НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОТЕХНОЛОГІЙ І ПРИРОДОКОРИСТУВАННЯ
Факультет інформаційних технологій
Кафедра комп'ютерних наук

Теорія розпізнавання образів та класифікації

Лабораторна робота №8

«Дослідження багатошарового нелінійного персептрону і алгоритму зворотнього роозповсюдження помилки»

(4 години)

Київ - 2023

Мета роботи: дослідження можливостей багатошарового персептрону, як універсального апроксиматора і класифікатора.

Підготовка до роботи: Вивчити і уяснити призначення і зміст завдання до лабораторної роботи.

Завдання:

1. Ознайомитись з методичною розробкою до лабораторноъ роботи.
2. Ознайомитись з рекомендованною літературою.
3. Розробити для проведення дослідження програмний додаток в середовищи розробки MS Visual Studio.
4. Дослідити зображення сформованих еталонних символів, їх характеристики.
5. Сформувати кілька нових еталонних символів, з різними рівнями та типами відмінностей у відповідності до прядку виконання лабораторної роботи.
6. Сформувати текст, куди входять однакові символи.
7. Вибрати величину зашумлення та метод розпізнавання, зафіксувати результат розпізнавання та відстань до найближчого еталона для кожного символу, що розпізнається.
8. Виконати дослідження у відповідності до п.п. 3, 4, 5 порядку виконання роботи.
9. Виконати завдання до лабораторної роботи..
10. За результатами досліджень скласти звіт з обґрунтованими висновками.

Форма звіту

1. Тема та мета.
2. Опис завдання згідно розроблюваної студентом системи.
3. Дослідження алгоритмів у відповідності до методичних матеріалів лабораторної роботи.
4. При обраному значенні *p* розрахувати у командному вікні MatLab пряме розповсюдження вхідного сигналу, зворотнє розповсюдження помилки і зміну вігових показників на першій ітерації алгоритму *Backpropagation*. Порівняйте з результатами, які отримані в алгоритмі *Backpropagation Calculation*.
5. Висновки.

Рекомендована література

1. Кудрявцев, В.Б. Теорія тестового розпізнавання/В.Б. Кудрявцев, А.Є. Андрєєв, Е.Е. Гасанів. - М.: Фізматліт, 2007. - 320 с. - ISBN 978-5-9221-0872-0.

2. Потапов, А.А. Найновіші методи обробки зображень: монографія / ід. заг. ред. А. А. Потапова. - М.: Фізматліт, 2008. - 496 с. - ISBN 978-5-9221-0841-6.

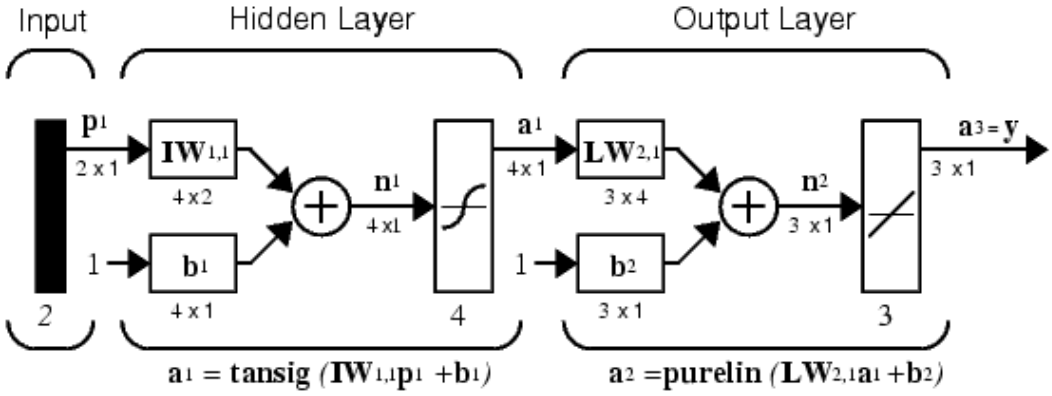
3. Федотов, Н.Г. Теорія ознак розпізнавання образів на основі стохастичної геометрії та функціонального аналізу / Н.Г. Федотов. - М.:Фізматліт, 2009. - 304 с. - ISBN 978-5-9221-0996-3.

4. Зіангірова Л.Ф. Технології хмарних обчислень [Електронний ресурс]: навчальний посібник / Зіангірова Л.Ф. - Електрон. текстові дані. - Саратов: Вузовська освіта, 2016. - 300 с. - Режим доступу: http://www.iprbookshop.ru/41948. - ЕБС «IPRbooks».

Теоретичні відомості

У лабораторній роботі розглядається нейронна мережа з прямою передачею сигналу (з прямим зв'язком), тобто мережа, в якій сигнали передаються тільки в напрямку від вхідного шару до вихідного, і елементи одного шару пов'язані з усіма елементами наступного шару. Найважливішим реалізації нейронних мереж є визначення алгоритму навчання мережі

В даний час одним з найефективніших та обґрунтованих методів опромінення нейронних мереж є *алгоритм зворотного розповсюдження помилки*, який застосовується до *односпрямованим багатошаровим мережам*. У багатошарових нейронних мережах є безліч прихованих нейронів, входи та виходи яких не є входами та виходами нейронної мережі, а з'єднують нейрони всередині мережі, тобто *приховані нейрони*. На малюнку 3.1 показано архітектуру нейронної мережі з прямою передачею сигналу.



Малюнок 3.1. Схема архітектури нейронної мережі з прямою передачею сигналу

Тут прийнято позначення: p_1 – вектор входу, $IW_{i,j}$, $LW_{i,j}$ - матриці весов входу та виходу, b_i - Зміщення, a_i - Вихід шару, y - Вихід мережі, tansig (гіперболічна тангенціальна), purelin (лінійна) – відповідні функції активації.

Ваги та усунення визначаються за допомогою алгоритму зворотного розповсюдження помилок.

Навчання мережі зворотного розповсюдження вимагає виконання наступних операцій:

1. Вибрати чергову навчальну пару з навчальної множини; подати вхідний вектор на вхід мережі.
2. Обчислити вихід мережі.
3. Обчислити різницю між виходом мережі та необхідним виходом (цільовим вектором навчальної пари).
4. Коригувати ваги мережі так, щоб мінімізувати помилку.
5. Повторювати кроки з 1 по 4 для кожного вектора навчальної множини доти, доки помилка на всій множині не досягне прийнятного рівня.

Функція *newff створює* нейронну мережу прямого поширення сигналу, що навчається за допомогою алгоритму зворотного поширення помилки:

net = newff (PR, [S1 S2 SNl], {TF1 TF2 TFNl}, BTF, BLF,PF). Розглянемо параметри функції *newff: PR* – матриця інтервалів значень для *R* вхідних елементів, що задаються мінімальним і максимальним значенням. ями; S_i-розмір *i*-го шару, для *N* шарів; TF_i – функція активації *i*-го шару, за умовчанням використовується функція *tansig* - Гіперболічний тангенс; *BTF* - функція навчання мережі методом зворотного розповсюдження помилки, за промовчанням використовується функція *traingdx; BLF* – функція зміни ваг при навчання, за замовчуванням використовується *learngdm; PF* – функція виміру помилки, за замовчуванням *mse.* Функція *newff* повертає багатошарову нейрон- ну мережу прямого та зворотного розповсюдження сигналу та помилки відповід-

ственно. Функції активації можуть бути вибрані з наступного переліку: гіперболічний тангенс*tansig;*логістична сигмоїда*logsig*або лінійна функція*purelin.*

2. Створення та навчання нейронної мережі за допомогою алгоритму обер- ного поширення помилки

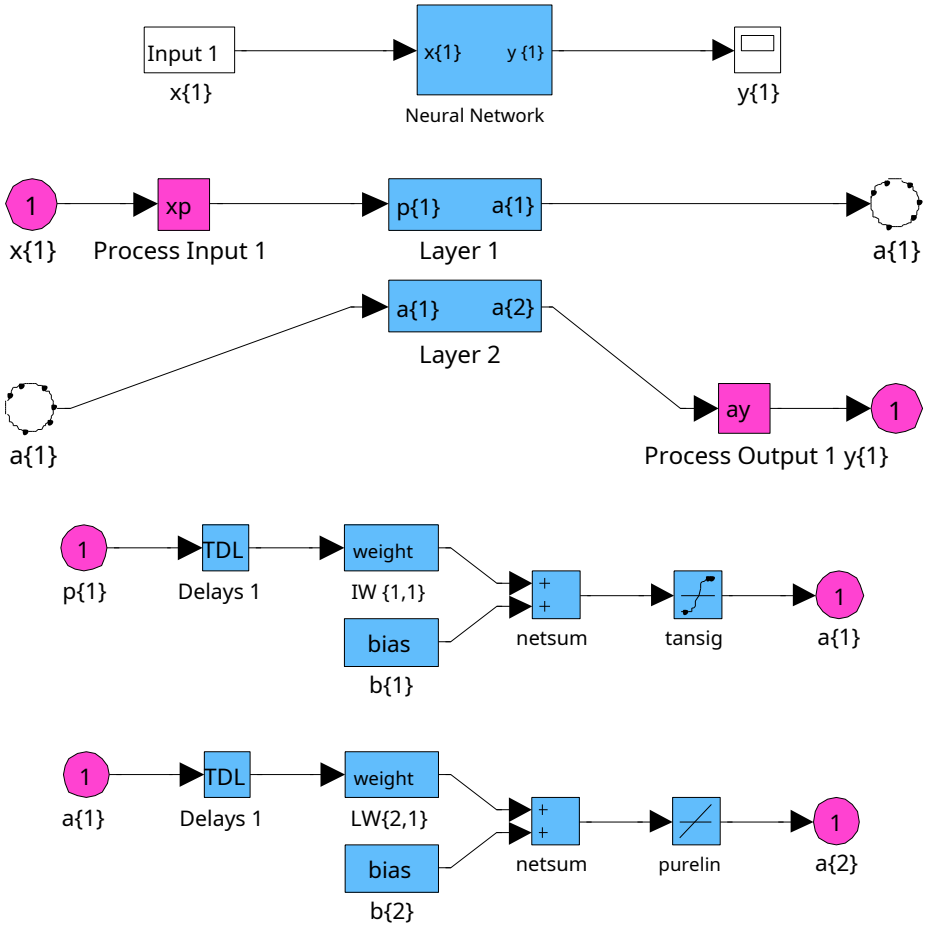
Створити нейронну мережу, щоб забезпечити наступне відображення послідовності входу P у послідовність цілей T:

P = [0.10 0.31 0.51 0.72 0.93 1.14 1.34 1.55 1.76 1.96 2.17 2.38 2.59 2.79 3.00];

T = [0.1010 0.3365 0.6551 1.1159 1.7632 2.5847 3.4686 4.2115 4.6152 4.6095 4.2887 3.8349 3.4160 3.1388 3.0603];

net = newff([0 3],[4 1], {'tansig' 'purelin'});

gensim(net)

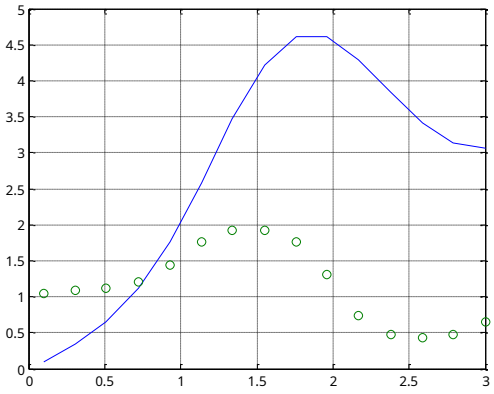


Малюнок 3.2. Структурна схема нейронної мережі

Виконаємо моделювання мережі та побудуємо графіки сигналів виходу і цілі:

Y = sim (net, P); figure(1), clf

plot(P, T, P, Y, 'o'), grid on

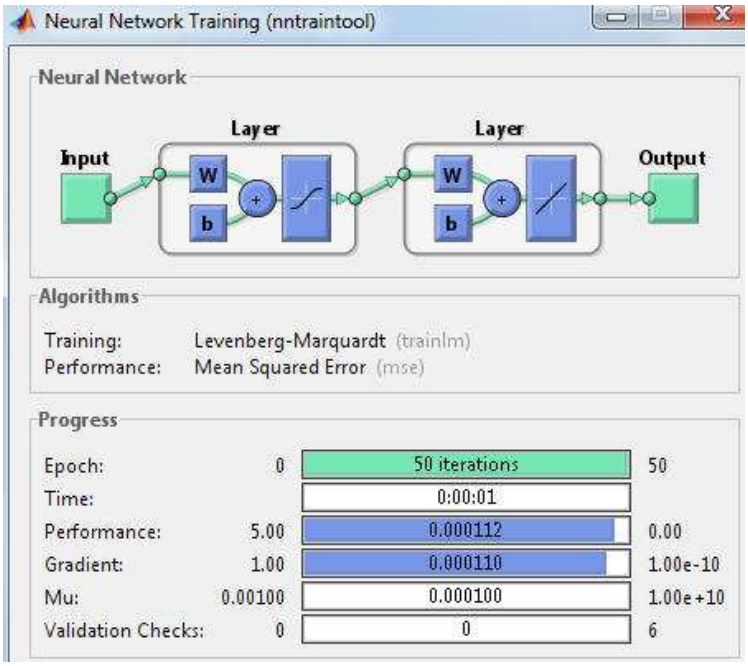


Малюнок 3.3. Графіки сигналів виходу та мети

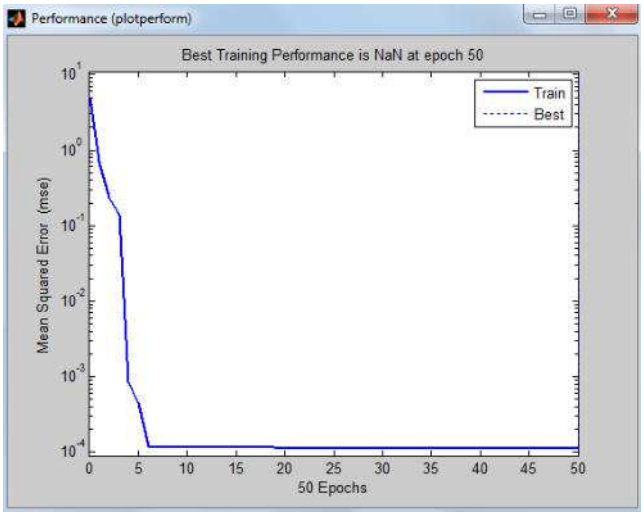
Навчимо мережу протягом 50 циклів:

net.trainParam.epochs = 50; net =

train(net, P, T);



Малюнок 3.4. Вікно навчання нейронної мережі



Малюнок 3.5. Зміна помилки мережі у процесі навчання

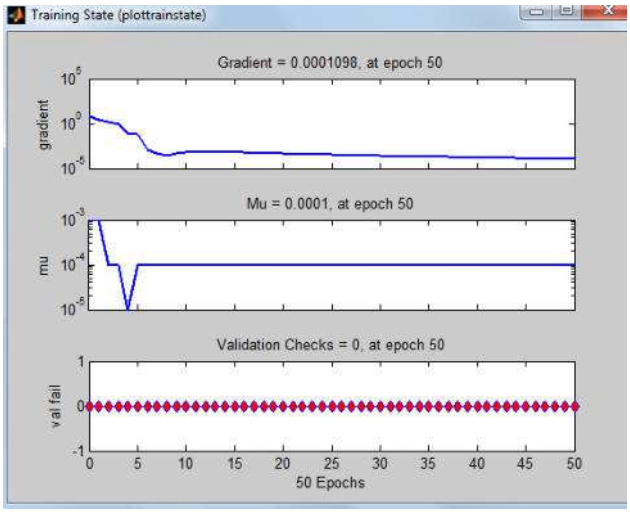


Рисунок 3.6 – Вікно стану навчання

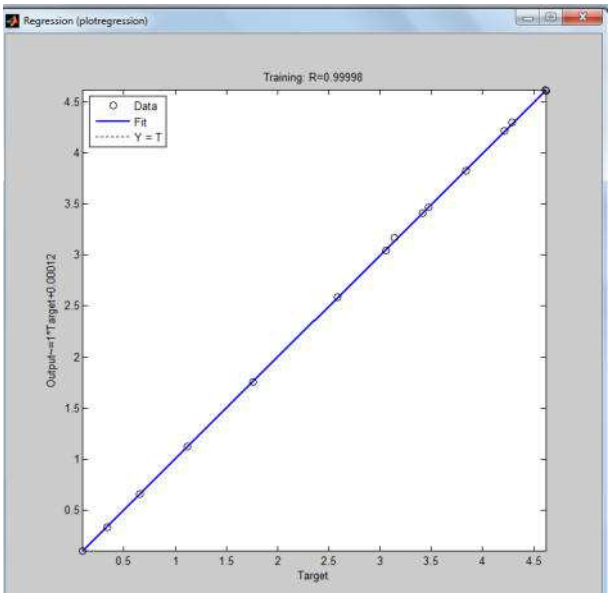
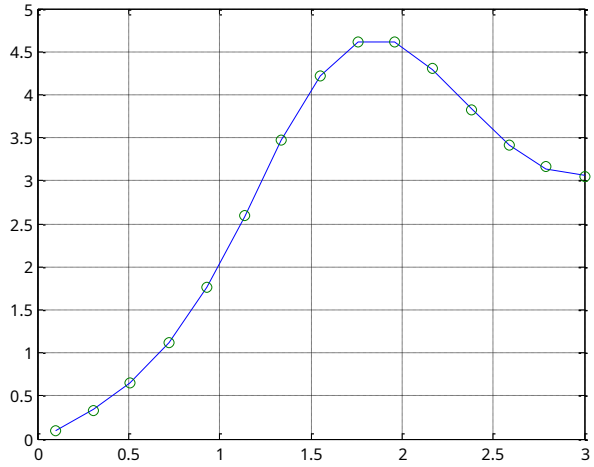


Рисунок 3.7 – Вікно лінійної регресії між виходом СР та цілями

Виконаємо моделювання сформованої двошарової мережі, використовуючи навчальну послідовність входу

Y = sim(net, P);

plot(P, T, P, Y, 'o'), grid on



Малюнок 3.8. Результат апроксимації векторів двошаровим персептроном.

Для дослідження роботи алгоритму зворотного розповсюдження помилки скористаємося прикладом, вбудованим у MatLab toolbox, набравши команду*demo.*

У діалоговому вікні необхідно послідовно вибирати пункти меню: *інструменти, інструменти, neural networks, інші Neural Network Design Textbook Demos, Table of Contents, 10–13, Backpropagation Calculation.*

У прикладі використовується двошаровий персептрон із двома нелінійними нейронами в першому шарі та одним у другому. Дія алгоритму зворотного розповсюдження помилки розбита на наступні кроки: призначення входу та бажаного виходу, прямий прохід вхідного сигналу до виходу, зворотне поширення помилки, зміна ваг. Змінні, що дозволяють простежити роботу алгоритму зворотного розповсюдження помилки, позначені таким чином:

P – вхідний сигнал;

$W_1(i)$ – вектор ваг першого шару, $W_1(1)$ – вага зв'язку, що передає вхідний сигнал на перший нейрон, а $W_1(2)$ – на другий;

$W_2(i)$ – вектор ваги другого шару, $W_2(1)$ – вага зв'язку, що передає вхідний сигнал з першого нейрона до другого шару, а $W_2(2)$ – з другого;

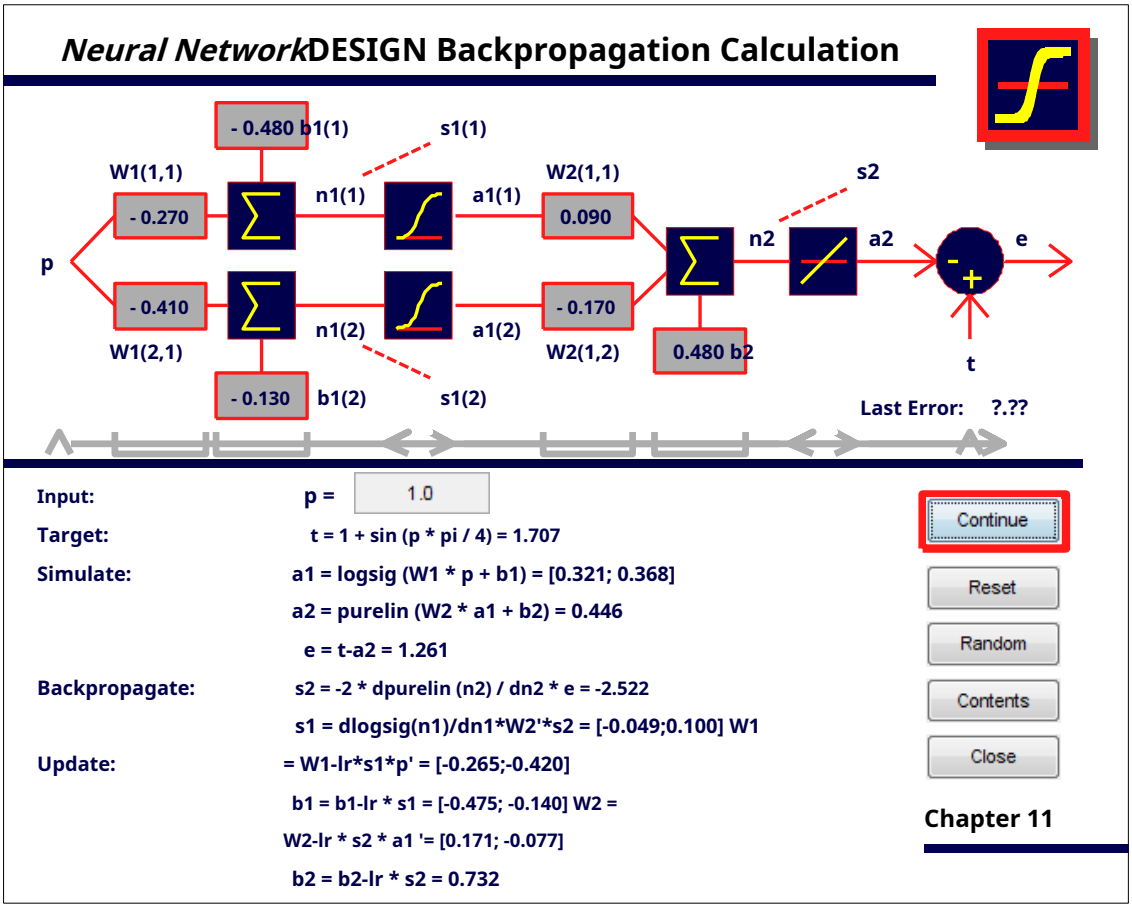
$B_2(i)$ - вектор порогових значень (bias) нейронів першого шару, i = 1,2; $У_2$- Порогове значення (bias) нейрона другого шару; $N_1(i)$ - вектор виходів першого шару, i = 1,2; N_2- Вихід другого шару;

$A_1(i)$ – вектор вихідних сигналів першого шару після виконання функції активації (сигмоїди), i = 1,2;

A_2-вихід другого шару після виконання функції активації (лінійної);

Lr-коефіцієнт навчання.

Нехай вхідний сигнал P = 1.0, а бажаний вихід t = 1 + sin (p * - / 4) = 1,707 (рисунок 3.9):



Малюнок 3.9. Demo приклад роботи алгоритму *Backpropagation*

3. Дослідження градієнтних алгоритмів навчання нейронних мереж

Алгоритми навчання, зазвичай, функціонують покроково; і ці кроки прийнято називати *епохами* або *циклами.* На кожному циклі на вхід мережі послідовно подаються всі елементи навчальної послідовності, потім обчислюються вихідні значення мережі, порівнюються з цільовими та обчислюється функціонал помилки. Значення функціоналу, а також його градієнта використовуються для коригування ваги та зсувів, після чого всі дії повторюються. Початкові значення ваг і зміщень вибираються випадковим чином, а процес навчання припиняється, коли виконано певну кількість циклів або коли помилка досягне деякого малого значення або перестане зменшуватися [54].

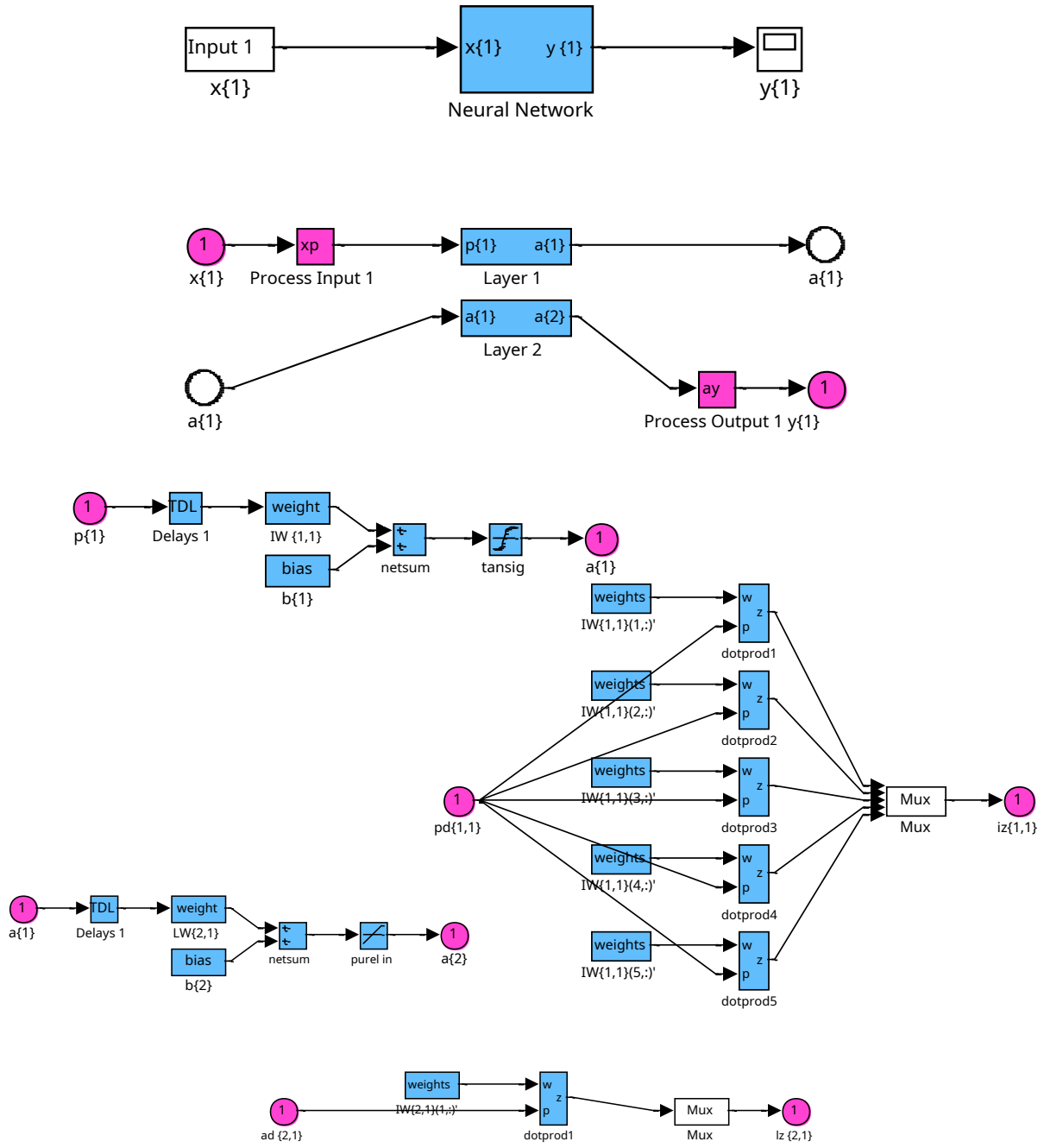
За такої формалізації завдання навчання передбачаються відомими бажані (цільові) реакції мережі на вхідні сигнали, що асоціюється з присутністю вчителя, а тому такий процес навчання називають *навчанням із учителем.* Для деяких типів нейронних мереж завдання цільового сигналу не потрібне, і в цьому випадку процес навчання називають *навчанням без вчителя*.

АлгоритмGD. АлгоритмGD, або алгоритм градієнтного спуску, використовується для такого коригування ваг і зсувів, щоб мінімізувати функціонал помилки, тобто забезпечити рух поверхнею функціоналу в напрямку, протилежному градієнту функціонала за параметрами, що настроюються.

net = newff([-1 1;-1 1],[5,1],{'tansig','purelin'},'traingd');% створення двох-шаровий НС з прямої передачі сигналу з сигмоїдальним та лінійним шарами для навчання її на основі методу зворотного поширення помилки

gensim(net)%

Структурну схему моделі нейронної мережі показано на рис 3.10.



Малюнок 3.10. Структурна схема моделі нейронної мережі

net.biases{1,1}.learnFcn='learngd';% Завдання імені функції налаштування learnFcn, що відповідає алгоритму градієнтного спуску, в даному випадку це M-функція learngd

net.biases{2,1}.learnFcn='learngd';

net.layerWeights{2,1}.learnFcn='learngd';

net.inputWeights{1,1}.learnFcn='learngd'; net.layerWeights{2,1}.learnParam.lr = 0.2;% встановлення параметра швидкості налаштування

x=-1:0.1:1;% Завдання безлічі входів

y=x;

p = [x; y];

p=num2cell(p,1);% оскільки використовується послідовний спосіб навчання, перетворимо масив входів на масив осередків

t = sin (x. ^ 2). / cos (y. 2?);% завдання безлічі цілей **t=num2cell(t,1);**% перетворення масиву цілей у масиви осередків **net.adaptParam.passes=300;**% число проходів при послідовному навчанні-

ні

tic, [net, a, e] = adapt (net, p, t); toc% налаштування параметрів НС, використовуючи процедуру адаптації

Elapsed time is 18.555497 seconds.

a = sim (net, p)% моделювання СР для перевірки якості навчання a =

[1.2764] [1.1352] [0.8998] [0.5909] [0.2899] [0.0750] [-0.0316] [-0.0387] [0.0315] [0.1196] [0.1581] [0.1359] [0.1359] [0.2564] [0.4907] [0.8103] [1.2054] [1.6578]

mse(e)% середньоквадратична помилка

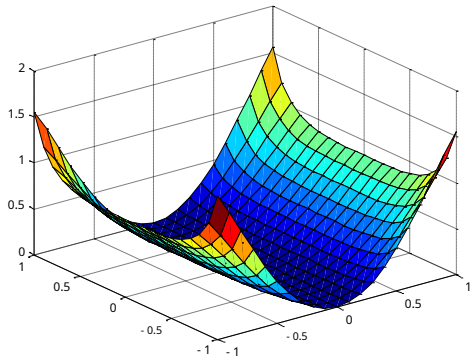
ans =

0.0334

[x, y] = meshgrid (x, y);

z = sin (x. 2?). / cos (y. 2?);

surf(x,y,z)% побудова поверхні цільової функції

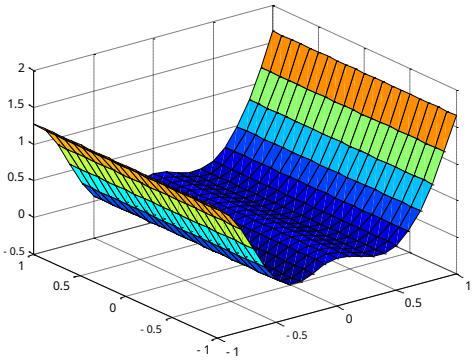


Малюнок 3.11. Графік поверхні цільової функції

a=cat(1,a{:});% перетворення масиву осередків результату моделювання НС в масив чисел

[a]=meshgrid(a);

surf(x,y,a)% побудова поверхні функції апроксимованої нейронною мережею



Малюнок 3.12. Графік поверхні функції нейронною мережею, що апроксимується.

*Групове навчання.*Для навчання мережі на основі алгоритмуGD необхідно використовувати М-функцію traingd замість функції налаштування learngd. У цьому випадку немає потреби задавати індивідуальні функції навчання для ваг та зміщень, а достатньо вказати одну навчальну функцію для всієї мережі.

net = newff([-1 1;-1 1],[5,1],{'tansig','purelin'},'traingd');% знову створимо ту ж двошарову нейронну мережу прямої передачі сигналу з сигмоїдальним та лінійним шарами для навчання за методом зворотного розповсюдження помилки

net.trainParam.show = 50;%show – інтервал виведення інформації

net.trainParam.lr = 0.05;%lr – параметр швидкості налаштування

net.trainParam.goal = 1e-005;%goal – граничне значення критерію навчання

ня

x=-1:0.1:1;

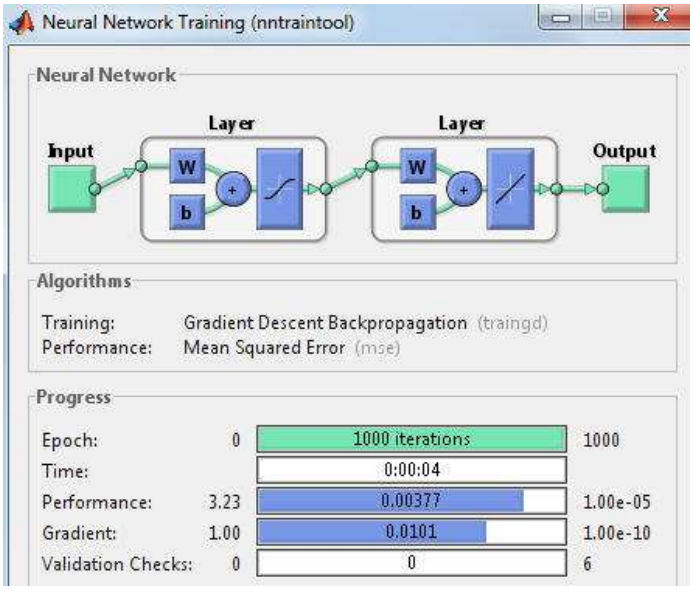
y=x;

p = [x; y];

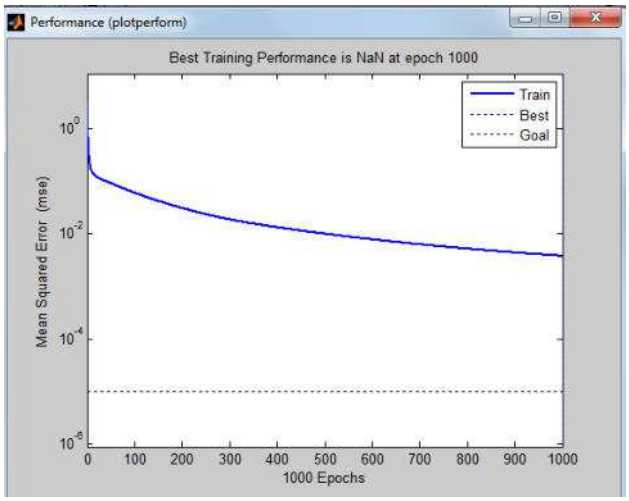
t = sin (x. ^ 2). / cos (y. 2?); tic, net =

train (net, p, t); toc Elapsed time is

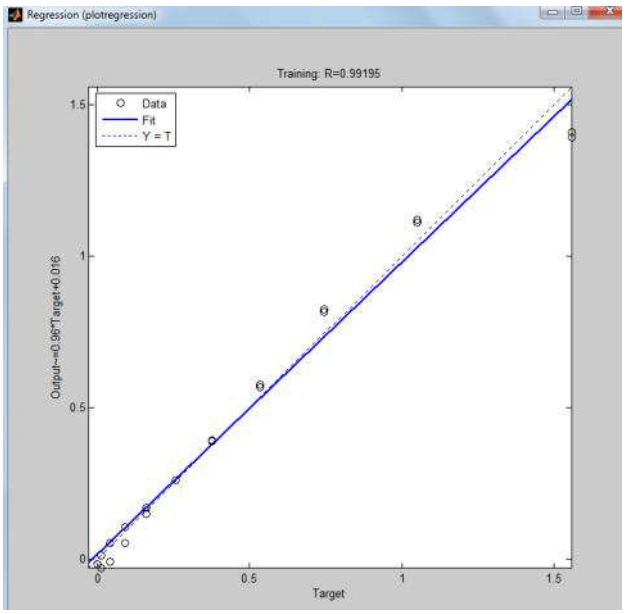
4.796924 seconds.



Малюнок 3.13. Вікно навчання нейронної мережі



Малюнок 3.14. Графік зміни помилки в залежності від кількості виконаних циклів навчання



Малюнок 3.15. Вікно лінійної регресії між виходом НС та цілями

a = sim (net, p)

a =

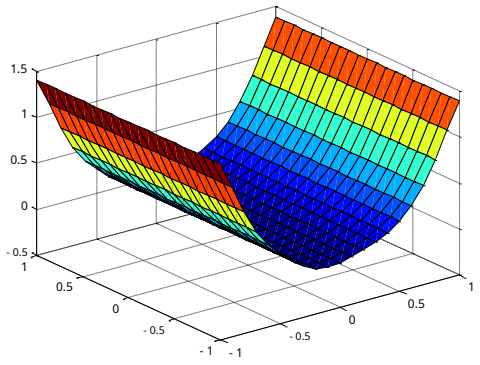
1.4087 1.1199 0.8169 0.5667 0.3896 0.2606 0.1488 0.0524 -0.0088 -0.0281 - 0.0165 0.0132 0.0542 0.1057 0.1718 0.2621 0.3915 0.5765 0.8232 1.1106 1.3917 **e=ta**

0.1487 -0.0695 -0.0723 -0.0333 -0.0132 -0.0052 0.0126 0.0378 0.0488 0.0381 0.0165 -0.0032 -0.0142 -0.015 -0.015 -0.015

mse(e)

ans =

0.0038



Малюнок 3.16. Графік поверхні функції нейронною мережею, що апроксимується.

Більш ретельно ознайомитися з методом градієнтного спуску можна за допомогою демонстраційної програми nnd12sd1, яка ілюструє роботу алгоритму GD.

АлгоритмGDM.АлгоритмGDM, або алгоритм градієнтного спуску з обуренням, призначений для налаштування та навчання мереж прямої передачі. Цей алгоритм дозволяє долати локальні нерівності поверхні помилки та не зупинятися у локальних мінімумах. З урахуванням обурення метод зворотного розповсюдження помилки реалізує наступне співвідношення для збільшення вектора параметрів, що настроюються:

$$-w\text{-}_k mc\text{-}w_{k\text{-}1} - (1\text{-}mc)lrq, \quad (3.1)$$

де Δw*k*- Збільшення вектора ваг;*транспорт*- Параметр обурення;*lr*- Параметр скорості навчання;*gk*- Вектор градієнта функціоналу помилки на*k*-ї ітерації.

Якщо параметр обурення дорівнює 0, то зміна вектора параметрів, що настроюються, визначається тільки градієнтом, якщо параметр дорівнює 1, то поточне збільшення дорівнює попередньому як за величиною, так і за напрямом.

net = newff([-1 1;-1 1],[5,1],{'tansig','purelin'},'traingdm');

net.biases{1,1}.learnFcn='learngdm';% Завдання імені функції налаштування learnFcn, що відповідає алгоритму градієнтного спуску з обуренням - M-функція learngdm

net.biases{2,1}.learnFcn='learngdm';

net.layerWeights{2,1}.learnFcn='learngdm';

net.inputWeights{1,1}.learnFcn='learngdm';

net.layerWeights{2,1}.learnParam.lr=0.2;% збільшимо значення параметра швидкості навчання до 0.2

x=-1:0.1:1;

y=x;

p = [x; y];

p=num2cell(p,1);

t = sin (x. ^ 2). / cos (y.

2?); t=num2cell(t,1);

net.adaptParam.passes=300;% значення проходів **tic,**

[net, a, e] = adapt (net, p, t); toc Elapsed time is

19.187137 seconds. **a**

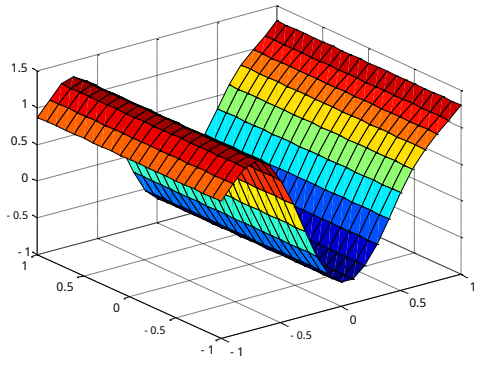
a =

[0.8683] [1.0634] [1.2363] [1.2927] [1.1903] [0.9339] [0.5665] [0.1546] [-0.2283] [-0.5175] [-0.6697] [-0.69] [0.3291] [0.6065] [0.8297] [0.9997] [1.1413] [1.3017]

mse(e)

ans =

0.2196



Малюнок 3.17. Графік поверхні функції нейронною мережею, що апроксимується.

Результати апроксимації виявилися гіршими за результати роботи НР з алгоритмом навчання GD і набагато довше.

*Групове навчання.*Альтернативою послідовної адаптації є групове навчання, яке ґрунтується на застосуванні функції train. У цьому режимі параметри мережі модифікуються тільки після того, як реалізовано всю навчальну множину, і градієнти, розраховані для кожного елемента множини, підсумовуються, щоб визначити збільшення настроюваних параметрів.

Для навчання мережі на основі алгоритму GDM необхідно використовувати М-функцію traingdm замість функції налаштування learngdm. Відмінність цих двох функцій ось у чому. Алгоритм функції traingdm підсумовує градієнти, розраховані на кожному циклі навчання, а параметри модифікуються

тільки після того, як усі навчальні дані будуть представлені. Якщо проведено *N* циклів навчання та для функціоналу помилки виявилося виконаним умова $J_N \geq 1.04 J_{N-1}$, то параметр обурення mc слід встановити 0.

net = newff([-1 1;-1 1],[5,1],{'tansig','purelin'},'traingdm');

net.trainParam.epochs=1000;

net.trainParam.goal=1e-5;

net.trainParam.lr=0.05;

net.trainParam.mc = 0.9;% у порівнянні з функцієютraingd тут додається тільки 1 параметр обурення mc

net.trainParam.show=50;

x=-1:0.1:1;

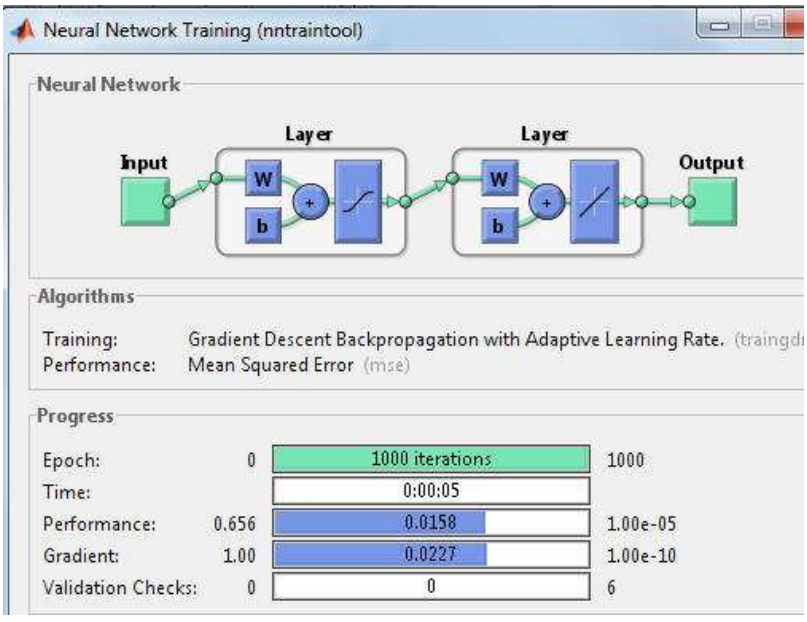
y=x;

p = [x; y];

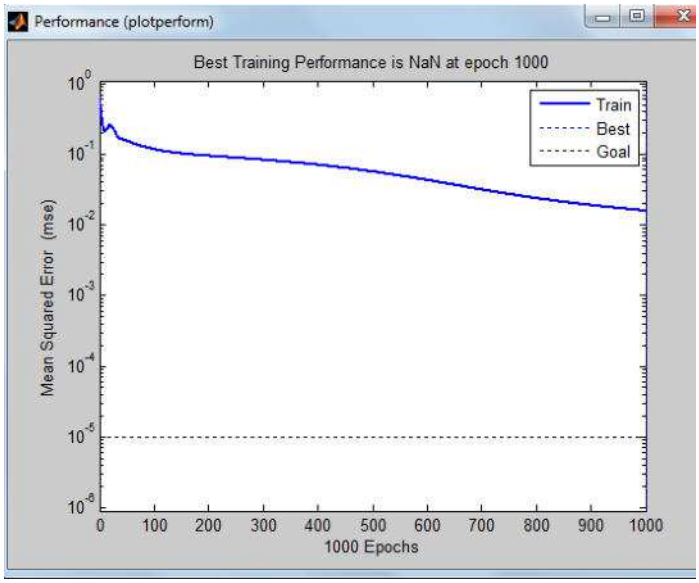
t = sin (x. 2). / cos (y. 2); tic, net =

train (net, p, t); toc Elapsed time is

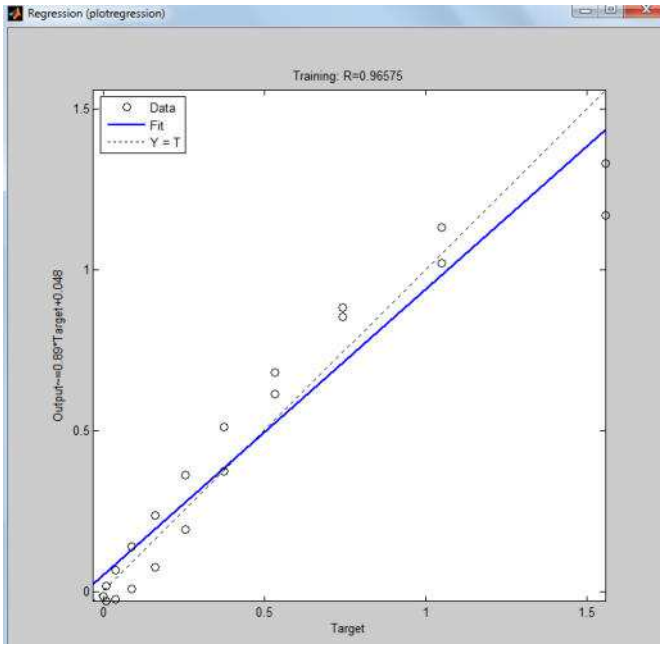
5.533868 seconds.



Малюнок 3.18. Вікно навчання нейронної мережі



Малюнок 3.19. Графік зміни помилки навчання в залежності від кількості виконаних циклів навчання



Малюнок 3.20. Вікно лінійної регресії між виходом НС та цілями

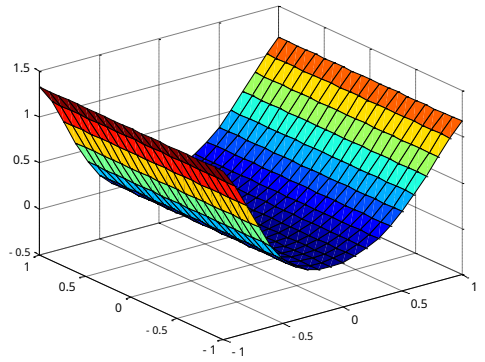
a = sim (net, p);

e=ta;

mse(e)

ans =

0.0158



Малюнок 3.21. Графік поверхні функції нейронною мережею, що апроксимується.

Ретельніше ознайомитися з методом градієнтного спуску з обуренням можна за допомогою демонстраційної програми nnd12mo, яка ілюструє роботу алгоритму GDM.

Практика застосування описаних вище алгоритмів градієнтного спуску показує, що ці алгоритми надто повільні для вирішення реальних завдань. Нижче обговорюються алгоритми групового навчання, які сходяться в десятки та сотні разів швидше. Нижче наведено 2 різновиди таких алгоритмів: один заснований на стратегії вибору параметра швидкості налаштування та реалізований у вигляді алгоритму GDA, інший – на стратегії вибору кроку за допомогою порогового алгоритму зворотного розповсюдження помилки та реалізований у вигляді алгоритму Rprop.

АлгоритмGDA.АлгоритмGDA, або алгоритм градієнтного спуску з вибором параметра швидкості налаштування, використовує евристичну стратегію зміни цього параметра у процесі навчання.

Ця стратегія полягає у наступному. Обчислюються вихід та похибка ініціалізованої нейронної мережі. Потім на кожному циклі навчання обчислюються нові значення параметрів, що настроюються, і нові значення виходів і похибок. Якщо відношення нового значення похибки до колишнього перевищує величину max_perf_inc (за замовчуванням 1.04), то нові значення параметрів, що налаштовуються, до уваги не беруться. При цьому параметр швидкості налаштування зменшується з коефіцієнтом lr_dec (за умовчанням 0.7). Якщо нова похибка менша

колишньої, то параметр швидкості налаштування збільшується з коефіцієнтом lr_inc (за умовчанням 1.05).

Ця стратегія сприяє збільшенню швидкості та скорочення тривалості навчання.

Функція traingda характеризується наступними параметрами, заданими за умовчанням:

net.trainParam

show: 50

showWindow: 1

showCommandLine: 0

epochs: 300

time: Inf

goal: 1.0000e-005

max_fail: 6

lr: 0.0500

lr_inc: 1.0500

lr_dec: 0.7000

max_perf_inc: 1.0400

min_grad: 1.0000e-010

mc: 0.9000

Алгоритм GDA у поєднанні з алгоритмом GD визначає функцію навчання traingda, а у поєднанні з алгоритмом GDM – функцію навчання traingdx.

Знову звернемося до тієї ж нейронної мережі, але використовуватимемо функцію навчання traingda:

```
net = newff([-1 1;-1 1],[5,1],{'tansig','purelin'},'traingda');
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
net. trainParam.lr=0.05;
net.trainParam.mc=0.9;
net.trainParam.show=50;
```

x=-1:0.1:1;

y=x;

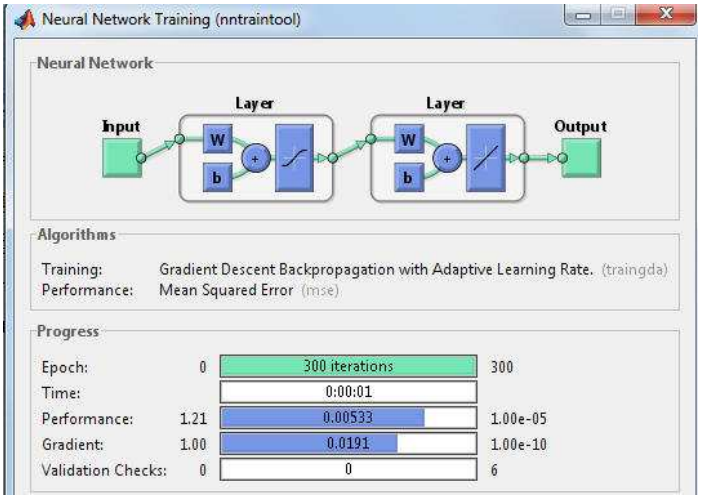
p = [x; y];

t = sin (x. 2). / cos (y. 2); tic, net =

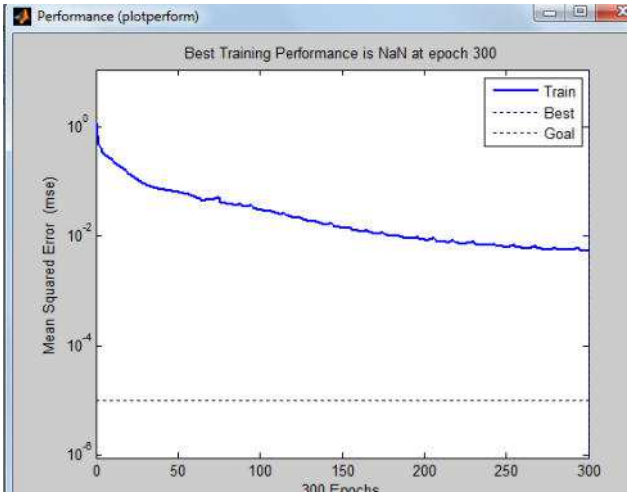
train (net, p, t); toc Elapsed time is

1.706964 seconds.

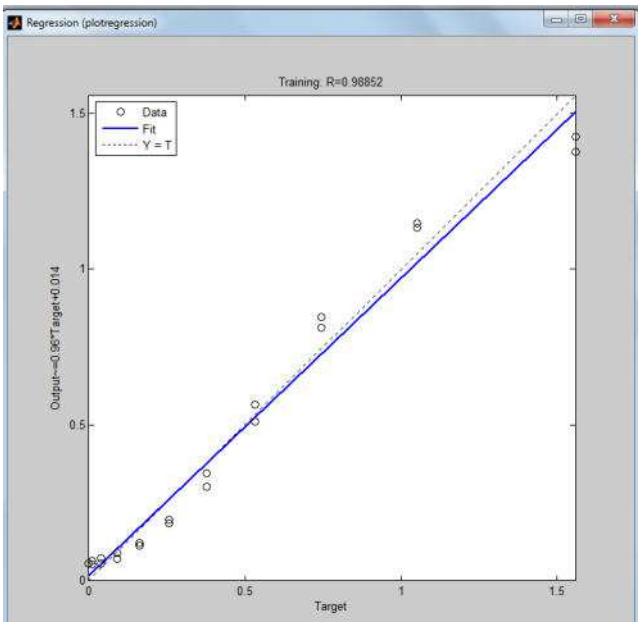
На малюнку 3.22 наведено графік зміни помилки навчання залежно від кількості виконаних циклів.



Малюнок 3.22. Вікно навчання нейронної мережі



Малюнок 3.23. Графік зміни помилки навчання в залежності від кількості виконаних циклів навчання



Малюнок 3.24. Вікно лінійної регресії між виходом НС та цілями

a = sim (net, p);

e=ta;

mse(e)

0.0053

Неважко помітити, що кількість циклів навчання порівняно з попереднім прикладом скоротилася практично в 3 рази за зменшення похибки навчання.

Демонстраційна програма nnd12vl ілюструє продуктивність алгоритму зі змінним параметром швидкості налаштування.

АлгоритмRprop.АлгоритмRprop, або пороговий алгоритм зворотного розповсюдження помилки, реалізує наступну евристичну стратегію зміни кроку збільшення параметрів для багатошарових нейронних мереж.

Багатошарові мережі зазвичай використовують сигмоїдні функції активації у прихованих шарах. Ці функції належать до класу функцій зі стисненим відображенням, оскільки вони відображають нескінченний діапазон значень аргументу в кінцевий діапазон значень функції. Сигмоїдні функції характеризуються тим, що їхній нахил наближається до нуля, коли значення входу нейрона суттєво зростають. Наслідком цього є те, що при використанні найшвидшого методу

спуску величина градієнта стає малою і призводить до малих змін параметрів, що настроюються, навіть якщо вони далекі від оптимальних значень.

Мета порогового алгоритму зворотного поширення помилки Rprop (Resilient propagation) полягає в тому, щоб підвищити чутливість методу за більших значень входу функції активації. У цьому випадку замість значень самих виробних використовується лише їхній знак.

Значення збільшення для кожного параметра, що налаштовується, збільшується з коефіцієнтом delt_inc (за замовчуванням 1.2) щоразу, коли похідна функціонала помилки за даним параметром зберігає знак для двох послідовних ітерацій. Значення збільшення зменшується з коефіцієнтом delt_dec (за замовчуванням 0.5) щоразу, коли похідна функціонала помилки за даним параметром змінює знак порівняно з попередньою ітерацією. Якщо похідна дорівнює 0, то збільшення залишається незмінним. Оскільки за умовчанням коефіцієнт збільшення збільшення становить 20%, а коефіцієнт зменшення – 50%, то у разі поперемінного збільшення та зменшення загальна тенденція буде спрямована на зменшення кроку зміни параметра. Якщо параметр від ітерації до ітерації змінюється в одному напрямку, крок зміни буде постійно зростати.

Алгоритм Rprop визначає функцію навчання trainrp.

Функція trainrp характеризується наступними параметрами, заданими за умовчанням:

net.trainParam

ans =

show: 25

showWindow: 1

showCommandLine: 0

epochs: 1000

time: Inf

goal: 0

max_fail: 6

min_grad: 1.0000e-010

delt_inc: 1.2000

delt_dec: 0.5000

delta0: 0.0700

deltamax: 50

Тут epochs – максимальна кількість циклів навчання; show – інтервал виведення інформації, виміряний у циклах; goal – граничне значення критерію навчання; time - граничний час навчання; min_grad – мінімальне значення градієнта; max_fail – максимально допустимий рівень перевищення помилки контрольного підмножини порівняно з навчальним; delt_inc – коефіцієнт збільшення кроку налаштування; delt_dec - коефіцієнт зменшення кроку налаштування; delta0 - початкове значення кроку налаштування; deltamax – максимальне значення кроку налаштування.

Знову звернемося до мережі, показаної на малюнку 3.10, але використовуватимемо функцію навчання trainrp:

net=newff([-1 2;0 5],[3,1],{'tansig','purelin'},'trainrp');

Встановимо такі значення цих параметрів:

net.trainParam.show = 10; net.trainParam.epochs =

300; net.trainParam.goal = 1e-5; x=-1:0.1:1;

y=x;

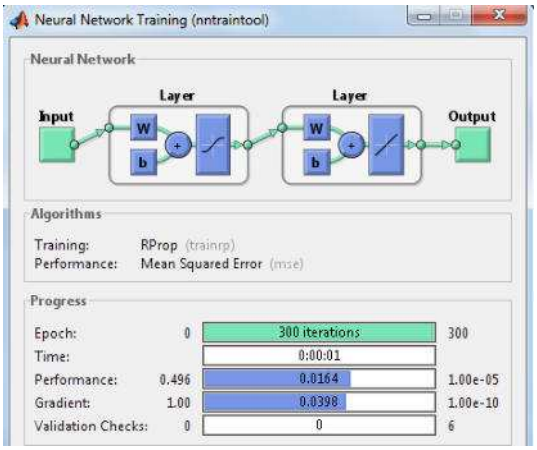
p = [x; y];

t = sin (x. 2). / cos (y. 2); tic, net =

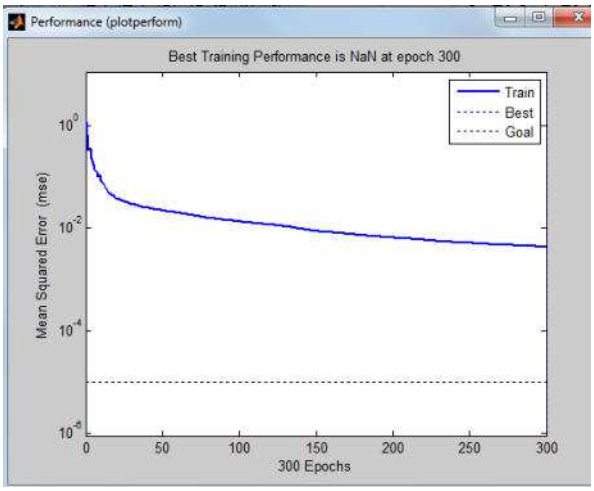
train (net, p, t); toc Elapsed time is

1.928695 seconds.

На малюнку 3.26 наведено графік зміни помилки навчання залежно від кількості виконаних циклів навчання.



Малюнок 3.26. Вікно навчання нейронної мережі



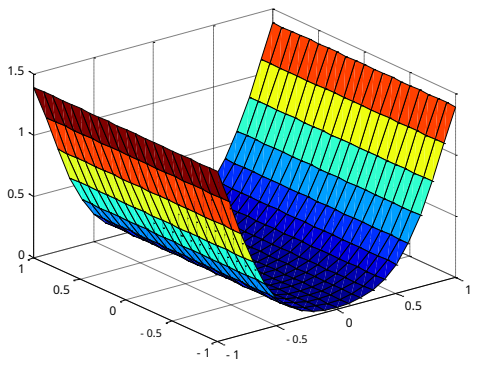
Малюнок 3.26. Графік зміни помилки навчання в залежності від числа виповних циклів навчання

a = sim (net, p);

e=ta;

mse(e)

0.0043



Малюнок 3.27. Графік поверхні функції нейронною мережею, що апроксимується.

Неважко помітити, що кількість циклів навчання порівняно з алгоритмом GDA скоротилася практично ще в 3 рази і становила по відношенню до алгоритму GD значення, близьке до 9.

Програма роботи та методичні вказівки

При заданому викладачем значенні p розрахуйте в командному вікні MatLab пряме поширення вхідного сигналу, зворотне поширення помилки та зміна ваг на першій ітерації алгоритму *Backpropagation*. Порівняйте з результатами, отриманими в *Backpropagation Calculation*.

Контрольні питання

1. Яким алгоритмом навчають багатошарові нейронні мережі?
2. З яких основних етапів складається алгоритм зворотного розповсюдження помилки?
3. Чому алгоритм зворотного розповсюдження помилки відноситься до класу алгоритмів градієнтного спуску?
4. Як впливає функція приналежності на правило зміни ваг у обратному алгоритмі поширення помилки?

Зміст звіту

- мета роботи;
- короткий опис дій щодо пунктів; графіки
- за всіма пунктами програми; розрахунки
- та висновки по роботі.