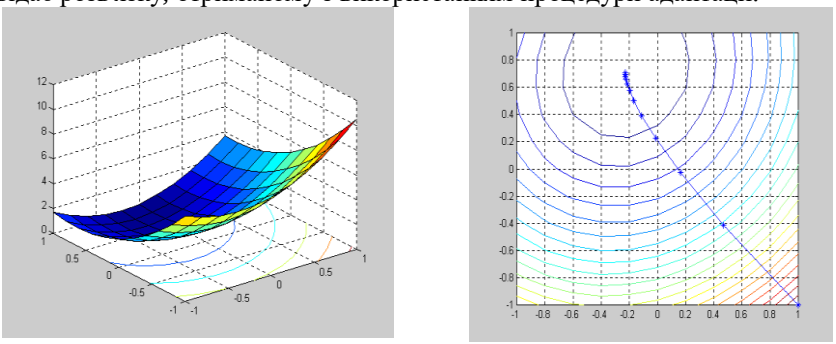
Звернемося до прикладу, який використовувався при розгляді процедури адаптації, і виконаємо процедуру навчання.

```
P = [1 -1.2]; % Вектор входів
T=[0.5, 1]; % Вектор цілей
% Максимальне значення параметра навчання
maxlr = 0.40 * maxlinlr (P, 'bias');
% Створення лінійної мережі
net = newlin([-2,2],1,[0],maxlr);
% Розрахунок функції критерію якості
w_range=-1:0.2:1; b_range=-1:0.2:1; % Діапазони значень ваг та зміщення
ES = errsurf (P, T, w_range, b_range, 'purelin');
% Побудова поверхні функції критерію якості
surfc(w_range, b_range, ES) % Рис.33,а
```

На рис. 33 *а* побудована поверхня функції критерію якості в просторі параметрів мережі. У процесі навчання траєкторія навчання переміщатиметься з початкової точки до точки мінімуму критерію якості. Виконаємо розрахунок та побудуємо траєкторію навчання лінійної мережі для заданих початкових значень ваг та зміщення.

```
% Розрахунок траєкторії навчання
x = zeros (1,50); y = zeros(1,50);
net.IW{1}=1; net.b {1} = -1; % Початкові значення ваг та зміщення
x(1) = net.IW{1}; y(1) = net.b{1};
net.trainParam.goal = 0.001; % Порогове значення критерію якості
net.trainParam.epochs = 1; % Число епох
% Цикл обчислення ваги та зміщення для однієї епохи
for i =2:50,
[net, tr] = train (net, P, T);
x(i) = net.IW{1};
y(i) = net.b{1};
end
% Побудова ліній рівня та траєкторії навчання
clf, contour(w_range, b_range, ES, 20), hold on
plot(x, y,'-*'), hold off, % Рис.33,б
```

На рис. 33 *б* символом * відзначені значення ваг та зміщення на кожному кроці навчання; видно, що за 10 кроків при заданій точності (порогове значення критерію якості) 0.001 отримаємо w = –0.22893, b = 0.70519. Це відповідає розв'язку, отриманому з використанням процедури адаптації.



*а* *б*

Рис. 33

Якщо не потрібно будувати траєкторію процесу навчання, то можна виконати навчання, звернувшись до М-функції train лише 1 раз:

```
net.IW{1}=1; net.b {1} = -1; % Початкові значення ваг та зміщення
net.trainParam.epochs = 50; % Число епох навчання
net.trainParam.goal = 0.001; % Порогове значення критерію якості
[net, tr] = train (net, P, T);
TRAINWB, Epoch 0/50, MSE 5.245/0.001.
TRAINWB, Epoch 11/50, MSE 0.000483544/0.001.
TRAINWB, Performance goal met.
net.IW, net.b
ans = [-0.22893]
ans = [0.70519]
```

На рис. 34 показано, як змінюється критерій якості в кожному циклі навчання.

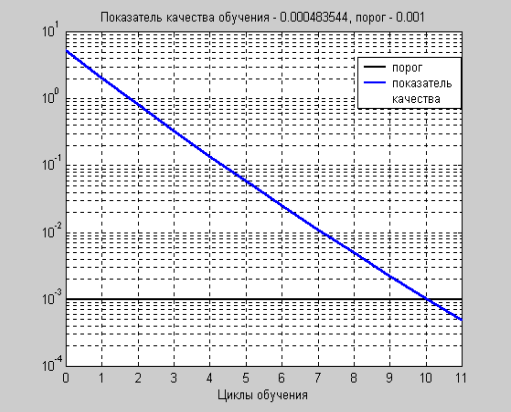



Рис. 34

Якщо підвищити точність навчання до значення 0.00001, то отримаємо такі результати:

```
net.trainParam.goal = 0.00001;
[net, tr] = train (net, P, T);
net.IW, net.b
TRAINWB, Epoch 0/50, MSE 0.000483544/1e-005.
TRAINWB, Epoch 6/50, MSE 5.55043e-006/1e-005.
TRAINWB, Performance goal met.
ans = [-0.22785]
ans = [0.72495]
```

## 5.4. Застосування лінійних мереж

### *Завдання класифікації векторів*

Покажемо, як лінійні мережі можуть бути використані для вирішення завдань класифікації. Якщо використовується процедура навчання train, параметри мережі налаштовуються з урахуванням сумарного значення функції помилки. Це відрізняється від процедури адаптації adapt, для роботи якої характерне налаштування параметрів з урахуванням помилки при поданні кожного вектора входу. Потім навчання застосовується до скоригованої мережі, обчислюються виходи, порівнюються з відповідними цілями та знову обчислюється помилка навчання. Якщо досягнуто припустимої похибки або перевищено максимальну кількість циклів (епох) навчання, то процедура налаштування припиняється. Алгоритм навчання та налаштування сходиться, якщо завдання класифікації можна розв'язати.

Проілюструємо розв'язання задачі класифікації, раніше вирішеної персептроном. Використовуємо при цьому найпростішу лінійну мережу, представлену на рис. 29. Навчальна множина представлена наступними чотирма парами векторів входів та цілей:

$$\left\{\mathbf{p}_1=\begin{bmatrix}2\\2\end{bmatrix}, t_1=0\right\}\quad\left\{\mathbf{p}_2=\begin{bmatrix}1\\-2\end{bmatrix}, t_2=1\right\}\quad\left\{p_3=\begin{bmatrix}-2\\2\end{bmatrix}, t_3=0\right\}\quad\left\{p_4=\begin{bmatrix}-1\\1\end{bmatrix}, t_4=1\right\}.$$

Визначимо лінійну мережу з початковими значеннями ваг та зміщення, які використовуються за замовчуванням, тобто нульовими; задаємо допустиму похибку навчання рівну 0.1:

```
p = [2 1 -2 -1; 2 -2 2 1];
t = [0 1 0 1];
net = newlin([-2 2; -2 2],1);
%Ініціалізація лінійної мережі з двома входами та одним виходом
net.trainParam.goal = 0.1;
[net, tr] = train (net, p, t);
TRAINWB, Epoch 0/100, MSE 0.5/0.1.
TRAINWB, Epoch 25/100, MSE 0.181122/0.1.
TRAINWB, Epoch 50/100, MSE 0.111233/0.1.
TRAINWB, Epoch 64/100, MSE 0.0999066/0.1.
TRAINWB, Performance goal met.
```

Порогове значення функції якості досягається за 64 цикли навчання, а відповідні параметри мережі набувають значення

```
weights = net.iw {1,1}
bias = net.b(1)
weights = -0.061482 -0.21938
bias = [0.5899]
```

Виконаємо моделювання створеної мережі з векторами входу з навчальної множини та обчислимо помилки мережі:

```
A = sim (net, p)
err = t - sim (net, P)
```

```
A = 0.028173 0.96718 0.2741 0.432
err = -0.028173 0.03282 -0.2741 0.568
```

Зауважимо, що похибки мережі дуже значні. Спроба задати більшу точність в цьому випадку не приведе до мети, оскільки можливості лінійної мережі обмежені. Демонстраційний приклад demolin4 ілюструє проблему лінійної залежності векторів, яка властива в тому числі й цьому випадку.

Навчання лінійної нейронної мережі ілюструється демонстраційною програмою demolin2, яка повертає значення ваг та похибку в процесі навчання. У зв'язку з аналізованою проблемою класифікації можна також звернутися до демонстраційної програми nnd10lc, у якій розглянуто класичне завдання класифікації об'єктів під час дії шумів.

### *Фільтрування сигналу*

На рис. 35 представлена структурна схема цифрового фільтра, відмінною особливістю якого є те, що він включає динамічний компонент – лінію затримки (ЛЗ) та 1 прошарок лінійної нейронної мережі.

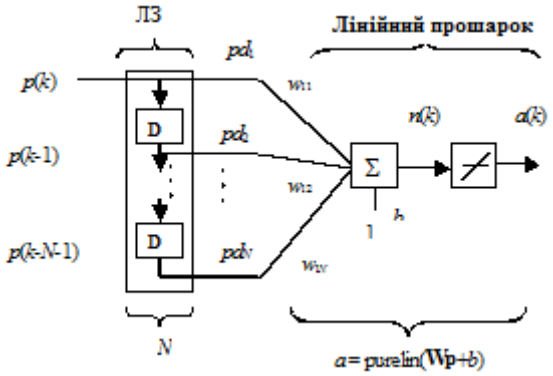



Рис. 35

Послідовність значень вхідного сигналу {p(k)} надходить на ЛЗ, що складається з N-1 блоку затримки; вихід ЛЗ – N-мірний вектор pd, складений із значень входу на моменти часу k, k–1, …, k–N–1.

Вихід лінійного нейронного прошарку та фільтра в цілому описується наступним динамічним співвідношенням:

$$a(k) = \text{purelin}(\mathbf{W}\mathbf{p} + b) = \sum_{i=1}^{R} w_{1i} a(k-i+1) + b.$$

Розглянемо конкретний приклад цифрового фільтра, поданий на рис. 36.

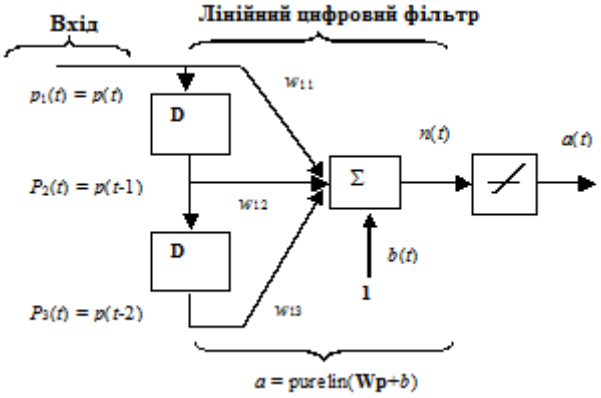



Рис. 36

Припустимо, що вхідний сигнал приймає значення в діапазоні від 0 до 10 і сформуємо лінійну мережу з одним входом і одним виходом, використовуючи М-функцію newlin:

```
net = newlin([0,10],1);
```

Введемо ЛЗ із двома тактами затримки:

```
net.inputWeights{1,1}.delays = [0 1 2];
```

визначимо такі початкові значення ваг та зміщення:

```
net.IW{1,1} = [7 8 9];
net.b{1} = [0];
```

задаємо початкові умови для динамічних блоків лінії затримки:

```
pi = {1 2}
```

Послідовність їх завдання зліва направо відповідає блокам затримки, розташованим на малюнку зверху вниз. Цим завершується формування мережі.

Тепер визначимо вхідний сигнал у вигляді наступної послідовності значень:

```
p = {3 4 5 6}
```

і промоделюємо цю мережу:

```
[a, pf] = sim (net, p, pi);
a = [46] [70] [94] [118]
pf = [5] [6].
```

Щоб отримати бажану послідовність сигналів на виході, необхідно виконати налаштування сформованої мережі. Припустимо, що задана наступна бажана послідовність для виходу фільтра:

```
T = {10 20 30 40};
```

Виконаємо налаштування параметрів, використовуючи процедуру адаптації adapt та 10 циклів навчання:

```
net.adaptParam.passes = 10;
[net, y, E pf, af] = adapt (net, p, T, pi); % Процедура адаптації
```

Виведемо отримані значення ваг, зміщення та вихідного сигналу:

```
wts = net.IW {1,1}, bias = net.b {1}, y
wts = 0.5059 3.1053 5.7046
bias = 1.5993
y = [11.856] [20.774] [29.668] [39.004]
```

Якщо продовжити процедуру налаштування, можна ще точніше наблизити вихідний сигнал до бажаного:

```
net.adaptParam.passes = 500;
[net, y, E, pf, af] = adapt (net, p, T, pi);
y
y = [10.004] [20.002] [29.999] [39.998]
```

Таким чином, лінійні динамічні нейронні мережі можуть бути адаптовані для вирішення задач фільтрації часових сигналів. Для мереж такого класу часто використовується назва ADALINE (ADaptive LInear NEtwork) – лінійні мережі, що адаптуються. Розглянемо інші застосування цих мереж.

### *Передбачення сигналу*

Спробуємо застосувати мережу ADALINE для передбачення значень детермінованого процесу p(t). Звернемося до рис. 37.

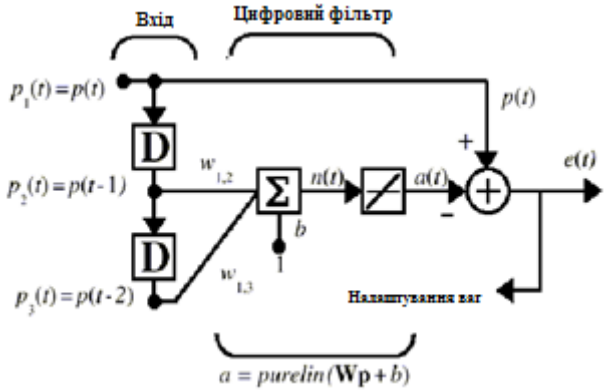



Рис. 37

Деякий сигнал надходить на лінію затримки так, що на її виході формуються 2 сигнали: p(t-1), p(t-2). Налаштування мережі реалізується за допомогою М-функції adapt, яка змінює параметри мережі на кожному кроці з метою мінімізувати похибку *e*(t) = a (t) - p (t). Якщо ця похибка нульова, вихід мережі a(t) точно дорівнює p(t) і мережа виконує передбачення належним чином.

Нижче наведено сценарій, який призначений для вирішення задачі передбачення сигналу на 1 крок уперед. Вхідний детермінований процес отримано в результаті проходження порогового сигналу через блок коливання.

Запишемо наступний сценарій для вирішення задачі передбачення сигналу:

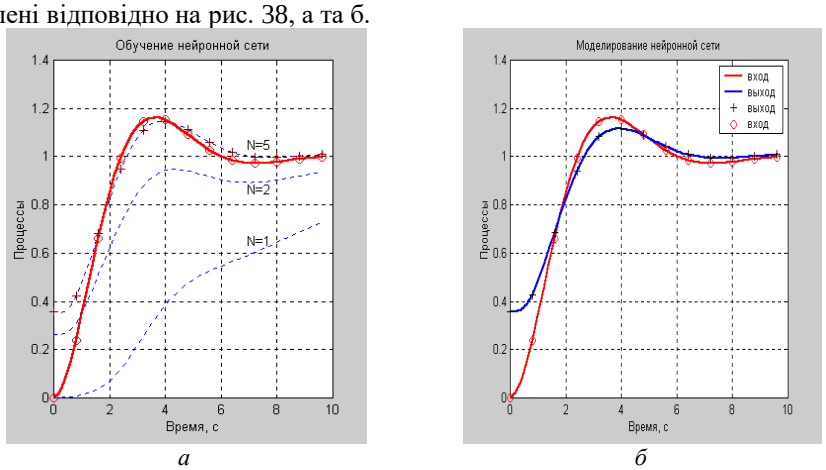
```
clear
sys = ss(tf(1,[1 1 1])); % Формування коливальної ланки

% Отримання реакції на порогову вхідну дію
time = 0:0.2:10;
[y,time] = step(sys,0:0.2:10);

% Формування навчальної множини
p = y(1: length(time)-2)';
t = y(3: length(time))';
```

```
time = time(1:length(time)-2);

% Формування нейронної мережі
net = newlin([-1 1], 1, [1 2]);
P = num2cell(p);
T = num2cell(t);

% Налаштування нейронної мережі
pi = {0 0};
net.adaptParam.passes = 5;
[net, Y, E, Pf, Af] = adapt (net, P, T, pi);
Y1 = cat (1, Y {:}); % Перетворення масиву комірок на масив double
% Побудова графіків
plot(time,Y1,'b:',time,p,'r-'), xlabel('Час, с'), ylabel('Процеси')
title('Навчання нейронної мережі')

% Моделювання нейронної мережі
x = sim (net, P);
x1 = cat (1, x {:});
plot(time,x1,'b')
```

Знайдені значення ваг і зміщення рівні

```
net.IW{1,1}, net.b
ans = 0.33427 0.31816
ans = [0.35853]
```

Результати навчання та моделювання результуючої нейронної мережі, що вирішує задачу передбачення сигналу, представлені відповідно на рис. 38, а та б.




*а* *б*

Рис. 38

Як очевидно з цих малюнків, прийнятна для цієї постановки задачі кількість циклів навчання дорівнює п'яти; результати моделювання показують, що при тривалості процесу 10 секунд діапазон успішного передбачення знаходиться в межах 1.8–10 с.

## *Придушення шумів*

Завдання придушення шумів виникає у наступній ситуації. Коли пілот говорить у мікрофон, шум двигуна в кабіні додається до голосового сигналу і пасажири чують мову зі спотвореннями. Потрібно відновити промову пілота, видаливши з неї звуки шумів двигуна. Для вирішення цього завдання побудуємо адаптивний фільтр, вважаючи, що у нашому розпорядженні є записи шумів двигуна (рис. 39).

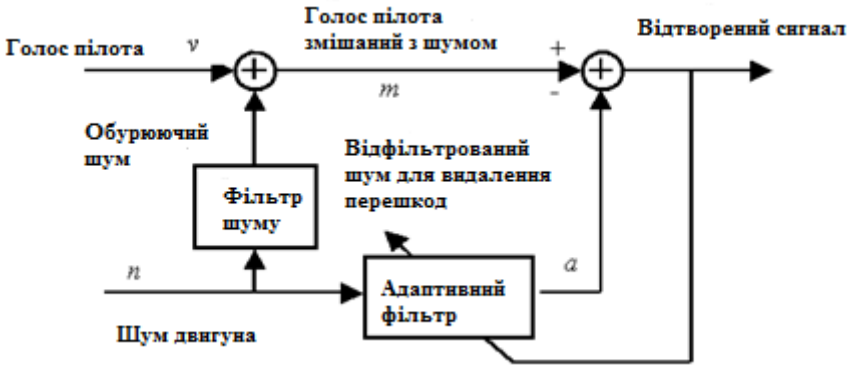



Рис. 39

Використовуючи нейронну мережу ADALINE, що адаптується, побудуємо такий фільтр, який дозволить видалити шум *n* із зашумленого сигналу *m*. Розглянемо докладніше структурну схему, подану на рис. 39. Голос пілота *v* змішаний із перешкодою від шумів двигуна, яка формується фільтром шуму за відомими записами сигналу *n*. Зашумлений сигнал *m* моделює спотворену мову пілота. Завдання полягає в тому, щоб за допомогою адаптивного фільтра сформувати таку модель шуму *a*, яка при відніманні із зашумленого сигналу дозволила б виділити промову пілота.

Адаптивний фільтр, побудований на базі лінійної нейронної мережі ADALINE, повинен бути налаштований так, щоб усунути шум двигуна. Зауважимо, що така адаптивна фільтрація краща за класичний фільтр, в якому шум не віднімається, а відфільтровується від сигналу *m*.

На закінчення відзначимо основні переваги та обмеження лінійних мереж:

- Одношарові лінійні мережі можуть вирішувати завдання лінійної апроксимації функцій та розпізнавання образів.
- Одношарові лінійні мережі можуть бути розраховані безпосередньо, або навчені з використанням правила навчання WH. Крім того, для їх налаштування можуть застосовуватись процедури адаптації.
- Архітектура одношарової лінійної мережі повністю визначається завданням, яке повинне буде вирішене, причому число входів мережі та число нейронів у прошарку визначається числом входів та виходів задачі.
- Адаптовані лінійні мережі ADALINE знаходять велике практичне застосування при побудові цифрових фільтрів для обробки сигналів.
- Лінійні нейронні мережі можуть бути успішно навчені лише у тому випадку, коли входи та виходи пов'язані лінійно. Проте навіть у тому випадку, коли лінійна мережа не може знайти точного рішення, вона може побудувати найближче рішення в сенсі мінімуму середньоквадратичної помилки за умови, що параметр навчання досить малий. Така мережа знайде найточніший ров'язок у межах лінійної структури мережі. Це пов'язано з тим, що поверхнею помилки навчання є багатовимірний параболоїд, що має єдиний мінімум, і алгоритм градієнтного спуску повинен привести розв'язок до цього мінімуму.
- При роботі з моделями лінійних мереж можуть виникати ситуації, коли кількість параметрів, що налаштовуються недостатня, щоб виконати всі умови; у цьому випадку кажуть, що мережа перевизначена. Однак може мати місце і зворотна ситуація, коли кількість параметрів, що настроюються, занадто велика, і в цьому випадку говорять, що мережа недовизначена. Проте в обох випадках метод найменших квадратів здійснює налаштування, прагнучи мінімізувати помилку мережі. Ці ситуації пояснюються демонстраційними прикладами demolin4 та demolin5.
- Розв'язність лінійної задачі за допомогою лінійної нейронної мережі може бути встановлена таким чином. Якщо сумарна кількість ваг і зміщень лінійної мережі S*R + S, де R – кількість входів, S – кількість прошарків, дорівнює кількості пар векторів входу та цілей Q, то таке завдання можна розв'язати за допомогою лінійної нейронної мережі. Це справедливо, за вийнятком того випадку, коли вектори входу є лінійно залежними і використовується мережа без зсувів. Демонстраційний приклад demolin6 пояснює цю ситуацію.