ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Ганчо Венков Богдан Гилев

ПРИЛОЖЕНИЕ НА НЕВРОННИТЕ МРЕЖИ В КОМПЮТЪРНОТО МОДЕЛИРАНЕ

София, 2009

Учебникът е предназначен за студентите от специалност “Приложна математика” на ФПМИ при ТУ- София и е написан за курса “Цифрово моделиране на динамични системи” от учебния план на тази специалност. Той е съставен от две глави и приложение.

Първият раздел е озаглавен “Приложение на невронните мрежи в моделирането”. В него са разгледани класически архитектури на невронни мрежи с еднопосочно обработване на данните с приложение в апроксимирането на функции и в прогнозирането на времеви редове и рекурентни невронни мрежи с приложение в моделирането на динамични системи.

Вторият раздел е озаглавен “Невронни мрежи с радиални базисни функции, самоорганизиращи се невронни мрежи и невронни мрежи на Хопфилд”. Тук са разгледани невронни мрежи с радиални функции с приложение в задачи за апроксимиране на нелинейни изображения. Дадени са два невронни класификатора на вектори и невронна мрежа за самоорганизиращо се изображение. Накрая е разгледана рекурентната невронна мрежа на Хопфилд.

В приложението са описани накратко използваните програми от MATLAB.

Въведение

Интензивните изследвания в областта на невронните мрежи са свързани с идеята за създаване на интелигентна система за паралелни изчисления, която да работи като мозъка. Все още това е една научна хипотеза, но постигнатите конкретни резултати в различни приложни области са забележителни. Например в [4] е направен преглед на приложения на невронните мрежи, които основно са в следните направления: в управлението на системи и по-специално в авиацията, в банковата система, във военните системи, в електронните системи за разпознаване на образи, в роботиката и медицината и т.н.

Невронните мрежи притежават редица характеристики, които определят множеството им приложения в компютърното моделиране. Най-съществените от тях са:

- Невронните мрежи са свързани с изследвания и моделиране в областта на нелинейните технически системи. Това се дължи на възможността им да апроксимират нелинейни изображения и да моделират широк клас динамични системи;
- Разпаралеляване на изчислителния процес. Невронните мрежи имат паралелна структура. Компютърната реализация на тази паралелна структура осигурява голямо бързодействие в сравнение с конвенционалните изчислителни схеми;
- Настройването на параметрите на невронните модели става с обучение , което се извършва с данни, получени от експерименти с реален обект. Получената невронна мрежа може да обобщава познанията си в случаите, когато на входа й се подаде непознат за нея процес от обучаващото множество от данни. Невронните мрежи могат да бъдат обучени и в реално време чрез адаптивни алгоритми.

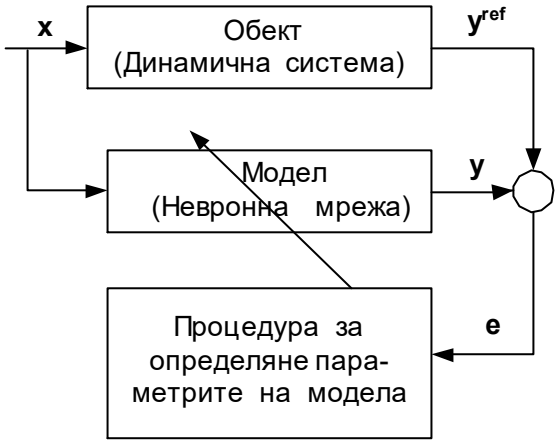
В литературата по невронни мрежи няма общоприето определение за невронна мрежа, но повечето дефиниции имат следния смисъл: система от обработващи елементи, наричани неврони, които са свързани в мрежова структура посредством връзки с тегла.

Основна характеристика на невронната мрежа е нейната архитектура. В зависимост от архитектурата си невронните мрежи се

делят основно на мрежи с еднопосочно обработване на данните и рекурентни невронни мрежи.

Използваните подходи за обучение на невронните мрежи, т.е. за определяне на тегловните коефициенти (теглата) на мрежата, са основно два: обучение на невронни мрежи с учител (с еталон) и обучение на невронни мрежи без учител (без еталон).

Типичен случай за обучение на невронна мрежа с еталон е създаването на невронен модел на реален обект (например динамична система). Нека са проведени експерименти и са направени записи на входните въздействия $\mathbf{x} = (x_1, x_2, ..., x_m)^T$ и съответните реакции на обекта $\mathbf{y}^{\text{ref}} = (y_1^{ref}, y_2^{ref}, ..., y_n^{ref})^T$, при което се генерират множество от обучаващи данни. Етапът на обучението може да се съвмести с генерирането на данните или да се реализира като самостоятелна процедура.



Фиг.1.1

При обучение на мрежата се подава входен вектор $\mathbf{x} = (x_1, x_2, ..., x_m)^T$ и се изчислява съответният изход-вектор $\mathbf{y} = (y_1, y_2, ..., y_n)^T$ и грешката $\mathbf{e} = \mathbf{y}^{ref} - \mathbf{y}$. Целта е да се изберат теглата на невронната мрежа (невронния модел) така, че изходът на модела $\mathbf{y}$ да апроксимира (например в средноквадратичен смисъл) реакцията на реалния обект $\mathbf{y}^{ref}$. За целта се използва подходяща оптимизационна процедура. Най-често прилаганите процедури са делта-правило (например за персептронни мрежи) и методи от градиентен тип за невронни мрежи с гладки активиращи функции. Коригирането на теглата може да се извършва за всеки комплект от данни, т.е. последователно или пакетно за целия комплект от данни. Резултатът

от тази процедура може да се прилага в две направления:

а) получаване на модел на динамичната система, който е близък до моделирания обект за зададено множество от входни въздействия;

б) получаване на модел, който предсказва състоянието на обекта само за няколко времеви стъпки напред, т.е. реализиране на предиктор.

Типични мрежи, обучавани без еталон, са самоорганизиращите се невронни мрежи. При тях обучението става чрез евристични правила, без да се използва еталонен процес и грешка между еталонния и реализирания от невронната мрежа процес. Примери за такъв подход са правилата на Кохонен и на Хеб.

След обучението на невронната мрежа тя може да бъде приложена за реална обработка на данни (за симулиране). В етапа на симулиране на входа на невронната мрежа обикновено се задават данни от обучаващото множество, различни от тези, които са използвани при самото обучение.

Учебникът се състои от два раздела и приложение. Първият раздел е озаглавен “Приложение на невронните мрежи в моделирането” и в него са разгледани класически архитектури на невронни мрежи с еднопосочно обработване на данните и рекурентни невронни мрежи, като се прилагат методи за обучение с еталон (с учител). Вторият раздел е озаглавен “Невронни мрежи с радиални базисни функции, самоорганизиращи се невронни мрежи и невронни мрежи на Хопфилд”, като са използвани и двата похода за обучение. В приложенията са описани накратко някои програми от MATLAB.

Глава I. Приложение на невронните мрежи в моделирането

В тази глава ще бъдат разгледани две основни архитектури на невронни мрежи:

Невронни мрежи с еднопосочно обработване на данните. Това са класически невронни мрежи, които се описват с нелинейни изображения. Основните им приложения са за решаване на апроксимационни задачи и за прогнозиране на времеви редове.

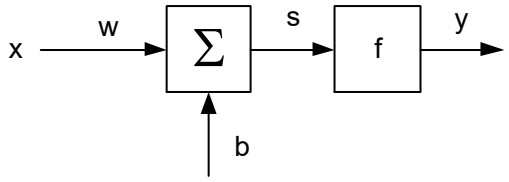
Рекурентни невронни мрежи. Тези невронни мрежи се описват със системи рекурентни уравнения или със системи диференциални уравнения. Основните им приложения са в моделирането на динамични системи.

1.1. Архитектура и основни теоретични резултати за невронните мрежи с еднопосочно обработване на данните

1.1.1. Неврони и активиращи функции

Класическият неврон се разглежда като изчислителен аналог на невроните на мозъка. В литературата се среща следното описание на неговите функции.

Невронът (или възелът от мрежата) е елементарен обработващ елемент с определен брой входни величини, които сумира умножени с тегловните коефициенти на връзките. Резултатът се използва като аргумент на активираща функция, която определя стойността на изхода на неврона (фиг.1.2).



фиг.1.2

Изходът на неврона се определя по следния начин:

$y = f(wx + b)$, където

f е активиращата функция;

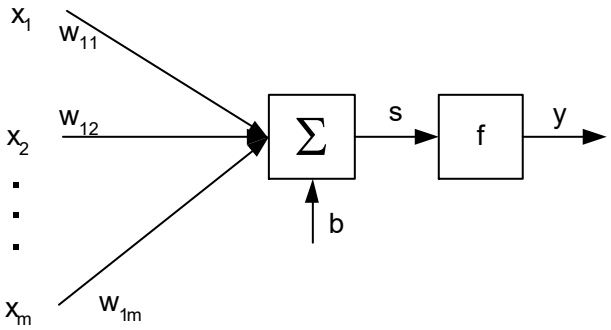
y - изход на неврона;

x - вход на неврона;

w - тегло;

b - свободно тегло (отместване).

Неврон с векторен вход е даден на фиг. 1.3. Този неврон има m входа и един изход.



фиг.1.3

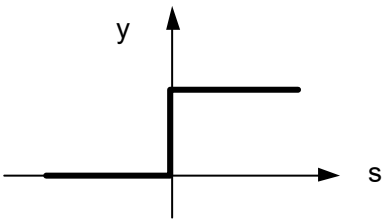
Функцията, реализирана от неврона, е:

$$\mathbf{s} = \left(w_{1,1} w_{1,2} \ldots w_{1,m}\right) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} + b$$

$$y = f(\mathbf{s}).$$

Активиращите функции са твърде разнообразни. Тези функции са линейни, нелинейни , а също така гладки или с прекъсване. Тук са дадени някои от най- използваните активиращи функции.

Прагова функция:

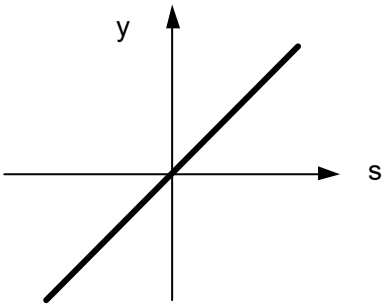


фиг.1.4

$$y = \begin{cases} 0 & за\ s < 0 \\ 1 & за\ s \geq 0 \end{cases}$$

Тъй като тази функция ограничава изхода на неврона на две стойности 0 или 1, то тя се прилага основно при обработване на двоични данни.

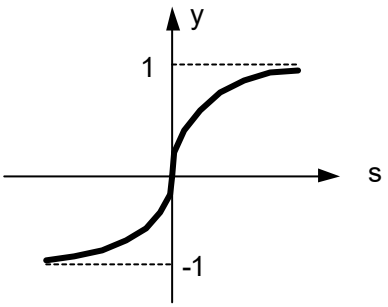
Линейна функция:



фиг.1.5

Тази функция определя изхода y на неврона от уравнението $y = s$.

Сигмоидна функция:

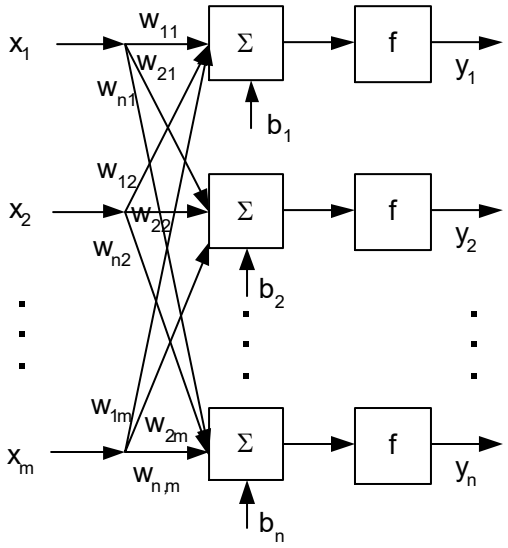


фиг.1.6

Аналитичният израз на тази функция е $y = th(s)$. Удачно е стойността й да се пресмята по формулата $y = \frac{2}{1+e^{-2s}} - 1$ и тя очевидно има важното свойство да ограничава стойността на изхода в интервала (-1, 1).

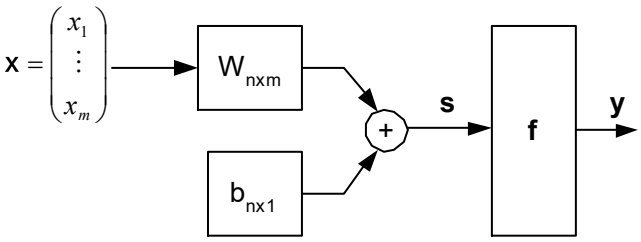
1.1.2. Архитектура на невронните мрежи

Еднослойната мрежа има следната архитектура:



фиг.1.7

или еквивалентна схема в матрична форма:



фиг.1.8

Невронната мрежа от фигура 1.8 има m входа и n изхода.

Уравненията, описващи невронния слой, са следните:

$$\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,m} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}; \qquad \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} f(s_1) \\ f(s_2) \\ \vdots \\ f(s_n) \end{pmatrix}.$$

Ако се въведат векторите и матрицата:

$$\mathbf{x} = (x_1, x_2, ..., x_m)^T,\ \mathbf{b} = (b_1, b_2, ..., b_n)^T,\ \mathbf{y} = (y_1, y_2, ..., y_n)^T,\ \mathbf{s} = (s_1, s_2, ..., s_n)^T$$

$$\mathbf{W} = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,m} \end{pmatrix},$$

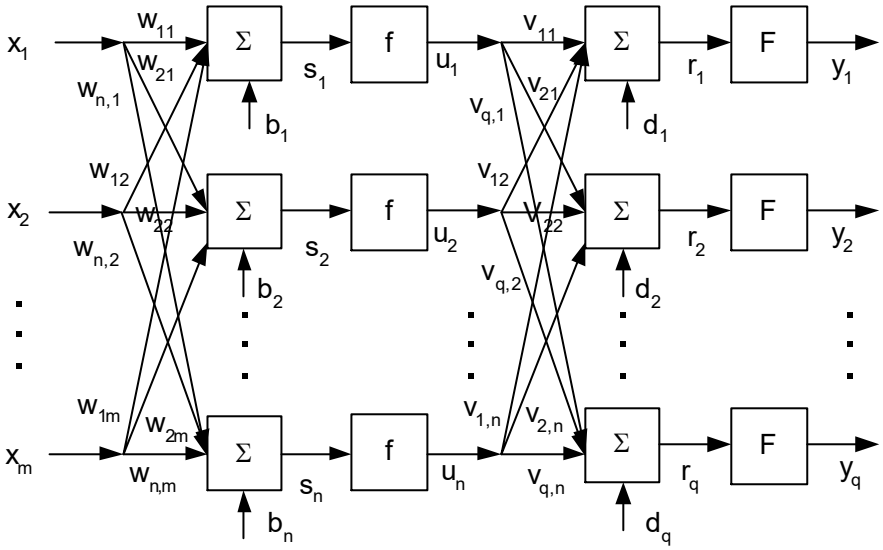
то тези уравнения се записват по следния начин:

$$\mathbf{s} = \mathbf{W}\mathbf{x} + \mathbf{b} \qquad\qquad \mathbf{y} = \mathbf{f}(\mathbf{s}),$$

където **f(s)** е векторната функция $\mathbf{f}(\mathbf{s}) = (f(s_1), f(s_2), ..., f(s_n))^T$. Следователно уравнението на слоя в матрична форма е

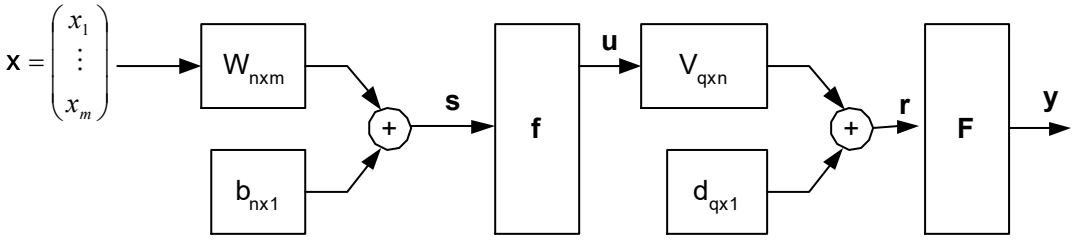
$$\mathbf{y} = \mathbf{f}(\mathbf{W}\mathbf{x} + \mathbf{b}).$$

Двуслойна мрежа. На фиг.1.9 е показана структурата на двуслойна невронна мрежа.



фиг.1.9

Тази мрежа има еквивалентната схема в матрична форма (фиг.1.10).



фиг.1.10

Вторият слой на мрежата е изходен слой и в него се определя изходният вектор $\mathbf{y}=(y_1,y_2,...,y_q)^T$ на мрежата. Изходните неврони имат активираща функция F. Първият слой се нарича скрит слой, тъй като се намира между външните входове $\mathbf{x}=(x_1,x_2,...,x_m)^T$ и изходите $\mathbf{y}=(y_1,y_2,...,y_q)^T$ на втория слой. Представената невронна мрежа е изцяло свързана, тъй като всички входове на мрежата са свързани с невроните от първи слой, а всички изходи на първия слой са свързани с всички неврони от следващия слой.

Уравненията, описващи невронната мрежа, са следните:

а)за първия слой

$$\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,m} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}; \quad \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} f(s_1) \\ f(s_1) \\ \vdots \\ f(s_n) \end{pmatrix}$$

или в матрична форма

$$\mathbf{s}=\mathbf{W}\mathbf{x}+\mathbf{b} \qquad\qquad \mathbf{u}=\mathbf{f}(\mathbf{s}),$$

където $\mathbf{f}(\mathbf{s})$ е векторната функция $\mathbf{f}(\mathbf{s})=(f(s_1),f(s_2),...,f(s_n))^T$;

б) за втория слой

$$\begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_q \end{pmatrix} = \begin{pmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,n} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,n} \\ \vdots & \vdots & & \vdots \\ v_{q,1} & v_{q,2} & \cdots & v_{q,n} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} + \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_q \end{pmatrix}; \quad \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_q \end{pmatrix} = \begin{pmatrix} F(r_1) \\ F(r_1) \\ \vdots \\ F(r_q) \end{pmatrix}$$

или в матрична форма:

$$\mathbf{r}=\mathbf{V}\mathbf{u}+\mathbf{d} \qquad\qquad \mathbf{y}=\mathbf{F}(\mathbf{r}),$$

където $\mathbf{F}(\mathbf{r})$ е векторната функция $\mathbf{F}(\mathbf{r})=(F(r_1),F(r_2),...,F(r_q))^T$. Матричният запис на тези уравнения е следният:

$$\mathbf{u}=\mathbf{f}(\mathbf{W}\mathbf{x}+\mathbf{b})$$
$$\mathbf{y}=\mathbf{F}(\mathbf{V}\mathbf{u}+\mathbf{d})$$

или зависимостта вход- изход е

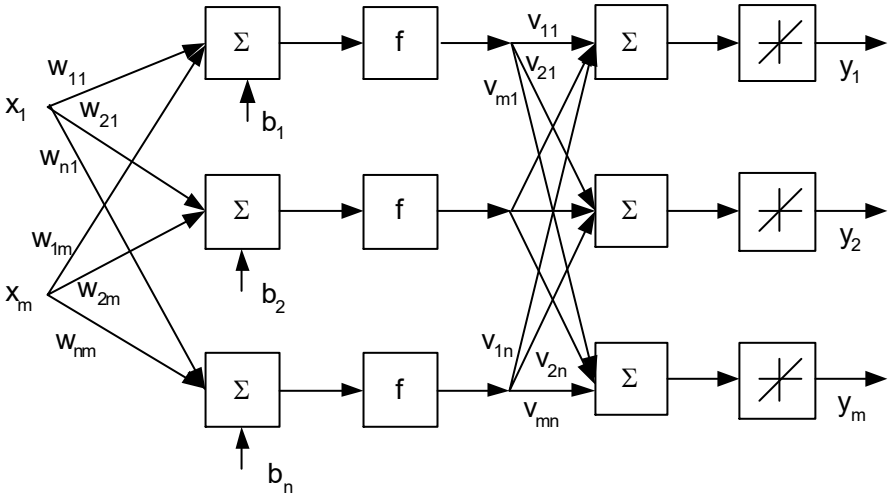
$$\mathbf{y}=\mathbf{F}(\mathbf{V}\mathbf{f}(\mathbf{W}\mathbf{x}+\mathbf{b})+\mathbf{d}).$$

Следва да се отбележи, че тегловният коефициент на връзката между трети вход x_3 и първи неврон от първи слой е означен с $w_{1,3}$, а съответното отместване (свободно тегло) на неврона е b_1. В тази мрежа изходите на всеки слой са входове на следващия и няма

обратни връзки. Определянето на изходните променливи $\mathbf{y}$ става, като се извършат изчисленията от входа към изхода на мрежата и затова тези мрежи се наричат мрежи с еднопосочно обработване на данните (feedforward neural networks). Матриците $\mathbf{W}$ и $\mathbf{V}$ и векторите $\mathbf{b}$ и $\mathbf{d}$ съдържат параметрите на невронната мрежа (отмествания и тегловни коефициенти на невроните), които могат да се настройват. Задачата за определяне на стойностите на параметрите се нарича обучение на невронната мрежа и най-често представлява стандартна параметрична оптимизационна задача.

1.1.3. Основни теоретични резултати за невронни мрежи с гладки активиращи функции

Невронните мрежи с еднопосочно обработване на данните и гладки активиращи функции се използват основно за решаване на апроксимационни задачи. Затова тук ще бъде цитирана една основна теорема, доказана в [2]. Съгласно тази теорема структура от класа на невронна мрежа, изградена от един скрит слой с ограничена и монотонно растяща активираща функция и линеен изходен слой, е в състояние да апроксимира всяко непрекъснато изображение (в частност функция). Условието е да са налице достатъчен брой неврони в скрития слой, т.е. тя не определя колко неврона са необходими за постигането на зададена точност на апроксимацията.



Фиг. 1.11

Нека е дадена невронна мрежа със следната архитектура (фиг.1.11). Описанието на тази мрежа е частен случай на предходната

двуслойна мрежа, тъй като последният й слой е линеен. Следователно за тази мрежа може да се запише следното матрично уравнение:

$$\mathbf{y} = \mathbf{V}\,\mathbf{f}(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

където

$$\mathbf{x} = (x_1, x_2, ..., x_m)^T,\ \mathbf{b} = (b_1, b_2, ..., b_n)^T,\ \mathbf{y} = (y_1, y_2, ..., y_m)^T,$$

$$\mathbf{W} = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,m} \end{pmatrix},\ \mathbf{V} = \begin{pmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,n} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,n} \\ \vdots & \vdots & & \vdots \\ v_{m,1} & v_{m,2} & \cdots & v_{m,n} \end{pmatrix}$$

и $\mathbf{f}(\mathbf{s}) = (f(s_1), f(s_2), ..., f(s_n))^T$ е векторна функция.

Теорема 1. Нека K е компактно подмножество на R^m и $\mathbf{F}: K \to R^m$ е непрекъснато изображение. Тогава за всяко $\varepsilon > 0$ съществуват n неврона, матрици $\mathbf{W}_{n\times m}$, $\mathbf{V}_{m\times n}$ и n-мерен вектор $\mathbf{b}$ такива, че е изпълнено

$$\max_{x\in K} \left\| \mathbf{F}(x) - \mathbf{V}\mathbf{f}(\mathbf{W}\mathbf{x} + \mathbf{b}) \right\| < \varepsilon,$$

където $\mathbf{f}: R^n \to R^n$ е векторна функция, дефинирана чрез

$$\mathbf{f}\left((s_1, s_2, ..., s_n)^T\right) = \left(f(s_1), f(s_2), ..., f(s_n)\right)^T$$

и активиращата функция f е гладка.

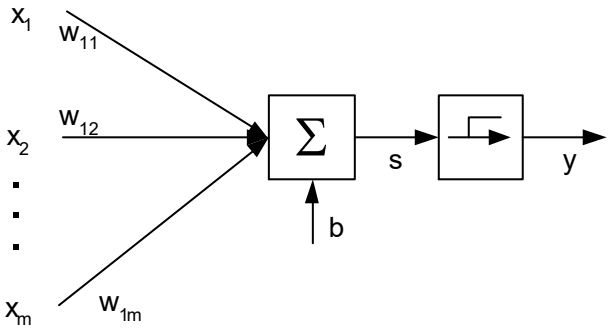
Устойчивост

Правите невронни мрежи са устойчиви, тъй като връзката между изходните и входните величини е чисто алгебрична. Следователно предикторите, реализирани с прави невронни мрежи, също са устойчиви.

1.2. Персептрони. Делта правило за обучение на еднослойна персептронна мрежа

1.2.1. Архитектура на единичен персептрон

Архитектурата на единичен персептрон с векторен вход е дадена на фиг.1.12.

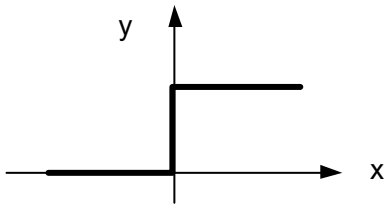


фиг.1.12

На входа на неврона постъпва векторът $\mathbf{x} = (x_1, x_2, \ldots, x_m)^T$. Тегловната матрица на неврона е $\mathbf{W} = (w_{11}, w_{12}, \ldots, w_{1m})$,а отместването е съответно b и резултатът след сумиращия блок е

$$s = \mathbf{W}\,\mathbf{x} + b\,.$$

Този резултат се преобразува от праговата активираща, функция изобразена на фиг.1.13.



фиг. 1.13

Изходът на персептрона е

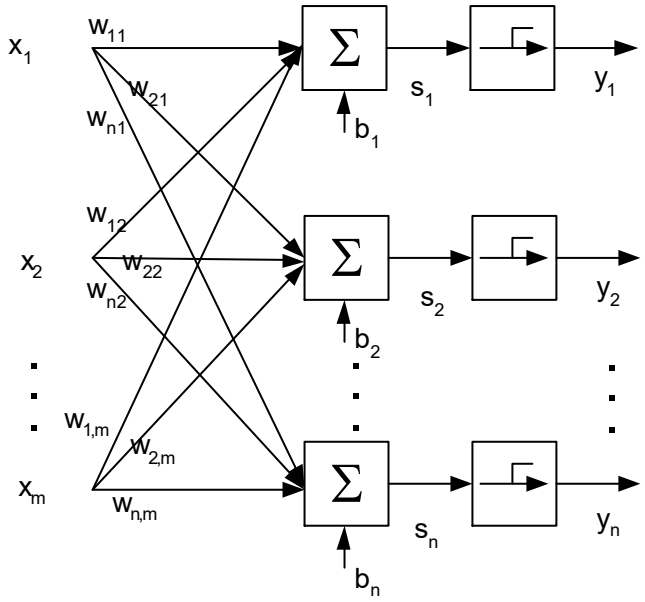
$$y = f(s)\,,$$

където $$f(s) = \begin{cases} 1, & за\ s \geq prag \\ 0, & за\ s < prag \end{cases}.$$

В този израз константата $prag$ се фиксира на някаква постоянна стойност, най-често $prag = 0$, както е показано на фиг.1.13.

1.2.2. Еднослойна персептронна мрежа

Архитектурата на еднослойна персептронна мрежа с n неврона е показана на фиг.1.14.



фиг.1.14

На входа на мрежата постъпва векторът $\mathbf{x} = (x_1, x_2, \ldots, x_m)^T$. Ако тегловната матрица на слоя е $\mathbf{W} = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix}$, а векторът на отместванията е $\mathbf{b} = (b_1, b_2, \ldots b_n)^T$, то уравненията на персептронния слой ще са:

$$\begin{pmatrix} s_1 \\ s_2 \\ \ldots \\ s_m \end{pmatrix} = \begin{pmatrix} w_{11}\ w_{12} \ldots w_{1m} \\ w_{11}\ w_{12} \ldots w_{1m} \\ \ldots\ldots\ldots\ldots\ldots \\ w_{n1}\ w_{n2} \ldots w_{nm} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_m \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{pmatrix} \quad , \quad \begin{pmatrix} y_1 \\ y_2 \\ \ldots \\ y_n \end{pmatrix} = \begin{pmatrix} f(s_1) \\ f(s_2) \\ \ldots \\ f(s_n) \end{pmatrix}$$

или в матрична форма:

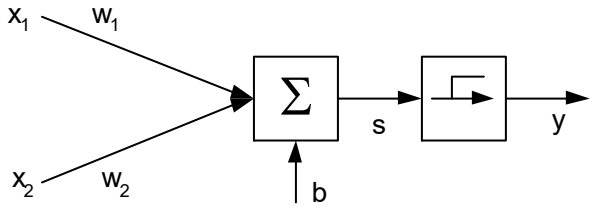
$$\mathbf{s} = \mathbf{W}\,\mathbf{p} + \mathbf{b} \qquad\qquad \mathbf{y} = \mathbf{f(s)}\,,$$

където $\mathbf{f}(\mathbf{s}) = \left(f(s_1), f(s_2) \ldots f(s_n)\right)^T$ е векторна функция.

1.2.3. Функции, възпроизвеждани от единичен персептрон

Въпросът за изясняване е какво означава персептронът да възпроизвежда дадена функция. За целта ще разгледаме функцията "изключващо или" (XOR).

Нека е даден единичен персептрон с два входа (фиг.1.15). Ако



фиг.1.15

персептронът възпроизвежда тази функция, то при подаване на входовете на нули или единици на изхода трябва да се получат стойностите на функцията "изключващо или", дадени в табл.1

Стойности на x_1	Стойности на x_2	Стойности на y
0	0	0
1	0	1
0	1	1
1	1	0

Табл.1

В сила е твърдението, че за произволни стойности на теглата w_1 и w_2 персептронът не може да възпроизвежда желания изход y.

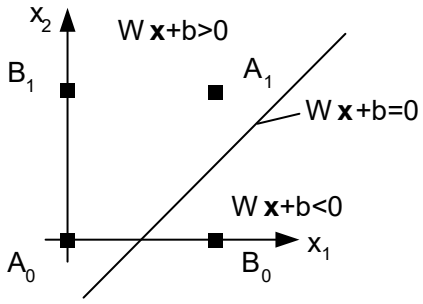
На изхода на сумиращия блок на персептрона се получава

$$s = x_1 w_1 + x_2 w_2 + b .$$

При това, ако стойността на s превишава дадената прагова стойност, то изходът y ще е единица, в противен случай ще е нула. Нека праговата стойност $prag$ е нула и правата с уравнение

$$x_1 w_1 + x_2 w_2 + b = 0$$

е нанесена на фиг.1.16.



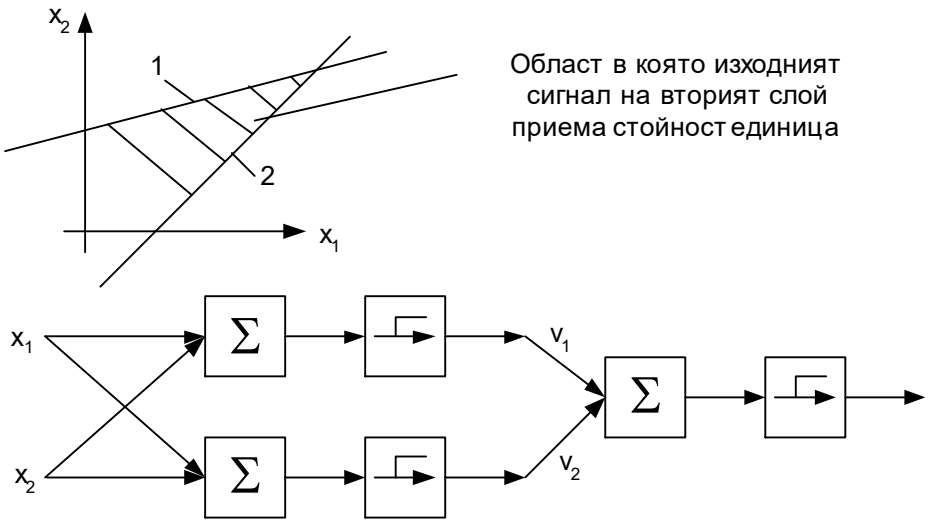
фиг.1.16

Тази права разделя равнината на две полуравнини такива, че ако точката $\mathbf{x} = (x_1, x_2)$ попадне в едната полуравнина, стойността на променливата s ще е по-малка от стойността на променливата $prag = 0$, т.е. $s = \mathbf{W}\,\mathbf{x} + b < 0$ и съответно $y = 0$, а ако попадне в другата полуравнина $s = \mathbf{W}\,\mathbf{x} + b \geq 0$ и $y = 1$. Съгласно таблицата се иска в точките $A_0(0,0)$ и $A_1(1,1)$ да се получи $y = 0$ и съответно в точките $B_0(1,0)$ и $B_1(0,1)$ $y = 1$. Следователно се търсят такива стойности на теглата (w_1, w_2), че точките A_0 и A_1 да са от едната страна на правата $x_1 w_1 + x_2 w_2 + b = 0$, а точките B_0 и B_1 да са от другата. Очевидно това е невъзможно, както и да е разположена тази права в равнината. Това показва, че единичен персептрон не може да възпроизвежда функцията “изключващо или”, но ще бъде показано, че една двуслойна персептронна мрежа може успешно да възпроизвежда тази функция

Очевидно един персептрон с m входа може да се разглежда като една двоична функция с m аргумента. Горният пример показва, че един персептрон може да възпроизвежда само линейноразделими функции.

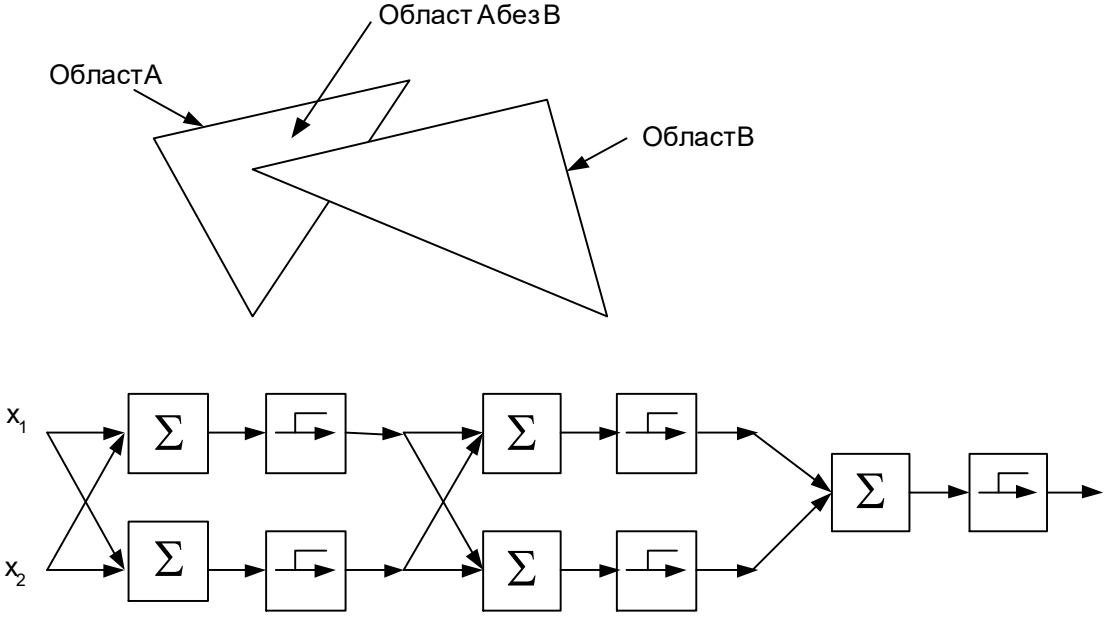
1.2.4. Функции, възпроизвеждани от двуслойни и трислойни персептронни мрежи

Известно е, че двуслойните персептронни мрежи могат да възпроизвеждат изпъкнали области. Ще илюстрираме това с един пример. Нека е дадена една двуслойна невронна мрежа с два неврона в първия слой и един неврон във втория (фиг.1.17), като



фиг.1.17

невронът от втория слой има равни тегла $v_1 = v_2 = 0.5$ и праг на чувствителност $prag = 0.75$. В такъв случай изходът на тази мрежа ще е единица само ако и двата неврона от първия слой подадат единици. Всеки от двата неврона от първия слой разбива равнината Ox_1x_2 на две полуравнини. Нека за първия слой е в сила следното: първият неврон обезпечава изход единица, когато точката (x_1, x_2) попадне под горната права, а вторият неврон осигурява изход единица, когато тази точка попадне над долната права (фиг.1.17). В такъв случай цялата мрежа ще осигури изход единица, когато (x_1, x_2) попадне в защрихованата област на фиг.1.17.



фиг.1.18

Не е трудно да се покаже, че условието за изпъкналост отпада при трислойна мрежа (фиг.1.18). В тази мрежа всеки от невроните във втория слой може да възпроизвежда изпъкнал многоъгълник. Нека тези многоъгълници са два, съответно A и B. В третия слой над тези многоъгълници се извършва логическата операция $A \setminus B$ и резултатът е неизпъкнала област (фиг.1.18).

1.2.5. Обучение на единичен персептрон с делта-правило

Разглеждаме единичния персептрон от фиг.1.12 с входен вектор $\mathbf{x} = (x_1, x_2, \ldots, x_m)^T$ и тегла $\mathbf{W} = (w_1, w_2, \ldots, w_m)$, b. За този вход реализираният изход на персептрона е y, а желаният (еталонният) изход е t. Целта на обучението е по зададен вход $\mathbf{x}$ и еталонен изход t да се намерят такива тегла на персептрона $\mathbf{W}$ и b, така че полученият изход y да е максимално близък до еталонния t, т.е. реализира се обучение с еталон. Идеята, стояща в основата на алгоритъма за обучение на персептрона (делта правило), е следната. Подава се вектор x и се изчислява изходът на неврона y. Ако той е правилен, теглата не се променят, ако не е правилен, теглата се променят, така че да се намали грешката $e = t - y$.

Едно правило (делта-правило), осигуряващо намаляване на грешката, е следното:

1. Подава се входен вектор x и се изчислява изходът на неврона y.
2. Ако изходът е верен ($y = t$, т.е. $e = t - y = 0$), то теглата не се коригират, т.е. $\Delta\mathbf{W} = 0$.
3. Ако изходът е неверен и е нула ($y = 0$ и $t = 1$, т.е. $e = t - y = 1$), то към теглата $\mathbf{W}$ се добавя вектор, пропорционален на входния $\mathbf{x}$, т.е. $\Delta\mathbf{W} = \mathbf{x}^T$.
4. Ако изходът е неверен и е единица ($y = 1$ и $t = 0$, т.е. $e = t - y = -1$), то от теглата $\mathbf{W}$ се изважда вектор, пропорционален на входния $\mathbf{x}$, т.е. $\Delta\mathbf{W} = -\mathbf{x}^T$.
5. Следва стъпка 1.

Смисълът на това правило е следният. Примерно, ако персептронът дава изход нула за входен вектор, при който изходът трябва да е единица, то очевидно теглата w_i, отговарящи на единиците във входния вектор $\mathbf{x}$, са малки и те не могат да осигурят достатъчно голяма стойност на променливата $s = \mathbf{W}\,\mathbf{x} + b$, така че тя да надмине праговата стойност на персептрона. Следователно точно

тези тегла w_i трябва да се увеличат. Аналогично, ако персептронът дава изход единица за входен вектор, за който изходът трябва да е нула, то трябва да се намалят тези тегла, които отговарят на единиците във входния вектор $\mathbf{x}$.

Горният алгоритъм е известен под названието делта-правило. Неговата реализация е следната. Пресмята се

$$\delta = t - y .$$

Ако $\delta = 0$, то изходът на неврона е верен и теглата не трябва да се променят. В противен случай теглата се коригират по формулата

$$\mathbf{W}_{new} = \mathbf{W}_{old} + \delta \mathbf{x}$$
$$b_{new} = b_{old} + \delta \ .$$

1.2.6. Обучение на персептронен слой с делта-правилото

Разглеждаме персептронния слой от фиг.1.14. На входа на слоя постъпва векторът $\mathbf{x} = (x_1, x_2, \ldots, x_m)^T$. Тегловата матрица на слоя е $\mathbf{W} = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix}$, а векторът от свободни тегла на слоя е $\mathbf{b} = (b_1, b_2, \ldots b_n)^T$. За входния вектор x реализираният изход на персептронния слой е векторът $\mathbf{y} = (y_1, \ldots, y_n)^T$, а желаният (еталонният) изход е векторът $\mathbf{t} = (t_1, \ldots, t_n)^T$. Целта на обучението е по зададен вход $\mathbf{x}$ и еталонен изход t да се намерят такива тегла на персептронния слой $\mathbf{W}$ и $\mathbf{b}$, че реализираният изход $\mathbf{y}$ да е близък до еталонния изход t. Алгоритъмът за обучение на единичен персептрон (делта-правило) може да се обобщи за персептронен слой по следния начин.

Пресмята се

$$\boldsymbol{\delta} = \mathbf{t} - \mathbf{y} .$$

Теглата се коригират по формулата

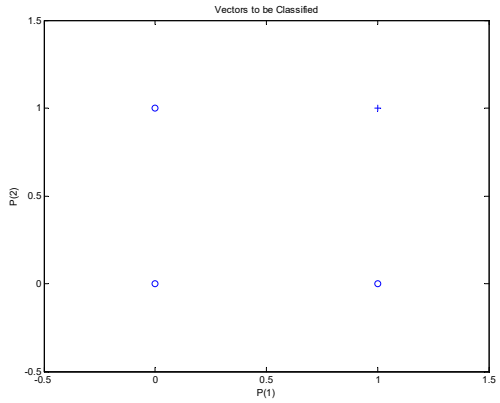
$$\mathbf{W}_{new} = \mathbf{W}_{old} + \boldsymbol{\delta} \mathbf{x}^T$$
$$\mathbf{b}_{new} = \mathbf{b}_{old} + \boldsymbol{\delta} .$$

Пример 1. Обучава се персептрон да възпроизвежда логическата функция ’и’.

Програмата, с която се създава тази мрежа, е следната:

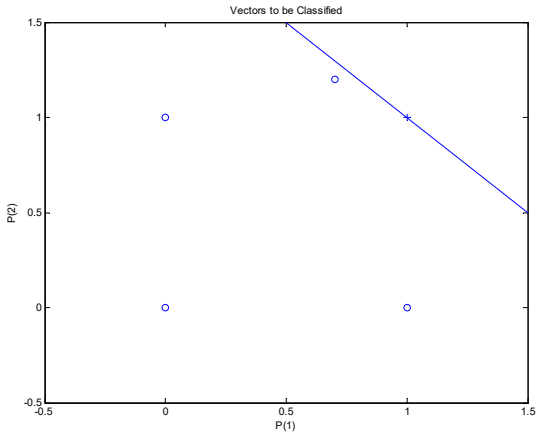
```
clear all
P = [0 0 1 1; 0 1 0 1];
T = [0 0 0 1];
plotpv(P,T);figure(gcf);pause;
net=newp(minmax(P),1);
for k=1:10
[net,Y,E]=adapt(net,P,T);
plotpc(net.IW{1},net.b{1});figure(gcf);pause;
if (sse(E)==0) break; end;
end;
p=[0.7; 1.2];
a = sim(net,p);
plotpv(p,a);
hold on;plotpv(P,T);plotpc(net.IW{1},net.b{1});hold
off;figure(gcf);
```

Данните, с които се обучава мрежата, са показани на фиг.1.19. С кръгче са данните с изход нула, а с плюс - данните с изход единица.



фиг.1.19

След като персептронът е обучен, се симулира с входния вектор $\mathbf{p} = (0.7, 1.2)^T$. Изходът е нула и резултатът е показан на фиг.1.20.



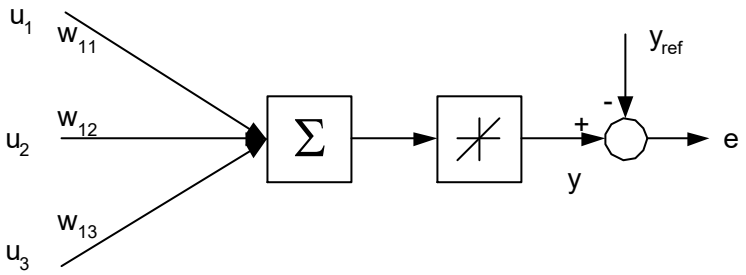
Фиг.1.20

На фиг.1.20 е изобразена и линията, която разделя входния клас вектори на две линейноразделими множества. За входове, попадащи под правата, изходът е нула, а за входове, попадащи над правата, изходът е единица.

1.3. Линейни невронни мрежи. Линейна задача на най-малките квадрати

Следният пример илюстрира задачата на линейните най-малки квадрати. Нека е даден единичен линеен неврон. Определя се изходът на неврона

$$y(k) = u_1(k)w_{11} + u_2(k)w_{12} + u_3(k)w_{13}, \text{ за } k = 1,..., N.$$



фиг.1.21

Въвеждат се грешките между реализирания от мрежата изход $y(k)$ и еталона $y_{ref}(k)$:

$$e(k) = u_1(k)w_{11} + u_2(k)w_{12} + u_3(k)w_{13} - y_{ref}(k), \text{ за } k = 1,..., N$$

или в матрична форма $\mathbf{e} = \mathbf{Ax} - \mathbf{b}$,

където $\mathbf{e} = (e(1), e(2),..., e(N))^T$; $\mathbf{x} = (w_{11}\ w_{12}\ w_{13})^T$

$$\mathbf{b} = (y_{ref}(1)\ y_{ref}(2)...y_{ref}(N))^T; \qquad \mathbf{A} = \begin{pmatrix} u_1(1) & u_2(1) & u_3(1) \\ u_1(2) & u_2(2) & u_3(2) \\ \vdots & \vdots & \vdots \\ u_1(N) & u_2(N) & u_3(N) \end{pmatrix}.$$

Тогава за вектора от грешките е в сила следното:

$$\|\mathbf{e}\|_2 = \left(\sum_{k=1}^{N} e^2(k)\right)^{\frac{1}{2}} = \| \mathbf{Ax} - \mathbf{b} \|_2.$$

За теглата w_{ij} се поставя следната оптимизационна задача:

(1) $$\min_{\mathrm{x}} \left(\sum_{k=1}^{N} e^2(k)\right)^{\frac{1}{2}} = \min_{\mathrm{x}} \| \mathbf{Ax} - \mathbf{b} \|_2,$$

при която се минимизира средноквадратичната грешка на изхода на неврона.

Задачата (1) може да бъде решена с крайна процедура. В този случай входните данни се обработват пакетно, като грешката на изхода на неврона се получава за всички $u(k),\ k=1,...,N$ и след това се решава (1). Задачата може да бъде решена и чрез последователно обработване на входните данни, при което се минимизира грешката за всяко $k=1,...,N$ чрез итерационната процедура на Уидроу-Хоф. Този алгоритъм ще бъде разгледан в следващия раздел.

Обща постановка на задачата за линейните най-малки квадрати.

Постановката на оптимизационната задача е следната:

(2) $$\| \mathbf{Ax}^* - \mathbf{b} \|_2 = \min_{\mathbf{x}\in R^n} \| \mathbf{Ax} - \mathbf{b} \|_2 = \min_{\mathbf{x}\in R^n} \left(\sum_{i=1}^{m} e_i^{\,2} \right)^{\frac{1}{2}},$$

където $e_i = a_{i1}x_1 + a_{i2}x_2 + \cdots a_{in}x_n - b_i,\ \ i=1,2,...,m,\ \ \mathbf{x}\in R^n,\ \ \mathbf{A}\in R^{m\times n}$ и се приема, че $m>n$. В случая с $\mathbf{x}^*\in R^n$ е означено решението на оптимизационната задача.

Нека рангът на $\mathbf{A}$ е пълен, т.е. $rang(\mathbf{A})=n$. Тогава задачата (2) има единствено решение, определено в (3). При условие че $rang(\mathbf{A})<n$, то задача (2) има повече от едно решение и едно от тях се получава чрез псевдообратната матрица на $\mathbf{A}$.

1.3.1. Едно решение на задачата за линейни най-малки квадрати при пълен ранг на матрицата $\mathbf{A}$

Ако рангът на матрицата $\mathbf{A}$ е пълен, тогава задачата (2) има единствено решение и то се определя по формулата

(3) $$\mathbf{x}^* = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b}.$$

Извеждането на решението (3) се илюстрира чрез следния пример. Нека в задачата (2) $\mathbf{x}\in R^2,\ \ \mathbf{A}\in R^{3\times 2},\ \ \mathbf{b}\in R^3$ и се получава

$$\mathbf{Ax}-\mathbf{b} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

Този израз може да се представи и по следния начин:

$$\mathbf{Ax}-\mathbf{b}=\begin{pmatrix}a_{11}\\a_{21}\\a_{31}\end{pmatrix}x_1+\begin{pmatrix}a_{12}\\a_{22}\\a_{32}\end{pmatrix}x_2-\begin{pmatrix}b_1\\b_2\\b_3\end{pmatrix}.$$

След въвеждане на векторите:

$$\mathbf{a_1}=\begin{pmatrix}a_{11}\\a_{21}\\a_{31}\end{pmatrix},\ \mathbf{a_2}=\begin{pmatrix}a_{12}\\a_{22}\\a_{32}\end{pmatrix},\ \mathbf{b}=\begin{pmatrix}b_1\\b_2\\b_3\end{pmatrix}\ \text{и}\ \mathbf{x}^*=\begin{pmatrix}x_1^*\\x_2^*\end{pmatrix},$$

векторът $\mathbf{Ax}^*$ се записва както следва:

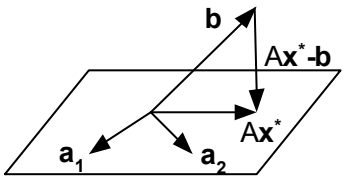
$$\mathbf{Ax}^*=\mathbf{a_1}x_1^*+\mathbf{a_2}x_2^*$$

и следователно е линейна комбинация на векторите $\mathbf{a_1}$ и $\mathbf{a_2}$. В такъв случай той лежи в една равнина с тези два вектора (фиг.1.22). От друга страна, за да бъде $\mathbf{x}^*$ решение на (2), дължината на вектора $\mathbf{Ax}^*-\mathbf{b}$ трябва да бъде минимална, т.е. $\|\mathbf{Ax}^*-\mathbf{b}\|_2\to\min$. Очевидно това се постига, когато векторът $\mathbf{Ax}^*-\mathbf{b}$ е ортогонален на $\mathbf{a_1}$ и $\mathbf{a_2}$ и съответно на равнината (фиг.1.22). Тогава $\mathbf{x}^*$ се определя от условията за ортогоналност на векторите $\mathbf{a_1}$ и $\mathbf{a_2}$ с вектора $\mathbf{Ax}^*-\mathbf{b}$, които са:

$$\mathbf{a_1}^T\left(\mathbf{Ax}^*-\mathbf{b}\right)=0$$
$$\mathbf{a_2}^T\left(\mathbf{Ax}^*-\mathbf{b}\right)=0.$$

Тези две уравнения се обединяват в системата

(4) $$\mathbf{A}^T\left(\mathbf{Ax}^*-\mathbf{b}\right)=0.$$



фиг.1.22

Рангът на матрицата $\mathbf{A}$ е пълен и следователно $\mathbf{A}^T\mathbf{A}$ не е особена. В такъв случай системата (4) има единствено решение

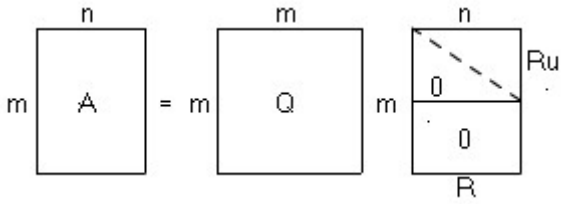
(5) $$\mathbf{x}^* = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b}.$$

Прилагането на формула (5) има този недостатък, че числото на обусловеност на матрицата $\mathbf{A}^T\mathbf{A}$ е квадратът на числото на обусловеност на $\mathbf{A}$ и обръщането на матрицата $\mathbf{A}^T\mathbf{A}$ може да води до голяма грешка от закръгляване. За да се избегне този недостатък, задачата (2) се решава чрез използване на $\mathbf{QR}$-декомпозиция на матрицата $\mathbf{A}$.

1.3.2. Решаване на задачата за линейните най-малки квадрати при пълен ранг на матрицата $\mathbf{A}$ чрез $\mathbf{QR}$-декомпозиция

Под $\mathbf{QR}$-декомпозиция на матрицата $\mathbf{A}$ се разбира следното разлагане:

$$\mathbf{A}_{m\times n} = \mathbf{Q}_{m\times m}\,\mathbf{R}_{m\times n},$$



фиг.1.23

където $\mathbf{Q}_{m\times m}$ е ортогонална матрица и $\mathbf{R}_{m\times n}$ е горна триъгълна матрица (фиг.1.23). Следователно матрицата $\mathbf{A}$ се представя по следния начин:

$$\mathbf{A} = \mathbf{Q}\begin{pmatrix}\mathbf{Ru}\\ 0\end{pmatrix}.$$

Търси се минимум на $\| \mathbf{Ax}^* - \mathbf{b} \|_2$. Но дължината на един вектор не се променя, ако се умножи с ортогонална матрица, следователно

(6) $$\| \mathbf{Ax} - \mathbf{b} \|_2 = \| \mathbf{Q}^T(\mathbf{Ax} - \mathbf{b}) \|_2 = \| \begin{pmatrix}\mathbf{Ru}\\ 0\end{pmatrix}\mathbf{x} - \mathbf{Q}^T\mathbf{b} \|_2 =$$

$$= \sum_{i=1}^{n}[\mathbf{R}_i\,\mathbf{x} - (\mathbf{Q}^T\mathbf{b})_i]^2 + \sum_{i=n+1}^{m}[(\mathbf{Q}^T\mathbf{b})_i]^2,$$

където $\mathbf{R}_i$ е i-тият ред на $\mathbf{Ru}$, а $(\mathbf{Q}^T\mathbf{b})_i$ е i-тият елемент на $\mathbf{Q}^T\mathbf{b}$. Тъй като горното е една сума от квадрати, то очевидно минимумът се постига за

(7) $$\mathbf{x}^* = \mathbf{R}\mathbf{u}^{-1}\begin{pmatrix} (\mathbf{Q}^T\mathbf{b})_1 \\ \vdots \\ (\mathbf{Q}^T\mathbf{b})_n \end{pmatrix}.$$

От представянето (6) се вижда още, че минималната стойност на (2) е точно

$$\sum_{i=n+1}^{m}[(\mathbf{Q}^T\mathbf{b})_i]^2.$$

1.3.3. Решаване на задачата за линейните най-малките квадрати при непълен ранг на матрицата $\mathbf{A}$

Ако рангът на $\mathbf{A}$ е непълен, то оптимизационната задача (2) има много решения. В такъв случай се поставя допълнителното изискване да се намери това решение $\mathbf{x}^*$ на (2), за което векторът $\mathbf{x}^*$ има минимална дължина. Тогава новата оптимизационна задача се формулира така:

(8) $\mathbf{x}^* = \min_{\mathbf{x}\in R^n} \| \mathbf{x} \|_2$ при условие, че $\| \mathbf{A}\mathbf{x}^* - \mathbf{b} \|_2 \leq \| \mathbf{A}\mathbf{x} - \mathbf{b} \|_2$ за всяко $\mathbf{x} \in R^n$.

Решението на (8) може да се намери чрез сингулярното разлагане на матрицата $\mathbf{A}$.

Определение 1.

Нека $\mathbf{A} \in R^{mxn}$ и $k = \min\{m,n\}$. Сингулярното разлагане на матрицата $\mathbf{A}$ е представянето във вида $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$, където $\mathbf{U} \in R^{mxm}$ и $\mathbf{V} \in R^{nxn}$ са ортогонални матрици, а $\mathbf{D} \in R^{mxn}$ е диагонална матрица с елементи по главния диагонал $d_{ii} = \sigma_i \geq 0$, за $i = 1,...k$. Величините $\sigma_1,...,\sigma_k$ се наричат сингулярни числа на матрицата $\mathbf{A}$.

Теорема 1.

Нека $\mathbf{A}_{m\times n}$ има сингулярно разлагане $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$. Тогава сингулярните числа σ_i са неотрицателните квадратни корени от собствените стойности на матрицата $\mathbf{A}\mathbf{A}^T$, ако $m \leq n$ или на матрицата $\mathbf{A}^T\mathbf{A}$, ако $m > n$. Стълбовете на матриците $\mathbf{U}$ и $\mathbf{V}$ представляват собствените вектори съответно на матриците $\mathbf{A}\mathbf{A}^T$ и $\mathbf{A}^T\mathbf{A}$. При това броят на ненулевите сингулярни числа е равен на ранга на матрицата $\mathbf{A}$.

Теорема 2.

Нека $\mathbf{A} \in R^{mxn}$ има сингулярно разлагане от вида $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$, където $\mathbf{U}$, $\mathbf{D}$ и $\mathbf{V}$ имат същия смисъл както в определение 1.

Образуваме матрицата $\mathbf{A}^+$ /наречена псевдообратна на $\mathbf{A}$/

$$\mathbf{A}^+ = \mathbf{V}\mathbf{D}^+\mathbf{U}^T, \ \mathbf{D}^+ = \begin{cases} d_{ii}^+ = \begin{cases} 1/\sigma_i, \ за \ \sigma_i > 0 \\ 0, \ за \ \sigma_i = 0 \end{cases} \\ d_{ij}^+ = 0, \ за \ i \neq j \end{cases},$$

тогава единственото решение на задачата (8) се явява

$$\mathbf{x}^* = \mathbf{A}^+\mathbf{b}.$$

Пример 1. Към линеен сигнал се прибавя гаусов шум. Полученият нов сигнал (еталон) $y(k)$ се апроксимира с една линейна невронна мрежа, която се синтезира по метода на линейните най-малки квадрати, т.е. с командата `newlind`. Мрежата се състои от един неврон с един вход. Изходът на мрежата се приравнява на еталона и се получава

$$y = wu(k) + b \ \text{ за } \ k = 1,...,N.$$

Ако тази система се запише в матричен вид

$$\begin{pmatrix} u(1) & 1 \\ u(2) & 1 \\ \vdots & \vdots \\ u(N) & 1 \end{pmatrix} \begin{pmatrix} w \\ b \end{pmatrix} = \begin{pmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{pmatrix} \tag{9}$$

и се въведат означенията:

$$\mathbf{A} = \begin{pmatrix} u(1) & 1 \\ u(2) & 1 \\ \vdots & \vdots \\ u(N) & 1 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} w \\ b \end{pmatrix} \text{ и } \mathbf{b} = \begin{pmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{pmatrix},$$

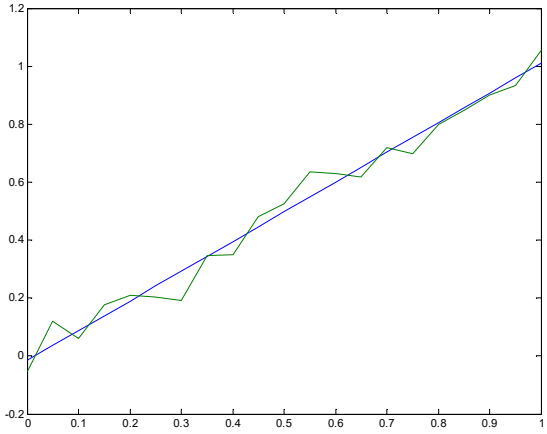
то се получава следната задача на линейните най-малки квадрати:

$$\min_{\bar{\mathbf{x}}} \| \mathbf{A}\mathbf{x} - \mathbf{b} \|_2.$$

Решението на тази задача се получава с метода, описан по-горе и заложен в процедурата `newlind`.

```
clear all
x=0:0.05:1;
n=length(x);
x1=normrnd(0,0.05,1,n);
y=x+x1;
cnet=newlind(x,y);
for k=1:n
   y22(k)=sim(cnet,x(k));
end;
plot(x,y22,x,y),figure(gcf)
```

Резултатът от действието на програмата е показан на фиг. 1.24.



фиг.1.24

Пример 2. Еталонът се генерира от следната дискретна система:
$y_{ref}(k) = u(k) + 0.5u(k-1) - 1.5u(k-2)$, за $k = 1,...,N$, $u(0) = u(-1) = 0$,
където $u(k) = \sin(\sin(t_k)10t_k)$ за $t_{k+1} = t_k + \Delta t$ и $\Delta t = 1/40$.

Линейната невронна мрежа, с която се апроксимира $y_{ref}(k)$, има един неврон с три входа и един изход:

$$y(k) = w_1u(k) + w_2u(k-1) + w_3u(k-2) + b, \text{ за } k = 1,...,N.$$

Ако се приравнят $y(k) = y_{ref}(k)$ и резултатът се запише в матричен вид, ще се получи

(10)
$$\begin{pmatrix} u(1) & 0 & 0 & 1 \\ u(2) & u(1) & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ u(N) & u(N-1) & u(N-2) & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ b \end{pmatrix} = \begin{pmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{pmatrix}.$$

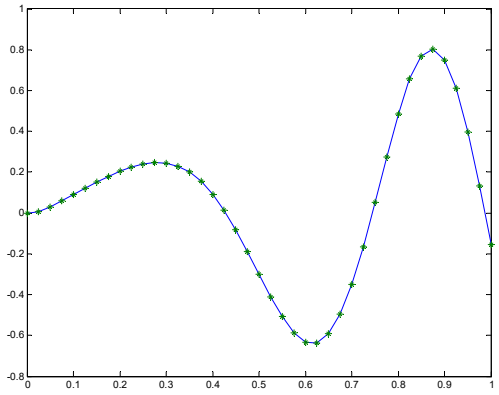
Означаваме

$$\mathbf{A} = \begin{pmatrix} u(1) & 0 & 0 & 1 \\ u(2) & u(1) & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ u(N) & u(N-1) & u(N-2) & 1 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ b \end{pmatrix} \text{ и } \mathbf{b} = \begin{pmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{pmatrix},$$

тогава по метода на най-малките квадрати се търси решението на задачата:

$$\min_{\mathbf{x}} \| \mathbf{Ax} - \mathbf{b} \|_2 .$$

```
clear all
time=0:0.025:1;
n=length(time);
x=sin(sin(time).*time*10);
y(1)=x(1);
y(2)=x(2)+0.5*x(1);
for k=3:n
   y(k)=x(k)+0.5*x(k-1)-1.5*x(k-2);
end;
p=[x(1:n);0 x(1:n-1);0 0 x(1:n-2)];
fnet=newlind(p,y);
a=sim(fnet,p);
plot(time,a,time,y,'*'),figure(gcf),pause;
```



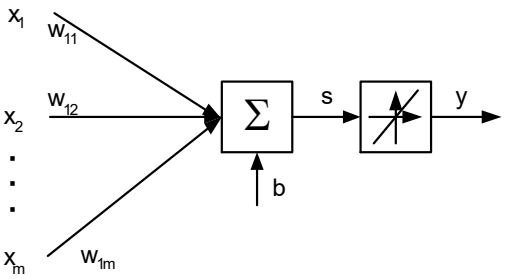
фиг.1.25

Резултатът от работата на програмата е показан на фигура 1.25. На тази фигура с непрекъсната линия е означен изходът на еталонната дискретна система, а със звездички е даден изходът на невронната мрежа.

1.4. Линейни невронни мрежи. Определяне на теглата с градиентен метод на Уидроу-Хоф

1.4.1. Единичен линеен неврон

Архитектурата на единичен линеен неврон с m входа и един изход е дадена на фиг. 1.26.



фиг.1.26

В тази схема $\mathbf{x} = (x_1,...,x_m)^T$ е входният вектор, y е изходът и $\mathbf{w} = (w_{11},...,w_{1m})^T$ е векторът от теглата на неврона, а b е свободно тегло на неврона. В такъв случай функцията, реализирана от този неврон, е следната:

$$y = \sum_{i=1}^{m} w_{1i}\ x_i + b\,.$$

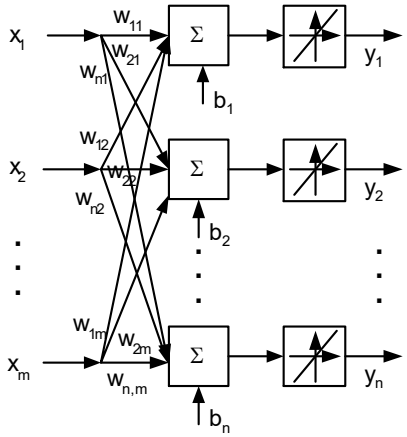
Много често входният вектор $\mathbf{x}$ е функция на времето и тогава функцията има вида

$$y(k) = \sum_{i=1}^{m} w_{1i}\ x_i(k) + b\,, \text{ за } k = 0,1,2,3,...\,,$$

където индексът k се отнася за k-тия отчет. Активиращата функция е линейна и нейната графика е ъглополовящата на първи и трети квадрант.

1.4.2. Единичен слой от линейни неврони

Архитектурата на единичен линеен слой с m входа и n изхода (n неврона) е следната:



фиг.1.27

В тази схема:

$\mathbf{x} = (x_1,...,x_m)^T$ е входният вектор;

$\mathbf{y} = (y_1,...,y_n)^T$ е изходният вектор;

$\mathbf{W} = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix}$ е матрицата от теглата на невронния слой;

$\mathbf{b} = (b_1,...,b_n)^T$ е векторът от свободните тегла на невронния слой;

$\mathbf{W}_i$ е i-тият ред на матрицата $\mathbf{W}$ и съдържа теглата на i-тия неврон.

В такъв случай изображението, реализирано от този невронен слой, е следното:

$$\mathbf{y} = \mathbf{W}\,\mathbf{x} + \mathbf{b}\,.$$

Ако входният вектор $\mathbf{x}$ е функция на времето, то

$$\mathbf{y}(k) = \mathbf{W}\,\mathbf{x}(k) + \mathbf{b}\,, \text{ за } k = 0,1,2,3,...,$$

където индексът k се отнася за k-тия отчет.

1.4.3. Обучение на единичен линеен слой с метода на Уидроу-Хоф

Целта е да се разгледа алгоритъм за обучение на мрежата от фиг.1.27 по вход $\mathbf{x}(k)$ и еталон $\mathbf{t}(k)$, където входният вектор има размерност m, а еталонният вектор има размерност n:

$$\mathbf{x}(k) = (x_1(k),...,x_m(k))^T$$

$$\mathbf{t}(k) = (t_1(k),...,t_n(k))^T\,.$$

Уравненията на тази мрежа са следните:

$$\begin{pmatrix} y_1(k) \\ \vdots \\ y_n(k) \end{pmatrix} = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix} \begin{pmatrix} x_1(k) \\ \vdots \\ x_m(k) \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

или

$$y_i = w_{i1}x_1(k) + w_{i2}x_2 + ... + w_{im}x_m + b_i; \quad i = 1,2,...,n .$$

Сумата от квадратите на грешките, т.е. разликите между компонентите на еталона $\mathbf{t}$ и реализирания изход $\mathbf{y}$, е

$$S(k) = \frac{1}{2}(e_1^{\,2}(k) + ... + e_n^{\,2}(k)) =$$

$$= \frac{1}{2}(t_1(k) - \sum_{j=1}^{m} w_{1j}x_j(k) - b_1)^2 + ... + \frac{1}{2}(t_n(k) - \sum_{j=1}^{m} w_{nj}x_j(k) - b_n)^2 .$$

Диференцира се S спрямо теглата на мрежата w_{ij} и се получава

$$\frac{\partial S(k)}{\partial w_{ij}} = -2e_i(k)x_j(k), \text{ за } i = 1,...n \text{ и } j = 1,...,m .$$

Диференцира се S спрямо теглата на мрежата b_i и се получава

$$\frac{\partial S(k)}{\partial b_i} = -2e_i(k), \text{ за } i = 1,...n .$$

Записват се производните в матричен вид

$$\frac{\partial S(k)}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial S(k)}{\partial w_{11}} & \frac{\partial S(k)}{\partial w_{12}} & .... & \frac{\partial S(k)}{\partial w_{1m}} \\ ..... & & & \\ \frac{\partial S(k)}{\partial w_{n1}} & \frac{\partial S(k)}{\partial w_{n2}} & .... & \frac{\partial S(k)}{\partial w_{nm}} \end{pmatrix} = -2 \begin{pmatrix} e_1(k) \\ \vdots \\ e_n(k) \end{pmatrix} \begin{pmatrix} x_1(k) ... x_m(k) \end{pmatrix} =$$

$$= -2 \begin{pmatrix} x_1(k)e_1(k) & \cdots & x_m(k)e_1(k) \\ \vdots & & \vdots \\ p_1(k)e_n(k) & \cdots & x_m(k)e_n(k) \end{pmatrix} \text{ и}$$

$$\frac{\partial S(k)}{\partial \mathbf{b}} = \begin{pmatrix} \frac{\partial S(k)}{\partial b_1} \\ ..... \\ \frac{\partial S(k)}{\partial b_n} \end{pmatrix} = -2 \begin{pmatrix} e_1(k) \\ \vdots \\ e_n(k) \end{pmatrix},$$

т.е. $\frac{\partial S(k)}{\partial \mathbf{w}} = -2\mathbf{e}(k)\ \mathbf{x}(k)^T$ и $\frac{\partial S(k)}{\partial \mathbf{b}} = -2\mathbf{e}(k)$,

където $\mathbf{e}(k) = \begin{pmatrix} e_1(k) \\ \vdots \\ e_n(k) \end{pmatrix}$ и $\mathbf{x}(k) = \begin{pmatrix} x_1(k) \\ \vdots \\ x_m(k) \end{pmatrix}$.

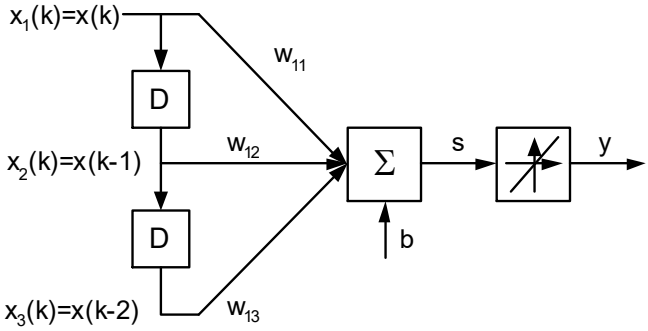
В такъв случай теглата на мрежата се коригират по формулите:

$$\begin{aligned} \mathbf{W}(k+1) &= \mathbf{W}(k) + \alpha\,\mathbf{e}(k)\,\mathbf{x}(k)^T \\ \mathbf{b}(k+1) &= \mathbf{b}(k) + \alpha\,\mathbf{e}(k) \end{aligned}$$, където $\mathbf{W} = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$,

където k е номерът на итерацията, а α е реална константа - скорост на обучение ($0 < \alpha < 1$). Тази итерационна процедура е известна като метод на Уидроу- Хоф.

1.4.4. Пример на линеен адаптивен филтър

Структурата на мрежата е дадена на фиг.1.28.



фиг.1.28

Уравнението, реализирано от мрежата, е следното:

(11) $$y(k) = x(k)\ w_{11} + x(k-1)\ w_{12} + x(k-2)\ w_{13} + b\,.$$

В случая множеството $\{x(k)\}_{k=0}^{N}$ се нарича времеви ред. Индексът k показва, че съответната стойност се отнася за k-тия момент от време. Затова се приема при $k<0$ $x(k)=0$. За този ред се въвежда операторът закъснение $Dx(k) = x(k-1)$. Последователното прилагане на D дава линия от закъснения (фиг.1.28).

Ако се предположи, че размахът на входния сигнал е от 0 до 10, то една невронна мрежа може да се създаде с командата

```
net=newlin([0 10],1,[0 1 2]);
```

Вторият входен параметър на командата `newlin` показва, че мрежата има един неврон, а третият, че закъсненията на първия, втория и

третия вход са съответно 0 1 2 времеви единици. Нека са зададени следните стойности на теглата:

```
net.IW{1,1}=[1 5 7];
net.b{1}=[0];
```

В такъв случай формула (11) добива вида

(12) $$y(k) = x(k) + 5x(k-1) + 7x(k-2).$$

Важно е да се отбележи, че при използване на процедурата `adapt` за обучение на мрежата с метода на Уидроу-Хоф входните данни трябва да се зададат като клетъчен масив, а изходните се получават в същия вид. Например в случая входните данни са

```
x={2 4 6};
```

и симулиране с тези входни данни и генерираната невронна мрежа се постига с командата

```
y=sim(net,x,{1 2});
```

Резултатът от работата на тази команда е

```
y =[19]    [28]    [40]
```

При така зададените входни данни и закъснения ще се реализира обучение на мрежата при следния еталон:

```
T={8 12 16};
```

Това се постига с програмата

```
net.adaptParam.passes=10;
[net,y,E]=adapt(net,x,T,{1 2});
```

Резултатът е съответно

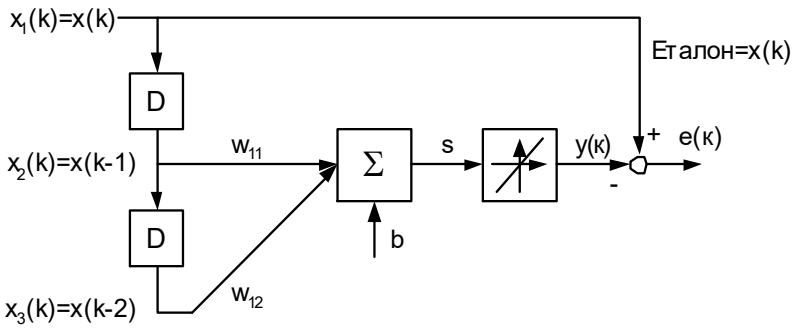
```
y = [9.0620]    [11.9406]    [15.6091
```

Очевидно резултатът y от симулацията е близък до еталонния 8,12,16.

1.4.5. Предиктор за времеви редове

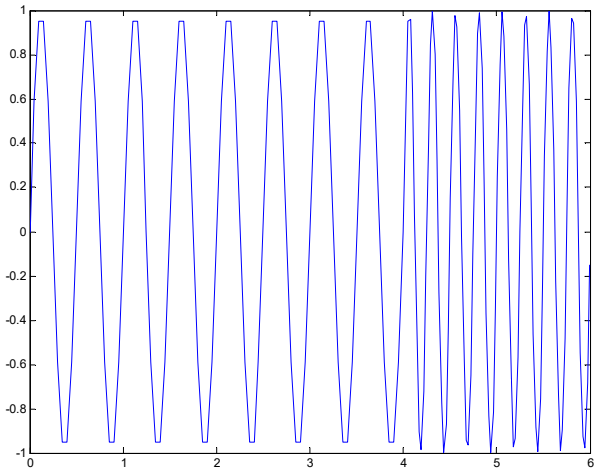
Адаптивен филтър може да се използва за предсказване (прогнозиране) на следващата стойност $x(\kappa)$ на стационарен случаен процес по предходните стойности на същия процес $x(\kappa-1)$, $x(\kappa-2)$,… Невронната мрежа от фиг. 1.29 илюстрира този процес и съответно тя се нарича невронен предиктор. Сигналът $x(k)$, който трябва да се прогнозира, постъпва отляво на линията от закъснения (фиг.1.29). Предхождащите две стойности на процеса $x(k-1)$ и $x(k-2)$ са известни и се задават като входове от линията на закъсненията. Теглата на мрежата се коригират на всяка стъпка чрез метода на Уидроу-Хоф (чрез процедурата `adapt`). Целта е да се минимизира грешката $e(k)$ на изхода на неврона. Следващият пример илюстрира

този алгоритъм. По-долу са дадени и някои разяснения за съответната програма.



фиг.1.29

В интервала [0, 8] със стъпка $\Delta t_1 = 0.04$ е зададена функцията $\sin(3\pi k\Delta t_1)$ и в $t \in [8.04, 16]$ със стъпка $\Delta t_2 = 0.02$ функцията $\cos(6\pi k\Delta t_2)$, които съвместно образуват времевия ред, подлежащ на прогнозиране. Графиката на този времеви ред е дадена на фиг. 1.30.

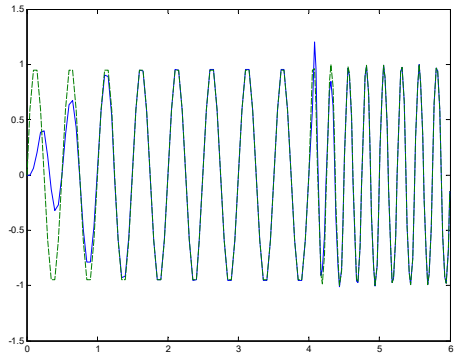


фиг. 1.30

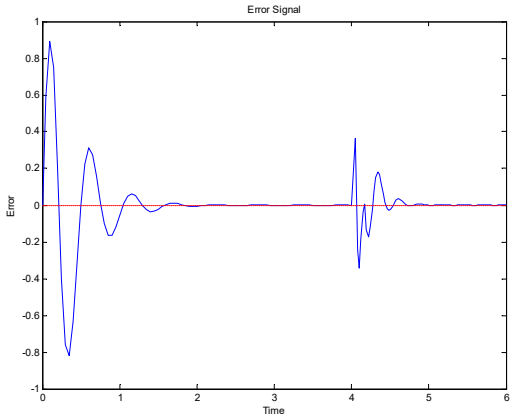
Обучение на мрежата

Мрежата се обучава с програмата `adapt`, входните параметри на която са входният сигнал P и еталонният изход T. Важна особеност на командата `adapt` е, че данните P и T трябва да се преобразуват от матрици в клетъчни масиви. Това се постига с процедурата `con2seq`. Структурата на линейния предиктор е зададена в командата `newlind`. Първият входен параметър показва, че размахът на входните данни е в интервала $[-1\ 1]$. Вторият показва, че в линейния слой има един неврон. Следват параметрите `delayes` и `lr`, с които съответно се задават закъсненията и скоростта на обучение.

След завършване на обучението на мрежата изходният сигнал и еталонният се сравняват графично на фиг.1.31. Тъй като изходните данни y и e от `adapt` са отново клетъчни масиви, то се налага обратно преобразуването на тези клетъчни масиви в матрици. Това става с командите `cat(1,y{:})` и `cat(1,e{:})`. Параметърът 1 в `cat` показва, че съответните данни са вектори.



фиг. 1.31

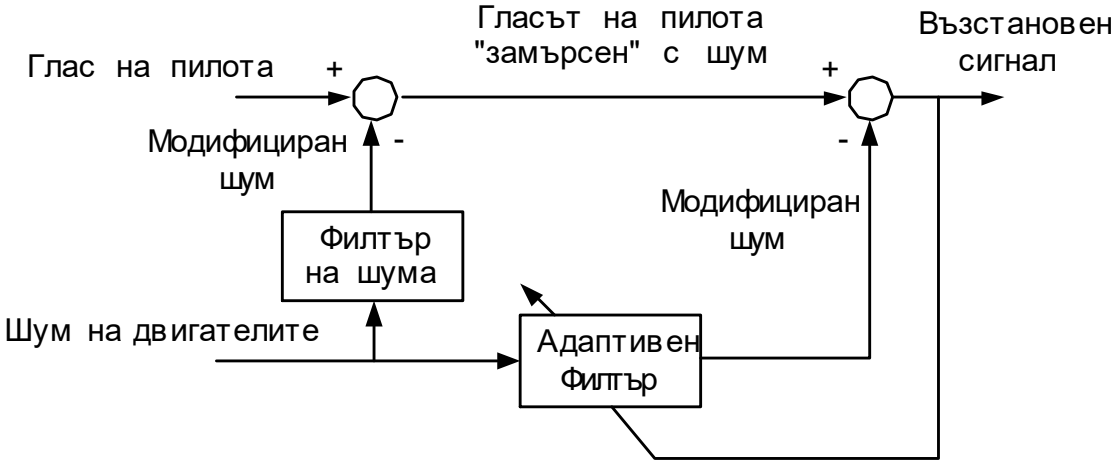


фиг. 1.32

```
clear all
tk=8;om=3;
time1=0:0.04:8;time2=tk+0.04:0.02:16;
time=[time1 time2];
T1=[sin(time1*om*pi) cos(time2*2*om*pi)];
plot(time,T1);
xlabel('t');
ylabel('Etalon'),
figure(gcf),pause
T=con2seq(T1);
P=T;
lr=0.05;
delays=[1 2 3 4 5];
netpr=newlin([-1 1],1,delays,lr);
[netpr,y,e]=adapt(netpr,P,T);
plot(time,cat(1,y{:}),time,cat(1,T{:}),'--k');
xlabel('t');
ylabel('Etalon+Resultat');
figure(gcf),pause
plot(time,cat(1,e{:}));
xlabel('t');
ylabel('Error');
figure(gcf)
```

1.4.6. Пример за филтриране на шум с адаптивна мрежа

Полезният сигнал в този модел е стационарен случаен процес (например глас на пилота, предаван чрез микрофон). Смущаването на този сигнал става с шум (създаван от двигателите на самолета), който има основна съставка хармоничен процес със зададена честота. Целта е да се отдели полезният сигнал от шума, което може да се реализира с линейна невронна мрежа (адаптивен филтър). Линейната невронна мрежа се обучава да предсказва полезния сигнал, като използва за вход шума. За целта грешката на изхода на мрежата се образува като разлика на полезния сигнал, замърсен с модифицирания шум и предсказания от мрежата модифициран шум. В резултат грешката съдържа само полезния сигнал. Следователно обучената линейна мрежа отстранява шума.

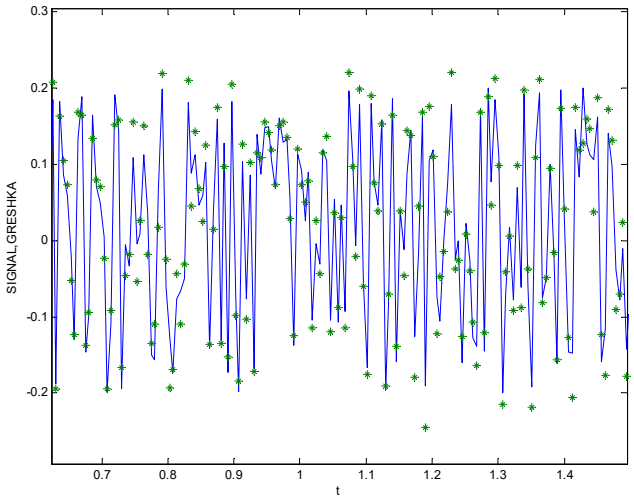


фиг.1.33

```
clear all
 tk=1.5;  %Vreme na modelirane
 mc=0.9;  %Moment  0:1
 lr=0.1;  %Parametar na obuchenie 0.1:1.5

  % KONSTANTI
  N=3;                          % Broi otcheti za period
  f = 60;                       % Chestota na shuma
  ts = N*f*tk+1;                 % Broi na otcheti

  % Signal,shum amplitudi
  A = 0.2;  A1=1.2;
```

```
  G=0.1;                        % Amplituda na filtriran shum
  signal = A*rands(1,ts);       % Polezen signal
  i = 1:ts;
  shum = A1*sin(2*pi*(i-1)/N);
  filter_shum = G*sin (2*pi*(i-1)/N + pi/2)+…
G*normrnd(0,0.05,1,length(i));
  vhod_Nmreja = [shum; 0 shum(1:length(shum)-1)];
  signal_shum = signal + filter_shum;

%  Nachalni tegla
   w=[-0.08 -0.14];
  time = [1:ts]/ts*tk;      % Vremevi otcheti

  % VHOD, ETALON
  P = vhod_Nmreja;
  T= signal_shum;

%  VHODNI DANNI
  q = size(P,2);          % Number of timesteps
  a = zeros(1,q);         % Izhod na mrejata
  e = zeros(1,q);         % Error over time
  time = [1:q]/q*tk;      % Vremevi otcheti na sinulirane
  % OBUCHENIE
  dw0 = [0 0];
  for i=1:q
    a(i) = w*P(:,i);
    e(i) = T(i) - a(i);
    dw = mc*dw0 + (1-mc)*lr*e(i)*P(:,i)';
    w = w + dw;
    dw0 = dw;
 end

 % REZULTATI
 plot(time,T);
 xlabel('t');
 ylabel('ETALON');
 figure(gcf),pause
 plot(time,signal,time,e,'*');
 xlabel('t');
ylabel('SIGNAL,GRESHKA');
 figure(gcf),pause
```

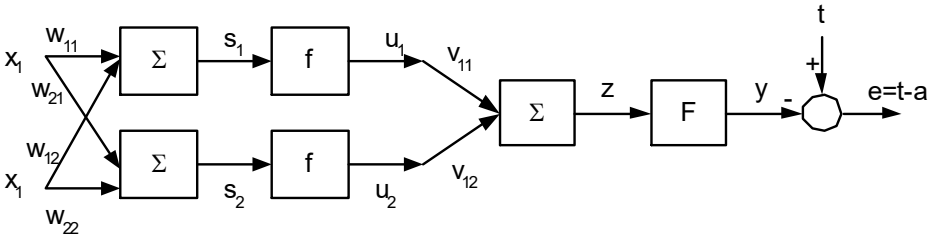
Резултатът от работата на програмата е показан на фиг.1.34



фиг. 1.34

1.5 Алгоритъм на обратно разпространяване на грешките (backpropagation)

Тук ще бъде разгледан един пример, с който се илюстрира алгоритъмът на обратно разпространяване на грешките. След това алгоритъмът ще бъде обобщен. За целта се разглежда двуслойна невронна мрежа (фиг.1.35).



фиг.1.35

Функцията, реализирана от мрежата, се описва с изразите

(13)
$$\begin{aligned} s_1 &= w_{11}x_1 + w_{12}x_2, \quad u_1 = f(s_1), \\ s_2 &= w_{21}x_1 + w_{22}x_2, \quad u_2 = f(s_2), \end{aligned} \qquad z = v_{11}u_1 + v_{12}u_2, \quad y = F(z).$$

Формулите (13) дават аналитичното представяне на правия ход на алгоритъма, при който изчисленията се извършват от входа към изхода на мрежата.

Нека t е еталонът за обучение на мрежата. Тогава средноквадратичната грешка на мрежата се определя по формулата

(14)
$$s = \frac{1}{2}(y-t)^2 = \frac{1}{2}e^2.$$

Задачата, която се поставя, е минимизиране на грешката (14) спрямо теглата на мрежата [9]. Това е една нелинейна оптимизационна задача, за решаването на която се използват процедури от градиентен тип, разглеждани в следващия раздел. За реализирането на една такава оптимизационна процедура задължително трябва да се определи градиентът на грешката от (14) по отношение на теглата на мрежата w_{ij} и v_{ij}. Това се постига чрез диференциране на изразите (13), като в този случай изчисленията се извършват от изхода към входа на мрежата. По този начин се реализира обратният ход на алгоритъма.

Пресмятане на градиента на грешката по отношение на теглата от втория слой v_{ij}:

$$\begin{aligned}\frac{\partial s}{\partial v_{11}} &= \frac{\partial s}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial v_{11}} = e\,F'(z)u_1 = \delta\,u_1 \\ \frac{\partial s}{\partial v_{12}} &= \frac{\partial s}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial v_{12}} = e\,F'(z)u_2 = \delta\,u_2\end{aligned}, \tag{15}$$

където $\delta = e\,F'(z)$.

Пресмятане на градиента на грешката по отношение на теглата от първия слой w_{ij}:

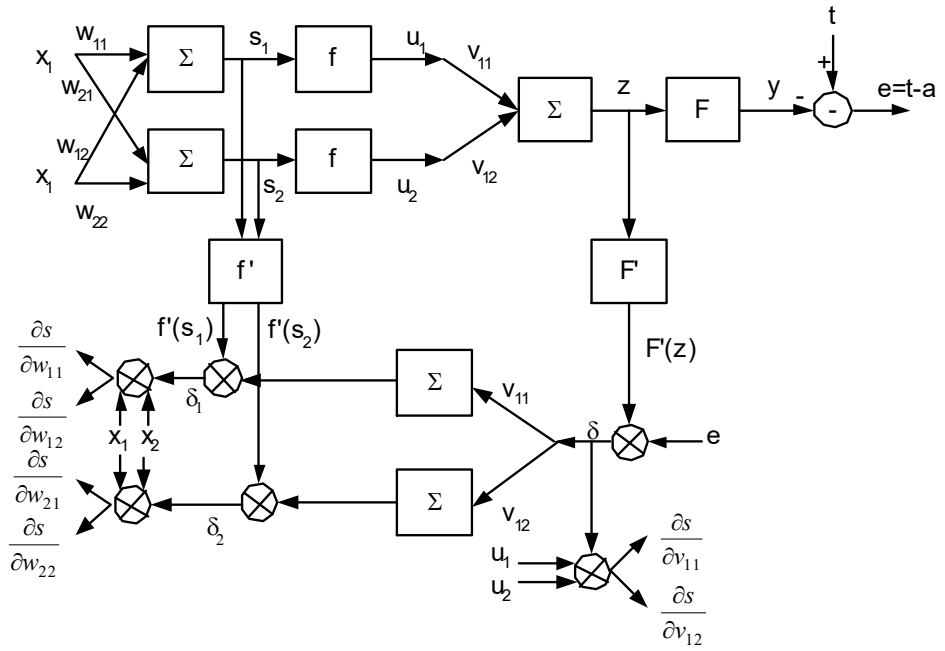
$$\begin{aligned}\frac{\partial s}{\partial w_{11}} &= \frac{\partial s}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial u_1}\frac{\partial u_1}{\partial s_1}\frac{\partial s_1}{\partial w_{11}} = e\,F'(z)v_{11}f'(s_1)x_1 = \delta_1 x_1 \\ \frac{\partial s}{\partial w_{12}} &= \frac{\partial s}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial u_1}\frac{\partial u_1}{\partial s_1}\frac{\partial s_1}{\partial w_{12}} = e\,F'(z)v_{11}f'(s_1)x_2 = \delta_1 x_2\end{aligned}, \tag{16}$$

където $\delta_1 = e\,F'(z)v_{11}f'(s_1)$

$$\begin{aligned}\frac{\partial s}{\partial w_{21}} &= \frac{\partial s}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial u_2}\frac{\partial u_2}{\partial s_2}\frac{\partial s_2}{\partial w_{21}} = e\,F'(z)v_{12}f'(s_2)x_1 = \delta_2 x_1 \\ \frac{\partial s}{\partial w_{22}} &= \frac{\partial s}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial u_2}\frac{\partial u_2}{\partial s_2}\frac{\partial s_2}{\partial w_{22}} = e\,F'(z)v_{12}f'(s_2)x_2 = \delta_2 x_2\end{aligned}, \tag{17}$$

където $\delta_2 = e\,F'(z)v_{12}f'(s_2)$.

Чрез изразите (13) (15), (16) и (17) може да се построи мрежа за правия и обратния ход на алгоритъма (фиг.1.36).



фиг.1.36

В този пример на практика е пресметнат директно градиентът на грешката s относно теглата $(v_{11}, v_{12}, w_{11}, w_{12}, w_{21}, w_{22})$. Това позволява да се приложи градиентна оптимизационна процедура. При използването на процедури от по-висок ред, които се разглеждат в следващия раздел, се изисква определяне на якобиана на грешките. За този пример якобианът се определя от (15), (16) и (17), както следва:

$$\mathbf{J} = \left(\frac{\partial e}{\partial v_{11}}, \frac{\partial e}{\partial v_{12}}, \frac{\partial e}{\partial w_{11}}, \frac{\partial e}{\partial w_{12}}, \frac{\partial e}{\partial w_{21}}, \frac{\partial e}{\partial w_{22}} \right) =$$

$$= \left(F'(z)u_1, F'(z)u_2, F'(z)v_{11}f'(s_1)x_1, F'(z)v_{11}f'(s_1)x_2, F'(z)v_{12}f'(s_2)x_1, F'(z)v_{12}f'(s_2)x_2 \right)$$

1.5.1. Обобщаване на алгоритъма с обратно разпространяване на грешките за трислойна невронна мрежа

Алгоритъмът е разгледан за права трислойна невронна мрежа (фиг.1.37) с активиращи функции $g(.)$[8]. Тази схема изразява както изчислителния процес от входа към изхода (прав ход на метода), така и изчисляването на компонентите на градиента от изхода към входа (обратен ход на метода). Въвежда се функция на грешките

(18) $$S = \frac{1}{2} \sum_{i=1}^{m} e_i^2 ,$$

където грешките на изхода на мрежата са $e_i = y_i - t_i, i = 1,2,...,m$, а t_i са еталоните.

Въвеждат се следните матрици на тегловните коефициенти за съответните слоеве:

$$\mathbf{W_1} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & ...w_{1n}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & ...w_{2n}^{(1)} \\ ..... & & \\ w_{p1}^{(1)} & w_{p2}^{(1)} & ...w_{pn}^{(1)} \end{pmatrix}. \quad \mathbf{W}_2 = \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} & ...w_{1p}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & ...w_{2p}^{(2)} \\ ..... & & \\ w_{q1}^{(2)} & w_{q2}^{(2)} & ...w_{qp}^{(2)} \end{pmatrix} \text{ и } \mathbf{W}_3 = \begin{pmatrix} w_{11}^{(3)} & w_{12}^{31)} & ...w_{1q}^{(3)} \\ w_{21}^{(3)} & w_{22}^{(3)} & ...w_{2q}^{(3)} \\ ..... & & \\ w_{m1}^{(1)} & w_{m2}^{(1)} & ...w_{mq}^{(3)} \end{pmatrix}.$$

$$\mathbf{W_1} \in R_{p\times n} \qquad \mathbf{W}_2 \in R_{q\times p} \qquad \mathbf{W}_3 \in R_{m\times q}$$

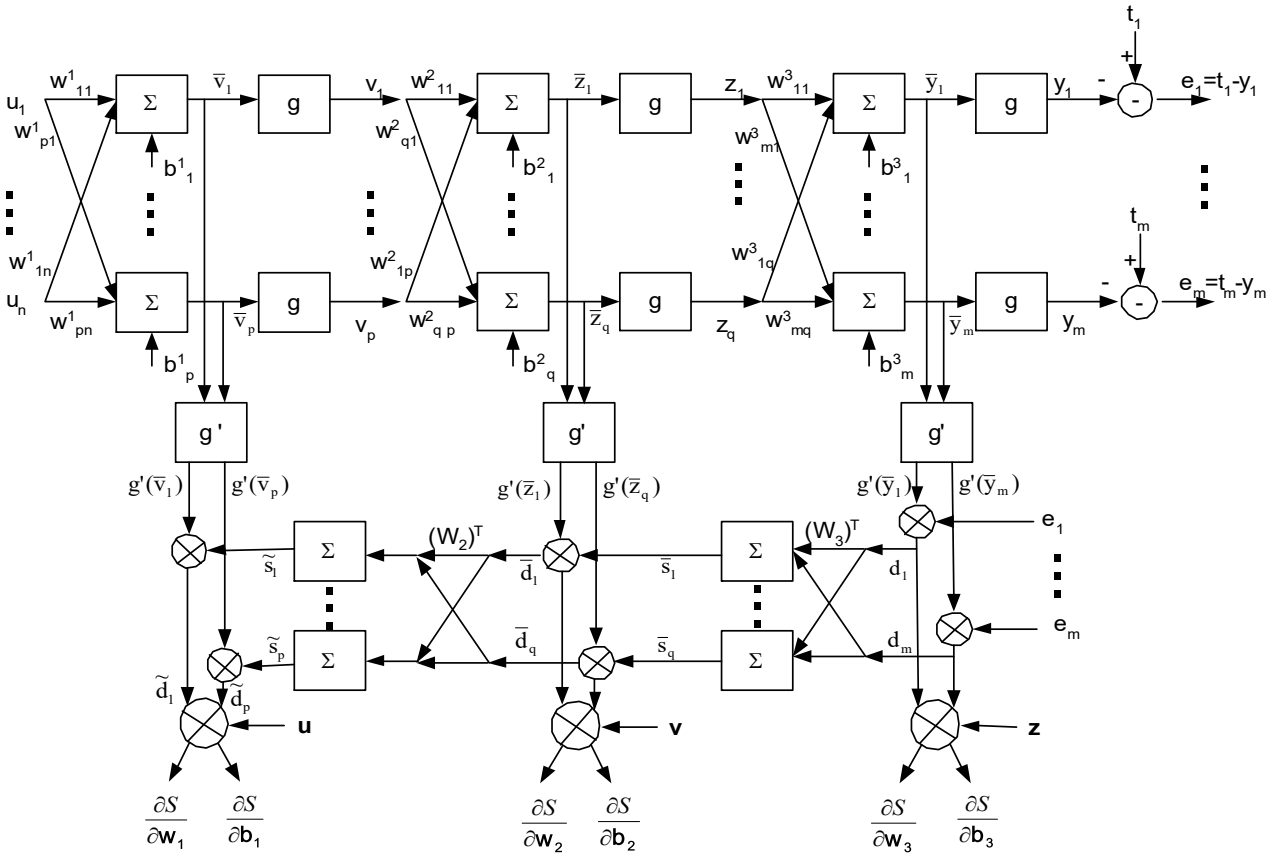
Прав ход на алгоритъма

Основните връзки за слоевете се задават, както следва:

(19) $\mathbf{W_1 u + b_1} = \overline{\mathbf{v}} \qquad \mathbf{W}_2\mathbf{v} + \mathbf{b}_2 = \overline{\mathbf{z}} \qquad \mathbf{W}_3\mathbf{z} + \mathbf{b}_3 = \overline{\mathbf{y}}$,

$\mathbf{v} = \left(g(\overline{v}_1), g(\overline{v}_2),..., g(\overline{v}_p)\right)^T$, $\mathbf{z} = \left(g(\overline{z}_1), g(\overline{z}_2),..., g(\overline{z}_q)\right)^T$, $\mathbf{y} = \left(g(\overline{y}_1), g(\overline{y}_2),...g(\overline{y}_m)\right)^T$

и изчисленията се извършват от входа към изхода съгласно (19).



фиг.1.37

Обратен ход на алгоритъма

Градиентът на грешката (18) в матрична форма се получава както се реализира процедурата на обратния ход, показана чрез паралелната схема от фиг. 1.37. Не е трудно да се проследят изчисленията в обратна посока:

$$(20)\qquad \left(\frac{\partial S}{\partial \mathbf{w}_3}:\frac{\partial S}{\partial \mathbf{b}_3}\right)=\begin{pmatrix} d_1z_1\ d_1z_2\ ...d_1z_q : d_1 \\ d_2z_1\ d_2z_2\ ...d_2z_q : d_2 \\ ..... \\ d_mz_1\ d_mz_2\ ...d_mz_q : d_m \end{pmatrix},\ \text{за}\ \mathbf{d}=\begin{pmatrix} d_1 \\ d_2 \\ .... \\ d_m \end{pmatrix}=\begin{pmatrix} e_1g'(\bar{y}_1) \\ e_2g'(\bar{y}_2) \\ ..... \\ e_mg'(\bar{y}_m) \end{pmatrix}$$

$$(21)\qquad \left(\frac{\partial S}{\partial \mathbf{w}_2}:\frac{\partial S}{\partial \mathbf{b}_2}\right)=\begin{pmatrix} \bar{d}_1v_1\ \bar{d}_1v_2\ ...\bar{d}_1v_p : \bar{d}_1 \\ \bar{d}_2v_1\ \bar{d}_2v_2\ ...\bar{d}_2v_p : \bar{d}_2 \\ ..... \\ \bar{d}_qv_1\ \bar{d}_qv_2\ ...\bar{d}_qv_p : \bar{d}_q \end{pmatrix},\ \text{за}\ \bar{\mathbf{d}}=\begin{pmatrix} \bar{d}_1 \\ \bar{d}_2 \\ .... \\ \bar{d}_q \end{pmatrix}=\begin{pmatrix} \bar{s}_1g'(\bar{z}_1) \\ \bar{s}_2g'(\bar{z}_2) \\ ..... \\ \bar{s}_qg'(\bar{z}_q) \end{pmatrix}\ \text{и}$$

$$\begin{bmatrix} \bar{s}_1 \\ \bar{s}_2 \\ ... \\ \bar{s}_q \end{bmatrix}=\mathbf{W}_3^T\begin{bmatrix} d_1 \\ d_2 \\ .... \\ d_m \end{bmatrix}$$

$$(22)\qquad \left(\frac{\partial S}{\partial \mathbf{w}_1}:\frac{\partial S}{\partial \mathbf{b}_1}\right)=\begin{pmatrix} \tilde{d}_1u_1\ \tilde{d}_1u_2\ ...\tilde{d}_1u_n : \tilde{d}_1 \\ \tilde{d}_2u_1\ \tilde{d}_2u_2\ ...\tilde{d}_2u_n : \tilde{d}_2 \\ ..... \\ \tilde{d}_pu_1\ \tilde{d}_pu_2\ ...\tilde{d}_pu_n : \tilde{d}_p \end{pmatrix},\ \text{за}\ \tilde{\mathbf{d}}=\begin{pmatrix} \tilde{d}_1 \\ \tilde{d}_2 \\ .... \\ \tilde{d}_p \end{pmatrix}=\begin{pmatrix} \tilde{s}_1g'(\bar{v}_1) \\ \tilde{s}_2g'(\bar{v}_2) \\ ..... \\ \tilde{s}_pg'(\bar{v}_p) \end{pmatrix}\ \text{и}$$

$$\begin{pmatrix} \tilde{s}_1 \\ \tilde{s}_2 \\ ... \\ \tilde{s}_p \end{pmatrix}=\mathbf{W}_2^T\begin{pmatrix} \bar{d}_1 \\ \bar{d}_2 \\ .... \\ \bar{d}_q \end{pmatrix}.$$

В (22) е въведено означението

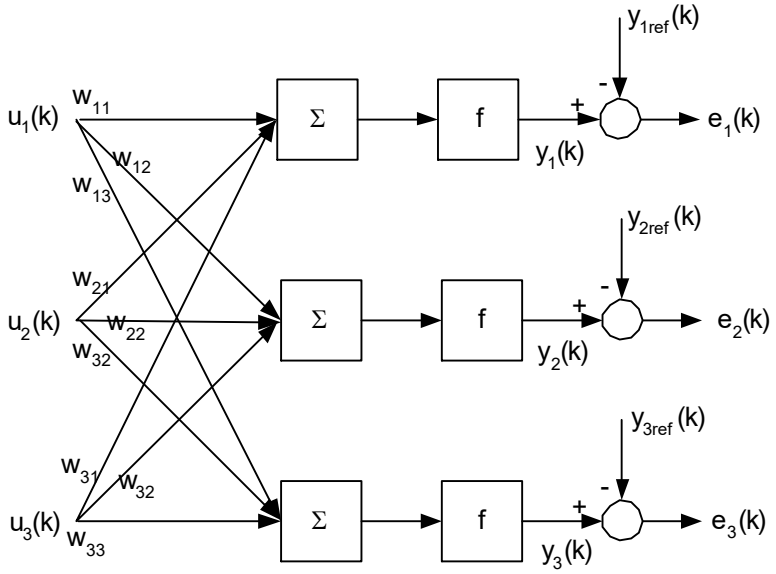
$$\left(\frac{\partial S}{\partial \mathbf{W}_1} : \frac{\partial S}{\partial \mathbf{b}_1}\right) = \begin{pmatrix} \frac{\partial S}{\partial w_{11}^{(1)}} & \frac{\partial S}{\partial w_{12}^{(1)}} \cdots \frac{\partial S}{\partial w_{1n}^{(1)}} : \frac{\partial S}{\partial b_1^1} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \frac{\partial S}{\partial w_{p1}^{(1)}} & \frac{\partial S}{\partial w_{p2}^{(1)}} \cdots \frac{\partial S}{\partial w_{pn}^{(1)}} : \frac{\partial S}{\partial b_p^1} \end{pmatrix},$$

а за (21) и (20) се използват аналогични означения.

Описаният алгоритъм определя градиента на средно-квадратичната грешка (18). Със съответна модификация чрез него може да се определи и якобианът на грешките e_i за $i = 1,2,...,m$.

1.6. Задача на нелинейните най-малки квадрати

Следният пример илюстрира задачата на нелинейните най-малки квадрати при една еднослойна невронна мрежа с три входа и три изхода (фиг.1.38).



фиг.1.38

Изходните грешки се определят от изразите:

$e_1(k) = y_{1ref}(k) - f\left(u_1(k)\,w_{11} + u_2(k)\,w_{12} + u_1(k)\,w_{13}\right)$, за $k = 1,...,N$

$e_2(k) = y_{2ref}(k) - f\left(u_1(k)\,w_{21} + u_2(k)\,w_{22} + u_1(k)\,w_{23}\right)$, за $k = 1,...,N$

$e_3(k) = y_{3ref}(k) - f\left(u_1(k)\,w_{31} + u_2(k)\,w_{32} + u_1(k)\,w_{33}\right)$, за $k = 1,...,N$

Чрез тях се определя средноквадратичната грешка на мрежата:

$$S(\mathbf{x}) = \sum_{k=1}^{N} \sum_{i=1}^{3} e_i^2(k),$$

където $\mathbf{x} = (w_{11}, w_{12} w_{13}, w_{2,1}, w_{2,2}, w_{2,2} w_{3,1}, w_{3,2}, w_{3,3})^T$.

Настройването на теглата w_{ij} изисква минимизиране на функцията на грешката S относно теглата $\mathbf{x}$ на невронната мрежа, т.е. решаване на задачата

$$\min_{\mathbf{x}} S(\mathbf{x}).$$

1.6.1. Постановка на задачата на нелинейните най-малки квадрати

Алгоритъмът на обратното разпространение (backpropagation) изчислява градиента на средноквадратичната грешка (функция на грешката) за една невронна мрежа с еднопосочно обработване на данните. След това може да се реализира съответна градиентна оптимизационна процедура за определяне теглата на мрежата, които минимизират грешката. При прилагането на оптимизационна процедура от по-висок ред (от нютонов тип) е необходимо да се пресметне и матрицата на Хесе за функция на грешката или нейно приближение. По-долу ще бъде разгледана основната идея на оптимизационните методи от този тип.

Нека е дадена функцията на грешката

(23) $$S(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^{m} e_i^2(\mathbf{x}),$$

където $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ и $\mathrm{e}(\mathbf{x}) = (e_1(\mathbf{x}), \ldots, e_m(\mathbf{x}))^T$ е векторът на грешките.

Поставя се следната оптимизационна задача:

(24) $$\min_{\mathrm{x} \in R^n} S(\mathbf{x}) = \min \frac{1}{2} \mathrm{e}^{\mathrm{T}}(\mathbf{x}) \mathrm{e}(\mathbf{x}) = \min \frac{1}{2} \sum_{i=1}^{m} e_i(\mathbf{x})^2,$$

която е известна като задача на нелинейните най- малки квадрати. За улеснение тук ще бъде разгледан случаят $m = n$.

1.6.2. Градиентен метод за решаване на задачата за нелинейните най-малки квадрати

Ако функцията (23) е два пъти непрекъснато диференцируема в зададена област, то тя може да бъде минимизирана чрез стандартна оптимизационна процедура:

(25) $$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \lambda_k \, grad\, S\left(\mathbf{x}^{(k)}\right), \quad k = 0,1,2,\ldots,$$

където $\mathbf{x}^{(0)}$ е начално приближение на търсеното решение, а $\lambda_k > 0$ е реална константа, наречена стъпка на спускане. В конкретния случай формулата (25) може да се преработи по следния начин. Използва се представянето за градиента

(26) $$gradS = \nabla S = \begin{pmatrix} \frac{\partial S}{\partial x_1} \\ \vdots \\ \frac{\partial S}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} e_i \frac{\partial e_i}{\partial x_1} \\ \vdots \\ \sum_{i=1}^{n} e_i \frac{\partial e_i}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial e_1}{\partial x_1} & \cdots & \frac{\partial e_n}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial e_1}{\partial x_n} & \cdots & \frac{\partial e_n}{\partial x_n} \end{pmatrix} \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} = \mathbf{J}^T \mathrm{e},$$

където $\mathbf{J}^T = \begin{pmatrix} \frac{\partial e_1}{\partial x_1} & \cdots & \frac{\partial e_n}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial e_1}{\partial x_n} & \cdots & \frac{\partial e_n}{\partial x_n} \end{pmatrix}$ е транспонираният якобиан на функциите $e_i(x)$ и формулата за градиента придобива вида

(27) $$grad\, S(\mathbf{x}) = \mathbf{J}^T(\mathbf{x})\mathrm{e}(\mathbf{x}).$$

Тогава итерационната процедура (25) се записва във вида

(28) $$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \lambda_k \mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathrm{e}\left(\mathbf{x}^{(k)}\right), \quad k = 0,1,2,\ldots.$$

Основната задача е изчисляването на матрицата $\mathbf{J(x)}$, което може да се реализира чрез алгоритъма с обратно разпространение на грешките. По-долу ще бъдат разгледани алгоритми от по-висок ред за решаване на поставената задача (24).

Представяне на матрицата на Хесе за функцията (23)

За функцията на грешката $S(\mathbf{x})=\frac{1}{2}\mathbf{e}^T(\mathbf{x})\mathbf{e}(\mathbf{x})$ се търси представяне за матрицата на Хесе $\nabla^2 S$, което ще бъде илюстрирано при $n=2$. Тогава

$$\nabla^2 S=\begin{pmatrix} \dfrac{\partial^2 S}{\partial x_1^2} & \dfrac{\partial^2 S}{\partial x_1 \partial x_2} \\ \dfrac{\partial^2 S}{\partial x_2 \partial x_1} & \dfrac{\partial^2 S}{\partial x_2^2} \end{pmatrix}=\begin{pmatrix} \displaystyle\sum_{i=1}^{2}\frac{\partial e_i}{\partial x_1}\frac{\partial e_i}{\partial x_1}+e_i\frac{\partial^2 e_i}{\partial x_1^2} & \displaystyle\sum_{i=1}^{2}\frac{\partial e_i}{\partial x_1}\frac{\partial e_i}{\partial x_2}+e_i\frac{\partial^2 e_i}{\partial x_1 \partial x_2} \\ \displaystyle\sum_{i=1}^{2}\frac{\partial e_i}{\partial x_2}\frac{\partial e_i}{\partial x_1}+e_i\frac{\partial^2 e_i}{\partial x_2 \partial x_1} & \displaystyle\sum_{i=1}^{2}\frac{\partial e_i}{\partial x_2}\frac{\partial e_i}{\partial x_2}+e_i\frac{\partial^2 e_i}{\partial x_2^2} \end{pmatrix}=$$

$$\begin{pmatrix} \dfrac{\partial e_1}{\partial x_1} & \dfrac{\partial e_2}{\partial x_1} \\ \dfrac{\partial e_1}{\partial x_2} & \dfrac{\partial e_2}{\partial x_2} \end{pmatrix}\begin{pmatrix} \dfrac{\partial e_1}{\partial x_1} & \dfrac{\partial e_1}{\partial x_2} \\ \dfrac{\partial e_2}{\partial x_1} & \dfrac{\partial e_2}{\partial x_2} \end{pmatrix}+e_1\begin{pmatrix} \dfrac{\partial^2 e_1}{\partial x_1^2} & \dfrac{\partial^2 e_1}{\partial x_1 \partial x_2} \\ \dfrac{\partial^2 e_1}{\partial x_2 \partial x_1} & \dfrac{\partial^2 e_1}{\partial x_2^2} \end{pmatrix}+e_2\begin{pmatrix} \dfrac{\partial^2 e_2}{\partial x_1^2} & \dfrac{\partial^2 e_2}{\partial x_1 \partial x_2} \\ \dfrac{\partial^2 e_2}{\partial x_2 \partial x_1} & \dfrac{\partial^2 e_2}{\partial x_2^2} \end{pmatrix}.$$

В резултат горното равенство може да се запише така:

(29) $$\nabla^2 S(\mathbf{x})=\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x})+\mathbf{H}(\mathbf{x}),$$

където $\mathbf{H}(\mathbf{x})=\sum_{i=1}^{2}e_i(\mathbf{x})\nabla^2 e_i(\mathbf{x})$, $\nabla^2 e_i(x)$ са матриците на Хесе за функциите $e_i(x)$.

1.6.3. Метод на Нютон за решаване на задачата за нелинейните най-малки квадрати

Развива се функцията $S(\mathbf{x})$ в околност на точката $\mathbf{x}^{(k)}$ в ред на Тейлър до третия член включително и се получава

(30) $$S(\mathbf{x})=S\left(\mathbf{x}^{(k)}\right)+\nabla S^T\left(\mathbf{x}^{(k)}\right)\left(\mathbf{x}-\mathbf{x}^{(k)}\right)+\frac{1}{2}\left(\mathbf{x}-\mathbf{x}^{(k)}\right)^T\nabla^2 S\left(\mathbf{x}^{(k)}\right)\left(\mathbf{x}-\mathbf{x}^{(k)}\right).$$

Това е квадратичен модел на $S(\mathbf{x})$ в околност на $\mathbf{x}^{(k)}$. За минимизиране на грешката $S(\mathbf{x})$ се прилага необходимото условие за екстремум. В резултат се получава

$$\nabla S\left(\mathbf{x}^{(k)}\right)+\nabla^2 S\left(\mathbf{x}^{(k)}\right)\left(\mathbf{x}-\mathbf{x}^{(k)}\right)=0.$$

Полага се $\mathbf{p}^{(k)} = \mathbf{x} - \mathbf{x}^{(k)}$, от което следва:

$$\nabla S\left(\mathbf{x}^{(k)}\right) + \nabla^2 S\left(\mathbf{x}^{(k)}\right)\mathbf{p}^{(k)} = 0$$

(31) $$\mathbf{p}^{(k)} = -\left(\nabla^2 S\left(\mathbf{x}^{(k)}\right)\right)^{-1}\nabla S\left(\mathbf{x}^{(k)}\right) = \left(\mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathbf{J}\left(\mathbf{x}^{(k)}\right) + \mathbf{H}\left(\mathbf{x}^{(k)}\right)\right)^{-1}\mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathbf{e}\left(\mathbf{x}^{(k)}\right).$$

Векторът $\mathbf{p}^{(k)}$ представлява нютоново направление на спускане.Тогава може да се приложи следната оптимизационна процедура:

(32) $$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \lambda_k \mathbf{p}^{(k)}, \text{ за } k = 0,1,2,\ldots,$$

където $\mathbf{p}^{(k)}$ е определено от (31), а $\lambda_k > 0$.

Трудността в използването на метода на Нютон се състои в това, че е необходимо аналитично представяне на $\mathbf{H(x)}$ в (29), а апроксимацията на тази матрица с крайни разлики е неефективна. Затова за предпочитане са следващите два метода, при които направлението на спускане $\mathbf{p}^{(k)}$ се получава чрез якобиана $\mathbf{J(x)}$.

1.6.4. Метод на Гаус-Нютон за решаване на задачата за нелинейните най-малки квадрати

Ако се предположи, че компонентите $r_i(\mathbf{x})$ на вектора $\mathbf{R(x)}$ са линейни функции на $\mathbf{x}$, то $\mathbf{H(x)} = \mathbf{0}$ и от (29) ще се получи съответна апроксимация за матрицата на Хесе от вида $\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x})$. В такъв случай от (32) , където

$$\mathbf{p}^{(k)} = -\left(\mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathbf{J}\left(\mathbf{x}^{(k)}\right)\right)^{-1}\mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathbf{e}\left(\mathbf{x}^{(k)}\right),$$

се получава методът на Гаус-Нютон

(33) $$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \lambda_k\left(\mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathbf{J}\left(\mathbf{x}^{(k)}\right)\right)^{-1}\mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathbf{e}\left(\mathbf{x}^{(k)}\right).$$

1.6.5. Метод на Левенберг-Маркуарт за решаване на задачата за нелинейните най-малки квадрати

При метода на Левенберг-Маркуарт направлението на спускане от формулата (33) се модифицира така:

$$\mathbf{p}^{(k)} = -\left(\mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathbf{J}\left(\mathbf{x}^{(k)}\right) + \mu_k\mathbf{E}\right)^{-1}\mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathbf{e}\left(\mathbf{x}^{(k)}\right), \text{ където } \mu_k > 0,$$

т.е. се прибавя корекцията $\mu_к \mathbf{E}$ ($\mathbf{E}$ - единична матрица) и методът на Левенберг-Маркуарт има вида:

(34) $$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \lambda_k \left(\mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathbf{J}\left(\mathbf{x}^{(k)}\right) + \mu_k \mathbf{E}\right)^{-1} \mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathrm{e}\left(\mathbf{x}^{(k)}\right)$$

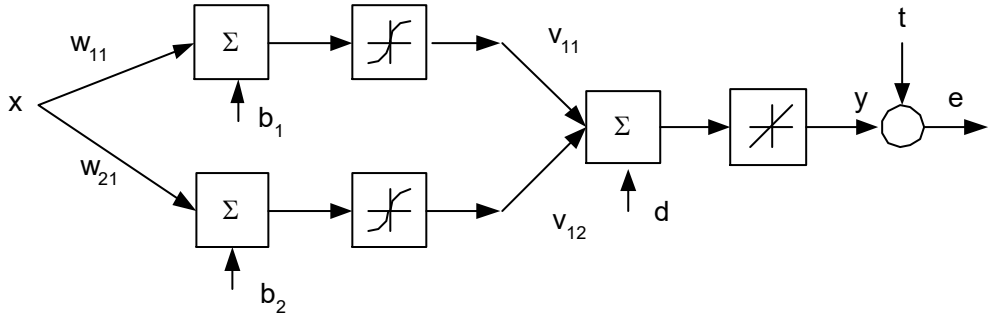
Очевидно, при $\mu_k = 0$ от (34) се получава методът на Гаус-Нютон, а при μ_k голямо се получава градиентен метод, т.е. $\mathbf{p}^{(k)} \approx -\mu_k \mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathrm{e}\left(\mathbf{x}^{(k)}\right)$.

Съществено е, че при подходящ избор на μ_k добавянето на члена $\mu_k \mathbf{E}$ води до положителна определеност на матрицата $\mathbf{J}^T\left(\mathbf{x}^{(k)}\right)\mathbf{J}\left(\mathbf{x}^{(k)}\right) + \mu_k \mathbf{E}$ и съответното $\mathbf{p}^{(k)}$ е направление на спускане.

1.7. Приложение на нелинейни невронни мрежи за апроксимиране на функции

1.7.1. Апроксимиране на криви с невронни мрежи

Нека функцията $t = f(x)$ е дефинирана и непрекъсната в интервала $x \in [0,T]$. От тази функция са зададени N отчета $t(k) = f(x(k))$, за $k = 1,...,N$, които са елементи на векторите $\mathbf{x} = (x(1),(2),...,x(N))^T$ и $\mathbf{t} = (t(1),t(2),...,t(N))^T$. Това са съответно входните и еталонните данни, с които ще се обучава мрежата. Избрана е невронна мрежа с двуслойна структура, която има два неврона в първия слой и един неврон във втория. В първия слой активиращата функция е сигмоидната функция $F(x)$, а функцията от втория слой е линейна (фиг.1.39).



фиг.1.39

Поставя се задачата да се намерят теглата w_{i1} и b_i за $i=1,2$ от първия слой и v_{1i} и d за $i=1,2$ от втория слой, така че функцията, реализирана от мрежата, да апроксимира данните $(x(k),t(k))$ за $k=1,...,N$. Описанието на мрежата е, както следва:

$$y(k)=v_{11}F(w_{11}x(k)+b_1)+v_{12}F(w_{12}x(k)+b_2)+d;\quad k=1,2,...,N.$$

Ако на входа на мрежата се подаде векторът от входни данни $\mathbf{x}=(x(1),x(2),...,x(N))$, а на изхода се сравняват реализираните изходи $\{y(1),...,y(N)\}$ с еталонните $\mathbf{t}=\{t(1),...,t(N)\}$, то средноквадратичната грешка има вида

(35) $$S(\mathbf{w})=\frac{1}{2}\sum_{k=1}^{N}(t(k)-y(k))^2=\frac{1}{2}\sum_{k=1}^{N}e(k)^2,$$

където $\mathbf{w}=(w_{11},w_{21},b_1,b_2,v_{11},v_{12},d)$ е векторът от теглата. Поставя се задачата на нелинейните най- малки квадрати

(36) $$\min_{\mathbf{w}} S(\mathbf{w}).$$

За решаването на тази задача може да се използва процедура от градиентен тип, като градиентът се изчислява с метода на обратното разпространяване на грешките.

Определяне броя на невроните в първия слой

При апроксимиране на функции с невронни мрежи могат да възникнат два практически проблема. Първият се дължи на избирането на прекалено голям брой неврони в първия слой, а вторият- на недостатъчен брой неврони в този слой. Изясняването на тези два проблема ще стане чрез следния пример.

Пример 1. Да се апроксимира функцията $f(x)=2x^3-1.5x+0.5$ в интервала $x\in[-1,1]$, като се използва невронната мрежа от фиг.1.39. Избират се следните вектори от входни данни $\mathbf{X}=(-1,-0.5,0,0.5,1)$ и от еталонни изходи $\mathbf{T}=(f(-1),f(-0.5),f(0),f(0.5),f(1))=(0,1,0.5,0,1)$. При това се разглеждат два случая:

а) мрежата е с 2 неврона в първия слой;

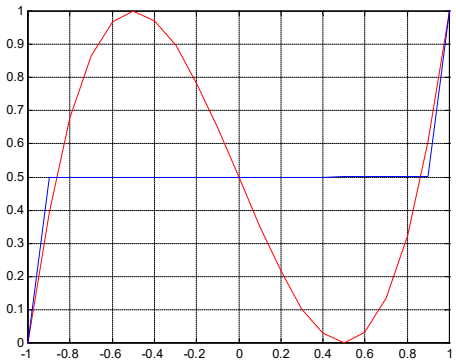
б) мрежата е с 10 неврона в първия слой.

Следва програмата ,с която се реализира тази апроксимация.

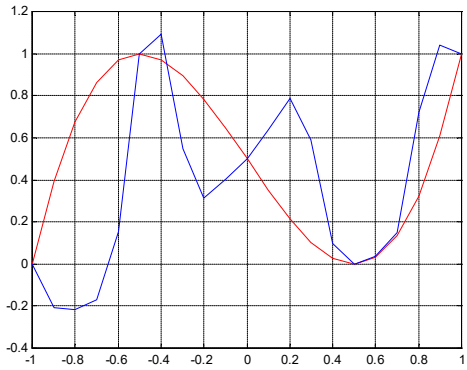
```
clear all
x=[-1 -0.5 0 0.5 1];
y=2*x.^3-1.5*x+0.5;
x1=-1:0.1:1;
```

```
y1=2*x1.^3-1.5*x1+0.5;
for k=1:20
m=2; %m=10
linnet=newff(minmax(x),[m,1],{'logsig','purelin'});
m=1000;
err=0;
linnet.trainparam.epochs=m;
linnet.trainparam.goal=err;
linnet=train(linnet,x,y);
ysim=sim(linnet,x1);
plot(x1,y1,'r',x1,ysim),grid,figure(gcf),pause
end
```

Резултатът от работата на програмата е даден на фиг. 140а,б.



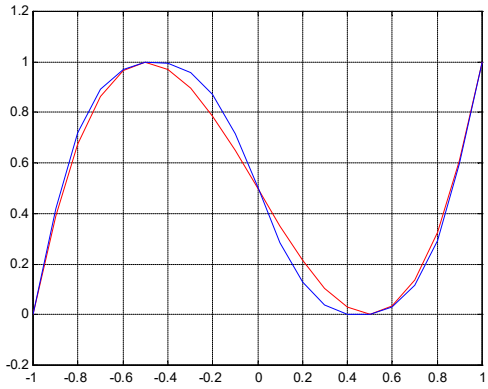
фиг.1.40а



фиг.1.40б

На фиг.1.40а и фиг.1.40б са изобразени еталонната функция и функцията, реализирана от мрежата съответно при 2 и 10 неврона в първия слой. В първия случай (фиг.1.40а) функцията, реализирана от мрежата, не възпроизвежда добре еталонната функция, тъй като при два неврона в първия слой функцията, реализирана от мрежата, не може да има повече от един екстремум, докато еталонната има два екстремума.

Във втория случай (фиг.1.40б) апроксимацията отново не е добра. Тук, функцията реализирана от мрежата, минава през зададените точки, но вследствие на варирането на останалите свободни тегла, тя има неопределена форма между еталонните точки. Най-добрата апроксимация се постига при 3 неврона в първия слой (фиг.1.41).



фиг.1.41

1.7. Апроксимиране на повърхнини с невронни мрежи

Нека функцията $z = f(x_1, x_2)$ е непрекъсната в правоъгълната област $\{0 \le, x_1 \le T; 0 \le x_2 \le T\}$. Тя е дискретизирана с постоянни стъпки $\Delta x_1; \Delta x_2$ и са получени точките от таблица 2.

	$x_1(1)$	…	$x_1(n)$
$x_2(1)$	$z(1,1)$	…	$z(n,1)$
⋮	⋮		⋮
$x_2(n)$	$z(1,n)$	…	$z(n,n)$

Таблица 2

Данните от таблицата се подреждат в матрици, както следва:

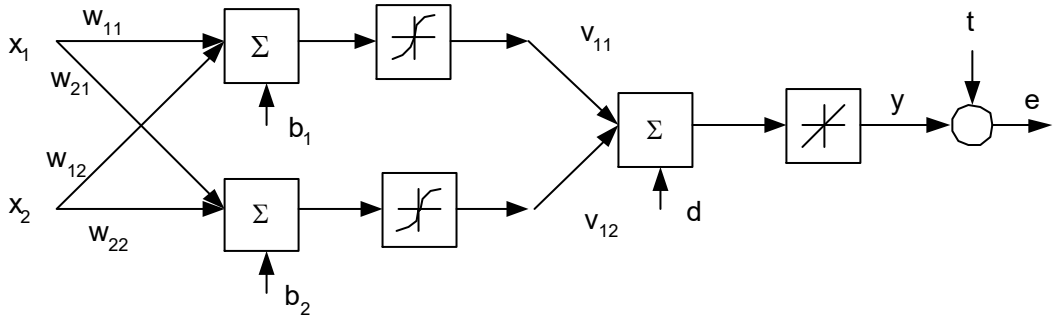
(37) $$\mathbf{X} = \begin{pmatrix} x_1(1) & \dots & x_1(1) & \dots & x_1(n) & \dots & x_1(n) \\ x_2(1) & \dots & x_2(n) & \dots & x_2(1) & \dots & x_2(n) \end{pmatrix} = \begin{pmatrix} X_1(1) & \cdots & X_1(n^2) \\ X_2(1) & \cdots & X_2(n^2) \end{pmatrix}$$

$$\mathbf{t} = (z(1,1),...,z(1,n),...,z(n,1),...,z(n,n)) = \left(t(1),...,t(n^2)\right).$$

Тези матрици са съответно входните и еталонните данни, с които ще се обучава мрежата.

Структурата на мрежата е от два неврона със сигмоидална активираща функция $F(x)$ в първия слой и един линеен неврон във втория слой (фиг.1.42).

Поставя се задачата да се намерят теглата w_{i1}, w_{2i} и b_i за $i = 1,2$ от първия слой и v_{1i} и d за $i = 1,2$ от втория слой, така че средноквадратичната грешка на мрежата да се минимизира.



фиг.1.42

Нека входните вектори са стълбовете на матрицата $\mathbf{X}$, за които съгласно (37) са въведени следните означения: $\mathbf{X}(k) = \left(X_1(k), X_2(k)\right)^T$, за $k = 1,...,n^2$ и е пресметната изходната променлива на мрежата $y(k)$ от израза

$$y(k) = v_{11}F(w_{11}X_1(k) + w_{12}X_2(k) + b_1) + v_{12}F(w_{21}X_1(k) + w_{22}X_2(k) + b_2) + d\,.$$

Средноквадратичната грешка е

$$S(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{n^2}(t(k) - y(k))^2 = \frac{1}{2}\sum_{l=1}^{n^2}e(k)^2\,,$$

където $\mathbf{w} = (w_{11}, w_{12}, w_{21}, w_{22}, b_1, b_2, v_{11}, v_{12}, d)$ е векторът от теглата на мрежата. Поставя се задачата на нелинейните най-малки квадрати

$$\min_{\mathbf{w}} S(\mathbf{w})\,,$$

за която по метода на обратното разпространение на грешките се изчислява градиентът, след това оптимизационната задача се решава с алгоритъм от градиентен тип.

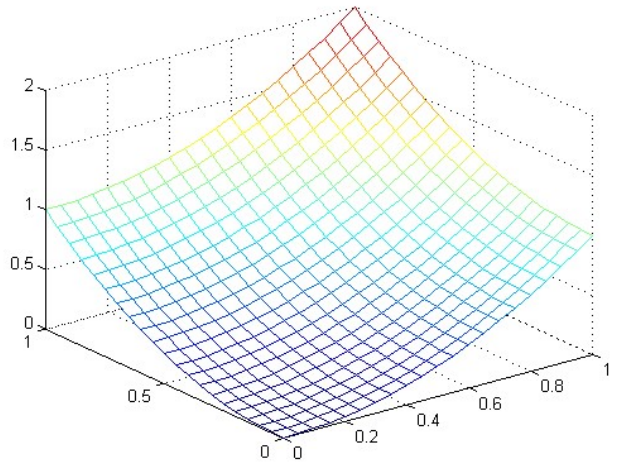
Пример 2. Да се апроксимира функцията $f(x_1, x_2) = {x_1}^2 + {x_2}^2$ в областта $\{0 \le, x_1 \le 1; 0 \le x_2 \le 1\}$, като се използва невронната мрежа от фиг.1.42. Точките, в които се изчислява функцията, са получени със стъпки $\Delta x_1 = 0{,}05$ и $\Delta x_2 = 0{,}05$.

Програмата, с която се реализира тази апроксимация, е следната:

```
clear all
x=0:0.05:1;y=0:0.05:1;
n=length(x);
N1=1;
for i=1:n
   for j=1:n
      r(i,j)=x(j)^2+y(i)^2;
      inp(1,N1)=x(j);
      inp(2,N1)=y(i);
```

```
      ref(N1)=r(i,j);
      N1=N1+1;
   end;
end;
mesh(x,y,r),figure(gcf),pause
l=input('l=');
net1=newff(minmax(inp),[l,1],{'tansig','purelin'},'trainlm');
net1.trainparam.epochs=50;
net1.trainparam.goal=1e-3;
for N2=1:5
   net1=train(net1,inp,ref);
   for i=1:n
      for j=1:n
         z(i,j)=sim(net1,[x(j);y(i)]);
      end;
   end;
   mesh(x,y,z),figure(gcf),pause
end;
```

На фиг.1.43 е даден резултатът от изпълняване на програмата.



фиг. 1.43

1.8. Приложение на невронни мрежи за прогнозиране на времеви редове

Под времеви ред се разбира една редица от данни от вида $\{x(k)\}_{k=0}^{N}$. Най често стойностите на тази редица представляват отчети от един непрекъснат сигнал $x(t)$, свалени през равни интервали от време. Оттам и наименованието на тази редица.

Най-често сигналът $x(t)$ е някакъв динамичен (във времето) процес, който се описва с математичен модел или се получава в резултат от някакъв експеримент. В случая моделът е например динамична система, описана с диференциалното уравнение

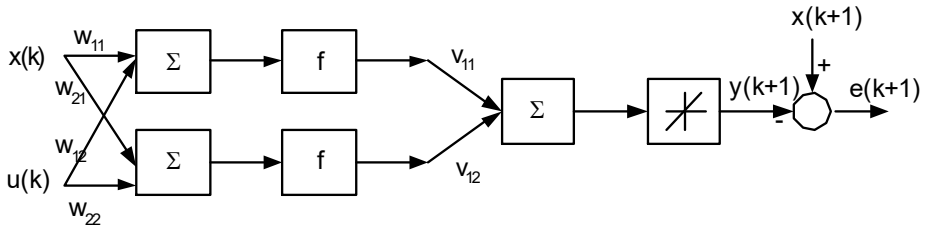
(38) $\dot{x}(t) = f(t, x(t), u(t))$, за $t \in [0, T]$ и $x(0) = 0$,

със зададено външно въздействие $u(t)$.

За получаването на времеви ред от тази динамична система може да се използва числен експеримент. За целта със стъпка $h = T / N$ се разделя интервалът $[0, T]$ на N равни части с точките $t_k = k\,h$ за $k = 0,1,..., N$ и чрез формулата на Ойлер се получава численото решение на (38)

(39) $x(h(k+1)) = x(hk) + h\,f\big(hk, x(hk), u(hk)\big)$ за $k = 0,1,..., N$ и $x(0) = 0$.

Поставя се задачата да се разработи невронна мрежа (предиктор), която по известни стойности $x_k = x(hk)$ и $u_k = u(hk)$ в k-та стъпка да определя достатъчно добро приближение y_{k+1} на $x_{k+1} = x(h(k+1))$ в $k+1$-та стъпка, т.е. $y_{k+1} \approx x_{k+1}$. За целта може да се използва двуслойната невронна мрежа от фиг.1.44.



фиг.1.44

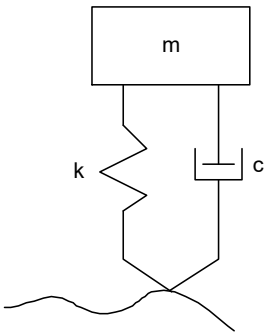
Функцията, реализирана от тази мрежа, е

(40) $$y(k+1) = \sum_{i=1}^{2} v_{1i}\, f(w_{i1}\, x(k) + w_{i2}\, u(k)),\ k = 0,1,2,..., N.$$

Търсят се такива тегла на мрежата w_{i1}, w_{i2} и v_{1i} за $i = 1,2$, при които изходът $y(k+1)$ на мрежата да апроксимира еталона $x(k+1)$. Следващият пример илюстрира тази идея.

Пример 1. Моделира се динамичната система от фиг.1.45, която е подложена на ударно входно въздействие от вида

(41) $$u(t) = 21.2\, t^{1.5} e^{-4t} \sin(e^{1.89t} - 1).$$



Фиг.1.45

Тази система се описва със следната система ОДУ:

$$\dot{x}_1 = x_2$$

(42)

$$\dot{x}_2 = -\frac{c}{m}x_2 - \frac{k}{m}x_1 + \frac{1}{m}(k\,u(t) + c\,\dot{u}(t))$$

с начални условия

(43) $$x_1(0) = x_2(0) = 0\,,$$

където

m - маса;

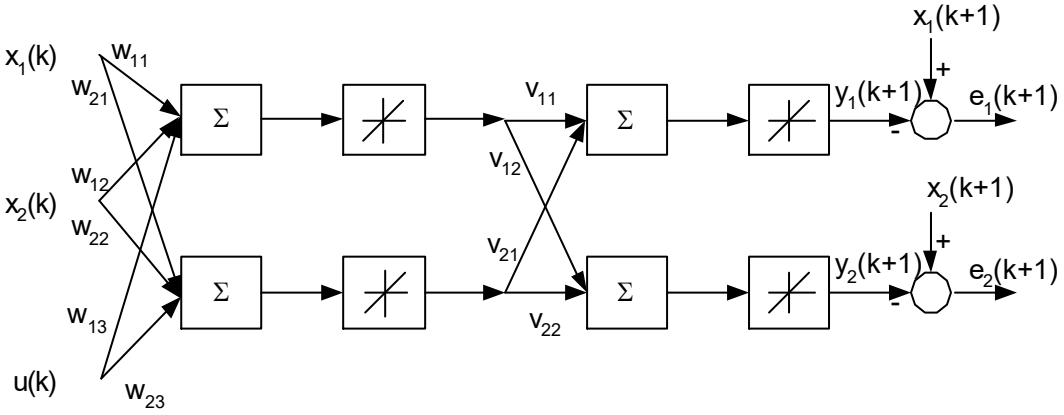
c - коефициент на демпфиране;

k - коефициент на еластичност;

$x_1(t)$ - преместване на масата m;

$x_2(t) = \dot{x}_1(t)$ - скорост на масата m.

На фиг.1.46 е даден избраният невронен модел (линейна мрежа)



фиг.1.46

Програмата, с която се реализира поставената задача, има пет обособени части. В първата се решава системата ОДУ (42-43), като външното въздействие $u(t)$ е бял гаусов шум и се получават обучаващите времеви редове. Във втората част се създава структурата на мрежата, отговаряща на фиг.1.46. Тя се обучава при

използването на генерираните данни. В третата част се решава системата ОДУ при ударно външно въздействие $u(t)$, зададено в (41). В следващите две части мрежата се използва първо като невронен предиктор, а след това като невронен модел. Разликата в двата случая е следната. При симулация като невронен предиктор на всяка времева стъпка k на входа на мрежата се подава стойността на експериментално снетия входен процес $x(k)$ и на изхода се получава реализираният от мрежата процес $y(k+1)$. Получените времеви редове $\{x(k)\}_{k=0}^{N}$ и $\{y(k)\}_{k=0}^{N}$ са показани на фиг.1.47. При симулацията с мрежата като невронен модел на всяка времева стъпка k на входа на мрежата се подава стойността на симулирания от мрежата изход $y(k)$ /получен в предходния момент от време/ и на изхода се получава реализираният от мрежата изход $y(k+1)$ /получен в следващия момент от време/. Получените времеви редове $\{x(k)\}_{k=0}^{N}$ и $\{y(k)\}_{k=0}^{N}$ са показани на фиг.1.48.

```
clear all
global c k m u1
%Генериране на обучаващи данни при гаусово входно въздействие
u(t)
c=40;k=25;m=10;
n=1000;
y0=[0;0];timestep=0.05;t0=0;tk=2*timestep;
t(1)=0;y(1,1)=0;y(2,1)=0;
for i=1:n
   u1=normrnd(0,1);
   u(i)=u1;
   [t1,y1]=ode15s('fun1',[t0:timestep:tk],y0);
      t(i+1)=tk;
   y0=y1(3,:);
   t0=tk;tk=tk+2*timestep;
   y(1,i+1)=y1(3,1);
   y(2,i+1)=y1(3,2);
end;
Pm=[y(1,1:n-1);y(2,1:n-1);u(1:n-1)];
Tm=[y(1,2:n);y(2,2:n)];
%Обучение на мрежата с генерираните данни
S1=2;
[S2,Q] = size(Tm);
Pm,
nets = newff(minmax(Pm),[S1 S2],{'purelin' 'purelin'},'trainlm');
nets.trainParam.epochs = 50;       % Maximum number of epochs to
train.
nets.trainParam.goal =0.00001;      % Mean-squared error goal.
```

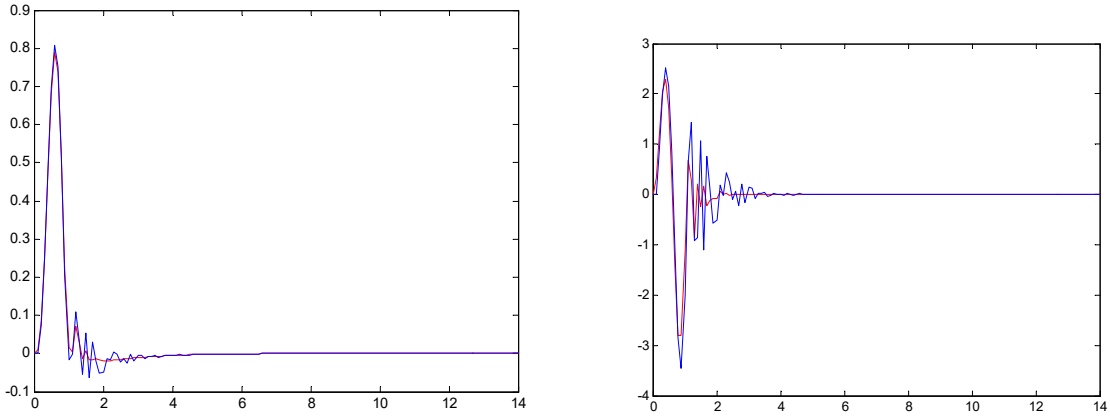
```
nets = train(nets,Pm,Tm);
%Генериране на обучаващи данни при ударно входно въздействие u(t)
c=40;k=25;m=10;
n=140;
t0=0;tk=n*2*timestep;y0=[0;0];
[t1,y1]=ode15s('fun2',[t0:2*timestep:tk],y0);
%Симулиране на мрежата като невронен предиктор
init_state=[0;0];
time = 0:2*timestep:n*2*timestep;
state = init_state;
states = zeros(2,length(time));
states(:,1) =state;
for i=2:n
   u0=udar(time(i-1));
   state=sim(nets,[state;u0]);
   states(:,i) = state;
   state=y1(i,1:2)';
end;
plot(t1,y1(:,1),'r',time,states(1,:)),figure(gcf),pause
plot(t1,y1(:,2),'r',time,states(2,:)),figure(gcf),pause
%Симулиране на мрежата като невронен модел
init_state=[0;0];
time = 0:2*timestep:n*2*timestep;
state = init_state;
states = zeros(2,length(time));
states(:,1) =state;
for i=2:n
   u0=udar(time(i-1));
   state=sim(nets,[state;u0]);
   states(:,i) = state;
end;
plot(t1,y1(:,1),'r',time,states(1,:)),figure(gcf),pause
plot(t1,y1(:,2),'r',time,states(2,:)),figure(gcf),pause
```

Така описаната главна програма ползва следните три помощни програми: udar, fun1 и fun2.

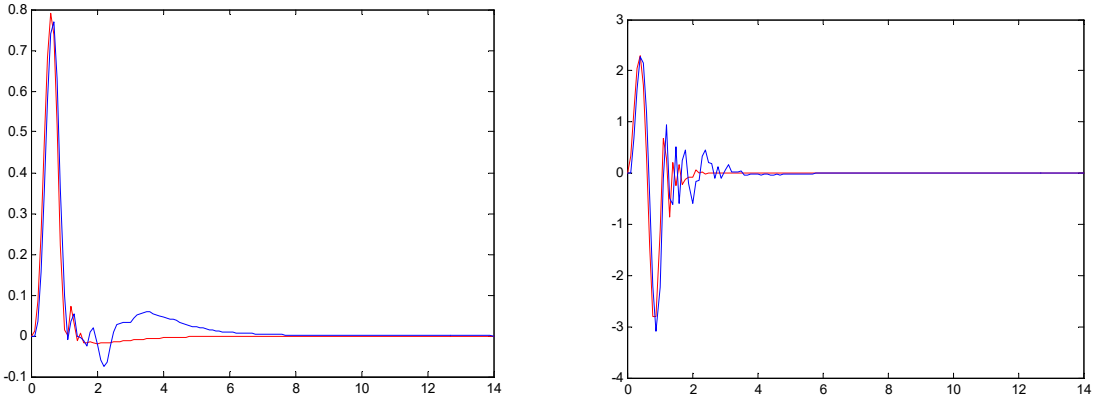
```
%Програма udar.m
function [uu]=udar(t)
global c k m
x01=21.2*t^1.5;
x02=exp(-4*t);
x03=exp(1.89*t);
x04=sin(x03-1);
x0=x01*x02*x04;
x0t=21.2*1.5*(t)^0.5*x02*x04+...
   x01*(-4*x02)*x04+...
   x01*x02*cos(x03-1)*1.89*x03;
uu=(k*x0+c*x0t)/m;
```

```
%Програма fun1.m
function [df]=fun1(t,y)
global c k m u1
df(1)=y(2);
df(2)=-(c/m)*y(2)-(k/m)*y(1)+u1;
df=df';
%Програма fun2.m
function [df]=fun2(t,y)
global c k m u1
u1=udar(t);
df(1)=y(2);
df(2)=-(c/m)*y(2)-(k/m)*y(1)+u1;
df=df';
```

В програмата udar е реализирано ударното входно въздействие от (41),а в програмите fun1 и fun2 се описват десните части на системата (42), съответно с входно смущение бял гаусов шум или ударно входно въздействие $u(t)$.



Фиг.1.47

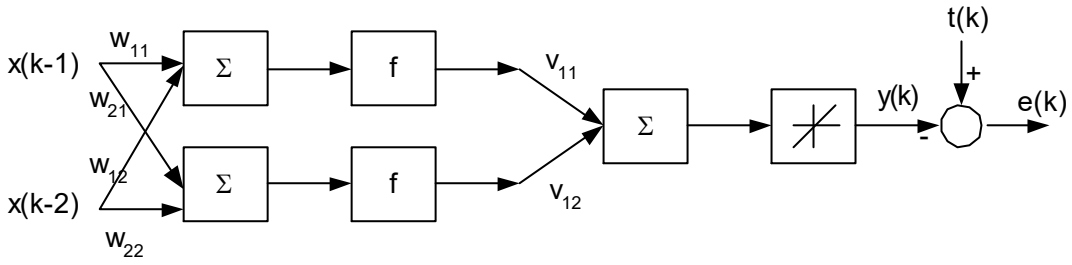


Фиг.1.48

Пример 2. Разглежда се диференчното уравнение

$$z(k) = 0.5x(k-1) - 0.6x(k-2).$$

Еталонният времеви ред се получава по формулата $t(k) = z^2(k)$. Избраният модел е нелинейната невронна мрежа със структура, показана на фиг.1.49.



Фиг.1.49

Целта е след обучението на невронната мрежа изходът на мрежата $y(k)$ и еталонът, получен от моделираната дискретна динамична система $t(k)$, да са близки в средноквадратичен смисъл.

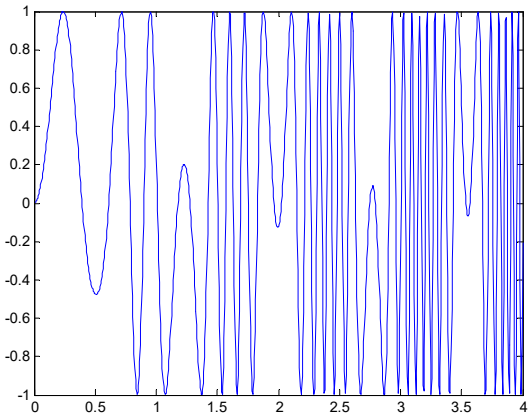
Програмата, с която се реализира поставената задача, е следната:

```
clear all
t=0:0.005:4;
p=sin(sin(t*4).*t*8);
n=length(p);
y(1)=0;p(1)=0;y(2)=0;
for k=3:n
    y(k)=p(k-1)-0.6*p(k-2);
end;
T=y.^2;
plot(t,T),figure(gcf),pause
S1=input('S1='),pause
P=[0 p(1:n-1);0 0 p(1:n-2)];
netp=newff(minmax(P),[S1 1],{'logsig','purelin'});
    netp.trainParam.epochs = 200;
     netp.trainParam.show = 1;
     netp=train(netp,P,T);
     t1=4:0.005:6;
     x1=sin(sin(t1*4).*t1*8);
     m=length(t1);
     P1=[0 x1(1:m-1);0 0 x1(1:m-2)];
     y(1)=0;x(1)=0;y(2)=0;
```

```
for k=3:m
    y1(k)=x1(k-1)-0.6*x1(k-2);
end;
     Y1=sim(netp,P1);
     plot(t1,Y1,t1,y1.^2,'*'),figure(gcf),pause
```

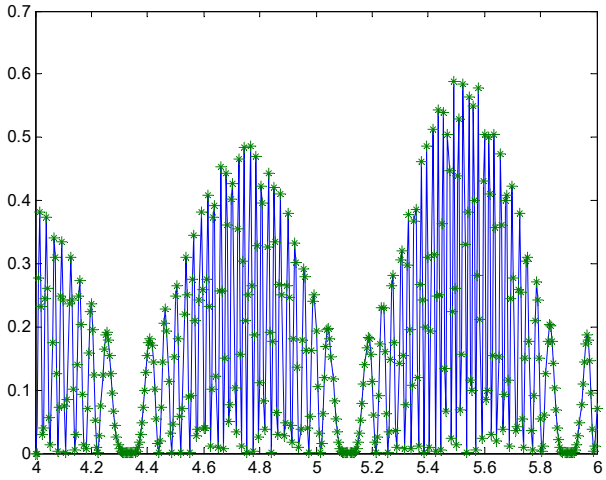
Входният сигнал на динамичната система x_k е показан на фиг.1.50.

$$x_k = \sin(\sin(4t_k)8t_k) \text{ за } t_k = 0.005k$$



фиг.1.50

Еталонът и реализираният от невронната мрежа процес са показани на фиг.1.51



Фиг.1.51

1.9. Архитектура и основни теоретични резултати за рекурентни невронни мрежи

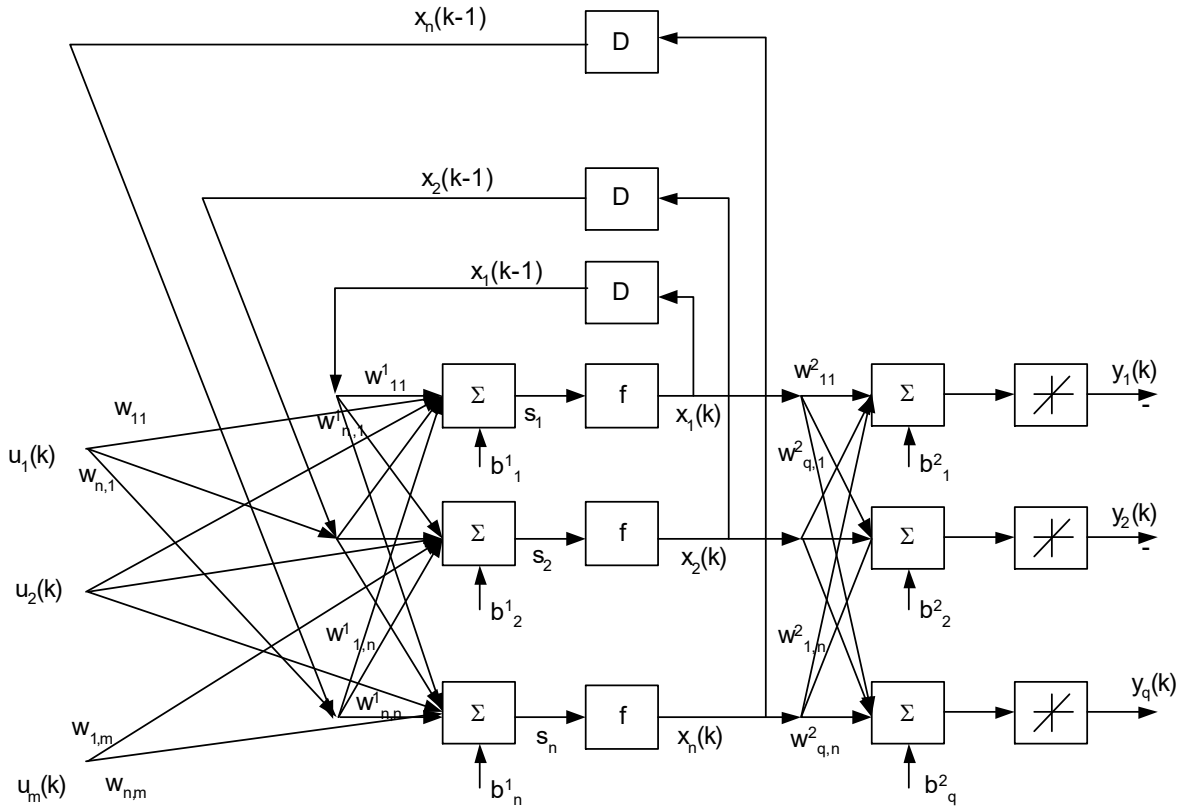
Един клас невронни мрежи се характеризират с това, че изходите на даден слой могат да бъдат свързани с неговите входове или с входовете на невроните от предходен слой, т.е в тях съществуват обратни връзки (рекурсия). Поради това те се наричат рекурентни невронни мрежи. Съществуват разнообразни рекурентни невронни структури, но според начина на обработване на информацията се разделят на дискретни и непрекъснати. Дискретните се описват със системи рекурентни уравнения, а непрекъснатите- със системи диференциални уравнения и следователно самите те са динамични системи. Последното, заедно с това, че изчислителният алгоритъм на мрежата е разпаралелен, правят рекурентните невронни мрежи предпочитани за моделиране на динамични системи. Тук трябва да се отбележи, че тъй като рекурентните невронни мрежи са динамични, методите за обучение на тези мрежи са значително по-сложни от прилаганите методи при невронните мрежи с еднопосочно обработване на данните.

1.9.1. Архитектура на рекурентни невронни мрежи

Дискретни рекурентни невронни мрежи

Дискретните рекурентни невронни мрежи се използват основно за апроксимиране на времеви редове. Освен това, както вече беше въведен в първа глава , с D се означава оператор закъснение, т.е. изходът се подава на входа с една времева стъпка назад. Това може да се зададе и по следния начин. Ако $\{y(k)\}$ е времеви ред, то като се приложи оператор закъснение за $y(k)$, следва връзката $Dy(k) = y(k-1)$ или съответно $z^{-1}y(k) = y(k-1)$. Прието е първото означение, тъй като е предпочитано в MATLAB.

Пример за рекурентна архитектура е мрежата на Елман, която е изобразена на фиг. 1.52. Скритият слой на тази мрежа е с нелинейни активиращи функции, а изходният слой е с линейни активиращи функции.



фиг. 1.52

Невронната мрежа се описва със следната система от рекурентни уравнения:

$$\begin{pmatrix} s_1(k) \\ s_2(k) \\ \dots \\ s_n(k) \end{pmatrix} = \begin{pmatrix} w_{11}\ w_{12} \dots w_{1m} \\ w_{21}\ w_{22}\ \dots w_{2m} \\ \dots\dots\dots\dots\dots \\ w_{n1}\ w_{n2}\ \dots w_{nm} \end{pmatrix} \begin{pmatrix} u_1(k) \\ u_2(k) \\ \dots \\ u_m(k) \end{pmatrix} + \begin{pmatrix} w^1_{11}\ w^1_{12} \dots w^1_{1n} \\ w^1_{21}\ w^1_{22} \dots w^1_{2n} \\ \dots\dots\dots\dots\dots \\ w^1_{n1}\ w^1_{n2} \dots w^1_{nn} \end{pmatrix} \begin{pmatrix} x_1(k-1) \\ x_2(k-1) \\ \dots \\ x_n(k-1) \end{pmatrix} + \begin{pmatrix} b^1_1 \\ b^1_2 \\ \dots \\ b^1_n \end{pmatrix}$$

$$\begin{pmatrix} x_1(k) \\ x_2(k) \\ \dots \\ x_n(k) \end{pmatrix} = \begin{pmatrix} f(s_1(k)) \\ f(s_2(k)) \\ \dots \\ f(s_n(k)) \end{pmatrix}$$

$$\begin{pmatrix} y_1(k) \\ y_2(k) \\ \dots \\ y_q(k) \end{pmatrix} = \begin{pmatrix} w^2_{11}\ w^2_{12} \dots w^2_{1n} \\ w^2_{21}\ w^2_{22} \dots w^2_{2n} \\ \dots\dots\dots\dots\dots \\ w^2_{q1}\ w^2_{q2} \dots w^2_{qn} \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \\ \dots \\ x_n(k) \end{pmatrix} + \begin{pmatrix} b^2_1(k) \\ b^2_2(k) \\ \dots \\ b^2_q(k) \end{pmatrix}$$

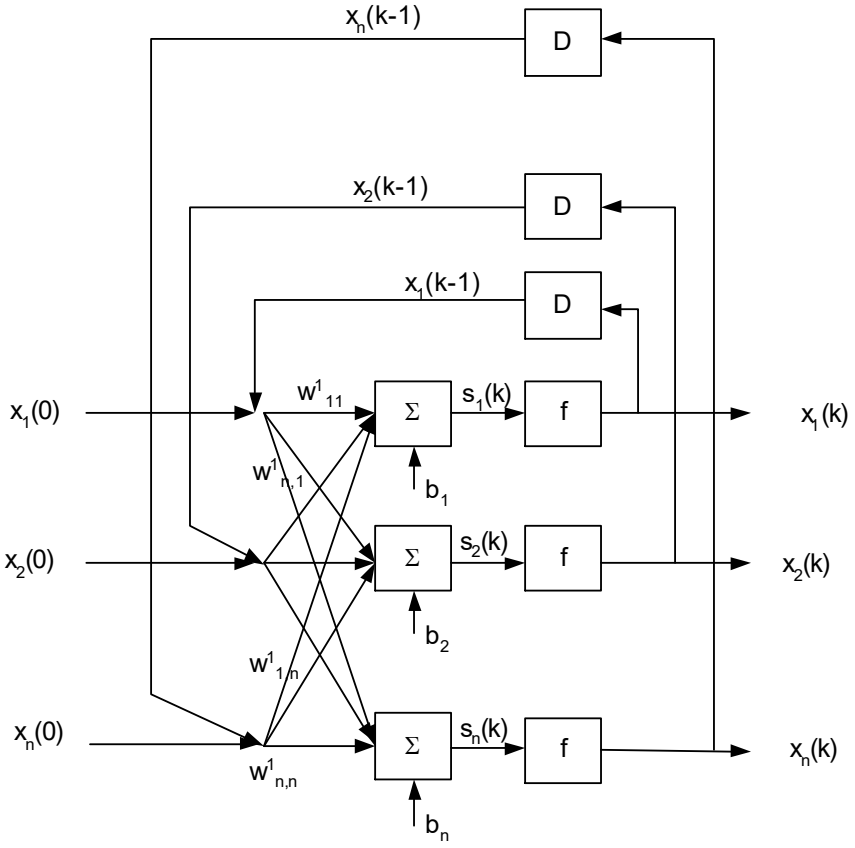
Тази система се представя в матрична форма:

$$\mathbf{x}(k) = \mathbf{f}(\mathbf{W}\,\mathbf{u}(k) + \mathbf{W}_1\,\mathbf{x}(k-1) + \mathbf{b}^1)$$

$$\mathbf{y}(k) = \mathbf{W}_2\,\mathbf{x}(k) + \mathbf{b}^2,$$

където **f(s)** е векторната функция $\mathbf{f}(\mathbf{s}) = (f(s_1), f(s_2), ..., f(s_n))^T$.

Пример за рекурентна мрежа на Хопфилд е даден на фиг.1.53. Тази мрежа е еднослойна със специфична активираща функция - “линейна с насищане”.



фиг.1.53

Системата рекурентни уравнения за тази мрежа е следната:

$$\mathbf{x}(k) = \mathbf{f}(\mathbf{W}\ \mathbf{x}(k-1) + \mathbf{b}),\ \text{за}\ k = 1,2,...\ ,$$

където

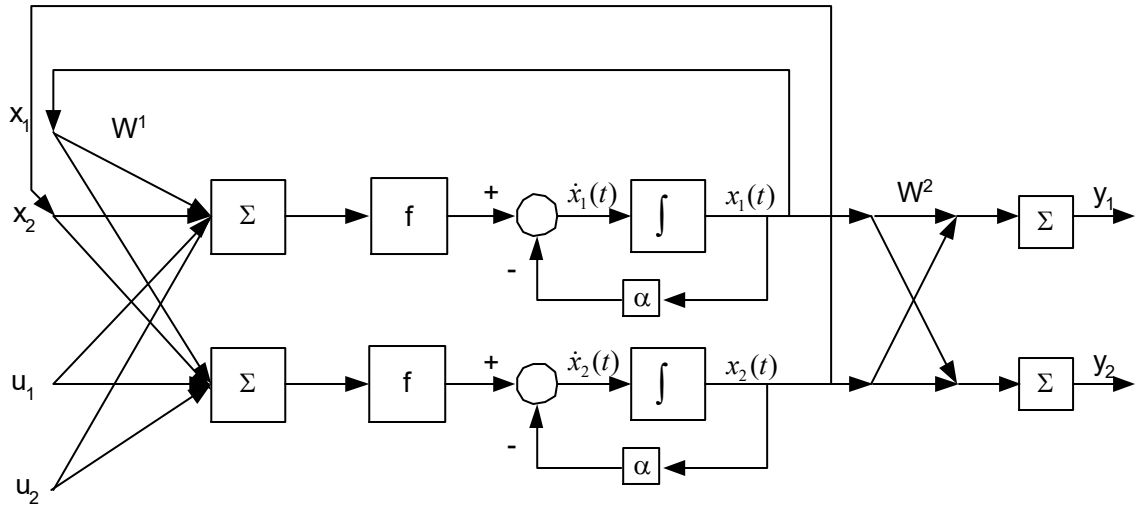
$$\mathbf{x} \in R^n,\ \mathbf{b} \in R^n \quad \text{и} \quad \mathbf{W} = \begin{pmatrix} w_{11}^1\ w_{12}^1 ... w_{1n}^1 \\ w_{21}^1\ w_{22}^1 ... w_{2n}^1 \\ ..... \\ w_{n1}^1\ w_{n2}^1 ... w_{nn}^1 \end{pmatrix},$$

f(s) е векторната функция $\mathbf{f}(\mathbf{s}) = (f(s_1), f(s_2), ..., f(s_n))^T$, началното условие $\mathbf{x}(0)$ е зададено и активиращата функция на мрежата се дефинира по следния начин:

$$f(x) = \begin{cases} -1, \text{ за } x \in (-\infty, -1) \\ x, \text{ за } x \in [-1,1] \\ 1, \text{ за } x \in (1, +\infty) \end{cases} .$$

Непрекъснати (динамични) невронни мрежи

На фиг.1.54 е представена една непрекъсната рекурентна невронна мрежа.



Фиг. 1.54

Математическото описание на невронната мрежа от фиг. 1.54 е:

$$\dot{x}_1(t) = -\alpha x_1(t) + f\left(w^1_{1,1}x_1(t) + w^1_{1,2}x_2(t) + w^1_{1,3}u_1(t) + w^1_{1,4}u_2(t)\right)$$
$$\dot{x}_2(t) = -\alpha x_2(t) + f\left(w^1_{2,1}x_1(t) + w^1_{2,2}x_2(t) + w^1_{2,3}u_1(t) + w^1_{2,4}u_2(t)\right)$$
$$y_1(t) = w^2{}_{1,1}x_1(t) + w^2{}_{1,2}x_2(t)$$
$$y_2(t) = w^2{}_{2,1}x_1(t) + w^2{}_{2,2}x_2(t),$$

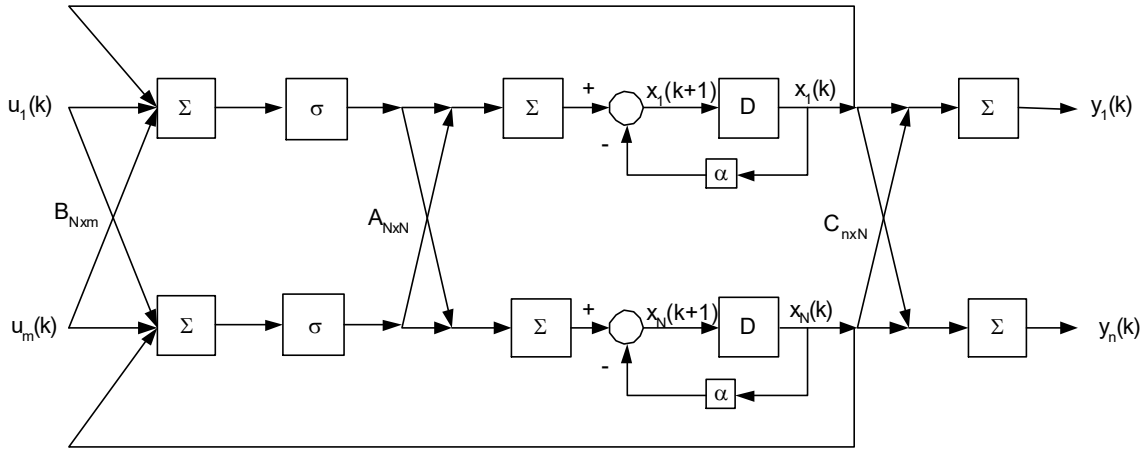
където външните въздействия са $u_1(t)$ и $u_2(t)$, променливите на състоянието са $x_1(t)$ и $x_2(t)$, а изходите на мрежата са $y_1(t)$ и $y_2(t)$.

Параметърът α принадлежи на интервала [0,1]. Невроните имат сигмоидални активиращи функции. Тук вместо оператора закъснение, характерен за дискретните рекурентни мрежи, се използва интегрален оператор.

1.9.2. Теоретични резултати при моделирането на динамични системи с рекурентни невронни мрежи

А.Моделиране с дискретни рекурентни невронни мрежи.

В [6] е изследвана апроксимацията на дискретни динамични системи с използването на дискретни рекурентни невронни мрежа. Разглежда се невронна мрежа, представена на фиг. 1.55. Доказано е, че дискретните динамични траектории в затворен интервал могат да бъдат апроксимирани със зададена точност чрез дискретни рекурентни невронни мрежи.



Фиг. 1.55

Общият запис на дискретната рекурентна невронна мрежи с два слоя (N неврона в първи и n неврона във втори слой) се дава със следната дискретна нелинейна система:

$$\begin{aligned}\mathbf{x}(k+1) &= -\alpha\mathbf{x}(k) + \mathbf{A}\boldsymbol{\sigma}(\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k)) \\ \mathbf{y}(k) &= \mathbf{C}\mathbf{x}(k),\end{aligned}, \text{ за } k = 0,1,2,...$$

където $\mathbf{x} \in R^N$, $\mathbf{y} \in R^n$ и $\mathbf{u} \in R^m$ са съответно вектор на състоянията, вектор на изхода на мрежата и входен вектор на мрежата, а $\mathbf{A} \in R^{N\times N}$, $\mathbf{B} \in R^{N\times m}$ и $\mathbf{C} \in R^{n\times N}$ са матрици на теглата на връзките и α е фиксирана константа, като $\alpha \in [-1,1]$. Тук $\boldsymbol{\sigma}(\mathbf{s})$ е векторната функция $\boldsymbol{\sigma}(\mathbf{s}) = (\sigma(s_1),\sigma(s_2),...,\sigma(s_N))^T$ и $\sigma(.)$ е гладка активираща функция.

Теорема 1. Нека $S \subset R^n$ и $U \subset R^m$ са отворени множества, $D_S \subset S$ и $D_U \subset U$ са компактни подмножества, $\mathbf{f}: S \times U \to R^n$ е непрекъсната вектор-функция. Тогава за дискретната нелинейна динамична система

$$\mathbf{z}(k+1) = \mathbf{f}(\mathbf{z}(k),\mathbf{u}(k)), \quad \mathbf{z} \in R^n, \quad \mathbf{u} \in R^m \text{ ,}$$

с начално условие $z(0)$ и решение $\mathbf{z} \in D_S$ съществува невронна мрежа от вида

$$\mathbf{x}(k+1) = -\alpha \mathbf{x}(k) + \mathbf{A}\boldsymbol{\sigma}(\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k))$$
$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k)$$

с начално условие $\mathbf{x}(0)$ такова, че за всяко ограничено входно въздействие $\mathbf{u}(k) \in D_U$ и $\varepsilon > 0$ е изпълнено

$$\max_{0 \le k \le n_f} \|\mathbf{z}(k) - \mathbf{y}(k)\| < \varepsilon .$$

Б. Моделиране с непрекъснати невронни мрежи

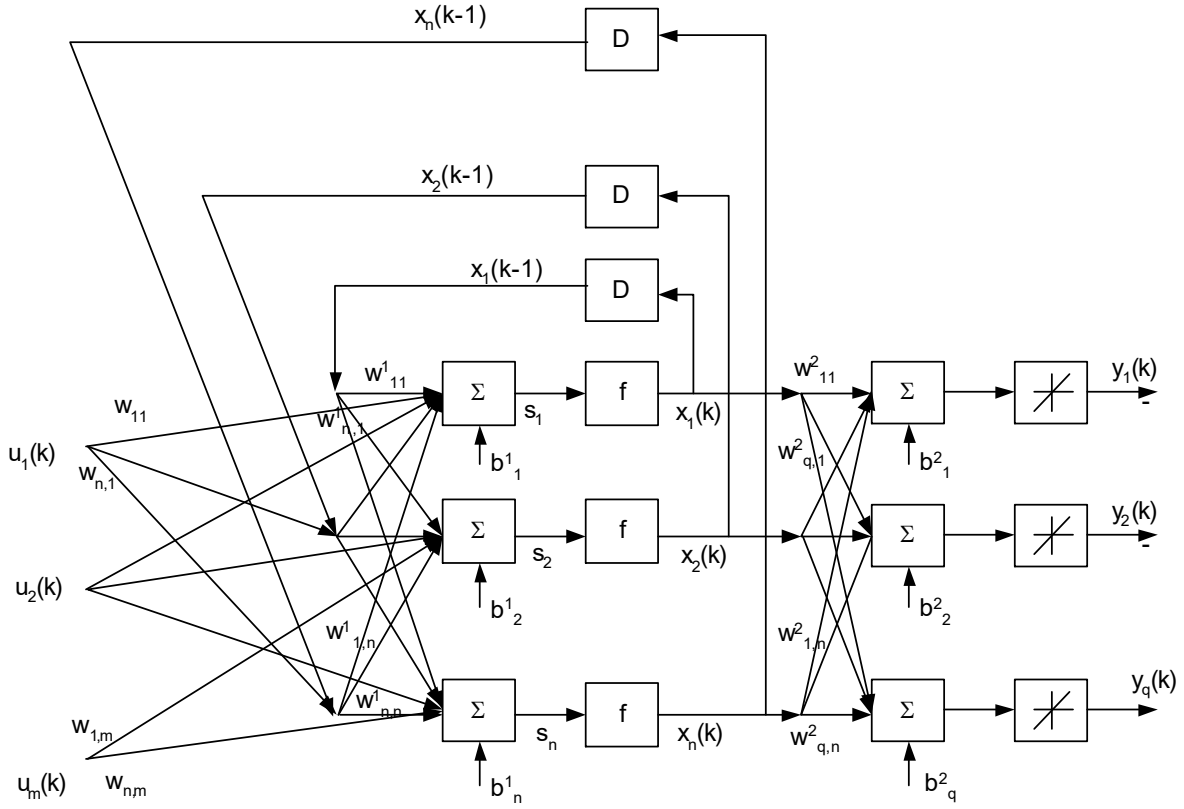
Този тип невронни мрежи се реализират с непрекъснати елементи и притежават обратни връзки. Те се описват със системи ОДУ. Изследване относно апроксимацията на непрекъснати динамични системи с непрекъснати динамични невронни мрежи е направено примерно в [11] и са доказани аналогични теореми на тези за дискретни мрежи.

В. Устойчивост

Многослойните рекурентни невронни мрежи с нелинейни активиращи функции представляват нелинейни динамични системи. Изследването на тяхната устойчивост е сложен проблем. При моделирането на нелинейни обекти може да бъде полезно следното условие за устойчивост: при ограничено входно въздействие изходът на модела да бъде ограничен. Съгласно дефиницията в този случай една нелинейна динамична система е устойчива , ако съществуват константи C_u и C_x така, че при $\|u(t)\|_L < C_u$ е изпълнено $\|x(t)\|_L < C_x$, където $u(t)$ е входното въздействие, $x(t)$ е състоянието на системата , а L е подходящо избрана норма. При рекурентните дискретни невронни мрежи условието за устойчивост в този смисъл е свързано с наличието на невронен слой с ограничени активиращи функции (например $th(\bullet)$). Последното може да гарантира ограниченост на изходните величини на невронната мрежа при ограничени входни въздействия.

1.10.Невронни мрежи на Елман

Архитектура на мрежа на Елман е показана на фиг.1.56.



фиг.1.56

Мрежите на Елман са двуслойни с обратни връзки от изхода на първия слой към входа на мрежата, при това в тези обратни връзки има закъснително звено D. Активиращите функции на първия (скрития) слой са сигмоидални, а на втория (изходния) слой са линейни.

Мрежата се описва от системата

$$\mathbf{x}(k) = \mathbf{f}(\mathbf{W}\,\mathbf{u}(k) + \mathbf{W}_1\,\mathbf{x}(k-1) + \mathbf{b}^1)$$

$$\mathbf{y}(k) = \mathbf{W}_2\,\mathbf{x}(k) + \mathbf{b}^2,$$

където $\mathbf{f}(\mathbf{s}) = (f(s_1), f(s_2), ..., f(s_n))^T$ е векторна функция;

$\mathbf{W}$ е матрицата от теглови вектори на първия слой, $\mathbf{W} \in R^{n \times m}$;

$\mathbf{W_1}$ - матрицата от теглови вектори, свързващи изхода на първия слой към входа на мрежата, $\mathbf{W_1} \in R^{n \times n}$;

$\mathbf{W_2}$ - матрицата от теглови вектори на втория слой, $\mathbf{W_2} \in R^{q \times n}$;

$\mathbf{u}$ - вектор от входовете на невронната мрежа, $\mathbf{u} \in R^m$;

$\mathbf{b}^1$ - вектор от свободни тегла на първия слой, $\mathbf{b}^1 \in R^n$;

$\mathbf{b}^2$ - вектор от свободни тегла на втория слой, $\mathbf{b}^2 \in R^q$;

$\mathbf{x}$ - вектор от изходите на първия слой, $\mathbf{x} \in R^n$;

$\mathbf{y}$ - вектор от изходите на втория слой, $\mathbf{y} \in R^q$;

Мрежата на Елман е рекурентна, затова тя еднакво добре може да се използва за моделиране на динамични и статични обекти.

1.10.1. Обучение на мрежата на Елман

Тази мрежа се обучава чрез процедурата обратно разпространение на грешките, като градиентът се изчислява с пренебрегване на рекурсията, т.е. обратните връзки се разкъсват и мрежата се разглежда като невронна мрежа без рекурсия. При това се препоръчва да се използва процедурата `traingdx`, т.е. градиентна процедура с тегло. Тя се реализира по формулата

$$\mathbf{w}^{k+1} = mc\ \mathbf{w}^k + lr\ \mathrm{grad}(S(\mathbf{w}^k)),$$

където:

$S(\mathbf{w}^k)$ е грешката на мрежата;

$\mathbf{w}$ - векторът от всички тегла на мрежата;

lr - константа, наречена скорост на обучение;

mc - така наречената масова константа.

Обучението на мрежата на Елман може да се извърши, като на входа й се подаде цялата входна информация, т.е в пакетен режим или тази информация се подава последователно на всяка стъпка (адаптивно).

При първия подход (процедурата `train`) стъпките са следните:

1.Цялата редица от входни сигнали се подава на входа на мрежата и се изчислява изходът на мрежата и средноквадратичната грешка спрямо теглата.

2. За всички времеви отчети се изчислява градиентът на грешката по отношение на теглата, като се пренебрегва обратната връзка.

3. Чрез този градиент се определя следващата итерация на теглата.

При втория подход (процедурата `adapt`) стъпките са следните.

1. За всеки отделен времеви отчет се подава входен вектор и се намира изходната грешка.

2. Изчислява се градиентът на грешката по отношение на теглата, като се пренебрегват обратните връзки.

3. Този приближен градиент се използва, за да се намери следващата итерация на теглата.

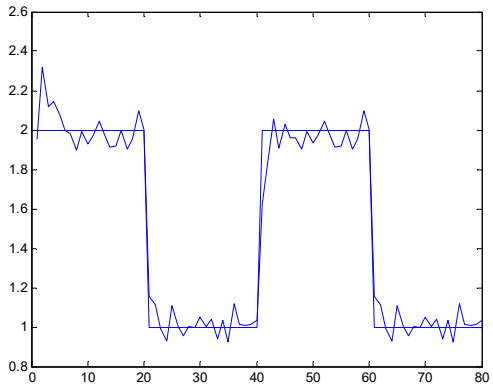
Пример 1. Детекция на амплитуди с мрежа на Елман.

Целта е да се обучи една мрежа на Елман да разпознава амплитудите на два времеви сигнала $u_1(t_k)=\cos(t_k)$, за $t_k=1,2,...,20$ и $u_2(t_k)=2\cos(t_k)$, за $t_k=20,21,...,40$. Тези два сигнала се подават един след друг на входа на една мрежа на Елман и се иска на изхода на мрежата да се получат техните амплитуди.

Програмата, с която се реализира тази задача, е следната:

```
clear all
%Масиви от обучаващи данни;
p1=2*cos(1:20);
p2=1*cos(1:20);
t1=2*ones(1,20);
t2=1*ones(1,20);
p=[p1 p2 p1 p2];
t=[t1 t2 t1 t2];
Pseq=con2seq(p);
Tseq=con2seq(t);
%Структора на мрежата
s2=1;
s1=10;
net=newelm([-2 2],[s1 s2],{'tansig','purelin'},'traingdx');
%Обучение и симулация на мрежата
for k=1:10
   net=train(net, Pseq, Tseq);
   out=sim(net,Pseq);
   q=seq2con(out);
   q1=q{1,1}
   plot(q1),hold on,plot(t),hold off,figure(gcf),pause
end;
```

На фиг.1.57 са дадени еталонът и изходът, реализиран от мрежата.



Фиг.1.57

Пример 2. Изследва се отново динамичната системата от фиг.1.45, която е подложена на ударно входно въздействие от вида

(44) $$u(t) = 21.2t^{1.5}e^{-4t}\sin(e^{1.89t} - 1).$$

Еталонът се задава със следната система ОДУ:

(45)
$$\dot{z}_1 = z_2$$
$$\dot{z}_2 = -\frac{c}{m}z_2 - \frac{k}{m}z_1 + \frac{1}{m}(k\,u(t) + c\dot{u}(t),$$

където:

m – маса;

c – коефициент на демпфиране;

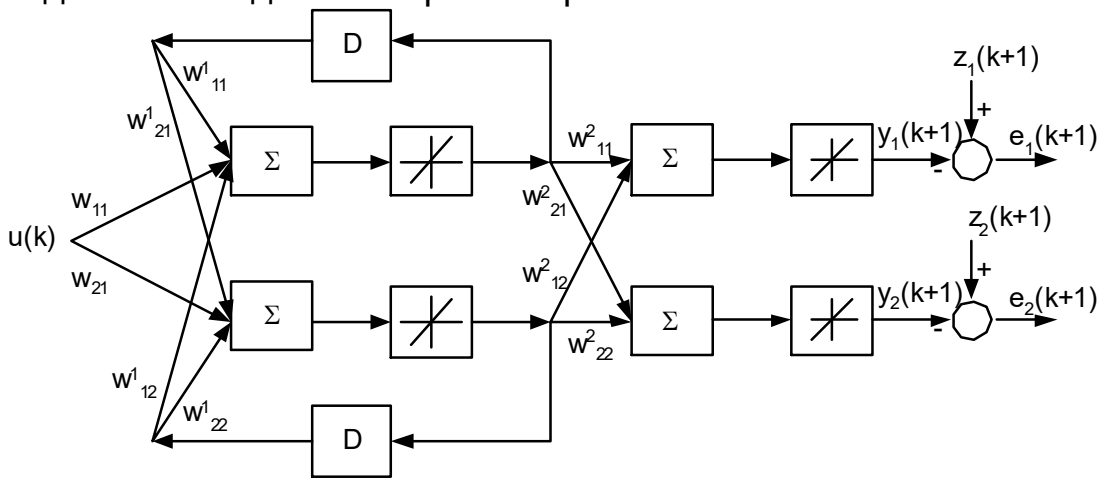
k – коефициент на еластичност;

$z_1(t)$ - преместване на масата;

$z_2(t) = \dot{z}_1(t)$ - скорост на масата.

Тази система се решава при нулеви начални условия $z_2(0) = z_1(0) = 0$.

Моделът е следната невронна мрежа:



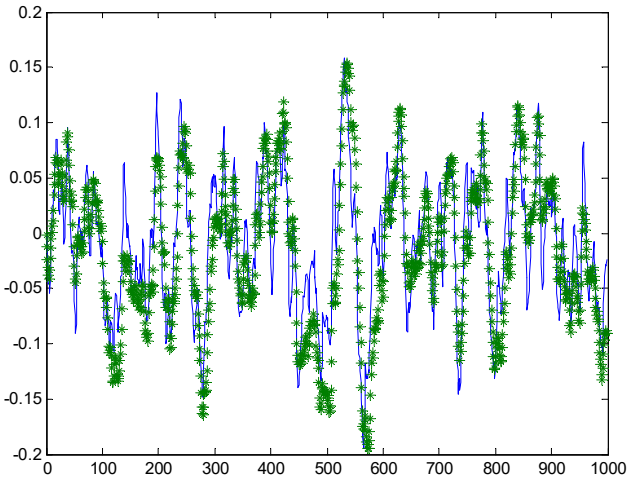
фиг.1.58

Програмата, с която се реализира поставената задача, има четири обособени части. В първата се решава системата ОДУ, в която външното въздействие $u(t)$ е бял гаусов шум. Във втората част се създава структурата на мрежата, отговаряща на фиг.1.58 и се обучава мрежата, като се използват вече генерираните обучаващи данни. Основната разлика в сравнение с правата невронна мрежа-предиктор (разработена в предния параграф) е, че сега на входа се подава само външното въздействие $u(t)$ и цялата динамика се моделира с рекурентната мрежа. След завършване на обучението се сравняват изходът (реализиран от мрежата) и еталонният процес (фиг.1.59). В

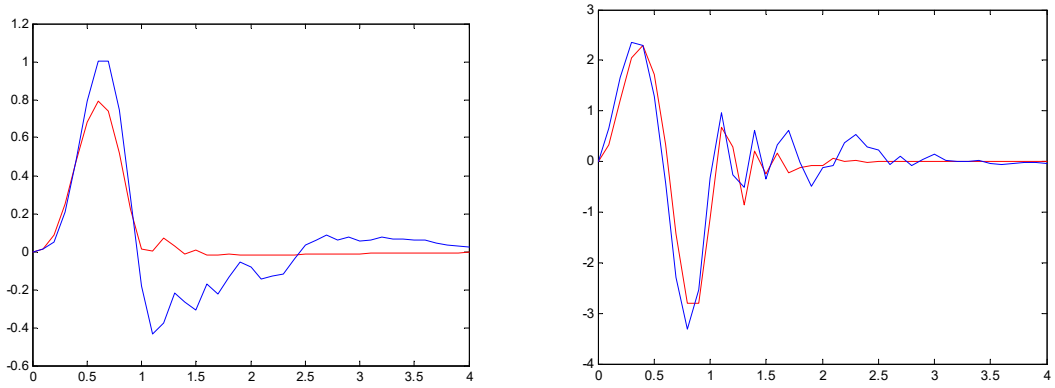
третата част на програмата се решава системата ОДУ при ударно входно въздействие $u(t)$, пресметнато по формула (44). В последната част се симулира с данните, получени при ударно въздействие. Резултатите, получени от изхода на мрежата $\{y_1(k)\}$ и $\{y_2(k)\}$, се сравняват с еталонните резултати $\{z_1(k)\}$ и $\{z_2(k)\}$, получени след решение на системата (45) (фиг.1.60).

```
clear all
global c k m u1

% Генериране на обучаващи данни при гаусово входно въздействие
u(t)
c=40;k=25;m=10;
n=1000;
y0=[0;0];timestep=0.05;t0=0;tk=2*timestep;
t(1)=0;y(1,1)=0;y(2,1)=0;
for i=1:n
   u1=normrnd(0,1);
   u(i)=u1;
   [t1,y1]=ode15s('fun1',[t0:timestep:tk],y0);
      t(i+1)=tk;
   y0=y1(3,:);
   t0=tk;tk=tk+2*timestep;
   y(1,i+1)=y1(3,1);
   y(2,i+1)=y1(3,2);
end;
Pm=[u(1:n-1)],
Tm=[y(1,2:n);y(2,2:n)],

% Обучение на мрежата с така генерираните данни
S1=3;
[S2,Q] = size(Tm);
Pm,Tm,
netse = newelm(minmax(Pm),[S1 S2],{'purelin'
'purelin'},'traingdx');
netse.b{1}=[0;0;0];
netse.biases{1}.learn=0;
netse.b{2}=[0;0];
netse.biases{2}.learn=0;
netse.trainParam.epochs = 300;       % Maximum number of epochs
to train.
netse.trainParam.goal =0.00001;     % Mean-squared error goal.
netse.trainParam.lr =0.01;          % Learning rate.
netse.trainParam.mc =0.85;          % Mass constant.
Pseq=con2seq(Pm);
```

```
Tseq=con2seq(Tm);
netse = train(netse,Pseq,Tseq);
Y=sim(netse,Pseq);
y2=seq2con(Y);
y1=y2{1};
t=0:1:length(y1)-1;
plot(t,y1(1,:),t,Tm(1,:),'*'),figure(gcf),pause

% Генериране на обучаващи данни при ударно входно въздействие
u(t)
c=40;k=25;m=10;
n=40;
t0=0;tk=n*2*timestep;y0=[0;0];
[t1,y1]=ode15s('fun2',[t0:2*timestep:tk],y0);

% Симулиране на мрежата като невронен модел с последните
генерирани данни
time = 0:2*timestep:n*2*timestep;
states = zeros(2,length(time));
u0=udar(time);
uu0=con2seq(u0);
states=sim(netse,uu0);
states1=seq2con(states);
states2=states1{1};
figure(1)
plot(t1,y1(:,1),'r',time,states2(1,:))
figure(2)
plot(t1,y1(:,2),'r',time,states2(2,:))
```



фиг.1.59



фиг.1.60

Приведената главна програма използва следните три помощни програми: програмата udar, в която е реализирано ударното входно въздействие, програмите fun1 и fun2, в които се описват десните части на системата (45) съответно със случайно входно въздействие и с ударно входно въздействие. Тези програми са дадени в пример 1, раздел 1.8.

1.11. Алгоритъм на множителите на Лагранж за обучение на рекурентна невронна мрежа

1.11.1. Алгоритъм на множителите на Лагранж за параметрична оптимизация на дискретна динамична система

Нека дискретната динамична система е зададена чрез рекурентната система:

(46) $\mathbf{y}(k+1)=\mathbf{f}(\mathbf{y}(k),\mathbf{u}(k),\mathbf{p}(k))$, за $\mathbf{y}(0)=\mathbf{y}_0; k=0,1,...,N-1$

(47) $\mathbf{p}(k+1)=\mathbf{p}(k)$,

където

$\mathbf{y},\mathbf{f}\in R^2$, $\mathbf{p}\in R^2$, $\mathbf{y}(k)=(y_1(k),y_2(k))^T$, $\mathbf{f}(k)=(f_1(k),f_2(k))^T$ и $\mathbf{p}(k)=(p_1(k),p_2(k))^T$.

Условието (47) дефинира вектора на параметрите $\mathbf{p}(k)$ на системата (46) като постоянен.

Поставя се условната оптимизационна задача да се минимизира целевата функцията

(48) $$J(\mathbf{p})=\sum_{k=0}^{N-1}\frac{1}{2}(\mathbf{y}(k)-\mathbf{z}(k))^T(\mathbf{y}(k)-\mathbf{z}(k)),$$

спрямо вектора на параметрите $\mathbf{p}(k)$ при ограниченията (46-47). Тук $\mathbf{z}(k)=(z_1(k),z_2(k))^T$, като $z_1(k)$ и $z_2(k)$ са зададени (еталонни) траектории. Целта е да се определи векторът на параметрите $\mathbf{p}(k)$, така че решенията (траекториите) $y_1(k)$ и $y_2(k)$ на (46) да са средноквадратично приближение на еталонните.

Условната оптимизационна задача (46)-(47)-(48) се свежда до еквивалентна оптимизационна задача без ограничения, при която се минимизира лагранжианът:

(49)

$$L(\mathbf{p})=\sum_{k=0}^{N-1}\frac{1}{2}\big(\mathbf{y}(k)-\mathbf{z}(k)\big)^T(\mathbf{y}(k)-\mathbf{z}(k))+\sum_{k=0}^{N-1}\{\boldsymbol{\lambda}^T(k+1)[\mathbf{f}(\mathbf{y}(k),\mathbf{u}(k),\mathbf{p}(k))-\mathbf{y}(k+1)]+$$

$$+\sum_{k=0}^{N-1}\{\boldsymbol{\Gamma}^T(k+1)[\mathbf{p}(k)-\mathbf{p}(k+1)]\},$$

където множителите на Лагранж са

(50) $\boldsymbol{\lambda}(k)=(\lambda_1(k),\lambda_2(k))^T$ и $\boldsymbol{\Gamma}(k)=(\Gamma_1(k),\Gamma_2(k))^T$.

За минимизиране на функцията $L(\mathbf{p})$ е необходимо да се определи градиентът й спрямо параметрите, след което тази функция се минимизира с подходяща градиентна процедура. Следва алгоритъмът за определяне на този градиент.

Твърдение 1. Диференциалът на лгаранжиана (49) може да се представи във вида

(51)
$$dL(\mathbf{p})=\boldsymbol{\lambda}^T(0)\Delta\mathbf{y}(0)-\boldsymbol{\lambda}^T(N)\Delta\mathbf{y}(N)+\boldsymbol{\Gamma}(0)\Delta\mathbf{p}(0)-\boldsymbol{\Gamma}(N)\Delta\mathbf{p}(N)+$$
$$+\sum_{k=0}^{N-1}\left\{\left(\frac{\partial H(k)}{\partial\mathbf{y}(k)}-\boldsymbol{\lambda}(k)\right)^T\Delta\mathbf{y}(k)+\left(\frac{\partial H(k)}{\partial\mathbf{p}(k)}-\boldsymbol{\Gamma}(k)\right)^T\Delta\mathbf{p}(k)\right\},$$

където $H(k)$ е функцията на Хамилтън:

(52)

$$H(k)=\frac{1}{2}\big(\mathbf{y}(k)-\mathbf{z}(k)\big)^T(\mathbf{y}(k)-\mathbf{z}(k))+\boldsymbol{\lambda}^T(k+1)\mathbf{f}(\mathbf{x}(k),\mathbf{u}(k),\mathbf{p}(k))+\boldsymbol{\Gamma}^T(k+1)\mathbf{p}(k),$$

а $\dfrac{\partial H(k)}{\partial\mathbf{y}(k)}=\left(\dfrac{\partial H(k)}{\partial y_1(k)}\quad\dfrac{\partial H(k)}{\partial y_2(k)}\right)^T$ и $\dfrac{\partial H(k)}{\partial\mathbf{p}(k)}=\left(\dfrac{\partial H(k)}{\partial p_1(k)}\quad\dfrac{\partial H(k)}{\partial p_2(k)}\right)^T$.

За доказване на твърдението се представя диференциалът на (49), както следва:

(53) $$dL(\mathbf{p}) = \sum_{k=0}^{N-1} \left(\mathbf{y}(k) - \mathbf{z}(k)\right)^T \Delta\mathbf{y}(k) + \sum_{k=0}^{N-1} \boldsymbol{\lambda}^T(k+1)\left[\frac{\partial \mathbf{f}(k)}{\partial \mathbf{y}(k)}\Delta\mathbf{y}(k) - \Delta\mathbf{y}(k+1)\right] +$$

$$+ \sum_{k=k_0}^{k_f-1} \boldsymbol{\lambda}^T(k+1)\frac{\partial \mathbf{f}(k)}{\partial \mathbf{p}(k)}\Delta\mathbf{p}(k) + \sum_{k=k_0}^{k_f-1} \boldsymbol{\Gamma}^T(k+1)\left[\Delta\mathbf{p}(k) - \Delta\mathbf{p}(k+1)\right],$$

където

$$\Delta\mathbf{y}(k) = (\Delta y_1(k), \Delta y_2(k))^T, \ \Delta\mathbf{p}(k) = (\Delta p_1(k), \Delta p_2(k))^T$$

$$\frac{\partial \mathbf{f}(k)}{\partial \mathbf{y}(k)} = \begin{pmatrix} \frac{\partial f_1(k)}{\partial y_1(k)} & \frac{\partial f_1(k)}{\partial y_2(k)} \\ \frac{\partial f_2(k)}{\partial y_1(k)} & \frac{\partial f_2(k)}{\partial y_2(k)} \end{pmatrix} \text{ и } \frac{\partial \mathbf{f}(k)}{\partial \mathbf{p}(k)} = \begin{pmatrix} \frac{\partial f_1(k)}{\partial p_1(k)} & \frac{\partial f_1(k)}{\partial p_2(k)} \\ \frac{\partial f_2(k)}{\partial p_1(k)} & \frac{\partial f_2(k)}{\partial p_2(k)} \end{pmatrix}$$

и се определят производните на функцията на Хамилтън (52):

(54) $$\frac{\partial H(k)}{\partial \mathbf{y}(k)} = [y(k) - z(k)]^T + \lambda^T(k+1)\frac{\partial \mathbf{f}(k)}{\partial \mathbf{y}(k)}$$

(55) $$\frac{\partial H(k)}{\partial \mathbf{p}(k)} = \lambda^T(k+1)\frac{\partial \mathbf{f}(k)}{\partial \mathbf{y}(k)} + \boldsymbol{\Gamma}^T(k+1).$$

Тогава в сумата на първите два члена от (53) се прибавя и изважда $\boldsymbol{\lambda}^T(0)\Delta\mathbf{y}(0)$ и като се използва (54), тя се преработва така:

$$\sum_{k=0}^{N-1} \left(\mathbf{y}(k) - \mathbf{z}(k)\right)^T \Delta\mathbf{y}(k) + \sum_{k=0}^{N-1} \{\boldsymbol{\lambda}^T(k+1)\left[\frac{\partial \mathbf{f}(k)}{\partial \mathbf{x}(k)}\Delta\mathbf{y}(k)\right] - \Delta\mathbf{y}(k+1)\} =$$

$$= \sum_{k=0}^{N-1} \left(\mathbf{y}(k) - \mathbf{z}(k)\right)^T \Delta\mathbf{y}(k) + \sum_{k=0}^{N-1} \{\boldsymbol{\lambda}^T(k+1)\left[\frac{\partial \mathbf{f}(k)}{\partial \mathbf{x}(k)}\Delta\mathbf{y}(k)\right] - \Delta\mathbf{y}(k+1)\} -$$

$$- \boldsymbol{\lambda}^T(0)\Delta\mathbf{y}(0) + \boldsymbol{\lambda}^T(0)\Delta\mathbf{y}(0) =$$

$$= \sum_{k=0}^{N-1} \left(\mathbf{y}(k) - \mathbf{z}(k)\right)^T \Delta\mathbf{y}(k) - \boldsymbol{\lambda}^T(N)\Delta\mathbf{y}(N) - \sum_{k=0}^{N-1} \boldsymbol{\lambda}^T(k)\Delta\mathbf{y}(k) + \boldsymbol{\lambda}^T(0)\Delta\mathbf{y}(0) +$$

$$+ \sum_{k=0}^{N-1} \boldsymbol{\lambda}^T(k+1)\frac{\partial \mathbf{f}(k)}{\partial \mathbf{y}(k)}\Delta\mathbf{y}(k) =$$

$$= \boldsymbol{\lambda}^T(0)\Delta\mathbf{y}(0) - \boldsymbol{\lambda}^T(N)\Delta\mathbf{y}(N) +$$

$$+ \sum_{k=0}^{N-1} \{\left(\mathbf{y}(k) - \mathbf{z}(k)\right)^T + \boldsymbol{\lambda}^T(k+1)\frac{\partial \mathbf{f}(k)}{\partial \mathbf{y}(k)} - \boldsymbol{\lambda}^T(k)\}\Delta\mathbf{y}(k) =$$

$$= \boldsymbol{\lambda}^T(0)\Delta\mathbf{y}(0) - \boldsymbol{\lambda}^T(N)\Delta\mathbf{y}(N) + \sum_{k=0}^{N-1} \{\frac{\partial H(k)}{\partial \mathbf{y}(k)} - \boldsymbol{\lambda}(k)\}^T \Delta\mathbf{y}(k).$$

Аналогично се преобразува сумата на вторите два члена от (52), като се прибавя и изважда $\boldsymbol{\Gamma}(0)\Delta\mathbf{p}(0)$ и се използва (55), т.е.

$$\sum_{k=0}^{N-1}\boldsymbol{\lambda}^T(k+1)\frac{\partial \mathbf{f}(k)}{\partial \mathbf{p}(k)}\Delta\mathbf{p}(k)+\sum_{k=0}^{N-1}\boldsymbol{\Gamma}^T(k+1)[\Delta\mathbf{p}(k)-\Delta\mathbf{p}(k+1)]-$$
$$-\boldsymbol{\Gamma}(0)\Delta\mathbf{p}(0)+\boldsymbol{\Gamma}(0)\Delta\mathbf{p}(0)=$$
$$=\boldsymbol{\Gamma}^T(0)\Delta\mathbf{p}(0)-\boldsymbol{\Gamma}^T(N)\Delta\mathbf{p}(N)+$$
$$+\sum_{k=0}^{N-1}\{\ \boldsymbol{\lambda}^T(k+1)\frac{\partial \mathbf{f}(k)}{\partial \mathbf{p}(k)}+\boldsymbol{\Gamma}^T(k+1)-\boldsymbol{\Gamma}^T(k)\}\Delta\mathbf{p}(k)\}=$$
$$=\boldsymbol{\Gamma}^T(0)\Delta\mathbf{p}(0)-\boldsymbol{\Gamma}^T(N)\Delta\mathbf{p}(N)+\sum_{k=0}^{N-1}\{\frac{\partial H(k)}{\partial \mathbf{p}(k)}-\boldsymbol{\Gamma}(k)\}^T\Delta\mathbf{p}(k).$$

Следователно диференциалът от (53) се привежда във вида (51) и твърдението е доказано. Изразът на този диференциал може да се опрости чрез полагането:

(56) $$\boldsymbol{\lambda}(k)=\frac{\partial H(k)}{\partial \mathbf{y}(k)},\quad k=N-1,...,0$$

(57) $$\boldsymbol{\Gamma}(k)=\frac{\partial H(k)}{\partial \mathbf{p}(k)},\quad k=N-1,...,0.$$

Уравненията (56),(57) образуват спрегнатата система за задачата (46)-(47)-(48).

Тъй като граничните условия за множителите на Лагранж могат да са произволни, те се приемат нулеви $\boldsymbol{\Gamma}(N)=0,\ \boldsymbol{\lambda}(N)=0$ и тогава диференциалът се получава, както следва:

$$dL=\boldsymbol{\Gamma}(0)^T\Delta\mathbf{p}(0)+\boldsymbol{\lambda}(0)^T\Delta\mathbf{y}(0)$$

Освен това, ако началното условие на (46) е зададено, то $\Delta\mathbf{y}(0)=0$ и окончателно диференциалът се записва така:

$$dL(\mathbf{p})=\boldsymbol{\Gamma}(0)^T\Delta\mathbf{p}(0).$$

Тогава градиентът на лагранжиана спрямо параметрите се получава от следната формула:

(58) $$grad\ L(\mathbf{p})=\boldsymbol{\Gamma}(0).$$

Окончателно алгоритъмът за пресмятане градиента на функцията (48) е следният:

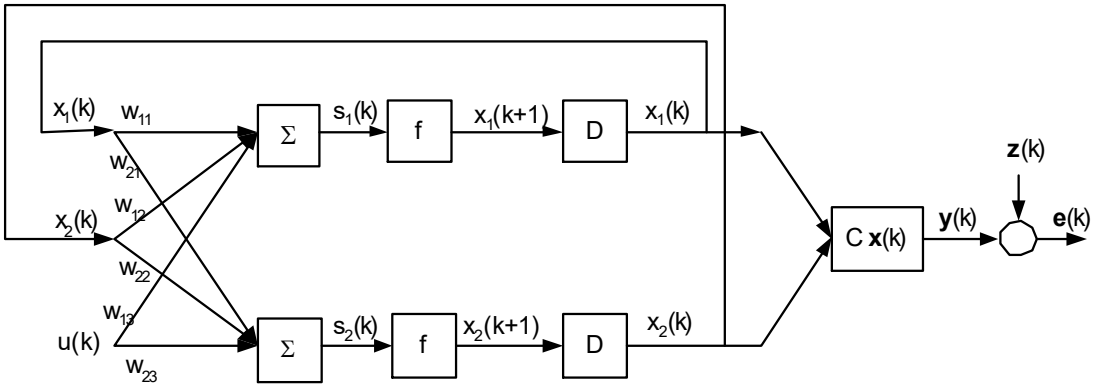
Стъпка 1. Пресмята се $\{\mathbf{y}(k)\}$ от (46) за $\mathbf{y}(0)=\mathbf{y}_0;\ k=0,1,...,\ N-1$.

Стъпка 2. Решава се спрегнатата система (56) и (57) за $k=N-1,...,0$.

Стъпка 3. Градиентът се определя от (58) $grad\ L(\mathbf{p})=\ \boldsymbol{\Gamma}(0)$.

1.11.2. Алгоритъм на множителите на Лагранж за двуслойна рекурентна невронна мрежа

Тук се разглежда рекурентната невронна двуслойна мрежа, показана на фиг.1.61. Целта е да се приложи алгоритъмът на множителите на Лагранж от предния раздел за определяне градиента на средноквадратичната грешка на мрежата. След определяне на градиента невронната мрежа се обучава със стандартна градиентна процедура.



фиг. 1.61

Тази рекурентна невронна мрежа се описва с рекурентната система от уравнения, както следва:

Описание на първи слой на мрежата

$$x_1(k+1) = f(w_{11}x_1(k) + w_{12}x_2(k) + w_{13}u(k))$$
(59)
$$x_2(k+1) = f(w_{21}x_1(k) + w_{22}x_2(k) + w_{23}u(k))\,,$$

(60) $$x_1(0) = 0; \quad x_2(0) = 0\,.$$

Описание на втори слой на мрежата

(61) $$\begin{pmatrix} y_1(k) \\ y_1(k) \end{pmatrix} = \begin{pmatrix} c_1 & 0 \\ 0 & c_2 \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix} \qquad k = 0,1,2,...,N-1\,.$$

Записва се средноквадратичната грешка на мрежата

(62) $$J(\mathbf{p}) = \frac{1}{2}\sum_{k=1}^{N-1}\left[(y_1(k) - z_1(k))^2 + (y_2(k) - z_2(k))^2\right],$$

където $\mathbf{p} = (w_{11}, w_{12}, ..., w_{23}, c_1, c_2) = (p_1, ..., p_8)$ е векторът от теглата на мрежата, които се определят чрез минимизиране на функцията (62).

Въвеждат се следните вектори и матрици:

$\mathbf{y}(k) = (y_1(k)\ y_2(k))^T$ -изход, $\mathbf{x}(k) = (x_1(k) \quad x_2(k))^T$ -вход,

$\boldsymbol{\lambda} = (\lambda_1(k) \quad \boldsymbol{\lambda}_2(k))$ и $\boldsymbol{\Gamma} = (\Gamma_{11}(k)\Gamma_{12}(k)...\Gamma_{23}(k)\Gamma_{c1}(k)\ \Gamma_{c2}(k))^T$ - множители на Лагранж,

$\mathbf{e}(k) = (y_1(k) - z_1(k) \quad y_2(k) - z_2(k))^T$ - вектор на грешките

$\mathbf{W}_x = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}$ - тегла на първия слой спрямо $\mathbf{x}(k)$.

Използва се алгоритъмът описан в предходния раздел. За целта се въвежда хамилтонианът:

$$H(\mathbf{x}(k), \mathbf{p}(k)) = \frac{1}{2}(y_1(k) - z_1(k))^2 + \frac{1}{2}(y_2(k) - z_2(k))^2 +$$

(63) $$+ (\boldsymbol{\lambda}_1(k+1)\ \boldsymbol{\lambda}_2(k+1))(f(s_1(k))\ f(s_2(k)))^T +$$

$$+ (\Gamma_1(k+1)\ \Gamma_2(k+1)...\Gamma_8(k+1))(p_1\ p_2\ ...p_8)^T.$$

Чрез този хамилтониан се образува спрегнатата система със съответните гранични условия:

(64) $\boldsymbol{\lambda}(k) = \dfrac{\partial H(k)}{\partial \mathbf{x}}$ $\boldsymbol{\lambda}(N) = 0$, за $k = N-1,...,2,1,0$

(65) $\boldsymbol{\Gamma}(k) = \dfrac{\partial H}{\partial \mathbf{p}}$ $\boldsymbol{\Gamma}(N) = 0$, за $k = N-1,...,2,1,0$.

За задачата (64) и (65) уравненията на мрежата се получават:

$$\lambda_1(k) = \frac{\partial H(k)}{\partial x_1} = (y_1(k) - z_1(k)) + \lambda_1(k+1)w_{11}f(s_1(k))w_{12}f(s_2(k))$$

$$\lambda_2(k) = \frac{\partial H(k)}{\partial x_2} = (y_2(k) - z_2(k)) + \lambda_1(k+1)w_{12}f(s_1(k))w_{22}f(s_2(k))$$

(66)

$$\begin{pmatrix} \lambda_1(k) \\ \lambda_2(k) \end{pmatrix} = \begin{pmatrix} c_1 & 0 \\ 0 & c_2 \end{pmatrix}\begin{pmatrix} y_1(k) - z_1(k) \\ y_2(k) - z_2(k) \end{pmatrix} + \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{pmatrix}\begin{pmatrix} f'(s_1(k)) & 0 \\ 0 & f'(s_2(k)) \end{pmatrix}\begin{pmatrix} \lambda_1(k+1) \\ \lambda_2(k+1) \end{pmatrix}$$

Следва матричният запис на (66):

$$\boldsymbol{\lambda}(k) = \mathrm{e} + \mathbf{W}_x^T\ diag(f'(s_1(k))\ f'(s_2(k)))\ \boldsymbol{\lambda}(k+1).$$

$$\Gamma_{11}(k) = \frac{\partial H(k)}{\partial w_{11}} = \lambda_1(k+1)f'(s_1(k))x_1(k) + \Gamma_{11}(k+1)$$

$$\Gamma_{12}(k) = \frac{\partial H(k)}{\partial w_{12}} = \lambda_1(k+1)f'(s_1(k))x_2(k) + \Gamma_{12}(k+1)$$

$$\Gamma_{13}(k) = \frac{\partial H(k)}{\partial w_{13}} = \lambda_1(k+1)f'(s_1(k))x_3(k) + \Gamma_{13}(k+1)$$

$$\Gamma_{21}(k)=\frac{\partial H(k)}{\partial w_{21}}=\lambda_2(k+1)f'(s_2(k))x_1(k)+\Gamma_{21}(k+1)$$

$$\Gamma_{22}(k)=\frac{\partial H(k)}{\partial w_{22}}=\lambda_2(k+1)f'(s_2(k))x_2(k)+\Gamma_{22}(k+1)$$

$$\Gamma_{23}(k)=\frac{\partial H(k)}{\partial w_{23}}=\lambda_2(k+1)f'(s_2(k))x_3(k)+\Gamma_{23}(k+1),\ \text{за}\ \Gamma_{11}(N)=....=\Gamma_{23}(N)=0$$

(67)
$$\begin{pmatrix}\Gamma_{11}(k) & \Gamma_{12}(k) & \Gamma_{13}(k)\\ \Gamma_{21}(k) & \Gamma_{22}(k) & \Gamma_{23}(k)\end{pmatrix}=\begin{pmatrix}\Gamma_{11}(k+1) & \Gamma_{12}(k+1) & \Gamma_{13}(k+1)\\ \Gamma_{21}(k+1) & \Gamma_{22}(k+1) & \Gamma_{23}(k+1)\end{pmatrix}+$$
$$+\begin{pmatrix}f'(s_1(k)) & 0\\ 0 & f'(s_2(k))\end{pmatrix}\begin{pmatrix}\lambda_1(k+1)\\ \lambda_2(k+1)\end{pmatrix}\begin{pmatrix}x_1(k) & x_2(k) & u(k)\end{pmatrix},$$

Следва матричният запис на (66):

$$\mathbf{\Gamma}(k)=\mathbf{\Gamma}(k+1)+diag(f'(s_1(k))\ f'(s_2(k)))\boldsymbol{\lambda}(k+1)\bar{\mathbf{x}}^T(k),$$

където $\bar{\mathbf{x}}(k)=\begin{pmatrix}x_1(k) & x_2(k) & u(k)\end{pmatrix}^T$, $\boldsymbol{\lambda}(N)=0$, $\mathbf{\Gamma}(N)=0$, за $k=N-1,...,2,1,0$.

$$\Gamma_{c1}(k)=\frac{\partial H}{\partial c_1}=(y_1(k)-z_1(k))x_1(k)+\Gamma_{c1}$$

, за $\Gamma_{c1}(N)=\Gamma_{c2}(N)=0$.

$$\Gamma_{c2}(k)=\frac{\partial H}{\partial c_2}=(y_2(k)-z_2(k))x_2(k)+\Gamma_{c2}$$

(68)
$$\begin{pmatrix}\Gamma_{c1}(k) & \Gamma_{c2}(k)\end{pmatrix}=\begin{pmatrix}\Gamma_{c1}(k+1) & \Gamma_{c2}(k+1)\end{pmatrix}+\begin{pmatrix}x_1(k) & x_2(k)\end{pmatrix}\begin{pmatrix}y_1(k)-z_1(k) & 0\\ 0 & y_2(k)-z_2(k)\end{pmatrix}$$

Алгоритъмът се реализира в три стъпки:

Стъпка 1. Решава се системата (59) при съответното начално условие и се запаметяват траекториите $x_1(k),x_2(k)$ за $k=0,1,...,N-1$, като се изчислява и функцията (61).

Стъпка 2. Решава се спрегнатата система (66), (67) и (68), при съответните гранични условия за $k=N-1,...,2,1,0$.

Стъпка 3. Изчислява се градиентът на функцията (61) по формулата

$$grad\ J(\mathbf{p})=(\Gamma_{11}(0),\Gamma_{12}(0),...,\Gamma_{23}(0),\Gamma_{c1}(0),\Gamma_{c2}(0))^T.$$

Пример 1. Разглежда се следният дискретен филтър

(69) $$z(k)=-a_1y(k-1)-a_2z(k-2)+b_0u(k)+b_1u(k-1)+b_2u(k-2),$$

където $a_1=-1.4496$, $a_2=0.6455$, $b_0=0.1098$, $b_1=-0.0358$ и $b_2=0.1098$.

При нулеви начални условия

(70) $$z(-1) = z(0) = 0$$

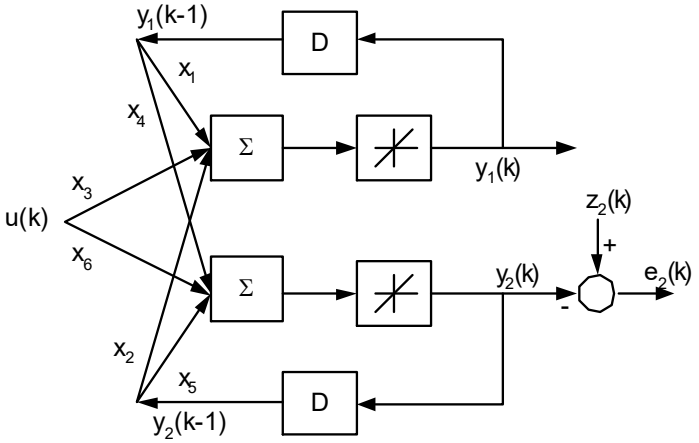
и единично външно въздействие

(71) $$u(1) = 1,\ u(2) = u(3) = ... = 0.$$

Това рекурентно уравнение описва нискочестотен филтър с импулсна характеристика, изобразена на фиг.1.63 с непрекъсната линия. Тази характеристика се получава при единично входно въздействие (71).

Тогава обучаващите данни за невронната мрежа ще бъдат единично входно въздействие (71) и еталонен изход, получен като решение на уравнението (69) при начални условия (71), т.е. еталонният изход е импулсната характеристика на филтъра.

Избира се модел на филтъра - линейна рекурентна еднослойна невронна мрежа с два неврона (фиг.1.62).



фиг.1.62

Мрежата се описва със следната рекурентна система:

$$y_1(k) = x_1 y_1(k-1) + x_2 y_2(k-1) + x_3 u(k)$$

$$y_2(k) = x_4 y_1(k-1) + x_5 y_2(k-1) + x_6 u(k).$$

Поставя се задача чрез алгоритъма на множителите на Лагранж да се минимизира средноквадратичната грешка

$$S = \frac{1}{2}\sum_{k=1}^{N}(y_2(k) - z_2(k))^2 = \frac{1}{2}\sum_{k=1}^{N} e_2(k)^2$$

спрямо теглата $x_1, x_2, ..., x_6$. За целта се образува хамилтонианът

$$H(k) = \frac{1}{2}(z_2(k) - y_2(k))^2 +$$

$$+ \begin{pmatrix}\lambda_1(k) & \lambda_2(k)\end{pmatrix}\begin{pmatrix} x_1 y_1(k-1) + x_2 y_2(k-1) + x_3 u(k-1) \\ x_4 y_1(k-1) + x_5 y_2(k-1) + x_6 u(k-1)\end{pmatrix} +$$

$$+\begin{pmatrix}\Gamma_1(k) & \Gamma_2(k) & \Gamma_3(k) & \Gamma_4(k) & \Gamma_5(k) & \Gamma_6(k)\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\\x_4\\x_5\\x_6\end{pmatrix}.$$

Образува се спрегнатата диференчна система:

$$\begin{aligned}\lambda_1(k-1)&=\frac{\partial H(k)}{\partial y_1(k)}=x_1\lambda_1(k)+x_2\lambda_2(k)\\ \lambda_2(k-1)&=\frac{\partial H(k)}{\partial y_2(k)}=(z_2(k)-y_2(k))+x_1\lambda_1(k)+x_2\lambda_2(k)\end{aligned}, \tag{72}$$

$$\lambda_1(N)=\lambda_2(N)=0 \tag{73}$$

$$\begin{aligned}\Gamma_1(k-1)&=\frac{\partial H(k)}{\partial x_1}=\lambda_1(k)z_1(k-1)+\Gamma_1(k)\\ \Gamma_2(k-1)&=\frac{\partial H(k)}{\partial x_2}=\lambda_1(k)z_2(k-1)+\Gamma_2(k)\\ \Gamma_3(k-1)&=\frac{\partial H(k)}{\partial x_3}=\lambda_1(k)u(k-1)+\Gamma_3(k)\\ \Gamma_4(k-1)&=\frac{\partial H(k)}{\partial x_4}=\lambda_2(k)y_1(k-1)+\Gamma_4(k)\\ \Gamma_5(k-1)&=\frac{\partial H(k)}{\partial x_5}=\lambda_2(k)y_2(k-1)+\Gamma_5(k)\\ \Gamma_6(k-1)&=\frac{\partial H(k)}{\partial x_6}=\lambda_2(k)u(k-1)+\Gamma_6(k)\end{aligned}, \tag{74}$$

$$\Gamma_1(N)=...=\Gamma_6(N)=0 \tag{75}$$

Системата (72)-(75) се решава последователно за $k=N,N-1,...,2,1$ при нулеви гранични условия. Тогава градиентът на грешката спрямо теглата x_i се получава, както следва:

$$grad\,S(x_1,...,x_6)=(\frac{\partial S}{\partial x_1},\frac{\partial S}{\partial x_2},\frac{\partial S}{\partial x_3},\frac{\partial S}{\partial x_4},\frac{\partial S}{\partial x_5},\frac{\partial S}{\partial x_6}).$$

$$grad\,S=\left(\Gamma_1(1),\Gamma_2(1),\Gamma_3(1),\Gamma_4(1),\Gamma_5(1),\Gamma_6(1)\right). \tag{76}$$

Задачата се реализира със следната програма.

```
clear all
global N ye y1 y2 u
%
N=128;
a1=-1.4496;a2=0.6455;b0=0.1098;b1=-0.0358;b2=0.1098;
y(1)=b0;y(2)=-a1*y(1)+b1;y(3)=-a1*y(2)-a2*y(1)+b2;
u=[1 zeros(1,N-1)];
for k=4:N
    y(k)=-a1*y(k-1)-a2*y(k-2);
end;
ye=y; y1=zeros(1,N); y2=[b0 zeros(1,N-1)]; u=[1 zeros(1,N-1)];
%
options(1)=1;
options(14)=1000;
options(2)=1e-3;
options(3)=1e-3;
x0=normrnd(0,0.5,1,6);
vlb=-2*ones(1,6);
vub=2*ones(1,6);
options=optimset('GradObj','on','Display','iter');
x=fmincon('fung1',x0,[],[],[],[],vlb,vub,[],options)
for k=2:N
    y1(k)=x(1)*y1(k-1)+x(2)*y2(k-1)+x(3)*u(k-1);
    y2(k)=x(4)*y1(k-1)+x(5)*y2(k-1)+x(6)*u(k-1);
end;
t=0:1:N-1;
plot(t,y2,'*',t,y),figure(gcf),pause
z1=fft(y2);
z2=fft(y);
plot(t,abs(z1),t,abs(z2),'*'),figure(gcf),pause
plot(t,angle(z1),t,angle(z2),'*'),figure(gcf),pause
```

Главна програма използва градиента $grad\ S$, пресметнат в подпрограмата fung1. Кодът на тази програма е

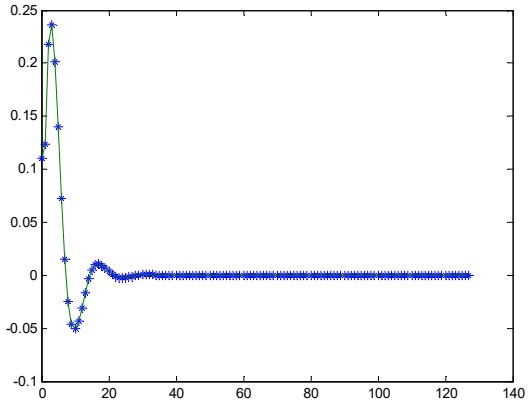
```
function [J,dJ]=fung1(x)
global N ye y1 y2 u
for k=2:N
    y1(k)=x(1)*y1(k-1)+x(2)*y2(k-1)+x(3)*u(k-1);
    y2(k)=x(4)*y1(k-1)+x(5)*y2(k-1)+x(6)*u(k-1);
end;
J=0.5*sum((y2-ye).^2);
la1=0;la2=0;g1=0;g2=0;g3=0;g4=0;g5=0;g6=0;g7=0;g8=0;g9=0;g10=0;
for k=N:-1:2
    la1=la1*x(1)+la2*x(4);%+(y1(k)-ye1(k));
    la2=la1*x(2)+la2*x(5)+(y2(k)-ye(k));
    g1=la1*y1(k-1)+g1;
    g2=la1*y2(k-1)+g2;
    g3=la1*u(k-1)+g3;
```

```
    g4=la2*y1(k-1)+g4;
    g5=la2*y2(k-1)+g5;
    g6=la2*u(k-1)+g6;
end;
dJ=[g1 g2 g3 g4 g5 g6];
```

Резултатът от работата на програмата е следният. Неизвестните параметри (тегла на мрежата) имат стойности:

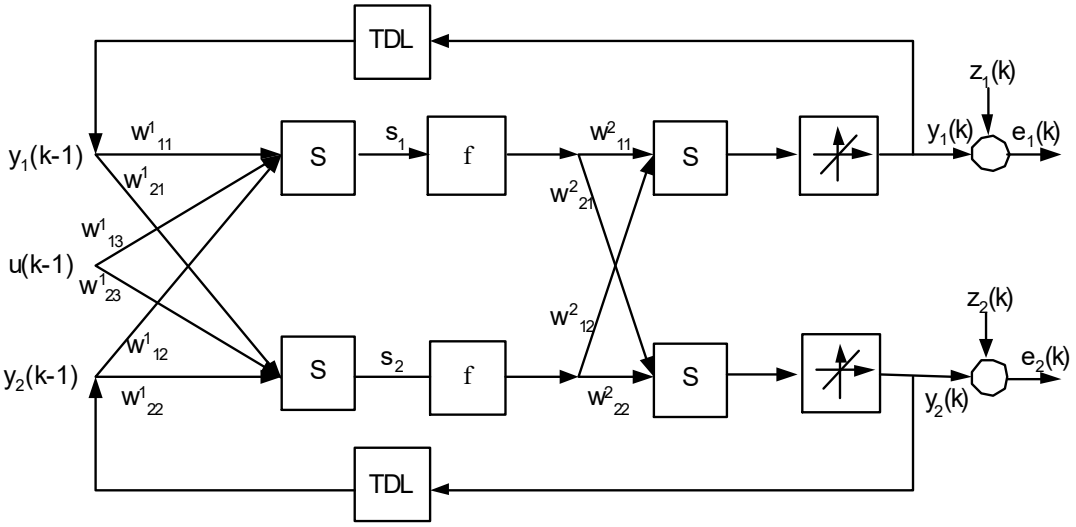
$x_1 = 0.8404$, $x_2 = 0.3139$, $x_3 = 0.3697$, $x_4 = 0.4256$, $x_5 = 0.6092$ и $x_6 = 0.0564$.

На фиг.1.63 са сравнени импулсните характеристики на еталонния филтър и на мрежата. Резултатът показва една добра апроксимация на еталонната характеристика от получената чрез мрежата характеристика.



фиг.1.63

1.12. Алгоритъм на коефициентите на чувствителност за обучение на двуслойна рекурентна невронна мрежа



Фиг. 1.64

Невронна мрежа от фиг.1.64 се описва със следната система от рекурентни уравнения:

(77)
$$y_1(k) = w^2_{11}\, f(s_1) + w^2_{12}\, f(s_2) = w^2_{11}\, f(w^1_{11}\, y_1(k-1) + w^1_{12}\, y_2(k-1) + \\ + w^1_{13}\, u(k-1)) + w^2_{12}\, f(w^1_{21}\, y_1(k-1) + w^1_{22}\, y_2(k-1) + w^1_{23}\, u(k-1))$$
$$y_2(k) = w^2_{21}\, f(s_1) + w^2_{22}\, f(s_2) = w^2_{21}\, f(w^1_{11}\, y_1(k-1) + w^1_{12}\, y_2(k-1) + \\ + w^1_{13}\, u(k-1)) + w^2_{22}\, f(w^1_{21}\, y_1(k-1) + w^1_{22}\, y_2(k-1) + w^1_{23}\, u(k-1))$$

(78) $$y_1(0) = y_2(0) = 0\,.$$

Въвежда се спомагателен вектор на теглата на невронната мрежа

(79) $$\mathrm{p} = (w^1_{11}\ w^1_{12}\ w^1_{13}\ w^1_{21}\ w^1_{22}\ w^1_{23}\ w^2_{11}\ w^2_{12}\ w^2_{21}\ w^2_{22})^T = (p_1,...,p_{10})^T\,.$$

Записва се функцията на средноквадратичната грешка на мрежата:

(80) $$J(\mathbf{p}) = \frac{1}{2}\sum_{k=1}^{N}\{[y_1(k) - z_1(k)]^2 + [y_2(k) - z_2(k)]^2\},$$

където N е броят на времевите отчети, а $z_1(k)$ и $z_2(k)$ са еталонните изходи.

За минимизирането на тази функция ще бъде приложен алгоритъм на нелинейните най-малки квадрати, а градиентът-изчислен чрез коефициентите на чувствителност. За целта се въвеждат квадратичните грешки:

(81) $$e_i(k) = [y_i(k) - z_i(k)],\ \text{за}\ i = 1,2;\ \ k = 1,2,...,N\,.$$

Необходимо е да се намери градиентът на грешката $E(\mathbf{p})$ спрямо параметрите p_j, т.е.:

(82)
$$\frac{\partial J}{\partial p_1} = \sum_{k=1}^{N}[e_1(k)\frac{\partial e_1(k)}{\partial p_1} + e_2(k)\frac{\partial e_2(k)}{\partial p_1}]$$
$$\frac{\partial J}{\partial p_2} = \sum_{k=1}^{N}[e_1(k)\frac{\partial e_1(k)}{\partial p_2} + e_2(k)\frac{\partial e_2(k)}{\partial p_2}]\ .$$
$$.....$$
$$\frac{\partial J}{\partial p_{10}} = \sum_{k=1}^{N}[e_1(k)\frac{\partial e_1(k)}{\partial p_{10}} + e_2(k)\frac{\partial e_2(k)}{\partial p_{10}}]$$

За целта трябва да се пресметнат производните на функциите $e_i(k)$ спрямо параметрите p_j. За целта се диференцира (81) и се получава

(83) $$\frac{\partial e_i(k)}{\partial p_j} = \frac{\partial y_i(k)}{\partial p_j}\ ,\ \text{за}\ i = 1,2\ \text{и}\ j = 1,...,10\,.$$

Тези производни се наричат коефициенти на чувствителността и се пресмятат за първия слой така (само за елемента $w^1_{11} = p_1$):

(84)
$$\frac{\partial y_1(k)}{\partial p_1} = \frac{\partial y_1(k)}{\partial w^1_{11}} = w^2_{11}\ f'(s_1)\left(y_1(k-1) + w^1_{11}\ \frac{\partial y_1(k-1)}{\partial w^1_{11}} + w^1_{12}\ \frac{\partial y_2(k-1)}{\partial w^1_{11}} \right) +$$
$$+ w^2_{12}\ f'(s_2)\left(w^1_{21}\ \frac{\partial y_1(k-1)}{\partial w^1_{11}} + w^1_{22}\ \frac{\partial y_2(k-1)}{\partial w^1_{11}} \right) =$$
$$= w^2_{11}\ f'(s_1) y_1(k-1) + \left(w^2_{11} f'(s_1) w^1_{11} + w^2_{12} f'(s_2) w^1_{21} \right) \frac{\partial y_1(k-1)}{\partial w^1_{11}} +$$
$$+ \left(w^2_{11} f'(s_1) w^1_{21} + w^2_{12} f'(s_2) w^1_{22} \right) \frac{\partial y_2(k-1)}{\partial w^1_{11}}$$

(85)
$$\frac{\partial y_1(k)}{\partial p_1} = \left(\frac{\partial y_1(k)}{\partial w^1_{11}} \right)^F + \left(\frac{\partial y_1(k)}{\partial y_1(k-1)} \right)^f \frac{\partial y_1(k-1)}{\partial w^1_{11}} + \left(\frac{\partial y_1(k)}{\partial y_2(k-1)} \right)^F \frac{\partial y_2(k-1)}{\partial w^1_{11}} =$$
$$= \left(\frac{\partial y_1(k)}{\partial p_1} \right)^F + \left(\frac{\partial y_1(k)}{\partial y_1(k-1)} \right)^F \frac{\partial y_1(k-1)}{\partial p_1} + \left(\frac{\partial y_1(k)}{\partial y_2(k-1)} \right)^F \frac{\partial y_2(k-1)}{\partial p_1}.$$

В тази формула индексът F показва, че производните с този индекс се пресмятат като в мрежа с еднопосочно обработване на данните, т.е. с пренебрегване на обратните връзки.

Производните (коефициенти на чувствителността) за втория слой се пресмятат, както следва (само за елемента $w^2_{11} = p_7$):

(86)
$$\frac{\partial y_1(k)}{\partial p_7} = \frac{\partial y_1(k)}{\partial w^2_{11}} = z_1 + w^2_{11}\ f'(s_1)\left(w^1_{11}\ \frac{\partial y_1(k-1)}{\partial w^2_{11}} + w^1_{12}\ \frac{\partial y_2(k-1)}{\partial w^2_{11}} \right) +$$
$$+ w^2_{12}\ f'(s_2)\left(w^1_{21}\ \frac{\partial y_1(k-1)}{\partial w^2_{11}} + w^1_{22}\ \frac{\partial y_2(k-1)}{\partial w^2_{11}} \right) =$$
$$= z_1 + \left(w^2_{11} f'(s_1) w^1_{11} + w^2_{12} f'(s_2) w^1_{21} \right) \frac{\partial y_1(k-1)}{\partial w^2_{11}} +$$
$$+ \left(w^2_{11} f'(s_1) w^1_{21} + w^2_{12} f'(s_2) w^1_{22} \right) \frac{\partial y_2(k-1)}{\partial w^2_{11}}$$

(87)
$$\frac{\partial y_1(k)}{\partial p_7} = \left(\frac{\partial y_1(k)}{\partial w^2_{11}} \right)^F + \left(\frac{\partial y_1(k)}{\partial y_1(k-1)} \right)^F \frac{\partial y_1(k-1)}{\partial w^2_{11}} + \left(\frac{\partial y_1(k)}{\partial y_2(k-1)} \right)^F \frac{\partial y_2(k-1)}{\partial w^2_{11}} =$$
$$= \left(\frac{\partial y_1(k)}{\partial p_7} \right)^F + \left(\frac{\partial y_1(k)}{\partial y_1(k-1)} \right)^F \frac{\partial y_1(k-1)}{\partial p_7} + \left(\frac{\partial y_1(k)}{\partial y_2(k-1)} \right)^F \frac{\partial y_2(k-1)}{\partial p_7}.$$

Уравненията (85) и (87) могат да се обобщят за всички останали тегла по следния начин [5]:

$$\frac{\partial y_i(k)}{\partial p_j} = \left(\frac{\partial y_i(k)}{\partial p_j}\right)^F + \left(\frac{\partial y_i(k)}{\partial y_1(k-1)}\right)^F \frac{\partial y_1(k-1)}{\partial p_j} + \left(\frac{\partial y_i(k)}{\partial y_2(k-1)}\right)^F \frac{\partial y_2(k-1)}{\partial p_j}$$

(88)

$$\frac{\partial y_i(0)}{\partial p_j} = 0; \quad i = 1,2; \quad j = 1,...,10; \quad k = 1,...,N,$$

където индексът F показва пресмятане на производните при пренебрегване на обратните връзки.

Уравненията (88) представляват една рекурентна система със съответните начални условия, решенията на която са коефициентите на чувствителност $\frac{\partial y_i(k)}{\partial p_j}$. Следователно, като се използуват тези формули и връзките (82), може да се пресметне градиентът и да се приложи градиентна оптимизационна процедура за минимизиране на функцията (80) и определяне на теглата на мрежата.

Този алгоритъм може да се приложи и за определяне на якобиана $J(\mathbf{p})$ на грешките дефинирани в (80) и да се реализират оптимизационни процедури от по- висок ред, например алгоритъм на Левенберг-Маркуард.

Предложеният алгоритъм лесно може да бъде обобщен за двуслойна невронна мрежа с повече от два неврона в слой.

Пример 1. В този пример еталонните данни се генерират чрез следната дискретна динамична система:

(92) $$z(k) = -a_1 y(k-1) - a_2 z(k-2) + b_0 u(k) + b_1 u(k-1) + b_2 u(k-2),$$

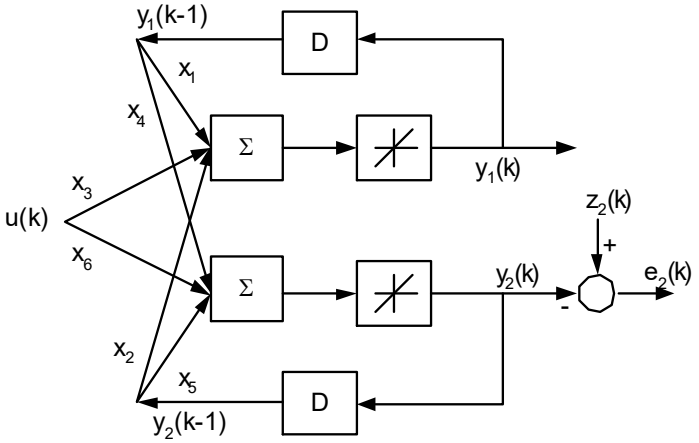
където $a_1 = -1.4496$, $a_2 = 0.6455$, $b_0 = 0.1098$, $b_1 = -0.0358$ и $b_2 = 0.1098$, с нулеви начални условия

(93) $$z(-1) = z(0) = 0$$

и единично външно въздействие

(94) $$u(1) = 1,\ u(2) = u(3) = ... = 0.$$

Обучаващите данни са единичното входно въздействие (94) и еталонен изход, явяващ се решението на (92) при начални условия (93) /импулсна характеристика на филтъра/.



фиг.1.65

Невронната мрежата се описва със следната система:

$$y_1(k) = x_1 y_1(k-1) + x_2 y_2(k-1) + x_3 u(k)$$
$$y_2(k) = x_4 y_1(k-1) + x_5 y_2(k-1) + x_6 u(k)$$

.

Минимизира се средноквадратичната грешка

$$S = \frac{1}{2}\sum_{k=1}^{N}(y_2(k) - z_2(k))^2 = \frac{1}{2}\sum_{k=1}^{N} e_2(k)^2$$

спрямо теглата на мрежата.

Градиентът на грешката S се получава, като следва

(95) $$grad(S(x_1,...,x_6)) = (\sum_{k=1}^{N} e_2(k)\frac{\partial y_2(k)}{\partial x_1},...,\sum_{k=1}^{N} e_2(k)\frac{\partial y_2(k)}{\partial x_6}),$$

където производните (коефициенти на чувствителността) се пресмятат от рекурентната система при нулеви начални условия:

(96)
$$\frac{\partial y_1(k)}{\partial x_1} = y_1(k-1) + x_1\frac{\partial y_1(k-1)}{\partial x_1} + x_2\frac{\partial y_2(k-1)}{\partial x_1}$$
$$\frac{\partial y_1(k)}{\partial x_2} = y_2(k-1) + x_1\frac{\partial y_1(k-1)}{\partial x_2} + x_2\frac{\partial y_2(k-1)}{\partial x_2}$$
$$\frac{\partial y_1(k)}{\partial x_3} = u(k-1) + x_1\frac{\partial y_1(k-1)}{\partial x_3} + x_2\frac{\partial y_2(k-1)}{\partial x_3}$$
$$\frac{\partial y_1(k)}{\partial x_4} = x_1\frac{\partial y_1(k-1)}{\partial x_4} + x_2\frac{\partial y_2(k-1)}{\partial x_4}$$
$$\frac{\partial y_1(k)}{\partial x_5} = x_1\frac{\partial y_1(k-1)}{\partial x_5} + x_2\frac{\partial y_2(k-1)}{\partial x_5}$$
$$\frac{\partial y_1(k)}{\partial x_6} = x_1\frac{\partial y_1(k-1)}{\partial x_6} + x_2\frac{\partial y_2(k-1)}{\partial x_6}$$

(97)

$$\frac{\partial y_2(k)}{\partial x_1} = x_4 \frac{\partial y_1(k-1)}{\partial x_1} + x_5 \frac{\partial y_2(k-1)}{\partial x_1}$$
$$\frac{\partial y_2(k)}{\partial x_2} = x_4 \frac{\partial y_1(k-1)}{\partial x_2} + x_5 \frac{\partial y_2(k-1)}{\partial x_2}$$
$$\frac{\partial y_2(k)}{\partial x_3} = x_4 \frac{\partial y_1(k-1)}{\partial x_3} + x_5 \frac{\partial y_2(k-1)}{\partial x_3}$$
$$\frac{\partial y_2(k)}{\partial x_4} = y_1(k-1) + x_4 \frac{\partial y_1(k-1)}{\partial x_4} + x_5 \frac{\partial y_2(k-1)}{\partial x_4}$$
$$\frac{\partial y_2(k)}{\partial x_5} = y_2(k-1) + x_4 \frac{\partial y_1(k-1)}{\partial x_5} + x_5 \frac{\partial y_2(k-1)}{\partial x_5}$$
$$\frac{\partial y_1(k)}{\partial x_6} = u(k-1) + x_4 \frac{\partial y_1(k-1)}{\partial x_6} + x_5 \frac{\partial y_2(k-1)}{\partial x_6}$$

за $k = 1,..., N$.

Като се използват формулите (96)-(97), се определя градиентът $grad\ S$ от (95).

Задачата се реализира със следната програма.

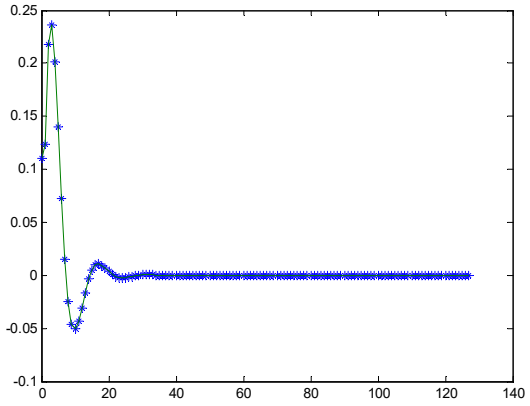
```
clear all
global N ye y1 y2 u
N=128;
a1=-1.4496;a2=0.6455;b0=0.1098;b1=-0.0358;b2=0.1098;
y(1)=b0;y(2)=-a1*y(1)+b1;y(3)=-a1*y(2)-a2*y(1)+b2;
u=[1 zeros(1,N-1)];
for k=4:N
    y(k)=-a1*y(k-1)-a2*y(k-2);
end;
ye=y; y1=zeros(1,N); y2=[b0 zeros(1,N-1)]; u=[1 zeros(1,N-1)];
options(1)=1;
options(14)=1000;
options(2)=1e-3;
options(3)=1e-3;
x0=normrnd(0,0.5,1,6);
vlb=-2*ones(1,6);
vub=2*ones(1,6);
options=optimset('GradObj','on','Display','iter');
x=fmincon('fung2',x0,[],[],[],[],vlb,vub,[],options)
for k=2:N
    y1(k)=x(1)*y1(k-1)+x(2)*y2(k-1)+x(3)*u(k-1);
    y2(k)=x(4)*y1(k-1)+x(5)*y2(k-1)+x(6)*u(k-1);
end;
t=0:1:N-1;
```

```
plot(t,y2,'*',t,y),figure(gcf),pause
z1=fft(y2);
z2=fft(y);
plot(t,abs(z1),t,abs(z2),'*'),figure(gcf),pause
plot(t,angle(z1),t,angle(z2),'*'),figure(gcf),pause
```

Горната главна програма използва градиента *grad S* пресметнат в подпрограмата fung2. Кода на програмата fung2 е

```
function [J,dJ]=fung2(x)
global N ye y1 y2 u

for k=2:N
    y1(k)=x(1)*y1(k-1)+x(2)*y2(k-1)+x(3)*u(k-1);
    y2(k)=x(4)*y1(k-1)+x(5)*y2(k-1)+x(6)*u(k-1);
end;
J=0.5*sum((y2-ye).^2);
dydx1=zeros(6,N);dydx2=zeros(6,N);
for k=2:N
    dydx1(1,k)=y1(k-1)+x(1)*dydx1(1,k-1)+x(2)*dydx2(1,k-1);
    dydx1(2,k)=y2(k-1)+x(1)*dydx1(2,k-1)+x(2)*dydx2(2,k-1);
    dydx1(3,k)=u(k-1)+x(1)*dydx1(3,k-1)+x(2)*dydx2(3,k-1);
    dydx1(4,k)=x(1)*dydx1(4,k-1)+x(2)*dydx2(4,k-1);
    dydx1(5,k)=x(1)*dydx1(5,k-1)+x(2)*dydx2(5,k-1);
    dydx1(6,k)=x(1)*dydx1(6,k-1)+x(2)*dydx2(6,k-1);
    dydx2(1,k)=x(4)*dydx1(1,k-1)+x(5)*dydx2(1,k-1);
    dydx2(2,k)=x(4)*dydx1(2,k-1)+x(5)*dydx2(2,k-1);
    dydx2(3,k)=x(4)*dydx1(3,k-1)+x(5)*dydx2(3,k-1);
    dydx2(4,k)=y1(k-1)+x(4)*dydx1(4,k-1)+x(5)*dydx2(4,k-1);
    dydx2(5,k)=y2(k-1)+x(4)*dydx1(5,k-1)+x(5)*dydx2(5,k-1);
    dydx2(6,k)=u(k-1)+x(4)*dydx1(6,k-1)+x(5)*dydx2(6,k-1);
end;
S=zeros(1,6);
for j=1:6
for k1=1:N
    S(j)=S(j)+(y2(k1)-ye(k1))*dydx2(j,k1);
end;end;
dJ=S;
dg=[];
```

Решението се получава при следните тегловни коефициенти: $x_1 = 0.6574$, $x_2 = -0.3007$, $x_3 = 0.3231$, $x_4 = 0.4145$, $x_5 = 0.7916$, $x_6 = 0.0364$ и резултатът е показан на следващата графика.

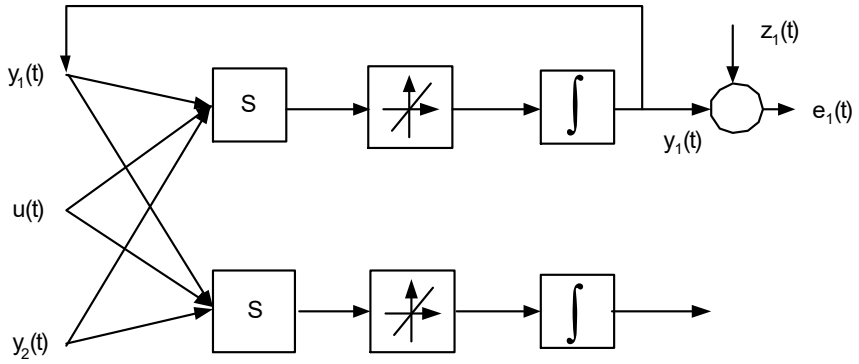


фиг.1.66

1.13 Непрекъснати невронни мрежи

Архитектура и обучение

Пример за непрекъсната невронна мрежа е показан на фиг.1.67.



Фиг. 1.67

Системата обикновени диференциални уравнения на мрежата е следната:

$$\begin{aligned} \dot{y}_1(t) &= w_{11}y_1(t) + w_{12}y_2(t) + w_{13}u(t) \\ \dot{y}_2(t) &= w_{21}y_1(t) + w_{22}y_2(t) + w_{23}u(t) \end{aligned} \quad . \tag{98}$$

при нулеви начални условия $y_1(0) = y_2(0) = 0$.

Поставя се задача да се определят неизвестните тегла на мрежата w_{ij}, така че средноквадратичната грешка

$$J(T) = \frac{1}{2}\int_0^T (y_1(t) - z_1(t))^2\, dt \tag{99}$$

да се минимизира. В (99) $y_1(t)$ се получава от мрежата, а $z_1(t)$ е еталонът.

За определянето на градиента на функцията (99) се използва алгоритъмът на множителите на Лагранж. За целта първо се съставя функцията на Хамилтън за задачата (98),(99)

$$H = \frac{1}{2}(y_1(t) - z_1(t))^2 + \lambda_1(w_{11}y_1(t) + w_{12}y_2(t) + w_{12}u(t)) +$$
$$+ \lambda_2(w_{21}y_1(t) + w_{22}y_2(t) + w_{23}u(t))$$

Чрез нея се определя спрегнатата система:

(100)
$$\begin{cases} \dot{\lambda}_1(t) = -\dfrac{\partial H}{\partial y_1} = -(y_1(t) - z_1(t)) - \lambda_1 w_{11} - \lambda_2 w_{21} \\ \dot{\lambda}_2(t) = -\dfrac{\partial H}{\partial y_2} = -\lambda_1 w_{12} - \lambda_2 w_{22} \end{cases}$$

(101)
$$\begin{cases} \dot{\gamma}_1(t) = -\dfrac{\partial H}{\partial w_{11}} = -\lambda_1 y_1(t) \\ \dot{\gamma}_2(t) = -\dfrac{\partial H}{\partial w_{12}} = -\lambda_1 y_2(t) \\ \dot{\gamma}_3(t) = -\dfrac{\partial H}{\partial w_{13}} = -\lambda_1 u(t) \\ \dot{\gamma}_4(t) = -\dfrac{\partial H}{\partial w_{21}} = -\lambda_2 y_1(t) \\ \dot{\gamma}_5(t) = -\dfrac{\partial H}{\partial w_{22}} = -\lambda_2 y_2(t \\ \dot{\gamma}_6(t) = -\dfrac{\partial H}{\partial w_{23}} = -\lambda_2 u(t) \end{cases}$$

с граничните условия:

(102) $\lambda_1(T) = \lambda_2(T) = 0$, $\gamma_1(T) = \gamma_2(T) = \gamma_3(T) = \gamma_4(T) = \gamma_5(T) = \gamma_6(T) = 0$.

Алгоритъмът за определяне на градиента е двустъпков:

а) решава се задачата (98) при съответните начални условия и траекториите й се запаметяват;

б) решава се системата (100-101) при граничните условия (102) от $t = T$ до $t = 0$, т.е. в обратно време и след като се пресметнат $\gamma_1(0), \gamma_2(0), \gamma_3(0)$ и $\gamma_4(0)$, това са компонентите на градиента на функцията (99):

$$(\frac{\partial J}{\partial w_{11}}\frac{\partial J}{\partial w_{12}}\frac{\partial J}{\partial w_{13}}\frac{\partial J}{\partial w_{21}}\frac{\partial J}{\partial w_{22}}\frac{\partial J}{\partial w_{23}})^T = (\gamma_1(0)\,\gamma_2(0)\,\gamma_3(0)\,\gamma_4(0)\,\gamma_5(0)\,\gamma_6(0))^T .$$

За илюстрация на алгоритъма се разгледа следния тестов пример.

Пример 1. Използва се структурата на мрежата от фиг.1.68 без входно въздействие $u(t)$:

$$\begin{aligned} \dot{y}_1(t) &= w_{11}y_1(t) + w_{12}y_2(t) \\ \dot{y}_2(t) &= w_{21}y_1(t) + w_{22}y_2(t) \end{aligned} \tag{103}$$

при начални условия $y_1(0) = 0,\ y_2(0) = 1$ и същата функция

$$J(T) = \frac{1}{2}\int_0^T (y_1(t) - z_1(t))^2\,dt . \tag{104}$$

За еталон се използва решението на системата (едномасов динамичен модел):

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= -\frac{c}{m}z_2 - \frac{k}{m}z_1 + \frac{1}{m}u(t) \end{aligned} \tag{105}$$

при начални условия $z_1(0) = 0; z_2(0) = 1$, с входно въздействие $u(t) \equiv 0$, т.е. за еталонът е импулсната характеристика на системата (105).

Пресмятането на функцията (104) може да се получи, като се реши системата:

$$\begin{aligned} \dot{y}_1(t) &= w_{11}y_1(t) + w_{12}y_2(t) \\ \dot{y}_2(t) &= w_{21}y_1(t) + w_{22}y_2(t) \\ \dot{y}_3(t) &= \frac{1}{2}(y_1(t) - z_1(t))^2 \end{aligned} \tag{106}$$

при начални условия $y_1(0) = 0; y_2(0) = 1; y_3(0) = 0$. Стойността на интеграла (104) се получава от равенството $J(T) = y_3(T)$.

След обучението на мрежата и определяне на параметрите w_{ij} се симулира поведението на мрежата и изходният процес се сравнява с решението на еталонната задача (100-102) (фиг.1.69), като се подава едно и също ударно входно въздействие

$$\begin{aligned} x0(t) &= 21.2\,t^{1.5}e^{-4t}\sin(e^{1.89t} - 1) \\ u(t) &= \frac{1}{m}(kx0(t) + c\dot{x}0(t) \end{aligned} \tag{107}$$

и нулеви начални условия Съответните параметри са зададени в програмата.

Конкретната задача е решена с оптимизационната процедура fmincon, като се използва режимът с числено пресмятане на градиента на целевата функция (104). При реализиране на пълния алгоритъм трябва да бъде добавена спрегнатата система (100-101), като тя се решава на всяка оптимизационна стъпка при граничните условия (102).

Главната програма modelnet се обръща още и към подпрограмите funh, funh2, funh3, в които се задават десните страни на системата (105) със или без входното въздействие $u(t)$ от (107).

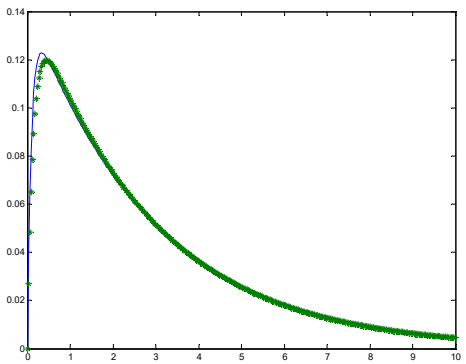
```
function modelnet
clear all
global w N t1 et c k m
N=10;c=75;k=25;m=10;
[t1,et]=ode15s(@funh,[0:0.03:N],[0 1]);
plot(t1,et(:,1)),figure(gcf),pause
v0=normrnd(0,1,4,1);
vub=[10;10;10;10];
vlb=[-10;-10;-10;-10];
options=optimset('Display','iter','MaxIter',100);
[w,fval]=fmincon(@funopth,v0,[],[],[],[],vlb,vub,[],options)
[t,y]=ode15s(@funh1,[0:0.03:N],[0 1 0]);
plot(t,y(:,1),t1,et(:,1),'*'),figure(gcf),pause
y0=[0 0];
[t2,y2]=ode15s(@funh2,[0:0.03:N],y0);
[t3,y3]=ode15s(@funh3,[0:0.03:N],y0);
plot(t2,y2(:,1),t3,y3(:,1),'*'),figure(gcf),pause
%-------------------------------------------------
function [f,g]=funopth(x)
global w N t1 et
w=x;
y0=[0;1;0];
[t,y]=ode15s(@funh1,[0 50],y0);
N1=length(t);
f=y(N1,3);
g=[];
%-------------------------------------------------
function [df]=funh(t,y)
global c k m
df(1)=y(2);
df(2)=-(c/m)*y(2)-(k/m)*y(1);
df=df';
function [df]=funh1(t,y)
global w N t1 et
et1=interp1(t1,et(:,1),t);%,pause;
df(1)=w(1)*y(1)+w(2)*y(2);
df(2)=w(3)*y(1)+w(4)*y(2);
df(3)=1000*(y(1)-et1)^2;
```

```
df=df';
function [df]=funh2(t,y)
global c k m
df(1)=y(2);
df(2)=-(c/m)*y(2)-(k/m)*y(1)+udar(t);
df=df';
function [df]=funh3(t,y)
global w N t1 et
df(1)=w(1)*y(1)+w(2)*y(2);
df(2)=w(3)*y(1)+w(4)*y(2)+udar(t);
df=df';
function u=udar(t)
global c k m
x01=21.2*t.^1.5;
x02=exp(-4*t);
x03=exp(1.89*t);
x04=sin(x03-1);
x0=x01.*x02.*x04;
x0t=21.2*1.5*t.^0.5.*x02.*x04+...
   x01.*(-4*x02).*x04+...
   x01.*x02.*cos(x03-1)*1.89.*x03;
u=(k*x0+c*x0t)/m;
```

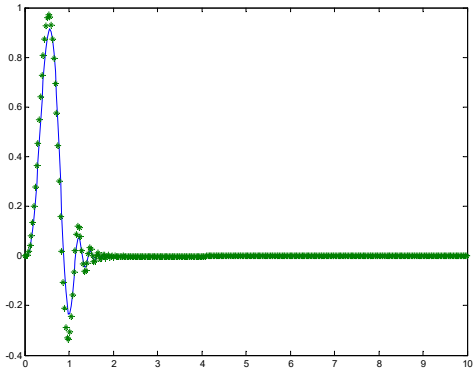
След обучението на мрежата са получени следните стойности за теглата:

$$w_{11} = -2.0435\,, \quad w_{12} = 1.4593\,, \quad w_{21} = 10.000\,, \; w_{22} = 8.9014\,.$$

Еталонната импулсна характеристика и тази на вече обучената мрежа са изобразени на фиг.1.68.



фиг.1.68



фиг.1.69

Глава 2. Невронни мрежи с радиални базисни функции, самоорганизиращи се невронни мрежи и невронни мрежи на Хопфилд

2.1. Невронни мрежи с радиални базисни функции

Невронните мрежи от този тип се използват основно за решаване на задачи, свързани с апроксимации, които се реализират чрез проектиране от безкрайно мерно пространство в крайно мерно пространство с базис от радиални функции. Затова по-долу ще бъдат дадени някои основни радиални функции.

2.1.1. Примери за радиални базисни функции:

а) равнинни сплайн - функции

$$\phi(x) = x^2 \log(x);$$

б) гаусови функции

$$\phi(x) = e^{-x^2/a^2};$$

в) обобщени квадратични функции

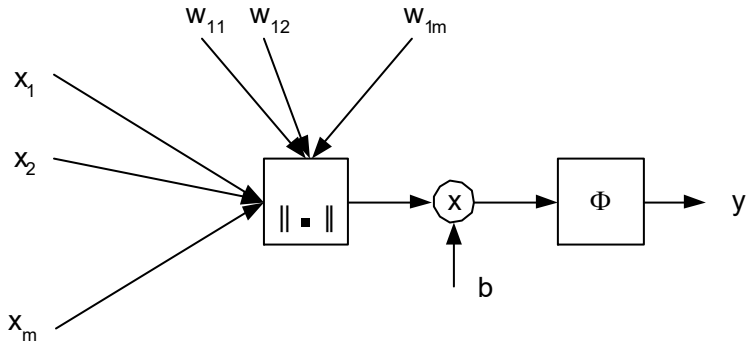
$$\phi(x) = (x^2 + a^2)^{1/2};$$

г) обратни обобщени квадратични функции

$$\phi(x) = (x^2 + a^2)^{-1/2}.$$

Радиалните базисни невронни мрежи най-често са с гаусови активиращи функции. Затова по-нататък се разглеждат мрежи с активираща функция $\Phi(x) = e^{-x^2}$.

2.1.2. Структура на единичен неврон с гаусова активираща функция.



фиг.2.1

Изходът на неврона се получава по следната формула

(1) $$y = \Phi(\| \mathbf{W} - \mathbf{x} \| b),$$

където

$\Phi(x) = e^{-x^2}$ и $\Phi(\| \mathbf{W} - \mathbf{x} \| b) = e^{-(\|\mathbf{w}-\mathbf{x}\|b)^2}$;

$\| \mathbf{W} - \mathbf{x} \| = \sqrt{(\mathbf{W}_{1,1} - x_1)^2 + ... + (\mathbf{W}_{1,\mathrm{m}} - x_m)^2}$ е дължината на вектора $\mathbf{W} - \mathbf{x}$;

b е свободно тегло;

$\mathbf{W} = (w_{1,1}...w_{1,\mathrm{m}})^{\mathrm{T}}$ - вектор от теглата на неврона;

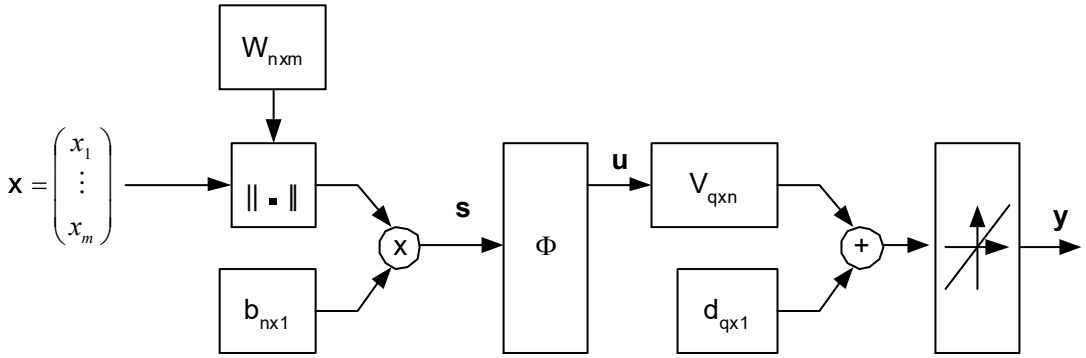
$\mathbf{x} = (x_1,...,x_m)^T$ - векторът от входовете на неврона.

Най-често входовете зависят от дискретната променлива k, затова формулата (1) може да се запише по-общо така:

(2) $$y(k) = \Phi(\mathbf{W} - \mathbf{x}(k) \| b)$$, за $k = 1,2,...$.

Структура на невронна мрежа с гаусови активиращи функции

Разглежда се права двуслойна невронна мрежа с първи слой гаусови активиращи функции и втори линеен слой. Първият слой има n неврона, а вторият има q неврона (фиг.2.2).



фиг.2.2

Мрежата се описва от системата:

а) за първия слой

$$\begin{pmatrix} s_1(k) \\ s_2(k) \\ ..... \\ s_n(k) \end{pmatrix} = \begin{pmatrix} \|\mathbf{W_1} - \mathbf{x}(k)\| b_1 \\ \|\mathbf{W_2} - \mathbf{x}(k)\| b_2 \\ ..... \\ \|\mathbf{W_n} - \mathbf{x}(k)\| b_n \end{pmatrix} \qquad \begin{pmatrix} u_1(k) \\ u_2(k) \\ ..... \\ u_n(k) \end{pmatrix} = \begin{pmatrix} \Phi(s_1(k)) \\ \Phi(s_2(k)) \\ ..... \\ \Phi(s_n(k)) \end{pmatrix}$$

или съкратено $\mathbf{u} = \mathbf{\Phi}(\mathbf{s})$,

където $\mathbf{\Phi(s)}$ е векторната функция $\Phi(\mathbf{s}) = (\Phi(s_1),...,\Phi(s_n))^T$;

б) за втория слой

$$\begin{pmatrix} y_1(k) \\ y_2(k) \\ ..... \\ y_q(k) \end{pmatrix} = \begin{pmatrix} v_{11}\ v_{12}\ ...v_{1n} \\ v_{21}\ v_{22}\ ...v_{2n} \\ ..... \\ v_{q1}\ v_{q2}\ ...v_{qn} \end{pmatrix} \begin{pmatrix} u_1(k) \\ u_2(k) \\ .... \\ u_n(k) \end{pmatrix} + \begin{pmatrix} d_1(k) \\ d_2(k) \\ .... \\ d_q(k) \end{pmatrix}.$$

Окончателно уравненията на мрежата в матрична форма са:

$$\begin{aligned} \mathbf{u} &= \mathbf{\Phi}(\| \mathbf{W}_1 - \mathbf{x} \| b_1,...,\| \mathbf{W}_n - \mathbf{x} \| b_n) \\ \mathbf{y} &= \mathbf{V}.\mathbf{u} + \mathbf{d}, \end{aligned} \tag{3}$$

където

$\Phi(x) = e^{-x^2}$ и $\Phi(\| \mathbf{W_i} - \mathbf{x} \| b) = e^{-(\|\mathbf{w_i}-\mathbf{x}\|b)^2}$;

$\mathbf{W_i}$ - i-тият ред на матрицата $\mathbf{W} \in R^{n \times m}$ от тегловни вектори на първия слой;

$\mathbf{V}$ - матрицата от тегловни вектори на втория слой, $\mathbf{V} \in R^{q \times n}$;

$\mathbf{b}$ - векторът от свободни тегла на първия слой, $\mathbf{b} \in R^n$;

$\mathbf{d}$ - векторът от свободни тегла на втория слой, $\mathbf{d} \in R^q$;

$\| \mathbf{W_i} - \mathbf{x} \| = \sqrt{(\mathbf{W}_{i,1} - x_1)^2 + ... + (\mathbf{W}_{i,m} - x_m)^2}$ е дължината на вектора $\mathbf{W_i} - \mathbf{x}$.

Тъй като векторът от входове най-често зависи от дискретната променлива k, то системата (3) може да се запише така:

$$\begin{aligned} \mathbf{u}(k) &= \mathbf{\Phi}(\| \mathbf{W_1} - \mathbf{x}(k) \| b_1,...,\| \mathbf{W_n} - \mathbf{x}(k) \| b_n) \\ \mathbf{y}(k) &= \mathbf{V}.\mathbf{u}(k) + \mathbf{d} \end{aligned} \text{ ,за } k = 1,2,... \tag{4}$$

Обучение на невронна мрежа с гаусови активиращи функции

Мрежата се обучава с метода на ортогоналните линейни най-малки квадрати [1]. Този алгоритъм ще бъде разяснен върху невронна мрежа, получена от невронната мрежа на фиг.2.2 чрез следните опростявания. Броят на изходите на втория слой се свежда до един и в (4) $\mathbf{V}$ е матрица- ред $\mathbf{V} = (v_1, v_2, ..., v_n)$. Свободните тегла на първия слой се приемат за единици ($b_i = 1$), а свободните тегла на втория слой - за нули ($d_i = 0$). Тогава от (4) ще се получи уравнението на мрежата:

$$y(k) = \sum_{i=1}^{n} v_i \, \Phi(\|\mathbf{x}(k) - \mathbf{c}_i \|) \text{ ,за } k = 1,2,..., N, \tag{5}$$

където

$\mathbf{c}_i = \mathbf{W_i} \in R^m$ - i-тият ред на матрицата $\mathbf{W} \in R^{n \times m}$ от тегла на първия слой (векторите $\mathbf{c}_i$ се наричат още и центрове на активиращите функции);

$\mathbf{x}(k) = (x_1(k),...,x_m(k))^T$ - вектор от входовете на мрежата;

v_i - неизвестни тегла на втория слой (състоящ се от един неврон), които ще бъдат намерени по метода на ортогоналните линейни най-малки квадрати.

Центровете на активиращите функции $\mathbf{c}_i \in R^m$ и векторите $\mathbf{x}(k) \in R^m$ са с еднаква размерност. Това позволява да се пресметне нормата

$$|| \mathbf{c}_i - \mathbf{x}(k) ||= \sqrt{(c_{i,1} - x_1(k))^2 + ... + (c_{i,m} - x_m(k))^2} .$$

Мрежата се обучава по редицата от входове

$$\mathbf{x}(k) = (x_1(k),...,x_m(k))^T \text{ за } k = 1,2,...,N$$

и еталонни изходи

$$y(k) \text{, за } k = 1,2,...,N ,$$

при това най-често $N > n$.

Задачата за обучение на мрежата е аналог на задачата за апроксимация на функцията

$$y(k) = v_1 e^{-(x(k)-c_1)^2} + v_2 e^{-(x(k)-c_2)^2} + ... + v_n e^{-(x(k)-c_n)^2} \text{, за } k = 1,2,...,N ,$$

по данните $(x(k), y(k))$, за $k = 1,2,...,N$, т.е. апроксимиращата функция е линейна комбинация от гаусови функции. Целта е да се намерят неизвестните v_i, така че “максимално добре” да се приближат дадените данни.

Обучението на мрежата се прави по следния алгоритъм. Центровете се избират равни на входните вектори. Ако $N > n$ (т.е. броят на отчетите е по-голям от броя на невроните в първия слой), то за центрове се избират само n на брой от входните данни:

$$\mathbf{c}_i = \mathbf{x}(i) \text{, за } i = 1,2,...,n .$$

С това обучението на първия слой на мрежата е завършено. Остава само да се намерят коефициентите v_i за $i = 1,...,n$, представляващи теглата на единствения неврон от втория слой. Това става по метода на ортогоналните линейни най-малки квадрати . За целта се заместват известните стойности $\mathbf{x}(i)$ и $y(i)$ за $i = 0,1,...,N$ в (5) и се получава системата

$$(6) \quad \begin{pmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{pmatrix} = \begin{pmatrix} \Phi(|| \mathbf{x}(1) - \mathbf{c}_1 ||) & \Phi(|| \mathbf{x}(1) - \mathbf{c}_2 ||) & \cdots & \Phi(|| \mathbf{x}(1) - \mathbf{c}_n ||) \\ \Phi(|| \mathbf{x}(2) - \mathbf{c}_1 ||) & \Phi(|| \mathbf{x}(2) - \mathbf{c}_2 ||) & \cdots & \Phi(|| \mathbf{x}(2) - \mathbf{c}_n ||) \\ \vdots & \vdots & & \vdots \\ \Phi(|| \mathbf{x}(N) - \mathbf{c}_1 ||) & \Phi(|| \mathbf{x}(N) - \mathbf{c}_2 ||) & \cdots & \Phi(|| \mathbf{x}(N) - \mathbf{c}_n ||) \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} .$$

Тъй като $N > n$, тази система е преопределена. В такъв случай тя се решава (относно неизвестните v_i) по метода на линейните най-малки квадрати. Тук се използва една разновидност на този метод, наречена ортогонални линейни най-малки квадрати. В основата на алгоритъма стои разлагането на основната матрица $\mathbf{A}$ на системата (6) на две матрици, една с ортогонални колони $\mathbf{Q}$ и една горна триъгълна $\mathbf{R}$, т.е.

(7) $$\mathbf{A} = \mathbf{Q}\,\mathbf{R},$$

където:

$\mathbf{Q}_{N\times n}$ - е матрица, чиито колони са ортогонални вектори, т.е. $\mathbf{Q}^T_{n\times N}\,\mathbf{Q}_{N\times n} = \mathbf{H}_{n\times n}$, където $\mathbf{H}$ е диагонална матрица;

$\mathbf{R}_{n\times n} = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1,n} \\ 0 & r_{22} & \cdots & r_{2,n} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & r_{n,n} \end{pmatrix}$ - горна триъгълна матрица.

Това разлагане може да се направи по метода на Грам-Шмид.

Ако матриците $\mathbf{Q}$ и $\mathbf{R}$ са вече намерени, тогава системата (6) има вида

(8) $$\mathbf{y} = \mathbf{QR}\,\mathbf{v},$$

където $\mathbf{y} = (y(1), y(2), ..., y(N))^T$ и $\mathbf{v} = (v_1, v_2, ..., v_n)^T$.

Преобразува се (8) и се получава:

$$\mathbf{y} = \mathbf{Q}(\mathbf{R}\,\mathbf{v})$$

$$\mathbf{Q}^T\mathbf{y} = \mathbf{Q}^T\mathbf{Q}(R\,\mathbf{v})$$

$$\mathbf{Q}^T\mathbf{y} = \mathbf{H}(\mathbf{R}\,\mathbf{v})$$

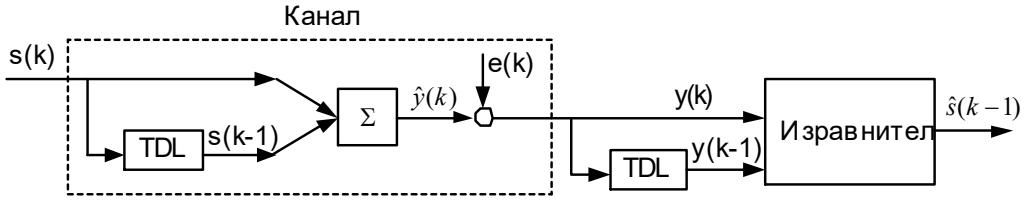
$$(\mathbf{R}\,\mathbf{v}) = \mathbf{H}^{-1}\mathbf{Q}^T\mathbf{y}$$

(9) $$\mathbf{g} = \mathbf{H}^{-1}\mathbf{Q}^T\mathbf{y}, \text{ където } \mathbf{g} = \mathbf{Rv}$$

Тъй като $\mathbf{H}$ е диагонална матрица, т.е. $\mathbf{H} = diag(h_1, h_2, ..., h_n)$, следователно $\mathbf{H}^{-1} = diag(\frac{1}{h_1}, \frac{1}{h_2}, ..., \frac{1}{h_n})$. Тогава, като се използва (9), директно се намира $\mathbf{g}$, след което се решава триъгълната система $\mathbf{g} = \mathbf{Rv}$ и се получава търсеният вектор $\mathbf{v}$ (теглата на втория слой). При това матрицата $\mathbf{R}$ е горна триъгълна и следователно системата $\mathbf{g} = \mathbf{Rv}$ винаги може да бъде решена. С това процесът на обучение е завършен.

Разгледаният по- горе алгоритъм се състои от две части: първата е получаване на разлагането (7) $\mathbf{A} = \mathbf{Q}\,\mathbf{R}$ на основната матрица $\mathbf{A}$ на системата (6) чрез алгоритъма за ортогонализация на Грам-Шмид, а втората е решаване на една система с диагонална матрица и втора система с триъгълна матрица. Следователно този алгоритъм е краен и представлява една ефективна процедура за обучение на невронна мрежа с радиални базисни функции.

Пример 1. Използване на невронна мрежа с радиални базисни функции като филтър. Разглежда се цифровата комуникационна система, показана на фиг.2.3



Фиг.2.3

В тази система $s(k)$ е входен дискретен сигнал, състоящ се от минус единици и плюс единици, които се появяват с равни вероятности. Първата част от системата е канал, който се моделира като нерекурсивен филтър с импулсен вход. Филтърът се описва с рекурентното уравнение

$$\hat{y}(k) = 0.5s(k) + s(k-1) \text{ за } k = 1,2,3,\ldots .$$

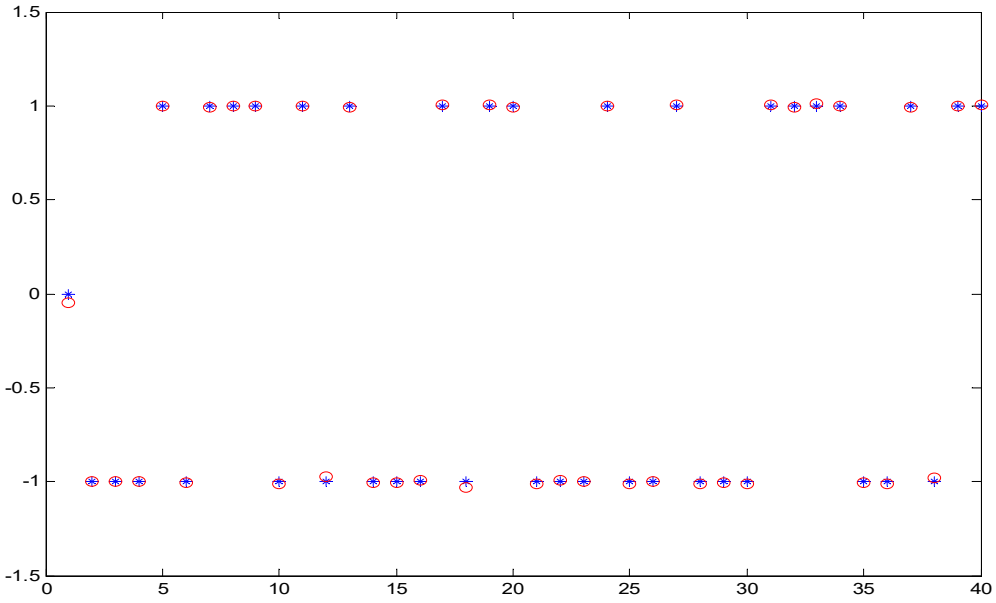
Към сигнала $\hat{y}(k)$ се добавя гаусов шум $e(k)$ с математическо очакване 0 и дисперсия 0.2 и на изхода на канала се получава сигналът $y(k)$. Така полученият сигнал $y(k)$ се подава на входа на изравнителя от фиг.2.3. Задачата му е да възстанови приблизително входния сигнал $s(k-1)$. Следователно тук се решава една обратна задача. Търсеният изравнител представлява нелинеен филтър и като такъв успешно може да се използва една мрежа с радиални базисни функции. Желателно е изравнителят да е от по-нисък ред, т.е. броят на центровете $\mathbf{c}_i$ да е възможно най-малък.

Следва програма за обучение на невронната мрежа на изравнителя от фиг.2.3.

```
clear all
k=1000
s=randsrc(1,k,[-1 1])
s1=[0, s(1:k-1)]
y=0.5*s+s1
e=normrnd(0,0.2,1,k)
```

```
y1=y+e
y2=[0,y1(1:k-1)]
P=[y1;y2]
T=s1
net=newrbe(P,T);
TT=sim(net,P);
t=1:40
plot(t,T(1:40),'b*',t,TT(1:40),'r*'),figure(gcf)
```

Резултатът от работата на тази програма е показан на фиг.2.4.



фиг.2.4

На тази фигура еталонният сигнал $s(k)$ е означен със звездички, а сигналът, получен от изравнителя $\hat{s}(k)$ - с кръгчета.

Пример 2. Апроксимира се на функцията $y(x) = e^{-x}\sin(2\pi x)$ в интервала $[-1,1]$ с невронна мрежа с радиални базисни функции. В програмата, реализираща алгоритъма, входният вектор е X, а еталонният изходен вектора е Y.

```
X = -1:.1:1;
Y=exp(-P).*sin(2*pi*P)
```

За получаване структурата на мрежата се използват командите

```
eg = 0.02; % среднoквадратична грешка
sc = 1;    % "ширина" на радиалните базисни функции
net=newrb(X,Y,eg,sc);
```

След като мрежата е обучена, тя се запомня в променливата net. С тази мрежа се симулира, като се използват командите

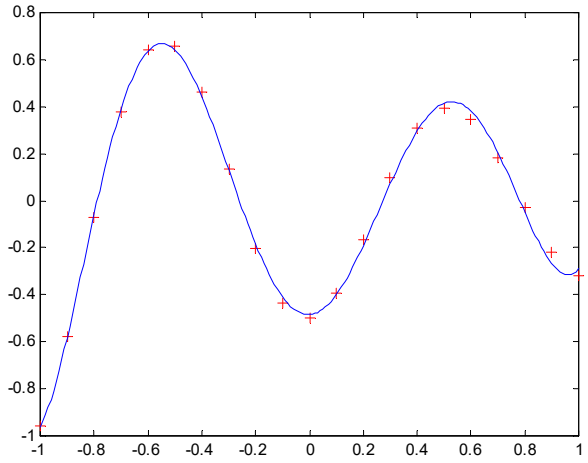
```
x=-1:.01:1;
```

```
y=sim(net,x);
```

При симулацията на входа на мрежата се подават същите входни данни, с които се извършва обучението. Накрая еталонните данни Y и симулираните Z се чертаят на обща графика, като за целта се използва командата

```
plot(X,Y,'r+',x,y)
```

Резултатът е показан на фиг.2.5. На тази фигура еталонните данни Y са показани с непрекъсната линия, а данните, получени от мрежата, – с плюсове.



фиг. 2.5

2.1.3. Два проблема при апроксимиране с невронна мрежа с радиални базисни функции

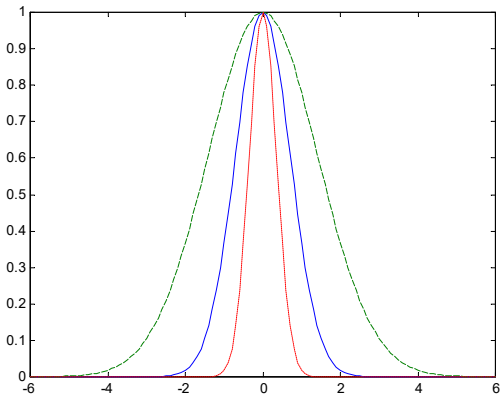
Тези проблеми се появяват, когато в командата

```
newrb(X,Y,eg,sc);
```

се избере неподходяща стойност за константата `sc`, определяща "ширината" на гаусовата функция (фиг.2.6). Тази функция се задава с израза

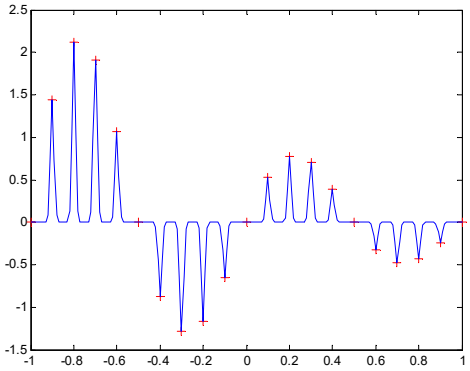
$$\phi(x) = \exp(-(x.b)^2).$$

На фиг.2.6 са дадени графики на тази функция при различни стойности на b. Вижда се от графиките,че ако се избере $b<1$ $(b>1)$, това би довело до"свиване" ("разширяване") на гаусовата функция.



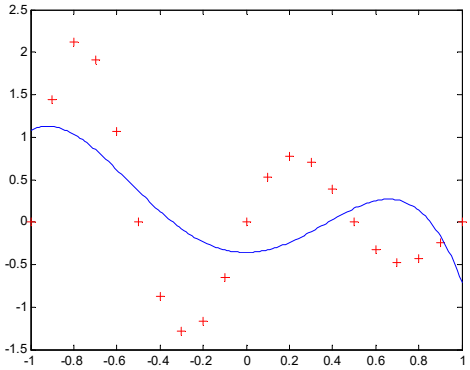
фиг.2.6

Това влияе върху апроксимацията по следния начин. Нека в пример 2 се зададе стойност `sc = 0.01`. В този случай се оказва, че параметърът е твърде малък, тъй като една гаусова функция отговаря само за една данна (точка). Тази данна се апроксимира точно, но в околност функцията на мрежата се стреми към нула (фиг.2.7).



Фиг.2.7

Ако се повтори същият пример 2 със стойност на `sc = 100`, то програмата ще даде резултата, показан на фиг.2.8.

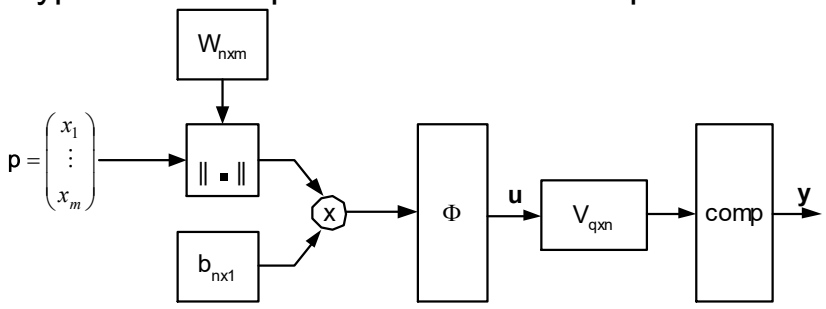


Фиг.2.8

В случая една гаусова функция отговаря за всички данни. Така каквато и линейна комбинация от гаусови функции да се състави, то тя ще се стреми да заглади максимално данните. В резултат функцията реализирана от мрежата няма да преминава през нито една от точките.

2.2. Вероятностни мрежи

Архитектура на тези мрежи е показана на фиг. 2.9.



фиг.2.9

Вероятностните мрежи и мрежите с радиални базисни функции (фиг.2.2) имат близка структура. При двете мрежи първият слой е с гаусови активиращи функции с центрове $\mathbf{c}_i$, които се избират равни на входните вектори. Разликата между тези две мрежи е във втория слой. При вероятностните мрежи той не е линеен, а се реализира на базата на съревнование (конкуренция), т.е. активиращата функция е `compet`. Тази активираща функция работи по следния алгоритъм. Ако на входа й постъпи вектор, то функцията връща вектор със същата размерност като входния. При това изходният вектор има една - единствена координата, равна на единица и всички останали координати - нули. Единствената ненулева координата на изходния вектор е разположена на мястото на най-голямата координата на входния вектор. Поради това си свойство функцията `compet` се нарича още и “победителят печели всичко”. Следователно вероятностната мрежа още със самото си съставяне е вече обучена, т.е. няма свободни тегла, които да бъдат определяни.

Тази мрежа може да се използва за класификация на входните данни, т.е. за причисляване на входните данни в съответни класове.

Пример 1. Във вектора T се задават центровете на класовете

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

които се асоциират с класовете T_c:

```
Tc = [1 2 3];
```

След това мрежата се обучава с командата `newpnn`

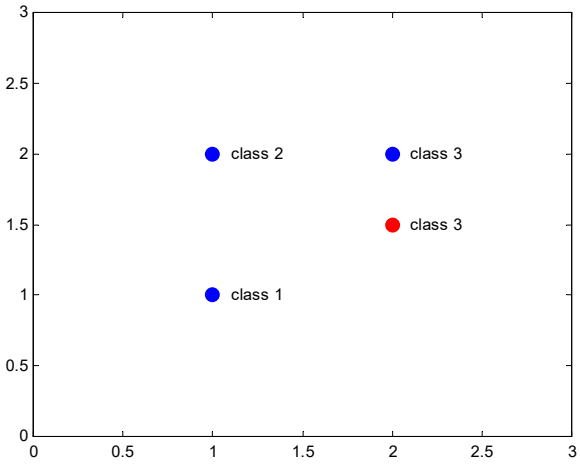
```
T = ind2vec(Tc);
spread = 1;
net = newpnn(P,T,spread);
```

С обучената мрежа се симулира, като се избира нов входен вектор $\mathbf{p} = (2, 1.5)^T$

```
p = [2; 1.5];
a = sim(net,p);
ac = vec2ind(a);
```

Пълния текст на програмата е следният:

```
P = [1 1 2; 1 2 2];
Tc = [1 2 3];
plot(P(1,:),P(2,:),'.','markersize',30)
axis([0 3 0 3])
text(P(1,1)+0.1,P(2,1),sprintf('class %g',Tc(1)))
text(P(1,2)+0.1,P(2,2),sprintf('class %g',Tc(2)))
text(P(1,3)+0.1,P(2,3),sprintf('class %g',Tc(3)))
figure(gcf), pause
T = ind2vec(Tc);
spread = 1;
net = newpnn(P,T,spread);
p = [2; 1.5];
a = sim(net,p);
ac = vec2ind(a);
hold on
plot(p(1),p(2),'r.','markersize',30)
text(p(1)+0.1,p(2),sprintf('class %g',ac))
figure(gcf)
hold off
```



Фиг.2.10

Резултатът от работата на програмата е показан на фиг.2.10. От фигурата се вижда, че мрежата причислява входния вектор $\mathbf{p} = (2 \quad 1{,}5)^T$ към третия клас.

2.3. Самоорганизиращи се невронни мрежи

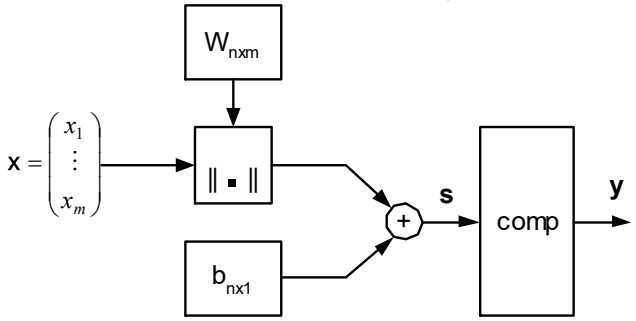
В този раздел ще бъдат разгледани следните три типа мрежи: еднослоен невронен класификатор, двуслоен невронен класификатор и самоорганизиращи се изображения (Self Organizing Maps). И трите типа мрежи се отличават първо с това, че активиращата функция на първия слой е `compet` и второ, че обучението на този слой става по правилото на Кохонен.

Обучението по правилото на Кохонен се различава принципно от разгледаните дотук подходи за обучение с това, че се използват само входни данни и изходът на мрежата не се сравнява със съответен еталон,т.е. реализира се обучение без учител.

Основната задача, която се решава с тези мрежи, е за класификация на вектори, т.е. причисляване на входен вектор към клас от вектори. Това се прави по следния алгоритъм. Подава се един вектор на входа на мрежата и на изхода й се получава вектор с координати една единица и останалите нули. Ако например тази единица се намира на второ място, то следва, че входният вектор се причислява към втория клас.

2.3.1. Еднослоен невронен класификатор

Това са еднослойни мрежи с архитектура, показана на фиг.2.11.



фиг.2.11

На тази фигура:

n е брой на невроните;

$\mathbf{W} \in R^{n \times m}$ - матрица от теглови вектори на слоя;

$\mathbf{b} \in R^{n \times 1}$ - вектор от свободни тегла на слоя;

$\mathbf{x} \in R^{m \times 1}$ - входен вектор на мрежата;
$comp$ - активираща функция.
Изходът на мрежата се получава по формулата

$$\mathbf{y} = compet(-\| \mathbf{W}_1 - \mathbf{x} \| + b_1, ..., -\| \mathbf{W}_\mathbf{n} - \mathbf{x} \| + b_n),$$

където
$\mathbf{W_i}$ - i-тият ред на матрицата $\mathbf{W}$;
b_i - i-тата координата на $\mathbf{b}$;
$\| \mathbf{W_i} - \mathbf{x} \| = \sqrt{(\mathbf{W_{i,1}} - x_1)^2 + ... + (\mathbf{W_{i,m}} - x_m)^2}$.

Функцията $\mathbf{y} = compet(\mathbf{s})$ връща вектор $\mathbf{y}$, който има същата размерност като входния вектор $\mathbf{s}$. При това векторът $\mathbf{y}$ има за координати една единица и останалите нули. Единицата е на позицията на най-големия елемент на входния вектор $\mathbf{s}$.

Обучението става по следната процедура:
Пресмятат се изразите
(10) $-\| \mathbf{W_i} - \mathbf{x} \| + b_i$ за $i = 1, ..., n$.

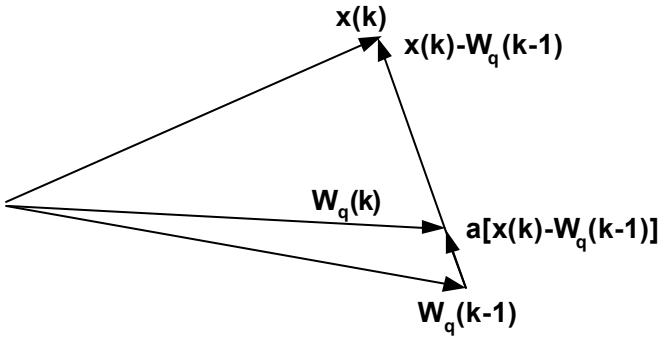
Първи случай. Приема се $b_i = 0$. Тогава членовете $-\| \mathbf{W_i} - \mathbf{x} \|$ са отрицателни и най-близкият до нулата определя вектора $\mathbf{W_i} - \mathbf{x}$ с най-малка дължина. Ще считаме, че векторите $\mathbf{W_i}$ и $\mathbf{x}$ са толкова "по-близки", колкото по-малка е дължината на вектора $\mathbf{W_i} - \mathbf{x}$. Нека векторът с най-малка дължина се получава за $i = q$, т.е. входният вектор $\mathbf{x}$ и q-тият ред $\mathbf{W_q}$ са "най-близки". В q-тия ред се съдържат теглата на q-тия неврон. Този неврон се нарича неврон победител.
След като е намерен невронът победител с тегла $\mathbf{W_q}$, то те се коригират по правилото на Кохонен, а теглата на останалите неврони не се променят. Правилото на Кохонен е следното:

$$\mathbf{W_q}(k) = \mathbf{W_q}(k-1) + a(\mathbf{x}(k) - \mathbf{W_q}(k-1)),$$

където
$\mathbf{W_q}(k)$ е новата (коригирана) стойност на теглата на неврона победител;
$\mathbf{W_q}(k-1)$ е старата стойност на теглата на неврона победител;
$\mathbf{x}(k)$ е входният вектор, чрез който е определен невронът победител;
a е константа - скорост на обучение (примерно $a = 0.01$).

Смисълът на това обучение е следният. Първо се намира вектора победител $\mathbf{W_q}(k-1)$, който е "най-близо" до входния вектор $\mathbf{x}(k)$. Това

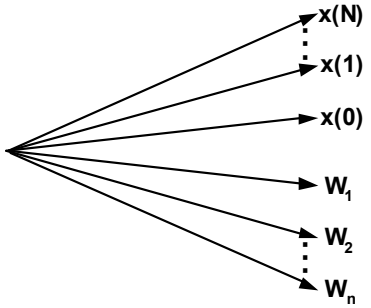
показва, че $\mathbf{W_q}(k-1)$ се явява център на класа, към който принадлежи този входен вектор. След това се премества този център по посока на вектора $\mathbf{x}(k)-\mathbf{W_q}(k-1)$. По този начин новата стойност на център $\mathbf{W_q}(k)$ ще се премести по-близо до входния вектор $\mathbf{x}(k)$ със скорост на обучение a (фиг.2.12).



Фиг.2.12

Втори случай. Векторът от свободни членове $\mathbf{b}$ не е нулев. Векторът $\mathbf{b}$ се добавя, за да се избегне така наречената ситуация на "мъртви неврони". Това е ситуация, при която част от невроните въобще не се обучават и добавянето на $\mathbf{b}$ позволява всеки един неврон да се обучава приблизително за едно и също време.

Следващият пример изяснява ситуацията "мъртви неврони". Нека входните вектори $\mathbf{x}(k)$ за $k=0,1,2,...,N$ са двумерни, следователно и матрицата от теглата на първия слой $\mathbf{W}$ е с размерност $\mathbf{W}\in R^{n\times 2}$. Да предположим, че началните стойности на входните вектори $\mathbf{x}(k)$,за $\kappa=0,1,2,...,N$ и теглата $\mathbf{W_i}$, за $i=1,...,n$ са изобразени на фиг.2.13.



Фиг.2.13

В такъв случай след постъпване на вектора $\mathbf{x}(0)$ на входа на невронната мрежа той ще е "най-близко" до вектора $\mathbf{W_1}$, следователно на първата стъпка ще се коригира (по правилото на Кохонен) векторът $\mathbf{W_1}$, като се премести "по-близо" до $\mathbf{x}(0)$. Но на втората стъпка новопостъпилият вектор $\mathbf{x}(1)$ ще е отново "най-близко" до вече

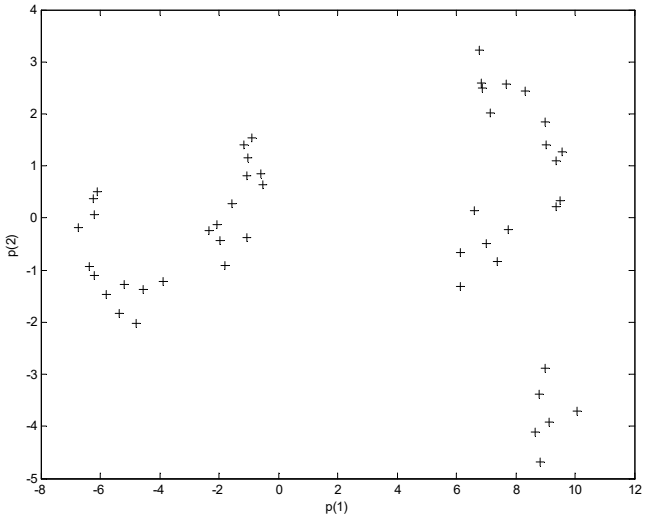
коригирания вектор $\mathbf{W_1}$ и отново ще се коригира този вектор. Очевидно, на следващата стъпка отново ще се коригира $\mathbf{W_1}$. Следователно през цялото време ще се променя само векторът $\mathbf{W_1}$, а останалите невронни тегла $\mathbf{W_i}$, $i = 2,...,n$ ще си остават непроменени ("мъртви неврони"). За да се избегне тази ситуация, към отрицателните разстояния $-\|\mathbf{W_i}\text{-}\mathbf{x}\|$ се прибавят положителни b_i, които осигурят коригиране на общата сума $-\|\mathbf{W_i}\text{-}\mathbf{x}\| + b_i$. При това се прилага следната стратегия, коефициентите b_i на тези неврони, които печелят, се намаляват, а b_i на невроните, които не печелят, не се променят. Това би позволило на невроните, които не са печелили да станат печеливши и да започне обучението и на тези неврони. Така се премахва ефектът на "мъртвите неврони". Вторият положителен ефект е, че приблизително един и същ процент от входните вектори $\mathbf{x}(k)$ ($k = 0,1,2,...,N$) обучават един неврон. Това позволява, ако една значителна част от входните вектори $\mathbf{x}(k)$ са концентрирани в една област, те да привлекат повече центрове $\mathbf{W_i}$ и така да се разбие един голям клас от входни вектори на няколко подкласа.

Пример 1. Следващата програмата генерира 48 данни, групирани в 8 класа с по 6 данни във всеки клас и обучава една мрежа от 8 неврона. Следователно мрежата ще класифицира векторите в 8 класа. За обучената мрежа се проверява към кой клас се причислява векторът $p = (0, 0.2)^T$.

```
clear all
%генериране на 48 данни, съсредоточени в 8 класа с по 6 данни във
всеки клас
x = [-10 10; -5 5];
nc=8;   %брой на класовете
n=6;    %брой на данните в един клас
d=0.5;
[r,q] = size(x);
minv = min(x')';
maxv = max(x')';
v = rand(r,nc) .* ((maxv-minv) * ones(1,nc)) + (minv *
ones(1,nc));%данните в един клас
t = nc*n;       %брой на всички данни от всички класове
y=nncopy(v,1,n);     %размножаване на данните от един клас
p = y + randn(r,t)*d;     %добавяне на случаен шум към данните
plot(p(1,:),p(2,:),'+r'),figure(gcf)
title('Input Vectors')
xlabel('p(1)')
```

```
ylabel('p(2)')
pause
%инициализиране на мрежата
net=newc([0 1;0 1],8,.1);
net=init(net);
%трениране на мрежата
net.trainParam.epochs=7;
net=train(net,P);
%симулиране на мрежата
P = [0; 0.2]
a=sim(net,P)
```

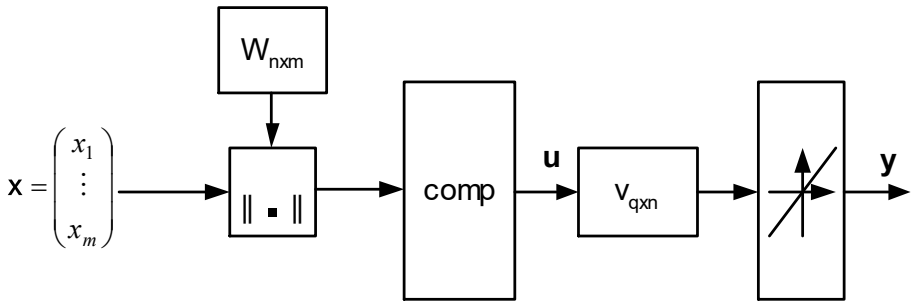
Генерираните случайни данни са показани на фиг.2.14. След симулирането е получен следният резултат a = (4,1) 1 ,т.е. единственият ненулев елемент на вектора a е на позиция (4,1). Това показва, че векторът $p = (0, 0.2)^T$ е причислен към четвъртия клас.



фиг.2.14

2.3.2. Двуслоен невронен класификатор

Това са двуслойни мрежи, първият слой на които е с активираща функция *compet* , а вторият слой е линеен (фиг.2.15).



фиг.2.15

На тази фигура:

n е броят на невроните;

$\mathbf{W} \in R^{n \times m}$ - матрицата от тегловни вектори на слоя;

$\mathbf{x} \in R^m$ - входен вектор на мрежата;

$comp$ - активираща функция;

$\mathbf{u} \in R^n$ - изходен вектор на първия слой;

q - брой на невроните във втория слой;

$\mathbf{V} \in R^{q \times n}$ - матрицата от теглови вектори на втория слой;

$\mathbf{y} \in R^q$ - изходен вектор на мрежата.

При симулирането с тези невронни мрежи изходът $\mathbf{y} \in R^q$ на мрежата се получава от уравненията:

$$\mathbf{y} = compet(-\| \mathbf{W_1} - \mathbf{x} \|,...,-\| \mathbf{W_n} - \mathbf{x} \|),$$

$$\mathbf{y} = \mathbf{V}.\mathbf{x},$$

където $\mathbf{W_i}$ - i-тият ред на матрицата $\mathbf{W}$.

Обучението е аналогично на това, разгледано в предходния раздел, със следните разлики: няма свободни тегла $\mathbf{b}$ в първия слой и има обучение на втори линеен слой.

Обучението на втория слой е свързано с това, че тези мрежи се обучават освен по входни данни $\mathbf{x}(k) \quad k = 1,2,...,N$ с правилото на Кохонен и по еталонни изходни данни $\mathbf{y}(k) \quad k = 1,2,...,N$. При това еталонните изходни вектори $\mathbf{y} \in R^q$ се отличават със свойството, че една от координатите им е единица, а всички останали са нули. Единицата има пореден номер, съвпадащ с номера на класа, към който трябва да бъде причислен входният вектор.

При обучението на тези мрежи първо се намира вектора победител $\mathbf{W_i}(k-1)$ от първия слой, който е “най-близо” до входния вектор $\mathbf{x}(k)$. В случая приемаме, че това е i-тият неврон. След това теглото $\mathbf{W_i}(k-1)$ се коригира по правилото на Кохонен. Остава да се изясни как се намират теглата на втория линеен слой. Ако за входа $\mathbf{x}$ трябва на изхода да се получи примерно векторът $\mathbf{y} = (0,0,1,0)^T$ и освен това за този вход невронът победител от първия слой има пореден номер i, то i-тият ред на втория слой $\mathbf{V_i}$ се приема равен на вектора $\mathbf{y}$, т.е. $\mathbf{V_i} = (0,0,1,0)$. Следователно тегловната матрица $\mathbf{V}$ на втория слой има във всеки ред само по една единица и всички останали елементи от реда са нули.

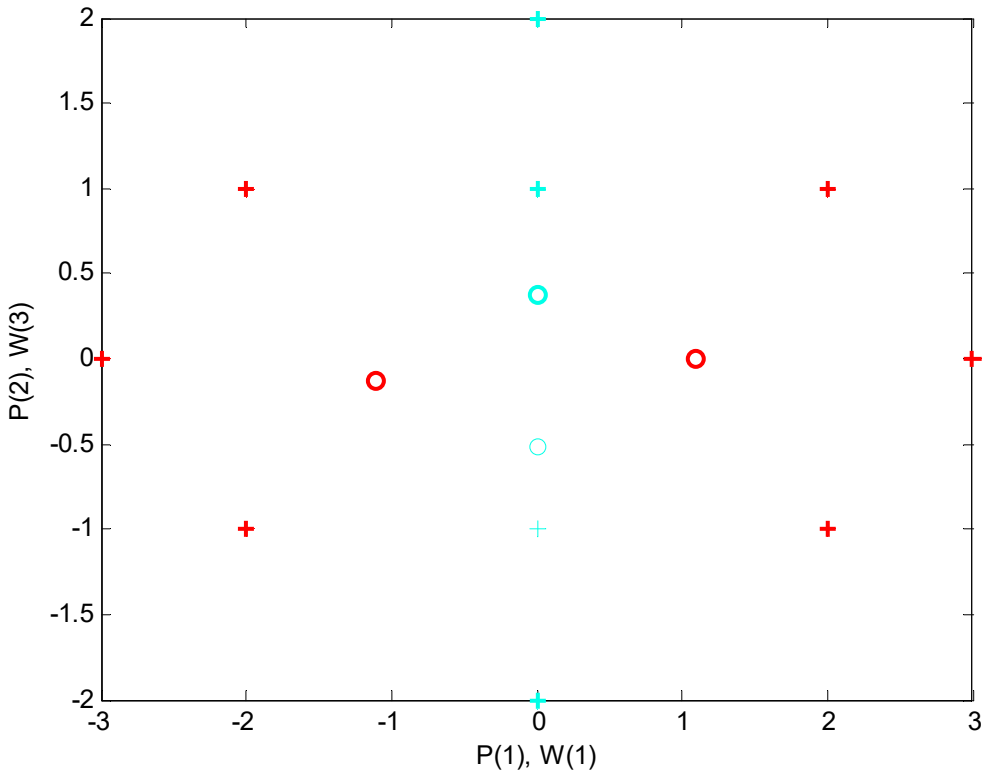
На пръв поглед добавянето на втория слой не би подобрило с нищо мрежата, защото както изходът от първия слой $\mathbf{u}$ така и изходът от втория слой $\mathbf{y}$, са вектори с координати една единица и останалите нули. Следователно със или без втория слой мрежата се използва за решаване на една и съща задача, т.е. задача за класификация Добавянето на втория слой обаче позволява примерно няколко класа от първия слой да се обединят в един клас от втория слой или изходните вектори на първия слой $\mathbf{u}$ да се разглеждат като подкласове на изходните вектори на втория слой $\mathbf{y}$.

Пример 2. В следващата програма се задават данните и класовете, към които тези данни принадлежат. Обучава се една мрежа с 4 неврона в първия слой. Симулира се работата на вече обучената мрежа, като се проверява към кой клас ще бъде причислен векторът $p = (0.2, 1)^T$.

```
clear all
%въвеждане на данни и класове, към които те принадлежат
cla
P = [-3 -2 -2  0  0  0  0 +2 +2 +3;  0 +1 -1 +2 +1 -1 -2 +1 -1
0];
C = [1 1 1 2 2 2 2 1 1 1];
T = ind2vec(C);
cla
colormap(hsv)
plotvec(P,C)
figure(gcf),set(gcf,'Color',[1 1 1]),set(gca,'Color',[1 1 1])
xlabel('P(1)');
ylabel('P(2)');
fprintf('Red = class 1, Cyan = class 2')
pause
%архитектура и инициализиране на мрежата
net = newlvq(minmax(P),4,[.6 .4],0.1);
cla
plotvec(P,C),hold on
W1=net.IW{1}
plot(W1(1,1),W1(1,2),'ow'),hold off,
figure(gcf),set(gcf,'Color',[1 1 1]),set(gca,'Color',[1 1 1])
xlabel('P(1), W(1)');
ylabel('P(2), W(3)');
pause
%трениране на мрежата
net.trainParam.epochs=150;
net.trainParam.show=Inf;
net=train(net,P,T);
cla
```

```
plotvec(P,C),hold on
plotvec(net.IW{1}',vec2ind(net.LW{2}),'o'),hold off,
figure(gcf),set(gcf,'Color',[1 1 1]),set(gca,'Color',[1 1 1])
pause
%симулиране на мрежата
P = [0.2; 1]
a = vec2ind(sim(net,P))
```

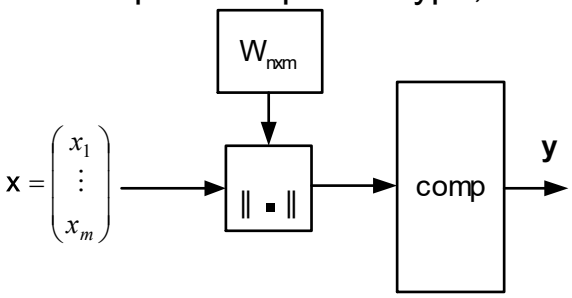
Въведените данни и резултатът от обучението са показани на фиг.2.16. На тази фигура с плюсове са изобразени данните, с които се обучава мрежата, а с кръгчета са изобразени теглата на вече обучените неврони (центровете на класовете). При това данните, принадлежащи към първия клас, са в тъмно, а към втория клас са в светло. След симулирането се получава резултат $a = 2$. Той показва, че векторът $p = (0, 0.2)^T$ е причислен към втория клас.



Фиг.2.16

2.3.3. Самоорганизиращи се изображения

Това са еднослойни мрежи с архитектура, показана на фиг.2.17.



фиг.2.17

На схемата 2.17 са зададени:

n - брой на невроните;

$\mathbf{W} \in R^{n \times m}$ - матрицата от тегловни вектори на слоя;

$\mathbf{x} \in R^{m}$ - входен вектор на мрежата;

$comp$ - активираща функция.

В структурата на тази мрежата също няма вектор от свободни тегла. Това позволява векторът $\mathbf{W_i}$, представляващ i-тия ред на тегловата матрица $\mathbf{W}$, да играе ролята на център на i-тия клас. От друга страна отсъствието на коригиращ коефициент b_i не позволява да се избегне със сигурност ситуацията "мъртви неврони", т.е. при неблагоприятно множество от входни вектори тази ситуация може и да се появи.

При тези невронни мрежи изходът $\mathbf{y} \in R^{n}$ на мрежата се получава по формулата

$$\mathbf{y} = compet(-\| \mathbf{W_1} - \mathbf{x} \|,...,-\| \mathbf{W_n} - \mathbf{x} \|),$$

където

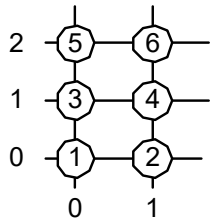
$\mathbf{W_i}$ - i-тият ред на матрицата $\mathbf{W}$;

$\| \mathbf{W_i} - \mathbf{x} \| = \sqrt{(\mathbf{W_{i,1}} - x_1)^2 + ... + (\mathbf{W_{i,m}} - x_m)^2}$.

Обучението е аналогично на обучението на мрежите от раздел 2.3.1. Тук за разлика от тях след определяне на неврона победител по правилото на Кохонен се обучава освен този неврон и съседните му. Освен това скоростите на обучение се избират различни за неврона победител и за съседните неврони. Тези разлики позволяват да се постигне по-добро настройване на мрежата.

За да се определи кои са съседните на даден неврон, се въвеждат топология на мрежата и разстояние между невроните. Чрез тях се намира множеството $N_i(d)$, състоящо се от невроните, намиращи се на разстояние по-малко от d от i-тия неврон.

В MATLAB се използват следните три топологии: правоъгълна, шестоъгълна и случайна (`gridtop`, `hextop` и `randtop`). Най-често използваната топология е правоъгълна (фиг.2.18) и затова тук се разглежда предимно тази топология.



фиг.2.18

В нея може да се използва всяко едно от следните разстояния: евклидово разстояние, разстояние, мерено с брой свързващи стъпки, манхатаново разстояние, правоъгълно разстояние (`dist`, `linkdist`, `mandist`, `boxdist`).

Евклидовото и манхатановото разстояния се задават така:

$\| \mathbf{x} - \mathbf{y} \|_2 = \sqrt{(x_1 - y_1)^2 + ... + (x_m - y_m)^2}$ - евклидово разстояние,

$\| \mathbf{x} - \mathbf{y} \|_1 = | x_1 - y_1 | + ... + | x_m - y_m |$ - манхатаново разстояние.

Останалите две разстояния ще бъдат изяснени със следния пример.

Разглежда се двумерната правоъгълна топология от фиг.2.18, която се получава с командата

```
pos=gridtop(2,3).
```

Резултат от действието на командата е

```
pos =     0     1     0     1     0     1
          0     0     1     1     2     2
```

Той показва, че първият неврон е разположен на позицията (0,0), вторият на позиция (1,0), третият на (0,1) и т.н. - фиг.2.18. В тази топология двете по-специфични разстояния са разстоянието, мерено с брой свързващи стъпки и правоъгълното разстояние. Разстоянието мерено с брой свързващи-стъпки се определя с командата

```
linkdist(pos)
```

Резултатът от тази команда е

```
ans =
     0     1     1     2     2     3
     1     0     2     1     3     2
     1     2     0     1     1     2
     2     1     1     0     2     1
     2     3     1     2     0     1
     3     2     2     1     1     0.
```

В първия стълб от получените резултати е разстоянието от първия неврон до всички останали. Разстоянието от първия неврон до втория и третия е 1, а разстоянието до четвъртия и петия е 2 и разстоянието до шестия е 3. Получените разстояния 1,2 и 3 са минималният брой стъпки, необходими да се стигне от първия неврон до останалите неврони (фиг.2.18). Полученият резултат показва, че първият неврон има три зони на близост, а именно: в първата зона на близост са втори и трети неврон, във втората зона на близост са четвърти и пети неврон и в третата зона на близост е шести неврон (фиг.2.18).

Пресмятането на правоъгълното разстояние се прави с командата

```
boxdist(pos)
```

Резултатът от работата на тази команда е

```
ans =
     0     1     1     1     2     2
     1     0     1     1     2     2
     1     1     0     1     1     1
     1     1     1     0     1     1
     2     2     1     1     0     1
     2     2     1     1     1     0.
```

Първият стълб от получените резултати показва, че разстоянието от първия неврон до втория, третия и четвъртия е 1, а до петия и шестия е 2. Следователно първият неврон спрямо това разстояние има само две зони на близост, а именно: в първата зона на близост са 2-ри, 3-ти и 4-ти неврон, а във втората зона на близост са 5-ти и 6-ти неврон.

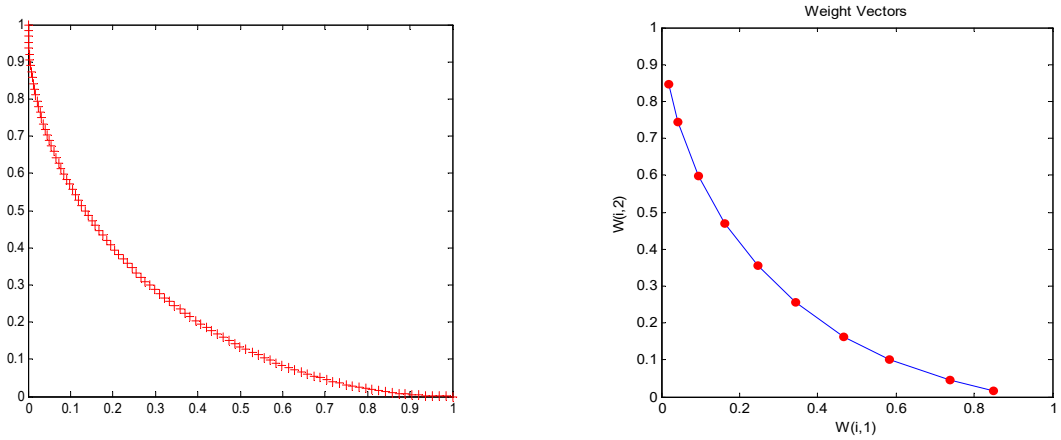
Тези два примера показват, че един неврон може да попадне в различни зони на близост в зависимост от използваното разстояние.

Последната особеност на тези мрежи е наличието на два етапа на обучение. Първо се определя невронът победител и се започва обучение на този неврон и на неговите съседни, като се стартира с най-голямата скорост на обучение за победителя и половината от тази скорост за съседните неврони. След което скоростта на обучение се намалява и зоната на близост (на неврона победител) се свива по подходящ алгоритъм.

Пример 3. Със следващата програма се генерират 100 равноотдалечени точки, които лежат на частта от единичната окръжност, разположена в първи квадрант. Обучава се една самоорганизираща се мрежа от 10 неврона, следователно данните ще се групират в 10 класа. Симулира се работата на вече обучената мрежа, като се проверява към кой клас ще бъде причислен векторът $p = (0, 0.8)^T$.

```
clear all
%генериране на данни
angles = pi:0.5*pi/99:1.5*pi;
p = [1+sin(angles);1+cos(angles)];
plot(p(1,:),p(2,:),'+r'),xlim([0 1]),ylim([0 1]),figure(gcf)
pause
%архитектура на мрежата
net=newsom([0 1;0 1],[10]);
%трениране на мрежата
net.trainParam.epochs=10;
net.trainParam.show=5;
net=train(net,p);
plotsom(net.iw{1,1},net.layers{1}.distances),xlim([0 1]),ylim([0
1]),figure(gcf)
pause
%симулиране на мрежата
P = [0; 0.8]
a=sim(net,P)
```

Генерираните данни и резултатът от обучението са показани на фиг.2.19.



фиг.2.19

На фиг.2.19 отляво с плюсове са изобразени 100 данни, с които се обучава мрежата, а отдясно с кръгчета са изобразени теглата на вече обучените 10 неврона. Това са и центровете на 10 класа, на които се разбиват 100-те входни данни.

След симулирането е получен резултатът a=(10,1) 1, т.е. десетата координата на a е единица, а останалите координати на a са нули. Това показва, че векторът $p=(0,0.8)^T$ е причислен към десетия клас.

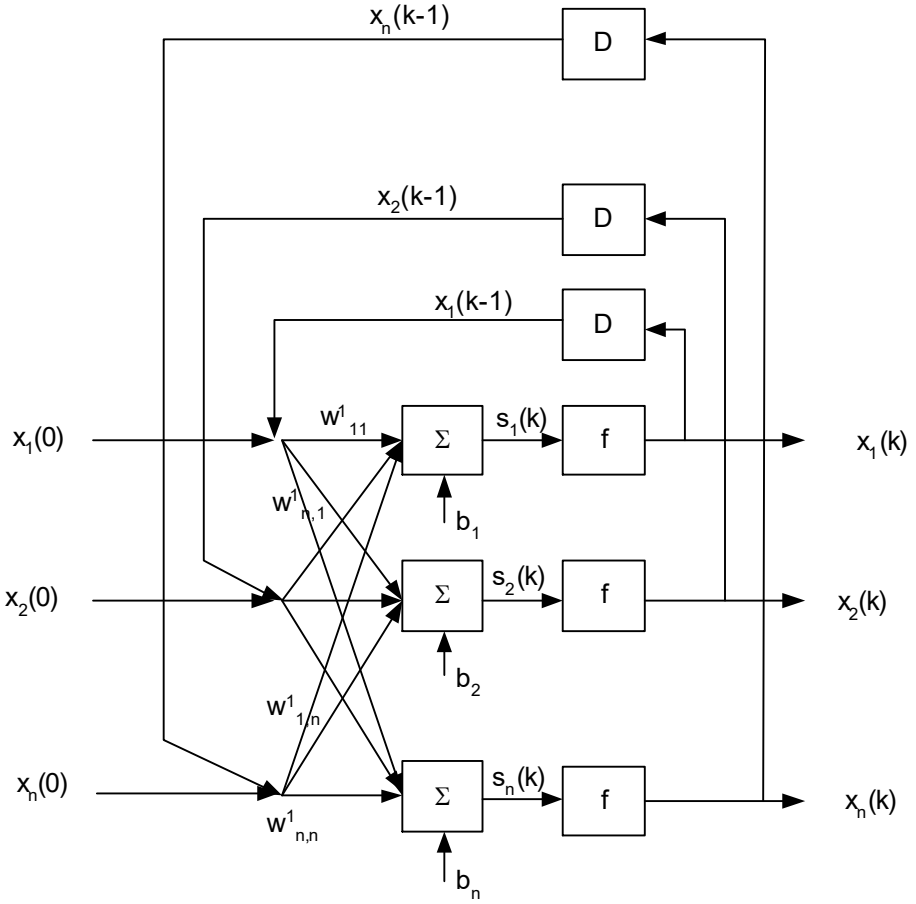
2.4. Мрежи на Хопфилд

При този тип рекурентни мрежи целта е да се запомни някакво множество от устойчиви равновесни точки. Това означава, че при задаване на съответно начално условие изходите на мрежата се установяват в някоя от равновесните точки.

Реализираният алгоритъм е свързан със система от линейни ОДУ от първи ред, за която променливите на състоянията се изменят в затворен хиперкуб. При това равновесните точки на тази система са на границата на хиперкуба.

Устойчивите равновесни точки могат да се третират като обекти, които подлежат на запомняне. Реализираният чрез тази мрежа процес на запомняне наподобява процеса на асоциативната човешка памет. Това сходство дава основание да се твърди, че мрежата на Хопфилд е аналог на асоциативната памет.

Мрежата на Хопфилд е еднослойна рекурентна и нейната структура е показана на фиг. 2.20.



фиг.2.20

Уравненията, описващи мрежата са следните:

$$\begin{pmatrix} s_1(k) \\ s_2(k) \\ \dots \\ s_n(k) \end{pmatrix} = \begin{pmatrix} w_{11}\ w_{12} \dots w_{1n} \\ w_{21}\ w_{22}\ \dots w_{2n} \\ \dots\dots\dots\dots\dots \\ w_{n1}\ w_{n2}\ \dots w_{nn} \end{pmatrix} \begin{pmatrix} x_1(k-1) \\ x_2(k-1) \\ \dots\dots \\ x_n(k-1) \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix} \quad \text{и} \quad \begin{pmatrix} x_1(k) \\ x_2(k) \\ \dots\dots \\ x_n(k) \end{pmatrix} = \begin{pmatrix} f(s_1(k)) \\ f(s_2(k)) \\ \dots\dots\dots \\ f(s_n(k)) \end{pmatrix}$$

или в матрична форма

(11) $\mathbf{x}(k) = \mathbf{f}(\mathbf{W}\,\mathbf{x}(k-1) + \mathbf{b})$, за $k = 1,2,3,\dots$,

където $\mathbf{x}_{n\times 1}$, $\mathbf{W}_{n\times n}$ и $\mathbf{b}_{n\times 1}$, $\mathbf{f}(s_1,\dots,s_n) = (f(s_1),\dots,y(s_n))^T$, а активиращата функция на мрежата е линейна с насищане:

$$f(x) = \begin{cases} -1, \text{ за } x \in (-\infty,-1) \\ x, \text{ за } x \in [-1,1] \\ 1, \text{ за } x \in (1,+\infty) \end{cases} .$$

Налага се ограничението устойчивите равновесни точки на мрежата да са по границата на единичен хиперкуб. За тази хопфилдова мрежа теглата се пресмятат по метода на Li [8]. При този метод предварително се задават устойчивите равновесни точки $\bar{\mathbf{x}}$ и се търсят теглата на мрежата $\mathbf{W}$ и $\mathbf{b}$.

Областта на привличане на дадена устойчива равновесна точката $\bar{\mathbf{x}}$ има следния смисъл. Ако началната точка е от тази околност, то след определен брой стъпки изходът на мрежата ще се установи в равновесната точка $\bar{\mathbf{x}}$.

За да се намерят теглата $\mathbf{W}$ и $\mathbf{b}$, се решава следната задача. Определя се такава линейна система обикновени диференциални уравнения

(12) $$\dot{\mathbf{x}}(t) = \mathbf{T}\,\mathbf{x}(t) + \mathbf{I},$$

чиито устойчиви равновесни точки са точно точките $\bar{\mathbf{x}}$. Това означава да се намерят числовите матрици $\mathbf{T}$ и $\mathbf{I}$, участващи в системата (12).

Под равновесна точка на системата (12) се разбира точка $\bar{\mathbf{x}}$, за която решението на (12) е константният вектор

$$\mathbf{x}(t) = \bar{\mathbf{x}}, \text{ за } t \in (t_0,+\infty),$$

при начално условие

(13) $$\mathbf{x}(t_0) = \bar{\mathbf{x}}.$$

Една равновесна точка на (12) е устойчива, ако малки отклонения от началното условие (13) водят след известно време t_1 $(t_1 > t_0)$ до установяване на решението в същата точка.

Следният пример илюстрира този алгоритъм. Търси се система от вида (12) с три равновесни точки:

$$\mathbf{x_1} = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}, \quad \mathbf{x_2} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, \quad \mathbf{x_3} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Пресмятат се:

$$\mathbf{y_1} = \mathbf{x_1} - \mathbf{x_3} = \begin{pmatrix} -2 \\ 0 \\ 0 \end{pmatrix}, \qquad \mathbf{y_2} = \mathbf{x_2} - \mathbf{x_3} = \begin{pmatrix} 0 \\ -2 \\ 0 \end{pmatrix}.$$

Съставя се матрицата:

$$\mathbf{Y} = [\mathbf{y_1}\ \mathbf{y_2}] = \begin{pmatrix} -2 & 0 \\ 0 & -2 \\ 0 & 0 \end{pmatrix}$$

и се намира сингулярното й разлагане. То се получава със стандартна програма в MATLAB и е следното:

$$\mathbf{Y} = [\mathbf{U},\mathbf{S},\mathbf{V}] = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 2 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

където диагоналната матрица $\mathbf{S}$ съдържа сингулярните стойности $\sigma_1 = \sigma_2 = 2$ на $\mathbf{Y}$.

Тъй като броят на ненулевите сингулярни стойности е две, то помощната матрица $\mathbf{T}^+$ има две събираеми от вида $\mathbf{u}_i \mathbf{u}_i^T$, а $\mathbf{T}^-$ има само едно събираемо от този вид. Тук $\mathbf{u}_i$ са стълбовете на матрицата $\mathbf{U}$, получена при сингулярното разлагане на $\mathbf{Y}$:

$$\mathbf{T}^+ = \mathbf{u}_1 \mathbf{u}_1^T + \mathbf{u}_2 \mathbf{u}_2^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{и} \quad \mathbf{T}^- = \mathbf{u}_3 \mathbf{u}_3^T = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Търсените матрици $\mathbf{T}$ и $\mathbf{I}$ от (12) се пресмятат по формулите

$$\mathbf{T} = \mathbf{T}^+ - 10\mathbf{T}^- = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -10 \end{pmatrix} \quad \text{и} \quad \mathbf{I} = \mathbf{x_3} - \mathbf{T}\mathbf{x_3} = \begin{pmatrix} 0 \\ 0 \\ 11 \end{pmatrix}.$$

След като се намерят матриците $\mathbf{T}$ и $\mathbf{I}$, участващи в (12), остава да се пресметнат тегловните матрици на мрежата $\mathbf{W}$ и $\mathbf{b}$. За целта

първо решението на системата (12) се дискретизира, като се използва формулата на Коши

(14) $$\mathbf{x}(t) = e^{\mathbf{T}(t-t_0)}\mathbf{x}(t_0) + \int_{t_0}^{t} e^{\mathbf{T}(t-\tau)}\mathbf{I}\,d\tau\,,$$

в която се полага

$$t_0 = (k-1)h\,,\ t = kh\,,\ \mathbf{x}((k-1)h) = \mathbf{x_{k-1}} \text{ и } \mathbf{x}(kh) = \mathbf{x_k}\,.$$

Тук h е стъпката по времето. Пресмята се интегралът, участващ в (14) и се получава формулата

(15) $$\mathbf{x_k} = e^{\mathbf{T}h}\mathbf{x_{k-1}} + \mathbf{T}^{-1}(-\mathbf{E} + e^{\mathbf{T}h})\mathbf{I}\,,$$

където $\mathbf{E}$ е единична матрица. След това се сравняват формули (11) и (15) и при $h = 0.15$ за теглата на мрежата се получава:

$$\mathbf{W} = e^{\mathbf{T}h} = \begin{pmatrix} 1.1618 & 0 & 0 \\ 0 & 1.1618 & 0 \\ 0 & 0 & 0.2231 \end{pmatrix}$$

$$\mathbf{b} = \mathbf{T}^{-1}(-\mathbf{E} + e^{\mathbf{T}h})\mathbf{I} = \begin{pmatrix} 0 \\ 0 \\ 0.8546 \end{pmatrix}.$$

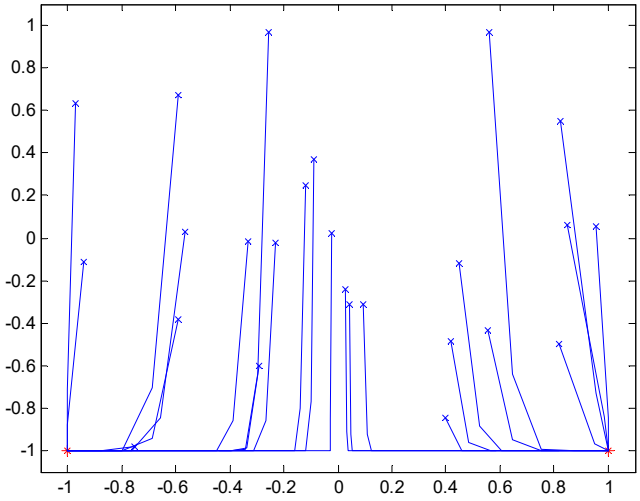
Този метод има и следните два съществени недостатъка. Първо, не всяко множество от точки на границата на хиперкуба може да се заложи като устойчиви равновесни точки на системата (12). Второ, дори едно множество от точки да се заложи като устойчиви равновесни точки на (12), то може да се окаже, че системата (12) има и други равновесни точки (наречени лъжливи).

Пример 1. Следващата програма първо синтезира мрежа на Хопфилд с две равновесни точки, запомнени във вектора $T = [-1\,1;1\,-1]$ (за целта се използва командата $newhop(T)$). След това се изследват областите на привличане на равновесните точки, като се генерират 25 случайни точки от вътрешността на квадрата $[-1\,1]\times[-1\,1]$ и се проследяват техните траектории до привличането им в една от двете устойчиви равновесни точки.

```
clear all
cla,
T = [-1  1; -1 -1];
figure(1),
plot(T(1,:),T(2,:),'r*'),
axis([-1.1 1.1 -1.1 1.1]),
pause,
```

```
net = newhop(T);
for i=1:25
   a = {rands(2,1)};
   [y,Pf,Af] = sim(net,{1 20},{},a);
   record=[cell2mat(a) cell2mat(y)];
   start=cell2mat(a);
   hold on,
   plot(start(1,1),start(2,1),'bx',record(1,:),record(2,:)),
end
```

Резултатът от работата на програмата е даден на фиг.2.21.



фиг.2.21

Приложение
1. Програмиране в среда MATLAB

Потребителят може да работи в средата MATLAB в диалогов режим, като записва командите в командния ред на прозореца на програмата и ги стартира с клавиша Enter. Другата възможност е, като в текстовия редактор се състави програма на езика за програмиране в MATLAB и тази програма може да се стартира от командния ред. Първоначално ще бъдат разгледани някои основни правила за работа в MATLAB, а след това основните оператори и функции за програмиране в средата MATLAB.

1.1. Работа в средата MATLAB

- В средата MATLAB командите се въвеждат в прозореца на програмата от командния ред съгласно възприетия от версията на програмата синтаксис;
- Когато не е известен синтаксисът на даден символ или функция, може да се обърнете за помощ, като напишете в командния ред: **help, help символ или help име на функция**. След като натиснете клавиша Enter, на екрана се появява кратко описание и евентуално примери;
- Когато след командата се постави символът ***;*** резултатът от командата не се показва на екрана, а само се запазва в паметта.
- Нормално резултатът се представя с 6 значещи цифри, но има възможност да се представи и с 12, като се използва командата **format long** (за подробности вижте описанието чрез **help format**).
- Когато е необходимо вече използвани команди да се въведат отново, те лесно могат да бъдат открити чрез използване на клавишите ↑ и ↓.

1.2. Матрици в MATLAB

Най-често използваните обекти са матриците (векторите и числата са частен случай на матрици).

Вектори

Един вектор може да бъде въведен по следния начин:

```
V=[2 3 4 5 6 6 5 4 3 2 1 0]
```

Важното е, че векторът `Vs` съдържа синусите на елементите на вектора V:

```
Vs=sin(V)
```

т.е. функциите са векторизирани.

Матрици

Една матрица може да бъде въведена по следния начин:

```
mat=[1 3 5 7; 9 8 2 3; 4 7 2 3]
```

и резултатът е

```
mat=  1  3  5  7
      9  8  2  3
      4  7  2  3
```

Използват се някои запазени символи. Примерно числото π се записва с ***pi***, а имагинерната единица с ***i*** или ***j***.

Образуване на масив от данни с постоянна стъпка

Такъв масив може да бъде създаден с командния ред

```
vect1=1:2:9
```

и резултатът е

```
vec1=  1  3  5  7  9
```

В средата MATLAB са създадени следните специални матрици

zeros(m,n)	създава матрица m x n с нули (нулева матрица)
ones(m,n)	създава матрица m x n с единици
eye(n)	създава единична матрица nxn
rand(m,n)	създава матрица m x n с елементи – случайни числа в интервала (0,1)
randn(m,n)	матрицата има елементи с нормално разпределение

Действия с матрици

+	Събиране на матрици от един и същи тип
-	Изваждане на матрици от един и същи тип
*	Умножение на матрици
/	Деление на скалари
.*	Поелементно умножение на две матрици
./	Поелементно деление на две матрици
^	Повдигане на степен
.^	Поелементно повдигане на степен
‘	Транспониране на матрица
\	Ляво деление (A\B = A^{-1}*B)
/	Дясно деление (B/A = B*A^{-1})
inv(A)	Обръщане на матрица
det(A)	Изчисляване на детерминантата на матрицата A

Действията поелементно умножение(.*), деление(./) и степенуване(.^) ще бъдат илюстрирани със следните примери:

```
mat1=[1 2 3]
mat2=[2 3 4]
mat3=mat1.*mat2
```

и резултатът е

```
mat3=
   2   6   12
```

```
mat4=mat1.^2
```

и резултатът е

```
mat4=
  1   4   9
```

Функции от матрици

pinv(A)	Псевдообратна матрица
eig(A)	Собствени стойности на матрица
poly(A)	Дава коефициентите на характеристичния полином на матрицата A

1.3. Решаване на линейни системи

В MATLAB се използват два символа за решаване на линейни системи: *slas*h / и *backslas*h \.

X = A\B означава решаването на матричното равенство AX = B.

X = B/A означава решаването на матричното равенство XA = B.

При това :

а) ако броят на уравненията е равен на броя на неизвестните, търси се единствено или базисно решение;

б) ако броят на уравненията е по-голям от броя на неизвестните (преопределена система), определя се решение по метода на най-малките квадрати;

в) ако броят на уравненията е по-малък от броя на неизвестните (недоопределена система), търси се базисно решение на системата.

1.4. Някои оператори в MATLAB

Оператори за релации

< - по- малко

<= - по- малко и равно

> - по голямо

>= - по- голямо и равно

== - равно

~= - не равно

Логически оператори

& -AND
| -OR
~ -NOT

Оператор за цикъл if

Операторът **if** изчислява логически израз и изпълнява група от оператори в зависимост от стойността на израза.

Основният синтаксис на този оператор е:

if логически израз
оператори
end

Ако стойността на израза е **true** , MATLAB изпълнява всички оператори между **if** и **end**. След това се изпълняват и операторите след **end.** Ако изразът има стойност **false**, MATLAB не изпълнява операторите между **if** и **end** и следва директно изпълняване на операторите след **end.**

Операторите **else** и **elseif** разширяват възможностите на оператора **if**. В такъв случай синтаксисът на оператора **if** е:

if логически израз
оператори
else
оператори
end

Операторът **else** няма логическо условие. Операторите след него се изпълняват , ако в предшестващия **if** условието е **false** .

В синтаксисът на оператора **if** може да се включи и операторът **elseif**, които има логическо условие, примерно

if логически израз
оператори
elseif
оператори
else
оператори
end

Операторът **elseif** има логическо условие, което се изчислява ,ако условието в предшестващия **if** е **false**. Операторите след него се изпълняват, ако логическото условие на **elseif** е **true**.

Оператор за цикъл for

Операторът **for** осигурява многократно изпълняване на група от оператори, образуващи тялото на цикъла. Неговата структура е следната:

for *index* **= *старт:нарастване:крайна стойност***
оператори
end

Подразбиращото се нарастване е 1. Може да се зададе всякакво нарастване, включително и отрицателно.

Примери:

```
for i = 1:6
x(i) = 2*i;
end
```

резултатът е

```
x =
     2     4     6     8    10    12
```

Може да се програмира и цикъл в цикъл.

Оператор while

Операторът **while** изпълнява многократно група от оператори докато условието е **true**. Синтаксисът му е следният:

while условие
операторите
end

Например този while цикъл определя първите n цели числа, за които n! е по-малък от 100.

```
n = 1;
while prod(1:n) < 100
n = n + 1;
end
```

Оператор break

Операторът **break** прекъсва изпълняването на **for** или **while** цикъл. След изпълняването на **break** се преминава към първия оператор извън цикъла.

1.5. Функции в MATLAB

Най-често функцията е М-файл (текстов файл с разширение **.m**), който има входни аргументи и връща изходни аргументи.

В средата на MATLAB всички функции се активират с инструкцията:

[res1, res2, res3,…]=име на функция(arg1, arg2, arg3,…),
където res1, res2, res3,… са резултатите от прилагането на функцията, а arg1, arg2, arg3,… са аргументите на функцията.

М - файлът може да съдържа повече от една функция (за версиите след **MATLAB 6.0)** и да представлява самостоятелна програма. Първата функция във файла е главната функция и има името на файла, а другите са подпрограми-функции, които се извикват само от главната функция или от други подпрограми в същия файл.

Функция за решаване на нелинейно уравнение

Пример. Да се реши нелинейното уравнение

$$x - 0.5\sin(x) - 0.2 = 0$$

с начално приближение на корена x0=0.1.

```
% Главна програма
function main
clear all
x0=0.1;
koren=fzero(@fun,x0)
% Подпрограма
function f=fun(x)
f=x-0.5*sin(x)-0.2;
```

Резултат от действието на програмата

```
koren =
    0.3902
```

Това е най-близкият до 0.1 корен на уравнението $x - 0.5\sin(x) - 0.2 = 0$.

Програмата в MATLAB може да има и друга структура – главната програма и подпрограмите са зададени в отделни М-файлове. В този случай функцията fun не се извиква с @fun, а името ѝ трябва да е зададено като низ ‘fun’.

Функция за решаване на система диференциални уравнения

Пример. Да се реши системата диференциални уравнения:

$$\dot{y}_1(t) = y_2(t)$$

$$\dot{y}_2(t) = -y_1(t)$$

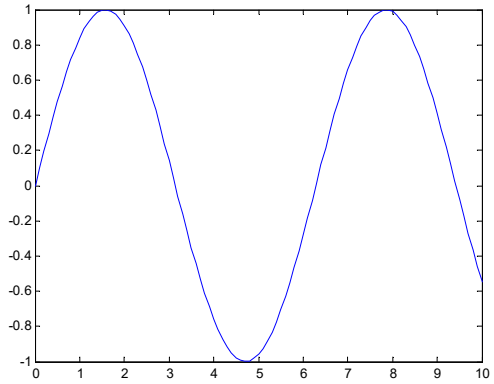
в интервала $t \in [0\,10]$ при началното условие $y_1(0) = 0;\ y_2(0) = 1$.

```
% Главна програма
function main
clear all
 [t,y]=ode45(@func,[0:0.1:10],[0  1]);
plot(t,y(:,1),'r'),figure(gcf)
% Подпрограма, в която се описва дясната част на системата
function df=func(t,x)
df(1)=x(2);
df(2)=-x(1);
```

```
df=df';
```

В процедурата ode45 векторът t=0:0.1:10 дава моментите от време, в които ще се пресмятат решенията $y_1(t)$ и $y_2(t)$, а векторът [0 1] о съдържа началните условия, при които се решава системата, т.е $y_1(0)=0$ и $y_2(0)=1$.

С командата plot се чертае решението $y_1(t)$ за $t\in[0\,10]$. Резултатът от работа на програмата е показан на фиг.1.



фиг.1

Функция за условна оптимизация

Процедурата fmincon се стартира с реда

x=fmincon('funopth', v0, [], [], [], [], vlb, vub, [], options),

където

- funopth е функция на оптимизирания;
- v0 е начално приближение за оптимизационните параметри;
- vlb и vub задават долните и горните граници на оптимизираните параметри;
- options задава параметри на процедурата ;
- в празните полета, означени с [ ], могат да се задават други параметри на процедурата, които са описани в help.

Пример. Да се намери минимумът на функцията $z(x,y)=x^2+y^2+1$, за $-1\le x\le 1$ и $-1\le y\le 1$. Програмата на MATLAB, с която се решава тази задача, е следната:

```
% Главна програма
function main
vub=[1;1];
vlb=[-1;-1];
v0=[1;-1];
options=optimset('Display','iter','MaxIter',100);
[v_end,z_min]=fmincon(@fun,v0,[],[],[],[],vlb,vub,[],options)
```

```
% Подпрограма
function z=fun(v)
z=v(1)^2+v(2)^2+1;
```

Резултатът от работата на програмата е

```
v_end =
  1.0e-007 *
   -0.2723
   -0.1718
z_min =
    1.0000
```

Това е резултат много близък до очевидния минимум $z_{\min} = z(x = 0, y = 0) = 1$.

2. Основни функции в подсистема “Невронни мрежи” в MATLAB

2.1.Функции за описване на архитектурата на невронна мрежа

Функцията newp

С тази функция се създава персептронен слой. Пример

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

С горната команда се създава един персептронен слой с един вход и два изхода. При това входните данни ще попадат в интервала $[-1,1]$.

Функцията newlin

С тази функция се създава линеен слой. Пример

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

С горната команда се създава един линеен слой с два входа и два изхода. При това данните, подавани и към двата входа, ще попадат в интервала $[-1,1]$.

Функцията newlind

С тази функция се създава и едновременно с това се обучава един линеен слой, като обучението става по метода на линейните най-малки квадрати. Примерно

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

където в масивите P и T се задават съответно входните данни и еталонът. Размерността на мрежата се определя автоматично, като входовете и изходите на мрежата са равни съответно на броя на редовете на масивите P и T.

Функцията newff

С тази функция се създава еднослойна или многослойна нелинейна невронна мрежа. Примерно

```
net=newff([-1 1;-1 1],[2,1],{'logsig','purelin'});
```

С тази команда се създава една двуслойна невронна мрежа с два неврона в първия слой и един във втория. При това активиращите функции на първия слой са `logsig` , а на втория са `purelin`. Входните данни на мрежата трябва да се изменят в интервала $[-1,1]$.

Функцията newrb

С тази функция се създава и едновременно с това се обучава една мрежа с радиални базисни функции. Тези мрежи имат характерна структура. Първият слой е с радиални базисни активиращи функции, а вторият е линеен. Примерно

```
net=newrb(P,T,spread);
```

Масивите P и T се съответно масивът от входните данни и масивът на еталона. Броят на невроните в първия и втория слой са съответно равни на броя на редовете на масивите P и T. Променливата `spread` е скалар, задаващ параметъра “ширина” на радиалните базисни функции от първия слой.

Функцията newpnn

Тази функция генерира и обучава вероятностни мрежи. Тя работи аналогично на функцията newrb и има следния синтаксис:

```
net = newpnn(P,T,spread);
```

Разликата между мрежите с радиални базисни функции и вероятностните мрежи е само в това, че вторият слой на мрежите с радиални базисни функции е линеен, а на вероятностните е compet.

Функцията newc

С тази функция се създава еднослойна мрежа с активираща функция compet. Примерно

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

С горната команда се създава един слой с два входа и два изхода. При това данните, подавани и към двата входа, трябва да са в интервала $[0,1]$.

Функцията newsom

С тази функция се създава еднослойна самоорганизираща се мрежа. Примерно

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

С горната команда се създава мрежа, състояща се от десет неврона и имаща два входа. Данните, подавани и към двата входа, трябва да попадат в интервала $[0,1]$.

Функцията newlvq

С тази функция се създава двуслойна мрежа с първи слой compet и втори линеен. Примерно

```
net = newlvq(minmax(P),4,[.6 .4]);
```

С горната команда се създава една мрежа с четири неврона в първия слой. Броят на входовете на мрежата е равен на броя на редовете на масива P. При това данните, подавани към входовете, ще се изменят в интервала между най-малката и най-голямата данна от съответния ред на P. Означението [.6 .4] показва, че изходите на мрежата са два, като в 60% от случаите входовете ще се причисляват към първия клас, а в 40% от случаите входовете ще се причисляват към втория клас.

2.2.Функции за обучение на невронна мрежа

Обучението на невронните мрежи може да се извършва с помощта както на крайни алгоритми, така и на итерационни процедури. При обучението с крайни алгоритми, примерно алгоритъма на линейните най-малки квадрати, описанието на архитектурата на мрежата и обучението на мрежата се осъществяват с една команда (в случая newlind). Обучението с итерационни процедури се прави с една от двете команди adapt или train. В повечето случаи една мрежа може да се обучава както с adapt, така и с train. Обучение с adapt става с последователно обработване на данните, а при обучението с train данните се обработват пакетно.

Функцията adapt

Пример за обучение на една вече създадена мрежа с име net с командата adapt:

```
net=adapt(net,P,T);
```

С тази команда се обучава мрежата net по вход P и еталонен изход T. Важна особеност на масивите P и T е, че те трябва да са в формат клетъчен масив (cell array). Примери за клетъчни масиви са:

```
P={1,2,3,4}
T={5,6,7,8};
```

Функцията train

Пример за обучение на една вече създадена мрежа с име net с командата `train` :

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

С тази команда се обучава мрежата net по вход P и еталонен изход T. Важна особеност на масивите P и T е, че те трябва да са матрици. Примери за матрици:

```
P=[1,2,3,4]
T=[5,6,7,8];
```

2.3.Функции за симулиране на невронна мрежа

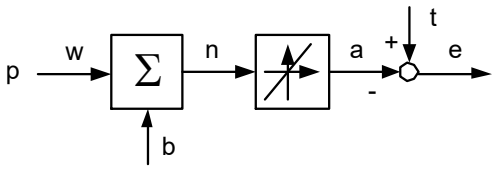
Една вече обучена невронна мрежи може да се симулира с функцията **sim.** Пример за симулиране, с една вече създадена и обучена мрежа net, с командата sim:

```
a=sim(net,P);
```

С тази команда се получава изходът на вече обучената мрежа net, при условие че на входа се подаде масивът P. Важна особеност на масивите P и a е, че те са клетъчни масиви при обучение с adapt или са матрици при обучение с train.

2.4. Примери за обработване на данните последователно и пакетно .

Нека е даден линеен неврон с един вход p и един изход a.



фиг.2

Този неврон ще се обучава с входни данни $P=(1,2,3,4,5)$ и съответстващите им еталонни изходи $T=(5,8,4,3,3)$. За определяне на свободните параметри w и b могат да се използват градиентният метод с обратно разпространяване на грешките или градиентната процедура на Уидроу-Хоф. При използване на първия метод се съставя средноквадратична грешка между всички реализирани от мрежата изходи и еталонните изходи, т.е.

$$S=\frac{1}{2}\sum_{k=1}^{5}e^2(k)=\frac{1}{2}\sum_{k=1}^{5}(t(k)-a(k))^2=\frac{1}{2}\sum_{k=1}^{5}(t(k)-w\,p(k)-b)^2\ .$$

За да се минимизира се определя градиентът на сумарната грешка относно теглата w и b, т.е. $gradS=\left(\frac{\partial S}{\partial w},\frac{\partial S}{\partial b}\right)$. За целта се пресмятат производните:

$$\frac{\partial S}{\partial w}=\sum_{k=1}^{5}e(k)\frac{\partial e(k)}{\partial w}=-\sum_{k=1}^{5}e(k)p(k)=-\mathbf{e}^T\mathbf{p}$$

$$\frac{\partial S}{\partial b}=\sum_{k=1}^{5}e(k)\frac{\partial e(k)}{\partial b}=-\sum_{k=1}^{5}e(k)=-\mathbf{e}^T\mathbf{1},$$

където $\mathbf{e}=\big(e(1),...,e(5)\big)^T$, $\mathbf{p}=\big(p(1),...,p(5)\big)^T$, $\mathbf{1}=(11111)^T$.

В такъв случай неизвестните w и b се коригират по направление на антиградиента, т.е. по формулите:

$$w^{n+1} = w^n + \alpha\, \mathbf{e}^{\mathbf{T}}\mathbf{p}$$

$$b^{n+1} = b^n + \alpha\, \mathbf{e}^T \mathbf{1},$$

където n е номерът на итерацията, а α е специфичен параметър на алгоритъма, наречен скорост на обучение. При този метод на обучение за всяка итерация на теглата w и b се използват всички входни данни $p(k)$ за $k = 1,...,5$ и всички еталонни изходи $t(k)$ за $k = 1,...,5$. Казва се, че това обучение е с обработване на данните в пакет (batch training). Едно такова обучение се реализира с командите

```
net=newff([0, 5],1,{'purelin'});
P=[1,2,3,4,5];
T=[5,8,4,3,3];
net=train(net,P,T);
```

При използване на градиентна процедура на Уидроу-Хоф се намират производните на грешката в момента k

$$s(k) = \frac{1}{2}e^2(k) = \frac{1}{2}(t(k) - a(k))^2$$

спрямо теглата w и b:

$$\frac{\partial s(k)}{\partial w} = e(k)\frac{\partial e(k)}{\partial w} = -e(k)\,p(k)$$

$$\frac{\partial s(k)}{\partial b} = e(k)\frac{\partial e(k)}{\partial b} = -e(k)$$

и неизвестните w и b се коригират по направление на антиградиента, т.е. по формулите:

$$w(k+1) = w(k) + \alpha\, e(k)\,p(k)$$

$$b(k+1) = b(k) + \alpha\, e(k),$$

където k е номерът на времевата стъпка, а α е специфичен параметър на алгоритъма, наречен скорост на обучение.

При този метод на обучение очевидно за пресмятане на изменението на неизвестните w и b на всяка времева стъпка k се използват само входът $p(k)$ и еталонният изход $t(k)$ от същата времева стъпка. Затова обучението е с последователно обработване на данните (incremental training). Едно такова обучение се реализира с командите:

```
net = newlin([0,5],1,0,0.1);
P={1,2,3,4,5};
T={5,8,4,3,3};
net=adapt(net,P,T);
```

2.5. Структура на невронна мрежа

Пример. Разглежда се невронна мрежа, създадена с командите:

```
net=newff([0,5;0,5],[2 1],{'tansig' 'purelin'});
P=[1,2,3,4,5;1,2,3,4,5];
T=[5,10,15,20,25];
net=train(net,P,T);
```

Това е една двуслойна мрежа с два неврона в първия и един неврон във втория слой. Активиращите функции на първия слой са сигмоидални, а вторият слой е линеен. Тази мрежа има два входа. Данните, които се подават към тези входове, трябва да са в интервала [0, 5]. Мрежата се обучава с метода обратно разпространяване на грешките, по вход P и еталонен изход T. След като мрежата net е вече обучена, то структурата й се запомня в структурен масив. Това е обект, който се състои от множество полета и подполета. За да се види пълната структура на мрежата net, се използва командата

```
struct(net)
```

и резултатът от работата на командата е

```
ans =
         numInputs: 1
         numLayers: 2
    numInputDelays: 0
    numLayerDelays: 0
       biasConnect: [2x1 double]
      inputConnect: [2x1 double]
      layerConnect: [2x2 logical]
     outputConnect: [0 1]
     targetConnect: [0 1]
        numOutputs: 1
        numTargets: 1
            inputs: {[1x1 struct]}
            layers: {2x1 cell}
            biases: {2x1 cell}
      inputWeights: {2x1 cell}
      layerWeights: {2x2 cell}
           outputs: {[]  [1x1 struct]}
           targets: {[]  [1x1 struct]}
          adaptFcn: 'trains'
        adaptParam: [1x1 struct]
           initFcn: 'initlay'
         initParam: []
        performFcn: 'mse'
      performParam: []
           trainFcn: 'trainlm'
        trainParam: [1x1 struct]
                IW: {2x1 cell}
```

```
            LW: {2x2 cell}
             b: {2x1 cell}
      userdata: [1x1 struct]
          hint: [1x1 struct]
        revert: [1x1 struct]train(net,P,T);
```

Този резултат показва, че структурният масив net има 32 полета. За да се види съдържанието примерно на първото поле, достатъчно е да се запише в командния ред

```
net.numInputs
```

и на екрана ще се изведе съдържанието на това поле и то е

```
ans =
     1.
```

Променяне на съдържанието на същото поле се прави с командния ред

```
net.numInputs=2
```

и съдържанието на полето ще бъде променено от 1 на 2.

Повечето от полетата на структурния масив net имат съдържание, характерно за синтаксиса, приет в MATLAB. Изключение правят полетата IW, LW и b, тъй като в тези полета се запомнят теглата на вече обучената мрежа. В полето IW се запомнят теглата, свързващи входовете на мрежата с невроните от мрежата, в полето LW се запомнят теглата, свързващи отделните слоеве на мрежата, а в полето b се запомнят свободните тегла на съответните слоеве.

Съдържанието на полетата, свързващи съответно входовете с първия слой и входовете с втория слой, се извикват с командите net.IW{1,1} и net.IW{2,1}. Очевидно, второто поле е празно, тъй като входовете на генерираната по-горе мрежа net нямат връзки към втория слой. Съдържанието на първото поле е

```
ans =
    0.3002   -0.2306
  -40.4579  -41.5751
```

Съдържанието на полетата, свързващи съответно изходите на първия слой с входовете на първия слой, изходите на първия слой с входовете на втория слой, изходите на втория слой с входовете на първия слой и изходите на втория слой с входовете на втория слой, се извикват съответно с командите: net.LW{1,1}, net.LW{2,1} net.LW{1,2} и net.LW{2,2}. Като изключим полето net.LW{2,1}, останалите полета са празни, а съдържанието на полето net.LW{2,1} е

```
ans =
   72.2178   -7.4361
```

Свободните тегла на първия и втория слой са запомнени в полетата net.b{1} и net.b{2}. Съдържанията на тези полета са съответно

```
net.b{1}=
   -0.2088
   -2.9267
```

и

```
net.b{2}=
    7.5639.
```

Съдържанието на структурния масив net може да се запомни във файл, примерно във файла data.mat (очевидно разширението на този файл е mat). Това се прави с командата

```
save net
```

Ако след това се налага използване на структурния масив net, той може да се извика по всяко време с командата

```
load net
```

Използвана литература

1. Chen, S., C.F.N. Cowan, and P. M. Grant, “Orthogonal least squares learning algorithm for radial basis function networks,” IEEE Transactions on Neural Networks, vol. 2, No. 2, March 1991, pp. 302-309.
2. G. Cybenko, “Approximation by superpositions of a sigmoidal Function”, Mathematics of Control Signals, and Systems, Vol. 2, 1989, pp. 303-314,
3. J. E. Dennis and R. B. Schnabel. “Numerical methods for unconstrained optimization and nonlinear equations”, Prentice-Hall, Englewood Clifs, NJ, 1983.
4. DARPA Neural Network Study, Lexington, MA:M.I.T. Lincoln, Laboratory, 1988
5. Y. Fang, F. Wang , Q. Zhang, A new micromodeling approach for nonlinear microwave circuits based on recurrent neural networks, Trans. on microwave theory an techniques, vol. 48, No 12, 2000.
6. L. Jin, P.Nikiforuk, M.Gupta, “Approximation of discrete-time state-space trajectories using dynamical recurrent neural networks”, IEEE Transactions on Automatic Control, vol. 40, No7, 1995.
7. J. Hertz, A. Krogh and R. Palmer, “Introduction to the theory of neural computation”, Addison-Wesley Publishing Company, 1990.
8. Li, J., A. N. Michel, and W. Porod, “Analysis and synthesis of a class of neural networks: linear systems operating on a closed hypercube”, *IEEE Trans. on Circuits and Systems*, vol. 36, No.11,1989.
9. Neural Network Toolbox, The Math Works, Inc., MATLAB 6.0 Help.
10. Norgaard, O. Rivan, N. Poulsen, L. Hansen. “Neural networks for modelling and control of dynamic systems”, Springer 2000.
11. W. Tommy, S. Chow and X. Li, “Modelling of continuous time dynamical systems with input by Recurrent Neural Networks”, IEEE Trans. CAS-I, Vol. 47, April 2000.
12. G. Venkov, S. Minchev, Two algorithms for neural networks training. XXIX International Summer School “Applications of Mathematics in Engineering and Economics”, Sozopol 2004 .
13. Г. Венков, Цифрово моделиране на динамични системи, ВМЕИ, 1986.
14. В. Младенов, С. Йорданова, Размито управление и невронни мрежи, ТУ - София, 2006.

СЪДЪРЖАНИЕ